# Source-free Domain Adaptation via Avatar Prototype Generation and Adaptation

**Zhen Qiu**[1,4*], **Yifan Zhang**[2*], **Hongbin Lin**[1*], **Shuaicheng Niu**[1], **Yanxia Liu**[1], **Qing Du**[1†] and **Mingkui Tan**[1,3,4†]
[1]School of Software Engineering, South China University of Technology
[2]School of Computing, National University of Singapore
[3]Key Laboratory of Big Data and Intelligent Robot, Ministry of Education
[4]Pazhou Laboratory

## Abstract

We study a practical domain adaptation task, called source-free unsupervised domain adaptation (UDA) problem, in which we cannot access source domain data due to data privacy issues but only a pre-trained source model and unlabeled target data are available. This task, however, is very difficult due to one key challenge: the lack of source data and target domain labels makes model adaptation very challenging. To address this, we propose to mine the hidden knowledge in the source model and exploit it to generate source avatar prototypes (*i.e.,* representative features for each source class) as well as target pseudo labels for domain alignment. To this end, we propose a Contrastive Prototype Generation and Adaptation (CPGA) method. Specifically, CPGA consists of two stages: (1) prototype generation: by exploring the classification boundary information of the source model, we train a prototype generator to generate avatar prototypes via contrastive learning. (2) prototype adaptation: based on the generated source prototypes and target pseudo labels, we develop a new robust contrastive prototype adaptation strategy to align each pseudo-labeled target data to the corresponding source prototypes. Extensive experiments on three UDA benchmark datasets demonstrate the effectiveness and superiority of the proposed method.

## 1 Introduction

Unsupervised domain adaptation (UDA) has achieved remarkable success in many applications, such as image classification and semantic segmentation [Yan *et al.*, 2017; Liang *et al.*, 2019; Tang *et al.*, 2020; Zhang *et al.*, 2020]. The goal of UDA is to leverage a label-rich source domain to improve the model performance on an unlabeled target domain, which bypasses the dependence on laborious target data annotation. Generally, UDA methods can be divided into two categories, *i.e.,* data-level UDA and feature-level UDA. Data-level methods [Sankaranarayanan *et al.*, 2018; Hoffman *et al.*, 2018] attempt to mitigate domain shifts by image transformation between domains via generative adversarial networks [Goodfellow *et al.*, 2014]. By contrast, feature-level methods [Ganin and Lempitsky, 2015; Wei *et al.*, 2016] focus on alleviating domain discrepancies by learning domain-invariant feature representations. In real-world applications, however, one may only access a source trained model instead of source data due to the law of privacy protection. As a result, many existing UDA methods are incapable due to the lack of source data. Therefore, this paper considers a more practical task, called source-free UDA [Liang *et al.*, 2020; Li *et al.*, 2020], which seeks to adapt a well-trained source model to a target domain without using any source data.

Due to the absence of source data as well as target domain labels, it is difficult to estimate the source domain distribution and exploit target class information for alleviating domain discrepancy as previous UDA methods do. Such a dilemma makes source-free UDA very challenging. To solve this task, existing source-free UDA methods seek to refine the source model either by generating target-style images (*e.g.,* MA [Li *et al.*, 2020]) or by pseudo-labeling target data (*e.g.,* SHOT [Liang *et al.*, 2020]). However, directly generating images from the source model can be very difficult and pseudo-labeling may lead to wrong labels due to domain shifts, both of which compromise the training procedure.

To handle the absence of source data, our motivation is to mine the hidden knowledge in the source model. By exploring the source model, we seek to generate feature prototypes of each source class and target pseudo labels for domain alignment. To this end, we propose a new Contrastive Prototype Generation and Adaptation (CPGA) method. Specifically, CPGA contains two stages: (1) Prototype generation: by exploring the classification boundary information in the source classifier, we train a prototype generator to generate source prototypes based on contrastive learning. (2) Prototype adaptation: to mitigate domain discrepancies, based on the generated feature prototypes and target pseudo labels, we develop a new contrastive prototype adaptation strategy to align each pseudo-labeled target data to the source prototype with the same class. To alleviate label noise, we enhance the alignment via confidence reweighting and early learning regularization. Meanwhile, we further boost the alignment via feature clustering to make the target features more compact.

*Authors contributed equally.
†Corresponding author.

In this way, we are able to well adapt the source-trained model to the unlabeled target domain even without any source data.

The contributions of this paper are summarized as follows:

- In CPGA, we propose a contrastive prototype generation strategy for source-free UDA. Such a strategy can generate representative (*i.e.,* intra-class compact and inter-class separated) avatar feature prototypes for each class. The generated prototypes can be applied to help conventional UDA methods to handle source-free UDA.
- In CPGA, we also propose a robust contrastive prototype adaptation strategy for source-free UDA. Such a strategy can align each pseudo-labeled target data to the corresponding source prototype and meanwhile alleviate the issue of pseudo label noise.
- Extensive experiments on three domain adaptation benchmark datasets demonstrate the effectiveness and superiority of the proposed method.

## 2 Related Work

**Unsupervised Domain Adaptation (UDA).** UDA has been widely studied in recent years [Tang *et al.*, 2020; Jin *et al.*, 2020]. Most existing methods alleviate the domain discrepancy either by adding adaptation layers to match high-order moments of distributions, *e.g.,* DDC [Tzeng *et al.*, 2014], or by devising a domain discriminator to learn domain-invariant features in an adversarial manner, *e.g.,* DANN [Ganin and Lempitsky, 2015] and MCD [Saito *et al.*, 2018]. Recently, prototypical methods and contrastive learning has been introduced to UDA. For instance, TPN [Pan *et al.*, 2019] and PAL [Hu *et al.*, 2020] attempts to align the source and target domains based on the learned prototypical feature representations. Besides, CAN [Kang *et al.*, 2019] and CoSCA [Dai *et al.*, 2020] leverages contrastive learning to explicitly minimize intra-class distance and maximize inter-class distance in terms of both intra-domain and inter-domain. However, the source data may be unavailable in practice due to privacy issues, making these methods incapable.

**Source-free Unsupervised Domain Adaptation.** Source-free UDA [Kim *et al.*, 2020] aims to adapt the source model to an unlabeled target domain without using the source data. Existing methods seek to refine the source model either by pseudo-labeling (*e.g.,* SHOT [Liang *et al.*, 2020]) or by generating target-style images (*e.g.,* MA [Li *et al.*, 2020]). However, due to the domain discrepancy, the pseudo labels can be noisy, which is ignored by SHOT. Besides, directly generating target-style images from the source model can be very difficult due to training difficulties of GANs. Very recently, BAIT [Yang *et al.*, 2020b] proposes to use the source classifier as source anchors and use them for domain alignment. However, BAIT requires dividing target data into certain and uncertain sets via prediction entropy of source classifier, which may lead to wrong division due to domain shifts.

Compared with the above methods, we propose to generate source feature prototypes for each class instead of directly generating images. Besides, we alleviate the negative transfer brought by noisy pseudo labels through confidence reweighting and regularization.

## 3 Proposed Method

### 3.1 Problem Definition

We focus on the task of source-free unsupervised domain adaptation (UDA) in this paper, where only a well-trained source model and unlabeled target data are accessible. Specifically, we consider a $K$-class classification task, where the source and target domains share with the same label space. We assume that the pre-trained source model consists of a feature extractor $G_e$ and a classifier $C_y$. Moreover, we denote the unlabeled target domain by $D_t=\{\mathbf{x}_i\}_{i=1}^{n_t}$, where $n_t$ is the number of target samples.

The key goal is to adapt the source model to the target domain with access to only unlabeled target data. Such a task, however, is very challenging due to the lack of source domain data and target domain annotations. Hence, conventional UDA methods requiring source data are unable to tackle this task. To address this task, we innovatively propose a Contrastive Prototype Generation and Adaptation (CPGA) method.

### 3.2 Overall Scheme

Inspired by that feature prototypes can represent a group of semantically similar instances [Snell *et al.*, 2017], we explore to generate avatar feature prototypes to represent each source class and use them for class-wise domain alignment. As shown in Figure 1, the proposed CPGA consist of two stages: prototype generation and prototype adaptation.

In the stage one (Section 3.3), inspired by that the classifier of the source model contains class distribution information [Xu *et al.*, 2020], we propose to train a class conditional generator $G_g$ to learn such class information and generate avatar feature prototypes for each class. Meanwhile, we use the source classifier $C_y$ to judge whether $G_g$ generates correct feature prototypes w.r.t. classes. By training the generator $G_g$ to confuse $C_y$ via both cross-entropy $\mathcal{L}_{ce}$ and the contrastive loss $\mathcal{L}_{con}^{p}$, we are able to generate intra-class compact and inter-class separated feature prototypes. Meanwhile, to overcome the lack of target domain annotations, we resort to a self pseudo-labeling strategy to generate pseudo labels for each target data (Section 3.4).

In the stage two (Section 3.5), we adapt the source model to the target by aligning the pseudo-labeled target features to the source prototypes. Specifically, we conduct class-wise alignment through a contrastive loss $\mathcal{L}_{con}^{w}$ based on a domain projector $C_p$. Meanwhile, we devise an early learning regularization term $\mathcal{L}_{elr}$ to prevent remembering noisy pseudo labels. Lastly, to make the feature more discriminative, we further impose a neighborhood clustering loss $\mathcal{L}_{nc}$.

The overall training procedure of CPGA can be summarized as follows:

$$\min_{\theta_g} \mathcal{L}_{ce}(\theta_g) + \mathcal{L}_{con}^{p}(\theta_g), \tag{1}$$

$$\min_{\{\theta_e,\theta_p\}} \mathcal{L}_{con}^{w}(\theta_e,\theta_p) + \lambda\mathcal{L}_{elr}(\theta_e,\theta_p) + \eta\mathcal{L}_{nc}(\theta_e), \tag{2}$$

where $\theta_g$, $\theta_e$ and $\theta_p$ denotes the parameters of the generator $G_g$, the feature extractor $G_e$ and the projector $C_p$, respectively. Moreover, $\lambda$ and $\eta$ are trade-off parameters to balance losses. For simplicity, we set the trade-off parameter to 1 in Eq. (1) based on our preliminary studies.

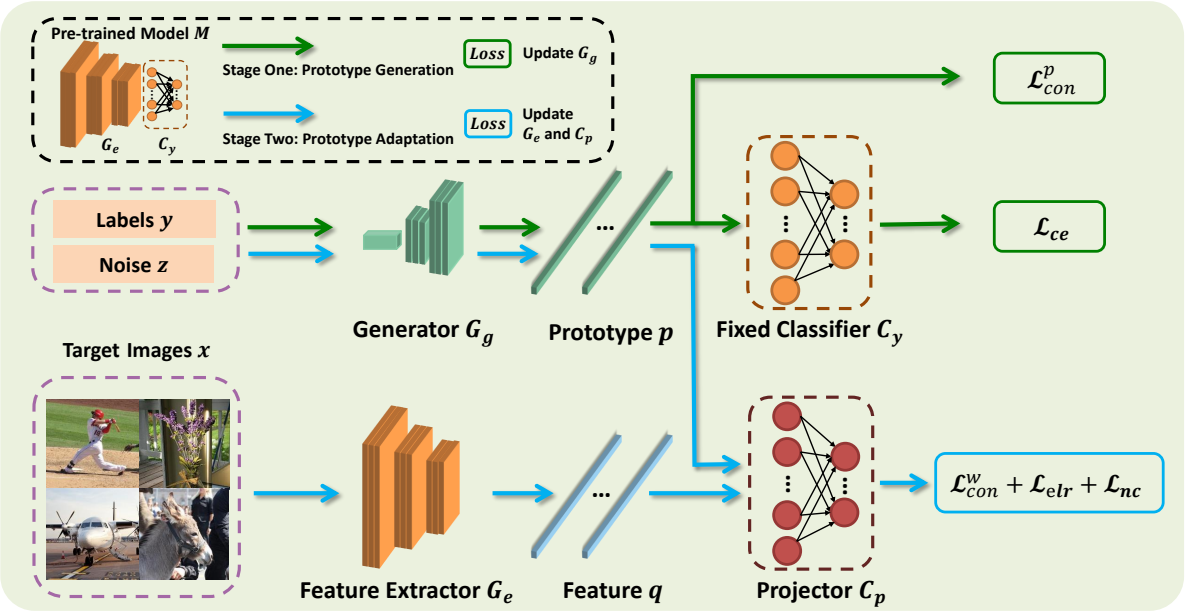



Figure 1: An overview of CPGA. CPGA contains two stages: (1) **Prototype generation**: under the guidance of the fixed classifier, a generator $G_g$ is trained to generate avatar feature prototypes via $\mathcal{L}_{ce}$ and $\mathcal{L}_{con}^{p}$. (2) **Prototype adaptation**: in each training batch, we use the learned prototype generator to generate one prototype for each class. Based on the generated prototypes and pseudo labels obtained by clustering, we align each pseudo-labeled target feature to the corresponding class prototype by training a domain-invariant feature extractor via $\mathcal{L}_{con}^{w}$, $\mathcal{L}_{elr}$ and $\mathcal{L}_{nc}$. Note that the classifier $C_y$ is fixed during the whole training phase.

### 3.3 Contrastive Prototype Generation

The absence of the source data makes UDA challenging. To handle this, we propose to generate feature prototypes for each class by exploring the class distribution information hidden in the source classifier [Xu *et al.*, 2020]. To this end, we use the source classifier $C_y$ to train the class conditional generator $G_g$. To be specific, as shown in Figure 1, given a uniform noise $\mathbf{z}\sim U(0,1)$ and a label $\mathbf{y}\in\mathbb{R}^K$ as inputs, the generator $G_g$ first generates the feature prototype $\mathbf{p}=G_g(\mathbf{y},\mathbf{z})$ (More details of the generator and the generation process can be found in Supplementary). Then, the classifier $G_y$ judges whether the generated prototype belongs to $\mathbf{y}$ and trains the generator via the cross entropy loss:

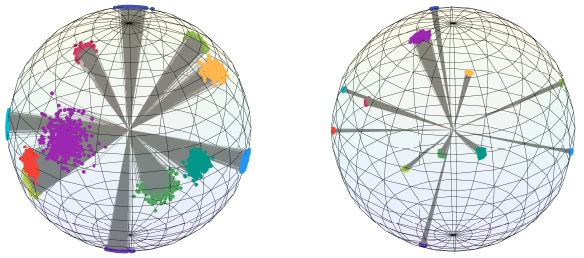
$$\mathcal{L}_{ce} = -\mathbf{y}\log C_y(\mathbf{p}), \tag{3}$$

where $\mathbf{p}$ is the generated prototype and $C_y(\mathbf{p})$ denotes the prediction of the classifier. In this way, the generator is capable of generating feature prototypes for each category.

However, as shown in Figure 3(a), training the generator with only the cross entropy may make the feature prototypes not well compact and prototypical. As a result, domain alignment with these prototypes may make the adapted model less discriminative, leading to insufficient performance (See Table 4). To address this, motivated by InfoNCE [van den Oord *et al.*, 2018; Zhang *et al.*, 2021], we further impose a contrastive loss for all generated prototypes to encourage more prototypical prototypes:

$$\mathcal{L}_{con}^{p} = -\log \frac{\exp(\phi(\mathbf{p},\mathbf{k}^{+})/\tau)}{\exp(\phi(\mathbf{p},\mathbf{k}^{+})/\tau)+\sum_{j=1}^{K-1}\exp(\phi(\mathbf{p},\mathbf{k}_j^{-})/\tau)}, \tag{4}$$

where $\mathbf{p}$ denotes any anchor prototype. For each anchor, we sample the positive pair $\mathbf{k}^{+}$ by randomly selecting a generated prototype with the same category to the anchor $\mathbf{p}$, and sample $K-1$ negative pairs $\mathbf{k}^{-}$ that have diverse classes with the anchor. Here, in each training batch, we generate at least 2 prototypes for each class in the stage one. Moreover, $\phi(\cdot,\cdot)$ denotes the cosine similarity and $\tau$ is a temperature factor.



(a) Training with $\mathcal{L}_{ce}$ (b) Training with $\mathcal{L}_{ce}+\mathcal{L}_{con}^{p}$

Figure 2: Visualizations of the generated feature prototypes by the generator trained with different losses. Compared to training with only the cross entropy $\mathcal{L}_{ce}$, the contrastive loss $\mathcal{L}_{con}^{p}$ encourages the prototypes of the same category to be more compact and those of different categories to be more separated. Better viewed in color.

As shown in Figure 3(b), by training the generator with $\mathcal{L}_{ce}+\mathcal{L}_{con}^{p}$, the generated prototypes are more representative (*i.e.,* intra-class compact and inter-class separated). Interestingly, we empirically observe that the inter-class cosine distance will converge closely to 1 (*i.e.,* cosine similarity close to 0) by training with $\mathcal{L}_{ce}+\mathcal{L}_{con}^{p}$ (See Table 4), if the feature dimensions are larger than the number of classes. That is, the generated prototypes of different categories are approximatively orthometric in the high-dimensional feature space.

### 3.4 Pseudo Label Generation for Target Data

Domain alignment can be conducted based on the generated avatar source prototypes, However, the alignment is non-trivial due to the lack of target annotations, which makes the class-wise alignment difficult [Pei *et al.*, 2018; Kang *et al.*, 2019]. To address this, we generate pseudo labels based on a self-supervised pseudo-labeling strategy, proposed in [Liang *et al.*, 2020]. To be specific, let $\mathbf{q}_i{=}G_e(\mathbf{x}_i)$ denote the feature vector and let $\hat{y}_i^k{=}C_y^k(\mathbf{q})$ be the predicted probability of the classifier regarding the class $k$. We first attain the initial centroid for each class $k$ by:

$$\mathbf{c}_k = \frac{\sum_{i=1}^{n_t} \hat{y}_i^k \mathbf{q}_i}{\sum_{i=1}^{n_t} \hat{y}_i^k}, \tag{5}$$

where $n_t$ is the number of target data. These centroids help to characterize the distribution of different categories [Liang *et al.*, 2020]. Then, the pseudo label of the $i$-th target data is obtained via a nearest centroid approach:

$$\bar{y}_i = \arg\max_k \phi(\mathbf{q}_i, \mathbf{c}_k), \tag{6}$$

where $\phi(\cdot,\cdot)$ denotes the cosine similarity, and the pseudo label $\bar{y}_i{\in}\mathbb{R}^1$ is a scalar index. During the training process, we update the centroid of each class by $\mathbf{c}_k{=}\frac{\sum_{i=1}^{n_t}\mathbb{I}(\bar{y}_i=k)\mathbf{q}_i}{\sum_{i=1}^{n_t}\mathbb{I}(\bar{y}_i=k)}$ and then update pseudo labels based on Eqn. (6) in each epoch, where $\mathbb{I}(\cdot)$ is the indicator function.

### 3.5 Contrastive Prototype Adaptation

Based on the generated prototypes and target pseudo labels, we conduct prototype adaptation to alleviate domain shifts. Here, in each training batch, we generate one prototype for each class. However, due to domain discrepancies, the pseudo labels can be quite noisy, making the adaptation difficult. To address this, we propose a new contrastive prototype adaptation strategy, which consists of three key components: (1) weighted contrastive alignment; (2) early learning regularization; (3) target neighborhood clustering.

**Weighted Contrastive Alignment.** Based on the pseudo-labeled target data, we then conduct class-wise contrastive learning to align the target data to the corresponding source feature prototype. However, the pseudo labels may be noisy, which degrades contrastive alignment. To address this, we propose to differentiate pseudo-labeled target data and assign higher importance to the reliable ones. Motivated by [Chen *et al.*, 2019] that reliable samples are generally more close to the class centroid, we compute the confidence weight by:

$$w_i = \frac{\exp(\phi(\mathbf{q}_i, \mathbf{c}_{\bar{y}_i})/\tau)}{\sum_{k=1}^{K}\exp(\phi(\mathbf{q}_i, \mathbf{c}_k)/\tau)}, \tag{7}$$

where the feature with higher similarity to the corresponding centriod will have higher importance. Then, we can conduct weighted contrastive alignment. To this end, inspired by [Chen *et al.*, 2020], we first use a non-linear projector $C_p$ to project the target features and source prototypes to a $l_2$-normalized contrastive feature space. Specifically, the target contrastive feature is denoted as $\mathbf{u}{=}C_p(\mathbf{q})$, while the prototype contrastive feature is denoted as $\mathbf{v}{=}C_p(\mathbf{p})$. Then, for any target feature $\mathbf{u}_i$ as an anchor, we conduct prototype adaptation via a weighted contrastive loss:

$$\mathcal{L}_{con}^{w} = -w_i \log \frac{\exp(\mathbf{u}_i^\top \mathbf{v}^+/\tau)}{\exp(\mathbf{u}_i^\top \mathbf{v}^+/\tau) + \sum_{j=1}^{K-1}\exp(\mathbf{u}_i^\top \mathbf{v}_j^-/\tau)}, \tag{8}$$

where the positive pair $\mathbf{v}^+$ is the prototype with the same class to the anchor $\mathbf{u}_i$, while the negative pairs $\mathbf{v}^-$ are the prototypes with different classes.

**Early Learning Regularization.** To further prevent the model from memorizing noise, we propose to regularize the learning process via an early learning regularizer. Since DNNs first memorize the clean samples with correct labels and then the noisy data with wrong labels [Arpit *et al.*, 2017], the model in the "early learning" phase can be more predictable to the noisy data. Therefore, we seek to use the early predictions of each sample to regularize learning. To this end, we devise a memory bank $\mathcal{H}{=}\{\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_{n_t}\}$ to record non-parametric predictions of each target sample, and update them based on new predictions via a momentum strategy. Formally, for the $i$-th sample, we predict its non-parametric prediction regarding the $k$-th prototype by $o_{i,k}{=}\frac{\exp(\mathbf{u}_i^\top \mathbf{v}_k/\tau)}{\sum_{j=1}^{K}\exp(\mathbf{u}_i^\top \mathbf{v}_j/\tau)}$, and update the momentum by:

$$\mathbf{h}_i \leftarrow \beta \mathbf{h}_i + (1-\beta)\mathbf{o}_i, \tag{9}$$

where $\mathbf{o}_i{=}[o_{i,1}, ..., o_{i,K}]$, and $\beta$ denotes the momentum coefficient. Based on the memory bank, for the $i$-th data, we further train the model via an early learning regularizer $\mathcal{L}_{elr}$, proposed in [Liu *et al.*, 2020]:

$$\mathcal{L}_{elr} = \log(1 - \mathbf{o}_i^\top \mathbf{h}_i). \tag{10}$$

This regularizer enforces the current prediction to be close to the prediction momentum, which helps to prevent overfitting to label noise. Note that the use of $\mathcal{L}_{elr}$ in this paper is different from [Liu *et al.*, 2020], which focuses on classification tasks and uses parametric predictions.

**Target Neighborhood Clustering.** To enhance the contrastive alignment, we further resort to feature clustering to make the target features more compact. Inspired by [Saito *et al.*, 2020] that the intra-class samples in the same domain are generally more close, we propose to close the distance between each target sample and its nearby neighbors. To this end, we maintain a memory bank $\mathcal{Q}{=}\{\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_{n_t}\}$ to restore all target features, which are updated when new features are extracted in each iteration. Based on the bank, for the $i$-th sample's feature $\mathbf{q}_i$, we can compute its normalized similarity with any feature $\mathbf{q}_j$ by $\mathbf{s}_{i,j}{=}\frac{\exp(\phi(\mathbf{q}_i,\mathbf{q}_j)/\tau)}{\sum_{l=1,l\neq i}^{n_t}\exp(\phi(\mathbf{q}_i,\mathbf{q}_l)/\tau)}$. Motivated by that minimizing the entropy of the normalized similarity helps to learn compact features for similar data [Saito *et al.*, 2020], we further train the extractor via a neighborhood clustering loss:

$$\mathcal{L}_{nc} = -\sum_{j=1,j\neq i}^{n_t} \mathbf{s}_{i,j}\log(\mathbf{s}_{i,j}). \tag{11}$$

Note that the entropy minimization here does not use pseudo labels, so the learned compact target features are (to some degree) robust to pseudo label noise. We summarize the overall training scheme of CPGA in Algorithms 1, while the inference is provided in the supplementary.

**Algorithm 1** Training of CPGA

**Require:** Unlabeled target data $D_t=\{\mathbf{x}_i\}_{i=1}^{n_t}$; Source model $\{G_e, C_y\}$; Training epoch $E$, $M$; Parameters $\eta$, $\beta$, $\tau$, $\lambda$.
**Initialize:** Projector $C_p$; Generator $G_g$.
1: **for** $e = 1 \rightarrow E$ **do**
2: Generate prototypes $\mathbf{p}$ based on $G_g$;
3: Compute $\mathcal{L}_{ce}$ and $\mathcal{L}_{con}^{p}$ based on Eqns. (3) and (4);
4: loss.backward() based on Eqn. (1).
5: **end for**
6: **for** $m = 1 \rightarrow M$ **do**
7: Generate prototypes $\mathbf{p}$ for each class based on fixed $G_g$;
8: Extract target data features $G_e(\mathbf{x})$ based on $G_e$;
9: Obtain target pseudo labels based on Eqn. (6);
10: Obtain contrastive features $\mathbf{h}_t$ based on $C_p$;
11: Compute $\mathcal{L}_{con}^{w}$, $\mathcal{L}_{elr}$, $\mathcal{L}_{nc}$ based on Eqns. (8), (10), (11);
12: loss.backward() based on Eqn. (2).
13: **end for**
14: **Output:** $G_e$ and $C_y$.

# 4 Experiments

**Datasets.** We conduct the experiments on three benchmark datasets: (1) **Office-31** [Saenko *et al.*, 2010] is a standard domain adaptation dataset that is made up of three distinct domains, *i.e.,* Amazon (A), Webcam (W) and DSLR (D). Three domains share 31 categories and contain 2817, 795 and 498 samples, respectively. (2) **VisDA** [Peng *et al.*, 2017] is a large-scale challenging dataset that concentrates on the 12-class synthesis-to-real object recognition task. The source domain contains 152k synthetic images while the target domain has 55k real object images. (3) **Office-Home** [Venkateswara *et al.*, 2017] is a medium-sized dataset, which contains four distinct domains, *i.e.,* Artistic images (Ar), Clip Art (Cl), Product images (Pr) and Real-world images (Rw). Each of the four domains has 65 categories.

**Baselines.** We compare CPGA with three types of baselines: (1) source-only: ResNet [He *et al.*, 2016]; (2) unsupervised domain adaptation with source data: MCD [Saito *et al.*, 2018], CDAN [Long *et al.*, 2018], TPN [Pan *et al.*, 2019], SAFN [Xu *et al.*, 2019], SWD [Lee *et al.*, 2019], MDD [Zhang *et al.*, 2019b], CAN [Kang *et al.*, 2019], DMRL [Wu *et al.*, 2020], BDG [Yang *et al.*, 2020a], PAL [Hu *et al.*, 2020], MCC [Jin *et al.*, 2020], SRDC [Tang *et al.*, 2020]; (3) source-free unsupervised domain adaptation: SHOT [Liang *et al.*, 2020], PrDA [Kim *et al.*, 2020], MA [Li *et al.*, 2020] and BAIT [Yang *et al.*, 2020b].

**Implementation Details.** We implement our method based on PyTorch[1]. For a fair comparison, we report the results of all baselines in the corresponding papers. For the network architecture, we adopt a ResNet [He *et al.*, 2016], pre-trained on ImageNet, as the backbone of all methods. Following [Liang *et al.*, 2020], we replace the original fully connected (FC) layer with a task-specific FC layer followed by a weight normalization layer. The projector consists of three FC layers with hidden feature dimensions of 1024, 512 and 256. We train the source model via label smoothing technique [Müller *et al.*, 2019] and train CPGA using SGD optimizer. We set the learning rate and epoch to 0.01 and 40 for VisDA and to 0.001 and 400 for Office-31 and Office-Home. For hyper-parameters, we set $\eta$, $\beta$, $\tau$ and batch size to 0.05, 0.9, 0.07 and 64, respectively. Besides, we set $\lambda$=7 for Office-31 and Office-home while $\lambda$=5 for VisDA. Following [Xu *et al.*, 2020], the dimension of noise $\mathbf{z}$ is 100. We put more implementation details in the supplementary.

[1]The source code is available: github.com/SCUT-AILab/CPGA.

| Method | Source-free | A→D | A→W | D→W | W→D | D→A | W→A | Avg. |
|---|---|---|---|---|---|---|---|---|
| ResNet-50 [He *et al.*, 2016] | ✗ | 68.9 | 68.4 | 96.7 | 99.3 | 62.5 | 60.7 | 76.1 |
| MCD [Saito *et al.*, 2018] | ✗ | 92.2 | 88.6 | 98.5 | 100.0 | 69.5 | 69.7 | 86.5 |
| CDAN [Long *et al.*, 2018] | ✗ | 92.9 | 94.1 | 98.6 | 100.0 | 71.0 | 69.3 | 87.7 |
| MDD [Zhang *et al.*, 2019b] | ✗ | 90.4 | 90.4 | 98.7 | 99.9 | 75.0 | 73.7 | 88.0 |
| CAN [Kang *et al.*, 2019] | ✗ | 95.0 | 94.5 | 99.1 | 99.6 | 70.3 | 66.4 | 90.6 |
| DMRL [Wu *et al.*, 2020] | ✗ | 93.4 | 90.8 | 99.0 | 100.0 | 73.0 | 71.2 | 87.9 |
| BDG [Yang *et al.*, 2020a] | ✗ | 93.6 | 93.6 | 99.0 | 100.0 | 73.2 | 72.0 | 88.5 |
| MCC [Jin *et al.*, 2020] | ✗ | 95.6 | 95.4 | 98.6 | 100.0 | 72.6 | 73.9 | 89.4 |
| SRDC [Tang *et al.*, 2020] | ✗ | 95.8 | 95.7 | 99.2 | 100.0 | 76.7 | 77.1 | 90.8 |
| PrDA [Kim *et al.*, 2020] | ✓ | 92.2 | 91.1 | 98.2 | 99.5 | 71.0 | 71.2 | 87.2 |
| SHOT [Liang *et al.*, 2020] | ✓ | 93.1 | 90.9 | **98.8** | 99.9 | 74.5 | 74.8 | 88.7 |
| BAIT [Yang *et al.*, 2020b] | ✓ | 92.0 | **94.6** | 98.1 | **100.0** | 74.6 | 75.2 | 89.1 |
| MA [Li *et al.*, 2020] | ✓ | 92.7 | 93.7 | 98.5 | 99.8 | 75.3 | **77.8** | 89.6 |
| CPGA (ours) | ✓ | **94.4** | 94.1 | 98.4 | 99.8 | **76.0** | 76.6 | **89.9** |

Table 1: Accuracy (%) on the small-sized **Office-31** (ResNet-50).

## 4.1 Comparison with State-of-the-arts

In this section, we compare our proposed CPGA with the state-of-the-art methods. For **Office-31**, as shown in Table 1, the proposed CPGA achieves the best performance compared with source-free UDA methods w.r.t. the average accuracy over 6 transfer tasks. Moreover, our method shows its superiority in the task of A→D and D→A and comparable results on the other tasks. Note that even compared with the state-of-the-art methods using source data (*e.g.,* SRDC), our CPGA is able to obtain a competitive result as well. Besides, from Table 2, CPGA outperforms all the state-of-the-art methods w.r.t. the average accuracy (*i.e.,* per-class accuracy) on the more challenging dataset **VisDA**. Specifically, CPGA gets the best accuracy in the eight categories and obtains comparable results in others. Moreover, our CPGA is able to surpass the baseline methods with source data (*e.g.,* CoSCA), which demonstrates the superiority of our proposed method. For **Office-Home**, we put the results in the supplementary.

## 4.2 Ablation Study

To evaluate the effectiveness of the proposed two modules (*i.e.,* prototype generation and prototype adaptation) and the sensitivity of hyper-parameters, we conduct a series of ablation studies on VisDA.

**Effectiveness of Prototype Generation.** In this section, we verify the effect of our generated prototypes in the existing domain adaptation methods (*e.g.,* DANN [Ganin and Lempitsky, 2015], ADDA [Tzeng *et al.*, 2017] and DMAN [Zhang *et al.*, 2019a]), which, previously, cannot solve the domain adaptation problem without source data. To this end, we introduce our prototype generation module to replace their source data-oriented parts. From Table 3, based on prototypes, the existing methods achieve competitive performance compared with the counterparts using source data, or even perform better in some tasks. It demonstrates the superiority and applicability of our prototype generation scheme.

| Method | Source-free | plane | bicycle | bus | car | horse | knife | mcycl | person | plant | sktbrd | train | truck | Per-class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ResNet-101 [He *et al.*, 2016] | ✗ | 55.1 | 53.3 | 61.9 | 59.1 | 80.6 | 17.9 | 79.7 | 31.2 | 81.0 | 26.5 | 73.5 | 8.5 | 52.4 |
| CDAN [Long *et al.*, 2018] | ✗ | 85.2 | 66.9 | 83.0 | 50.8 | 84.2 | 74.9 | 88.1 | 74.5 | 83.4 | 76.0 | 81.9 | 38.0 | 73.9 |
| SAFN [Xu *et al.*, 2019] | ✗ | 93.6 | 61.3 | 84.1 | 70.6 | 94.1 | 79.0 | 91.8 | 79.6 | 89.9 | 55.6 | 89.0 | 24.4 | 76.1 |
| SWD [Lee *et al.*, 2019] | ✗ | 90.8 | 82.5 | 81.7 | 70.5 | 91.7 | 69.5 | 86.3 | 77.5 | 87.4 | 63.6 | 85.6 | 29.2 | 76.4 |
| TPN [Pan *et al.*, 2019] | ✗ | 93.7 | 85.1 | 69.2 | 81.6 | 93.5 | 61.9 | 89.3 | 81.4 | 93.5 | 81.6 | 84.5 | 49.9 | 80.4 |
| PAL [Hu *et al.*, 2020] | ✗ | 90.9 | 50.5 | 72.3 | 82.7 | 88.3 | 88.3 | 90.3 | 79.8 | 89.7 | 79.2 | 88.1 | 39.4 | 78.3 |
| MCC [Jin *et al.*, 2020] | ✗ | 88.7 | 80.3 | 80.5 | 71.5 | 90.1 | 93.2 | 85.0 | 71.6 | 89.4 | 73.8 | 85.0 | 36.9 | 78.8 |
| CoSCA [Dai *et al.*, 2020] | ✗ | 95.7 | 87.4 | 85.7 | 73.5 | 95.3 | 72.8 | 91.5 | 84.8 | 94.6 | 87.9 | 87.9 | 36.8 | 82.9 |
| PrDA [Kim *et al.*, 2020] | ✓ | 86.9 | 81.7 | **84.6** | 63.9 | **93.1** | 91.4 | 86.6 | 71.9 | 84.5 | 58.2 | 74.5 | 42.7 | 76.7 |
| SHOT [Liang *et al.*, 2020] | ✓ | 92.6 | 81.1 | 80.1 | 58.5 | 89.7 | 86.1 | 81.5 | 77.8 | 89.5 | 84.9 | 84.3 | 49.3 | 79.6 |
| MA [Li *et al.*, 2020] | ✓ | 94.8 | 73.4 | 68.8 | **74.8** | **93.1** | 95.4 | 88.6 | **84.7** | 89.1 | 84.7 | 83.5 | 48.1 | 81.6 |
| BAIT [Yang *et al.*, 2020b] | ✓ | 93.7 | 83.2 | 84.5 | 65.0 | 92.9 | 95.4 | 88.1 | 80.8 | 90.0 | 89.0 | 84.0 | 45.3 | 82.7 |
| CPGA (ours, 40 epochs) | ✓ | 94.8 | 83.6 | 79.7 | 65.1 | 92.5 | 94.7 | **90.1** | 82.4 | 88.8 | 88.0 | **88.9** | 60.1 | 84.1 |
| CPGA (ours, 400 epochs) | ✓ | **95.6** | **89.0** | 75.4 | 64.9 | 91.7 | **97.5** | 89.7 | 83.8 | **93.9** | **93.4** | 87.7 | **69.0** | **86.0** |

Table 2: Classification accuracies (%) on the large-scale **VisDA** dataset (ResNet-101).

| Method | A→D | A→W | D→W | W→D | D→A | W→A | Avg. |
|---|---|---|---|---|---|---|---|
| DANN (with source data) | 79.7 | 82.0 | 96.9 | 99.1 | 68.2 | 67.4 | 82.2 |
| DANN (with prototypes) | **83.7** | 81.1 | **97.5** | **99.8** | 63.4 | 63.6 | 81.5 |
| DMAN (with source data) | 83.3 | 85.7 | 97.1 | 100.0 | 65.1 | 64.4 | 82.6 |
| DMAN (with prototypes) | **86.3** | 84.2 | **97.7** | **100.0** | 64.7 | **64.5** | **82.9** |
| ADDA (with source data) | 82.9 | 79.9 | 97.4 | 99.4 | 64.9 | 63.6 | 81.4 |
| ADDA (with prototypes) | **83.5** | **81.9** | 97.2 | **100.0** | 63.8 | 63.0 | **81.6** |

Table 3: Comparisons of the existing domain adaptation methods with source data or prototypes on **Office-31** (ResNet-50).

| Objective | Inter-class distance | Intra-class distance | Per-class (%) |
|---|---|---|---|
| $\mathcal{L}_{ce}$ | 0.7860 | $3.343 \times e^{-4}$ | 85.0 |
| $\mathcal{L}_{ce} + \mathcal{L}^{p}_{con}$ | 1.0034 | $2.670 \times e^{-6}$ | 86.0 |

Table 4: Ablation studies on prototype generation in the stage one with different losses. Inter-class distance and intra-class distance is based on cosine distance (range from 0 to 2). We report per-class accuracy (%) after training the model on **VisDA** for 400 epochs.

| Backbone | $\mathcal{L}_{con}$ | $\mathcal{L}^{w}_{con}$ | $\mathcal{L}_{elr}$ | $\mathcal{L}_{nc}$ | Per-class (%) |
|---|---|---|---|---|---|
| ✓ | | | | | 52.4 |
| ✓ | ✓ | | | | 80.9 |
| ✓ | | ✓ | | | 82.7 |
| ✓ | | ✓ | ✓ | | 85.4 |
| ✓ | | ✓ | ✓ | ✓ | **86.0** |

Table 5: Ablation study for the losses (*i.e.,* $\mathcal{L}^{w}_{con}$, $\mathcal{L}_{elr}$ and $\mathcal{L}_{nc}$) of prototype adaptation. We show the per-class accuracy (%) of the model trained on **VisDA** for 400 epochs. $\mathcal{L}_{con}$ denotes $\mathcal{L}^{w}_{con}$ without confidence weight $w$.

| Parameter | $\lambda$ | | | | | $\eta$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 3 | 5 | 7 | 9 | 0.001 | 0.005 | 0.01 | 0.05 | 0.1 |
| Acc. (40 epochs) | 83.2 | 83.9 | **84.1** | 83.3 | 82.2 | 82.7 | 83.1 | 83.3 | **84.1** | 81.0 |
| Acc. (400 epochs) | 83.3 | 85.0 | **86.0** | 85.5 | 85.3 | 85.5 | 85.6 | 85.5 | **86.0** | 83.0 |

Table 6: Influence of the trade-off parameter $\lambda$ and $\eta$ in terms of per-class accuracy (%) on **VisDA**. The value of $\lambda$ is chosen from $[1, 3, 5, 7, 9]$ and $\eta$ is chosen from $[0.001, 0.005, 0.01, 0.05, 0.1]$. In each experiment, the rest of hyper-parameters are fixed.

**Ablation Studies on Prototype Generation.** To study the impact of our contrastive loss $\mathcal{L}^{p}_{con}$, we compare the generated prototype results from models with and without $\mathcal{L}^{p}_{con}$. From Table 4[2], compared with training by cross-entropy loss $\mathcal{L}_{ce}$ only, optimizing the generator via $\mathcal{L}_{ce}+\mathcal{L}^{p}_{con}$ makes the inter-class features separated (*i.e.,* larger inter-class distance) and intra-class features compact (*i.e.,* smaller intra-class distance). The $\mathcal{L}^{p}_{con}$ loss also helps to enhance the performance from $85.0\%$ to $86.0\%$.

**Ablation Studies on Prototype Adaptation.** To investigate the losses of prototype adaptation, we show the quantitative results of the models optimized by different losses. As shown in Table 5, compared with the conventional contrastive loss $\mathcal{L}_{con}$, our proposed contrastive loss $\mathcal{L}^{w}_{con}$ achieves a more promising result on VisDA. Such a result verifies the ability of alleviating pseudo label noise of the confidence weight $w$. Besides, our model has the ability to further improve the performance when introducing the losses $\mathcal{L}_{elr}$ and $\mathcal{L}_{nc}$. When combining all the three losses (*i.e.,* $\mathcal{L}^{w}_{con}$, $\mathcal{L}_{elr}$ and $\mathcal{L}_{nc}$), we obtain the best performance.

[2]Figure 2 shows the corresponding visual results of Table 4.

**Influence of Hyper-parameters.** In this section, we evaluate the sensitivity of two hyper-parameters $\lambda$ and $\eta$ on VisDA via an unsupervised reverse validation strategy [Ganin *et al.*, 2016] based on the source prototypes. For convenience, we set $\eta = 0.05$ when studying $\lambda$, and set $\lambda = 5$ when studying $\eta$. As shown in Table 6, the proposed method achieves the best performance when setting $\lambda = 5$ and $\eta = 0.05$ on VisDA. The results also demonstrate that our method is non-sensitive for the hyper-parameters. Besides, we put more analysis of hyper-parameters in the supplementary.

## 5 Conclusions

This paper has proposed a prototype generation and adaptation (namely CPGA) method for source-free UDA. Specifically, we overcome the lack of source data by generating avatar feature prototypes for each class via contrastive learning. Based on the generated prototypes, we develop a robust contrastive prototype adaptation strategy to pull the pseudo-labeled target data toward the corresponding source prototypes. In this way, CPGA adapts the source model to the target domain without access to any source data. Extensive experiments verify the effectiveness and superiority of CPGA.

## Acknowledgments

This work was partially supported by Key Realm R&D Program of Guangzhou (202007030007), National Natural Science Foundation of China (NSFC) 62072190, Program for Guangdong Introducing Innovative and Enterpreneurial Teams 2017ZT07X183, Fundamental Research Funds for the Central Universities D2191240, Guangdong Natural Science Foundation Doctoral Research Project (2018A030310365), International Cooperation Open Project of State Key Laboratory of Subtropical Building Science, South China University of Technology (2019ZA02).

## References

[Arpit *et al.*, 2017] Devansh Arpit, Stanislaw K Jastrzebski, et al. A closer look at memorization in deep networks. In *ICML*, 2017.

[Chen *et al.*, 2019] Chaoqi Chen, W. Xie, et al. Progressive feature alignment for unsupervised domain adaptation. *CVPR*, 2019.

[Chen *et al.*, 2020] Ting Chen, Simon Kornblith, et al. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.

[Cui *et al.*, 2020] Shuhao Cui, Shuhui Wang, et al. Towards discriminability and diversity: Batch nuclear-norm maximization under label insufficient situations. *CVPR*, 2020.

[Dai *et al.*, 2020] Shuyang Dai, Yu Cheng, et al. Contrastively smoothed class alignment for unsupervised domain adaptation. In *ACCV*, 2020.

[Ganin and Lempitsky, 2015] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *ICML*, 2015.

[Ganin *et al.*, 2016] Yaroslav Ganin, Evgeniya Ustinova, et al. Domain-adversarial training of neural networks. *JMLR*, 2016.

[Goodfellow *et al.*, 2014] Ian J. Goodfellow, Jean Pouget-Abadie, et al. Generative adversarial networks. In *NeruIPS*, 2014.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, et al. Deep residual learning for image recognition. *CVPR*, 2016.

[Hoffman *et al.*, 2018] Judy Hoffman, Eric Tzeng, et al. Cycada: Cycle-consistent adversarial domain adaptation. In *ICML*, 2018.

[Hu *et al.*, 2020] Dapeng Hu, Jian Liang, et al. Panda: Prototypical unsupervised domain adaptation. *ArXiv*, 2020.

[Jin *et al.*, 2020] Ying Jin, Ximei Wang, et al. Minimum class confusion for versatile domain adaptation. In *ECCV*, 2020.

[Kang *et al.*, 2019] Guoliang Kang, Lu Jiang, et al. Contrastive adaptation network for unsupervised domain adaptation. *CVPR*, 2019.

[Kim *et al.*, 2020] Youngeun Kim, Donghyeon Cho, et al. Progressive domain adaptation from a source pre-trained model. *ArXiv*, 2020.

[Lee *et al.*, 2019] Chen-Yu Lee, Tanmay Batra, Mohammad Haris Baig, and Daniel Ulbricht. Sliced wasserstein discrepancy for unsupervised domain adaptation. In *CVPR*, 2019.

[Li *et al.*, 2020] Rui Li, Qianfen Jiao, et al. Model adaptation: Unsupervised domain adaptation without source data. In *CVPR*, 2020.

[Liang *et al.*, 2019] Jian Liang, Ran He, et al. Distant supervised centroid shift: A simple and efficient approach to visual domain adaptation. *CVPR*, 2019.

[Liang *et al.*, 2020] Jian Liang, Dapeng Hu, et al. Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation. In *ICML*, 2020.

[Liu *et al.*, 2020] Sheng Liu, Jonathan Niles-Weed, et al. Early-learning regularization prevents memorization of noisy labels. *NeruIPS*, 2020.

[Long *et al.*, 2018] Mingsheng Long, Zhangjie Cao, et al. Conditional adversarial domain adaptation. In *NeruIPS*, 2018.

[Müller *et al.*, 2019] Rafael Müller, Simon Kornblith, et al. When does label smoothing help? In *NeurIPS*, 2019.

[Odena *et al.*, 2017] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *ICML*, 2017.

[Pan *et al.*, 2019] Yingwei Pan, Ting Yao, et al. Transferrable prototypical networks for unsupervised domain adaptation. In *CVPR*, 2019.

[Pei *et al.*, 2018] Zhongyi Pei, Zhangjie Cao, et al. Multi-adversarial domain adaptation. In *AAAI*, 2018.

[Peng *et al.*, 2017] Xingchao Peng, Ben Usman, et al. Visda: The visual domain adaptation challenge. *ArXiv*, 2017.

[Saenko *et al.*, 2010] Kate Saenko, Brian Kulis, et al. Adapting visual category models to new domains. In *ECCV*, 2010.

[Saito *et al.*, 2018] Kuniaki Saito, Kohei Watanabe, et al. Maximum classifier discrepancy for unsupervised domain adaptation. In *CVPR*, 2018.

[Saito *et al.*, 2020] Kuniaki Saito, Donghyun Kim, et al. Universal domain adaptation through self supervision. *NeruIPS*, 2020.

[Sankaranarayanan *et al.*, 2018] Swami Sankaranarayanan, Yogesh Balaji, et al. Generate to adapt: Aligning domains using generative adversarial networks. *CVPR*, 2018.

[Snell *et al.*, 2017] Jake Snell, Kevin Swersky, et al. Prototypical networks for few-shot learning. In *NeurIPS*, 2017.

[Tang *et al.*, 2020] Hui Tang, Ke Chen, and Kui Jia. Unsupervised domain adaptation via structurally regularized deep clustering. In *CVPR*, 2020.

[Tzeng *et al.*, 2014] Eric Tzeng, Judy Hoffman, et al. Deep domain confusion: Maximizing for domain invariance. *ArXiv*, 2014.

[Tzeng *et al.*, 2017] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. *CVPR*, 2017.

[van den Oord *et al.*, 2018] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *ArXiv*, 2018.

[Venkateswara *et al.*, 2017] Hemanth Venkateswara, Jose Eusebio, et al. Deep hashing network for unsupervised domain adaptation. *CVPR*, 2017.

[Wei *et al.*, 2016] Pengfei Wei, Yiping Ke, et al. Deep nonlinear feature coding for unsupervised domain adaptation. In *IJCAI*, 2016.

[Wu *et al.*, 2020] Yuan Wu, Diana Inkpen, and Ahmed El-Roby. Dual mixup regularized learning for adversarial domain adaptation. In *ECCV*, 2020.

[Xu *et al.*, 2019] Ruijia Xu, Guanbin Li, et al. Larger norm more transferable: An adaptive feature norm approach for unsupervised domain adaptation. In *ICCV*, 2019.

[Xu *et al.*, 2020] Shoukai Xu, Haokun Li, et al. Generative low-bitwidth data free quantization. In *ECCV*, 2020.

[Yan *et al.*, 2017] Yuguang Yan, W. Li, et al. Learning discriminative correlation subspace for heterogeneous domain adaptation. In *IJCAI*, 2017.

[Yang *et al.*, 2020a] Guanglei Yang, Haifeng Xia, et al. Bidirectional generation for unsupervised domain adaptation. In *AAAI*, 2020.

[Yang *et al.*, 2020b] Shiqi Yang, Yaxing Wang, et al. Unsupervised domain adaptation without source data by casting a bait. *ArXiv*, 2020.

[Zhang *et al.*, 2019a] Yifan Zhang, Hanbo Chen, et al. From whole slide imaging to microscopy: Deep microscopy adaptation network for histopathology cancer image classification. In *MICCAI*, 2019.

[Zhang *et al.*, 2019b] Yuchen Zhang, Tianle Liu, et al. Bridging theory and algorithm for domain adaptation. In *ICML*, 2019.

[Zhang *et al.*, 2020] Yifan Zhang, Y. Wei, et al. Collaborative unsupervised domain adaptation for medical image diagnosis. *IEEE Transactions on Image Processing*, 29:7834–7844, 2020.

[Zhang *et al.*, 2021] Yifan Zhang, Bryan Hooi, et al. Unleashing the power of contrastive self-supervised visual models via contrast-regularized fine-tuning. *ArXiv*, 2021.

# Appendix

In this appendix, we provide the algorithm of inference scheme (Section 5), more implementation details (Section 5), and more experimental results (Section 5).

## A. Inference Details of CPGA

In this section, we present the pseudo-code of CPGA during inference. Specifically, when getting a well-trained CPGA, we can obtain the target prediction based on the feature extractor $G_e$ and the classifier $C_y$. As shown in Algorithm 2, given an input image $\mathbf{x}$, we first capture the corresponding feature $G_e(\mathbf{x})$ and then feed the feature into the classifier $C_y$ to generate the target prediction.

**Algorithm 2** Inference of CPGA

**Require:** Target data $\mathbf{x}$, feature extractor $G_e$ and classifier $C_y$.
1: Extract feature $G_e(\mathbf{x})$ regrading $\mathbf{x}$ using $G_e$;
2: Compute the prediction $C_y(G_e(\mathbf{x}))$ using $C_y$;
3: **Output:** $C_y(G_e(\mathbf{x}))$.

## B. More Implementation Details

**Generator Architecture.** As shown in Table 7, the generator consists of an embedding layer, two FC layers and two deconvolution layers. Similar to ACGAN [Odena *et al.*, 2017], given an input noise $\mathbf{z}\sim U(0,1)$ and a label $\mathbf{y}\in\mathbb{R}^K$, we first map the label into a vector using the embedding layer. After that, we combine the vector with the given noise by a element-wise multiplication and then feed it into the following layers. Since we propose to obtain feature prototypes instead of images, we reshape the output of the generator into a feature vector with the same dimensions as the last FC layer.

**Training.** In the stage one, we train the generator by optimizing $\mathcal{L}_{ce}+\mathcal{L}^p_{con}$. The batchsize is set to 128. We use the SGD optimizer with learning rate $=0.001$. In the stage two, to achieve class-wise domain alignment, we generate feature prototypes for K classes in each epoch.

**Optional Hyper-parameter Selection.** Following [Ganin *et al.*, 2016], we select the hyper-parameters via an unsupervised reverse validation strategy. Such a strategy consists of two steps: (1) We generate source prototypes for K classes and predicted labels for the target domain via a well-trained CPGA. (2) We train another CPGA with pseudo-labeled target data served as the source domain and evaluate the model on the source prototypes. By the end, we obtain the corresponding hyper-parameters based on the best accuracy on source prototypes.

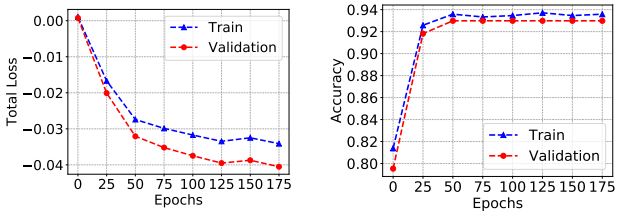
## C. More Experimental Results

**Comparison with State-of-the-art Methods.** We verify the effectiveness of our method on the Office-Home dataset. From Table 8, the results show that: (1) CPGA outperforms all the conventional unsupervised domain adaptation methods, which needs to use the source data. (2) CPGA achieve the competitive performance compared with the state-of-the-art source-free UDA methods, *i.e.,* SHOT [Liang *et al.*, 2020] and BAIT [Yang *et al.*, 2020b]. Besides, we also provide our reimplemented results of the published source-free UDA methods on VisDA and Office-31 based on their published source codes (See Table 9 and Table 11).

**Influence of Hyper-parameters.** In this section, we provide more results for the hyper-parameters $\lambda$ and $\beta$ on VisDA. As shown in Table 10, our method achieves the best performance with the setting $\beta$=0.9 and $\lambda$=5 on VisDA.

**Visualization of Optimization Curve.** Figure 3 shows our method converges well in terms of the total loss and accuracy in the training phase. Also, the curve on the validation set means our method does not suffer from pseudo label noise.

**Robustness Comparisons with BAIT.** As shown in Figure 4, BAIT [Yang *et al.*, 2020b] may overfit to mistaken divisions of certain and uncertain sets, leading to poor generalization abilities. In contrast, our method is more robust and can conquer the issue of pseudo label noise.




(a) Total loss curve (b) Accuracy curve

Figure 3: Optimization curves of CPGA on **Office-31**(A→W).

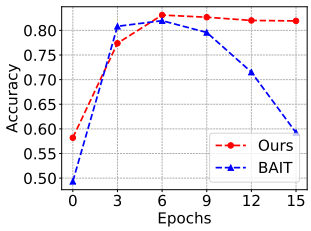



Figure 4: Testing curves of CPGA and BAIT on **VisDA** dataset.

| Backbone Network | | | | | |
|---|---|---|---|---|---|
| Part | Input → Output | Kernel | Padding | Stride | Activation |
| Embedding | $(batch\_size, 1) \rightarrow (batch\_size, 100)$ | - | - | - | - |
| Linear | $(batch\_size, 100) \rightarrow (batch\_size, 1024)$ | - | - | - | ReLU |
| BatchNorm1d | $(batch\_size, 1024) \rightarrow (batch\_size, 1024)$ | - | - | - | - |
| Linear | $(batch\_size, 1024) \rightarrow (batch\_size, \frac{d}{4} * 7 * 7)$ | - | - | - | ReLU |
| BatchNorm1d | $(batch\_size, \frac{d}{4} * 7 * 7) \rightarrow (batch\_size, \frac{d}{4} * 7 * 7)$ | - | - | - | - |
| Reshape | $(batch\_size, \frac{d}{4} * 7 * 7) \rightarrow (batch\_size, \frac{d}{4}, 7, 7)$ | - | - | - | - |
| ConvTranspose2d | $(batch\_size, \frac{d}{4}, 7, 7) \rightarrow (batch\_size, \frac{d}{8}, 6, 6)$ | 2 | 1 | 2 | - |
| BatchNorm2d | $(batch\_size, \frac{d}{8}, 6, 6) \rightarrow (batch\_size, \frac{d}{8}, 6, 6)$ | - | - | - | ReLU |
| ConvTranspose2d | $(batch\_size, \frac{d}{8}, 6, 6) \rightarrow (batch\_size, \frac{d}{16}, 4, 4)$ | 3 | 1 | 2 | - |
| BatchNorm2d | $(batch\_size, \frac{d}{16}, 4, 4) \rightarrow (batch\_size, \frac{d}{16}, 4, 4)$ | - | - | - | ReLU |
| Reshape | $(batch\_size, \frac{d}{16}, 4, 4) \rightarrow (batch\_size, d)$ | - | - | - | - |

Table 7: Detailed architecture of the generator, where $d$ denote the output dimensions, *e.g.,* 2048.

| Method | Source-free | Ar→Cl | Ar→Pr | Ar→Rw | Cl→Ar | Cl→Pr | Cl→Rw | Pr→Ar | Pr→Cl | Pr→Rw | Rw→Ar | Rw→Cl | Rw→Pr | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ResNet-50 [He *et al.*, 2016] | ✗ | 34.9 | 50.0 | 58.0 | 37.4 | 41.9 | 46.2 | 38.5 | 31.2 | 60.4 | 53.9 | 41.2 | 59.9 | 46.1 |
| MCD [Saito *et al.*, 2018] | ✗ | 48.9 | 68.3 | 74.6 | 61.3 | 67.6 | 68.8 | 57.0 | 47.1 | 75.1 | 69.1 | 52.2 | 79.6 | 64.1 |
| CDAN [Long *et al.*, 2018] | ✗ | 50.7 | 70.6 | 76.0 | 57.6 | 70.0 | 70.0 | 57.4 | 50.9 | 77.3 | 70.9 | 56.7 | 81.6 | 65.8 |
| MDD [Zhang *et al.*, 2019b] | ✗ | 54.9 | 73.7 | 77.8 | 60.0 | 71.4 | 71.8 | 61.2 | 53.6 | 78.1 | 72.5 | 60.2 | 82.3 | 68.1 |
| BNM [Cui *et al.*, 2020] | ✗ | 52.3 | 73.9 | 80.0 | 63.3 | 72.9 | 74.9 | 61.7 | 49.5 | 79.7 | 70.5 | 53.6 | 82.2 | 67.9 |
| BDG [Yang *et al.*, 2020a] | ✗ | 51.5 | 73.4 | 78.7 | 65.3 | 71.5 | 73.7 | 65.1 | 49.7 | 81.1 | 74.6 | 55.1 | 84.8 | 68.7 |
| SRDC [Tang *et al.*, 2020] | ✗ | 52.3 | 76.3 | 81.0 | 69.5 | 76.2 | 78.0 | 68.7 | 53.8 | 81.7 | 76.3 | 57.1 | 85.0 | 71.3 |
| PrDA [Kim *et al.*, 2020] | ✓ | 48.4 | 73.4 | 76.9 | 64.3 | 69.8 | 71.7 | 62.7 | 45.3 | 76.6 | 69.8 | 50.5 | 79.0 | 65.7 |
| SHOT [Liang *et al.*, 2020] | ✓ | 56.9 | 78.1 | 81.0 | 67.9 | **78.4** | **78.1** | 67.0 | 54.6 | 81.8 | 73.4 | 58.1 | **84.5** | 71.6 |
| SHOT [Liang *et al.*, 2020] | ✓ | 57.5 | 77.9 | 80.3 | 66.5 | 78.3 | 76.6 | 65.8 | 55.7 | 81.7 | 74.0 | 61.2 | 84.2 | 71.6 |
| BAIT [Yang *et al.*, 2020b] | ✓ | 57.4 | 77.5 | **82.4** | **68.0** | 77.2 | 75.1 | **67.1** | 55.5 | **81.9** | **73.9** | 59.5 | 84.2 | 71.6 |
| BAIT [Yang *et al.*, 2020b] | ✓ | 52.2 | 71.3 | 72.5 | 59.9 | 70.6 | 69.9 | 60.3 | 53.9 | 78.2 | 68.4 | 58.9 | 80.7 | 66.4 |
| CPGA (ours) | ✓ | **59.3** | **78.1** | 79.8 | 65.4 | 75.5 | 76.4 | 65.7 | **58.0** | 81.0 | 72.0 | **64.4** | 83.3 | **71.6** |

Table 8: . Classification accuracies (%) on the Office-Home dataset (ResNet-50). We adopt underline to denote reimplemented results.

| Method | Source-free | plane | bicycle | bus | car | horse | knife | mcycl | person | plant | sktbrd | train | truck | Per-class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SHOT [Liang *et al.*, 2020] | ✓ | 92.6 | 81.1 | 80.1 | 58.5 | 89.7 | 86.1 | 81.5 | 77.8 | 89.5 | 84.9 | 84.3 | 49.3 | 79.6 |
| SHOT [Liang *et al.*, 2020] | ✓ | 88.5 | 85.9 | 77.9 | 49.8 | 90.2 | 90.8 | 82.0 | 79.0 | 88.5 | 84.4 | 85.6 | 50.5 | 79.4 |
| BAIT [Yang *et al.*, 2020b] | ✓ | 93.7 | 83.2 | 84.5 | 65.0 | 92.9 | 95.4 | 88.1 | 80.8 | 90.0 | 89.0 | 84.0 | 45.3 | 82.7 |
| BAIT [Yang *et al.*, 2020b] | ✓ | 93.8 | 75.4 | **86.1** | 64.0 | **93.9** | 96.4 | 88.5 | 81.2 | 88.9 | 88.7 | 86.9 | 39.9 | 82.0 |
| CPGA (ours, 40 epochs) | ✓ | 94.8 | 83.6 | 79.7 | **65.1** | 92.5 | 94.7 | **90.1** | 82.4 | 88.8 | 88.0 | **88.9** | 60.1 | 84.1 |
| CPGA (ours, 400 epochs) | ✓ | **95.6** | **89.0** | 75.4 | 64.9 | 91.7 | **97.5** | 89.7 | **83.8** | **93.9** | **93.4** | 87.7 | **69.0** | **86.0** |

Table 9: Classification accuracies (%) on large-scale VisDA dataset (ResNet-101). We adopt underline to denote reimplemented results.

| Method | Source-free | A→D | A→W | D→W | W→D | D→A | W→A | Avg. |
|---|---|---|---|---|---|---|---|---|
| SHOT [Liang *et al.*, 2020] | ✓ | 93.1 | 90.9 | **98.8** | 99.9 | 74.5 | 74.8 | 88.7 |
| SHOT [Liang *et al.*, 2020] | ✓ | 91.4 | 90.0 | 99.1 | 100.0 | 74.8 | 73.6 | 88.2 |
| BAIT [Yang *et al.*, 2020b] | ✓ | 92.0 | **94.6** | 98.1 | **100.0** | 74.6 | 75.2 | 89.1 |
| BAIT [Yang *et al.*, 2020b] | ✓ | 91.3 | 87.4 | 97.6 | 99.7 | 71.4 | 67.2 | 85.8 |
| CPGA (ours) | ✓ | **94.4** | 94.1 | 98.4 | 99.8 | **76.0** | **76.6** | **89.9** |

Table 11: Classification accuracies (%) on the Office-31 dataset (ResNet-50). We adopt underline to denote reimplemented results.

| $\lambda$ | $\beta$ | | | |
|---|---|---|---|---|
| | 0.5 | 0.7 | 0.9 | 0.99 |
| 3 | 81.2 | 83.0 | 83.9 | 83.0 |
| 5 | 81.3 | 82.2 | 84.1 | 83.2 |
| 7 | 79.7 | 81.6 | 83.3 | 83.0 |

Table 10: Influence of the trade-off parameters $\beta$ and $\lambda$ in terms of per-class accuracy (%) on **VisDA**. The value of $\beta$ is chosen from $[0.5, 0.7, 0.9, 0.99]$ and $\lambda$ is chosen from $[3, 5, 7]$. In each experiment, the rest of hyper-parameters are fixed to the values mentioned in the main paper. We report the results of the model trained on **VisDA** for 40 epochs.