Transferable Semantic Augmentation for Domain Adaptation Supplementary Materials

Shuang Li[1] Mixue Xie[1] Kaixiong Gong[1] Chi Harold Liu[1*] Yulin Wang[2] Wei Li[3]
[1]Beijing Institute of Technology [2]Tsinghua University [3]Inceptio Tech.
shuangli@bit.edu.cn michellexie102@gmail.com kxgong@bit.edu.cn liuchi02@gmail.com
wang-yl19@mails.tsinghua.edu.cn liweimcc@gmail.com

This supplementary materials provide the training process of Transferable Semantic Augmentation (TSA), the analysis of memory cost and performance gain, the algorithm for reverse mapping, extra visualization of the augmentation, the necessity stress test of target data and the visualization of learned features.

1. Training of TSA

In summary, the proposed Transferable Semantic Augmentation (TSA) can be simply implemented and optimized in an end-to-end deep learning framework through the overall objective function:

$$\begin{aligned}\mathcal{L}_{TSA} &= \mathcal{L}_{\infty} + \beta \mathcal{L}_{MI} \\ &= -\frac{1}{n_s}\sum_{i=1}^{n_s}\log\frac{e^{Z_{si}^{y_{si}}}}{\sum_{c=1}^{C} e^{Z_{si}^{c}}} \\ &+ \beta\Big(\sum_{c=1}^{C}\hat{P}^c\log\hat{P}^c - \frac{1}{n_t}\sum_{j=1}^{n_t}\sum_{c=1}^{C}P_{tj}^c\log P_{tj}^c\Big), \end{aligned} \tag{1}$$

where $\mathcal{L}_{\infty}$ is our proposed transferable loss and $\mathcal{L}_{MI}$ is the mutual information maximization loss.

The training process of TSA for domain adaptation is presented in Algorithm 1.

2. Memory Cost and Performance Gain

In Table 1, we analyze the GPU memory cost and performance gain of our memory module $\mathbb{M}$. TSA (w/ iterative) denotes that we employ the iterative manner in [7] to estimate the mean and covariance, while TSA (w/ memory) denotes that we estimate the mean and covariance according to our memory module $\mathbb{M}$. In our experimental settings, the feature dimensions are set as 256 for Office-31. Compared to ResNet-50, TSA (w/ memory) obtains a large improvement of 13.2% with negligible extra 0.3GB GPU memory cost, surpassing TSA (w/ iterative) by 2.8%. This is mainly

*Corresponding author.

Algorithm 1 Transferable Semantic Augmentation (TSA)

Input: Labeled source domain $\mathcal{S} = \{(\boldsymbol{x}_{si}, y_{si})\}_{i=1}^{n_s}$, unlabeled target domain $\mathcal{T} = \{\boldsymbol{x}_{tj}\}_{j=1}^{n_t}$; maximum iteration T and batch size B; hyper-parameters: λ_0 and β.
Output: Parameters of the final model: $\boldsymbol{\Theta}_F$, $\mathbf{W}$ and $\boldsymbol{b}$.

Initialize model parameters $\boldsymbol{\Theta}_F$, $\mathbf{W}$ and $\boldsymbol{b}$; and initialize memory module $\mathbb{M}$ with $\mathcal{S}$ and $\mathcal{T}$.
for $t = 1$ to T **do**
  $\lambda = (t/T) \times \lambda_0$ // λ controls augmentation strength
  Sample $\{(\boldsymbol{x}_{si}, y_{si})\}_{i=1}^{B}$ and $\{(\boldsymbol{x}_{tj})\}_{j=1}^{B}$ from $\mathcal{S}$ and $\mathcal{T}$, respectively.
  Obtain deep features $\{\mathbf{f}_{si}\}_{i=1}^{B}, \{\mathbf{f}_{tj}\}_{j=1}^{B}$ and logit outputs $\{\hat{\boldsymbol{y}}_{si}\}_{i=1}^{B}, \{\hat{\boldsymbol{y}}_{tj}\}_{j=1}^{B}$ for source and target samples, respectively.
  Compute probabilistic outputs of target samples: $\{\boldsymbol{P}_{tj} = softmax(\hat{\boldsymbol{y}}_{tj})\}_{j=1}^{B}$ and generate target pseudo labels: $\{y'_{tj} = \arg\max_c P_{tj}^c\}_{j=1}^{B}$.
  Update memory module $\mathbb{M}$ with features and pseudo labels in current batch t.
  for each class c **do**
    Estimate features means $\boldsymbol{\mu}_s^c$ and $\boldsymbol{\mu}_t^c$ according to memory module $\mathbb{M}$.
    Estimate inter-domain mean difference $\boldsymbol{\Delta\mu}^c = \boldsymbol{\mu}_t^c - \boldsymbol{\mu}_s^c$.
    Estimate the target intra-class covariance matrix $\boldsymbol{\Sigma}_t^c$ according to memory module $\mathbb{M}$.
  end for
  Update $\boldsymbol{\Theta}_F$, $\mathbf{W}$ and $\boldsymbol{b}$ by minimizing the loss $\mathcal{L}_{TSA}$ in Eq (1) with stochastic gradient descent (SGD).
end for

Table 1. GPU memory cost on Office-31 with batch-size 32.

Method	GPU Memory (GB)	Accuracy (%)	Gain (%)
ResNet-50	7.2	76.1	-
TSA (w/ iterative)	7.3	86.5	10.4 ↑
TSA (w/ memory)	7.5	89.3	13.2 ↑

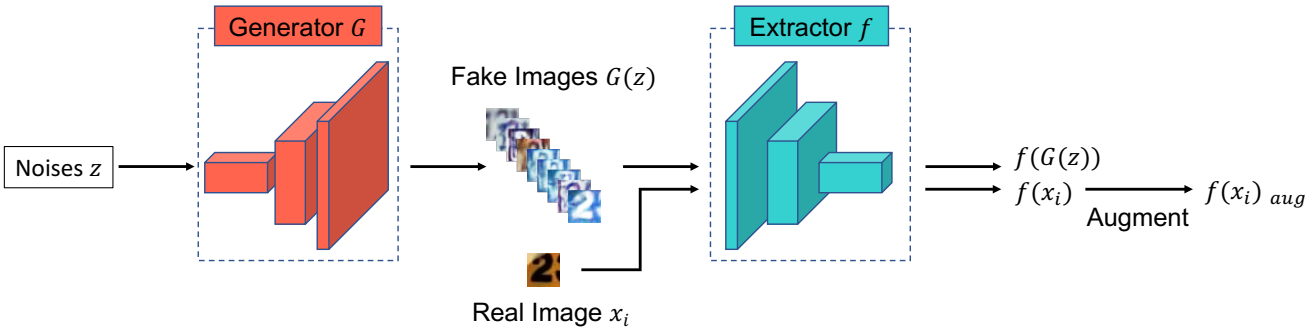

Figure 1. Outline of the reverse mapping algorithm. Generator G is to generate fake images visually similar to source domain, while extractor F is to extract the features of fake and source images. We use the two modules to search images in the pixel space corresponding to the augmented features in the deep feature space.

due to the fact that the iterative manner will bias the estimation of expected mean and covariance, due to its accumulation property. Besides, it is noteworthy that the memory module is not required in the inference phase.

Iterative Estimation Manner. Here, we elaborate the iterative manner of updating mean and covariance. To estimate the covariance matrix of features, ISDA [7] proposes an iterative manner by integrating covariances of batches from first to current batch. For class c, the iterative estimation manner is formulated as:

$$\eta_c^i = \frac{B_c^i}{N_c^{(i-1)} + B_c^i}, \tag{2}$$

$$\boldsymbol{\mu}_c^{(i)} = (1-\eta_c^i)\boldsymbol{\mu}_c^{(i-1)} + \eta_c^i \boldsymbol{\mu'}_c^i, \tag{3}$$

$$\begin{aligned}\boldsymbol{\Sigma}_c^{(i)} &= (1-\eta_c^i)\boldsymbol{\Sigma}_c^{(i-1)} + \eta_c^i \boldsymbol{\Sigma'}_c^i \\ &+ \eta_c^i(1-\eta_c^i)(\boldsymbol{\mu}_c^{(i-1)} - \boldsymbol{\mu'}_c^i)(\boldsymbol{\mu}_c^{(i-1)} - \boldsymbol{\mu'}_c^i)^\top,\end{aligned} \tag{4}$$

where $N_c^{(i-1)}$ is the total number of training samples for class c in all previous $(i-1)$ batches, and B_c^i is the number of training samples belonging to class c in current batch i. η_c^i is the proportion of samples of class c in current batch i to samples of class c in all previous $(i-1)$ batches. $\boldsymbol{\mu}_c^{(i-1)}$ and $\boldsymbol{\mu'}_c^i$ are the averages of features for class c in all previous $(i-1)$ batches and current batch i, respectively. $\boldsymbol{\Sigma}_c^{(i-1)}$ and $\boldsymbol{\Sigma'}_c^i$ are the covariances of features for class c in all previous $(i-1)$ batches and current batch i, respectively.

However, the weight distribution of network in early training stage will vastly differ from that in latter training stage. Thus, due to its accumulation property, such iterative manner might cause out-of-date features to bias the estimation of expected covariance matrix. By contrast, our memory module will discard the out-of-date batch and replace with the latest batch, conductive to more accurate estimations of mean and covariance.

3. Reverse Mapping Algorithm

To intuitively display how TSA generates the meaningful semantic transformations of source images, inspired by [7], we design a reverse mapping algorithm to generate images in the pixel-level space corresponding to the augmented features in the deep feature space.

The outline of the reverse mapping algorithm is shown in Fig. 1. There are two modules involved in this algorithm: 1) GANs-based generator G and 2) CNNs-based feature extractor F. Here, the generator G is pre-trained on a source dataset to generate fake images that are close to source domain, while feature extractor F is pre-trained on an adaptation task to extract the features of fake and source images. Taking the adaptation task SVHN→MNIST as an example, G is pre-trained on dataset SVHN, and F is pre-trained on task SVHN→MNIST. After the two modules are well trained separately, we freeze them during following phases.

Formally, let $\boldsymbol{z} \in \mathbb{R}^d$ denote the random noise variable and $\boldsymbol{x}_s$ denote the source image randomly sampled from a source domain, such as SVHN. At first, noise $\boldsymbol{z}$ is fed into generator G to synthesize fake image $G(\boldsymbol{z})$. Then, fake image $G(\boldsymbol{z})$ and source image $\boldsymbol{x}_s$ are input to the feature extractor F to obtain deep features $F(G(\boldsymbol{z}))$ and $F(\boldsymbol{x}_s)$, respectively. For achieving reliable visualization results, we first find the noise $\boldsymbol{z}'$ corresponding to source image $\boldsymbol{x}_s$ by solving the following optimization problem:

$$\boldsymbol{z}' = \arg\min_{\boldsymbol{z}} \|F(G(\boldsymbol{z})) - F(\boldsymbol{x}_s)\|_2^2 + \alpha\|G(\boldsymbol{z}) - \boldsymbol{x}_s\|_2^2, \tag{5}$$

where α is a trade-off parameter to balance the contributions of two parts. We use the mean square error to constrain noise $\boldsymbol{z}$ in both feature and pixel levels to obtain a reliable initial point $\boldsymbol{z}'$ for the following procedure.

After yielding the initial point $\boldsymbol{z}'$, we augment the original source feature $F(\boldsymbol{x}_s)$ with TSA, forming the augmented feature $F_{aug}(\boldsymbol{x}_s)$. Then we initialize $\boldsymbol{z}$ with $\boldsymbol{z}'$ to continue to search the optimal noise $\boldsymbol{z}^*$ corresponding to the aug-

Figure 2. Extra visualizations of semantically augmented images for task SVHN→MNIST. "Source" and "Augmented Source" columns present the images from SVHN and their corresponding augmented images synthesized by TSA, respectively. "Target" column provides several images representing corresponding classes from MNIST.

mented feature $F_{aug}(\boldsymbol{x}_s)$:

$$\boldsymbol{z}^* = \arg\min_{\boldsymbol{z}} \|F(G(\boldsymbol{z})) - F_{aug}(\boldsymbol{x}_s)\|_2^2. \tag{6}$$

When Eq (6) is optimized, the desired $\boldsymbol{z}^*$ can be obtained. Consequently, the image $G(\boldsymbol{z}^*)$ generated by G should be the appropriate visualization for the corresponding augmented source feature.

Implemented Details. We adopt the structure of WGAN-GP [2] for the generator G. The architecture of feature extractor F is based on the chosen task, e.g., we use the feature extractor of task SVHN→ MNIST as [6, 3]. We also adopt SGD optimizer with momentum 0.9 to optimize the reverse mapping algorithm. The dimension d of noise variable is set to be 128.

Extra Visualization Results of Augmentation. We adopt the aforementioned reverse mapping algorithm to produce more visualization results for augmented source features, which are shown in Fig. 2. The results further prove that TSA is indeed able to generate diverse, meaningful and even target-specific augmented features, which will facilitate adapting the classifier from source to target domain successfully.

4. Necessity Stress Test of Target Data

For reducing the training burden in practice, we test how much target data is necessary to achieve desirable performance of TSA by varying the amount of target data participated in the training. Specifically, for each class, we randomly sample a proportion (ρ) of target data as training samples along with the original source dataset, and use the entire target dataset for evaluating. The results of ResNet-50+TSA on Office-31 under $\rho \in \{0\%, 20\%, 40\%, 60\%, 80\%, 100\%\}$ are shown in Table 2, where we can see that when target data increases, higher accuracies are obtained. This is because more accurate and comprehensive semantics can be captured with more target data. Besides, the performances of $\rho = 80\%$ and $\rho = 100\%$ are comparable, which motivates us to reduce target data in training to save computation and memory cost for large-scale target datasets.

Table 2. Necessity stress test of target data on Office-31.

Method	ρ	A→W	D→W	W→D	A→D	D→A	W→A	Avg
ResNet-50 + TSA	0%	68.4	96.7	99.3	68.9	62.5	60.7	76.1
	20%	82.4	97.4	99.5	81.5	66.4	64.8	82.0
	40%	90.2	98.4	99.8	88.5	70.5	69.7	86.2
	60%	93.2	98.8	**100.0**	91.7	73.4	73.0	88.4
	80%	94.3	98.8	**100.0**	92.4	74.8	74.2	89.1
	100%	**94.8**	**99.1**	**100.0**	**92.6**	**74.9**	**74.4**	**89.3**

5. Visualization of Features

Though TSA strives to adapt classifiers from source domain to target domain, we surprisingly notice that TSA can

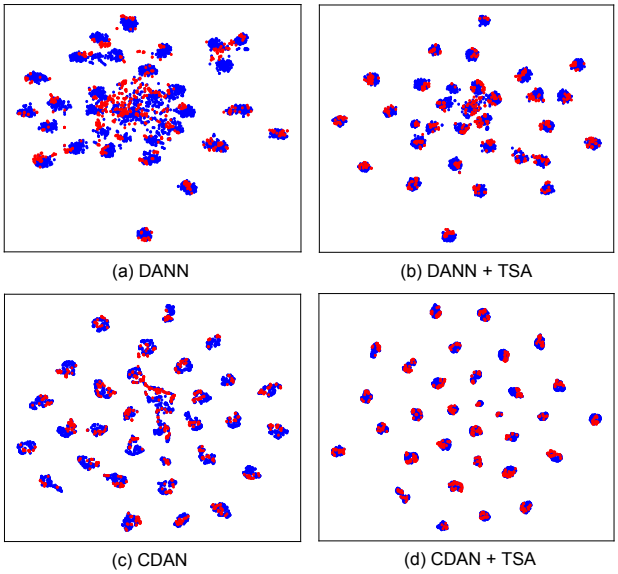

Figure 3. Visualization of the features learned by DANN, DANN+TSA, CDAN and CDAN+TSA on task A→W (Office-31). Blue and red dots stand for the source features and the target features, respectively.

also empower DA methods to learn more transferable feature representations. We adopt t-SNE tool [5] to visualize the features learned by DANN [1], DANN+TSA, CDAN [4], and CDAN+TSA on task A→W (Office-31) in Fig. 3. From results we observe that DANN and CDAN cannot align both domains perfectly, due to the mismatching of several classes. After applying TSA, the representations are more indistinguishable between two domains and the class boundaries are sharper, validating that TSA facilitates learning more transferable and discriminative features besides the effective classifier adaptation.

References

[1] Yaroslav Ganin and Victor S. Lempitsky. Unsupervised domain adaptation by backpropagation. In *ICML*, pages 1180–1189, 2015. 4

[2] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. In *NeurIPS*, pages 5769–5779, 2017. 3

[3] Shuang Li, Chi Harold Liu, Binhui Xie, Limin Su, Zhengming Ding, and Gao Huang. Joint adversarial domain adaptation. In *ACM MM*, pages 729–737. ACM, 2019. 3

[4] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. Conditional adversarial domain adaptation. In *NeurIPS*, pages 1647–1657, 2018. 4

[5] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *JMLR*, 9(Nov):2579–2605, 2008. 4

[6] Kuniaki Saito, Kohei Watanabe, Yoshitaka Ushiku, and Tatsuya Harada. Maximum classifier discrepancy for unsupervised domain adaptation. In *CVPR*, pages 3723–3732, 2018. 3

[7] Yulin Wang, Xuran Pan, Shiji Song, Honghua Zhang, Cheng Wu, and Gao Huang. Implicit semantic data augmentation for deep networks. In *NeurIPS*, pages 12614–12623, 2019. 1, 2