A. More Model and Training Details

Our implementation is based on the timm library\(^1\). We use ViT-B/16 [2] (vit\(_{\text{base}}\_\text{patch16}\_224\) in timm) and ViT-S/16 [2] (vit\(_{\text{small}}\_\text{patch16}\_224\) in timm) as the vision transformer backbones in the paper. Transformer weights are restored from the checkpoints released by official Google JAX implementation\(^2\), which are obtained by first training on ImageNet-21k [7] and then fine-tuning on ImageNet-1k [7, 8]. The classifier head consists of a bottleneck module (\(\text{Linear} \rightarrow \text{BatchNorm1d} \rightarrow \text{ReLU} \rightarrow \text{Dropout}(0.5)\)) and a class predictor (\(\text{Linear} \rightarrow \text{ReLU} \rightarrow \text{Dropout}(0.5) \rightarrow \text{Linear}\)). The domain discriminator has the same network structure as the class predictor except having only one output.

During the training procedure, images are first resized to 256 × 256 pixels, randomly flipped horizontally, and then randomly cropped and resized to 224 × 224 pixels. The only exception is for VisDA-2017 [6], where center-cropping of size 224 × 224 is used. During the test procedure, images are first resized to 256 × 256 pixels and then center-cropped to 224 × 224 pixels.

To train the model, we adopt mini-batch Stochastic Gradient Descent (SGD) with momentum of 0.9. Learning rate is scheduled as \(lr = lr_0 \cdot (1 + 1e^{-3} \cdot i)^{-0.75}\), where \(lr_0\) is initial learning rate, \(i\) is training step. The learning rate of parameters of vision transformer backbone is set to be 1/10 of \(lr\). Complete hyper-parameters used for our experiments are listed in Tab. 1. Note that the same hyper-parameters are used for source-only training and baseline methods whenever applicable.

B. More Analysis on Multi-layer Perturbation

Figure 1 provides additional results when adding the same amount of perturbation to each layer while not using safe training. As can be seen in the left figure, the best layer to apply perturbation varies across tasks. Besides, a layer that works for one task may fail on others. To see the importance of allowing gradient back-propagation for \(b^l_x\), see Sec. 3.3 and Sec. 3.4 in the paper, the right figure shows that the model collapses when add perturbation to relatively deep layers while blocking the gradients of \(b^l_x\).

Table 5 includes comparison results when adding the perturbation to raw input or a single layer (\{0\} or \{4\} or \{8\}) in our proposed SSRT method. As can be seen, perturbing raw input performs similarly to perturbing the 0-th transformer block. Besides, perturbing any single layer degrades the performance on some adaptations tasks. In contrast, multi-layer perturbation combines their merits and obtains the best results.

---

\(^1\)https://github.com/rwightman/pytorch-image-models/blob/master/timm/models/vision_transformer.py
\(^2\)https://github.com/google-research/vision_transformer

<table>
<thead>
<tr>
<th>Name</th>
<th>Office-31</th>
<th>Office-Home</th>
<th>VisDA-2017</th>
<th>DomainNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\alpha)</td>
<td>0.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\beta)</td>
<td>0.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\epsilon)</td>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(T)</td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(L)</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\text{batch}_\text{size})</td>
<td>64 (32 source images + 32 target images)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\text{center}_\text{crop})</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>(\text{lr}_0)</td>
<td>0.001</td>
<td>0.004</td>
<td>0.002</td>
<td>0.004</td>
</tr>
<tr>
<td>(\text{max}_\text{jiers})</td>
<td>10k</td>
<td>20k</td>
<td>20k</td>
<td>40k</td>
</tr>
<tr>
<td>(\text{bottleneck}_\text{dim})</td>
<td>1024</td>
<td>2048</td>
<td>1024</td>
<td>1024</td>
</tr>
</tbody>
</table>
C. More Analysis on Bi-directional Self-Refinement

Table 2 provides additional results when blocking gradient back-propagation for different variables. Similar to the results listed in the paper (see Tab. 7), allowing gradient back-propagation of the teacher probabilities in KL divergence and \( b^l_x \) works better than other variants.

Table 2. Blocking gradient back-propagation for different variables. Note that \( p_x \) and \( \bar{p}_x \) in the table only refer to the teacher probability in KL divergence. (Safe Training not applied)

<table>
<thead>
<tr>
<th>( \omega )</th>
<th>( b^l_x )</th>
<th>( p_x )</th>
<th>( \bar{p}_x )</th>
<th>Cl→Ar</th>
<th>Cl→Pr</th>
<th>Cl→Rw</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \omega = 0 )</td>
<td>( \times )</td>
<td>1.61</td>
<td>12.71</td>
<td>6.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \omega = 1 )</td>
<td>( \times )</td>
<td>81.17</td>
<td>85.00</td>
<td>87.28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \omega \sim B(0.5) )</td>
<td>( \times )</td>
<td>83.68</td>
<td>85.69</td>
<td>88.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \omega \sim B(0.5) )</td>
<td>( \times )</td>
<td>84.55</td>
<td>87.27</td>
<td>89.49</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \omega \sim B(0.5) )</td>
<td>( \times )</td>
<td>85.21</td>
<td>87.88</td>
<td>89.58</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

D. More Analysis on Safe Training

In our method, we adopt a Confidence Filter to remove noisy supervisions. If it not used (i.e., \( \epsilon = 0 \)), the performance may deteriorate. Table 3 shows that using Safe Training can avoid significant performance drops, making the method much safer.

Table 3. Accuracies (%) without Confidence Filter. (†Safe Training not applied)

<table>
<thead>
<tr>
<th></th>
<th>Cl→Ar</th>
<th>Cl→Pr</th>
<th>Cl→Rw</th>
<th>Pr→Ar</th>
<th>Pr→Cl</th>
<th>Pr→Rw</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline-B</td>
<td>80.06</td>
<td>84.12</td>
<td>86.67</td>
<td>79.52</td>
<td>67.03</td>
<td>89.44</td>
</tr>
<tr>
<td>SSRT-B†</td>
<td>59.33</td>
<td>86.98</td>
<td>89.74</td>
<td>73.92</td>
<td>20.30</td>
<td>90.59</td>
</tr>
<tr>
<td>SSRT-B</td>
<td>84.51</td>
<td>86.98</td>
<td>89.30</td>
<td>82.65</td>
<td>67.79</td>
<td>91.16</td>
</tr>
</tbody>
</table>

E. Analysis on Model’s Robustness

In our proposed SSRT, we use perturbed target domain data to refine the model during the training procedure. In this section, we provide analysis on model’s robustness against perturbation during the test procedure. For each testing target domain data, we follow the same way as described in the paper to add a random offset to its latent token sequence, and use the perturbed token sequence to make prediction. To analyze model’s robustness against perturbation at different layers, we add perturbation to different transformer block as well as the raw input. The perturbation magnitude is controlled by a scalar \( \alpha \) as used in the paper. Figure 3 shows results (averaged over 6 random runs) on \( Pr \rightarrow Ar \) and \( clp \rightarrow pnt \). As can be seen, our method is more robust than Baseline. Even when adding a larger amount of perturbation (\( \alpha = 0.4 \)) than seen during training, SSRT incurs less accuracy decrease.

F. Comparison with SSL methods

Since Unsupervised Domain Adaptation (UDA) is closely related to Semi-Supervised Learning (SSL), in this section, we compare our method with two representative techniques in SSL, i.e., Mixup [11] and VAT [4].

Mixup regularizes the model to predict linearly between samples. Specifically, let \( x_1 \) and \( x_2 \) be two target domain data, \( p_1 = h(x_1) \) and \( p_2 = h(x_2) \) be the corresponding model predictions, Mixup first interpolates between two samples by

\[
\lambda \sim \text{Beta}(\alpha, \alpha)
\]

\[
x' = \lambda x_1 + (1 - \lambda) x_2
\]

\[
p' = \lambda p_1 + (1 - \lambda) p_2
\]

Its loss function is

\[
\mathcal{L}_{\text{mixup}} = E_{x_1, x_2 \sim D_t} \| h(x') - p' \|^2
\]

VAT enforces the model to predict consistently within the norm-ball neighborhood of each target data \( x \). Its loss function is

\[
\mathcal{L}_{\text{VAT}} = E_{x \sim D_t} \left[ \max_{\|r\| \leq \rho} D_{KL}(h(x) || h(x + r)) \right]
\]

We use \( \mathcal{L}_{\text{mixup}} \) and \( \mathcal{L}_{\text{VAT}} \) as the \( \mathcal{L}_{\text{tgt}} \) in our objective function. The trade-off parameter \( \beta \) is set to be 0.2 for both, same as used in our method. For Mixup, \( \alpha \) is set to be 0.5. We linearly ramp up \( \beta \) to its maximum value over 1/4 of all training steps as used in [1,9]. Instead of interpolating probabilities, we interpolate unnormalized logits, as it is shown to perform slightly better. For VAT, \( \rho \) is set to be 100. Both two techniques are applied to the raw input images.

Table 4 presents results on three benchmarks using ViT-base backbone. Detailed numbers can be found in Tables 5-7. On Office-Home [10] and VisDA-2017 [6], Mixup and VAT perform better than Baseline-B, and slightly worse than ours. On DomainNet [5], VAT still works. However, for Mixup, although we tried different hyper-parameters, it is still inferior to Baseline-B. Figure 2 shows two adaptations tasks where Mixup fails.

Table 4. Comparisons with SSL methods. \( X^{†} \) means averaged over all 5 tasks with \( X \) being the target domain.

<table>
<thead>
<tr>
<th></th>
<th>Office-Home</th>
<th>VisDA</th>
<th>DomainNet</th>
<th>clp†</th>
<th>inf†</th>
<th>pnt†</th>
<th>qdr†</th>
<th>rel†</th>
<th>skt†</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline-B</td>
<td>81.1</td>
<td>85.2</td>
<td>38.5</td>
<td>50.6</td>
<td>25.6</td>
<td>44.9</td>
<td>11.6</td>
<td>57.0</td>
<td>41.5</td>
</tr>
<tr>
<td>Mixup-B</td>
<td>83.2</td>
<td>88.2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>VAT-B</td>
<td>84.1</td>
<td>88.5</td>
<td>41.1</td>
<td>54.8</td>
<td>27.6</td>
<td>48.3</td>
<td>12.5</td>
<td>58.4</td>
<td>45.0</td>
</tr>
<tr>
<td>SSRT-B</td>
<td>85.4</td>
<td>88.8</td>
<td>45.2</td>
<td>60.0</td>
<td>28.2</td>
<td>53.3</td>
<td>13.7</td>
<td>65.3</td>
<td>50.4</td>
</tr>
</tbody>
</table>
Figure 2. Mixup with different hyper-parameters. The legend for Mixup is formed as Mixup(β, αx).

Figure 3. Analysis of model’s robustness. The dashlines indicate true test accuracy on the target domain data. The bars show decreases of accuracies when adding perturbations to different layers during the test procedure.

Table 5. Accuracies (%) on DomainNet. In each sub-table, the column-wise means source domain and the row-wise means target domain. “-S/B” indicates ViT-small/base backbones, respectively.
Table 6. Accuracies (%) on Office-Home.

<table>
<thead>
<tr>
<th>Method</th>
<th>Ar→Cl</th>
<th>Ar→Pr</th>
<th>Ar→Rw</th>
<th>Cl→Ar</th>
<th>Cl→Pr</th>
<th>Cl→Rw</th>
<th>Pr→Ar</th>
<th>Pr→Cl</th>
<th>Pr→Rw</th>
<th>Rw→Ar</th>
<th>Rw→Cl</th>
<th>Rw→Pr</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline-B</td>
<td>66.96</td>
<td>85.74</td>
<td>88.07</td>
<td>80.06</td>
<td>84.12</td>
<td>86.67</td>
<td>79.52</td>
<td>67.03</td>
<td>89.44</td>
<td>83.64</td>
<td>70.15</td>
<td>91.17</td>
<td>81.05</td>
</tr>
<tr>
<td>Mixup-B [11]</td>
<td>71.32</td>
<td>86.66</td>
<td>88.82</td>
<td>82.45</td>
<td>84.79</td>
<td>87.58</td>
<td>82.90</td>
<td>71.68</td>
<td>90.77</td>
<td>85.46</td>
<td>74.36</td>
<td>91.37</td>
<td>83.18</td>
</tr>
<tr>
<td>VAT-B [4]</td>
<td>71.52</td>
<td>89.39</td>
<td>90.48</td>
<td>86.11</td>
<td>88.53</td>
<td>89.33</td>
<td>84.59</td>
<td>72.23</td>
<td>90.84</td>
<td>86.61</td>
<td>72.83</td>
<td>92.48</td>
<td>84.58</td>
</tr>
<tr>
<td>SSRT-B (ours)</td>
<td>75.17</td>
<td>88.98</td>
<td>91.09</td>
<td>85.13</td>
<td>88.29</td>
<td>89.95</td>
<td>85.04</td>
<td>74.23</td>
<td>91.26</td>
<td>85.70</td>
<td>78.58</td>
<td>91.78</td>
<td>85.43</td>
</tr>
</tbody>
</table>

Table 7. Accuracies (%) on VisDA-2017.

<table>
<thead>
<tr>
<th>Method</th>
<th>plane</th>
<th>bcycl</th>
<th>bus</th>
<th>car</th>
<th>horse</th>
<th>knife</th>
<th>mccyl</th>
<th>person</th>
<th>plant</th>
<th>sktbrd</th>
<th>train</th>
<th>truck</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline-B</td>
<td>98.55</td>
<td>82.59</td>
<td>85.97</td>
<td>57.07</td>
<td>94.93</td>
<td>97.20</td>
<td>94.58</td>
<td>76.68</td>
<td>92.11</td>
<td>96.54</td>
<td>94.31</td>
<td>52.24</td>
<td>85.23</td>
</tr>
<tr>
<td>Mixup-B [11]</td>
<td>98.88</td>
<td>86.56</td>
<td>88.64</td>
<td>72.32</td>
<td>98.06</td>
<td>98.07</td>
<td>95.91</td>
<td>83.00</td>
<td>94.09</td>
<td>98.07</td>
<td>94.55</td>
<td>50.36</td>
<td>88.21</td>
</tr>
<tr>
<td>VAT-B [4]</td>
<td>99.15</td>
<td>87.71</td>
<td>90.85</td>
<td>67.81</td>
<td>98.81</td>
<td>98.17</td>
<td>97.57</td>
<td>76.65</td>
<td>92.88</td>
<td>98.73</td>
<td>96.27</td>
<td>57.37</td>
<td>88.50</td>
</tr>
<tr>
<td>SSRT-B (ours)</td>
<td>98.93</td>
<td>87.60</td>
<td>89.10</td>
<td>84.77</td>
<td>98.34</td>
<td>98.70</td>
<td>96.27</td>
<td>81.08</td>
<td>94.86</td>
<td>97.90</td>
<td>94.50</td>
<td>43.13</td>
<td>88.76</td>
</tr>
</tbody>
</table>

G. Results with ViT-small Backbone

ViT-small is a smaller version of ViT-base by halving the number of Self-Attention Heads and token embedding dimension of ViT-base. It has fewer parameters (∼22M params) than ResNet-101 (∼45M params). We empirically found that it convergences much slower than ViT-base, so we double the maximum training iterations. An alternative is to pretrain on the source data first and then adapted to the target data. As can be seen from Tab. 5, our proposed SSRT-S achieves +5.1% higher accuracy than MDD+SCDA (ResNet-101 backbone) on DomainNet, despite that ViT-small has fewer parameters than ResNet-101.

References