Contrastive Vicinal Space for Unsupervised Domain Adaptation: Supplementary Material

Jaemin Na[1], Dongyoon Han[2], Hyung Jin Chang[3], and Wonjun Hwang[1]

[1]Ajou University, Korea [2]NAVER AI Lab [3]University of Birmingham, UK
osial46@ajou.ac.kr, dongyoon.han@navercorp.com,
h.j.chang@bham.ac.uk, wjhwang@ajou.ac.kr

A. Additional Experimental Results

A.1. Effects of our components with a different baseline.

In the main paper, we provided the effect of our components with baseline, MSTN [18]. We further investigate our method using DANN (Ganin *et al.,* JMLR 2016) [2] as a baseline, which is one of the simplest methods in unsupervised domain adaptation. As in Table A.1., we observed that each component is still effective even with the light baseline DANN. Note that we only obtain the initial weights from the baseline and do not use any losses from the baseline when training our method.

Baseline	$\mathcal{R}_{emp}$	$\mathcal{R}_{ct}$	$\mathcal{R}_{cs}$	A→W	D→W	W→D	A→D	D→A	W→A	Avg.
✓				82.0	96.9	99.1	79.7	68.2	67.4	82.2
✓	✓			94.5	99.0	100.0	94.2	75.6	75.2	89.8
✓	✓	✓		95.5	99.2	100.0	94.4	76.0	76.3	90.2
✓	✓	✓	✓	**95.6**	**99.2**	**100.0**	**95.8**	**76.9**	**78.3**	**91.0**

Table A.1. Ablation results (%) of investigating the effects of our components with baseline DANN on Office-31.

A.2. Empirical visualization of vicinal space.

We computed the entropy of vicinal instances in task A→W on Office-31 to support the demo Figure 1 in the main paper. As in Figure A.2.a, we observed that the entropy maximization point (*i.e.,* EMP) is biased toward the target domain before adaptation. Here, we define contrastive space within a certain margin from EMP. On the other hand, after applying our method, we observed that the EMP is formed near the center of the source and target domains (see Figure A.2.b).

A.3. The equilibrium collapse of labels in other scenarios.

As discussed in Section 4.3, the equilibrium collapse of labels problem occurs before adaptation by dominant-source and recessive-target vicinal instances. We

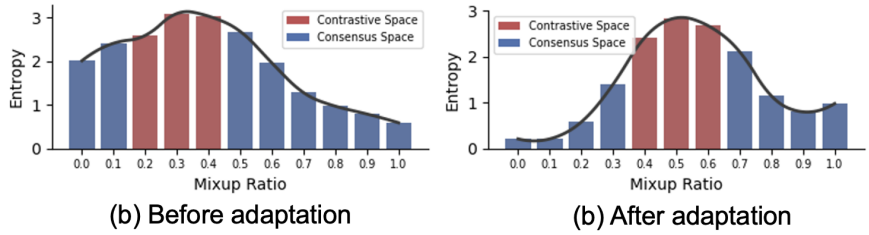

Fig. A.2. Empirical visualization of vicinal space.

analyzed whether this problem still exists after applying other UDA methods in Figure A.3.a. In this experiment, we use DANN as a baseline, which has relatively low accuracy (82.2%). We observe that there is still the problem of the equilibrium collapse of labels in some tasks. On the other hand, FixBi (Na *et al.*, CVPR 2021) (91.4%) achieved an equilibrium similar to the supervised learning method in all tasks. In addition, we experimented on both single-source and multi-source scenarios in Office-Home and PACS datasets, respectively. As shown in Figure A.3.b, we discovered that the problem of equilibrium collapse of labels occurs in both cases.

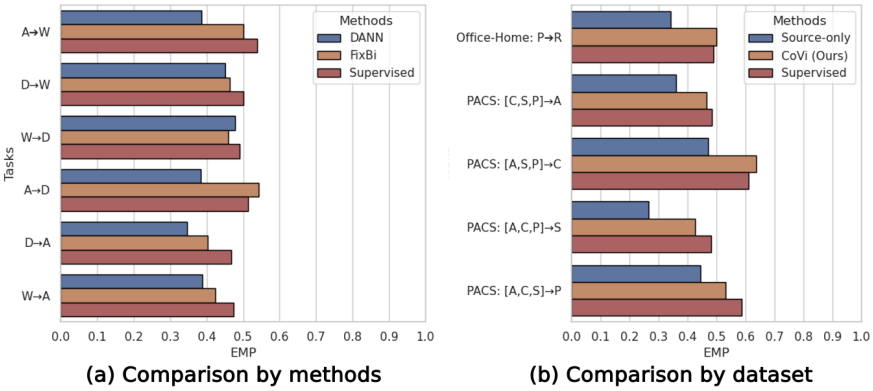

Fig. A.3. Ablation studies on the 'equilibrium collapse of labels'.

B. Implementation Details

B.1. Network Architectures

We describe the details of network architectures according to the dataset. As introduced in the main paper, our model consists of three subcomponents: an encoder, a classifier, and an EMP-learner.

Encoder. Following the standard architecture of previous studies on unsupervised domain adaptation [11,16], we adopt an ImageNet [6]-pretrained

ResNet [4,3] for the encoder. We use ResNet-50 for Office-31 [14] and Office-Home [17], and ResNet-101 for VisDA-C [13] dataset. For multi-source domain adaptation, we use ResNet-18 for PACS [7] dataset.

EMP-learner. We introduce a small network to produce entropy maximization points (EMPs) according to the convex combinations of the source and target instances. We design the EMP-learner with four convolutional layers, regardless of the dataset. We construct the EMP-learner with three 3x3 convolutional layers with stride one followed by Batch Normalization [5] and ReLU [12]. For the last layer, instead of the fully connected layer, we adopt 1x1 convolution [9]. The output channel of the last 1x1 convolutional layer is 11, yielding a ratio $\lambda \in \{0.0, 0.1, ..., 1.0\}$.

Classifier. We adopt only one fully connected layer for the classifier. The input feature size of the fully connected layer is decided by the output feature size of the encoder. The output feature size of the fully connected layer depends on the number of categories in each dataset.

B.2. Data Configurations

We implement our algorithm using PyTorch. The code runs with Python 3.7+, PyTorch 1.7.1, and Torchvision 0.8.2. In this section, we provide our training recipes for Office-31, Office-Home, VisDA-C, and PACS dataset.

Office-31 and Office-Home. In Configs 1, we describe our default configuration for Office-31 and Office-Home. The default configs for Office-Home are almost identical to Office-31, except for the resize factor of test transform that uses a scaling factor of 256 instead of 224.

Configs 1 PyTorch-style configs for Office-31 and Office-Home.

```
train_transforms = torch.nn.Sequential(
    transforms.Resize(256),
    transforms.RandomCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(
     mean=[0.485, 0.456, 0.406],
     std=[0.229, 0.224, 0.225]))
test_transforms = torch.nn.Sequential(
    transforms.Resize(224),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(
     mean=[0.485, 0.456, 0.406],
     std=[0.229, 0.224, 0.225]))
```

VisDA-C. We provide the configurations for VisDA-C in Configs 2. We use a stochastic gradient descent optimization (SGD) with a training batch size of 128, a momentum of 0.9, and a learning rate of 1e-4. The end-to-end pipeline is

trained for 100 epochs. We use the center crop instead of the random crop for image transformations in the training process. It is worth noting that we do not use the ten-crops ensemble technique used in [19,1,10] during evaluation for a fair comparison.

Configs 2 PyTorch-style configs for VisDA-C.

```
train_transforms = torch.nn.Sequential(
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]))
test_transforms = torch.nn.Sequential(
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]))
```

PACS. Following the previous protocols [8,15] for multi-source domain adaptation, we train on any three of the four domains (i.e., source domains) and then test on the remaining one domain (i.e., target domain). The total epoch is 100, with a batch size of 32 for the PACS dataset. The training details are described in Configs 3.

Configs 3 PyTorch-style configs for PACS.

```
train_transforms = torch.nn.Sequential(
    transforms.Resize(256),
    transforms.RandomCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225])
test_transforms = torch.nn.Sequential(
    transforms.Resize(224),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]))
```

C. Pseudocode

In Pseudo-code 1, 2, and 3, we provide PyTorch-like pseudo-codes for the EMP-Mixup, contrastive loss, and consensus loss, respectively. The entire code has been released at https://github.com/NaJaeMin92/CoVi.

Pseudo-code 1 PyTorch-like style pseudocode for EMP-Mixup.

```
# x_s, y_s: Source image and label
# x_t: Target image
# f: An encoder
# h: A classifier
# g: An EMP-learner
# ce_loss: Cross entropy loss

# compute embeddings except for avgpool in f
z_s, z_t = f(x_s), f(x_t)

# concat representations along the channel dimension
z_c = torch.cat([z_s, z_t], dim=1)

# Produce entropy maximization points
emp = torch.argmax(g(z_c), dim=1) * 0.1

# construct vicinal instances with EMP
x_emp = emp * x_s + (1 - emp) * x_t
z_emp = h(f(x_emp))

# compute entropy loss
entropy_loss = -Entropy(z_emp)

# optimization step
entropy_loss.backward()
update(g.params)

# compute cross-entropy loss
y_t = torch.argmax(h(f(x_t)), dim=1)
mixup_loss = emp * ce_loss(z_emp, y_s) + (1 - emp) * ce_loss(z_emp, y_t)

# optimization step
mixup_loss.backward()
update(f.params)
update(h.params)
```

Pseudo-code 2 PyTorch-like style pseudocode for contrastive loss.

```
# x_s, y_s: Source image and label
# x_t: Target image
# f: An encoder
# h: A classifier
# emp: Entropy maximization point
# w: Margin of ratio
# alpha: Confidence threshold
# space_sd: Source-dominant space constraint
# space_td: Target-dominant space constraint
# ce_loss: Cross entropy loss
# In practice, we replace top2_sd with y_hat in swap prediction to take
advantage of the higher accuracy top1 label. Also, we replace top2_td
with y_s because we can access source labels.

# construct space
sd_ratio, td_ratio = emp - w, emp + w
sd_cont = torch.ge(sd_ratio, space_sd)
td_cont = torch.le(td_ratio, space_td)

# compute threshold mask
z_t = f(x_t)
top1_prob = torch.topk(F.softmax(z_t, dim=1), k=1)[0].t().squeeze()
prob_mean, prob_std = top1_prob.mean(), top1_prob.std()
threshold = prob_mean - alpha * prob_std
th_mask = top1_prob.ge(threshold)

# construct vicinal instances
mask_idx = torch.nonzero(th_mask & td_cont & sd_cont).squeeze()
x_sd = emp[mask_idx] * x_s[mask_idx] + (1 - emp[mask_idx]) * x_t[mask_idx]
x_td = emp[mask_idx] * x_s[mask_idx] + (1 - emp[mask_idx]) * x_t[mask_idx]

# compute representations
z_sd, z_td = h(f(x_sd)), h(f(x_td))

# predict top-2 labels
top1_sd, top2_sd = torch.topk(F.softmax(z_sd, dim=1), k=2)[1].t()
top1_td, top2_td = torch.topk(F.softmax(z_td, dim=1), k=2)[1].t()

# swap predictions and compute contrastive loss
y_hat = torch.argmax(h(f(x_t)), dim=1)
sd_loss = sd_ratio * ce_loss(z_sd, top2_sd) + (1 - sd_ratio) *
ce_loss(z_sd, top1_td)
td_loss = td_ratio * ce_loss(z_td, top2_td) + (1 - td_ratio) *
ce_loss(z_td, top1_sd)

contrastive_loss = sd_loss + td_loss

# optimization step
contrastive_loss.backward()
update(f.params)
update(h.params)
```

Pseudo-code 3 PyTorch-like style pseudocode for consensus loss.

```
# x_s: Source image
# x_t: Target image
# f: An encoder
# h: A classifier
# w: Margin of ratio
# beta: Confidence threshold
# ce_loss: Cross entropy loss

# construct two perturbed versions
shuffle_idx = torch.randperm(batch_size)
x_v1 = lam * x_s + (1 - lam) * x_t
x_v2 = lam * x_s[shuffle_idx] + (1 - lam) * x_t

# construct representations
z_v1 = h(f(x_v1))
z_v2 = h(f(x_v2))

# compute threshold mask
z_t = f(x_t)
top1_prob = torch.topk(F.softmax(z_t, dim=1), k=1)[0].t().squeeze()
prob_mean, prob_std = top1_prob.mean(), top1_prob.std()
threshold = prob_mean - beta * prob_std
th_mask = top1_prob.ge(threshold)
mask_idx = torch.nonzero(th_mask).squeeze()

# Aggregate softmax probabilities
p = F.softmax(z_v1, dim=1) + F.softmax(z_v2, dim=1)

# compute consensus loss
y_hat = torch.argmax(p, dim=1)
loss = ce_loss(z_v1[mask_idx], y_hat[mask_idx]) + ce_loss(z_v2[mask_idx],
y_hat[mask_idx])

# optimization step
loss.backward()
update(f.params)
update(h.params)
```

References

1. Cao, Z., Ma, L., Long, M., Wang, J.: Partial adversarial domain adaptation. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 135–150 (2018) 4
2. Ganin, Y., Lempitsky, V.: Unsupervised domain adaptation by backpropagation. In: International conference on machine learning. pp. 1180–1189. PMLR (2015) 1
3. 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) 3
4. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: European conference on computer vision. pp. 630–645. Springer (2016) 3
5. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning. pp. 448–456. PMLR (2015) 3
6. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems **25**, 1097–1105 (2012) 2
7. Li, D., Yang, Y., Song, Y.Z., Hospedales, T.M.: Deeper, broader and artier domain generalization. In: Proceedings of the IEEE international conference on computer vision. pp. 5542–5550 (2017) 3
8. Li, R., Jia, X., He, J., Chen, S., Hu, Q.: T-svdnet: Exploring high-order prototypical correlations for multi-source domain adaptation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 9991–10000 (2021) 4
9. Lin, M., Chen, Q., Yan, S.: Network in network. arXiv preprint arXiv:1312.4400 (2013) 3
10. Long, M., Cao, Z., Wang, J., Jordan, M.I.: Conditional adversarial domain adaptation. arXiv preprint arXiv:1705.10667 (2017) 4
11. Na, J., Jung, H., Chang, H.J., Hwang, W.: Fixbi: Bridging domain spaces for unsupervised domain adaptation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1094–1103 (2021) 2
12. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Icml (2010) 3
13. Peng, X., Usman, B., Kaushik, N., Hoffman, J., Wang, D., Saenko, K.: Visda: The visual domain adaptation challenge. arXiv preprint arXiv:1710.06924 (2017) 3
14. Saenko, K., Kulis, B., Fritz, M., Darrell, T.: Adapting visual category models to new domains. In: European conference on computer vision. pp. 213–226. Springer (2010) 3
15. Seo, S., Suh, Y., Kim, D., Kim, G., Han, J., Han, B.: Learning to optimize domain specific normalization for domain generalization. In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXII 16. pp. 68–83. Springer (2020) 4
16. Tang, H., Chen, K., Jia, K.: Unsupervised domain adaptation via structurally regularized deep clustering. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 8725–8735 (2020) 2
17. Venkateswara, H., Eusebio, J., Chakraborty, S., Panchanathan, S.: Deep hashing network for unsupervised domain adaptation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 5018–5027 (2017) 3
18. Xie, S., Zheng, Z., Chen, L., Chen, C.: Learning semantic representations for unsupervised domain adaptation. In: International conference on machine learning. pp. 5423–5432. PMLR (2018) 1

19. Xu, R., Li, G., Yang, J., Lin, L.: Larger norm more transferable: An adaptive feature norm approach for unsupervised domain adaptation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 1426–1435 (2019) 4