

Supplementary: Towards Inheritable Models for Open-Set Domain Adaptation

This supplementary is organized as follows:

- Sec. 1: Derivation of Result 1
- Sec. 2: Algorithm and implementation details
 - Architecture (Sec. 2.1, Table 1)
 - Inheritable model training (Sec. 2.2, Algo. 1)
 - Feature-splicing algorithm (Algo. 2)
 - Target domain adaptation (Sec. 2.3, Algo. 3)
- Sec. 3: Alternate methods to generate OOD samples
- Sec. 4: Miscellaneous
 - Training time comparison (Sec. 4.1)
 - Dataset description (Sec. 4.2)

1. Derivation of Result 1

We derive the result using Theorem 3 in [1], which provides a generalized learning bound for a given pair of source and target domains. We begin by quoting the theorem, following which we apply the theorem to our setting. For quoting the theorem, we use the notations given in [1].

Theorem 3 [1]. *Let \mathcal{H} be a hypothesis space of VC dimension d . Let \mathcal{U}_S and \mathcal{U}_T be unlabeled samples of size m' each, drawn from \mathcal{D}_S and \mathcal{D}_T respectively. Let S be a labeled sample of size m generated by drawing βm points from \mathcal{D}_T and $(1 - \beta)m$ points from \mathcal{D}_S and labeling them according to f_S and f_T , respectively. If $\hat{h} \in \mathcal{H}$ is the empirical minimizer of $\hat{\epsilon}_\alpha(h)$ on S and $h_T^* = \min_{h \in \mathcal{H}} \epsilon_T(h)$ is the target error minimizer, then for any $\delta \in (0, 1)$, with probability at least $1 - \delta$ (over the choice of samples),*

$$\epsilon_T(\hat{h}) \leq \epsilon_T(h_T^*) + A + 2(1 - \alpha)B \quad (1)$$

where,

$$A = 4\sqrt{\frac{\alpha^2}{\beta} + \frac{(1 - \alpha)^2}{1 - \beta}} \sqrt{\frac{2d \log(2(m + 1)) + 2 \log(8/\delta)}{m}} \quad (2)$$

$$B = \frac{1}{2} \hat{d}_{\mathcal{H} \Delta \mathcal{H}}(\mathcal{U}_S, \mathcal{U}_T) + 4\sqrt{\frac{2d \log(2m') + \log(8/\delta)}{m'}} + \lambda \quad (3)$$

and the error $\hat{\epsilon}_\alpha(h) = \alpha \hat{\epsilon}_T(h) + (1 - \alpha) \hat{\epsilon}_S(h)$, with α being the relative importance given to the empirical target error. Here, f_S and f_T are the ground-truth labeling functions for the source and the target domains. Further note that, the notations used to describe the errors ϵ_T and $\hat{\epsilon}_\alpha$ are different from the notations we use in our paper (for instance, we refer to the target error as ξ_q). See [1] for more details.

In our formulation, we make two observations. Firstly, the objective of our adaptation step is to improve the performance on the target domain. This entails a choice of $\alpha = 1$ for defining the empirical error. Secondly, we do not have any data from the source domain, which implies $\beta = 1$.

Now, with $\alpha = 1$, the last term in Eq. 1 vanishes. Further, to evaluate Eq. 2 with $\alpha = 1, \beta = 1$ which obtains an indeterminate form, we take the limit as $\alpha \rightarrow 1, \beta \rightarrow 1$,

$$\lim_{\alpha \rightarrow 1, \beta \rightarrow 1} A = 4\sqrt{\frac{2d \log(2(m + 1)) + 2 \log(8/\delta)}{m}} \quad (4)$$

This reduces Eq. 1 to,

$$\epsilon_T(\hat{h}) \leq \epsilon_T(h_T^*) + 4\sqrt{\frac{2d \log(2(m + 1)) + 2 \log(8/\delta)}{m}} \quad (5)$$

Now, we describe the result. We argue that the knowledge of class-separability (*i.e.* the knowledge of how the classes are distinguished) is *inheritable*, by demonstrating that the *inheritability* criterion holds for shared classes.

During adaptation, we select the top- k percentile target instances based on the value of w . In Sec. 5.3c of the paper, we empirically verify that the pseudo-labeling precision for the top- k target instances is close to 1 (see Fig. 5B of the paper for epoch-0 at $k = 15$). Therefore, the pseudo-labeling process can be considered as obtaining target-shared instances with a small noise in the labels. For these instances, by minimizing $\mathcal{L}_{in,h}$, we search the hypothesis space \mathcal{H} for the empirical minimizer of $\xi_{q^{sh}}$, *i.e.* $\hat{h}_t = \operatorname{argmin}_{h \in \mathcal{H}} \hat{\xi}_{q^{sh}}(h)$. Thus, considering the correctly pseudo-labeled target instances as the sample S of size m in Eq. 5, \hat{h}_t as the empirical minimizer of $\xi_{q^{sh}}$, and h_t^* as the optimal hypothesis for q^{sh} we can obtain the relation in Eq. 5 for the target-shared distribution (q^{sh}) as,

Table 1. Architecture of vendor and client models. ‘FC(Inp, Out)’ denotes Fully-Connected Layers with ‘Inp’ input nodes and ‘Out’ output nodes. BN denotes BatchNorm layer

Component	Layers
E_s, E_t	FC(2048×1024) → ELU →
	FC(1024×1024) → BN → ELU →
	FC(1024×256) → ELU →
	FC(256×256) → BN → ELU
G_s	FC(256× \mathcal{C}_s)
G_n	FC(256× K)

$$\xi_{q^{sh}}(\hat{h}_t) \leq \xi_{q^{sh}}(h_t^*) + 4\sqrt{\frac{2d \log(2(m+1)) + 2 \log(8/\delta)}{m}} \quad (6)$$

Hence, we obtain Result 1 of the paper, which is stated as,

Result 1. Let \mathcal{H} be a hypothesis class of VC dimension d . Let \mathcal{S} be a labeled sample set of m points drawn from q^{sh} . If $\hat{h}_t \in \mathcal{H}$ be the empirical minimizer of $\xi_{q^{sh}}$ on \mathcal{S} , and $h_t^* = \operatorname{argmin}_{h \in \mathcal{H}} \xi_{q^{sh}}(h)$ be the optimal hypothesis for q^{sh} , then for any $\delta \in (0, 1)$, we have with probability of at least $1 - \delta$ (over the choice of samples),

$$\xi_{q^{sh}}(\hat{h}_t) \leq \xi_{q^{sh}}(h_t^*) + 4\sqrt{\frac{2d \log(2(m+1)) + 2 \log(8/\delta)}{m}} \quad (7)$$

the term ϵ in Eq. 1 of the paper

Note that, as we enforce the loss \mathcal{L}_{inh} while learning the target model h_t during adaptation, we prune the hypothesis space while searching for all such hypotheses which satisfy the above condition. This is the *inheritability* criterion obtained using target-shared instances, since we achieve the condition using unlabeled target instances \mathcal{D}_t and the source model h_s (which pseudo-labels the target instances). In this manner, the *inheritability* criterion is satisfied for the target-shared instances, making the knowledge of class-separability *inheritable* for the adaptation task.

Furthermore, \mathcal{L}_{tune} can be seen as a way of self-supervising the target model. We show in Sec. 5c of the paper that the precision of the target model iteratively increases as a result of adaptation (Fig. 5B of the paper, see ‘‘epoch-0’’ through ‘‘converged’’). Therefore, the self-supervision yields an increasing number of correctly labeled target-shared instances over iterations. This effectively tightens the bound in Eq. 7 (increasing m in Eq. 7 reduces the last term), resulting in a superior adaptation guarantee.

2. Algorithm and implementation details

In this section we provide the pseudo-code for the model training and the feature-splicing algorithm and present the implementation details.

Algorithm 1 Pseudo-Code for inheritable model training

1: **require:** labeled source dataset \mathcal{D}_s , parameters $\theta_{M_s}, \theta_{E_s}, \theta_{G_s}, \theta_{G_n}$ of M_s, E_s, G_s, G_n respectively, hyperparameters K, d , and no. of *negative* instances η_u .

Step 1: Pre-training on the source dataset

2: **while** \mathcal{L}_b has not converged **do**
3: $(X_s, Y_s) \leftarrow$ batch sampled from \mathcal{D}_s
4: $\hat{Y}_s \leftarrow \sigma(G_s(F_s(X_s)))$
5: compute mean \mathcal{L}_b for the batch using \hat{Y}_s and Y_s
6: **update** $\theta_{M_s}, \theta_{E_s}, \theta_{G_s}$ by minimizing \mathcal{L}_b using the Adam optimizer
7: **end while**

Step 2: Training the inheritable model

8: $\mathcal{D}_n \leftarrow$ FeatureSplicingAlgorithm($\mathcal{D}_s, M_s, K, d, \eta_u$)
9: **while** \mathcal{L}_s has not converged **do**
10: $(X_s, Y_s) \leftarrow$ batch sampled from \mathcal{D}_s
11: $(U_n, Y_n) \leftarrow$ batch sampled from \mathcal{D}_n
12: $\hat{Y}_s \leftarrow \sigma(G_s(F_s(X_s)))$
13: $\hat{Y}_n \leftarrow \sigma(G_n(E_s(U_n)))$
14: compute mean \mathcal{L}_s for the batch using $\hat{Y}_s, \hat{Y}_n, Y_s, Y_n$
15: **update** $\theta_{E_s}, \theta_{G_s}, \theta_{G_n}$ by minimizing \mathcal{L}_s using the Adam optimizer
16: **end while**

2.1. Architecture

For experiments on **Office-31** and **Office-Home**, we choose ResNet-50 upto the last AvgPool layer (2048 dimensions) as the backbone network for M_s . For experiments on **VisDA** dataset, we use the backbone as VGG-16 upto the last pooling layer (of shape $7 \times 7 \times 512$), followed by a global AvgPool along each channel to obtain an output of 512 dimensions. Modules E_s, E_t, G_s, G_n are composed of fully connected layers, batch norm layers and non-linearity (ELU) as shown in Table 1.

2.2. Vendor trains an inheritable model

The *vendor* has access to an annotated source dataset \mathcal{D}_s using which the *vendor* trains an *inheritable* model h_s . See Algorithm 1 for the pseudo-code. During training, we have the following image augmentations: random rotations, flip, color jitter and random crop. We pre-train the parameters of the components $\{M_s, E_s, G_s\}$ on the source domain by minimizing \mathcal{L}_b (L2-L7 in Algo. 1). We then freeze the backbone M_s and perform feature splicing (L8 in Algo. 1). See Algo. 2 for the pseudo-code for the feature splicing operation. We apply feature-splicing at the last layer of M_s to obtain *negative* instances u_n . Specifically, L8-L10 in Algo. 2 shows the feature splicing operation (ϕ_d), where the top- d percentile activations are replaced. In this manner, we

Algorithm 2 FeatureSplicingAlgorithm

1: **require:** labeled source dataset \mathcal{D}_s , parameters θ_{M_s} of M_s , hyperparameter K, d , no. of *negative* instances η_u

▷ Let $|\mathcal{D}|$ denote the cardinality of a set \mathcal{D} , $[\cdot]$ denote the indexing operation, $\cdot || \cdot$ denote the append operation and \mathcal{M} denote the output dimensionality of M_s (i.e. $\mathcal{M} = 2048$ for ResNet-50 and $\mathcal{M} = 512$ for VGG-16)

Step 1: Generating negative instances

2: Number of dimensions to splice $\eta_d = \mathcal{M} \times d/100$
3: $U_n \leftarrow \{\}$ ▷ Empty list
4: **while** $|U_n| \leq \eta_u$ **do**
5: $(x_s^{c_i}, x_s^{c_j}) \leftarrow$ sample 2 instances from \mathcal{D}_s belonging to different classes i.e. $c_i, c_j \in \mathcal{C}_s$ where $i \neq j$
6: $u_s^{c_i} \leftarrow M_s(x_s^{c_i})$
7: $u_s^{c_j} \leftarrow M_s(x_s^{c_j})$
8: $idx \leftarrow$ top η_d entries in $\text{argsort}(u_s^{c_i})$
9: $u_n \leftarrow u_s^{c_i}$
10: $u_n[idx] \leftarrow u_s^{c_j}[idx]$
11: $U_n \leftarrow U_n || u_n$
12: **end while**

Step 2: Assigning labels to negative instances

13: Perform K -means clustering on U_n and assign a unique *negative* class to each cluster
14: $\mathcal{D}_n \leftarrow \{(u_n, y_n) : u_n \in U_n, y_n = \text{negative class label of } u_n \text{ obtained from the previous step}\}$
15: **return** \mathcal{D}_n

generate $\eta_u = 20000$ *negative* instances for **Office-31** and $\eta_u = 50000$ for **Office-Home** and **VisDA**. We label these instances by performing a K -means clustering on the obtained features (L13-L14 in Algo. 2). Using \mathcal{D}_s and \mathcal{D}_n we train $\{E_s, G_s, G_n\}$ by minimizing \mathcal{L}_s (L9-L16 in Algo. 1).

2.3. Client adapts to the target domain

The *client* has access to unlabeled target dataset \mathcal{D}_t and vendor’s *inheritable* model $h_s = \{F_s, G\}$, using which the *client* performs adaptation to the target domain. See Algo. 3 for the pseudo-code. We use image augmentations as mentioned in Sec. 2.2. We obtain pseudo-labeled target instances (top- k percentile, based on the instance-level *inheritability* value w) into a collection \mathcal{D}_t^p (L2-L15). Thereafter, during adaptation, each batch contains pseudo-labeled target instances (L18) and unlabeled target instances (L21). We normalize the weights obtained for the unlabeled target instances (as mentioned in Sec. 5.1b of the paper), with the maximum weight in a batch (L28-L30). We then train F_t to adapt to the target domain by minimizing $\mathcal{L}_{inh} + \mathcal{L}_{tune}$.

Algorithm 3 Pseudo-Code for target domain adaptation

1: **require:** unlabeled target dataset \mathcal{D}_t , parameters $\theta_{F_s}, \theta_{F_t}, \theta_{G_s}, \theta_{G_n}$ of $\{F_s, F_t, G_s, G_n\}$, hyperparameter k .

▷ Let $[\cdot]$ denote the indexing operation, and, $\cdot || \cdot$ denote the append operation

Step 1: Pseudo-labeling process

2: Number of instances to pseudo-label $\eta_p = |\mathcal{D}_t| \times k/100$
3: $W \leftarrow \{\}$ ▷ Empty list
4: $Y_p \leftarrow \{\}$ ▷ Empty list
5: $X_p \leftarrow \{\}$ ▷ Empty list
6: **for** x_t in \mathcal{D}_t **do**
7: $\hat{y}_p \leftarrow \sigma(G(F_s(x_t)))$
8: $y_p \leftarrow \text{argmax}_{c_i \in \mathcal{C}_s} \hat{y}_p[c_i]$
9: $w \leftarrow \max_{c_i \in \mathcal{C}_s} \hat{y}_p[c_i]$
10: $W \leftarrow W || w$
11: $Y_p \leftarrow Y_p || y_p$
12: $X_p \leftarrow X_p || x_t$
13: **end for**
14: $idx \leftarrow$ top η_p entries in $\text{argsort}(W)$
15: $\mathcal{D}_t^p \leftarrow (X_p[idx], Y_p[idx])$

Step 2: Adaptation process

16: Initialize θ_{F_t} from θ_{F_s}
17: **while** $\mathcal{L}_{inh} + \mathcal{L}_{tune}$ has not converged **do**
18: $(X_p, Y_p) \leftarrow$ batch sampled from \mathcal{D}_t^p .
19: $\hat{Y}_p \leftarrow \sigma(G(F_t(X_p)))$
20: compute mean \mathcal{L}_{inh} for the batch using \hat{Y}_p, Y_p
21: $X_t \leftarrow$ batch sampled from \mathcal{D}_t
22: **for** each x_t in X_t **do**
23: $\hat{s} \leftarrow \sum_{c_i \in \mathcal{C}_s} \sigma(G(F_t(x_t)))[c_i]$
24: $w \leftarrow \max_{c_i \in \mathcal{C}_s} \sigma(G(F_s(x_t)))[c_i]$
25: $z_t^{sh} \leftarrow \sigma(G_s(F_t(x_t)))$
26: $z_t^{uk} \leftarrow \sigma(G_n(F_t(x_t)))$
27: **end for**
28: **for** each x_t in X_t **do** ▷ Normalize weights
29: $w'(x_t) \leftarrow w(x_t)/(\max_{x_t \in X_t} w(x_t))$
30: **end for**
31: compute mean \mathcal{L}_{tune} using $w', \hat{s}, z_t^{sh}, z_t^{uk}$
32: **update** θ_{F_t} by minimizing $\mathcal{L}_{inh} + \mathcal{L}_{tune}$ using the Adam optimizer
33: **end while**

3. Alternate methods to generate OOD samples

We argue in Sec. 4.1 of the paper that an *inheritable* model for the task of *open-set* DA should have the ability to mitigate the overconfidence issue. We achieve this by training an *out-of-distribution* (OOD) classifier (G_n). We explored potential techniques to generate OOD samples, and as mentioned in Sec. 4.1b in the paper, we found that the feature-splicing technique works well in practice for training an *inheritable* model for *open-set* DA. In Table 2, we report the adaptation performance on **Office-31** using *inheritable* models trained with different OOD generation strategies. Here, we discuss the two other strategies we explored.

a) Linear interpolation between classes. We randomly choose a pair of source instances corresponding to different classes, and obtain a *negative* feature by linearly interpolating between the features of the two instances as,

$$u_n = \gamma \times u_s^{c_i} + (1 - \gamma) \times u_s^{c_j} \quad (8)$$

where $\gamma \sim \text{Beta}(2, 2)$ as proposed in [6], $u_s^{c_i} = M_s(x_s^{c_i})$, $u_s^{c_j} = M_s(x_s^{c_j})$, and $x_s^{c_i}, x_s^{c_j}$ are two instances sampled from different source classes, *i.e.* $c_i, c_j \in \mathcal{C}_s$ ($i \neq j$). This is inspired by techniques such as *mixup* [7, 6] which encourage less confident predictions on the interpolations of latent features. As reported in Table 2, we find that linear interpolation performs worse than feature-splicing. This is because linear interpolation is less effective in producing OOD samples as it yields features from a constrained region between the source classes (as discussed in Sec. 4.1b of the paper, and shown in Fig. 3A of the paper). In contrast, feature-splicing is able to mimic plausible OOD samples through the suppression of *class-specific* traits.

b) Random suppression of the most active features. Given a latent-space feature, we randomly scale down the values of top-15 percentile activations as,

$$u_n = \Gamma \odot u_s \quad (9)$$

where $u_s = M_s(x_s)$ and Γ is a weight vector containing $\gamma \sim \text{uniform}(0.2, 0.3)$ at the indices corresponding to the top-15 percentile activations, and 1 elsewhere, and \odot denotes element-wise product. In Table 2, we observe that this performs similar to feature-splicing. Essentially, by suppressing the top activations, we obtain a feature which is devoid of the *class-specific* traits (Sec. 4.1b of the paper) thereby resulting in a plausible OOD sample. However, this method of random suppression requires a hyperparameter search to identify an optimal range for the scale of suppression (in this case, 0.2 to 0.3). The feature-splicing technique is meant to avoid this hyperparameter search, by choosing the appropriate replacement for the class-discriminative activations for a given instance.

Table 2. Adaptation performance (OS) of different OOD generation strategies, on **Office-31** (ResNet-50). $|\mathcal{C}_s| = 10$, $|\mathcal{C}_t| = 20$. See Sec. 3 for Interpolation (Sec. 3a) and Suppression (Sec. 3b).

Method	A→W	A→D	D→W	W→D	D→A	W→A	Avg
Interpolation	83.4	86.7	95.1	96.9	79.3	76.2	86.3
Suppression	91.0	93.3	96.4	99.6	90.2	88.2	93.1
Feature-splicing	91.3	94.2	96.5	99.5	90.1	88.7	93.4

4. Miscellaneous

In this section, we provide the details of the machine and the datasets used for experiments.

4.1. Training time comparison

The machine used to run all our experiments has the following hardware specifications. *CPU*: Intel Core i7-7700K, *GPU*: NVIDIA GeForce GTX 1080 with CUDA v8.0.61.

The training time reported in Sec. 5.3e in the paper is obtained on the machine with the above specifications. For a fair evaluation, we compare the training time required for our method and the previous state-of-the-art method STA [2] on identical settings. We use a batch size of 32, with identical optimizers and learning rates for adaptation.

a) STA. The STA method adapts to target domain in two steps. The first step trains a multi-binary classifier and a domain discriminator using both source and target samples (16 instances each, in a batch) and the second step involves adaptation to the target domain using both source and target samples (again, 16 instances from each domain). For both **A→D** and **A→W**, on an average, Step 1 took 177s while Step 2 took 398s.

b) Ours. For our method, we fixed all hyperparameters to the values as mentioned in Sec. 5.1b of the paper. Overall, our method is much more efficient in the case of multiple *clients*. This is clearly because in our approach, the source training is done only once (by the *vendor*) requiring about 250s. Following this, both the *clients* use the same *vendor* model to adapt to their respective target domains requiring 69s on an average. This is contrast to STA where each client has to train on the *vendor*'s source dataset, requiring additional computation during adaptation.

4.2. Dataset description

In our experiments, we follow STA [2] to choose the label sets. See Fig. 1 for sample images from each dataset.

The **Office-31** [4] dataset contains 4652 images from 3 domains: Amazon (**A**), DSLR (**D**) and Webcam (**W**). In alphabetical order, the first 10 classes are used as shared classes and, the classes 21-31 are chosen as target-*unknown*.

The **Office-Home** [5] dataset was curated by crawling the web, and thus exhibits a higher *domain-shift* as compared



Figure 1. Sample images from the benchmark datasets: **Office-31** showing images belonging to class “Speaker”, **Office-Home** showing images belonging to class “Flower” and **VisDA** showing images belonging to class “Truck”.

to the **Office-31** dataset. It contains 65 classes of objects with about 15,500 images split into four domains: Art (**Ar**), Clipart (**Cl**), Product (**Pr**) and Real-World (**Rw**). The first 25 classes in alphabetical order are chosen as shared classes and classes 26-65 are chosen as the target-*unknown* classes.

The **VisDA**[3] dataset exhibits a significant amount of *domain-shift* between its two domains: Synthetic (**S**) and Real World (**R**) having about 150k images and 56k images respectively. The Synthetic domain was created by rendering 3D models. The following classes are selected as the shared classes, $C_s = \{\text{bicycle, bus, car, motorcycle, train, truck}\}$, while the target-*unknown* classes are chosen as, $C_t^{uk} = \{\text{aeroplane, horse, knife, person, plant, skateboard}\}$.

References

- [1] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2010. 1
- [2] Hong Liu, Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Qiang Yang. Separate to adapt: Open set domain adaptation via progressive separation. In *CVPR*, 2019. 4
- [3] Xingchao Peng, Ben Usman, Neela Kaushik, Judy Hoffman, Dequan Wang, and Kate Saenko. Visda: The visual domain adaptation challenge. In *CVPRW*, 2018. 5
- [4] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *ECCV*, 2010. 4
- [5] Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. In *CVPR*, 2017. 4
- [6] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In *ICML*, 2019. 4
- [7] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018. 4