湘潭大学毕业论文

题　　目：基于对抗学习的域适应方法研究

学　　院：数学与计算科学学院

班　　级：2019 级信息与计算科学 2 班

学　　号：201905755824

姓　　名：米科润

指导教师：成央金教授 宫辰教授

完成日期：2023 年 5 月

摘　要

当前机器学习和深度学习技术已成为社会最重要的热点之一，但其在许多现实情况下面临诸多挑战，例如当训练集与测试集分布存在偏移时模型将无法良好部署，此即域适应问题，而通用域适应设置使问题更加现实：在源域和目标域中同时引入私有类。之前的方法基本都将私有类数据视作有害部分并尽可能降低其影响，这样的处理使得私有类数据包含的潜在价值被忽略了。

针对这样的通用域适应问题，同时为了弥补前人工作缺陷，本文提出了对抗域适应算法，算法基于对抗学习思想，并融合一些半监督学习算法的重要部分，首先在源域上进行有监督学习，并在无先验知识情况下划分两域数据的公共与私有部分以进行对抗性特征对齐，之后将筛选得到的源域与目标域私有类数据分别与目标域公共类数据进行对抗性重用对齐，以达到重新回收利用前者潜在价值的目的，这样的处理有助于目标域的分类。

本文进行了大量的实验以分析对抗域适应算法的有效性，结果表明对抗域适应算法在各个数据集中都展现出了优异的性能，特别是在困难的 Office-Home 数据集与 VisDA2017 数据集中超越了所有的基准比较算法。进一步的，诸如敏感性分析与消融实验等过程都表明了对抗域适应算法的稳健性以及算法各部分的重要性。

关键词：通用域适应；对抗学习；私有数据；重用

Abstract

Machine learning and deep learning technologies have become one of the most important and popular topics in society. However, these methods face many challenges in real-world situations, such as the domain adaptation problem when there is a distribution shift between the training and testing datasets. Moreover, the universal domain adaptation setting makes the problem more realistic by introducing private classes in both the source and target domains. Previous methods have generally considered the private class data as harmful and attempted to minimize their impact, ignoring the potential value contained in these data.

To address this universal domain adaptation problem and improve upon previous methods, this paper proposes an adversarial domain adaptation algorithm based on adversarial learning and incorporating important components of semi-supervised learning algorithms. The algorithm first performs supervised learning on the source domain and then partitions the common and private parts of the data between the two domains without prior knowledge to achieve adversarial feature alignment. The selected private class data from both the source and target domains are then adversarially reused with the target domain's common class data to recover the potential value of the former and improve target domain classification.

This paper conducted extensive experiments to analyze the effectiveness of adversarial domain adaptation algorithms. The results show that the adversarial domain adaptation algorithm exhibits excellent performance on various datasets, especially surpassing all baseline comparison algorithms on the challenging Office-Home dataset and VisDA2017 dataset. Furthermore, processes such as sensitivity analysis and ablation experiments demonstrate the robustness of the adversarial domain adaptation algorithm and the importance of each component of the algorithm.

Key Words: Universal domain adaptation; Adversarial learning; Private data; Reuse

目　　录

1 绪论

1.1 研究背景及意义

二十世纪以来，机器学习和深度学习技术已经在许多人工智能领域取得了重大成功，在分类、回归、聚类、计算机视觉以及自然语言处理问题方面都取得了令人印象深刻的进步。然而许多机器学习方法只有在一个共同的假设下才能很好地工作：训练和测试数据来自相同的特征空间并具有相同的分布。当分布发生变化时，机器学习、深度学习模型往往需要使用新收集的训练数据重建，而在许多现实应用中，重新收集所需的训练数据并重建模型是代价极高甚至难以实现的，所以在源域上学习到的机器学习或深度学习算法若能迁移在目标域上并实现良好的应用将带来巨大帮助，这就形成了域适应问题。

域适应方法的吸引力在于当目标域数据完全未标记（无监督域适应）或具有少量标记样本（半监督域适应）时，都能够实现这样的迁移。进一步的，在图像分类任务方面，基本的域适应问题往往假设标签集在域之间是相同的（闭集域适应）。而通用域适应问题[1]（如图1-1所示）通过囊括开放集域适应[2-3]和部分集域适应[4-5]来放宽这一假设。部分集域适应要求源标签集合包含目标标签集合，而开放集域适应问题其一是在两域中引入私有类，并假设两域之间的公共类在训练阶段已知；其二是去除了源域私有类的数据，使得源标签集是目标标签集的子集。这些工作是在实际域适应问题上取得的宝贵进展。

图 **1-1** 通用域适应背景图

对抗学习同样是一种机器学习方法，源自生成对抗网络[6]（如图1-2所示），它涉及

训练生成器与判别器两个模型，前者尝试生成伪造的数据，后者尝试区分真实的数据和伪造的数据。训练过程中，两个模型相互对抗，生成器希望能够生成伪造的数据使得判别器难以分辨，而判别器希望能够准确地区分真实的数据和伪造的数据。在这个过程中，生成器和判别器互相帮助对方改进，直到生成器能够生成高质量的伪造数据，而判别器也能够准确地区分真实数据和伪造数据。通过这种对抗训练的方式，对抗学习可以用来生成高质量的虚假图像、声音、文本等样本数据。

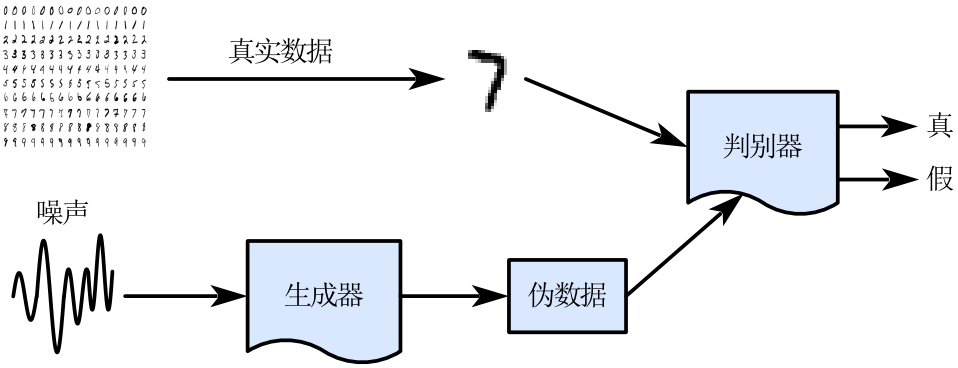


图 **1-2** 生成对抗网络示意图

基于对抗学习的域适应方法近年来在学界愈来愈得到关注，其旨在解决源域和目标域之间的分布差异问题，当源域和目标域之间存在显著的分布差异时，传统的机器学习方法可能会导致泛化性能下降，而对抗域适应的核心思想是利用对抗学习的框架，通过生成对抗样本或对抗特征，使得源域和目标域之间的分布差异最小化，从而提高模型在目标域上的性能。针对域适应问题内在数据分布差异性质，Ganin 等人[7]提出了领域对抗神经网络（DANN），DANN 使用域判别器来区分不同域产生的特征，其准确率刻画两域边缘分布的差异，而特征生成器的目标是欺骗域判别器，从而减少边缘分布的差异；Long 等人[8]提出了条件对抗神经网络（CDAN），CDAN 在训练域判别器的时候使用模型的预测代替条件分布中的标签，让域判别器观察模型的输出，而为了欺骗加强后的域判别器，特征生成器不得不将每个类别的特征分别拉近，使得在类别数多的分类问题上会获得更好的迁移效果；Tsai 等人[9]在实例分割这种像素级别的分类任务领域提出 Adapt-SegMap 算法，Adapt-SegMap 训练域判别器来区分模型的输出是来自源域还是目标域，同时鼓励特征生成器对齐目标域和源域的输出分布，这种方式比在特征空间进行对抗域适应效果更好。

尽管具备上述技巧的对抗域适应方法在许多域适应问题上效果极为显著，但从另一方面来看，对于通用域适应问题，当两域中包含私有类数据时，仅仅将它们区分、识别而后进行筛选、剔除，虽然在训练过程中不会对公共类数据的训练造成任何干扰导致误差，但这样的做法本身体现了域适应模型对更泛化问题的经验缺乏，其次，私有类数据可能对模型性能产生的正向提升往往会被忽视。基于上述考量，重用源域和目标域私有

类中有益的数据对于提高目标域公共类分类精确性是显而易见的，而对抗学习对此依然有独到的技巧。

1.2 本文主要工作

综上，本文考虑针对通用域适应问题，基于对抗学习思想的指导，一方面尽可能减少源域与目标域中公共类数据的特征偏移，另一方面倾向在源域和目标域中重用对公共类数据分类有益的私有类数据，提出一种端到端的对抗域适应网络，进一步的，考虑到域适应问题与半监督学习问题本身数据样本的构成类似（既有标签数据又有无标签数据），最终目标相近，并且某些域适应问题常采用半监督学习算法的变体，故网络考虑采纳相关经典半监督学习算法[10-13]可利用部分，在理论基础上加强了算法的泛化能力，并有望作为一种有效工具，适配各种设置下的图像分类通用域适应问题与数据集。

1.3 论文结构安排

一、绪论部分。首先主要介绍本文研究的背景与理论，研究的思路与框架，实践意义，并提出研究方法和研究的创新点。

二、相关预备理论。将在对抗学习算法、域适应理论基础与对抗域适应方法设计等方面进行系统介绍与理论分析。

三、概括性介绍本文提出的端到端对抗域适应网络，依次从网络结构、损失函数、对抗策略以及算法流程展开分析。

四、分析适应性准则。剖析对抗域适应网络中对抗策略设计参数，以前人工作为基础，详细介绍特征对齐系数与重用系数并解释其作用机理与合理性。

五、实验与分析部分。考虑在经典且具备现实意义的通用域适应数据集（Office-31[14]、Office-Home[15]、VisDA2017[16]等）上进行相关实验并与前人工作进行对比分析以及其他相关结果分析。

六、结论与展望。简要总结本文的一些工作，并对接下来进一步的研究工作给予展望。

2 预备理论

2.1 对抗学习基础知识

对抗学习[6]是机器学习中的一种方法，其基于两个相互对抗的神经网络模型: 捕获数据分布的生成模型 G 和估计样本是来自训练数据还是 G 的概率的判别模型 D。

生成器和判别器来源于两种主要的模型：生成模型和判别模型[17]。生成模型是由数据直接学习联合概率分布 $P(X,Y)$，然后求出条件概率分布 $P(Y|X)$ 作为预测的模型，即生成模型 $P(Y|X)=\frac{P(X,Y)}{P(X)}$。判别模型则是指由数据直接学习决策函数 $f(X)$ 或者条件概率分布 $P(Y|X)$ 作为预测的模型。

对抗学习中最常用的方法是对抗训练。在对抗训练中，生成器和判别器相互对抗，每次迭代时，生成器生成一些假样本，并将其传递给判别器，判别器则需要判断这些样本是否真实。生成器的目标是欺骗判别器，使其无法准确判断样本的真假，而判别器的目标是尽可能准确地判断样本的真假。

在对抗训练中，对抗损失函数起着关键的作用，它可以衡量生成器和判别器之间的对抗程度。通常使用交叉熵损失函数作为对抗损失函数，其数学形式如下：

$$\min_G \max_D V(D,G)=\mathbb{E}_{\boldsymbol{x}\sim P_{data}}[logD(\boldsymbol{x})]+\mathbb{E}_{\boldsymbol{z}\sim P_z}[log(1-D(G(\boldsymbol{z}))], \tag{2-1}$$

其中，$D(\boldsymbol{x})$ 表示判别器判断 $\boldsymbol{x}$ 是真实数据的概率，$G(\boldsymbol{z})$ 表示生成器生成的假样本，$\boldsymbol{z}$ 是从先验分布 P_z 中随机采样的噪声。

生成器的训练目标是最小化对抗损失函数，使得生成器能够生成逼真的样本。生成器的训练过程可以分为两个步骤：

（1）生成假样本。从先验分布 P_z 中采样噪声 $\boldsymbol{z}$，并将其输入生成器中，生成器将噪声 $\boldsymbol{z}$ 转换成逼真的样本 $G(\boldsymbol{z})$。

（2）欺骗判别器。生成器的目标是欺骗判别器，使其无法准确判断生成样本的真假，即期望生成器生成的假图越被误判为真图。因此，生成器的损失函数可以表示为（给定判别器 D，找到使 V 最小化的生成器 G）：

$$\min_G V(D,G)=\mathbb{E}_{\boldsymbol{z}\sim P_z}[1-logD(G(\boldsymbol{z}))]。 \tag{2-2}$$

判别器的训练目标是最大化对抗损失函数，使得判别器能够准确判断样本的真假。判别器的训练过程可以分为两个步骤：

（1）对真实样本进行判断。从真实数据分布 P_{data} 中随机采样一个样本 $\boldsymbol{x}$，并将其输入判别器中，判别器将 $\boldsymbol{x}$ 判断为真实数据的概率表示为 $D(\boldsymbol{x})$，此时希望判别器的输出越大越好。

（2）对生成样本进行判断。从生成器中生成一个样本 $G(\boldsymbol{z})$，并将其输入判别器中，判别器将 $G(\boldsymbol{z})$ 判断为假数据的概率表示为 $D(G(\boldsymbol{z}))$，此时期望生成器生成的假图越被判别器判定为假图。

判别器的损失函数可以表示为：

$$\max_{D} V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim P_{data}}[logD(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim P_z}[log(1-D(G(\boldsymbol{z}))]。 \tag{2-3}$$

生成器网络使用 RELU[18]和 Sigmoid 激活函数的混合，而判别器网络使用 Maxout[19]激活函数。在训练判别器网络时应用了 Dropout[20]。生成器和判别器的训练过程可以通过反向传播算法进行，即计算损失函数的梯度，并更新判别器中的参数。

2.2 域适应理论背景

域适应是机器学习中一个重要的研究方向，它的主要目的是将已经学习过的知识和经验迁移到新的领域或任务上，从而提高机器学习模型的泛化能力。在现实应用中，往往需要处理的是不同领域之间的数据差异和模型迁移问题，如计算机视觉中的跨领域图像分类、自然语言处理中的跨语言情感分类等。在这些问题中，往往需要将源域中已经标注好的数据和目标域中的数据进行对齐，从而让模型能够更好地适应新的数据。

以二分类任务上的域适应问题为例。首先将输入集记为 $\mathcal{X}$，标注空间是一个概率空间，二分类问题的标注集是 $[0,1]$ 区间。在 $\mathcal{X}$ 上，定义一个域是由 $(\mathcal{D}, f)$ 所构成的一个对。其中，$\mathcal{D}$ 是一个分布函数，刻画输入集 $\mathcal{X}$ 上每个样本的出现概率，而 f 是一个标签函数，$f:\mathcal{X}\to[0,1]$，表示从输入集到标注集的一个映射，其中 $f(\boldsymbol{x})\in[0,1], \boldsymbol{x}\in\mathcal{X}$ 可以是 $[0,1]$ 区间内的任意实数值，表示样本 $\boldsymbol{x}$ 的标签为 1 的概率。一般而言，用字母 S,T 区分源域和目标域，而两个域的数学表示分别记为 $(\mathcal{D}_S, f_S)$ 和 $(\mathcal{D}_T, f_T)$。

给定两个域 $(\mathcal{D}_S, f_S)$ 与 $(\mathcal{D}_T, f_T)$ 后，希望学习一个函数 $h:\mathcal{X}\to\{0,1\}$，这个目标函数直接预测输入所对应的标签，通常把这个模型习得的函数 h 称作是一个假设，将 h 与 f 的差距[21]记作：

$$\epsilon(h,f;\mathcal{D}) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}|h(\boldsymbol{x}) - f(\boldsymbol{x})|。 \tag{2-4}$$

为简化标记，在源域上记 $\epsilon_S(h) = \epsilon_S(h, f_S; \mathcal{D}_S)$，如果只有有限的样本，则采用这些样本对公式 (2-4) 的期望进行经验估计，记经验估计的结果为 $\hat{\epsilon}_S(h)$，同理可以得到 $\epsilon_T(h), \hat{\epsilon}_T(h)$。

给定两个域 $(\mathcal{D}_S, f_S)$ 和 $(\mathcal{D}_T, f_T)$，若采用基于 L_1 范数的变分散度来刻画它们的距

离，则有：

$$d_1(\mathcal{D}_S, \mathcal{D}_T) = 2 \sup_{B \subset \mathcal{X}} |\Pr_{\mathcal{D}_S}(B) - \Pr_{\mathcal{D}_T}(B)|, \tag{2-5}$$

其中，$\Pr(B)$ 表示集合 B 的概率。此距离可被理解为：寻找两个不同数据分布上分布差距最大的子集，将它们概率的差值绝对值作为距离。

但在实际使用过程中，该距离又有诸多不便。首先如果分布函数是离散的，那么寻找公式 (2-5) 的上确界就是一个 NP-Hard 问题；其次如果分布函数是连续的，那么公式 (2-5) 中取得上确界的 B 往往是由多个小区间联立拼凑而成，这实际上是无法计算的，因此该距离的计算意义不大，但其却具有极高的实际用处。求解域适应是希望在源域 $\mathcal{D}_S$ 上训练的模型 h 能泛化到目标域 $\mathcal{D}_T$ 上，即希望 $\epsilon_S(h)$ 很小的时候，$\epsilon_T(h)$ 也比较小，此时公式 (2-5) 中刻画的距离就可推导出：

$$\begin{aligned} \epsilon_T(h) \leq \epsilon_S(h) + d_1(\mathcal{D}_S, \mathcal{D}_T) \\ + \min\{\mathbb{E}_{\mathcal{D}_S}|f_S(\boldsymbol{x}) - f_T(\boldsymbol{x})|, \mathbb{E}_{\mathcal{D}_T}|f_S(\boldsymbol{x}) - f_T(\boldsymbol{x})|\}, \end{aligned} \tag{2-6}$$

此公式表示在 $\mathcal{D}_S$ 上训练的 h ，当 $\epsilon_S(h)$ 很小的时候，$\epsilon_T(h)$ 是否也比较小，这取决于两个数据集的距离以及两个数据集的标签函数 f_S, f_T 是否一致。

但是公式 (2-6) 所展现出对于目标域误差 $\epsilon_T(h)$ 的上确界比较宽松。一个很典型的想法是，模型只需要关注于对学到的假设 h 而言，两个数据集的距离不会过远就足够。现实任务中针对域适应问题，在目标任务上，两个数据集应该具有一定相似度，那么相比较考虑整个数据分布的距离，仅考虑在目标假设 h 上的距离就显得更加合适，在这个思想下，则有 $\mathcal{H}$ 距离：

$$d_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) = 2 \sup_{h \in \mathcal{H}} |\Pr_{\mathcal{D}_S}(I(h)) - \Pr_{\mathcal{D}_T}(I(h))|, \tag{2-7}$$

其中，$\mathcal{H}$ 是模型训练出的所有可能假设 h 的集合，可以将其视作模型容量，$I(h) = \{\boldsymbol{x} : h(\boldsymbol{x}) = 1, \boldsymbol{x} \in \mathcal{X}\}$ 是 $\mathcal{X}$ 的一个子集，但是与 h 有关。通过公式 (2-7) 的定义，数据集的距离与目标假设构成了联系。而公式 (2-7) 的距离进一步可融入公式 (2-6)。

由前述理论，域适应主要目的是想通过一系列手段，使得优化 $\epsilon_S(h)$ 所得到的模型 h 同样能在 $\epsilon_T(h)$ 上得到良好的表现。为此，需要建立这两者的联系，以得到以下形式的上界：

$$\epsilon_T(h) \leq \epsilon_S(h) + A + B + C。 \tag{2-8}$$

上文已经通过公式 (2-6) 给出了一个重要的上界，其表明 $\epsilon_T(h)$ 的上确界与优化 $\epsilon_S(h)$ 有关，同时也与数据集之间的差异，以及与数据集的标签函数 f_S, f_T 之间的差异有关。但是公式 (2-6) 所刻画的上界固然具有良好的分析形式，却并不实用，其中涉及

到 $d_1(\mathcal{D}_S, \mathcal{D}_T)$ 和标签函数 f_S, f_T 这几项更是无法推导得到。因此需要一些能够用于计算的上界来提供实践指导。

由前已知距离 $d_1(\mathcal{D}_S, \mathcal{D}_T)$ 是无法计算的，而针对某一具体的任务，可用 $d_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)$ 代替。显而易见 $d_1(\mathcal{D}_S, \mathcal{D}_T)$ 是 $d_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)$ 的一个上界，因此引入 $d_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)$ 必然可以缩小域适应的上界。

关于 $d_{\mathcal{H}}$ 的计算，若对源域与目标域的数据分布分别采样相同的个数 m 个，构成样本集 U_S, U_T，那么 $d_{\mathcal{H}}$ 的经验表达形式为：

$$\hat{d}_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) = 2 \sup_{h \in \mathcal{H}} |\frac{1}{m} \sum_{i=1}^{m} \mathbf{I}[h(\boldsymbol{x_i}) = 1] - \frac{1}{m} \sum_{i=1}^{m} \mathbf{I}[h(\boldsymbol{x_i}) = 1]|。 \tag{2-9}$$

但是经验表达形式仍然涉及一个上确界问题，需寻找到合适的 h 从而进行计算。首先考虑在某个假设空间上计算 $\mathcal{D}_S, \mathcal{D}_T$ 的距离。已知 $d_1(\mathcal{D}_S, \mathcal{D}_T)$ 距离的本质是在分布函数上寻找两个数据集“差异最大处”，那么对于某一类假设空间（分类器空间）$\mathcal{H}$，如果这个空间的分类器完全无法区分数据是来源于源域 $\mathcal{D}_S$ 还是目标域 $\mathcal{D}_T$，说明这个假设空间对于两个数据集的距离“很小”。依据此思路，可给 $\mathcal{D}_S$ 赋予标签 0，$\mathcal{D}_T$ 赋予标签 1，然后在整个假设空间上寻找一个分类器，使其尽量区分输入的数据是来自于源域分布 $\mathcal{D}_S$ 还是来自于目标域分布 $\mathcal{D}_T$。由于问题本质是二分类，可设交叉熵为损失函数，因此分类器易于寻找，接着可用其计算距离，那么就涉及到：

$$\hat{d}_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) = 2(1 - \min_{h \in \mathcal{H}}[\frac{1}{m} \sum_{\boldsymbol{x}: h(\boldsymbol{x})=0} \mathbf{I}[\boldsymbol{x} \in U_S] - \frac{1}{m} \sum_{\boldsymbol{x}: h(\boldsymbol{x})=1} \mathbf{I}[\boldsymbol{x} \in U_T]]), \tag{2-10}$$

式中等式成立的条件是 U_S, U_T 的采样数目相同（实际中一般选择每个批量采样的数目相同）。

根据公式 (2-10)，可通过交叉熵损失来训练分类器，接着使用最终分类器计算 $h(\boldsymbol{x})$，从而计算公式 (2-10)。一般而言，交叉熵损失越小，则说明 $\hat{d}_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)$ 越小，即两个数据集在整个假设空间（分类器空间）上的差距较小。

以上计算都是在 N 个采样样本上所得的经验估计，还需考虑用经验误差 $\hat{d}$ 度量分布距离 d 所产生的偏差：假设 $\mathcal{H}$ 是所有基于从输入集 $\mathcal{X}$ 到标签集 $\{0, 1\}$ 的分类器 h 的假设空间，且采用的分类器模型具有 VC 维 d，那么用采样数为 m 的经验距离 $\hat{d}_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)$ 估计 $d_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)$ 的偏差满足对于任意的 $\delta \in (0, 1)$，以下误差界在至少 $1 - \delta$ 的概率下成立：

$$d_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) \leq \hat{d}_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) + 4\sqrt{\frac{d \log(2m) + \log(\frac{2}{\delta})}{m}}。 \tag{2-11}$$

关于利用可计算的距离 $d_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)$ 刻画 $\epsilon_S(h), \epsilon_T(h)$ 的关系，首先引入理想化联合

假设概念，其被定义为能最小化源域与目标域的联合预测错误的假设：

$$h^* = \arg\min_{h\in\mathcal{H}} \epsilon_S(h) + \epsilon_T(h)。 \tag{2-12}$$

同时，记在理想化联合假设 h^* 的作用下，联合预测错误的值为：

$$\lambda = \epsilon_S(h^*) + \epsilon_T(h^*)。 \tag{2-13}$$

给定两个域 $(\mathcal{D}_S, f_S)$ 和 $(\mathcal{D}_T, f_T)$，以及假设空间 $\mathcal{H}$ 后，λ 随之也被确定，其应该是一个常数，而在域适应领域内往往假设 λ 是一个比较小的值。为联系两个域的误差关系，还涉及一个基于异或思想的对称差异假设空间定义 $\mathcal{H}\Delta\mathcal{H}$[22]（$\oplus$ 是异或操作）：

$$g \in \mathcal{H}\Delta\mathcal{H} \Leftrightarrow \exists h, h' \in \mathcal{H}, g(\boldsymbol{x}) = h(\boldsymbol{x}) \oplus h'(\boldsymbol{x})。 \tag{2-14}$$

简单而言，对于任意 $g \in \mathcal{H}\Delta\mathcal{H}$，它的映射结果代表了原假设空间 $\mathcal{H}$ 中两个假设的“不一致”。

由于对任意两个假设 $h, h' \in \mathcal{H}$，有：

$$\begin{aligned} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) &= 2 \sup_{h,h'\in\mathcal{H}} |\mathbb{E}_{\mathcal{D}_S}|h(\boldsymbol{x}) - h'(\boldsymbol{x})| - \mathbb{E}_{\mathcal{D}_T}|h(\boldsymbol{x}) - h'(\boldsymbol{x})|| \\ &= 2 \sup |\epsilon_S(h, h') - \epsilon_T(h, h')|。 \end{aligned} \tag{2-15}$$

则可得：

$$|\epsilon_S(h, h') - \epsilon_T(h, h')| \leq \frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)。 \tag{2-16}$$

进一步可得两个距离函数 $d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T), d_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)$ 的关系如下：

$$\frac{1}{2} d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) \leq d_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)。 \tag{2-17}$$

因此，有以下关系式：

$$\epsilon_T(h) \leq \epsilon_S(h) + \lambda + d_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)。 \tag{2-18}$$

在实际计算中，往往优化的是 $\hat{\epsilon}_S(h)$，并用前文所述的 $\hat{d}_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)$ 进行距离估计。结合上文并基于 VC 维理论，可得对于任意的 $\delta \in (0, 1)$，如果给定的有标注的数据集采样为 m'，下述上界在至少 $1 - \delta$ 的概率下成立：

$$\epsilon_S(h) \leq \hat{\epsilon}_S(h) + \sqrt{\frac{4}{m'}(d \log \frac{2em'}{d} + \log \frac{4}{\delta})}。 \tag{2-19}$$

综上所述，最终可得讨论最广泛的上界：

$$\epsilon_T(h) \leq \hat{\epsilon}_S(h) + \lambda + \hat{d}_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) + \sqrt{\frac{4}{m'}(d\log\frac{2em'}{d} + \log\frac{4}{\delta})} + 4\sqrt{\frac{d\log(2m) + \log(\frac{2}{\delta})}{m}}。 \tag{2-20}$$

2.3 对抗域适应算法设计

域适应问题属于直推式迁移学习范畴（Transductive Transfer Learning）[23]，主要有实例迁移和特征表示迁移两种方法，而对抗域适应方法属于后者，其目的是找到一种“好”的特征表示，以减少源域和目标域之间的差异以及模型的误差。

从公式 (2-20) 可知，对于在源域训练好的假设 h，其在目标域的表现 $\epsilon_T(h)$ 取决于：源域的验证表现 $\hat{\epsilon}_S(h)$；假设空间的合理程度 λ；域之间的距离 $\hat{d}_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)$。其中，假设空间 $\mathcal{H}$ 一旦给定，λ 就固定下来。那么，使域之间的距离变得尽可能小则成为了域适应的关键点，一个很自然的想法是通过特征空间过渡，如果设计出映射能将源域与目标域都映射到一个特征空间，在这个特征空间上，源域与目标域无法区分，最后在特征空间上构造分类器就能有效解决域适应问题。

对抗域适应领域的最经典方法是 DANN[7] 算法，其框架如图2-1所示。

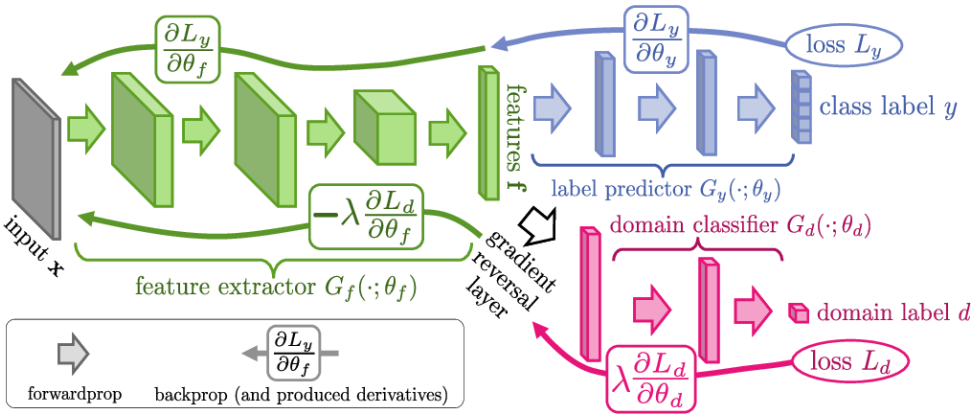


图 **2-1 DANN** 示意图

由图2-1，输入 $\boldsymbol{x}$ 后，模型先通过一个特征提取层（绿色）将输入映射到特征空间得到特征。之后模型将特征输入分类器（紫色），用于目标任务分类，这一部分是用于训练模型的 $\hat{\epsilon}_S(h)$。此外，并行的红色结构也是一个分类器，其用来区分输入的特征属于源域还是属于目标域，简单而言，红色的部分用于计算 $\hat{d}_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)$。在训练过程中，希望红色的分类器足够强，从而得到准确的 $\hat{d}_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T)$，同时又希望特征空间上源域特征与目标域特征尽量接近，从而混淆红色分类器，这就形成一个对抗的思想。通过这样的训练过程，源域与目标域” 差别不大” 的特征空间即可得到。

3 对抗域适应算法

3.1 问题设置

在通用域适应问题中，训练时，源域 $\mathcal{D}_s = \{(\boldsymbol{x}_i^s, y_i^s)\}$ 包含 n_s 个标签实例，其中公共类数据由 $\mathcal{D}_s^{co}$ 表示，私有类数据由 $\mathcal{D}_s^{pr}$ 表示。目标域 $\mathcal{D}_t = \{(\boldsymbol{x}_j^t)\}$ 包含 n_t 个标签实例，其中公共类数据由 $\mathcal{D}_t^{co}$ 表示，私有类数据由 $\mathcal{D}_t^{pr}$ 表示。

源数据从分布 p 采样，而目标数据从分布 q 采样。使用 $\mathcal{C}_s$ 表示源域的标签集，$\mathcal{C}_t$ 表示目标域的标签集。$\mathcal{C} = \mathcal{C}_s \cap \mathcal{C}_t$ 是两域共享的公共标签集，$\overline{\mathcal{C}_s} = \mathcal{C}_s \setminus \mathcal{C}$ 和 $\overline{\mathcal{C}_t} = \mathcal{C}_t \setminus \mathcal{C}$ 分别表示源域和目标域专用的标签集。$p_{\mathcal{C}_s}$ 和 $p_{\mathcal{C}}$ 分别用于表示具有 $\mathcal{C}_s$ 和 $\mathcal{C}$ 中的标签的源域数据的分布，以及 $q_{\mathcal{C}_t}$ 和 $q_{\mathcal{C}}$ 分别用于具有 $\mathcal{C}_t$ 和 $\mathcal{C}$ 中的标签的目标域数据分布，如图3-1所示。目标数据是完全未标记的，目标标签集（在训练时不可见）仅用于定义通用域适应问题。

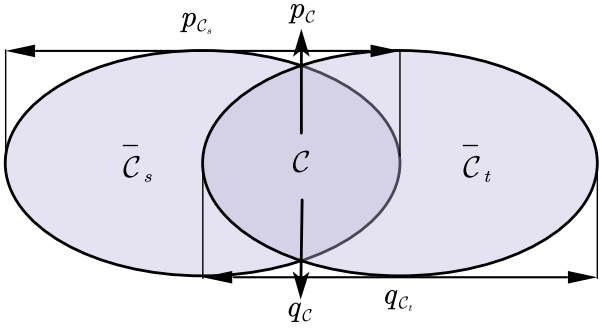


图 **3-1** 标签集与分布示意图

本文提出的对抗域适应网络，目标是：

1. 在没有两域标签重叠比例 $\frac{\mathcal{C}}{\mathcal{C}_s \cup \mathcal{C}_t}$ 这一先验知识的前提下，有效地识别来自源域和目标域的公共类的数据，以对抗性地消除公共类源域已标记数据和目标域未标记数据之间的特征分布不匹配；

2. 学习分类模型 f，以最小化目标域公共类数据的目标风险，即 $\min \mathbb{E}_{(\boldsymbol{x},y)\sim q_{\mathcal{C}}}[f(\boldsymbol{x}) \neq y]$；

3. 能适当地重利用 $\mathcal{D}_s^{pr}$ 和 $\mathcal{D}_t^{pr}$ 中有益数据，以帮助目标域上分类模型 f 对未知标签的图像进行分类的能力。

3.2 对抗域适应网络结构

本文提出的对抗域适应网络算法的框架流程如图3-2所示。

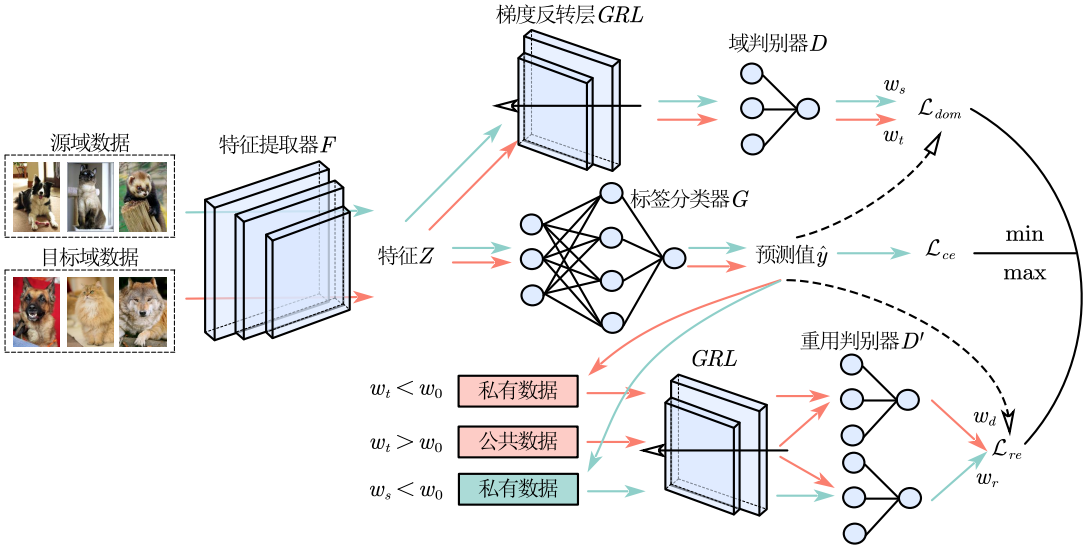


图 **3-2** 对抗域适应网络架构图

由图3-2可知，对抗域适应网络的框架结构由特征提取器 F、对抗性域判别器 D、对抗性重用判别器 D_1', D_2'、标签分类器 G 以及梯度反转层 GRL 组成。

关于特征提取器 F，其接收来自任一域的输入图片 $\boldsymbol{x}$，经过卷积层，输出图片特征 $Z = F(\boldsymbol{x})$。

关于标签分类器 G，当输入的图片来自源域时，其接收特征提取器 F 输出的图片特征 Z，之后经过全连接层，输出分类信息 $\hat{y} = G(Z)$。

关于具有对抗性的域判别器 D，其接收来自源域与目标域的特征 Z，旨在对抗性地匹配落在公共类中的源域数据和目标域数据的特征分布。

关于具有对抗性的重用判别器 D_1', D_2'，其接收来自源域和目标域的数据，但在输入前已被成功分离为私有数据与公共数据两类，两个重用判别器 D' 的作用是在源域和目标域的私有类数据中找到有益部分，将它们分别对抗性的匹配到目标域公共类数据 $\mathcal{D}_t^{co}$ 空间中，这样的处理使得它们包含的潜在有益信息可以被充分地提取出来提高目标域公共类数据 $\mathcal{D}_t^{co}$ 上模型的分类性能。

注意，此时域判别器 D 与重用判别器 D_1', D_2' 共用同一个特征提取器 F，即在消除两域公共类数据差异的同时，源域和目标域私有类可重用数据实际匹配到的是潜在的差异较小的公共类特征空间，这样的处理使得即便目标域公共类数据 $\mathcal{D}_t^{co}$ 占比小也不会影响训练效果，因为潜在的特征空间由于源域公共类数据 $\mathcal{D}_s^{co}$ 引入的关系，包含的分布信息已经足够丰富了。

关于梯度反转层 GRL，考虑到训练过程中反向传播过程将损失逐层向后传递，每层网络会根据回传的误差计算梯度，进而更新本层网络的参数，而 GRL 两侧特征提取

器 F 与判别器（D, D_1', D_2'）训练目标相反，则在更新特征提取器 F 的参数时，经判别器产生的误差关于 F 的导数需要乘以某个负数以实现对抗目的，因此在正向传播过程中，GRL 可被视作单位转换函数 $f(\boldsymbol{x}) = \boldsymbol{x}$，而在反向传播期间，$GRL$ 从后续层获取梯度并乘以一个负数，再将其传递到前一层。

3.3 对抗域适应损失函数

图3-2的网络由于其构成最终将输出三个损失项：监督损失 $\mathcal{L}_{ce}$，特征对齐损失 $\mathcal{L}_{dom}$ 以及数据重用损失 $\mathcal{L}_{re}$，而网络总体损失函数由三者组成：

$$\begin{aligned}\min_{\theta_F,\theta_G}\max_{\theta_D,\theta_{D_{1,2}'}} & \overbrace{\mathbb{E}_{(\boldsymbol{x}_i,y_i)\sim p_{\mathcal{C}_s}}\mathcal{L}_{ce}\left(\boldsymbol{x}_i,y_i;\theta_F,\theta_G\right)}^{\text{监督损失}} \\ & + \overbrace{\lambda\cdot\mathbb{E}_{\boldsymbol{x}_i\sim p_{\mathcal{C}_s},\boldsymbol{x}_j\sim q_{\mathcal{C}_t}} w_s(\boldsymbol{x}_i)\cdot w_t(\boldsymbol{x}_j)\cdot\mathcal{L}_{dom}(\boldsymbol{x}_i,\boldsymbol{x}_j;\theta_F,\theta_D)}^{\text{特征对齐损失}} \\ & + \overbrace{\tau\cdot\mathbb{E}_{\boldsymbol{x}_i\sim p_{\mathcal{C}_s},\boldsymbol{x}_j\sim q_{\mathcal{C}_t}} w_d(\boldsymbol{x}_j)\cdot w_r(\boldsymbol{x}_i)\cdot\mathcal{L}_{re}(\boldsymbol{x}_i;\boldsymbol{x}_j;\theta_F,\theta_{D_{1,2}'})}^{\text{数据重用损失}},\end{aligned} \tag{3-1}$$

其中 $\theta_F, \theta_C, \theta_D, \theta'_{D'_{1,2}}$ 分别是特征提取器 F，标签分类器 G，对抗性域判别器 D，对抗性重用判别器 D_1', D_2' 的参数。

具体而言，损失函数 (3-1) 中：

第一项为监督损失项，$\mathcal{L}_{ce}(\cdot)$ 表示采用标准交叉熵损失，以比较每个源域标记数据 $\boldsymbol{x}_i$ 经过特征提取器 F 与标签分类器 G 后得到的预测 $\hat{y} = G(F(\boldsymbol{x}_i))$ 与真实标签 y_i 的误差。其正式定义为：

$$\mathbb{E}_{\boldsymbol{x}_i\sim p_{\mathcal{C}_s}}\mathcal{L}_{ce}\left(\boldsymbol{x}_i,y_i;\theta_F,\theta_G\right) = -\mathbb{E}_{\boldsymbol{x}_i\sim p_{\mathcal{C}_s}}\sum_{c=1}^{|\mathcal{C}_s|} y_{ic} log(G(F(\boldsymbol{x}_i))), \tag{3-2}$$

式中，$|\mathcal{C}_s|$ 表示类别的数量；y_{ic} 表示符号函数，若样本 $\boldsymbol{x}_i$ 的真实类别等于 c 则取 1，否则取 0；$G(F(\boldsymbol{x}_i))$ 表示样本 $\boldsymbol{x}_i$ 属于类别 c 的预测概率。

第二项为特征对齐损失项，引入了对抗性学习损失 $\mathcal{L}_{dom}(\cdot)$ 对来自源域和目标域的公共类数据进行特征对齐，这里通过两个适应性准则系数 w_s 和 w_t 来检测公共类数据，将在第4章详细说明。其正式定义为：

$$\begin{aligned}&\mathbb{E}_{\boldsymbol{x}_i\sim p_{\mathcal{C}_s},\boldsymbol{x}_j\sim q_{\mathcal{C}_t}} w_s(\boldsymbol{x}_i)\cdot w_t(\boldsymbol{x}_j)\cdot\mathcal{L}_{dom}(\boldsymbol{x}_i,\boldsymbol{x}_j;\theta_F,\theta_D) \\ =&\mathbb{E}_{\boldsymbol{x}_i\sim p_{\mathcal{C}_s}} w_s(\boldsymbol{x}_i) log(1 - D(F(\boldsymbol{x}_i))) \\ &+ \mathbb{E}_{\boldsymbol{x}_j\sim q_{\mathcal{C}_t}} w_t(\boldsymbol{x}_j) log D(F(\boldsymbol{x}_j))。\end{aligned} \tag{3-3}$$

结合损失函数 (3-1) 中前部极大极小化对抗形式（$\min_{\theta_F}\max_{\theta_D}$），公式 (3-3) 中右端表示：

首先固定特征提取器 F，若输入数据来自目标域公共类 $\mathcal{D}_t^{co}$，希望训练域判别器 D 使得输出 $logD(F(\boldsymbol{x}_i))$ 最大化即 $D(F(\boldsymbol{x}_i))$ 最大化；而若输入数据来自源域公共类 $\mathcal{D}_s^{co}$，希望训练域判别器 D 使得输出 $log(1-D(F(\boldsymbol{x}_j)))$ 最大化即 $D(F(\boldsymbol{x}_j))$ 最小化，这样能成功的判别输入数据的来源。

其次固定域判别器 D，若输入数据来自目标域公共类 $\mathcal{D}_t^{co}$，旨在训练特征提取器 F 以最小化输出 $D(F(\boldsymbol{x}_i))$，若输入数据来自源域公共类 $\mathcal{D}_s^{co}$，旨在训练特征提取器 F 以最大化输出 $D(F(\boldsymbol{x}_j))$，这样就能达到混淆域判别器 D 的目的以体现对抗学习思想。

与域判别器 D 相对应的，上述对抗中特征提取器 F 可被视为生成器，而这样的对抗学习过程就使得源域与目标域公共类数据对齐到两域差别不大的特征空间中。

第三项为重用数据检测项，根据 w_s, w_t 检测到源域和目标域私有类数据后，并不会单纯地丢弃，相反本文认为不应忽略其中的有益数据。

具体来说，希望在 $\mathcal{D}_s^{pr}$ 和 $\mathcal{D}_t^{pr}$ 中利用适应性准则系数 w_r, w_d（同样在第4章详细说明）寻找可重用的数据，再将它们匹配到目标域公共类数据 $\mathcal{D}_t^{co}$ 的空间，以便提取潜在信息以帮助目标域的训练分类性能。为此，这里同样利用对抗性损失 $\mathcal{L}_{re}(\cdot)$ 来减小目标域公共类数据与私有类数据的差异。其正式定义为：

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{x}_i\sim p_{\mathcal{C}_s},\boldsymbol{x}_j\sim q_{\mathcal{C}_t}} w_d(\boldsymbol{x}_j)\cdot w_r(\boldsymbol{x}_i)\cdot \mathcal{L}_{re}(\boldsymbol{x}_i;\boldsymbol{x}_j;\theta_F,\theta_{D'_{1,2}}) \\
&\quad =\mathbb{E}_{\boldsymbol{x}_j\sim q_{\mathcal{C}}} logD'_1(F(\boldsymbol{x}_j)) + \mathbb{E}_{\boldsymbol{x}_j\sim q_{\mathcal{C}}} logD'_2(F(\boldsymbol{x}_j)) \\
&\qquad + \mathbb{E}_{\boldsymbol{x}_j\sim q_{\overline{\mathcal{C}}_t}} w_d(\boldsymbol{x}_j) log(1-D'_1(F(\boldsymbol{x}_j))) \\
&\qquad + \mathbb{E}_{\boldsymbol{x}_i\sim p_{\overline{\mathcal{C}}_s}} w_r(\boldsymbol{x}_i) log(1-D'_2(F(\boldsymbol{x}_i)))。
\end{aligned} \tag{3-4}$$

类似地结合损失函数 (3-1) 中前部极大极小化对抗形式（$\min_{\theta_F}\max_{\theta_{D'_{1,2}}}$），公式 (3-4) 中右端表示重用判别器 D'_1, D'_2 旨在判别输入的数据来源是目标域公共类还是源域或目标域私有类可重用部分，而特征提取器 F 被视作生成器，旨在混淆重用判别器 D'_1, D'_2，这样的对抗学习过程使得目标域公共类数据与源域或目标域私有类可重用数据匹配到相近特征空间中。

另外，损失函数 (3-1) 中参数 λ 和 τ 是控制上述监督损失、特征对齐损失以及数据重用损失三项相对权重的非负系数。

3.4 对抗域适应推理过程

推理阶段过程如图3-3所示，输入目标域测试数据，经过特征提取器 F 与标签分类器 G，计算得到其分类预测 $\hat{y}(\boldsymbol{x})$，进一步结合域判别器 D 可计算适应性准则系数 $w_t(\boldsymbol{x})$。

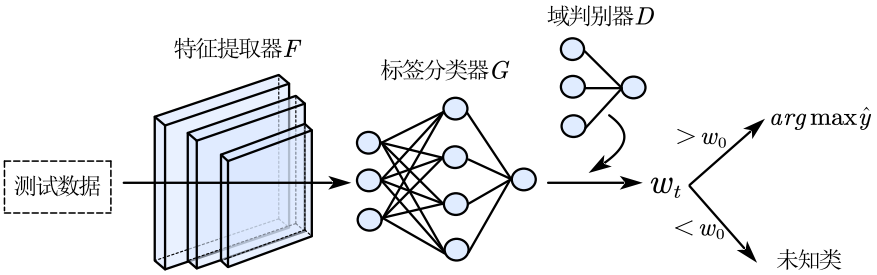


图 **3-3** 对抗域适应网络推理过程图

在给定阈值 w_0 条件下，根据 w_t 与 w_0 的大小关系最终将得到预测类别 y：

$$y = \begin{cases} \text{argmax}(\hat{\boldsymbol{y}}) & w_t \geq w_0 \\ \text{未知类} & w_t < w_0 \end{cases}, \tag{3-5}$$

此公式表示要么将目标域测试数据分类为公共类标签集 $\mathcal{C}$ 中的类别之一，要么将其分类（又称拒绝）到统一的“未知”类。

4 适应性准则

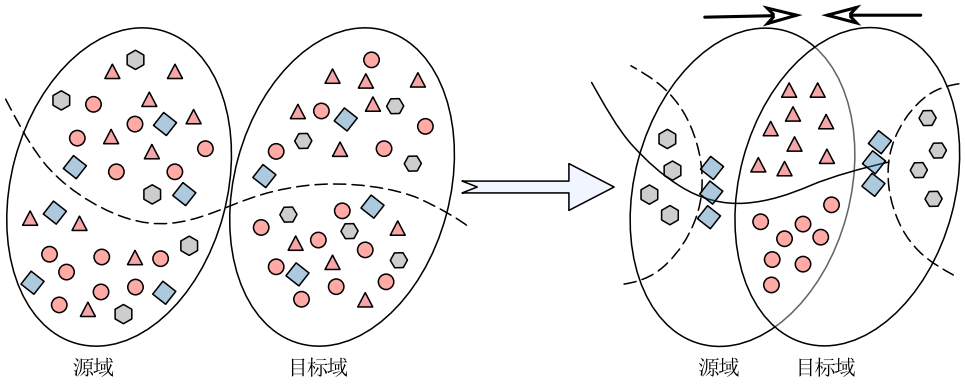


图 **4-1** 适应性准则作用机理图

第3章中提及的适应性准则系数 w_s, w_t, w_d, w_r 的作用机理如图4-1所示，任务以二分类为例，红色三角形与红色圆形代表源域与目标域公共类数据的两个类，灰色五边形与灰色六边形代表两域的私有类数据，而蓝色的四边形代表源域或目标域可重用的私有类数据。

可以发现，相较于图中左侧源域与目标域初始数据集，第3章提出的对抗域适应网络一方面利用适应性准则系数 w_s, w_t 尽可能对齐两域公共类数据分布，一方面利用适应性准则系数 w_d, w_r 尽可能挖掘可重用有益数据以帮助目标域分类任务。这在图中右侧展示：红色数据在“交集”被分类，灰色数据被尽可能排斥，蓝色数据被接纳重用。

适应性准则系数 w_s, w_t, w_d, w_r 是针对样本水平的，即对源域和目标域中的数据点进行加权（$w_* = w_*(\boldsymbol{x})$），实现上述作用机理。

4.1 特征对齐检测

特征对齐检测旨在正确区分属于公共类 $\mathcal{C}$ 和私有类 $\overline{\mathcal{C}_s} \cup \overline{\mathcal{C}_t}$ 的训练数据。为了达到这样的目的，本文对源域和目标域数据分别构造两个特征对齐检测系数 $w_s(\cdot)$ 和 $w_t(\cdot)$，并且系数应满足以下不等式：

$$\begin{aligned} \mathbb{E}_{\boldsymbol{x}\sim p_{\mathcal{C}}} w_s(\boldsymbol{x}) > \mathbb{E}_{\boldsymbol{x}\sim p_{\overline{\mathcal{C}}_s}} w_s(\boldsymbol{x}), \\ \mathbb{E}_{\boldsymbol{x}\sim q_{\mathcal{C}}} w_t(\boldsymbol{x}) > \mathbb{E}_{\boldsymbol{x}\sim q_{\overline{\mathcal{C}}_t}} w_t(\boldsymbol{x})。 \end{aligned} \tag{4-1}$$

上述不等式应尽量在一定范围内保持，以获得更好的检测性能。此处采用两个概念，即下文中的域相似度和标签预测偏移来构造 w_s 和 w_t 。

4.1.1 域相似度

域相似度用于衡量某个输入图像是否属于源域或目标域，通用适应网络（UAN）[1]训练了一个非对抗性判别器，通过最小化交叉熵损失，预测来自源域的数据为 0，来自目标域的数据为 1，且非对抗性判别器输出值可以被认为是输入图像的域相似度：对于源域样本，较大的输出值意味着它与目标域更相似，而对于目标域样本，较小的输出值意味着它与源域更相似。

然而，UAN 的训练策略缺乏对两个特征分布之间中间区域的利用，容易出现过拟合进而导致公共类数据无法识别的情况[13]，本文考虑直接利用对抗域适应网络的对抗性域判别器 D 的输出 $d = D(F(\boldsymbol{x}))$ 衡量域相似度，则存在假设：

$$\mathbb{E}_{\mathbf{x}\sim q_{\overline{\mathcal{C}}_t}} d > \mathbb{E}_{\mathbf{x}\sim q_{\mathcal{C}}} d > \mathbb{E}_{\mathbf{x}\sim p_{\mathcal{C}}} d > \mathbb{E}_{\mathbf{x}\sim p_{\overline{\mathcal{C}}_s}} d。 \tag{4-2}$$

由公式 (3-3)，若 $D(F(\boldsymbol{x}))$ 值越小，则输入图像 $\boldsymbol{x}$ 越属于源域，显然假设 (4-2) 的第二个大于号成立。由于源域公共类数据 $\mathcal{D}_s^{co}$ 和目标域公共类数据 $\mathcal{D}_t^{co}$ 共享相同的标签集，与目标域私有类数据 $\mathcal{D}_t^{pr}$ 相比，$\mathcal{D}_s^{co}$ 的数据更接近 $\mathcal{D}_t^{co}$ 的数据，则假设 (4-2) 的第一个大于号成立，同理可得假设 (4-2) 的第三个大于号成立。此时可利用 d 作为域适应度计算特征对齐检测系数 $w_s(\boldsymbol{x}), w_t(\boldsymbol{x})$。

4.1.2 对抗偏移度

标签分类器 G 输出的预测 $\hat{\boldsymbol{y}}$ 包含关于输入的判别信息，但这种判别信息在源域和目标域的公共类或私有类数据空间中有所不同，即模型无法保证在不同空间中预测的标签都具备较高的可靠性，这也被称为预测不确定性。UAN 使用熵来量化预测不确定性，但进一步的研究表明了熵的一些劣势，例如缺乏对极其尖锐预测的判别能力[24]。

进一步的，考虑到预测不确定性与模型对抗鲁棒性成正比[25]，同时受到前人工作[11,13,26]的启发，本文这里对每个图像施加对抗扰动，扰动后的图像由于其数据空间不同将展现不同的鲁棒性，进一步可反映预测不确定性以达到区分源域与目标域上公共类与私有类数据的目的。

具体来说，给定一个输入图像 $\boldsymbol{x}$，经由标签分类器 G 得到的标签预测为 $\boldsymbol{y} = [y_1(\boldsymbol{x}), y_2(\boldsymbol{x}), \cdots, y_{|\mathcal{C}_s|}(\boldsymbol{x})]^\top$，其中 $y_i(\boldsymbol{x})$ 可视作 $\boldsymbol{x}$ 属于某个公共类 i 的概率，之后在图像 $\boldsymbol{x}$ 上施加一个对抗扰动得到 $\boldsymbol{x}' = \boldsymbol{x} - \varepsilon sign(-\nabla_{\boldsymbol{x}} \mathcal{L}_{ce}(\boldsymbol{x}, \max_{i\in\{1,\cdots,|\mathcal{C}_s|\}} y_i(\boldsymbol{x})))$，其中 ε 为扰动量大小的权重。

对抗扰动会降低给定输入的最大概率，则对抗偏移度形式上可表示为：

$$\boldsymbol{\delta} = \max_{i\in\{1,\ldots,|\mathcal{C}_s|\}} y_i(\boldsymbol{x}) - \max_{i\in\{1,\ldots,|\mathcal{C}_s|\}} y_i(\boldsymbol{x'})。 \tag{4-3}$$

直观上，对抗偏移度能刻画预测不确定性，对抗偏移度越大，预测结果更可靠。于是存在假设：

$$\mathbb{E}_{\mathbf{x}\sim q_{\overline{\mathcal{C}}_t}}\boldsymbol{\delta} < \mathbb{E}_{\mathbf{x}\sim q_{\mathcal{C}}}\boldsymbol{\delta} < \mathbb{E}_{\mathbf{x}\sim p_{\mathcal{C}}}\boldsymbol{\delta} < \mathbb{E}_{\mathbf{x}\sim p_{\overline{\mathcal{C}}_s}}\boldsymbol{\delta}。 \tag{4-4}$$

由于源域数据是有标记的，而目标域数据是无标记的，因此预测对于源域的图像是可靠的，即源域图像的预测结果中，会存在某一类的得分远超其他类别，但预测对于目标域图像则不显得那样可靠，其预测得分较之源域并不会如此尖锐，即它对扰动具有更强的鲁棒性，所以假设 (4-4) 中第二个小于号自然成立。

同时，考虑到源域与目标域公共类数据相互“吸引”[1]，因此来自源域公共类 $\mathcal{D}_s^{co}$ 的图像会受到来自目标域公共类 $\mathcal{D}_t^{co}$ 的图像影响，则其不确定性变得更大，对抗鲁棒性稍强。而由于 $\overline{\mathcal{C}}_s$ 与 $\mathcal{C}_t$ 没有交集，来自源域私有类 $\mathcal{D}_s^{pr}$ 的图像不会受到目标域数据的影响，其预测结果最为可靠，从另一个角度来看，$\mathcal{D}_s^{pr}$ 图像的预测泛化性更有限，更容易受到对抗扰动的影响，即对抗鲁棒性最差，显然假设 (4-4) 中第三个小于号成立。

此外，目标域私有类数据 $\mathcal{D}_t^{pr}$ 中未标记的图像不属于任何已知的公共类，并且完全不在任何已知分布中。与 $\mathcal{D}_t^{co}$ 中的数据相比，对抗性扰动对其最大标签预测的影响较小，因此假设 (4-4) 中第一个小于号成立。

4.1.3 分数整合

综上所述，通过域相似度 d 和对抗偏移度 $\boldsymbol{\delta}$ 可整合得到特征对齐检测系数 $w_s(\boldsymbol{x})$，$w_t(\boldsymbol{x})$：

$$\begin{aligned} w_s(\boldsymbol{x}) &= d - \boldsymbol{\delta}, \\ w_t(\boldsymbol{x}) &= \boldsymbol{\delta} - d。 \end{aligned} \tag{4-5}$$

由前文对域相似度 d 和对抗偏移度 $\boldsymbol{\delta}$ 的定义可知，两者值域均为 $[0,1]$。

4.2 目标域重用数据检测

为识别目标域私有类数据 $\mathcal{D}_t^{pr}$ 中的可重用部分，引入公共类相似度与类别倾向度两个指标分析重用数据检测系数 $w_d(\cdot)$。前者的意义在于：如果重用判别器 D' 无法判断图像 $\boldsymbol{x}_j \sim q_{\mathcal{C}_t}$ 是来自公共类 $\mathcal{D}_t^{co}$ 还是私有类 $\mathcal{D}_t^{pr}$，则它很可能是一个有益的图像，应该被重用；后者的意义在于：若分类器 G 很倾向于将 $\boldsymbol{x}_j \sim q_{\mathcal{C}_t}$ 归属于某个类 $c \in \mathcal{C}$，则它很可能属于这个公共类，应该被重用。

4.2.1 公共类相似度

重用判别器 D_1' 的作用是为了区分目标域下，公共类 $\mathcal{D}_t^{co}$ 数据和私有类 $\mathcal{D}_t^{pr}$ 数据，由公式 (3-4)，重用判别器 D_1' 的输出可以被视作输入 $\boldsymbol{x}_j \in \mathcal{D}_t$ 属于公共类 $\mathcal{D}_t^{co}$ 的概率，

如果 $D_1'(F(\boldsymbol{x}_j))$ 值较大，则 $\boldsymbol{x}_j$ 与潜在的公共类特征空间相似，即私有类数据更接近于公共类数据，形式上可表示为：

$$D_1'(F(\boldsymbol{x}_j)) = p(\boldsymbol{x}_j \sim q_{\mathcal{C}} | \boldsymbol{x}_j \sim q_{\mathcal{C}_t})。 \tag{4-6}$$

所以若 $D_1'(F(\boldsymbol{x}_j))$ 值较大，应该赋予这些 $\boldsymbol{x}_j$ 较大的分数来适当地重用它们。进一步的，为了扩大潜在可重用数据的得分，同时降低“有害”数据的得分，防止其被重用，即获得更有判别力的评分分配，在前述重用判别器 D_1' 输出的基础上进行归一化以计算公共类相似度评分 w_d^1：

$$w_d^1 = \frac{D_1'(F(\boldsymbol{x}_j))}{\frac{1}{|batch|}\sum_{j=1}^{|batch|} D_1'(F(\boldsymbol{x}_j))}, \tag{4-7}$$

其中 $|batch|$ 代表批量大小，指对每一个批量输入数据 $\boldsymbol{x}_j$ 的分数进行归一化。

4.2.2 类别倾向度

在训练过程中，对于标签分类器 G 的输出，往往通过计算其 softmax 分数以判断输入数据的所属类别。

更具体地，给定一个输入图像 $\boldsymbol{x}$，经由标签分类器 G 可输出其标签预测 $\boldsymbol{y} = [y_1(\boldsymbol{x}), y_2(\boldsymbol{x}), \cdots, y_{|\mathcal{C}_s|}(\boldsymbol{x})]^\top$，其中 $y_i(\boldsymbol{x})$ 可被解释为 $\boldsymbol{x}$ 属于某个类 i 的概率，本文引入温度缩放标签预测[27] $\boldsymbol{T} = [T_1(\boldsymbol{x};\rho), T_2(\boldsymbol{x};\rho), \cdots, T_{|\mathcal{C}_s|}(\boldsymbol{x};\rho)]^\top$（$\rho \in \mathbb{R}^+$ 是一个温度缩放参数，控制分布的集中程度）：

$$T_i(\boldsymbol{x};\rho) = \frac{e^{\frac{y_i(\boldsymbol{x})}{\rho}}}{\sum_{j=1}^{|\mathcal{C}_s|} e^{\frac{y_j(\boldsymbol{x})}{\rho}}}。 \tag{4-8}$$

经此计算得到的 $\boldsymbol{T}$ 中元素最大值被称为 softmax 分数。

由于标签分类器 G 是在源域 $\mathcal{D}_s$ 上训练的，这使得 G 对于分类具有较高的精准性，因此除了公共类相似度以外，目标域私有数据 $\mathcal{D}_s^{pr}$ 的标签预测也包含一个数据是否可重用的关键信息。

具体而言，可计算标签预测 $\boldsymbol{T}(\boldsymbol{x};\rho)$ 的最大和次大元素之间的预测差值建立类别倾向得分，如果一个目标域私有数据的预测差值很大，说明该图像对分类成某个类别有较大的倾向性，那么可将这个私有数据视为可重用数据。形式上可表示为（类似公共类相似度进行归一化）：

$$\begin{aligned} w_d^{2'}(\boldsymbol{x}_j) &= \max_{i\in\{1,\ldots,|\mathcal{C}_s|\}} T_i(\boldsymbol{x};\rho) - \max_{j\in\{1,\ldots,|\mathcal{C}_s|\}, j\neq i} T_j(\boldsymbol{x};\rho), \\ w_d^2(\boldsymbol{x}_j) &= \frac{w_d^{2'}(\boldsymbol{x}_j)}{\frac{1}{|batch|}\sum_{j=1}^{|batch|} w_d^{2'}(\boldsymbol{x}_j)}。 \end{aligned} \tag{4-9}$$

4.2.3 分数整合

综上，可通过计算 $w_d^1=[w_d^1(\boldsymbol{x}_1),\cdots,w_d^1(\boldsymbol{x}_{|\mathcal{D}_t^{pr}|})]^\top$ 和 $w_d^2=[w_d^2(\boldsymbol{x}_1),\cdots,w_d^2(\boldsymbol{x}_{|\mathcal{D}_t^{pr}|})]^\top$ 为所有 $\boldsymbol{x}_j\sim q_{\overline{\mathcal{C}}_t}$ 赋予公共类相似度与类别倾向度，Huang Z 等人[12]提出利用两个分数的方差可计算出重用数据检测系数 w_d。

具体来说，如果 $w_d^1(\boldsymbol{x}_j)$ 或 $w_d^2(\boldsymbol{x}_j)$ 的方差较大，意味着它们中元素的值对于刻画所有 $\boldsymbol{x}_j\sim q_{\overline{\mathcal{C}}_t}$ 的可重用性具有较强的判别能力，那么在构成最终的可重用性得分 $w_d(\boldsymbol{x}_j)$ 时需要重点关注。注意，由于之前已经将 $w_d^1(\boldsymbol{x}_j)$ 和 $w_d^2(\boldsymbol{x}_j)$ 的值域范围归一化到相同区间，因此它们的方差 $var(\cdot)$ 可用来直接计算，数学形式上可表示为：

$$\begin{aligned} w_d(\boldsymbol{x}_j)=&\frac{var(w_d^1)}{var(w_d^1)+var(w_d^2)}w_d^1(\boldsymbol{x}_j)\\ &+\frac{var(w_d^2)}{var(w_d^1)+var(w_d^2)}w_d^2(\boldsymbol{x}_j)。\end{aligned} \tag{4-10}$$

4.3 源域对齐均值

前文提出的对抗域适应网络模型以及相关的适应性准则还面临着两个问题：

1. 由于标签分类器 G 是针对源域所有类训练的，则源域私有类数据 $\mathcal{D}_s^{pr}$ 可能会误导目标域的数据分类到错误的类。

2. 由于源域私有数据的标签已知，大多数情况下最大类别往往倾向于其正确标签，目标域重用数据检测中的类别倾向度指标不能直接沿用。

为此，本文提出源域对齐均值 w_c^{avg} 这一计算指标，其指的是源域的特征对齐检测系数 w_s 相对于每个类 c 的平均权重：

$$w_c^{avg}=\frac{1}{|\mathcal{D}_s|}\sum_{i=1}^{|\mathcal{D}_s|}\mathbb{I}(y_i=c)\cdot w_s(\boldsymbol{x}_i),c\in\mathcal{C}_s。 \tag{4-11}$$

基于前文对特征对齐检测系数 w_s 的讨论可知，如果 c 在 $\mathcal{C}$ 中，则计算的源域对齐均值 w_c^{avg} 将较大，如果 c 位于 $\overline{\mathcal{C}}_s$ 中，则源域对齐均值会较小。

4.3.1 伪标签校准

得到源域对齐均值后，可校准目标域数据经过标签分类器得到的伪标签 y_p:

$$[\hat{y}]_c=\frac{w_c^{avg}\cdot e^{[y_p]_c}}{\sum_{i=1}^{|\mathcal{C}_s|}w_c^{avg}\cdot e^{[y_p]_c}},c\in\mathcal{C}_s。 \tag{4-12}$$

其中 $[\cdot]_c$ 表示向量的第 c 项。通过这样的过程，标签向量 $\hat{y}$ 中属于 $\overline{\mathcal{C}}_s$ 的标签会被抑制，而属于 $\mathcal{C}$ 的标签会被增强，从而成功地校准了伪标签 y_p 的偏差。

4.3.2 源域重用数据检测

要识别源域私有类数据 $\mathcal{D}_s^{pr}$ 中的可重用部分，可类似地引入公共类相似度与改进的类别倾向度两个指标分析重用数据检测系数 $w_r(\cdot)$。

公共类相似度指标除了是由重用判别器 D_2' 输出，其余细节与目标域检测中一致，形式上可表示为：

$$w_r^1 = \frac{D_2'(F(\boldsymbol{x}_i))}{\frac{1}{|batch|}\sum_{j=i}^{|batch|} D_2'(F(\boldsymbol{x}_i))}。\tag{4-13}$$

而关于类别倾向度指标，直观的想法是忽略源域私有类标签得分，只关注源域公共类标签得分，但由于模型假设未知两域标签重叠比例 $\frac{\mathcal{C}}{\mathcal{C}_s \cup \mathcal{C}_t}$ 这一先验知识，即源域中哪些类是私有类完全不可知。

基于上述考虑，本文利用源域对齐均值 w_c^{avg} 与阈值 w_0，对于所有经标签分类器 G 输出的源域数据标签向量，保留标签分量中 $w_c^{avg} > w_0$ 的部分而忽略标签分量中 $w_c^{avg} < w_0$ 的部分，这样的处理实际上直观地达到了忽略源域私有类标签而只关注源域公共类标签的目的。

此时可近似假设经标签分类器 G 输出的源域数据标签集与公共类标签集 $\mathcal{C}$ 一致，进一步的，一方面考虑到忽略源域私有类标签得分（由于标签分类器 G 在源域私有类上也进行监督学习，分数最大值往往处于源域私有类标签）后公共类标签得分总和可能离单位值 1 较远，另一方面需要足够高的类别倾向度分数以充分挖掘私有类数据中可重用部分，这里对筛选后的标签得分进行标准化得到 $\tilde{y}$，使得总和为 1。

改进的类别倾向度形式上可表示为：

$$\begin{aligned} w_r^{2'}(\boldsymbol{x}_j) &= \max_{i\in\{1,\ldots,|\mathcal{C}|\}} \tilde{y}_i(\boldsymbol{x}) - \max_{j\in\{1,\ldots,|\mathcal{C}|\},j\neq i} \tilde{y}_j(\boldsymbol{x}), \\ w_r^2(\boldsymbol{x}_i) &= \frac{w_r^{2'}(\boldsymbol{x}_i)}{\frac{1}{|batch|}\sum_{i=1}^{|batch|} w_r^{2'}(\boldsymbol{x}_i)}。 \end{aligned}\tag{4-14}$$

综上，可类似目标域重用数据检测的分数整合部分计算可重用性得分 $w_r(\boldsymbol{x}_i)$：

$$\begin{aligned} w_r(\boldsymbol{x}_i) = &\frac{var(w_r^1)}{var(w_r^1)+var(w_r^2)} w_r^1(\boldsymbol{x}_i) \\ &+ \frac{var(w_r^2)}{var(w_r^1)+var(w_r^2)} w_r^2(\boldsymbol{x}_i)。 \end{aligned}\tag{4-15}$$

5 实验

5.1 实验设置

5.1.1 实验细节

所有的实验都基于 Python 3.7，PyTorch 1.7.1 实现，并在 Nvidia GeForce RTX 3090 上进行。程序使用在 ImageNet[28] 上预训练的 ResNet50[29] 微调得到特征提取器 F，判别器 D 的构造与 UAN 的构造类似，其中梯度反转层 GRL 的翻转系数 $\frac{2}{1+e^{-10\times\text{迭代步数}\times\frac{1}{10000}}}-1$ 用于在训练过程的早期阶段抑制来自判别器的噪声。程序代码如附录 B 所示，具体运行细节可参考本文开源的代码库：`https://github.com/RyunMi/Undergraduate-Thesis`。

5.1.2 数据集

Office-31[14] 数据集是计算机视觉的域适应数据集，其在 3 个视觉不同的领域（A、D、W）中有 31 个类别。实验使用与 Caltech-256[30-31] 数据集共享的 10 个类别作为公共标签集 $\mathcal{C}$，接下来选择按字母顺序的 10 个类别用作 $\overline{\mathcal{C}_s}$，剩余的 11 个类别用作 $\overline{\mathcal{C}_t}$。

Office-Home[15] 数据集是一个相比 Office-31 更大的计算机视觉数据集，在 4 个不同的领域中有 65 个对象类别：艺术图像（Ar）、剪贴画图像（Cl）、产品图像（Pr）和真实世界图像（Rw）。实验按照字母顺序使用前 15 个类作为 $\mathcal{C}$，接下来的 20 个类作为 $\overline{\mathcal{C}_s}$，其余的 30 个类作为 $\overline{\mathcal{C}_t}$。

VisDA2017[16] 数据集专注于一个特殊的域适应设置（模拟到真实）。源域由游戏引擎生成的图像组成，目标域由真实世界的图像组成。这个数据集中有 12 个类。实验按照字母顺序使用前 6 个类作为 $\mathcal{C}$，接下来的 3 个类作为 $\overline{\mathcal{C}_s}$，其余的 3 个类作为 $\overline{\mathcal{C}_t}$。

5.1.3 基准比较算法

本文将对抗域适应网络与：

1. 闭集域适应方法：域对抗性神经网络（DANN[7]）；
2. 部分域适应方法：重要性加权对抗性网络（IWAN[5]），部分对抗性域自适应（PADA[5]）；
3. 开集域适应方法：迭代分配和变换算法（ATI[2]）、开集反向传播算法（OSBP[3]）；
4. 通用域适应方法：通用域适应网络（UAN[1]）。

进行比较。

5.1.4 评估范式

实验采取与 UDA 中相同的评估范式，目标域私有标签集 $\overline{\mathcal{C}_t}$ 中的所有数据都被视为一个统一的“未知”类，所有 $|\mathcal{C}+1|$ 类中每个类的平均准确度是最终结果。

5.2 分类结果

表 **5-1 Office-31** 数据集与 **VisDA2017** 数据集的平均分类精度

算法	Office-31 数据集							VisDA 数据集
	A→D	D→A	A→W	W→A	D→W	W→D	平均值	精度
DANN[7]	73.9	73.9	77.6	81.9	79.9	85.7	78.8	49.5
IWAN[5]	75.3	83.2	82.0	84.5	88.9	87.5	83.6	54.9
PADA[5]	73.0	54.6	82.2	81.0	78.2	88.4	76.2	42.1
ATI[2]	75.4	77.9	76.4	80.0	91.4	87.6	81.5	51.3
OSBP[3]	65.2	46.8	63.6	59.3	72.6	83.3	65.1	28.3
UAN[1]	77.3	84.4	82.4	83.4	93.5	95.3	86.1	56.9
DA^2L	**79.3**	81.8	79.4	81.6	88.1	90.8	83.5	**61.8**

关于上述数据集的分类实验结果分别如表5-1、和表5-2所示。可以见得，本文提出的对抗域适应算法（以下简称为 DA^2L）在每个数据集的平均分类精度基本不输于甚至一定程度上要优于所有的比较算法，而由于不同数据集本身数据量、类别数目以及类别分布情况的不同，得到不同的实验结果是有一定逻辑的。

具体而言，对于 Office-31 数据集的实验结果（表5-1左侧），由表中可以看出，UAN 算法的平均分类精度要优于本文的 DA^2L 算法，IWAN 算法的平均分类精度与 DA^2L 算法的持平，而其余算法的平均分类精度与 DA^2L 算法的相比都更低。这可能是由于相比 UAN 算法，在通用域适应设置下，其余的比较算法都会产生负迁移[23]现象，导致性能甚至可能比只在源域上训练而不进行任何调整的模型差[1]。而关于 DA^2L 算法，比 UAN 算法表现较差的原因可能在于 Office-31 数据集数据量小、类别少的特点，导致源域私有类或者目标域私有类的数据可重用性实际上普遍较差，而在其中选取部分数据进行重用略显勉强，反倒影响分类效果，以至于在其他比较算法都表现较好的通用域适应问题上性能并不太突出，但可以注意到，对于 $\mathbf{A}\rightarrow\mathbf{D}$ 域适应问题，包括 UAN 算法在内的其他算法性能都较差，相比较而言 DA^2L 算法的表现更突出，这可能是因为在 $\mathbf{A}\rightarrow\mathbf{D}$ 的设置下，源域公共类数据的分类并不能完全帮助目标域公共类数据的分类，有一些可利用的信息蕴藏在源域或目标域私有类数据中，而 DA^2L 算法恰好将它们提取了出来以达到最高的平均分类精度。

进一步的，对于更难处理的 Office-Home 数据集，相较于 Office-31 数据集而言，其

表 **5-2 Office-Home** 数据集的平均分类精度

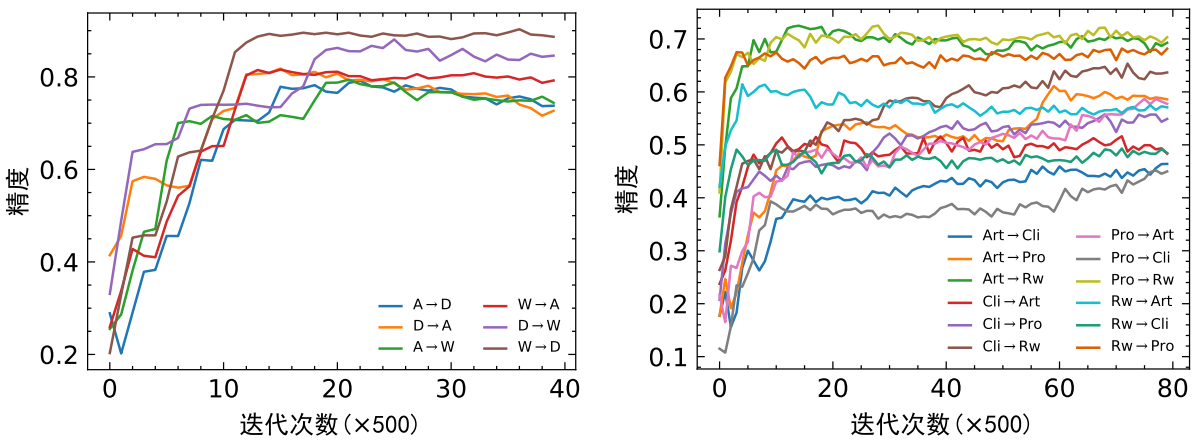
	算法	DANN[7]	IWAN[5]	PADA[5]	ATI[2]	OSBP[3]	UAN[1]	DA^2L
Office-Home 数据集	Ar→Cl	43.1	40.3	30.3	40.6	36.6	48.3	45.2
	Cl→Ar	48.2	49.6	43.9	49.9	41.6	54.0	51.6
	Ar→Pr	60.1	59.8	51.0	59.1	44.8	60.9	**61.1**
	Pr→Ar	54.3	58.2	37.6	57.7	47.9	60.8	58.7
	Ar→Rw	71.3	71.0	62.6	70.5	63.0	72.1	**72.5**
	Rw→Ar	58.0	56.6	55.5	55.6	51.6	60.9	**61.0**
	Cl→Pr	49.0	47.4	45.0	48.3	41.1	52.5	**55.7**
	Pr→Cl	42.6	43.0	26.8	43.3	33.3	43.9	**45.2**
	Cl→Rw	60.5	61.6	56.0	61.0	53.7	61.7	**65.3**
	Rw→Cl	41.6	43.3	32.3	43.7	40.0	45.9	**49.1**
	Pr→Rw	69.4	68.8	64.4	69.3	64.2	70.2	**72.5**
	Rw→Pr	69.9	70.5	69.7	70.0	67.1	70.9	68.2
	平均值	55.7	55.8	47.9	55.8	48.7	58.5	**58.8**

类别数目更多，数据量更庞大，由实验结果（表5-2）可以发现，本文提出的 DA^2L 算法在 12 个问题设置下有 8 个情景的平均分类精度要优于所有的比较算法，并且分类精度平均值也是最高的，这说明对于数据量和类别数都更多的 Office-Home 数据集，源域数据在公共类的分类大部分情况下已经无法满足目标域数据在公共类的分类需求了，而此时提取源域和目标域的私有部分中可重用的有益数据，可以对分类带来一定帮助并且在大部分情况下都不会产生负迁移。尽管分类精度平均值相比 UAN 算法的值提高很少，但依前文所述，DA^2L 算法已初步展示了其有能力处理数据量大、类别数目多的数据集并且在许多情况下达到最好的分类效果。

最后，如表5-1最右侧一列所示，对于三个数据集中数据量最大的 VisDA2017 数据集，本文提出的 DA^2L 算法的平均分类精度在很大程度上优于其余所有的比较算法，值得注意的是，此时 DA^2L 算法相比 UAN 算法提高的程度是所有数据集中最大的，这可能是因为在 VisDA2017 数据集中源域与目标域私有数据众多，可重用的有益数据比例也更高，合理的挖掘提取对分类任务是极有帮助的。

综上实验结果表明，DA^2L 算法面对数据量较小的数据集时，分类表现可圈可点，在部分任务设置下能达到最高水平，而在面对数据量庞大或者数据类别众多的困难数据集时，分类表现在整体上较大程度优于现有的一众先进算法，这表明 DA^2L 更加符合真实世界情景并具备极大潜力。

另外，图5-1展示了 DA^2L 算法在 Office-31 数据集和 Office-Home 数据集下随迭代次数的分类精度变化曲线图。由左图可以发现，DA^2L 算法在 Office-31 数据集中的



(a) Office-31　　(b) Office-Home

图 **5-1 Office-31** 和 **Office-Home** 下分类精度曲线

$W \to D, D \to W, W \to A$ 三种设置下都没有产生过拟合，但是在其他三种设置下都有轻微过拟合现象。反观右图，DA^2L 算法在 Office-Home 数据集除 $Rw \to Art$ 之外的所有设置都没有产生过拟合现象，这也进一步解释了为什么 DA^2L 算法在 Office-31 数据集的平均分类精度持平于 IWAN 算法且不如 UAN 算法但在 Office-Home 数据集上的表现优于所有的比较算法。

5.3 性能分析

5.3.1 不同类比例下分析

如图5-2，通过改变公共类 $\mathcal{C}$ 的大小来分析 DA^2L 算法的表现，分析是在 VisDA2017 数据集中完成的，此时 $|\mathcal{C}|+|\overline{\mathcal{C}_s}|+|\overline{\mathcal{C}_t}|=12$，改变 $|\mathcal{C}|$ 时 $|\overline{\mathcal{C}_s}|=|\overline{\mathcal{C}_t}|$，并且考虑到当 $|\mathcal{C}|=0$ 时，只要 w_0 足够高，则分类精度总能为 1，所以忽略此情况。由图可以发现，除了当

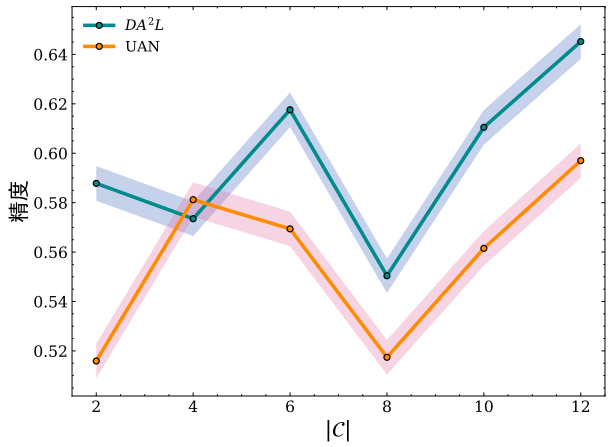
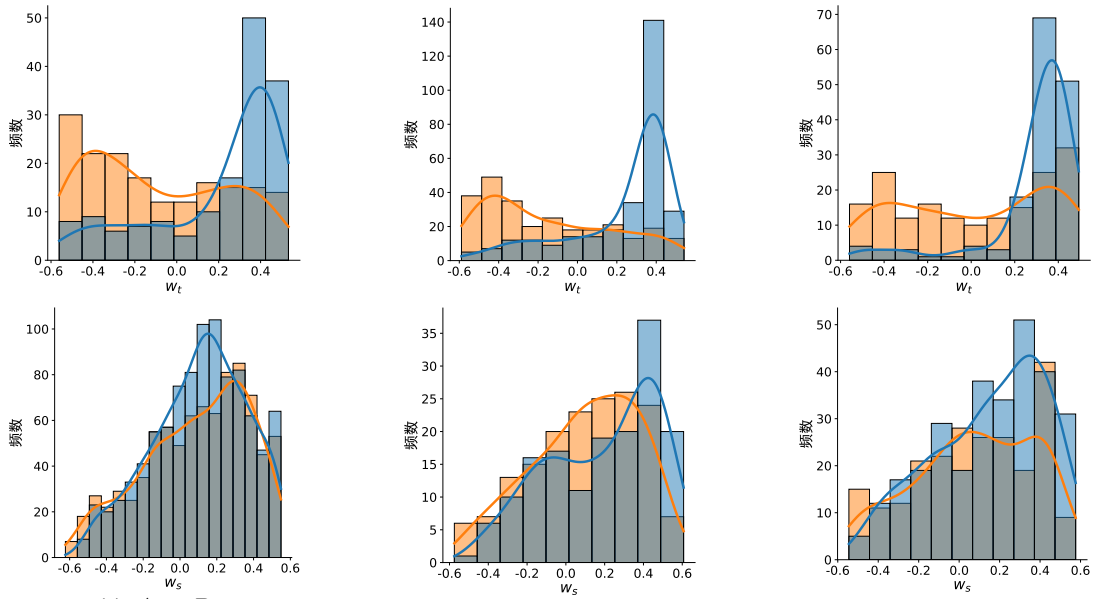


图 **5-2** DA^2L 与 **UAN** 改变公共类/私有类比例下的分类精度变化折线

$|\mathcal{C}| = 4$ 时，其余情况下 DA^2L 算法表现都要优于 UAN 算法，而随着 $|\mathcal{C}|$ 的增大，DA^2L 算法与 UAN 算法的变化趋势基本一致，这说明 DA^2L 算法的特征对齐检测与 UAN 中的“样本级迁移准则”取得了类似效果，并且 DA^2L 算法中对两域私有类数据的合理重用正是相比 UAN 算法而言提高分类精度的重要因素。另一方面，当 $|\mathcal{C}|$ 减小时，DA^2L 算法的分类精度相比 UAN 算法的变化更加平稳，且在公共类比例很小时（$|\mathcal{C}| = 2$）时依然有较高的得分，这说明了 DA^2L 算法的稳健性。

5.3.2 适应性准则假设验证

为了证明第4章中适应性准则的假设，此处选取 Office-31 数据集中 $A \rightarrow D, D \rightarrow W, W \rightarrow D$ 三种设置下源域与目标域数据绘制其特征对齐系数的概率密度分布如图5-3所示，三个子图中每个子图的上图为目标域系数分布，下图为源域系数分布。图中蓝色曲线与条柱代表公共类数据，橙色曲线与条柱代表私有类数据。



(a) $A \rightarrow D$ (b) $D \rightarrow W$ (c) $W \rightarrow D$

图 **5-3 Office-31** 中三种设置下特征对齐系数在源域和目标域不同类数据的分布

由图可知无论是源域还是目标域，公共类与私有类数据系数分布间都产生了一定距离即具备较高区分度，这样的结果表明，适应性准则中的假设基本都是正确的，这也是 DA^2L 算法表现优异的重要原因之一。

5.3.3 敏感性分析

为了证明不同的阈值 w_0 对分类精度的影响效果，本文选取 Office-31 数据集中的设置 $A \rightarrow D, W \rightarrow D$、Office-Home 数据集中的设置 $Art \rightarrow Cli, Art \rightarrow Pro, Art \rightarrow Rw$ 以及 VisDA2017 数据集展开了敏感性分析，实验结果如图5-4所示。

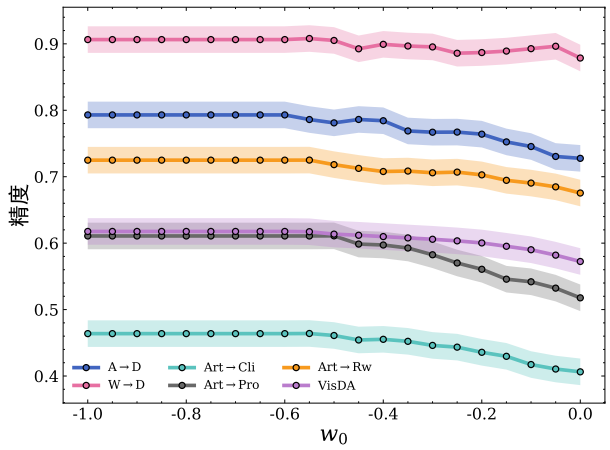


图 **5-4** 改变阈值 w_0 的分类精度变化折线

由图可以见得，从 -1 到 0 改变 w_0，当 $w_0 \leq -0.6$ 时，精度保持恒定，而随着 w_0 的增大，精度有所波动但整体维持在一个较小的范围内，这表明了 DA^2L 算法对阈值 w_0 这一参数的鲁棒性。

5.3.4 消融实验

表5-3展示了不同 DA^2L 算法变体的消融实验结果。具体而言，这里考虑去除监督损失项、特征对齐损失项、数据重用损失项中的目标域数据重用部分以及源域数据重用部分，分别记为 w/o ce，w/o dom，w/o d_t，w/o d_s。由表可见去除任何一个部分后算法的性能都会下降，具体来说去除监督损失项后，分类精度仅为 25.9%，说明此时算法远未达到拟合程度，而特征对齐损失、目标域数据重用损失以及源域数据重用损失对分类精度的提高分别为 2.4%，3.9% 与 5.4% ，充分说明 DA^2L 算法的各个部分都是重要且必要的。

表 **5-3 VisDA2017** 数据集上消融实验平均分类精度

算法	w/o ce	w/o dom	w/o d_t	w/o d_s	DA^2L
分类精度	25.9	59.4	57.9	56.4	**61.8**

6 结论与展望

本文针对通用域适应问题，提出了对抗域适应（DA^2L）算法，训练时：首先提取源域与目标域数据的特征并输出 softmax 分数以在源域数据上进行监督学习；其次利用带梯度反转层的域判别器输出与对抗扰动后的标签预测偏移得到特征对齐系数，在筛选出两域公共类与私有类数据的同时进行对抗性特征对齐；接着进一步计算目标域私有类数据与源域私有类数据的重用系数，期间需要利用两个独立的带梯度反转层的重用判别器输出与依靠 softmax 分数计算的类别倾向度，最后将可重用的目标域私有类数据和源域私有类数据分别与目标域公共类数据进行对抗性对齐以帮助分类。推理时：计算所有测试数据的特征对齐系数，若高于阈值 w_0 则预测其具体类别，反之将其归为统一的“未知”类。

本文进行的实验证明了 DA^2L 算法在面临数据量较大、类别数较多的数据集时，能优于 DANN、IWAN、PADA、ATI、OSBP 与 UAN 等域适应算法，在 Office-31 数据集、Office-Home 数据集与 VisDA2017 数据集上展现优异的性能，并有望推广至更真实复杂的问题中。另外，本文后续进行的包括不同公共类数据比例下的分析、适应性准则假设验证、敏感性分析与消融实验在内的性能分析无不证明了算法的合理性、有效性与稳健性。

进一步的，本文的 DA^2L 算法还有可改进之处：(1) 如何自适应的选择阈值 w_0 将有助于减少公共数据对齐与私有数据重用之间的矛盾；(2) 源域和目标域私有类数据与目标域公共类数据的对抗性重用结构能否更加精简；(3) 如何不将目标域私有类数据单纯地分为一个统一的类别而使其更接近真实的类别。

参考文献

[1] YOU K, LONG M, CAO Z, et al. Universal domain adaptation[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2019: 2720-2729.

[2] PANAREDA BUSTO P, GALL J. Open set domain adaptation[C]//Proceedings of the IEEE international conference on computer vision. 2017: 754-763.

[3] SAITO K, YAMAMOTO S, USHIKU Y, et al. Open set domain adaptation by backpropagation[C]// Computer Vision – ECCV 2018. Springer, 2018: 153-168.

[4] CAO Z, LONG M, WANG J, et al. Partial transfer learning with selective adversarial networks[C]// Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 2724-2732.

[5] ZHANG J, DING Z, LI W, et al. Importance weighted adversarial nets for partial domain adaptation [C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 8156-8164.

[6] GOODFELLOW I, POUGET-ABADIE J, MIRZA M, et al. Generative adversarial networks[J]. Communications of the ACM, 2020, 63(11): 139-144.

[7] GANIN Y, USTINOVA E, AJAKAN H, et al. Domain-adversarial training of neural networks[J]. The journal of machine learning research, 2016, 17(1): 2096-2030.

[8] LONG M, CAO Z, WANG J, et al. Conditional adversarial domain adaptation[J]. Advances in Neural Information Processing Systems, 2018, 31.

[9] TSAI Y H, HUNG W C, SCHULTER S, et al. Learning to adapt structured output space for semantic segmentation[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 7472-7481.

[10] LAINE S, AILA T. Temporal ensembling for semi-supervised learning[A]. 2017. arXiv: 1610.02242.

[11] MIYATO T, MAEDA S I, KOYAMA M, et al. Virtual adversarial training: a regularization method for supervised and semi-supervised learning[J]. IEEE transactions on pattern analysis and machine intelligence, 2018, 41(8): 1979-1993.

[12] HUANG Z, YANG J, GONG C. They are not completely useless: Towards recycling transferable unlabeled data for class-mismatched semi-supervised learning[J]. IEEE Transactions on Multimedia, 2022.

[13] HUANG Z, XUE C, HAN B, et al. Universal semi-supervised learning[J]. Advances in Neural Information Processing Systems, 2021, 34: 26714-26725.

[14] SAENKO K, KULIS B, FRITZ M, et al. Adapting visual category models to new domains[C]// Computer Vision – ECCV 2010. Springer, 2010: 213-226.

[15] VENKATESWARA H, EUSEBIO J, CHAKRABORTY S, et al. Deep hashing network for unsupervised domain adaptation[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 5018-5027.

[16] PENG X, USMAN B, KAUSHIK N, et al. Visda: A synthetic-to-real benchmark for visual domain adaptation[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. 2018: 2021-2026.

[17] 李航, 等. 统计学习方法[M]. 清华大学出版社, 2012.

[18] GLOROT X, BORDES A, BENGIO Y. Deep sparse rectifier neural networks[C]//Proceedings of the fourteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, 2011: 315-323.

[19] GOODFELLOW I, WARDE-FARLEY D, MIRZA M, et al. Maxout networks[C]//International conference on machine learning. PMLR, 2013: 1319-1327.

[20] HINTON G E, SRIVASTAVA N, KRIZHEVSKY A, et al. Improving neural networks by preventing co-adaptation of feature detectors[A]. 2012. arXiv: 1207.0580.

[21] BEN-DAVID S, BLITZER J, CRAMMER K, et al. Analysis of representations for domain adaptation [J]. Advances in Neural Information Processing Systems, 2006, 19.

[22] BEN-DAVID S, BLITZER J, CRAMMER K, et al. A theory of learning from different domains[J]. Machine learning, 2010, 79: 151-175.

[23] PAN S J, YANG Q. A survey on transfer learning[J]. IEEE Transactions on knowledge and data engineering, 2010, 22(10): 1345-1359.

[24] FU B, CAO Z, LONG M, et al. Learning to detect open classes for universal domain adaptation[C]// Computer Vision – ECCV 2020. Springer, 2020: 567-583.

[25] 陈思宏, 沈浩靖, 王冉, 等. 预测不确定性与对抗鲁棒性的关系研究[J]. 软件学报, 2022, 33(2): 524-538.

[26] GOODFELLOW I J, SHLENS J, SZEGEDY C. Explaining and harnessing adversarial examples[A]. 2015. arXiv: 1412.6572.

[27] LIANG S, LI Y, SRIKANT R. Enhancing the reliability of out-of-distribution image detection in neural networks[A]. 2020. arXiv: 1706.02690.

[28] DENG J, DONG W, SOCHER R, et al. Imagenet: A large-scale hierarchical image database[C]// Proceedings of the IEEE conference on computer vision and pattern recognition. 2009: 248-255.

[29] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[30] GRIFFIN G, HOLUB A, PERONA P. Caltech 256[M]. CaltechDATA, 2022.

[31] GONG B, SHI Y, SHA F, et al. Geodesic flow kernel for unsupervised domain adaptation[C]// Proceedings of the IEEE conference on computer vision and pattern recognition. 2012: 2066-2073.

致　谢

谨以此文献给我亲爱的外公。

附录 A 计算机程序文件结构

DA^2L

- config
  - config.py
  - office-train-config.yaml
  - office-test-config.yaml
  - officehome-train-config.yaml
  - officehome-test-config.yaml
  - visda-train-config.yaml
  - visda-test-config.yaml
- data
  - data_loader.py
  - office ..... Office-31 数据集
  - officehome ..... Office-Home 数据集
  - visda ..... VisDA2017 数据集
- models
  - net.py
  - utils.py
- pretrained
  - resnet50.pth
- main.py
- requirement.txt

附录 B 计算机程序代码

config.py

```
import yaml
import easydict
from os.path import join

class Dataset:
    def __init__(self, path, domains, files, prefix):
        self.path = path
        self.prefix = prefix
        self.domains = domains
        self.files = [(join(path, file)) for file in files]
        self.prefixes = [self.prefix] * len(self.domains)

import argparse

parser = argparse.ArgumentParser(description='Code for *Domain
    Adaptation Based on Adversarial Learning*', formatter_class=argparse.
    ArgumentDefaultsHelpFormatter)
parser.add_argument('--config', type=str, default='config.yaml', help='/
    path/to/config/file')

args = parser.parse_args()
config_file = args.config
args = yaml.safe_load(open(config_file))

save_config = yaml.safe_load(open(config_file))

args = easydict.EasyDict(args)

dataset = None
if args.data.dataset.name == 'office':
```

```
28      dataset = Dataset(
29          path=args.data.dataset.root_path,
30          domains=['amazon', 'dslr', 'webcam'],
31          files=[
32              'amazon.txt',
33              'dslr.txt',
34              'webcam.txt'
35          ],
36          prefix=args.data.dataset.root_path)
37  elif args.data.dataset.name == 'officehome':
38      dataset = Dataset(
39          path=args.data.dataset.root_path,
40          domains=['Art', 'Clipart', 'Product', 'Real_World'],
41          files=[
42              'Art.txt',
43              'Clipart.txt',
44              'Product.txt',
45              'Real_World.txt'
46          ],
47          prefix=args.data.dataset.root_path)
48  elif args.data.dataset.name == 'visda2017':
49      dataset = Dataset(
50          path=args.data.dataset.root_path,
51          domains=['train', 'validation'],
52          files=[
53              'train_list.txt',
54              'validation_list.txt'
55          ],
56          prefix=args.data.dataset.root_path)
57      dataset.prefixes = [join(dataset.path, 'train'), join(dataset.path, '
            validation')]
58  else:
59      raise Exception(f'dataset {args.data.dataset.name} not supported!')
60
61  source_domain_name = dataset.domains[args.data.dataset.source]
62  target_domain_name = dataset.domains[args.data.dataset.target]
```

```
source_file = dataset.files[args.data.dataset.source]
target_file = dataset.files[args.data.dataset.target]
```

data_loader.py

```
from config.config import *
from easydl import *
from collections import Counter
from torchvision.transforms.transforms import *
from torch.utils.data import DataLoader, WeightedRandomSampler

'''
assume classes across domains are the same.
[0 1
    ..................................................................
     N - 1]
|----common classes --||----source private classes --||----target
    private classes --|
'''
a, b, c = args.data.dataset.n_share, args.data.dataset.n_source_private,
    args.data.dataset.n_total
c = c - a - b
common_classes = [i for i in range(a)]
source_private_classes = [i + a for i in range(b)]
target_private_classes = [i + a + b for i in range(c)]

source_classes = common_classes + source_private_classes
target_classes = common_classes + target_private_classes

train_transform = Compose([
    Resize(256),
    RandomCrop(224),
    RandomHorizontalFlip(),
    ToTensor()
])

test_transform = Compose([
```

```
29      Resize(256),
30      CenterCrop(224),
31      ToTensor()
32  ])
33
34  source_train_ds = FileListDataset(list_path=source_file, path_prefix=
        dataset.prefixes[args.data.dataset.source], transform=train_transform,
         filter=(lambda x: x in source_classes))
35  source_test_ds = FileListDataset(list_path=source_file,path_prefix=
        dataset.prefixes[args.data.dataset.source], transform=test_transform,
        filter=(lambda x: x in source_classes))
36  target_train_ds = FileListDataset(list_path=target_file, path_prefix=
        dataset.prefixes[args.data.dataset.target], transform=train_transform,
         filter=(lambda x: x in target_classes))
37  target_test_ds = FileListDataset(list_path=target_file, path_prefix=
        dataset.prefixes[args.data.dataset.target], transform=test_transform,
        filter=(lambda x: x in target_classes))
38
39  classes = source_train_ds.labels
40  freq = Counter(classes)
41  class_weight = {x : 1.0 / freq[x] if args.data.dataloader.class_balance
        else 1.0 for x in freq}
42
43  source_weights = [class_weight[x] for x in source_train_ds.labels]
44
45  weight_random_sampler = WeightedRandomSampler(source_weights, len(
        source_train_ds.labels))
46
47  source_train_dl = DataLoader(dataset=source_train_ds, batch_size=args.
        data.dataloader.batch_size,shuffle=False, sampler=
        weight_random_sampler, num_workers=args.data.dataloader.data_workers,
        drop_last=True)
48  source_test_dl = DataLoader(dataset=source_test_ds, batch_size=args.data.
        dataloader.batch_size, shuffle=False, num_workers=1, drop_last=False)
49  target_train_dl = DataLoader(dataset=target_train_ds, batch_size=args.
        data.dataloader.batch_size,shuffle=True, num_workers=args.data.
```

```
    dataloader.data_workers, drop_last=True)
target_test_dl = DataLoader(dataset=target_test_ds, batch_size=args.data.
    dataloader.batch_size, shuffle=False, num_workers=1, drop_last=False)
```

net.py

```
import torch
import torch.nn as nn
from torchvision import models
from easydl import *


class BaseFeatureExtractor(nn.Module):
    def forward(self, *input):
        pass

    def __init__(self):
        super(BaseFeatureExtractor, self).__init__()

    def output_num(self):
        pass

    def train(self, mode=True):
        # freeze BN mean and std
        for module in self.children():
            if isinstance(module, nn.BatchNorm2d):
                module.train(False)
            else:
                module.train(mode)


class ResNet50Fc(BaseFeatureExtractor):
    """
    ** input image should be in range of [0, 1]**
    """
    def __init__(self, model_path=None, normalize=True):
        super(ResNet50Fc, self).__init__()
```

```
32          if model_path:
33              if os.path.exists(model_path):
34                  self.resnet_model = models.resnet50(pretrained=False)
35                  self.resnet_model.load_state_dict(torch.load(model_path))
36              else:
37                  raise Exception('invalid model path!')
38          else:
39              self.resnet_model = models.resnet50(pretrained=True)
40
41          if model_path or normalize:
42              # pretrain model is used, use ImageNet normalization
43              self.normalize = True
44              self.register_buffer('mean', torch.tensor([0.485, 0.456,
                    0.406]).view(1, 3, 1, 1))
45              self.register_buffer('std', torch.tensor([0.229, 0.224,
                    0.225]).view(1, 3, 1, 1))
46          else:
47              self.normalize = False
48
49          resnet_model = self.resnet_model
50          self.conv1 = resnet_model.conv1
51          self.bn1 = resnet_model.bn1
52          self.relu = resnet_model.relu
53          self.maxpool = resnet_model.maxpool
54          self.layer1 = resnet_model.layer1
55          self.layer2 = resnet_model.layer2
56          self.layer3 = resnet_model.layer3
57          self.layer4 = resnet_model.layer4
58          self.avgpool = resnet_model.avgpool
59          self.__in_features = resnet_model.fc.in_features
60
61      def forward(self, x):
62          if self.normalize:
63              x = (x - self.mean) / self.std
64          x = self.conv1(x)
65          x = self.bn1(x)
```

```
66          x = self.relu(x)
67          x = self.maxpool(x)
68          x = self.layer1(x)
69          x = self.layer2(x)
70          x = self.layer3(x)
71          x = self.layer4(x)
72          x = self.avgpool(x)
73          x = x.view(x.size(0), -1)
74          return x
75
76      def output_num(self):
77          return self.__in_features
78
79  class CLS(nn.Module):
80      """
81      a two-layer MLP for classification
82      """
83      def __init__(self, in_dim, out_dim, bottle_neck_dim=256):
84          super(CLS, self).__init__()
85          self.bottleneck = nn.Linear(in_dim, bottle_neck_dim)
86          self.fc = nn.Linear(bottle_neck_dim, out_dim)
87          self.main = nn.Sequential(self.bottleneck, self.fc, nn.Softmax(dim
                =-1))
88
89      def forward(self, x):
90          out = [x]
91          for module in self.main.children():
92              x = module(x)
93              out.append(x)
94          return out
95
96
97  class AdversarialNetwork(nn.Module):
98      """
99      AdversarialNetwork with a gredient reverse layer.
100     its ``forward`` function calls gredient reverse layer first, then
```

```
        applies ``self.main`` module.
    """
    def __init__(self, in_feature):
        super(AdversarialNetwork, self).__init__()
        self.main = nn.Sequential(
            nn.Linear(in_feature, 1024),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(1024,1024),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(1024, 1),
            nn.Sigmoid()
        )
        self.grl = GradientReverseModule(lambda step: aToBSheduler(step,
            0.0, 1.0, gamma=10, max_iter=10000))
        # coeff = 0.0 + (2.0 / (1 + np.exp(- gamma * step * 1.0 /
            max_iter)) - 1.0) * (1.0 - 0.0)

    def forward(self, x):
        x_ = self.grl(x)
        y = self.main(x_)
        return y
```

utils.py

```
from easydl import *
import torch.nn.functional as F
from config.config import *

def seed_everything(seed=1234):
    import random
    random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    np.random.seed(seed)
    import os
```

```
    os.environ['PYTHONHASHSEED'] = str(seed)

def TempScale(p, t):
    return p / t

def perturb(inputs, feature_extractor, classifier, class_temperature=10.0):
    with TrainingModeManager([feature_extractor, classifier], train=False):
        inputs.requires_grad = True
        features = feature_extractor.forward(inputs)
        _, _, score, _ = classifier.forward(features)
        score = score / class_temperature
        softmax_score = nn.Softmax(-1)(score)
        max_value, max_target = torch.max(softmax_score, dim=1)
        xent = F.cross_entropy(softmax_score, max_target.detach().long())

        d = torch.autograd.grad(xent, inputs)[0]
        d = torch.ge(d, 0)
        d = (d.float() - 0.5) * 2
        # Normalizing the gradient to the same space of image
        d[0][0] = (d[0][0] )/(0.229)
        d[0][1] = (d[0][1] )/(0.224)
        d[0][2] = (d[0][2] )/(0.225)
        inputs.data.add_(-args.train.eps, d.detach())

        features = feature_extractor.forward(inputs)
        _, _, output, _ = classifier.forward(features)
        softmax_output = TempScale(output, args.train.temp).softmax(1)
        max_value_hat = torch.max(softmax_output, dim=1).values
        pred_shift = torch.abs(max_value - max_value_hat).unsqueeze(1)

    return pred_shift

def reverse_sigmoid(y):
    return torch.log(y / (1.0 - y + 1e-10) + 1e-10)
```

```
46
47  def get_source_share_weight(domain_out, pred_shift, domain_temperature
        =1.0):
48      domain_logit = reverse_sigmoid(domain_out)
49      domain_logit = domain_logit / domain_temperature
50      domain_out = nn.Sigmoid()(domain_logit)
51
52      pred_shift = (pred_shift - pred_shift.min()) / (pred_shift.max() -
            pred_shift.min())
53      pred_shift = reverse_sigmoid(pred_shift)
54      pred_shift = pred_shift / domain_temperature
55      pred_shift = nn.Sigmoid()(pred_shift)
56
57      weight = domain_out - pred_shift
58      weight = weight.detach()
59
60      # entropy = torch.sum(- after_softmax * torch.log(after_softmax + 1e
            -10), dim=1, keepdim=True)
61      # entropy_norm = entropy / np.log(after_softmax.size(1))
62      # weight = entropy_norm - domain_out
63      # weight = weight.detach()
64
65      return weight
66
67  def get_target_share_weight(domain_out, pred_shift, domain_temperature
        =1.0):
68      return - get_source_share_weight(domain_out, pred_shift,
            domain_temperature)
69
70  def normalize_weight(x):
71      x = (x - x.min()) / (x.max() - x.min())
72      x = x / torch.mean(x)
73      return x.detach()
74
75  def common_private_spilt(share_weight, feature):
76      indices_private = torch.nonzero(torch.lt(share_weight, args.test.w_0)
```

```
        )
77      feature_private = torch.index_select(feature, 0, indices_private[:,
            0])
78
79      indices_common = torch.nonzero(torch.ge(share_weight, args.test.w_0))
80      feature_common = torch.index_select(feature, 0, indices_common[:, 0])
81
82      return feature_private.detach(), feature_common.detach()
83
84  def get_target_reuse_weight(reuse_out, before_softmax, reuse_temperature
        =1.0, common_temperature = 1.0):
85      reuse_logit = reverse_sigmoid(reuse_out)
86      reuse_logit = reuse_logit / reuse_temperature
87      reuse_out = nn.Sigmoid()(reuse_logit)
88
89      before_softmax = before_softmax / common_temperature
90      after_softmax = nn.Softmax(-1)(before_softmax)
91
92      max, _ = after_softmax.topk(2, dim=1, largest=True)
93      class_tend = max[:,0]-max[:,1]
94      class_tend = class_tend / torch.mean(class_tend)
95
96      # class_tend = reverse_sigmoid(class_tend)
97      # class_tend = class_tend / reuse_temperature
98      # class_tend = nn.Sigmoid()(class_tend)
99
100     w_r1 = torch.var(reuse_out)
101     w_r2 = torch.var(class_tend)
102     w_r1 = torch.where(torch.isnan(w_r1), torch.full_like(w_r1, 0), w_r1)
103     w_r2 = torch.where(torch.isnan(w_r2), torch.full_like(w_r2, 0), w_r2)
104
105     if w_r1 or w_r2:
106         w_r = (w_r1 / (w_r1 + w_r2) * reuse_out).view(-1) + \
107     (w_r2 / (w_r1 + w_r2) * class_tend).view(-1)
108     else:
109         w_r = reuse_out.view(-1) + class_tend.view(-1)
```

```
    w_r = w_r.detach()

    return w_r

def compute_avg_weight(weight, label, class_weight):
    for i in range(len(class_weight)):
        mask = (label == i)
        class_weight[i] = weight[mask].mean()
    class_weight = torch.where(torch.isnan(class_weight), torch.full_like
        (class_weight, 0), class_weight)
    return class_weight


def pseudo_label_calibration(pslab, weight, pslab_temperature = 1.0):
    #weight = weight.transpose(1, 0).expand(pslab.shape[0], -1)
    weight = normalize_weight(weight)
    pslab = torch.exp(pslab / pslab_temperature)
    pslab = pslab * weight
    pslab = pslab / torch.sum(pslab, 1, keepdim=True)
    return pslab, weight.detach()

def get_source_reuse_weight(reuse_out, fc, w_avg, reuse_temperature=1.0,
    common_temperature = 10.0):
    reuse_logit = reverse_sigmoid(reuse_out)
    reuse_logit = reuse_logit / reuse_temperature
    reuse_out = nn.Sigmoid()(reuse_logit)

    label_ind = torch.nonzero(torch.ge(w_avg, args.test.w_0))
    fc = torch.index_select(fc, 1, label_ind[:, 0])
    # fc = F.normalize(fc, p=1, dim=1)
    fc = fc / common_temperature
    fc_softmax = nn.Softmax(-1)(fc)

    if min(fc_softmax.shape) == 0:
        class_tend = torch.zeros((fc_softmax.shape[0]),1)

```

```
    if fc_softmax.shape[1] == 1:
        class_tend = fc_softmax
    else:
        max , _= fc_softmax.topk(2, dim=1, largest=True)
        class_tend = max[:,0]-max[:,1]
        class_tend = class_tend / torch.mean(class_tend)

    # class_tend = reverse_sigmoid(class_tend)
    # class_tend = class_tend / common_temperature
    # class_tend = nn.Sigmoid()(class_tend)

    w_r1 = torch.var(reuse_out)
    w_r2 = torch.var(class_tend)
    w_r1 = torch.where(torch.isnan(w_r1), torch.full_like(w_r1, 0), w_r1)
    w_r2 = torch.where(torch.isnan(w_r2), torch.full_like(w_r2, 0), w_r2)

    if w_r1 or w_r2:
        w_r = (w_r1 / (w_r1 + w_r2) * reuse_out).view(-1) + \
    (w_r2 / (w_r1 + w_r2) * class_tend).view(-1)
    else:
        w_r = reuse_out.view(-1) + class_tend.view(-1)
    w_r = w_r.detach()

    return w_r
```

main.py

```
from data.data_loader import *
from models.net import *
from models.utils import *
import datetime
from tqdm import tqdm
if is_in_notebook():
    from tqdm import tqdm_notebook as tqdm
from torch import optim
#from torch.utils.tensorboard import SummaryWriter
from tensorboardX import SummaryWriter
```

```
import torch.backends.cudnn as cudnn

cudnn.benchmark = True
cudnn.deterministic = True

seed_everything()

if args.misc.gpus < 1:
    import os
    os.environ["CUDA_VISIBLE_DEVICES"] = ""
    gpu_ids = []
    device = torch.device('cpu')
else:
    gpu_ids = select_GPUs(args.misc.gpus)
    device = gpu_ids[0]

now = datetime.datetime.now().strftime('%b%d-%H_%M_%S')

log_dir = f'{args.log.root_dir}{now}'

logger = SummaryWriter(log_dir)

with open(join(log_dir, 'config.yaml'), 'w') as f:
    f.write(yaml.dump(save_config))

model_dict = {
    'resnet50': ResNet50Fc,
    'resnext101':Resnext101Fc
}

class TotalNet(nn.Module):
    def __init__(self):
        super(TotalNet, self).__init__()
        self.feature_extractor = model_dict[args.model.base_model](args.
            model.pretrained_model)
        classifier_output_dim = len(source_classes)
```

```
46          self.classifier = CLS(self.feature_extractor.output_num(),
                classifier_output_dim, bottle_neck_dim=256)
47          self.domain_discriminator = AdversarialNetwork(256)
48          #self.domain_discriminator_separate = AdversarialNetwork(256)
49          self.reuse_discriminator_s = AdversarialNetwork(256)
50          self.reuse_discriminator_t = AdversarialNetwork(256)
51
52      def forward(self, x):
53          f = self.feature_extractor(x)
54          f, _, __, y = self.classifier(f)
55          d = self.domain_discriminator(_)
56          #d_0 = self.domain_discriminator_separate(_)
57          hat_d_s = self.reuse_discriminator_s(_)
58          hat_d_t = self.reuse_discriminator_t(_)
59          return y, d, hat_d_t, hat_d_s #d_0#, hat_d_s, hat_d_t
60
61  totalNet = TotalNet()
62  totalNet.to(device)
63
64  feature_extractor = nn.DataParallel(totalNet.feature_extractor,
        device_ids=gpu_ids, output_device=device).train(True)
65  classifier = nn.DataParallel(totalNet.classifier, device_ids=gpu_ids,
        output_device=device).train(True)
66  domain_discriminator = nn.DataParallel(totalNet.domain_discriminator,
        device_ids=gpu_ids, output_device=device).train(True)
67  reuse_discriminator_s = nn.DataParallel(totalNet.reuse_discriminator_s,
        device_ids=gpu_ids, output_device=device).train(True)
68  reuse_discriminator_t = nn.DataParallel(totalNet.reuse_discriminator_t,
        device_ids=gpu_ids, output_device=device).train(True)
69
70  # ================== evaluation
71  if args.test.test_only:
72      assert os.path.exists(args.test.resume_file)
73      data = torch.load(open(args.test.resume_file, 'rb'))
74      feature_extractor.load_state_dict(data['feature_extractor'])
75      classifier.load_state_dict(data['classifier'])
```

```
    domain_discriminator.load_state_dict(data['domain_discriminator'])
    # w_avg.load_state_dict(data['w_avg'])

    counters = [AccuracyCounter() for x in range(len(source_classes) + 1)
        ]
    with TrainingModeManager([feature_extractor, classifier,
        domain_discriminator], train=False) as mgr, \
            Accumulator(['feature', 'predict_prob', 'label', 'domain_prob'
                , 'before_softmax', 'target_share_weight']) as
                target_accumulator:#, \
                #torch.no_grad():
        for i, (im, label) in enumerate(tqdm(target_test_dl, desc='
            testing ')):
            im = im.to(device)
            label = label.to(device)

            feature = feature_extractor.forward(im)
            feature, __, before_softmax, predict_prob = classifier.forward
                (feature)

            # predict_prob, _ = pseudo_label_calibration(predict_prob,
                w_avg)

            domain_prob = domain_discriminator.forward(__)

            pred_shift = perturb(im, feature_extractor, classifier,
                class_temperature=1.0)

            target_share_weight = get_target_share_weight(domain_prob,
                pred_shift, domain_temperature=1.0)

            for name in target_accumulator.names:
                globals()[name] = variable_to_numpy(globals()[name])

            target_accumulator.updateData(globals())

```

```
103     for x in target_accumulator:
104         globals()[x] = target_accumulator[x]
105 
106     def outlier(each_target_share_weight):
107         return each_target_share_weight < args.test.w_0
108 
109     counters = [AccuracyCounter() for x in range(len(source_classes) + 1)
            ]
110 
111     for (each_predict_prob, each_label, each_target_share_weight) in zip(
            predict_prob, label, target_share_weight):
112         if each_label in source_classes:
113             counters[each_label].Ntotal += 1.0
114             each_pred_id = np.argmax(each_predict_prob)
115             if not outlier(each_target_share_weight[0]) and each_pred_id
                    == each_label:
116                 counters[each_label].Ncorrect += 1.0
117         else:
118             counters[-1].Ntotal += 1.0
119             if outlier(each_target_share_weight[0]):
120                 counters[-1].Ncorrect += 1.0
121 
122     acc_tests = [x.reportAccuracy() for x in counters if not np.isnan(x.
            reportAccuracy())]
123     acc_test = torch.ones(1, 1) * np.mean(acc_tests)
124     print(f'test accuracy is {acc_test.item()}')
125     exit(0)
126 
127 # ================== optimizer
128 scheduler = lambda step, initial_lr: inverseDecaySheduler(step,
        initial_lr, gamma=10, power=0.75, max_iter=10000)
129 optimizer_finetune = OptimWithSheduler(
130     optim.SGD(feature_extractor.parameters(), lr=args.train.lr /10.0,
            weight_decay=args.train.weight_decay, momentum=args.train.momentum
            , nesterov=True), 0scheduler)
131 optimizer_cls = OptimWithSheduler(
```

```
132     optim.SGD(classifier.parameters(), lr=args.train.lr, weight_decay=
            args.train.weight_decay, momentum=args.train.momentum, nesterov=
            True), scheduler)
133 optimizer_domain_discriminator = OptimWithSheduler(
134     optim.SGD(domain_discriminator.parameters(), lr=args.train.lr,
            weight_decay=args.train.weight_decay, momentum=args.train.momentum
            , nesterov=True), scheduler)
135 optimizer_reuse_discriminator_t = OptimWithSheduler(
136     optim.SGD(reuse_discriminator_t.parameters(), lr=args.train.lr,
            weight_decay=args.train.weight_decay, momentum=args.train.momentum
            , nesterov=True), scheduler)
137 optimizer_reuse_discriminator_s = OptimWithSheduler(
138     optim.SGD(reuse_discriminator_s.parameters(), lr=args.train.lr,
            weight_decay=args.train.weight_decay, momentum=args.train.momentum
            , nesterov=True), scheduler)
139
140 global_step = 0
141 best_acc = 0
142
143 total_steps = tqdm(range(args.train.min_step),desc='global step')
144 epoch_id = 0
145 total_epoch = min(len(source_train_dl), len(target_train_dl))
146 source_share_weight_epoch = torch.zeros(total_epoch * args.data.
        dataloader.batch_size, 1).to(device)
147 label_source_epoch = torch.zeros(total_epoch * args.data.dataloader.
        batch_size).to(device)
148 w_avg = torch.zeros(len(source_classes)).to(device)
149
150 # ================== train
151 while global_step < args.train.min_step:
152
153     iters = tqdm(zip(source_train_dl, target_train_dl), desc=f'epoch {
            epoch_id} ', total=total_epoch)
154     epoch_id += 1
155
156     for i, ((im_source, label_source), (im_target, label_target)) in
```

```
        enumerate(iters):
157
158         save_label_target = label_target # for debug usage
159
160         label_source = label_source.to(device)
161         label_target = label_target.to(device)
162         label_target = torch.zeros_like(label_target)
163
164         # ========================= forward pass
165         im_source = im_source.to(device)
166         im_target = im_target.to(device)
167
168         fc1_s = feature_extractor.forward(im_source)
169         fc1_t = feature_extractor.forward(im_target)
170
171         fc1_s, feature_source, fc2_s, predict_prob_source = classifier.
               forward(fc1_s)
172         fc1_t, feature_target, fc2_t, predict_prob_target = classifier.
               forward(fc1_t)
173
174         # Output of Domain Discriminator
175         domain_prob_discriminator_source = domain_discriminator.forward(
               feature_source)
176         domain_prob_discriminator_target = domain_discriminator.forward(
               feature_target)
177
178         # Adversarial perturbation
179         pred_shift_source = perturb(im_source, feature_extractor,
               classifier, class_temperature=10.0)
180         pred_shift_target = perturb(im_target, feature_extractor,
               classifier, class_temperature=1.0)
181
182         # w_s and w_t
183         source_share_weight = get_source_share_weight(
               domain_prob_discriminator_source, pred_shift_source,
               domain_temperature=1.0)
```

```
184         source_share_weight_norm = normalize_weight(source_share_weight)
185         target_share_weight = get_target_share_weight(
                domain_prob_discriminator_target, pred_shift_target,
                domain_temperature=1.0)
186         target_share_weight_norm = normalize_weight(target_share_weight)
187
188         # Pseudo Label Calibration
189         source_share_weight_epoch[(global_step % total_epoch) * args.data.
                dataloader.batch_size : (global_step % total_epoch + 1) * args.
                data.dataloader.batch_size] = source_share_weight.clone()
190         label_source_epoch[(global_step % total_epoch) * args.data.
                dataloader.batch_size : (global_step % total_epoch + 1) * args.
                data.dataloader.batch_size] = label_source.clone()
191
192         if epoch_id == 1:
193             w_avg = compute_avg_weight(source_share_weight_epoch[0 : (
                    global_step % total_epoch + 1) * args.data.dataloader.
                    batch_size], label_source_epoch[0 : (global_step %
                    total_epoch + 1) * args.data.dataloader.batch_size], w_avg)
194             _, w_avg = pseudo_label_calibration(fc2_t, w_avg,
                    pslab_temperature = 1.0)
195         else:
196             w_avg = compute_avg_weight(source_share_weight_epoch,
                    label_source_epoch, w_avg)
197             _, w_avg = pseudo_label_calibration(fc2_t, w_avg,
                    pslab_temperature = 1.0)
198
199         # # Reuse Detect and Reuse Loss
200         feature_target_private, feature_target_common =
                common_private_spilt(target_share_weight, feature_target)
201
202         dt_loss = torch.zeros(1, 1).to(device)
203         ds_loss = torch.zeros(1, 1).to(device)
204
205         if min(feature_target_private.shape) == 0:
206             pass
```

```
207         else:
208             fc_target_private, _ = common_private_spilt(
                    target_share_weight, fc2_t)
209             # target_share_weight_private, _ = common_private_spilt(
                    target_share_weight, target_share_weight_norm)
210             reuse_prob_discriminator_private_t = reuse_discriminator_t.
                    forward(feature_target_private)
211
212             target_reuse_weight = get_target_reuse_weight(
                    reuse_prob_discriminator_private_t, fc_target_private)
213             # * (1 / (1 + target_share_weight_private)).view(-1)
214             tmp = target_reuse_weight * nn.BCELoss(reduction='none')\
215                 (reuse_prob_discriminator_private_t, torch.zeros_like(
                        reuse_prob_discriminator_private_t)).view(-1)
216             dt_loss += torch.mean(tmp, dim=0, keepdim=True)
217
218         if min(feature_target_common.shape) == 0:
219             pass
220         else:
221             reuse_prob_discriminator_common_t1 = reuse_discriminator_t.
                    forward(feature_target_common)
222             tmp = nn.BCELoss(reduction='none')(
                    reuse_prob_discriminator_common_t1, torch.ones_like(
                    reuse_prob_discriminator_common_t1))
223             dt_loss += torch.mean(tmp, dim=0, keepdim=True)
224
225             reuse_prob_discriminator_common_t2 = reuse_discriminator_t.
                    forward(feature_target_common)
226             tmp = nn.BCELoss(reduction='none')(
                    reuse_prob_discriminator_common_t2, torch.ones_like(
                    reuse_prob_discriminator_common_t2))
227             ds_loss += torch.mean(tmp, dim=0, keepdim=True)
228
229         feature_source_private, _ = common_private_spilt(
                source_share_weight, feature_source)
230
```

```
231         if min(feature_source_private.shape) == 0:
232             pass
233         else:
234             fc_source_private, _ = common_private_spilt(
                    source_share_weight, fc2_s)
235             reuse_prob_discriminator_private_s = reuse_discriminator_s.
                    forward(feature_source_private)
236 
237             source_reuse_weight = get_source_reuse_weight(
                    reuse_prob_discriminator_private_s, fc_source_private,
                    w_avg, reuse_temperature=1.0, common_temperature = 10.0)
238             tmp = source_reuse_weight * nn.BCELoss(reduction='none')(
                    reuse_prob_discriminator_private_s, torch.zeros_like(
                    reuse_prob_discriminator_private_s)).view(-1)
239             ds_loss += torch.mean(tmp, dim=0, keepdim=True)
240 
241         # ============================== domain loss
242         dom_loss = torch.zeros(1, 1).to(device)
243 
244         tmp = source_share_weight_norm * nn.BCELoss(reduction='none')(
                domain_prob_discriminator_source, torch.zeros_like(
                domain_prob_discriminator_source))
245         dom_loss += torch.mean(tmp, dim=0, keepdim=True)
246         tmp = target_share_weight_norm * nn.BCELoss(reduction='none')(
                domain_prob_discriminator_target, torch.ones_like(
                domain_prob_discriminator_target))
247         dom_loss += torch.mean(tmp, dim=0, keepdim=True)
248 
249         # ============================== cross entropy loss
250         ce = nn.CrossEntropyLoss(reduction='none')(predict_prob_source,
                label_source)
251         ce = torch.mean(ce, dim=0, keepdim=True)
252 
253         with OptimizerManager(
254                 [optimizer_finetune, optimizer_cls,
                        optimizer_domain_discriminator,
```

```
255                 optimizer_reuse_discriminator_t,
                        optimizer_reuse_discriminator_s]):
256             loss = ce + dom_loss + dt_loss + ds_loss
257             loss.backward()
258
259         global_step += 1
260         total_steps.update()
261
262         if global_step % args.log.log_interval == 0:
263             counter = AccuracyCounter()
264             counter.addOneBatch(variable_to_numpy(one_hot(label_source,
                    len(source_classes))), variable_to_numpy(
                    predict_prob_source))
265             acc_train = torch.tensor([counter.reportAccuracy()]).to(device
                    )
266             logger.add_scalar('dom_loss', dom_loss, global_step)
267             logger.add_scalar('ce', ce, global_step)
268             logger.add_scalar('dt_loss', dt_loss, global_step)
269             logger.add_scalar('ds_loss', ds_loss, global_step)
270             logger.add_scalar('acc_train', acc_train, global_step)
271
272         # ================== validation
273         if global_step % (args.test.test_interval) == 0:
274
275             counters = [AccuracyCounter() for x in range(len(
                    source_classes) + 1)]
276             with TrainingModeManager([feature_extractor, classifier,
                    domain_discriminator], train=False) as mgr, \
277             Accumulator(['feature', 'predict_prob', 'label', 'domain_prob'
                    , 'before_softmax','target_share_weight']) as
                    target_accumulator:#, \
                    #torch.no_grad():
278
279
280                 for i, (im, label) in enumerate(tqdm(target_test_dl, desc='
                        testing ')):
281                     im = im.to(device)
```

```
282                 label = label.to(device)
283
284                 feature = feature_extractor.forward(im)
285                 feature, __, before_softmax, predict_prob = classifier.
                        forward(feature)
286
287                 pred_shift_target = perturb(im, feature_extractor,
                        classifier, class_temperature=1.0)
288
289                 # predict_prob, _ = pseudo_label_calibration(
                        predict_prob, w_avg)
290
291                 domain_prob = domain_discriminator.forward(__)
292
293                 target_share_weight = get_target_share_weight(
                        domain_prob, pred_shift_target, domain_temperature
                        =1.0)
294
295                 for name in target_accumulator.names:
296                     globals()[name] = variable_to_numpy(globals()[name])
297
298                 target_accumulator.updateData(globals())
299
300             for x in target_accumulator:
301                 globals()[x] = target_accumulator[x]
302
303             def outlier(each_target_share_weight):
304                 return each_target_share_weight < args.test.w_0
305
306             counters = [AccuracyCounter() for x in range(len(
                    source_classes) + 1)]
307
308             for (each_predict_prob, each_label, each_target_share_weight)
                    in zip(predict_prob, label, target_share_weight):
309                 if each_label in source_classes:
310                     counters[each_label].Ntotal += 1.0
```

```
                each_pred_id = np.argmax(each_predict_prob)
                if not outlier(each_target_share_weight[0]) and
                    each_pred_id == each_label:
                    counters[each_label].Ncorrect += 1.0
            else:
                counters[-1].Ntotal += 1.0
                if outlier(each_target_share_weight[0]):
                    counters[-1].Ncorrect += 1.0

        acc_tests = [x.reportAccuracy() for x in counters if not np.
            isnan(x.reportAccuracy())]
        acc_test = torch.ones(1, 1) * np.mean(acc_tests)

        logger.add_scalar('acc_test', acc_test, global_step)

        clear_output()
        print(f'test accuracy is {acc_test.item()}')

        data = {
            "feature_extractor": feature_extractor.state_dict(),
            'classifier': classifier.state_dict(),
            'domain_discriminator': domain_discriminator.state_dict()
                if not isinstance(domain_discriminator, Nonsense) else
                1.0,
            # 'w_avg': w_avg.state_dict(),
        }

        if acc_test > best_acc:
            best_acc = acc_test
            with open(join(log_dir, 'best.pkl'), 'wb') as f:
                torch.save(data, f)

        with open(join(log_dir, 'current.pkl'), 'wb') as f:
            torch.save(data, f)
```