Two-phase Pseudo Label Densification for Self-training based Domain Adaptation

Inkyu Shin Sanghyun Woo Fei Pan In So Kweon
KAIST, South Korea
{dlsrbgg33, shwoo93, feipan664, 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.*

1. Introduction

Unsupervised domain adaptation (UDA) aims to transfer knowledge learned from the label-rich source domain to an unlabeled new target domain. 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.

The dominant paradigm in UDA is based on ***adversarial learning*** [1, 4]. In particular, it minimizes both (source domain) task-specific loss and domain adversarial loss. While the adversarial learning has achieved great success in UDA, recently another line of studies using ***self-training*** emerged [5, 6]. 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) [5], and confidence regularized self-training (CRST) [6]. Unlike adversarial learning methods which utilize two separate losses, CBST presents a single unified self-training loss. 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.

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. 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.

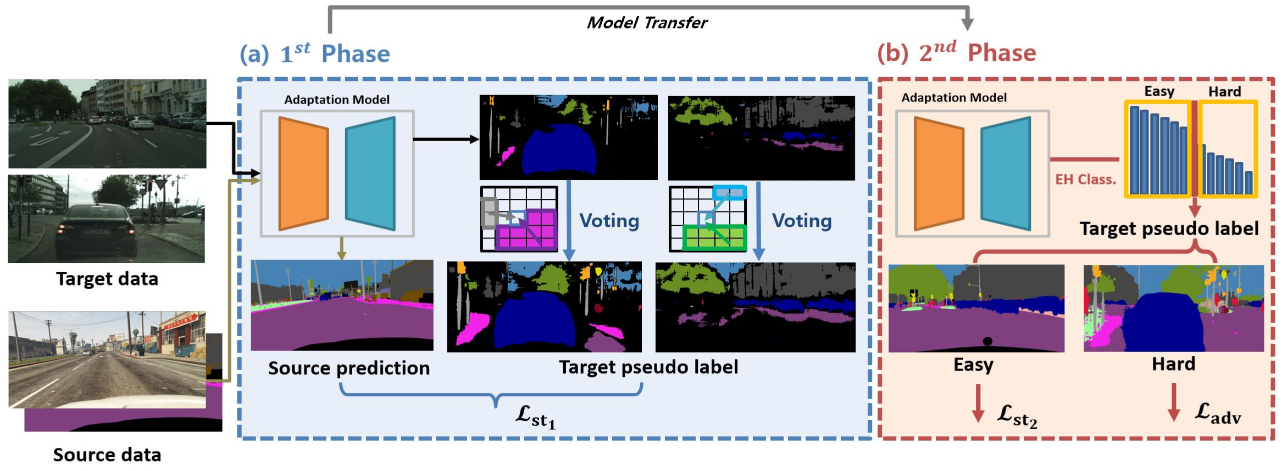


Figure 1: **The overview of the proposed two-phase pseudo-label densification framework.**

2. Method

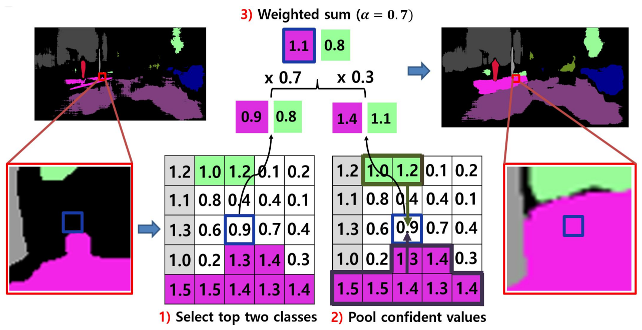
2.1. Problem Setting

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. With this self-training basic setting, we also borrow some terminology from CRST [6]; loss function $\mathcal{L}_{st}$, optimized pseudo-label $\hat{y}_t^{(k)*}$ and class-wise threshold value λ_k.

2.2. 1st phase: Voting based Densification

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 (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.

We call the above whole process voting-based densification. We abbreviate it as **Voting**. The pseudo label generation in the 1st phase can be summarized as:



1.2	1.0	1.2	0.1	0.2
1.1	0.8	0.4	0.4	0.1
1.3	0.6	0.9	0.7	0.4
1.0	0.2	1.3	1.4	0.3
1.5	1.5	1.4	1.3	1.4

Figure 2: **The overall procedure of the voting-based densification.**

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

Objective function for the 1st phase
To effectively train the model under the existence of noisy pseudo labels, we introduce bootstrapping [2] in our final objective function. The original self-training objective function can be re-formulated as the following:

$$\begin{aligned} \min_{\mathbf{w}, \hat{\mathbf{Y}}_\mathbf{T}} \mathcal{L}_{st_1}(\mathbf{w}, \hat{\mathbf{Y}}_\mathbf{T}) = - \sum_{s \in S} \sum_{k=1}^{K} y_s^{(k)} \log p(k|\mathbf{x_s}; \mathbf{w}) \\ - \sum_{t \in T} [\sum_{k=1}^{K} \{\beta \hat{y}_t^{(k)} \log + (1-\beta) \frac{p(k|\mathbf{x_t}; \mathbf{w})}{\lambda_k}\} \log \frac{p(k|\mathbf{x_t}; \mathbf{w})}{\lambda_k} \\ - \alpha r_c(\mathbf{w}, \hat{\mathbf{Y}}_\mathbf{T})] \\ s.t. \ \ \hat{y}^t \in \Delta^{K-1} \cup \{\mathbf{0}\}, \forall t \end{aligned} \tag{2}$$

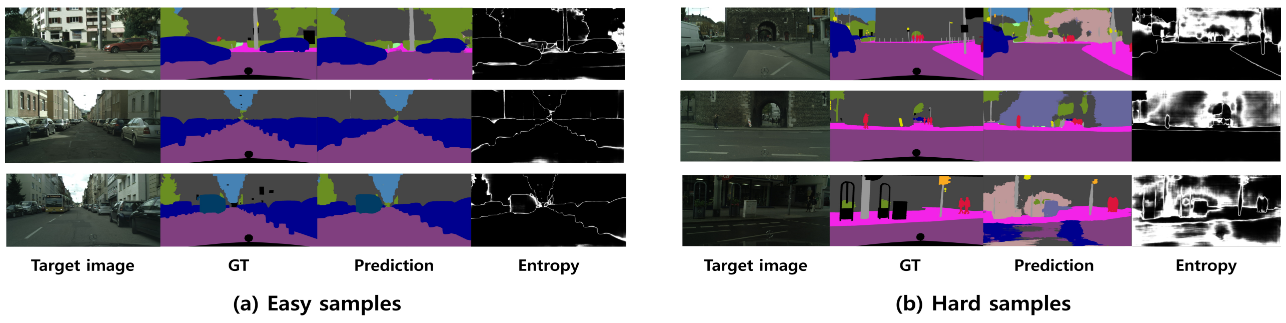


Figure 3: **Qualitative easy and hard samples.**

2.3. 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, 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. 3).

To reasonably categorize target samples into easy and hard, we present effective criteria. For a particular image **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 [5], 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.

2.3.1 Objective function for the 2nd phase

After classifying target images into easy and hard samples, we apply different objective functions to each. We describe the details below.

2.3.2 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:

$$\hat{y}_{t_e}^{(k)*} = \begin{cases} 1, & \text{if } k = \arg\max_k \{\frac{p(k|x_t;w)}{\lambda_k}\} \\ & \text{and } p(k|\boldsymbol{x_t};\boldsymbol{w}) > \lambda_k \\ \left(\frac{p(k|\boldsymbol{x_t};\boldsymbol{w})}{\lambda_k}\right)^{\gamma}, & \text{otherwise.} \end{cases} \tag{3}$$

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

$$\begin{aligned} &\min_{\mathbf{w},\hat{\mathbf{Y}}_{\mathbf{T}}} \mathcal{L}_{st_2}(\mathbf{w},\hat{\mathbf{Y}}_{\mathbf{T}}) \\ &= -\sum_{t\in T}[\sum_{k=1}^{K}\{\beta\hat{y}_t^{(k)}\log + (1-\beta)\frac{p(k|\mathbf{x_t};\mathbf{w})}{\lambda_k}\}\log\frac{p(k|\mathbf{x_t};\mathbf{w})}{\lambda_k} \\ &s.t.\ \ \hat{y}^t \in \Delta^{K-1}\cup\{\mathbf{0}\}, \forall t \end{aligned} \tag{4}$$

2.3.3 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 [4] is whether from easy samples or hard samples. The learning objective of the discriminator is:

$$\min_{\theta_{D_{intra}}} \frac{1}{|e|}\sum_{e} L_{D_{intra}}(I_e, 1) + \frac{1}{|h|}\sum_{h} L_{D_{intra}}(I_h, 0) \tag{5}$$

and the adversarial objective to train the segmentation network is:

$$\min_{\theta_{seg}} \frac{1}{|h|}\sum_{h} L_{D_{intra}}(I_h, 1) \tag{6}$$

Table 1: **Experimental results on GTA5 → Cityscapes.** ”V” and ”R” denote VGG-16 and ResNet-101 respectively. We highlight the rare classes [1] 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 [5]	Deeplabv2-V	**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) [6]	Deeplabv2-V	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	Deeplabv2-V	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 [3]	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 [1]	Deeplabv2-R	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 [4]	Deeplabv2-R	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 [5]	Deeplabv2-R	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) [6]	Deeplabv2-R	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	Deeplabv2-R	**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 [5]	Deeplabv3-R	**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) [6]	Deeplabv3-R	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	Deeplabv3-R	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: Performance improvements in mIoU of integrating our TPLD with existing self-training adaptation approaches. We use the Deeplabv2-R segmentation model.

(a) GTA5 → Cityscapes

GTA5 → Cityscapes (19 categories)			
Method	Base	+ TPLD	△
CBST [5]	45.9	47.8	+1.9
CRST(LRENT) [6]	45.9	47.3	+1.4
CRST(MRKLD) [6]	47.0	51.2	+4.2

(b) SYNTHIA → Cityscapes

SYNTHIA → Cityscapes (16 categories)			
Method	Base	+ TPLD	△
CBST [5]	42.6	45.6	+3.0
CRST(LRENT) [6]	42.7	47.0	+4.3
CRST(MRKLD) [6]	43.8	47.3	+3.5

3. Experiments

In this section, we conduct experiments to analyze our major proposals with same UDA benchmark setting of CRST [6]. Table 1 summarizes the adaptation performance of TPLD in GTA5 to Cityscpaes benchmark. We can obviously see that TPLD outperforms state-of-the-art approaches in all cases.

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 [5], CRST with label regularization (LRENT) [6], and CRST with model regularization (MRKLD) [6]. The results are summarized in Table. 2. We observe that TPLD consistently improves the performance of 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.

4. Conclusion

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. We hope many follow-up studies come up with our observations and results.

References

[1] Yawei Luo, Liang Zheng, Tao Guan, Junqing Yu, and Yi Yang. Taking a closer look at domain shift: Category-level adversaries for semantics consistent domain adaptation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 1, 4

[2] Scott Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. Training deep neural networks on noisy labels with bootstrapping. 12 2014. 2

[3] Yi-Hsuan Tsai, Wei-Chih Hung, Samuel Schulter, Kihyuk Sohn, Ming-Hsuan Yang, and Manmohan Chandraker. Learning to adapt structured output space for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7472–7481, 2018. 4

[4] Tuan-Hung Vu, Himalaya Jain, Maxime Bucher, Matthieu Cord, and Patrick Pérez. Advent: Adversarial entropy minimization for domain adaptation in semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2517–2526, 2019. 1, 3, 4

[5] Yang Zou, Zhiding Yu, BVK Vijaya Kumar, and Jinsong Wang. Unsupervised domain adaptation for semantic segmentation via class-balanced self-training. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 289–305, 2018. 1, 3, 4

[6] Yang Zou, Zhiding Yu, Xiaofeng Liu, B.V.K. Vijaya Kumar, and Jinsong Wang. Confidence regularized self-training. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019. 1, 2, 4