Direct Distillation between Different Domains

Jialiang Tang[1,2,3], Shuo Chen[4⋆], Gang Niu[4], Hongyuan Zhu[5,6], Joey Tianyi Zhou[6,7,8], Chen Gong[1,2,3*], and Masashi Sugiyama[4,9]

[1] School of Computer Science and Engineering, Nanjing University of Science and Technology, China
[2] Key Laboratory of Intelligent Perception and Systems for High-Dimensional Information of Ministry of Education, China
[3] Jiangsu Key Laboratory of Image and Video Understanding for Social Security, China
[4] Center for Advanced Intelligence Project, RIKEN, Japan
[5] Institute for Infocomm Research (I^{2}R), A*STAR, Singapore
[6] Centre for Frontier AI Research (CFAR), A*STAR, Singapore
[7] Institute of High Performance Computing (IHPC), A*STAR, Singapore
[8] The Centre for Advanced Technologies in Online Safety, Singapore
[9] The Graduate School of Frontier Sciences, The University of Tokyo, Japan
shuo.chen.ya@riken.jp chen.gong@njust.edu.cn

Here, we provide supplementary materials for our proposed Direct Distillation between Different Domains (4Ds), including: 1) algorithm of 4Ds, 2) Traditional knowledge distillation basics (in Section 2); 3) Comparison of complexity between our method and two-stage approaches (in Section 3); 4) Pseudo-codes for key components in our proposed 4Ds (in Section 4); 5) Feature mapping operations (in Section 5); 6) Structural analysis of the knowledge adapter (in Section 6); 7) Parameter sensitivity analysis of hyper-parameters in our proposed 4Ds (in Section 7); 8) Learnable parameter (λ) analysis (in Section 8); and 9) Weighting strategies for domain-invariant features (in Section 9).

1 Algorithm

As the training algorithm summarized in Alg. 1, during each training iteration, both the teacher network and student network are alternately updated by minimizing $\mathcal{L}_{\text{total}}(\mathcal{N}_T)$ and $\mathcal{L}_{\text{total}}(\mathcal{N}_S)$, respectively. The teacher network is promoted to capture domain-specific knowledge in the target domain and learn category relations that the student network can effortlessly learn. Subsequently, the well-preserved domain-invariant knowledge and appropriate category relations are transmitted to the student network to improve its performance in the target domain. As the training of the teacher network only requires a few calculations where the adapters encompass merely 2% of the parameters of the entire teacher network, our proposed 4Ds can efficiently train a reliable student network in the target domain.

⋆ Corresponding authors

Algorithm 1 Direct Distillation between Different Domains

Require: Pre-trained teacher network $\mathcal{N}_T$ in the source domain, target data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{|\mathcal{D}|}$, trade-off parameters β and γ.

Initialize and integrate the learnable adapters into $\mathcal{N}_T$;
Initialize the small student network $\mathcal{N}_S$;
repeat
  Employ $\mathcal{N}_T$, $\mathcal{N}_S$ to obtain their predictions $\{\mathcal{N}_T(\mathbf{x}_i)\}_{i=1}^{|\mathcal{D}|}$, $\{\mathcal{N}_S(\mathbf{x}_i)\}_{i=1}^{|\mathcal{D}|}$, and domain-invariant features $\{\boldsymbol{\mathcal{P}}_{\text{act},i}^{T}\}_{i=1}^{|\mathcal{D}|}$, $\{\boldsymbol{\mathcal{P}}_{\text{act},i}^{S}\}_{i=1}^{|\mathcal{D}|}$;
  Calculate cross-entropy losses $\mathcal{L}_{\text{ce}}(\mathcal{N}_T)$ and $\mathcal{L}_{\text{ce}}(\mathcal{N}_S)$;
  Calculate knowledge transfer losses $\mathcal{L}_{\text{kt}}(\mathcal{N}_T)$ and $\mathcal{L}_{\text{kt}}(\mathcal{N}_S)$ via Eq. (13);
  Calculate domain-invariant knowledge transfer loss $\mathcal{L}_{\text{dikt}}(\mathcal{N}_S)$ via Eq. (11);
  Update the learnable adapters in $\mathcal{N}_T$ via Adam;
  Update the student network $\mathcal{N}_S$ via SGD;
until convergence

Ensure: Lightweight student network in the target domain.

2 Traditional Knowledge Distillation

Vanilla Knowledge Distillation (KD) methods train a compact student network $\mathcal{N}_S$ by learning from a powerful pre-trained teacher network $\mathcal{N}_T$. Formally, the training dataset for both $\mathcal{N}_T$ and $\mathcal{N}_S$ is denoted as $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{|\mathcal{D}|}$. Given an input example $\mathbf{x}_i \in \mathcal{D}$, vanilla KD is achieved by minimizing the following loss function:

$$\mathcal{L}_{\text{kd}}\left(\mathcal{N}_S\right) = \frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}\left[\beta\mathcal{H}_{\text{ce}}\left(\mathcal{N}_S\left(\mathbf{x}_i\right), y_i\right) + (1-\beta)\mathcal{H}_{\text{kt}}\left(\mathbf{f}_i^S, \mathbf{f}_i^T\right)\right], \tag{1}$$

where $\mathcal{H}_{\text{ce}}$ is the **c**ross-**e**ntropy loss function, and it promotes $\mathcal{N}_S$ to predict results as consistent as the ground-truth labels; $\mathcal{H}_{\text{kt}}$ stands for the **k**nowledge **t**ransfer loss function, which encourages $\mathcal{N}_S$ to mimic the knowledge $\mathbf{f}_i^T$ acquired by $\mathcal{N}_T$, and here the notations with superscripts "T" and "S" denote that they are related to $\mathcal{N}_T$ and $\mathcal{N}_S$, respectively; $\beta \in (0, 1)$ is the trade-off parameter to balance $\mathcal{H}_{\text{ce}}$ and $\mathcal{H}_{\text{kt}}$.

Traditional KD methods, however, may encounter difficulties when training the student network in a new target domain that is significantly different from the teacher's source domain, as discussed in the Introduction. Therefore, to address this issue, we propose 4Ds, a novel one-stage framework for distilling a student network in the new target domain.

3 Complexity Comparison between Two-Stage and Our Methods

In this section, we evaluate the training expenses of our one-stage 4Ds against two-stage methods. As mentioned in the Introduction, there are two possible two-stage methods that can be used to train a student network in the target domain

Table 1: The training costs and classification accuracies (%) of the student networks trained by various methods on domain adaptation task "Art" to "Clipart" (A2C) of the Office-Home dataset. "#Learn Param." means the number of learnable parameters, and "M" stands for million.

Algorithm	Required data	A2C task of Office-Home		
		#Learn Param.	GPU hours	Acc.
DAA-NRC	Target	33.49M	0.95	77.13
DAA-SHOT	Target	33.49M	0.87	80.37
DAA-ProMix	Target	33.49M	3.68	79.54
DAA-BAIT	Target	33.49M	0.99	80.02
AAD-NRC	Target & source	33.49M	0.88	77.83
AAD-SHOT	Target & source	33.49M	0.83	78.57
AAD-ProMix	Target & source	33.49M	3.42	78.66
AAD-BAIT	Target & source	33.49M	0.80	79.21
VKD	Target	11.69M	0.27	77.89
4Ds	Target	12.21M	0.32	**81.41**

utilizing a teacher network trained in the source domain, namely "Adaptation After Distillation" (AAD) and "Distillation After Adaptation" (DAA). These two-stage approaches necessitate extensive learning of either the entire teacher network or student network across both the source and target data, thereby incurring massive training costs. Conversely, our efficient one-stage method directly learns the student network and the small adapters in the teacher network on the target data. We illustrate this efficiency by training the ResNet34 (teacher) and ResNet18 (student) pair on the "Art" to "Clipart" (A2C) task from the Office-Home dataset, utilizing A100 GPU hours as a metric for evaluating training cost.

Table 1 presents the training durations and classification accuracies for student networks trained using different methods. Our one-stage 4Ds method demonstrates substantially lower training times than the two-stage methods while also delivering superior performance. Moreover, compared to VKD, 4Ds introduces a minimal increase of 0.52M learnable parameters yet yields a 3.52% enhancement in performance. These findings highlight the efficiency and effectiveness of our 4Ds approach in training reliable student networks for novel target domains.

4 Pseudo-Code

In this section, we provide the pseudo-codes for the adapter and fusion-activation mechanism. Alg. 2 shows the pseudo-code of the adapter, the input feature is first adapted by two 1×1 convolutional layers with Batch Normalization (BN) layer and ReLU activation function on the target data. Then, the adapted feature and

Algorithm 2 Pseudo-code of Adapter in PyTorch-like Style.

```
# conv: 1x1 convolutional layer
# decompose: frequency feature to amplitude and phase
# compose: amplitude and phase to frequency feature
# weights: learnable parameters, t: temperatures
def adapter(f):
  weights_soft = softmax(weights/t, dim=0)
  f_ad = bn(conv(max(0, bn(conv(f))))) # adapt f
  F = FT(f) # FT: fourier transform
  F_ad = FT(f_ad)
  a, p = decompose(F) # a: amplitude, p: phase
  a_ad, p_ad = decompose(F_ad)
  f_ift = IFT(compose(a_ad * weights_soft[0] + a * weights_soft[1], p))
  # IFT: inverse FT
  return f_ift
```

Algorithm 3 Pseudo-code of Fusion-Activation Mechanism in PyTorch-like Style.

```
# fs: a set of features
# B: batchsize
def fusion-activation(fs):
  _, _, H, W = fs[3].size() # H: height, W: width
  f0 = avg_pool(fs[0], (H, W)) # resize fs[0] to H x W
  f1 = avg_pool(fs[1], (H, W))
  f2 = avg_pool(fs[2], (H, W))
  f3 = fs[3]
  f = cat((f0, f1, f2, f3), dim=1) # fuse f0, f1, f2, f3 as f: B x M x H x W
  z = avg_pool(f, (1, 1)) # squeeze f to 1 x 1
  s = sigmoid(fc(relu(fc(z)))) # map z as attention weights
  f_act = f * s.expand_as(f) # activate f
  return f_act
```

input feature are converted as frequency features by the Fourier transform and decomposed into amplitude and phase components. Subsequently, the amplitude is refurbished by composing the amplitudes decomposed from the adapted feature and original input feature with the learnable weights. In the end, the refurbished amplitude and original phase are composed as frequency feature and transferred to spatial feature via the inverse Fourier transform.

The pseudo-code of the fusion-activation mechanism is depicted in Alg. 3. Firstly, the features produced by various blocks of the network are scaled to a uniform dimension of $H \times W$ and concatenated together. Then, the concatenated features are squeezed to the size of 1×1 and mapped as the attention weights. Finally, the concatenated features are activated by the attention weights.

5 Feature Mapping Operation

The domain-invariant transfer in our method may be hindered by the feature dimension mismatched between the teacher and student networks. Therefore, we address this issue by employing a feature mapping operation to align the feature dimensions of the teacher and student networks.

Given the fused features $\mathcal{P}^T_{\text{fuse}} \in \mathbb{R}^{C_T \times H_T \times W_T}$ and $\mathcal{P}^S_{\text{fuse}} \in \mathbb{R}^{C_S \times H_S \times W_S}$ of the teacher and student networks, respectively. Here, C, H, and W denote the

Table 2: Average classification accuracies (%) of the student networks trained by our method using adapters with different structures on the twelve tasks for the Office-Home dataset. "#Conv.", "Kernel size", "#Param.", and "FLOPs" denote the number of convolutional layers, the size of the convolutional kernel, the number of learnable parameters, and the FLOating-Point operations (FLOPs) in the related adapter, respectively. Meanwhile, "#Param.^{A}/#Param.^{T}" denotes the ratio of adapter and teacher network parameters, and "M" stands for million.

#Conv.	Kernel size	#Param.	#Param.^{A}/#Param.^{T}	FLOPs	Avg.
1	1×1	1.05M	4.82%	51.78M	82.09
2	1×1	0.52M	2.38%	26.19M	**82.77**
3	1×1	0.60M	2.75%	29.50M	82.57
4	1×1	0.66M	3.03%	32.81M	82.46
5	1×1	0.73M	3.35%	36.12M	82.48
1	3×3	9.44M	43.30%	462.82M	82.41
2	3×3	4.72M	21.65%	231.71M	82.35
3	3×3	5.31M	24.35%	260.71M	82.46
4	3×3	5.90M	27.06%	289.71M	82.43
5	3×3	6.49M	29.77%	318.71M	82.33

number of channels, height, and width of the corresponding feature, respectively. When the feature map dimensions of the teacher network are larger than those of the student network (*i.e.*, $H_T > H_S$ and $W_T > W_S$), we first use average pooling to scale down the dimensions of the teacher network's feature maps to match those of student network:

$$\mathcal{P}^T_{\text{fuse}} = \text{AvgPool}(\mathcal{P}^T_{\text{fuse}}, (H_S, W_S)). \tag{2}$$

Meanwhile, if the feature map dimensions of the student network are larger than those of the teacher network, the features of the student network are also scaled as:

$$\mathcal{P}^S_{\text{fuse}} = \text{AvgPool}(\mathcal{P}^S_{\text{fuse}}, (H_T, W_T)). \tag{3}$$

Subsequently, the fused features $\mathcal{P}^S_{\text{fuse}}$ are projected to align the channel dimension of $\mathcal{P}^T_{\text{fuse}}$ via two linear layers:

$$\mathcal{P}^S_{\text{fuse}} = \max(0, \mathbf{W}_2 \max(0, \mathbf{W}_1 \mathcal{P}^S_{\text{fuse}})). \tag{4}$$

Here, $\mathbf{W}_1 \in \mathbb{R}^{\frac{C_S}{r} \times C_S}$, and $\mathbf{W}_2 \in \mathbb{R}^{C_T \times \frac{C_S}{r}}$ are the weights for two linear layers, respectively. After dimension alignment, the fused features $\mathcal{P}^T_{\text{fuse}}$ and $\mathcal{P}^S_{\text{fuse}}$ can be subjected to subsequent feature activation operation and domain-invariant feature transfer operation.

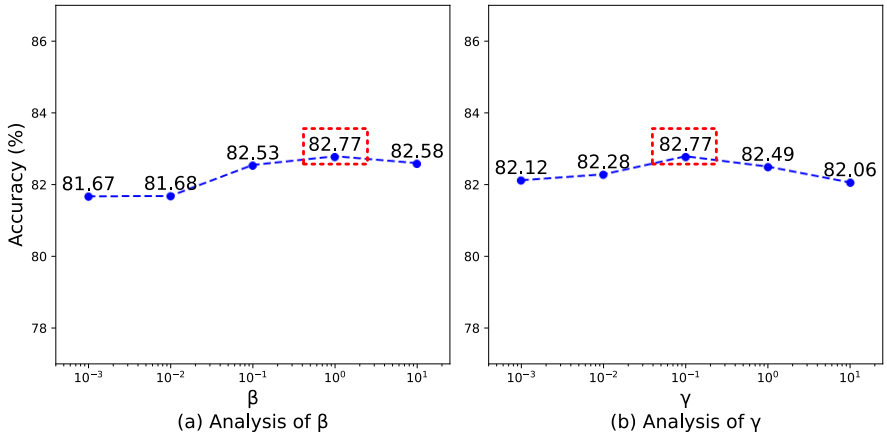

Fig. 1: Parametric sensitivities of (a) β and (b) γ in Eq. (14). The ResNet34 and ResNet18 are used as the teacher and student networks, respectively, and the training dataset is the Office-Home.

6 Structure Analysis of Knowledge Adapter

In our analysis, we examine the impact of different knowledge adapter structures within our 4Ds. In our proposed 4Ds, we default to configure the adapter with two 1×1 convolution layers followed by BN layers and ReLU activation function. Here, we experimented with adapters varying in convolution layer count and kernel size within the teacher network. We select the teacher-student pair ResNet34-ResNet18 to evaluate on the Office-Home dataset. Moreover, we input an image of size 3×224×224 into the teacher network to assess the FLOating-Point operations (FLOPs) required for the adapters, which is widely used to estimate the computational costs.

Table 2 reports the experimental results. Firstly, we can find that the adapter with two 1×1 convolution layers has the least number of parameters and computations while achieving the best performance. Therefore, we reasonably used this structure configuration of the knowledge adapter in our method. Secondly, although the adapters with 1×1 convolutional layers have significantly fewer parameters and FLOPs than those with 3×3 convolutional layers, their performance is comparable to that of the adapters with 3×3 convolutional layers. This indicates that adapters with 1×1 convolutional layers are sufficient to adapt the teacher network in the target domain. Note that adapters with only one convolutional layer have significantly higher parameters and FLOPs than adapters with multiple convolutional layers. This is because adapters with only one convolutional layer cannot utilize scaling parameters to reduce parameters and FLOPs.

7 Parametric Sensitivity

In our proposed 4Ds, there are two tuning parameters β=1.0 and γ=0.1 in Eq. (12) and Eq. (14). Here, we analyze the sensitivity of our 4Ds to these parameters on the Office-Home dataset with the teacher-student pair ResNet34-ResNet18. We examine the average accuracies of twelve tasks in the Office-Home dataset by changing one parameter while maintaining the others.

Fig. 1 illustrates the test accuracy curves for the student network when parameters change. The parameters β and γ fluctuate within $\{0.001, 0.01, 0.1, 1.0, 10\}$. Despite these parameters varying across a broad scope, we can observe that the accuracy curve for the student network maintains general stability. These experimental results indicate that the performance of the student network is robust to parameter modifications. Consequently, the parameters in our 4Ds are readily adjustable.

8 Analysis of λ

In our designed knowledge adapter, there is a learnable parameter $\lambda \in [0, 1]$ used to balance the information of the source and target domains, as follows:

$$\boldsymbol{\alpha}_{\text{ref}}^T = \lambda \boldsymbol{\alpha}_{\text{ad}}^T + (1 - \lambda)\boldsymbol{\alpha}^T. \tag{5}$$

Here, $\boldsymbol{\alpha}_{\text{ad}}^T$ and $\boldsymbol{\alpha}^T$ contain the target-domain-specific information and source-domain-specific information, respectively. In detail, a larger (or lower) λ means that the adapter focuses more (or lesser) on the target domain. In this section, we visualize the values of λ during the training process to clearly analyze its contribution. We utilize the ResNet34 and ResNet18 as the teacher network and student network, respectively, to train on the Office-Home dataset.

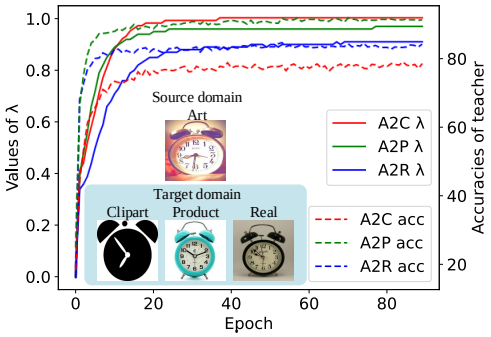

Fig. 2: The learning curves of λ on the Office-Home dataset.

As shown in Fig. 2, λ gradually increases with the performance improvement of the teacher network, suggesting that incorporating target-domain-specific information is beneficial for enhancing performance in the target domain. Meanwhile, we can observe λ with varying values for the teacher network trained in different source-target pairs, and a significant gap between domains (*e.g.*, "Art" and "Clipart") finally results in a large λ. These results show that the λ in our knowledge adapter can adaptively adjust the information for source and target domains according to their domain gaps. Therefore, our method with a learnable λ can eventually achieve

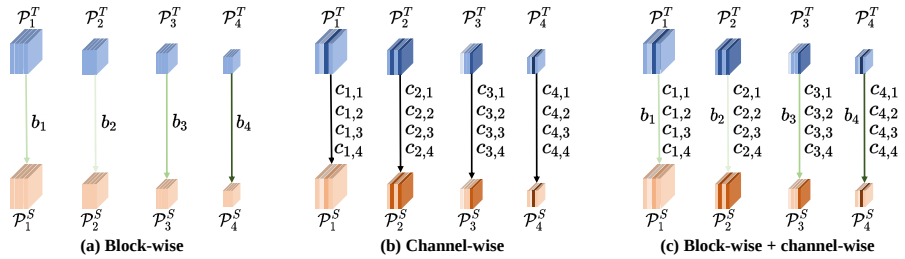

Fig. 3: The diagram of various weighting strategies. (a) Block-wise weighting strategy assigns a weight to the features produced by each block. (b) Channel-wise weighting strategy distributes weight to each feature map within each block. (c) Block-wise + channel-wise weighting strategy assigns a weight to each block as well as to the feature maps within them. In this figure, "b" and "c" represent the weights for specific blocks and feature maps, respectively.

better performance than the cases of λ=0 (81.17%) and λ=1 (81.96%). This also clearly demonstrates the usefulness of such a learnable parameter in balancing domain-specific information.

9 Weighting Strategy

In ablation studies, we employ various weighting strategies to calculate the attention weights for the domain-invariant features of the teacher network and student network. Here, we assume that both the teacher network and student network consist of 4 convolutional blocks, with each block having 4 feature maps. As shown in Fig. 3, there are various domain-invariant feature weighting strategies, including:

1) Block-wise weighting strategy treats the features produced by each block as a whole and assigns a uniform attention weight to the features of each block.

2) Channel-wise weighting strategy considers each feature map in the feature block separately and assigns an independent weight to each feature map.

3) Block-wise + channel-wise weighting strategy assigns an attention weight to each feature block and the feature maps within it.