Two-Phase Pseudo Label Densification for Self-training Based Domain Adaptation

Inkyu Shin, Sanghyun Woo, Fei Pan, and In So Kweon(✉)

KAIST, Daejeon, South Korea
{dlsrbgg33,shwoo93,feipan,iskweon77}@kaist.ac.kr

Abstract. Recently, deep self-training approaches emerged as a powerful solution to the unsupervised domain adaptation. The self-training scheme involves iterative processing of target data; it generates target pseudo labels and retrains the network. However, since only the confident predictions are taken as pseudo labels, existing self-training approaches inevitably produce sparse pseudo labels in practice. We see this is critical because the resulting insufficient training-signals lead to a sub-optimal, error-prone model. In order to tackle this problem, we propose a novel **T**wo-phase **P**seudo **L**abel **D**ensification framework, referred to as **TPLD**. In the first phase, we use sliding window voting to propagate the confident predictions, utilizing intrinsic spatial-correlations in the images. In the second phase, we perform a confidence-based easy-hard classification. For the easy samples, we now employ their full pseudo-labels. For the hard ones, we instead adopt adversarial learning to enforce hard-to-easy feature alignment. To ease the training process and avoid noisy predictions, we introduce the bootstrapping mechanism to the original self-training loss. We show the proposed TPLD can be easily integrated into existing self-training based approaches and improves the performance significantly. Combined with the recently proposed CRST self-training framework, we achieve new state-of-the-art results on two standard UDA benchmarks.

Keywords: Unsupervised domain adaptataion · Self-training

1 Introduction

Unsupervised domain adaptation (UDA) aims to transfer knowledge learned from the label-rich source domain to an unlabeled new target domain. It is a practical and crucial problem as it could be beneficial for various label-scarce real-world scenarios, e.g., simulated learning for robots [11] or autonomous driving [31]. In this paper, we focus on the UDA for semantic segmentation, aiming to adopt a source segmentation model to a target domain without any labels.

Electronic supplementary material The online version of this chapter (https://doi.org/10.1007/978-3-030-58601-0_32) contains supplementary material, which is available to authorized users.

A. Vedaldi et al. (Eds.): ECCV 2020, LNCS 12358, pp. 532–548, 2020.
https://doi.org/10.1007/978-3-030-58601-0_32

The dominant paradigm in UDA is based on ***adversarial learning*** [5,17,19,25,36,37]. In particular, it minimizes both (source domain) task-specific loss and domain adversarial loss. The method thus retains good performance on the source domain task, and at the same time, can bridge the gap between source and target feature distributions. While the adversarial learning has achieved great success in UDA, recently another line of studies using ***self-training*** emerged [39,40]. Self-training generates a set of pseudo labels corresponding to high prediction scores in the target domain and then re-trains the network based on the generated pseudo labels. Recently, Zou & Yu have proposed two seminal works on CNN-based self-training methods; class balanced self-training (CBST) [39], and confidence regularized self-training (CRST) [40]. Unlike adversarial learning methods which utilize two separate losses, CBST presents a single unified self-training loss. It allows learning of domain-invariant features and classifiers in an end-to-end manner, both from labeled source data and pseudo labeled target data. CRST further generalizes the feasible space of pseudo labels and adopts regularizer. These self-training methods show state-of-the-art results in multiple UDA settings. However, we observe that its internal pseudo label selection tends to excessively cut-out the predictions, which often leads to sparse pseudo labels. We argue that sparse pseudo labels significantly miss meaningful training signals, and thus, the final model may deviate from the optimal solution eventually. A natural way to obtain dense pseudo labels is by lowering the selection threshold. However, we observe this naive approach brings noisy, unconfident predictions at an early stage, and this accumulates and propagates the errors.

To effectively address this issue, we present a two-step, gradual pseudo label densification method. The overview is shown in Fig. 1. In the first phase, we use sliding window voting to propagate the confident predictions, utilizing the intrinsic spatial correlations in the images. In the second phase, we perform an easy-hard classification using a proposed image-level confidence score. Our intuition is simple: As the model improves over time, its predictions can be trusted more. Thus, if the model in the second stage is confident with their prediction, we now do not zero out them. Indeed, we empirically observe that the confident, easy samples are near to the ground truth and vice versa. This motivates us to utilize full pseudo labels for the easy samples, while for the hard samples, we enforce adversarial loss to learn hard-to-easy adaption. Meanwhile, to tackle noisy labels effectively for both first and second phase training, we introduce the bootstrapping mechanism into the self-training loss function. By connecting all together, we build a two-phase pseudo label densification framework called TPLD. Since our method is general, we can easily apply TPLD to the existing self-training based approaches. We show consistent improvements over the strong baselines. Finally, we achieve new state-of-the-art performances on two standard UDA benchmarks.

We summarize our contributions as follows:

1. To our best knowledge, it is the first time that pseudo label densification is formally defined and explored in the self-training based domain adaptation.

2. We present a novel two-phase pseudo label densification framework, called **TPLD**. In particular, for the first phase, we introduce voting-based densification method. For the second phase, we propose an easy-hard classification-based densification method. Both phases are complementary in constructing an accurate self-training model.
3. We propose a new objective function to ease the training. Specifically, we re-formulate the original self-training loss function by incorporating the bootstrapping mechanism.
4. We conduct extensive ablation studies to thoroughly investigate the impact of our proposals. We apply TPLD to the various existing self-training approaches and achieve new state-of-the-art results on two standard UDA benchmarks.

2 Related Works

Domain Adaptation is a classic problem in computer vision and machine learning. It aims to alleviate the performance drop caused by the distribution mismatch in cross-domains. It is mostly investigated in image classification problems by both conventional methods [8,12,13,20,22] and deep CNN-based methods [9,10,21,24,27,29,33]. Besides image recognition, domain adaptation is recently being applied other vision tasks such as object detection [4], depth estimation [1], and semantic segmentation [17]. In this work, we are particularly interested in *unsupervised* domain adaptation for the task of semantic segmentation. The primary approach is to minimize the discrepancy between source and target feature distribution using adversarial learning. This type of approaches is studied on three different levels in practice: input-level alignment [5,17,28,34], intermediate feature-level alignment [18,19,23,25,37], and output-level alignment [36]. Although these methods are proven to be effective, the potentially meaningful training signals from the target domain are under-utilized. Therefore, self-training based UDA approaches [39,40], described next, emereged recently and came to dominate the performance quickly.

Self-training has been initially explored in semi-supervised method [14,38]. Recently, two seminar works [39,40] have been presented for UDA semantic segmentation. Unlike adversarial learning approaches, these methods explicitly explore the supervision signals from the target domain. The key idea is to use the prediction from the source-trained model as pseudo-labels for the unlabeled data and re-trains the current model in the target domain. CBST [39] extends this basic idea with class balancing strategy and spatial priors. CRST [40] further adds regularization term in the loss function to prevent overconfident predictions. In this paper, we also investigate the self-training framework. However, different from the previous studies, we see that the spare pseudo label problem is a fundamental limitation of self-training. We empirically found that these sparse pseudo-labels inhibit effective learning; thus, the model significantly deviates from the optimal. We, therefore, propose to densify the sparse pseudo-labels in a two-step gradually. Also, we present a new loss function to handle noisy pseudo

labels and reduce optimization difficulties during training. We empirically confirm that our proposals greatly improve the strong state-of-the-art baselines with healthy margins.

3 Preliminaries

3.1 Problem Setting

Following the common UDA setting, we have full access to the data and labels, $(\mathbf{x_s}, \mathbf{y_s})$, in the labeled source domain. In contrast, in the unlabeled target domain, we can only utilize the data, $\mathbf{x_t}$. In self-training, we thus train the network to infer pseudo target label, $\hat{\mathbf{y}}_\mathbf{t} = (\hat{y}_t^{(1)}, ..., \hat{y}_t^{(K)})$, where K denotes the total number of classes.

3.2 Self-training for UDA

We first revisit the general self-training loss function [40] below:

$$\begin{aligned} \min_{\mathbf{w}, \hat{\mathbf{Y}}_\mathbf{T}} \mathcal{L}_{st}(\mathbf{w}, \hat{\mathbf{Y}}_\mathbf{T}) = & -\sum_{s \in S} \sum_{k=1}^{K} y_s^{(k)} \log p(k|\mathbf{x_s}; \mathbf{w}) \\ & - \sum_{t \in T} [\sum_{k=1}^{K} \hat{y}_t^{(k)} \log \frac{p(k|\mathbf{x_t}; \mathbf{w})}{\lambda_k} - \alpha r_c(\mathbf{w}, \hat{\mathbf{Y}}_\mathbf{T})] \\ & s.t.\, \hat{y}^t \in \Delta^{K-1} \cup \{\mathbf{0}\}, \forall t \end{aligned} \tag{1}$$

$\mathbf{x_s}$ denotes an image in source domain indexed by $s = 1, 2, ..., S$, and $\mathbf{x_t}$ is an image in target domain indexed by $t = 1, 2, ..., T$. $y_s^{(k)}$ is ground truth source label for class k, and $\hat{y}_t^{(k)}$ is generated pseudo target label. Note that feasible set of pseudo-label is the union of $\{\mathbf{0}\}$ and a probability simplex Δ^{K-1} (i.e., continuous). $\mathbf{w}$ is the network weights, and $p(k|\mathbf{x}; \mathbf{w})$ indicates the classifier's softmax probability for class k. λ_k is a parameter, controlling pseudo-label selection [39]. $\sum_{t \in T} r_c(\mathbf{w}, \hat{\mathbf{Y}}_\mathbf{T})$ is the confidence regularizer and $\alpha \geq 0$ is the weight coefficient.

We can better understand the Eq. (1) by dividing it into three terms; The first term is model training on source domain with source labels, y_s. The second term is model re-training on target domain with generated target pseudo labels, $\hat{y}_t$. The last term is confidence regularization, $\alpha r_c(\mathbf{w}, \hat{\mathbf{Y}}_\mathbf{T})$, which prevents overconfident predictions of target pseudo-labels. The first two terms are equivalent to the CBST formula [39]. With the additional confidence regularization term, we come up with the CRST formula [40]. In general, there are two types of regularization: label-regularization (e.g., LRENT) and model regularization (e.g., MRKLD).

To minimize Eq. (1), the optimization algorithm alternatively takes block coordinate descent on both 1) pseudo-label generation and 2) network retraining.

For solving step 1), there is an optimizer formulated as:

$$some = \begin{cases} 1, & \text{if } k = \arg\max_k \{\frac{p(k|\mathbf{x_t};\mathbf{w})}{\lambda_k}\} \\ & \text{and } p(k|\mathbf{x_t};\mathbf{w}) > \lambda_k \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

If the prediction is confident, $p(k|\mathbf{x_t};\mathbf{w}) > \lambda_k$, it is selected and labeled as a class $k^* = \arg\max_k \{\frac{p(k|\mathbf{x_t};\mathbf{w})}{\lambda_k}\}$. Otherwise, the less confident predictions are set to zero vector $\mathbf{0}$. For each class k, we determine λ_k by the confidence value that is selected from the most confident p portion of class k predictions in the entire target set [39]. To avoid selecting unconfident predictions at the early stage, the hyperparameter p is usually set to a low value (i.e., 0.2), and is gradually increased for each additional round. To solve step 2), we use typical gradient-based methods (e.g., SGD). For more details, please refer to the original papers [39,40].

We see the current self-training approach simply zeroes out the less confident predictions and in turn generates sparse pseudo labels. We argue that this limits the power of model representations and could produce sub-optimal model. Motivated by our empirical observations, we attempt to densify the sparse pseudo labels gradually, and avoid noisy predictions. In this work, we propose TPLD, which alleviates these fundamental issues successfully. We show the TPLD can be applied to any type of existing self-training based frameworks, and can consistently boost the performance significantly.

3.3 Noisy Label Handling

To handle noisy predictions, Reed et al. [30] proposed bootstrapping loss. It is a weighted sum of the standard cross-entropy loss and the (self) entropy loss. In this work, we apply it to the self-training formula as:

$$\sum_{t \in T} \sum_{k=1}^{K} [\beta \hat{y}_t^{(k)} + (1-\beta) \frac{p(k|\mathbf{x_t};\mathbf{w})}{\lambda_k}] \log \frac{p(k|\mathbf{x_t};\mathbf{w})}{\lambda_k} \tag{3}$$

Intuitively, it simultaneously encourages the model to predict the correct (pseudo) target label and have high confidence on its prediction.

4 Method

The overview of our two-phase pseudo-label densification algorithm is shown in Fig. 1. For the first phase, we design a sliding window-based voting method to propagate the confident predictions. After enough training, we enter the second phase. Here, we present confidence based easy-hard classification and hard/easy adversarial learning. For both phases, we use the proposed bootstrapped self-training loss (Eq. (3)). We detail each phase below.

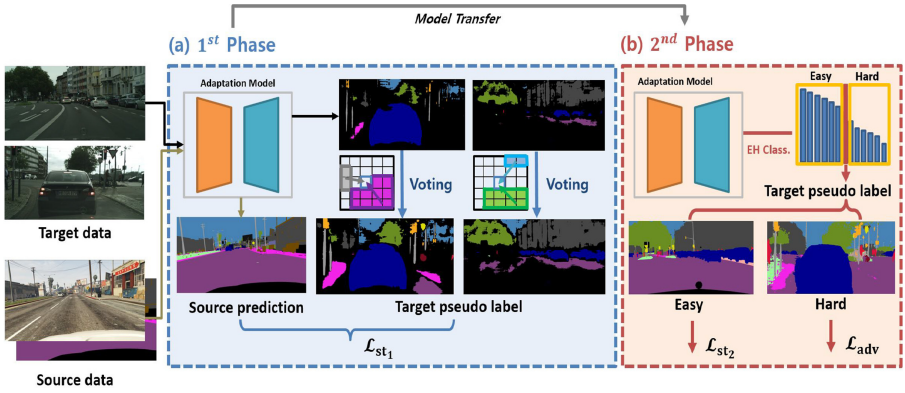

Fig. 1. The overview of the proposed two-phase pseudo-label densification framework. (a) *The first phase* utilizes the sliding window based voting in which it propagates neighbor confident predictions to fill in the unlabeled pixels. We use $\mathcal{L}_{st_1}$ to train the model in the first phase. (b) *The second phase* employs confidence-based easy-hard classification (EH class.) along with the hard-to-easy adversarial learning. This allows the model to utilize full pseudo labels for easy samples while pushing hard samples to be like easy. We use both $\mathcal{L}_{st_2}$ and $\mathcal{L}_{adv}$ to train the model in the second phase.

4.1 1st Phase: Voting Based Densification

As mentioned above, pseudo labels are generated only when the sample's prediction is confident (Eq. (2)). Specifically, the most confident p portion of predictions are selected class-wise. Because the hyperparameter p is set to a low value in practice, pseudo labels are inherently sparse during training. To overcome this issue, we present a sliding window-based voting, in which it relaxes the current hard-thresholding and propagates the confident predictions based on the intrinsic spatial correlations in the image. We attempt to utilize the fact that neighboring pixels tend to be alike. To efficiently employ this local spatial regularity in the image, we adopt the sliding-window approach. We detail the process in Fig. 2. Given the window with the unlabeled pixel at the center, we gather the neighboring confident prediction values (voting). To be more specific, for the unlabeled pixel, we first obtain the top two competing classes (i.e., classes with highest and second-highest prediction values, which would have caused ambiguity in deciding the correct label) (Fig. 2-1), and then pool the neighboring confident values for these classes (Fig. 2-2). The spatially-pooled prediction values are then weighted sum with the original prediction values (Fig. 2-3). Among the two values, we choose the bigger one. Finally, if it is above the threshold, we select the according class as a pseudo label. Note that, we use normalized prediction values (i.e., $\frac{p(k|\mathbf{x_t};\mathbf{w})}{\lambda_k}$) during the voting process, thus the thresholding criteria is $\frac{p(k|\mathbf{x_t};\mathbf{w})}{\lambda_k} > 1$. Otherwise, it continues to be a zero vector.

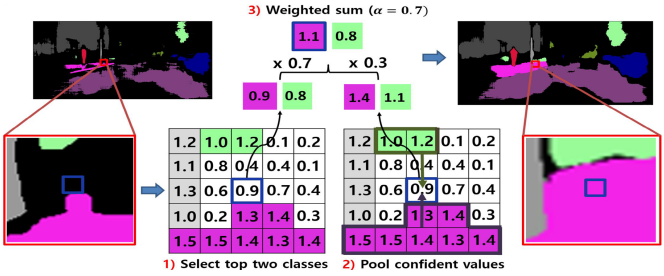

Fig. 2. The overall procedure of the voting-based densification. We describe the process in three steps. 1) We find the top two competing classes on the unlabeled pixel, 2) We pool neighboring confident values for these classes, 3) We combine the original prediction values and the pooled values (weighted-sum with hyperparameter α). We pick the bigger one and assign the corresponding class if it passes the thresholding criteria. We repeat this process by sliding the window across the images.

We call the above whole process voting-based densification. We abbreviate it as **Voting**. We iterate over total 3 times with the window size of 57×57. Those hyperparameters are set through the parameter analysis (see Table 4b). The qualitative voting results are shown in Fig. 3. We can clearly see that the initial sparse pseudo label gradually becomes dense. The pseudo label generation in the 1st phase can be summarized as:

$$\hat{y}_t^{(k)*} = \begin{cases} 1, & \text{if } k = \arg\max_k \{\frac{p(k|\mathbf{x_t};\mathbf{w})}{\lambda_k}\} \\ & \text{and } \; p(k|\mathbf{x_t};\mathbf{w}) > \lambda_k \\ \textbf{Voting}(\frac{p(k|\mathbf{x_t};\mathbf{w})}{\lambda_k}), & \text{otherwise} \end{cases} \tag{4}$$

Objective Function for the 1st Phase. To effectively train the model under the existence of noisy pseudo labels, we introduce bootstrapping (Eq. (3)) in our final objective function. The original self-training objective function can be re-formulated as the following:

$$\begin{aligned} \min_{\mathbf{w},\hat{\mathbf{Y}}_\mathbf{T}} \mathcal{L}_{st_1}(\mathbf{w}, \hat{\mathbf{Y}}_\mathbf{T}) = & -\sum_{s\in S}\sum_{k=1}^{K} y_s^{(k)} \log p(k|\mathbf{x_s};\mathbf{w}) \\ & -\sum_{t\in T}[\sum_{k=1}^{K}\{\beta\hat{y}_t^{(k)} \log + (1-\beta)\frac{p(k|\mathbf{x_t};\mathbf{w})}{\lambda_k}\} \log \frac{p(k|\mathbf{x_t};\mathbf{w})}{\lambda_k} \\ & -\alpha r_c(\mathbf{w}, \hat{\mathbf{Y}}_\mathbf{T})] \\ & s.t.\, \hat{y}^t \in \Delta^{K-1} \cup \{\mathbf{0}\}, \forall t \end{aligned} \tag{5}$$

As a result, the target domain training benefits from both densified pseudo label and bootstrapped training.

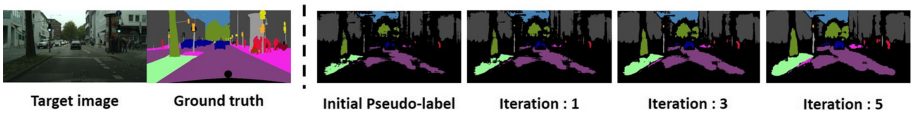

Fig. 3. Voting based densification results by iteration. We can see the initial sparse pseudo label becomes dense as iteration number increases. Though it may bring noisy predictions. We set the total iteration number to 3 after conducting parameter analysis in Table 4.

4.2 2nd Phase: Easy-Hard Classification Based Densification

As the predictions of model can be trusted more over time, we now attempt to use full pseudo-labels. One may attempt to use voting multiple times for full densificaiton. However, the experimental evidence shown in Table 4b proves that it is hard for voting to generate fully densified pseudo labels. By construction, the voting is operated with a local window, which can only capture and process local predictions. Thus, iterating the voting process multiple times brings some extent of smoothing effect and noisy predictions. We, therefore, present another phase which enables full-pseudo label training. Our key idea is to consider the confidence on image-level and classify the images into two groups: easy and hard. For the easy, confident samples, we utilize their full predictions, while for the hard samples, we instead enforce hard-to-easy adaption. Indeed, we observe that the easy samples are near to the ground truth and vice versa (see Fig. 4).

To reasonably categorize target samples into easy and hard, we present effective criteria. For a particular image $\mathbf{t}$, we define a confidence score as $conf_{\mathrm{t}} = \frac{1}{K'} \sum_{k=1}^{K'} \frac{N_{\mathrm{t}}^{k*}}{N_{\mathrm{t}}^{k}} \cdot \frac{1}{\lambda_k}$, where N_{t}^{k} is the total number of pixels predicted as class k. Among N_{t}^{k}, we count the number of pixels that have higher prediction values than the class-wise thresholding value λ_k [39], and is set to N_{t}^{k*}. As a result, the ratio $\frac{N_{\mathrm{t}}^{k*}}{N_{\mathrm{t}}^{k}}$ indicates how well the model predicts confident values for each class k. We average these values with K', which is the total number of (predicted) confident classes. Thus, the higher the value, we can say that the model is more confident with that target image (i.e., easy). Note that, we multiply $\frac{1}{\lambda_k}$ to avoid sampling too easy images and instead encourage sampling of images with rare classes. We compute these confidence scores for every target image. In practice, we picked up the top q portion as easy samples and consider the rest as hard samples for the training. We initially set q to 30% and add 5% in each round.

Objective Function for the 2nd Phase. After classifying target images into easy and hard samples, we apply different objective functions to each. For the easy samples, we utilize full pseudo label predictions and employ bootstrapping loss for training (Eq. 3). For the hard samples, we instead adopt adversarial learning to push hard examples to be like easy samples (i.e., feature alignment). We describe the details below.

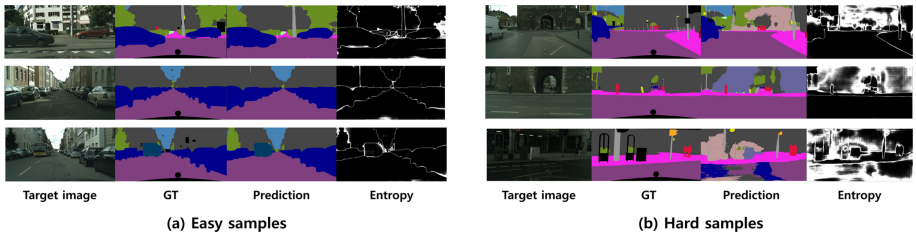

Fig. 4. Qualitative easy and hard samples. For the illustration, we randomly selected three samples from each. Note that easy samples are near to the ground truth with low entropy values, whereas hard samples are far from the ground truth and have high entropy values. Therefore, in the second phase, we train easy samples with their full-pseudo labels and make hard samples to be easy using adversarial loss.

Easy Sample Training. To effectively generate full pseudo labels, we calibrate the prediction values. Specifically, the full pseudo-label generation of easy samples is formulated as:

$$\hat{y}_{t_e}^{(k)*} = \begin{cases} 1, & \text{if } k = \arg\max_k \{\frac{p(k|x_t;w)}{\lambda_k}\} \\ & \text{and } \; p(k|\boldsymbol{x_t; w}) > \lambda_k \\ \left(\frac{p(k|\boldsymbol{x_t;w})}{\lambda_k}\right)^{\gamma}, & \text{otherwise.} \end{cases} \tag{6}$$

Note that the prediction value is calibrated with the hyper parameter γ, which is set to 2 empirically (see Table 4e). We then train the model using the following bootstrapping loss:

$$\begin{aligned} &\min_{\mathbf{w}, \hat{\mathbf{Y}}_\mathbf{T}} \mathcal{L}_{st_2}(\mathbf{w}, \hat{\mathbf{Y}}_\mathbf{T}) \\ &= -\sum_{t \in T} [\sum_{k=1}^{K} \{\beta \hat{y}_t^{(k)} \log + (1-\beta) \frac{p(k|\mathbf{x_t}; \mathbf{w})}{\lambda_k}\} \log \frac{p(k|\mathbf{x_t}; \mathbf{w})}{\lambda_k} \\ &s.t. \, \hat{y}^t \in \Delta^{K-1} \cup \{\mathbf{0}\}, \forall t \end{aligned} \tag{7}$$

Hard Sample Training. To minimize the gap between easy (e) and hard (h) samples in the target domain, we propose intra-domain adversarial loss, $\mathcal{L}_{adv}$. In order to align the feature from hard to easy, the discriminator D_{intra} is trained to discriminate that the target weighted self-information map I_t [37] is whether from easy samples or hard samples. The learning objective of the discriminator is:

$$\min_{\theta_{D_{intra}}} \frac{1}{|e|} \sum_e L_{D_{intra}}(I_e, 1) + \frac{1}{|h|} \sum_h L_{D_{intra}}(I_h, 0) \tag{8}$$

and the adversarial objective to train the segmentation network is:

$$\min_{\theta_{seg}} \frac{1}{|h|} \sum_h L_{D_{intra}}(I_h, 1) \tag{9}$$

5 Experiments

5.1 Dataset

We evaluate our model on the most common adaptation benchmarks. 1) GTA5 [31] to Cityscapes [6] and 2) SYNTHIA [32] to Cityscapes. GTA5 and SYNTHIA contain 24966 and 9,400 synthetic images, respectively. Following the standard protocols, we adapt the model to the Cityscapes training set and evaluate the performance on the validation set.

5.2 Implementation Details

To push the state-of-the-art benchmark performances, we apply TPLD to the CRST-MRKLD framework [40]. For the backbones, we use VGG-16 [35] and ResNet-101 [15]. For the segmentation models, we adopt different versions of deeplab; deeplab-v2 [2] and deeplab-v3 [3]. We pretrain the model on ImageNet [7] and fine-tune on source domain images using SGD. We train the model total 9 rounds: 6 rounds for the first phase training and 3 rounds for the second phase training. The detailed training settings are the followings: For the source domain pre-training, we use learning rate of 2.5×10^{-4}, weight decay of 5×10^{-4}, momentum of 0.9, batch size of 2, patch size of 512×1024, multiscale training augmentation (0.5–1.5), and horizontal flipping. For the self-training, we adopt SGD with the learning rate of 5×10^{-5}.

5.3 Main Results

GTA5 → Cityscapes: Table 1 summarizes the adaptation performance of TPLD and other state-of-the-art methods [25,36,37,39,40]. We can obviously see that TPLD outperforms state-of-the-art approaches in all cases. For example, with Deeplab-v2 and ResNet-101 backbone, our TPLD significantly outperforms CRST by 4.2%. Moreover, to analyze the effect on rare classes, we also put rare-class mIoU. With the R-mIoU metric, we see the improvement is even much higher; 4.8%. We provide qualitative results in Fig. 5. Clearly, our final model generates the most visually pleasurable results.

SYNTHIA → Cityscapes: Table 2 shows the adaptation results with SYNTHIA. Our approach again achieves the best performance among all the other methods. Specifically, with Deeplab-v3 and ResNet101 backbone, we greatly improve the baseline performance of 48.1% mIoU to 55.7% mIoU.

Combining with Existing Self-training Methods. We see the proposed TPLD is general, thus can be easily applied to the existing self-training based methods. In this experiment, we combine the TPLD with three different self-training approaches: CBST [39], CRST with label regularization (LRENT) [40], and CRST with model regularization (MRKLD) [40]. The results are summarized in Table 3. We observe that TPLD consistently improves the performance of

Table 1. Experimental results on GTA5 → Cityscapes. "V" and "R" denote VGG-16 and ResNet-101 respectively. We highlight the rare classes [25] and compute Rare class mIoU (R-mIoU) as well.

GTA5 → Cityscapes																						
Method	Seg Model	Road	SW	Build	Wall	Fence	Pole	TL	TS	Veg.	Terrain	Sky	PR	Rider	Car	Truck	Bus	Train	Motor	Bike	mIoU	R-mIoU
Source	Deeplabv2-V	52.6	20.7	56.0	6.0	9.8	22.9	8.1	1.4	77.2	11.0	**35.0**	41.5	2.7	52.1	2.1	0.0	0.0	4.7	0.3	21.3	5.8
CBST [39]		**84.2**	41.4	71.9	15.5	**18.1**	**30.8**	25.4	9.2	77.6	15.2	29.6	**49.3**	6.0	78.0	**4.0**	4.5	0.3	10.4	11.6	30.7	12.6
CRST(MRKLD) [40]		81.7	46.1	70.2	10.7	11.2	30.4	26.9	**15.8**	75.4	18.3	24.8	48.6	10.9	77.8	2.9	13.3	1.1	10.7	31.4	32.0	15.3
CRST(MRKLD) + TPLD		83.5	**49.9**	**72.3**	**17.6**	10.7	29.6	**28.3**	9.0	**78.2**	**20.1**	25.7	47.4	**13.3**	**79.6**	3.3	**19.3**	**1.3**	**14.3**	**33.5**	**34.1**	**16.7**
Adapt-SegMap [36]	Deeplabv2-R	86.5	36.0	79.9	23.4	23.3	35.2	14.8	14.8	83.4	33.3	75.6	58.5	27.6	73.7	32.5	35.4	3.9	30.1	28.1	42.4	25.2
CLAN [25]		87.0	27.1	79.6	27.3	23.3	28.3	**35.5**	24.2	83.6	27.4	74.2	58.6	28.0	76.2	33.1	36.7	6.7	31.9	31.4	43.2	27.8
ADVENT [37]		89.9	36.5	81.2	29.2	**25.2**	28.5	32.3	22.4	83.9	34.0	77.1	57.4	27.9	83.7	29.4	39.1	1.5	28.4	23.3	43.8	26.8
Source	Deeplabv2-R	71.3	19.2	69.1	18.4	10.0	35.7	27.3	6.8	79.6	24.8	72.1	57.6	19.5	55.5	15.5	15.1	11.7	21.1	12.0	33.3	18.2
CBST [39]		91.8	53.5	80.5	32.7	21.0	34.0	28.9	20.4	83.9	34.2	80.9	53.1	24.0	82.7	30.3	35.9	16.0	25.9	42.8	45.9	28.9
CRST(MRKLD) [40]		91.3	56.1	79.8	30.6	18.9	39.0	35.1	24.0	84.2	30.0	74.0	**62.1**	**28.2**	82.6	23.6	31.8	24.2	32.2	46.3	47.0	30.3
CRST(MRKLD) + TPLD		**94.2**	**60.5**	**82.8**	**36.6**	16.6	**39.3**	29.0	**25.5**	**85.6**	**44.9**	**84.4**	60.6	27.4	**84.1**	**37.0**	**47.0**	**31.2**	**36.1**	**50.3**	**51.2**	**35.1**
Source	Deeplabv3-R	80.3	17.6	75.8	18.0	24.5	19.7	34.9	19.0	83.2	15.8	63.7	57.2	22.8	73.4	36.6	21.0	0.0	19.0	**0.1**	35.9	19.3
CBST [39]		**86.9**	33.9	**80.0**	28.8	**26.2**	30.2	36.9	20.4	84.6	16.3	72.1	53.3	19.8	82.8	34.1	43.8	0.0	13.0	0.0	40.2	22.5
CRST(MRKLD) [40]		85.9	40.4	76.9	27.5	21.6	**35.0**	**39.0**	**25.6**	84.0	20.2	71.8	55.3	23.2	83.2	**38.8**	43.2	0.0	10.3	0.0	41.2	23.7
CRST(MRKLD) + TPLD		83.2	**46.3**	74.9	**29.8**	21.3	33.1	36.0	24.2	**86.7**	**43.2**	**87.1**	**58.7**	**24.0**	**84.0**	36.9	**49.7**	0.0	**29.7**	0.0	**44.7**	**27.3**

Table 2. Experimental results on SYNTHIA → Cityscapes. mIoU∗ is computed with 13 classes out of total 16 classes except the classes with ∗.

SYNTHIA → Cityscapes																			
Method	Seg Model	Road	SW	Build	Wall∗	Fence∗	Pole∗	TL	TS	Veg.	Sky	PR	Rider	Car	Bus	Motor	Bike	mIoU	mIoU∗	R-mIoU
Source	Deeplabv2-V	41.5	16.6	38.3	0.2	0.0	22.6	0.1	4.9	66.5	64.7	44.9	1.7	60.7	3.3	0.0	0.6	22.9	26.4	4.3
CBST [39]		75.7	32.3	70.2	3.5	0.0	28.6	1.4	9.0	79.8	65.6	52.9	13.7	65.8	**9.1**	**1.5**	36.4	34.1	39.5	11.5
CRST(MRKLD) [40]		75.1	33.5	70.8	5.6	0.0	28.7	**2.0**	**9.7**	78.9	**72.5**	51.7	11.6	63.4	7.3	1.4	38.6	34.4	39.7	11.7
CRST(MRKLD) + TPLD		**81.3**	**34.5**	**73.3**	11.9	0.0	26.9	0.2	6.3	**79.9**	71.2	**55.1**	**14.2**	**73.6**	5.7	0.5	**41.7**	**36.0**	**41.3**	**11.9**
Adapt-SegMap [36]	Deeplabv2-R	84.3	42.7	77.5	-	-	-	4.7	7.0	77.9	82.5	54.3	21.0	72.3	32.2	18.9	32.3	-	46.7	-
ADVENT [37]		**87.0**	44.1	79.7	9.6	0.6	24.3	4.8	7.2	80.1	**83.6**	56.4	23.7	72.7	32.6	12.8	33.7	40.8	47.6	16.6
CLAN [25]		81.3	37.3	80.1	-	-	-	16.1	13.7	78.2	81.5	53.4	21.2	73.0	32.9	22.6	30.7	-	47.8	-
Source	Deeplabv2-R	45.9	21.4	63.0	7.3	0.0	33.6	4.5	14.4	**81.6**	79.7	55.3	16.7	67.5	21.3	7.5	19.0	33.7	38.3	13.8
CBST [39]		68.0	29.9	76.3	10.8	1.4	33.9	**22.8**	29.5	77.6	78.3	60.6	28.3	81.6	23.5	18.8	39.8	42.6	48.9	23.2
CRST(MRKLD) [40]		67.7	32.2	73.9	10.7	1.6	37.4	22.2	**31.2**	80.8	80.5	**60.8**	**29.1**	82.8	25.0	19.4	**45.3**	43.8	50.1	24.7
CRST(MRKLD) + TPLD		80.9	**44.3**	**82.2**	19.9	0.3	40.6	20.5	30.1	77.2	80.9	60.6	25.5	**84.8**	**41.1**	**24.7**	43.7	**47.3**	**53.5**	**27.4**
Source	Deeplabv3-R	45.5	19.0	71.3	6.2	0.0	27.4	11.3	15.3	79.4	79.4	58.3	9.2	79.7	33.0	6.0	8.8	34.4	39.7	13.0
CBST [39]		45.2	19.4	**81.8**	15.7	0.2	33.3	**20.8**	24.9	85.0	82.2	64.6	26.7	**84.8**	48.8	22.9	43.9	43.8	50.1	26.4
CRST(MRKLD) [40]		52.3	21.9	80.0	17.2	0.8	32.4	17.9	**31.1**	84.8	83.5	63.7	28.5	83.1	37.2	19.1	52.5	44.1	50.4	26.3
CRST(MRKLD) + TPLD		**70.9**	**29.5**	80.6	18.4	0.4	26.6	19.9	30.9	**85.5**	**86.3**	**66.0**	**32.9**	84.4	**51.1**	**29.3**	**56.2**	**48.1**	**55.7**	**29.5**

all the baselines. The positive results imply that the sparse pseudo-label is indeed a fundamental problem in self-training, and the previous works notably overlooked this problem. We show that the proposed concept of two-phase pseudo-label densificaiton effectively addresses the issue.

5.4 Ablation Study

Lowering the Selection Threshold of CRST. A straightforward way to generate dense pseudo labels is by lowering the selection threshold (i.e., increasing p) of self-training models. We summarize the results in Table 4a. Since the scheme brings unconfident predictions at an early stage, either limited improvement ($p = 0.4$, 47.0 → 47.1 mIoU) or worse performance is obtained ($p = 0.6$, 47.0 → 45.7 mIoU). Compared to these naive baselines, our TPLD shows significant improvement (47.0 → 51.2 mIoU).

Table 3. Performance improvements in mIoU of integrating our TPLD with existing self-training adaptation approaches. We use the Deeplabv2-R segmentation model.

GTA5 → Cityscapes (19 categories)				SYNTHIA → Cityscapes (16 categories)			
Method	Base+	TPLD	△	Method	Base+	TPLD	△
CBST [39]	45.9	47.8	+1.9	CBST [39]	42.6	45.6	+3.0
CRST(LRENT) [40]	45.9	47.3	+1.4	CRST(LRENT) [40]	42.7	47.0	+4.3
CRST(MRKLD) [40]	47.0	51.2	+4.2	CRST(MRKLD) [40]	43.8	47.3	+3.5

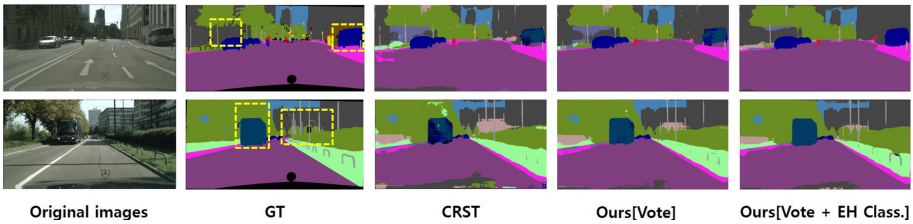

Fig. 5. Qualitative results on GTA5 → Cityscapes. We can clearly see that our full model generates the most visually pleasable results.

Framework Design Choices. The main components of our framework design are the two-phase pseudo label densification. The ablation results are shown in Table 4a. If we drop the voting stage, the model is trained alone with the easy-hard classification stage. However, using full pseudo labels without any proper early-stage training introduces too noisy training signals (51.2 → 38.1 mIoU). If we drop the easy-hard classification stage, the model misses a chance to receive rich training signals from the full pseudo labels (51.2 → 49.5 mIoU). We also explore the effect of ordering. We observe that the voting-first method performs better than the easy-hard classification-first method (51.2 vs. 49.1 mIoU). This implies that gradual densification is indeed important for stable model training.

Effect of $\frac{1}{\lambda_k}$ in Confidence Score $conf_t$. We suggest to multiply $\frac{1}{\lambda_k}$ in computing the confidence score $conf_t$. The rationale behind this is to oversample the images, which include rare classes, and thus prevent the learning from being biased by images composed of obvious frequent classes. The results without and with the $\frac{1}{\lambda_k}$ are (50.5 vs 51.2 mIoU) and (33.7 vs 35.1 R-mIoU). This demonstrates the efficacy of incorporating $\frac{1}{\lambda_k}$.

5.5 Parameter Analysis

Here, we conduct experiments to decide optimal hyper-parameters in our framework. For the first phase, we have a total of three hyper-parameters; voting field size, voting iteration number, and α. In Table 4b, we conduct a grid search on the first two, and we obtain the best result with voting field 57, and voting

Table 4. Results of ablation studies.

GTA5 → Cityscapes

Model	p	Voting	EH	Class.	mIoU
CRST	0.2		✗		47.0
	0.4		✗		47.1
	0.6		✗		45.7
TPLD	0.2			✓	38.1
		✓			49.5
		$\checkmark_2$	$\checkmark_1$		49.1
		$\checkmark_1$	$\checkmark_2$		**51.2**

(a) Framework design choices

Voting Num \ Voting Field	37	**57**	77
1	48.61	48.95	48.57
3	49.49	**49.52**	48.37
5	48.72	48.00	48.63

(b) Voting field / number

α	mIoU
0.6	48.34
0.7	**49.52**
0.8	49.25

(c) α

q	mIoU
1.00	48.44
0.40	50.15
0.35	50.27
0.30	**51.20**
0.25	49.81
0.20	50.02

(d) q

γ	mIoU
1.5	50.0
2	**51.2**
2.5	48.6

(e) γ

Table 5. Detailed analysis on the proposed objective functions. We note the corresponding equations for each proposals. *Adv.* denotes adversarial loss term for hard sample training.

	Bootstrap Eq.(5)	Voting Eq.(4)	mIoU
$\mathcal{L}_{st}$ [40]			47.00
	✓		48.47
$\mathcal{L}_{st1}$	✓	✓	49.52

	EH Cls. Eq.(6) + Eq.(7)	Adv. Eq.(8)+Eq.(9)	mIoU
$\mathcal{L}_{st1}$			49.51
	✓		50.11
$\mathcal{L}_{st1} + \mathcal{L}_{st2}$	✓	✓	51.20

number 3. The hyperparameter α controls how much to maintain the initial prediction value, and we observe that 0.7 produces the best result (see Table 4c). We see that the results are in the same line with the residual learning [16]. Providing residual features (i.e., pooled neighboring confident prediction values) while securing the initial behavior (i.e., initial prediction values) is important. For the second phase, we have a total of two hyper-parameters; q and γ. The hyperparameter q controls the 'easy' portions in the target images. For example, if we increase the value, more images will be used as easy samples for the training. We observe that setting q to 0.3 provides the best result (see Table 4d). Note that if we set q to 1 (i.e., making all the target images to be trained with the full pseudo labels), we instead obtain degraded performance. This implies that a proper portion of easy and hard samples are need to be set, and both the full pseudo label training and hard-to-easy feature alignment are important. The hyperparameter γ is related to the calibration degree of the prediction values in generating full pseudo labels (see Eq. (6)). We obtain the best result when γ equals 2.

5.6 Loss Function Analysis

Finally, we explore the impact of loss functions in Table 5. We begin with the standard self-training loss, $\mathcal{L}_{st}$. Introducing the bootstrapping mechanism boosts the performance significantly, from 47.00 to 48.47 mIoU. This implies that explicitly handling noisy pseudo labels is crucial but lacking in the original formulation. Also, using *voting* to densify the sparse pseudo labels further pushes the performance from 48.47 to 49.52 mIoU. The densified pseudo labels

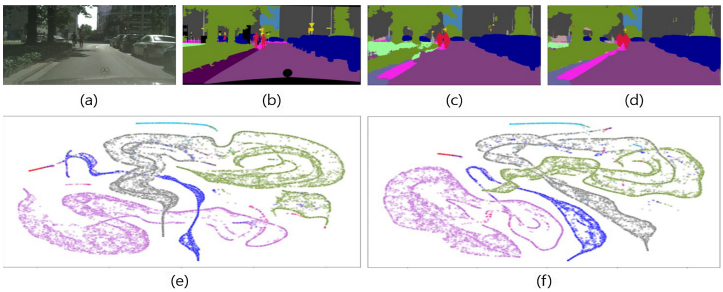

Fig. 6. A contrastive analysis of with and without hard sample training (Eq.(8)+Eq.(9)). (a): target image, (b): ground truth, (c): prediction result without hard sample training, (d): prediction result with hard sample training. We map high-dimensional features of (c) and (d) to 2-D space features of (e) and (f) respectively using t-SNE [26].

help model learning due to the increased training-signals and are complementary to the bootstrapping effect. In the second phase, we investigate the impact of both easy sample training (EH Cls.) and hard sample training (Adv.). The easy sample training pushes the performance from 49.52 to 50.11 mIoU, and the hard sample training further increases the performance from 50.11 to 51.20. The results demonstrate that the full-pseudo label training is indeed important and the hard-to-easy feature alignment further enhances the model learning. Especially for the hard sample training, we conduct a contrastive analysis in Fig. 6. We observe that hard sample training improves category-level feature alignment (Fig. 6 (e)→Fig. 6 (f)), and thus the prediction values become more accurate and clean (Fig. 6 (c) → Fig. 6 (d)).

6 Conclusions

In this paper, we point out that self-training methods for UDA suffer from the sparse pseudo label during training. Therefore, we present a novel two-phase pseudo label densification method. Combined with recently proposed CRST framework, we achieve new state-of-the-art results on UDA benchmarks.

Acknowledgement. This research is supported by the National Cancer Center(NCC).

References

1. Atapour-Abarghouei, A., Breckon, T.P.: Real-time monocular depth estimation using synthetic data with domain adaptation via image style transfer. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2800–2810 (2018)

2. Chen, L.C., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.: Deeplab: semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. IEEE Trans. Pattern Anal. Mach. Intell., June 2016. https://doi.org/10.1109/TPAMI.2017.2699184
3. Chen, L.C., Papandreou, G., Schroff, F., Adam, H.: Rethinking atrous convolution for semantic image segmentation, June 2017
4. Chen, Y., Li, W., Sakaridis, C., Dai, D., Van Gool, L.: Domain adaptive faster R-CNN for object detection in the wild. In: Proceedings of Computer Vision and Pattern Recognition (CVPR), pp. 3339–3348 (2018)
5. Chen, Y.C., Lin, Y.Y., Yang, M.H., Huang, J.B.: Crdoco: pixel-level domain transfer with cross-domain consistency. In: Proceedings of Computer Vision and Pattern Recognition (CVPR), June 2019
6. Cordts, M., et al.: The cityscapes dataset for semantic urban scene understanding. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
7. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Li, F.F.: Imagenet: a large-scale hierarchical image database, pp. 248–255, June 2009. https://doi.org/10.1109/CVPR.2009.5206848
8. Fernando, B., Habrard, A., Sebban, M., Tuytelaars, T.: Unsupervised visual domain adaptation using subspace alignment. In: Proceedings of International Conference on Computer Vision (ICCV), pp. 2960–2967 (2013)
9. Ganin, Y., Lempitsky, V.: Unsupervised domain adaptation by backpropagation. arXiv preprint arXiv:1409.7495 (2014)
10. Ghifary, M., Kleijn, W.B., Zhang, M., Balduzzi, D., Li, W.: Deep reconstruction-classification networks for unsupervised domain adaptation. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9908, pp. 597–613. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0_36
11. Golemo, F., Taiga, A.A., Courville, A., Oudeyer, P.Y.: Sim-to-real transfer with neural-augmented robot simulation. In: Billard, A., Dragan, A., Peters, J., Morimoto, J. (eds.) Proceedings of the 2nd Conference on Robot Learning. Proceedings of Machine Learning Research, vol. 87, pp. 817–828. PMLR, 29–31 October 2018. http://proceedings.mlr.press/v87/golemo18a.html
12. Gong, B., Shi, Y., Sha, F., Grauman, K.: Geodesic flow kernel for unsupervised domain adaptation. In: Proceedings of Computer Vision and Pattern Recognition (CVPR), pp. 2066–2073. IEEE (2012)
13. Gopalan, R., Li, R., Chellappa, R.: Domain adaptation for object recognition: an unsupervised approach. In: Proceedings of International Conference on Computer Vision (ICCV), pp. 999–1006. IEEE (2011)
14. Grandvalet, Y., Bengio, Y.: Semi-supervised learning by entropy minimization, pp. 529–536 (2005)
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778, June 2016. https://doi.org/10.1109/CVPR.2016.90
16. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
17. Hoffman, J., et al.: CyCADA: cycle-consistent adversarial domain adaptation. In: Proceedings of International Conference on Machine Learning (ICML), pp. 1989–1998 (2018)
18. Hoffman, J., Wang, D., Yu, F., Darrell, T.: FCNS in the wild: Pixel-level adversarial and constraint-based adaptation. arXiv preprint arXiv:1612.02649 (2016)

19. Hong, W., Wang, Z., Yang, M., Yuan, J.: Conditional generative adversarial network for structured domain adaptation. In: Proceedings of Computer Vision and Pattern Recognition (CVPR), June 2018
20. Kulis, B., Saenko, K., Darrell, T.: What you saw is not what you get: domain adaptation using asymmetric kernel transforms. In: Proceedings of Computer Vision and Pattern Recognition (CVPR), pp. 1785–1792. IEEE (2011)
21. Li, D., Yang, Y., Song, Y.Z., Hospedales, T.M.: Deeper, broader and artier domain generalization. In: Proceedings of International Conference on Computer Vision (ICCV), pp. 5542–5550 (2017)
22. Li, W., Xu, Z., Xu, D., Dai, D., Van Gool, L.: Domain generalization and adaptation using low rank exemplar SVMs. In: IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI), **40**, 1114–1127. IEEE (2017)
23. Long, M., Cao, Y., Cao, Z., Wang, J., Jordan, M.I.: Transferable representation learning with deep adaptation networks. IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI) **41**, 3071–3085 (2019). https://doi.org/10.1109/TPAMI.2018.2868685
24. Long, M., Cao, Y., Wang, J., Jordan, M.I.: Learning transferable features with deep adaptation networks. arXiv preprint arXiv:1502.02791 (2015)
25. Luo, Y., Zheng, L., Guan, T., Yu, J., Yang, Y.: Taking a closer look at domain shift: category-level adversaries for semantics consistent domain adaptation. In: Proceedings of Computer Vision and Pattern Recognition (CVPR) (2019)
26. van der Maaten, L., Hinton, G.: Visualizing data using t-sne. J. Mach. Learn. Res. **11**, 2579–2605 (2008)
27. Motiian, S., Piccirilli, M., Adjeroh, D.A., Doretto, G.: Unified deep supervised domain adaptation and generalization. In: Proceedings of International Conference on Computer Vision (ICCV), pp. 5715–5725 (2017)
28. Murez, Z., Kolouri, S., Kriegman, D., Ramamoorthi, R., Kim, K.: Image to image translation for domain adaptation. In: Proceedings of Computer Vision and Pattern Recognition (CVPR), pp. 4500–4509, June 2018. https://doi.org/10.1109/CVPR.2018.00473
29. Panareda Busto, P., Gall, J.: Open set domain adaptation. In: Proceedings of International Conference on Computer Vision (ICCV), pp. 754–763 (2017)
30. Reed, S., Lee, H., Anguelov, D., Szegedy, C., Erhan, D., Rabinovich, A.: Training deep neural networks on noisy labels with bootstrapping, December 2014
31. Richter, S.R., Vineet, V., Roth, S., Koltun, V.: Playing for data: ground truth from computer games. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9906, pp. 102–118. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46475-6_7
32. Ros, G., Sellart, L., Materzynska, J., Vazquez, D., Lopez, A.M.: The synthia dataset: a large collection of synthetic images for semantic segmentation of urban scenes. In: Proceedings of Computer Vision and Pattern Recognition (CVPR), pp. 3234–3243 (2016)
33. Sener, O., Song, H.O., Saxena, A., Savarese, S.: Learning transferrable representations for unsupervised domain adaptation. pp. 2110–2118 (2016)
34. Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W., Webb, R.: Learning from simulated and unsupervised images through adversarial training. In: Proc. of Computer Vision and Pattern Recognition (CVPR), pp. 2107–2116 (2017)
35. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv 1409.1556, September 2014

36. Tsai, Y.H., Hung, W.C., Schulter, S., Sohn, K., Yang, M.H., Chandraker, M.: Learning to adapt structured output space for semantic segmentation. In: Proceedings of Computer Vision and Pattern Recognition (CVPR), pp. 7472–7481 (2018)
37. Vu, T.H., Jain, H., Bucher, M., Cord, M., Pérez, P.: Advent: adversarial entropy minimization for domain adaptation in semantic segmentation. In: Proceedings of Computer Vision and Pattern Recognition (CVPR), pp. 2517–2526 (2019)
38. Zhu, X.: Semi-supervised learning tutorial. In: Proceedings of International Conference on Machine Learning (ICML) (2007)
39. Zou, Y., Yu, Z., Kumar, B.V., Wang, J.: Unsupervised domain adaptation for semantic segmentation via class-balanced self-training. In: Proceedings of European Conf. on Computer Vision (ECCV), pp. 289–305 (2018)
40. Zou, Y., Yu, Z., Liu, X., Kumar, B.V., Wang, J.: Confidence regularized self-training. In: Proceedings of International Conference on Computer Vision (ICCV), October 2019