Gradual Domain Adaptation with Multifidelity Learning and Generative Model

by

Shogo Sagawa

Dissertation

submitted to the Department of Statistical Science

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

The Graduate University for Advanced Studies, SOKENDAI

September 2024

Abstract

Applying machine learning to real-world tasks is often constrained due to the limited number of labeled data. Domain adaptation is a practical approach to mitigate the issue. However, conventional domain adaptation methods do not work well when a large gap exists between the source and target domains. Gradual domain adaptation is one of the approaches used to address the problem. This method utilizes the intermediate domains that are arranged to connect the source domain to the target domain. In previous studies, it is assumed that the number of intermediate domains is large and the gap between adjacent domains is small; hence, the gradual domain adaptation algorithm, involving self-training with unlabeled datasets, is applicable. In practice, however, gradual self-training will fail because the number of intermediate domains is limited and the discrepancies between adjacent domains is large.

For example, automatic driving systems require high object recognition accuracy not only during the daytime but also in the evening and nighttime. If the labeled data are obtained only from the daytime, the object recognition accuracy for the evening and nighttime will deteriorate. Gradual self-training will be effective when we have images that are captured continuously from day to night. However, when the interval of capturing images is long, the gaps between adjacent domains is also large, and gradual self-training may not be effective.

When the applicability of gradual self-training is constrained due to the limited number of available intermediate domains, one option is to discard these intermediate domains and apply conventional domain adaptation only to the source and target domains. We argue that given intermediate domains are valuable and should be used, even when the gaps between adjacent domains are large. In this dissertation, to address this issue, we propose two gradual domain adaptation methods that do not use self-training.

First, we consider a problem setting of semi-supervised gradual domain adaptation, in which we can query the label of a sample from the intermediate domains and the target domain. As a practical problem setting, we consider that querying the label of a sample in each domain comes with some cost, which is not uniform. While a simple query strategy is to devote the entire budget to queries from the target domain, we propose to utilize the notion of multifidelity learning and query from both the intermediate and target domains. Our experimental results demonstrate that the proposed query strategy is more suited to our problem setting, where the query cost of each domain is not uniform, than this simple query strategy.

Next, we propose a method that addresses the issue of large discrepancies between adjacent domains while maintaining the framework of unsupervised domain adaptation. We utilize the normalizing flow, which is one of the generative models, to capture the continuous change between domains. In general, normalizing flows learn the transformation between the distribution that the given data follows and a parametric distribution, such as the Gaussian distribution. To realize gradual domain adaptation without gradual self-training, we develop a non-parametric transformation method between distributions using normalizing flows. We evaluate the effectiveness of our proposed method on real-world datasets and confirm that it mitigates the problem, in which there are large discrepancies between adjacent domains, and improves the classification performance.

Contents

1 Introduction

1.1 Machine learning methods for handling small dataset

Machine learning is a subset of the research area of artificial intelligence. It aims to implement human-like learning and cognitive abilities in computers without explicit programming. In the learning process of machine learning, it attempts to extract meaningful patterns from the data automatically.

Machine learning methods are broadly classified into three categories: supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, annotators assign outputs (i.e., labels) to inputs (such as images), and a machine learning algorithm learns a map from inputs to outputs. The goal of unsupervised learning is to identify distinctive patterns or structures in unlabeled data. Reinforcement learning is suitable for decision-making tasks. A reinforcement learning algorithm learns the optimal behavior within a given environment to obtain the maximum reward. In this section, we focus on supervised learning and consider its advantages and issues.

Supervised learning enable the automation of tasks and the enhancement of decision-

making processes by predicting labels of inputs on specific tasks. Machine learning is widely used across many real-world tasks. We show examples as follows:

- Based on the purchase history of an individual, a machine learning model predicts the products that the person is likely to purchase next.
- A machine learning model uses the lifestyle habits of an individual as input to predict the possibility that the person will contract a specific disease.
- By inputting the cultivation conditions, such as the amount of pesticide, a machine learning model predicts the yield of crops.

A huge number of input-output pairs allows machine learning to model complex events accurately. Recently, there has been a significant increase in the number of parameters of deep learning models. GPT-3 is a well-known large-scale language model that utilizes up to 175 billion parameters, and executing GPT-3 requires a significant memory capacity of 700 GB.[1] Even when a large number of input-output pairs are available, only a limited number of organizations can perform the training since training a large model also requires substantial computational resources.

In practical problem scenarios, the number of available input-output pairs is often limited. We can consider the situation where the number of input-output pairs is limited, but the number of inputs is large. For example, we consider training a model that predicts the yield of crops using the cultivation conditions. It is not realistic to obtain a large number of input-output pairs from a single farmer because crops require several months to mature before they can be harvested. In contrast, we can prepare a large number of inputs by listing controllable cultivation conditions and considering their combinations. Applying machine learning to real-world tasks is often constrained due to the limited number of labeled data. Therefore, there is a strong demand for machine learning algorithms that can mitigate the problem of limited labeled data.

By leveraging past experiences, humans can tackle tasks even if they have not previously encountered them. For example, we consider the process of learning a programming language. It is commonly said that people who have mastered some programming language can learn other programming languages easily. Transfer learning (Zhuang et al., 2021) is inspired by the learning process of humans, aiming to

[1] See https://lambdalabs.com/blog/demystifying-gpt-3.

apply the rich knowledge gained from one task to another. When the number of labeled data is limited, it is hard to construct a large-scale model with numerous parameters from scratch. We utilize the rich knowledge from a pre-trained large-scale model for the task where the number of labeled data is limited. Transfer learning is a practical approach to mitigate the problem setting where the number of labeled data is limited.

In transfer learning, the problem setting in which the task of machine learning is the same but the probability distributions of the inputs are different is known as domain adaptation (Redko et al., 2019). In domain adaptation, the source domain is rich in labeled data, and we extract the knowledge for the task from this domain. The target domain has limited labeled data, and we transfer the knowledge extracted from the source domain. The problem setting, in which a limited number of labeled samples are available from the target domain, is called semi-supervised domain adaptation (Wang and Deng, 2018), while the problem setting with no labeled samples is termed unsupervised domain adaptation (Wilson and Cook, 2020).

1.2 Applicability of domain adaptation

Domain adaptation is promising for real-world tasks since it can leverage limited labeled data. When the discrepancy between the source and target domains is small, domain adaptation performs well. However, in practical situations, the discrepancy between the source and target domains might not be small. In order to extend the applicability of domain adaptation methods, it is necessary to develop domain adaptation algorithms that can handle cases with a large discrepancy between the source and target domains. To mitigate the problem of a large discrepancy between the source and target domains, Kumar et al. (2020) proposed gradual domain adaptation (GDA), which is categorized as unsupervised domain adaptation. They assumed that there is a gradual change from the distribution of the source domain to that of the target domain, and that the intermediate domains are arranged to connect the source domain to the target domain. A large discrepancy between the source and target domains is divided into several smaller discrepancies using the intermediate domains. We show a schematic of GDA in Figure 1.1.

We argue that the assumption that there is a gradual change from the distribution of the source domain to that of the target domain is realistic. For example, automatic

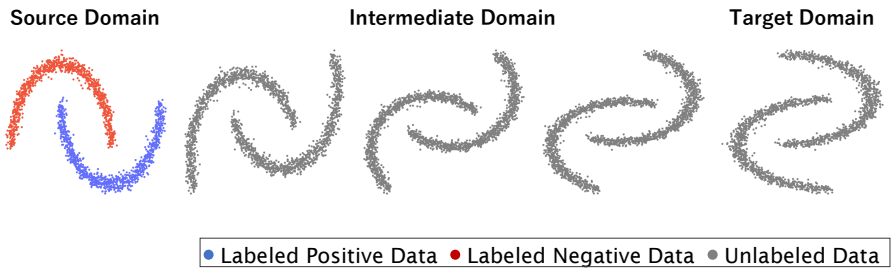


Figure 1.1: Schematic of GDA. The intermediate domains are arranged to connect the source domain to the target domain.

driving systems require high object recognition accuracy not only during the daytime but also in the evening and nighttime. If the labeled data are obtained only from the daytime, the object recognition accuracy for the evening and nighttime will deteriorate. Gradual domain adaptation will work when images are taken continuously throughout the day and night.

We can effectively utilize the rich knowledge gained from the source domain by applying GDA when the sequence of intermediate domains is given. Kumar et al. (2020) assumed that the discrepancies between adjacent domains are sufficiently small and proposed a GDA algorithm that utilizes self-training (Amini et al., 2022). We revisit the task of object recognition in automatic driving systems. Suppose that the self-training-based GDA method is applicable when the images are taken every minute. Namely, the discrepancies between adjacent domains are sufficiently small when the images are captured every minute. When the capture interval of the images increases from every minute to every hour, the gaps between domains become larger, and the self-training-based GDA method will not be effective.

When the applicability of self-training-based GDA methods is constrained due to the limited number of available intermediate domains, one option is to discard these intermediate domains and apply conventional domain adaptation only to the source and target domains. We argue that given intermediate domains are valuable and should be used, even when the gaps between adjacent domains are large. In this dissertation, we study a GDA problem in which there are large discrepancies between adjacent domains since the number of accessible intermediate domains is limited. To address this issue, we propose two GDA methods that do not use self-training.

First, we consider a problem setting of semi-supervised GDA, in which we can query

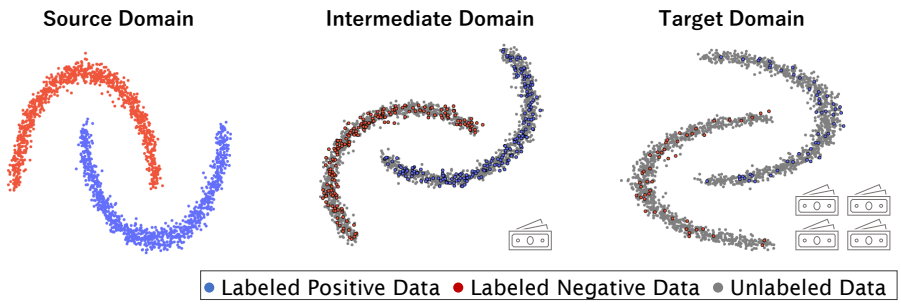


Figure 1.2: Extending the applicability of GDA with multifidelity learning.

the label of a sample from the intermediate domains and the target domain (Sagawa and Hino, 2023). In reality, querying the label of a sample in each domain comes with some cost, which is not uniform. In our problem setting, the query cost is highest in the target domain, and the query cost in the intermediate domain is higher when it is closer to the target domain and lower when it is closer to the source domain. Figure 1.2 shows a schematic of our problem setting. Since we are considering a situation where querying the label of a sample comes with some cost, we use active learning (Settles, 2009; Hino, 2020) for efficient queries. If we obtain abundant labels from the target domain or from the intermediate domains that are close to the target domain, we can expect acquiring a model that can precisely predict the label of a sample in the target domain. As queries from these domains are costly, there is a trade-off between the predictive performance and query cost. Multifidelity learning (Peherstorfer et al., 2018) aim to determine the query sample by considering both its query cost and the usefulness of the label. In this dissertation, we apply multifidelity learning to GDA. One simple query strategy is to devote the entire budget to queries from the target domain. Our experimental results show that utilizing the source domain and the intermediate domains is more effective than using only the target domain.

Next, we propose a method to deal with the problem of large discrepancies between adjacent domains while maintaining the framework of unsupervised domain adaptation (Sagawa and Hino, 2022). Our primary focus is to learn the transition from the source domain to the target domain through intermediate domains. We utilize the normalizing flow (Papamakarios et al., 2021), one of the generative models, to capture the change between the distributions of the source and target domains. A schematic of our proposed method is shown in Figure 1.3. Generally, normalizing flow learns

the transformation between the distribution that the given data follows and a known distribution, such as the Gaussian distribution. In this dissertation, our flow-based model is designed to learn the transformation between the distribution of one domain and that of another domain. While our main purpose is GDA, as a byproduct, our proposed method can generate synthetic samples from any intermediate domain even when no sample from the domain is observed. The results of experiments show that the proposed method mitigates the problems with large gaps between domains while maintaining the framework of unsupervised domain adaptation.

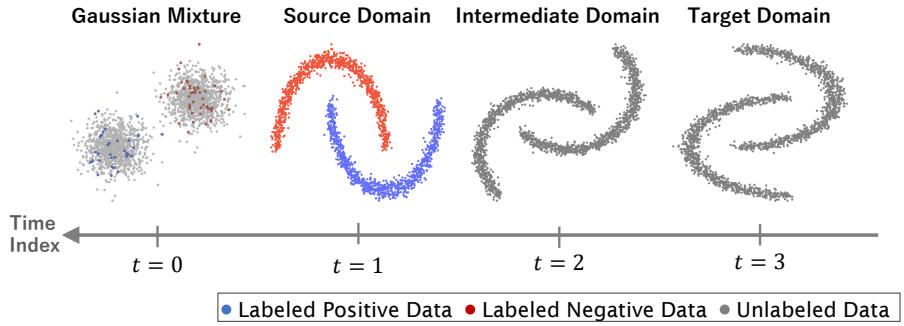


Figure 1.3: The continuous change between domains are captured using normalizing flows.

The rest of this dissertation is organized as follows. In Chapter 2, we provide an overview of conventional domain adaptation and explain in detail the GDA algorithm proposed by Kumar et al. (2020). We also introduce related work of GDA in this chapter. In Chapter 3, we consider a problem setting of semi-supervised GDA, in which we can query the label of a sample from the intermediate domains and the target domain. Note that we also suppose that querying the label of a sample in each domain comes with some cost, which is not uniform. To tackle the trade-off between the query cost of a sample and the usefulness of the label of the sample, we propose to utilize the notion of multifidelity learning. In Chapter 4, we address the problem of large gaps between adjacent domains while maintaining the framework of unsupervised domain adaptation. We propose the use of normalizing flows to capture the continuous change between domains. Chapter 5 is devoted to the conclusion of our study.

2
Preliminary

2.1 Domain adaptation

Domain adaptation aim to transfer the knowledge extracted from the source domain to the target domain, and to obtain a model with high predictive performance in the target domain. *Domain* is a notion that includes a dataset, the features extracted from the dataset, and the models trained on the dataset. We can broadly categorize domain adaptation methods based on what we transfer from the source domain.

Importance weighting (Sugiyama et al., 2012) is a well-known example of unsupervised domain adaptation methods that transfer the dataset in the source domain. This method mitigates the effect of covariate shift (Shimodaira, 2000). Covariate shift assume that while the marginal distribution of the input differs between the source and target domains, the conditional distribution of the output given the input remains unchanged. The discrepancy between the source and target domains is represented as the density ratio of the input distribution. The importance of each sample is determined by the estimated density ratio. In order to estimate the density ratio, input data from both

the source and target domains are used. Then, the model is trained with the weighted input-output pairs from the source domain.

In semi-supervised domain adaptation, it is effective to transfer the trained model in the source domain. The limited labeled samples from the target domain are utilized to fine-tune the model trained with the source dataset. When labels provided by annotators are not available, self-training can be performed as an alternative. The technique of self-training is frequently used in semi-supervised learning (Sohn et al., 2020), which does not account for changes in the probability distribution. Liu et al. (2021) experimentally demonstrated that pseudo-labels are ineffective under the condition where the distributions followed by the training data and the test data are different. Furthermore, they propose a cycle self-training method that is suitable for domain adaptation. Two classifiers are prepared for both the source and target datasets. During the model training process, penalties are imposed when the outputs from the two classifiers disagree.

Lastly, we review domain adaptation methods that aim to extract features from the source and target datasets that are invariant across domains. In order to extract domain-invariant features, it is important to measure the discrepancy between domains. Several definitions of discrepancy have been proposed, and domain adaptation methods have been designed to extract features that reduce the discrepancy between the source and target domains. We briefly describe well-known unsupervised domain adaptation algorithms that extract domain-invariant features.

DANN (Domain Adversarial Neural Network): To develop a domain adaptation method, Ganin et al. (2016) focused on the $\mathcal{H}$-divergence (Ben-David et al., 2010).

Definition 2.1 ($\mathcal{H}$-divergence (Ben-David et al., 2010))**.** *The distributions of the source and target domains over input space $\mathcal{X}$ are denoted as p_S and p_T, respectively. Let $\mathcal{H} = \{h \mid h : \mathcal{X} \to \{0, 1\}\}$ be a hypothesis space. The $\mathcal{H}$-divergence between p_S and p_T is*

$$d_{\mathcal{H}}(p_S, p_T) = 2 \sup_{h \in \mathcal{H}} \left| \Pr_{\boldsymbol{x} \sim p_S}[h(\boldsymbol{x}) = 1] - \Pr_{\boldsymbol{x} \sim p_T}[h(\boldsymbol{x}) = 1] \right|.$$

Ganin et al. (2016) proposed a domain adaptation method that uses adversarial training to extract features such that the $\mathcal{H}$-divergence between the source and target domains becomes small. Specifically, DANN extracts features from the datasets that make it

difficult for the domain discriminator to distinguish between the source and target domains.

MMD (Maximum Mean Discrepancy): The technique of kernel mean embedding (Smola et al., 2007) projects samples from an arbitrary distribution into a Reproducing Kernel Hilbert Space (RKHS) using a kernel function, and computes their average within the RKHS. The MMD (Gretton et al., 2012) is the distance between two probability distributions in an RKHS. Li et al. (2018) proposed a domain adaptation method that extracts domain-invariant features by adding the MMD regularization term to the loss function of the feature extractor.

CORAL (Correlation Alignment): Sun and Saenko (2016) proposed a simple domain adaptation method that uses the second-order statistics of distributions of the source and target domains. The CORAL loss is the square of the Frobenius norm of the difference between the covariance matrices of each domain. By adding the CORAL loss to the loss function of the feature extractor, domain-invariant features are extracted from the source and target datasets.

WDGRL (Wasserstein Distance Guided Representation Learning): In WDGRL (Shen et al., 2018), the discrepancy between the source and target domains is measured as the Wasserstein distance (Villani, 2009) between the distributions of the source and target domains. The Wasserstein distance is defined as follows.

Definition 2.2 (Wasserstein distance (Villani, 2009))**.** *Let ν_1 and ν_2 be the probability measure on input space $\mathcal{X}$, and $\pi \in \Pi(\nu_1, \nu_2)$ be a coupling between ν_1 and ν_2. For any $p \geq 1$, given a distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$, the p-Wasserstein distance is defined as*

$$W_p(\nu_1, \nu_2) = \left(\inf_{\pi \in \Pi(\nu_1, \nu_2)} \int_{\mathcal{X} \times \mathcal{X}} d(\boldsymbol{x}', \tilde{\boldsymbol{x}})^p d\pi(\nu_1, \nu_2) \right)^{1/p},$$

where $\boldsymbol{x}' \sim \nu_1$ and $\tilde{\boldsymbol{x}} \sim \nu_2$. Moreover, the ∞-Wasserstein distance is defined as

$$W_\infty(\nu_1, \nu_2) = \lim_{p \to \infty} W_p(\nu_1, \nu_2).$$

As a result of the visualizing the features extracted by WDGRL, it is confirmed that a

cluster structure is formed for each label. Consequently, it is suggested that WDGRL can learn features suitable for domain adaptation.

KL (Kullback–Leibler divergence): The KL divergence (Kullback and Leibler, 1951) is a popular measure of discrepancy between two probability distributions. The definition of the KL divergence is shown as follows.

Definition 2.3 (KL divergence (Kullback and Leibler, 1951))**.** *Given two distributions p_1 and p_2 over input space $\mathcal{X}$. The KL divergence between p_1 and p_2 is*

$$\mathrm{KL}[p_2|p_1] = \int_{\mathcal{X}} p_2(\boldsymbol{x}) \log \frac{p_2(\boldsymbol{x})}{p_1(\boldsymbol{x})} d\boldsymbol{x}.$$

Nguyen et al. (2022) proposed to utilize the KL divergence between the distributions of the source and target domains as a regularization term. To simplify the measurement of KL divergence, it was assumed that the extracted features are sampled from the Gaussian distribution. In the comparison of prediction performance, their proposed method outperformed the above mentioned DANN, MMD, CORAL, and WDGRL.

2.2 Gradual domain adaptation (GDA)

In this section, we provide a detailed description of GDA. The formulation of self-training-based GDA method (Kumar et al., 2020) is shown in Section 2.2.1. We review some related work in Section 2.2.2 that have been inspired by Kumar et al. (2020). Other problem settings similar to GDA are introduced in Section 2.2.3.

2.2.1 Formulation of gradual self-training

Consider a multiclass classification problem. Let $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{1, 2, \ldots, C\}$ be the input and label spaces, respectively. In GDA, multiple domains are available, and each domain is represented by the superscript (j), $1 \leq j \leq K$. The source domain corresponds to $j = 1$, the target domain corresponds to $j = K$, and the others correspond to the intermediate domains. We denote the source dataset as $D^{(1)} = \{(\boldsymbol{x}_i^{(1)}, y_i^{(1)})\}_{i=1}^{n_1}$, and $U^{(j)} = \{\boldsymbol{x}_i^{(j)}\}_{i=1}^{n_j}$ represents the intermediate and the target datasets. The subscript i, $1 \leq i \leq n_j$ indicates the i-th observed datum. We note that $\{U^{(j)}\}_{j=2}^{K-1}$ are the

intermediate datasets and $U^{(K)}$ is the target dataset. When j is small, the domain is considered to be similar to the source domain. In contrast, when j is large, the domain is considered to be similar to the target domain. The intermediate domains are arranged to connect the source domain to the target domain, and their order is known or *index* is given.

We consider training the model $h^{(1)}$ by minimizing the loss on the source dataset

$$h^{(1)} = \arg\min_{h \in \mathcal{H}} \frac{1}{n_1} \sum_{i=1}^{n_1} \ell(h(\boldsymbol{x}_i^{(1)}), y_i^{(1)}), \tag{2.1}$$

where $\mathcal{H} = \{h \mid h : \mathcal{X} \to \mathcal{Y}\}$ and $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$ are a hypothesis space and a loss function, respectively. The classifier of the current domain $h^{(j)}$ is used to make pseudo-labels on the unlabeled dataset $U^{(j+1)}$ in the next domain. A self-training algorithm selects the hypothesis that minimizes the loss on the pairs of input and pseudo-labels. Let $\mathrm{ST}(h^{(j)}, U^{(j+1)})$ be a function that returns the self-trained model for $\boldsymbol{x}^{(j+1)} \in U^{(j+1)}$ by inputting the current model $h^{(j)}$ and an unlabeled dataset $U^{(j+1)}$:

$$\mathrm{ST}(h^{(j)}, U^{(j+1)}) = \arg\min_{h \in \mathcal{H}} \frac{1}{n_{j+1}} \sum_{i=1}^{n_{j+1}} \ell(h(\boldsymbol{x}_i^{(j+1)}), h^{(j)}(\boldsymbol{x}_i^{(j+1)})).$$

Self-training is only applied between adjacent domains. The output of GDA is the classifier for the target domain $h^{(K)}$. Gradual self-training is a special case of self-training where the self-training process is applied along a sequence of unlabeled datasets. The classifier $h^{(K)}$ is obtained by applying gradual self-training to the model of the source domain $h^{(1)}$ along the sequence of unlabeled datasets $U^{(2)}, \ldots, U^{(K)}$ denoted as $h^{(2)} = \mathrm{ST}(h^{(1)}, U^{(2)}), \cdots, h^{(K)} = \mathrm{ST}(h^{(K-1)}, U^{(K)})$. Gradual self-training is categorized as a domain adaptation method that transfers the trained model in the source domain. The categorization of domain adaptation methods was described in Section 2.1.

In Kumar et al. (2020), it is assumed that the discrepancy between adjacent domains is sufficiently small, and the discrepancy between domains is defined based on the Wasserstein distance. Let $p^{(j)}$ be the probability density function of the j-th domain. Kumar et al. (2020) defined the discrepancy between adjacent domains as

$$\max_{s \in \mathcal{Y}} W_\infty(p^{(j-1)}_{\boldsymbol{x}|y=s}, p^{(j)}_{\boldsymbol{x}|y=s}).$$

To discuss a generalization error bound for gradual self-training, we introduce big O notation.

Definition 2.4 (Big O notation)**.** *Given two functions f and g, there exist two positive constants n_0 and M such that $f(n) \leq Mg(n)$ always holds for $n > n_0$. This is denoted as $f(n) = O(g(n))$, indicating that the computational cost of $f(n)$ is of the order of $g(n)$. Moreover, $f(n) = \tilde{O}(g(n))$ means $f(n) = O(g(n)\log^c(g(n)))$, where c is a constant.*

Kumar et al. (2020) derived the first generalization error bound for gradual self-training. For the joint distribution $p^{(j)}_{\boldsymbol{x},y}$, the expected loss with the classifier $h^{(j)}$ is defined as

$$\epsilon^{(j)}(h^{(j)}) = \mathbb{E}_{\boldsymbol{x},y\sim p^{(j)}_{\boldsymbol{x},y}}[\ell(h^{(j)}(\boldsymbol{x}), y)].$$

For simplicity, Kumar et al. (2020) assumed that the sample size in each domain is the same, i.e., $n_2 = \cdots = n_K = n$. The expected loss of $h^{(K)}$, obtained through gradual self-training, is bounded as

$$\epsilon^{(K)}(h^{(K)}) \leq e^{O(K-1)}\left(\epsilon^{(1)}(h^{(1)}) + O\left(\sqrt{\frac{\log(K-1)}{n}}\right)\right).$$

2.2.2 Related work of gradual domain adaptation

The topic of GDA has been actively researched in recent years. Here, we review some studies of GDA. The summary of related work is shown in Table 2.1.

Chen and Chao (2021) considered a problem setting where the intermediate domains are available, but their indices are unknown. In their problem setting, the sequence of intermediate datasets $U^{(2)}, \ldots, U^{(K-1)}$ is merged into one large dataset U. They proposed a method that decomposes the large dataset U into a sequence $U^{(2)}, \ldots, U^{(K-1)}$. To simplify the decomposition of U, they assumed that the sample size in each domain is equal $n_2 = \cdots = n_{K-1} = n$. Their proposed method consists of two steps. First, it utilizes a domain discriminator, which is trained on the input data from both the source and target domains, to output the probability that a sample in U belongs to the source domain. The samples in U are sorted by this probability value. Finally, the result of the sorting is adjusted to reduce the loss of cycle consistency. The cycle consistency loss ensures that the outputs from the two classifiers $h^{(j)}$ and $\hat{h}^{(j)}$ match. The classifier $\hat{h}^{(j)}$ is obtained by

updating $h^{(j)}$ from domain j to $j+1$ using self-training and then updating it again from domain $j+1$ to j using self-training.

Wang et al. (2022) considered gradual self-training from an online learning perspective and conducted a theoretical analysis. Their assumptions are more general than those of the previous work (Kumar et al., 2020). They defined the discrepancy between adjacent domains as p-Wasserstein distance instead of the per-class ∞-Wasserstein distance. The average of discrepancies between consecutive domains is defined as

$$\rho = \frac{1}{K-1} \sum_{j=2}^{K} W_p(p_{\boldsymbol{x},y}^{(j-1)}, p_{\boldsymbol{x},y}^{(j)}). \tag{2.2}$$

To simplify the theoretical analysis, they assumed $n_2 = \cdots = n_K = n$, and derived an improved generalization bound:

$$\epsilon^{(K)}(h^{(K)}) \leq \epsilon^{(1)}(h^{(1)}) + \tilde{O}\left(\rho \cdot (K-1) + \frac{K-1}{\sqrt{n}} + \frac{1}{\sqrt{n(K-1)}}\right). \tag{2.3}$$

He et al. (2023) considered a scenario where, similar to ours, the available intermediate domains are limited. They propose generating a pseudo-intermediate domain using optimal transport (Villani, 2009) and use self-training to propagate the label information of the source domain.

Zhou et al. (2022a) assumed that they have access to the labels of the intermediate and target domains, and they proposed a semi-supervised GDA method. Their proposed method utilizes self-training and requests of queries. When updating the hypotheses, pseudo-labels are used for the less uncertain samples, while queries are requested for the more uncertain samples. They also provided a new dataset suitable for GDA.

The key idea of Kumar et al. (2020) is to update the classifier gradually through self-training. Zhang et al. (2021) and Abnar et al. (2021) proposed to apply the idea to conventional domain adaptation. Since the intermediate domains are unavailable in their problem setting, they use pseudo-intermediate domains, which are generated by mixing the samples from the source and target domains. Huang et al. (2022) proposed the application of the key idea of Kumar et al. (2020) to reinforcement learning. They proposed a method, following a similar concept to curriculum learning (Bengio et al., 2009), that starts with simple tasks and gradually introduces more challenging problems

for learning.

2.2.3 Other problem settings

Several methods have been proposed to gradually transfer the knowledge extracted from the source domain to the target domain (Gadermayr et al., 2018; Gong et al., 2019; Hsu et al., 2020; Choi et al., 2020; Cui et al., 2020; Dai et al., 2021). These methods are categorized as domain adaptation methods that extract domain-invariant features from the source and target datasets. The interest of these methods is image recognition or image classification. A sequential domain adaptation is realized by using data generated by mixing the data from the source and target domains.

Liu et al. (2020) proposed evolving domain adaptation in which a target domain evolves over time. In contrast to GDA, which has only one target domain, the evolving domain adaptation assumes that the sequence of the target domains is given and aims at achieving accurate prediction over all the target domains. They proposed a method to use the notion of meta-learning (Hospedales et al., 2022) for this problem setting and demonstrate its feasibility.

Wang et al. (2020) considered the problem where there are multiple source domains with indices, which corresponds to the GDA problem where all labels of the intermediate domains can be accessed. Dong et al. (2022) considered a supervised GDA problem where all the labels of the intermediate domains are given, and they derived the first generalization upper bound on the expected loss of $h^{(K)}$. In multi-source domain adaptation (Zhao et al., 2020), it is considered that labeled data are collected from multiple different distributions. One simple approach to utilizing multiple source datasets is combining the source datasets into a single source dataset. Since this simple approach does not provide good prediction performance, various approaches have been studied. One of the standard approaches of the multi-source domain adaptation assumes that the target distribution can be approximated by a mixture of multiple source distributions. Therefore, a combination of classifiers with weight is used for multi-source domain adaptation.

A problem setting where samples are continuously available, such as online learning, is similar to that of GDA. In online learning, the model is required to adapt to the latest samples while maintaining the long-term trend learned in the past. Ye et al. (2022)

proposed a method for temporal domain generalization in online recommendation models. While GDA aims to reduce the loss in one target domain, their goal is to reduce the losses in all domains. The availability of labeled data from all domains also differs from GDA. Zhou et al. (2022b) proposed a domain generalization method for online settings. To reduce the number of labeled samples from each domain, they utilize self-training and requests of queries.

Table 2.1: Summary of related work

Reference	Intermediate domains	Unsupervised domain adaptation	# of domains to be adapted
Ours (Chapter 3)	limited available	No	single
Ours (Chapter 4)	limited available	Yes	single
Kumar et al. (2020)	available	Yes	single
He et al. (2023)	limited available	Yes	single
Abnar et al. (2021)	unavailable	Yes	single
Zhang et al. (2021)	unavailable	Yes	single
Liu et al. (2020)	available	Yes	multiple
Dong et al. (2022)	available	No	single
Zhou et al. (2022a)	available	No	single
Wang et al. (2020)	available	No	single
Ye et al. (2022)	available	No	multiple
Zhou et al. (2022b)	available	No	multiple

3

Gradual domain adaptation with multifidelity learning

3.1 Contributions

Our subject concerns a GDA problem characterized by large discrepancies between adjacent domains due to the limited number of accessible intermediate domains. In this chapter, we consider a problem setting of semi-supervised GDA, in which we can query the label of a sample from the intermediate domains and the target domain. We refer to the proposed method as a semi-supervised GDA method since it utilizes the information of unlabeled samples through the querying and annotation of $\boldsymbol{x}_i^{(j)} \in U^{(j)}$. As described in Chapter 1, the query cost is highest in the target domain, and the query cost in the intermediate domain is higher when it is closer to the target domain and lower when it is closer to the source domain. While we can obtain a reliable model by querying the labels of numerous samples from the target domain, this approach is not cost-effective, and there is a strong demand for methods to utilize the knowledge

of the source domain. Rai et al. (2010) proposed Domain-Separator based Active Online Domain Adaptation (DS-AODA) which combines domain adaptation with active learning. Their proposed method can utilize both the knowledge extracted from the source domain and the informative samples from the target domain, which are selected by an active learning algorithm. We can implement DS-AODA as GDA by executing them sequentially. However, during the process of sequential domain adaptation, the budget may be exhausted because DS-AODA does not consider the query cost across all domains. To address this issue, we propose to use the notion of multifidelity learning which aim to determine the number of query samples by considering both its query cost and the usefulness of the label. We use multifidelity active learning, which combines active learning and multifidelity learning, to collect labeled samples efficiently. The contributions of this chapter are as follows.

- We develop a GDA method for sequential domains that have different query costs.
- We utilize the notion of multifidelity learning to mitigate the issue of exhausting the budget before querying from the target domain, which is observed when a conventional domain adaptation method is executed sequentially.
- We propose to combine multifidelity learning with active learning to efficiently collect labeled samples since a multifidelity learning algorithm only distributes the budget to each domain.
- We demonstrate that the proposed method achieves a higher prediction performance than a method that uses a model trained only with the target data.

3.2 Multifidelity learning

We consider evaluating a function f that requires a high cost to obtain the output of f, and we call this function a high-fidelity model. In real-world tasks, it is important to obtain a surrogate model of f or to find the input where the output of f is maximized (or minimized). However, due to the high cost of evaluating f, the necessary budget becomes substantial. Multifidelity learning aim to reduce the necessary budget by using the low-fidelity models, which have lower evaluation costs than f and whose outputs are less reliable.

Here, we show a concrete example of multifidelity learning. In materials science, it is known that experimental costs are high, and computational simulations, such as density functional theory (DFT) (Kohn and Sham, 1965), are utilized to accelerate the design of new materials. A chemical structure is input into DFT, which then simulates the properties of the chemical compound. The accuracy and computational cost of DFT are determined by the functional used. For instance, the Perdew-Burke-Ernzerhof (PBE) (Perdew et al., 1996) functional has low accuracy and low computational cost, while the Strongly Constrained and Appropriately Normed (SCAN) (Sun et al., 2015) functional has high accuracy and high computational cost. Additionally, real-world experiments are considered the most accurate and the most costly method to obtain labels. Gong et al. (2022) demonstrated the feasibility of predicting the formation enthalpy, a fundamental property determining the thermodynamic stability of materials, by utilizing both DFT results (low-fidelity data) and experimental results (high-fidelity data). In the multifidelity learning considered by Gong et al. (2022), it is not assumed that low-fidelity and high-fidelity data follow different probability distributions. Moreover, Gong et al. (2022) did not propose to increase the number of the input-output pairs for each fidelity through active learning in a multifidelity setting. It is known that compounds with a large number of atoms require high computational costs to obtain the calculation results of DFT, and using a functional with lower computational costs, such as PBE, for these compounds is reasonable. As shown in Figure 3.1, when the low-fidelity dataset is composed of compounds with a large number of atoms and the high-fidelity dataset is composed of compounds with a small number of atoms, the change in fidelity can be seen to change the probability distribution. We consider a semi-supervised GDA problem under the assumption that the probability distribution changes gradually as the fidelity changes. Table 3.1 summarizes the difference in problem settings.

Table 3.1: Summary of problem settings

Reference	Probability distribution	Query cost
Multifidelity learning (Peherstorfer et al., 2018)	no change	consider
Active domain adaptation (Rai et al., 2010)	change	not consider
Ours	change	consider

In multifidelity learning, the training dataset consists of various pairs of inputs and outputs with different fidelities. To collect training dataset efficiently, active learning and

Bayesian optimization are applied to multifidelity learning (Takeno et al., 2020; Li et al., 2020, 2022). While the acquisition functions proposed in these studies are designed to improve the performance of the high-fidelity model, our acquisition function is designed to improve the performance of the classifier in each domain.

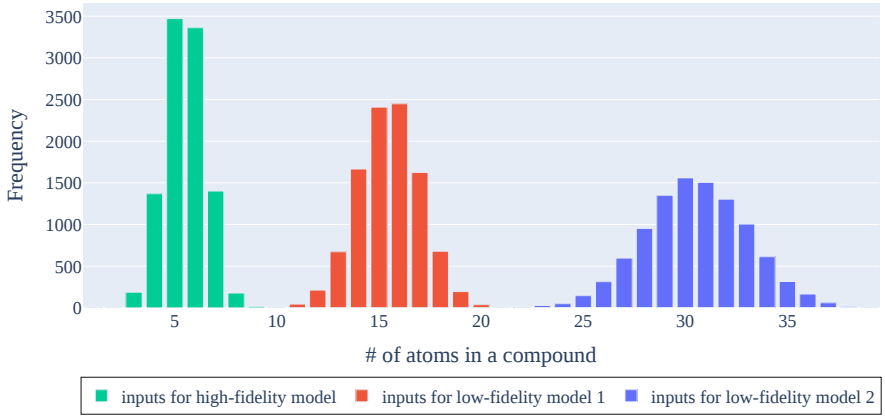


Figure 3.1: A concrete example of our considered problem.

3.3 Proposed method

3.3.1 Semi-supervised gradual domain adaptation

In our problem setting of semi-supervised GDA, we have a few labeled samples from both the intermediate and target domains. When a sample $\boldsymbol{x}_i^{(j)}$ from the unlabeled pool dataset $U^{(j)}$ is queried and annotated $y_i^{(j)}$, the labeled dataset in each domain is updated as

$$D^{(j)} \leftarrow D^{(j)} \cup \{(\boldsymbol{x}_i^{(j)}, y_i^{(j)})\}. \tag{3.1}$$

In this chapter, $\tilde{n}_j$ labeled samples from each domain are initially given, and we add more labeled samples by using the notion of multifidelity active learning. We describe the details of multifidelity active learning in Section 3.3.2. The proposed method sequentially updates the model of the source domain $h^{(1)}$ using the sequence of labeled datasets $D^{(2)}, \ldots, D^{(K)}$ to obtain a model for each domain $h^{(2)}, \ldots, h^{(K)}$. We consider updating the model for the current domain $h^{(j)}$ with the labeled samples $D^{(j+1)}$ from the

adjacent domain, and $\mathrm{SL}(D^{(j+1)})$ denotes the output of model training:

$$\mathrm{SL}(D^{(j+1)}) = \arg\min_{h \in \mathcal{H}} \frac{1}{\tilde{n}_{j+1}} \sum_{i=1}^{\tilde{n}_{j+1}} \ell(h(\boldsymbol{x}_i^{(j+1)}), y_i^{(j+1)}). \tag{3.2}$$

We update the model of the source domain $h^{(1)}$ using $D^{(2)}$ to obtain $h^{(2)}$, and sequentially update the model $h^{(j)}$ using $D^{(j+1)}$ until $h^{(K)}$ is obtained:

$$h^{(2)} = \mathrm{SL}(D^{(2)}), \ldots, h^{(K)} = \mathrm{SL}(D^{(K)}).$$

These models $h^{(1)}, \ldots, h^{(K)}$ are utilized to determine the number of queries from each domain. When training the model $h^{(j+1)}$, we use the parameters of $h^{(j)}$ as initial values for the parameters of $h^{(j+1)}$. This technique is known as warm-starting (Erhan et al., 2010). While self-training-based GDA methods use pseudo-labels to transfer knowledge from the source domain to the target domain via the intermediate domains, we transfer the knowledge by utilizing warm-starting. Since labeled data from the intermediate and target domains are limited, we aim to efficiently train a non-convex model by utilizing warm-starting. We demonstrate the effectiveness of using warm-starting in Section 3.4.3.

Generally, in semi-supervised learning, both unlabeled data and labeled data are used during the model training. We prepared the pseudo-labels using the inputs $\boldsymbol{x}^{(j+1)} \in U^{(j+1)}$ and attempted to utilize both unlabeled and labeled data. However, the performance of our proposed method did not improve. It is known that pseudo-labels do not work in our considered situation where the discrepancies between the adjacent domains are large (Liu et al., 2021).

3.3.2 Multifidelity active learning

We are considering a problem setting where querying needs some cost $c^{(j)} \in \mathbb{R}_+$, $\boldsymbol{c} = (c^{(2)}, \ldots, c^{(K)})^\top$. Moreover, the cost of obtaining a label from the intermediate domain is higher when it is closer to the target domain and lower when it is closer to the source domain $c^{(2)} < \cdots < c^{(K)}$. Let $m^{(j)} \in \mathbb{N}$, $\boldsymbol{m} = (m^{(2)}, \ldots, m^{(K)})^\top$ be the number of query samples from the unlabeled dataset in the j-th domain. The cost $\boldsymbol{c}$ is given according to the problem, while the number of query samples $\boldsymbol{m}$ is determined from the dataset as explained later. Finally, there is a pre-fixed budget $B \in \mathbb{R}$, and the total cost cannot

exceeds the budget, $\boldsymbol{m}^\top \boldsymbol{c} \leq B$.

On the basis of the multifidelity Monte Carlo (MFMC) method (Peherstorfer et al., 2016), we determine the number of query samples $\boldsymbol{m}$ in each domain by distributing budgets considering cost and fidelity. The goal of MFMC is to estimate $u = \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}}}[f^{(K)}(\boldsymbol{x})]$, where $f^{(K)} : \mathbb{R}^d \supset \mathcal{X} \to \mathcal{Z} \subset \mathbb{R}$ is the high-fidelity model that requires the highest cost $c^{(K)}$ to obtain the output. In multifidelity learning, the functions $f^{(2)}, \ldots, f^{(K-1)} : \mathcal{X} \to \mathcal{Z}$ with lower evaluation cost than $c^{(K)}$ are available, and the number of evaluations of each function satisfies $m^{(2)} \geq \cdots \geq m^{(K)}$. Let $\bar{z}(f, m) = \frac{1}{m}\sum_{i=1}^{m} f(\boldsymbol{x}_i)$ be the empirical estimator of $\mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}}}[f(\boldsymbol{x})]$ and σ_s^2 be the variance of $f^{(s)}(\boldsymbol{x})$. By utilizing the low-fidelity models $f^{(2)}, \ldots, f^{(K-1)}$, the MFMC estimator $\hat{u}$ is defined as

$$\hat{u} = \bar{z}(f^{(K)}, m^{(K)}) + \sum_{s=2}^{K-1} \alpha_s (\bar{z}(f^{(s)}, m^{(s)}) - \bar{z}(f^{(s)}, m^{(s+1)})),$$

where $\alpha_2, \ldots, \alpha_{K-1} \in \mathbb{R}$ are coefficients that weight the differences $\bar{z}(f^{(s)}, m^{(s)}) - \bar{z}(f^{(s)}, m^{(s+1)})$. Note that while $\bar{z}(f^{(s)}, m^{(s)})$ and $\bar{z}(f^{(s)}, m^{(s+1)})$ evaluate the same low-fidelity model, they have different numbers of samples. We denote Pearson's correlation coefficient between the outputs of $f^{(K)}$ and $f^{(s)}$ as λ_s. Furthermore we define $\lambda_s = 0$ for $s < 2$. The MFMC estimator $\hat{u}$ is an unbiased estimator of u (Peherstorfer et al., 2016, Lemma 3.1) and Peherstorfer et al. (2016) derived the mean squared error of $\hat{u}$ with respect to u as

$$\mathbb{E}[(u - \hat{u})^2] = \frac{\sigma_K^2}{m^{(K)}} + \sum_{s=2}^{K-1} \left(\frac{1}{m^{(s+1)}} - \frac{1}{m^{(s)}} \right) (\alpha_s^2 \sigma_s^2 - 2\alpha_s \lambda_s \sigma_1 \sigma_s). \tag{3.3}$$

We define the vector $\tilde{\boldsymbol{m}} = (\tilde{m}^{(2)}, \ldots, \tilde{m}^{(K)})^\top, \tilde{m}^{(s)} \in \mathbb{R}$ and solve the following optimization problem to determine the number of query samples $\tilde{m}^{(2)}, \ldots, \tilde{m}^{(K)}$ and coefficients

$\alpha_2, \ldots, \alpha_{K-1}$ that minimize Eq. (3.3) without exceeding the budget B:

$$\begin{aligned} \min_{\substack{\tilde{m}^{(2)},\ldots,\tilde{m}^{(K)} \in \mathbb{R}, \\ \alpha_2,\ldots,\alpha_{K-1} \in \mathbb{R}}} & \frac{\sigma_K^2}{\tilde{m}^{(K)}} + \sum_{s=2}^{K-1} \left(\frac{1}{\tilde{m}^{(s+1)}} - \frac{1}{\tilde{m}^{(s)}} \right) (\alpha_s^2 \sigma_s^2 - 2\alpha_s \lambda_s \sigma_1 \sigma_s) \\ \text{subject to } & \tilde{m}^{(s)} - \tilde{m}^{(s+1)} \leq 0, \ s = 2, \ldots, K-1, \\ & -\tilde{m}^{(K)} \leq 0, \\ & \tilde{\boldsymbol{m}}^\top \boldsymbol{c} = B. \end{aligned} \tag{3.4}$$

When the models $f^{(2)}, \ldots, f^{(K)}$ and the costs $c^{(2)}, \ldots, c^{(K)}$ satisfy the following assumption, the optimization problem given by Eq. (3.4) has an analytic solution.

Assumption 3.1 (Peherstorfer et al. (2016))**.** *The models $f^{(2)}, \ldots, f^{(K)}$ satisfy*

$$|\lambda_2| < \cdots < |\lambda_K| \tag{3.5}$$

and the costs $c^{(2)}, \ldots, c^{(K)}$ satisfy the ratios

$$\frac{c^{(s+1)}}{c^{(s)}} > \frac{\lambda_{s+1}^2 - \lambda_s^2}{\lambda_s^2 - \lambda_{s-1}^2}, \quad s = 2, \ldots, K-1. \tag{3.6}$$

The optimal coefficients are given by $\alpha_s = \lambda_s \sigma_K / \sigma_s, s = 2, \ldots, K-1$. The optimal numbers of query samples are

$$\tilde{m}^{(K)} = \frac{B}{\boldsymbol{r}^\top \boldsymbol{c}}, \ \tilde{m}^{(s)} = r^{(s)} \tilde{m}^{(K)}, \ s = 2, \ldots, K-1, \tag{3.7}$$

where the components of the vector $\boldsymbol{r} = (r^{(2)}, \ldots, r^{(K-1)}, 1)^\top$ are given by

$$r^{(s)} = \sqrt{\frac{c^{(K)}(\lambda_s^2 - \lambda_{s-1}^2)}{c^{(s)}(1 - \lambda_{K-1}^2)}}, \ s = 2, \ldots, K-1. \tag{3.8}$$

We round down $\tilde{m}^{(2)}, \ldots, \tilde{m}^{(K)}$ to obtain an integer number and denote the optimal number of queries as

$$\boldsymbol{m} = (m^{(2)}, \ldots, m^{(K)})^\top = (\lfloor \tilde{m}^{(2)} \rfloor, \ldots, \lfloor \tilde{m}^{(K)} \rfloor)^\top.$$

We discuss the details of Eq. (3.8). When $c^{(K)}/c^{(s)}$ is large, in other words, when $c^{(s)}$ is small, $r^{(s)}$ becomes large. Moreover, when $s = 2$, $\lambda_s - \lambda_{s-1}$ is large since $\lambda_{s-1} = 0$, which results in $r^{(s)}$ also becoming large. Since $c^{(2)} < \cdots < c^{(K)}$, the number of query samples to the function with the lowest fidelity $f^{(2)}$ becomes the largest.

While our subject differs from that of MFMC, it is reasonable to determine the number of query samples based on the correlations between models $h^{(2)}, \ldots, h^{(K)}$ and costs $c^{(2)}, \ldots, c^{(K)}$ in our problem setting. We determine the number of query samples from each domain according to Eqs. (3.7) and (3.8). Since our classifier $h : \mathcal{X} \to \mathcal{Y}$ is different from the function $f : \mathcal{X} \to \mathcal{Z}$ used in MFMC, we use Matthews correlation coefficient (Matthews, 1975) instead of Pearson's correlation coefficient. The Matthews correlation coefficient aims to evaluate a confusion matrix. Gorodkin (2004) derived Matthews correlation coefficient for multi-class classification from the definition of Pearson's correlation coefficient (see Appendices for details). While a confusion matrix is generally calculated using the true labels and the predicted labels, our confusion matrix is calculated using the predicted labels $h^{(K)}(\boldsymbol{x}^{(K)})$ and $h^{(s)}(\boldsymbol{x}^{(K)})$. The confusion matrix is denoted as

$$M = \begin{bmatrix} M_{11} & \cdots & M_{1C} \\ \vdots & \ddots & \vdots \\ M_{C1} & \cdots & M_{CC} \end{bmatrix},$$

where M_{ij} corresponds to the total number of samples that $h^{(K)}$ predicts as the i-th class, while $h^{(s)}$ predicts them as the j-th class. To simplify the notation of the definition of Matthews correlation coefficient, we introduce the following variables.

- The total number of samples where the predicted labels of the two classifiers match: $T = \sum_{i=1}^{C} M_{ii}$.
- The total number of elements in M: $N = \sum_{i=1}^{C} \sum_{j=1}^{C} M_{ij}$.
- The sum of the elements in the k-th row of M: $R_k = \sum_{i=1}^{C} M_{ki}$.
- The sum of the elements in the k-th column of M: $A_k = \sum_{i=1}^{C} M_{ik}$.

We calculate Matthews correlation coefficient as

$$\frac{T \times N - \sum_{k=1}^{C} R_k \times A_k}{\sqrt{(N^2 - \sum_{k=1}^{C} R_k^2)(N^2 - \sum_{k=1}^{C} A_k^2)}}.$$

Finally, we determine the query sample from the unlabeled pool dataset $U^{(j)}$ by active learning (Settles, 2009; Hino, 2020). The samples to be queried are determined on the basis of uncertainty (Ramirez-Loaiza et al., 2017), which is a standard measure in active learning. To calculate the uncertainty of a sample, we use a classifier h which outputs the conditional probability $p(y = l \mid \boldsymbol{x}), l \in \mathcal{Y}$ and predicts the class label of the sample as $\hat{y} = \arg\max_{l \in \mathcal{Y}} p(y = l \mid \boldsymbol{x})$. The uncertainty of a sample is computed as

$$\mathrm{unc}(\boldsymbol{x}^{(j)}) = 1 - \max_{l \in \mathcal{Y}} p_{h^{(j)}}(y = l \mid \boldsymbol{x}^{(j)}), \tag{3.9}$$

where $p_{h^{(j)}}$ represents the conditional probability under the classifier $h^{(j)}$. Then, we query for the sample $\boldsymbol{x}_*^{(j)}$ with the largest uncertainty:

$$\boldsymbol{x}_*^{(j)} = \arg\max_{\boldsymbol{x}_i^{(j)} \in U^{(j)}} \mathrm{unc}(\boldsymbol{x}_i^{(j)}). \tag{3.10}$$

We describe the outline of our algorithm below, and the details of our algorithm are described in Algorithm 3.1.

1. We train the model $h^{(1)}$ with the source dataset.

2. We update the model $h^{(1)}$ with the sequence of labeled datasets $D^{(2)}, \ldots, D^{(K)}$ and obtain the models for all domains $h^{(2)}, \ldots, h^{(K)}$.

3. Until the budget runs out, we query labels. We update all models $h^{(2)}, \ldots, h^{(K)}$ and recalculate the optimal number of queries $m^{(2)}, \ldots, m^{(K)}$ whenever one sample is queried from all domains. For an ablation study, we use random sampling instead of uncertainty sampling.

3.4 Experiments

We evaluate the efficiency of the proposed method on four real-world datasets, and Pytorch (Paszke et al., 2019) is used for all implementation. All experiments are conducted on our server with Intel Xeon Gold 6354 processors and NVIDIA A100 GPU. For image datasets, the classifier h was composed of three convolutional layers,

Algorithm 3.1 Gradual domain adaptation with multifidelity active learning

Input: datasets $D^{(1)}, \ldots, D^{(K)}$, and $U^{(2)}, \ldots, U^{(K)}$, budget B, cost $\boldsymbol{c}$
Output: models $h^{(1)}, \ldots, h^{(K)}$, updated datasets $D^{(2)}, \ldots, D^{(K)}$

train the model $h^{(1)}$ as Eq. (2.1)
initialize the number of queried samples from each domain $\bar{m}^{(2)}, \ldots, \bar{m}^{(K)} \leftarrow 0$
while $B > 0$ **do**
    for each $j \in \{2, \ldots, K\}$ **do**
        train the model on the next domain $j + 1$ with warm-starting
        minimize the loss given by Eq. (3.2)
    end for
    compute the optimal number of queries from each domain $m^{(2)}, \ldots, m^{(K)}$
    for each $j \in \{2, \ldots, K\}$ **do**
        if $\bar{m}^{(j)} < m^{(j)}$ and $B > c^{(j)}$ **then**
            compute the uncertainty of the samples in $U^{(j)}$ as Eq. (3.9)
            select the sample for query $\boldsymbol{x}_*^{(j)}$ by Eq. (3.10)
            query the label of $\boldsymbol{x}_*^{(j)}$, and obtain $y_*^{(j)}$
            update labeled dataset as Eq. (3.1)
            $D^{(j)} \leftarrow D^{(j)} \cup \{(\boldsymbol{x}_*^{(j)}, y_*^{(j)})\}$
            $\bar{m}^{(j)} \leftarrow \bar{m}^{(j)} + 1$
            $B \leftarrow B - c^{(j)}$
        end if
    end for
end while

one dropout, and one batch normalization layer. For tabular datasets, the classifier h consisted of two fully connected layers, one dropout, and one batch normalization layer.

3.4.1 Self-training-based GDA

We demonstrate that the existing GDA method (Kumar et al., 2020), which is based on self-training, is ineffective when there are large discrepancies between the adjacent domains. The `two-moon` dataset is used as the source dataset, and the target dataset is prepared by rotating the source dataset by $\pi/2$. The size of the source dataset n_1 is 2,000. We adjust the discrepancies between adjacent domains by varying the number of intermediate domains. For example, when the number of intermediate domains is two, the rotation angles of the intermediate domains are $\pi/6$ and $\pi/3$. We assign the minimum and the maximum number of the intermediate domain to one and nineteen, respectively. In Eq. (2.2), the average of discrepancies between consecutive domains is defined as

$$\rho = \frac{1}{K-1} \sum_{j=2}^{K} W_p(p_{\boldsymbol{x},y}^{(j-1)}, p_{\boldsymbol{x},y}^{(j)}).$$

When the number of intermediate domains is changed, we compute ρ.

Table 3.2 shows the results of twenty repetitions with different initial weights of neural networks. The prediction performance of the previous method deteriorates when ρ is large. In Table 3.2, the ρ of the maximum and the minimum number of the intermediate domains are shown as one and zero, respectively. The purpose of this experiment is to confirm that the accuracy of the self-training-based method deteriorates when ρ is large. We can confirm that self-training-based GDA is unsuitable when ρ is higher than approximately 0.3.

3.4.2 Datasets

The existing datasets are modified to emulate a GDA situation. Most of the datasets used in this chapter are adopted from existing work (Kumar et al., 2020). We describe the details of the datasets.

Table 3.2: The prediction performance of the self-training-based GDA method deteriorates when the average value of the distances between adjacent domains is large.

# of intermediate domains	ρ	Accuracy
3	1.000	0.459 ± 0.024
6	0.343	0.457 ± 0.024
9	0.173	0.537 ± 0.092
12	0.094	0.631 ± 0.095
15	0.050	0.717 ± 0.033
18	0.021	0.775 ± 0.123
21	0.000	0.901 ± 0.127

`Two Moon`**:** The `two-moon` dataset is used as the source dataset, and the intermediate dataset and the target dataset are prepared by rotating the source dataset by $\pi/4$ and $\pi/2$, respectively. All datasets have 2,000 samples.

`Rotating MNIST` **(Kumar et al., 2020):** We use 2,000 unrotated images from the `MNIST` dataset as the source dataset. The intermediate dataset of 42,000 samples gradually adds rotations from $\pi/63 - \pi/3.15$. Then, the intermediate dataset is divided into twenty parts. We prepare the target dataset of 2,000 samples with added rotations of $\pi/3$.

`Portraits` **(Ginosar et al., 2015):** A dataset of actual photos of American high school students taken from 1905 to 2013 is used, and the task is to guess the gender from the images. The dataset is sorted in descending order by year, with source, intermediate, and target datasets beginning with the newest. Each dataset has 2,000 samples, and the intermediate has six domains.

`Cover Type` **(Blackard and Dean, 1999):** The task is to guess the type of plants in a group from 54 features. Following the previous work (Kumar et al., 2020), we consider the binary classification problem by extracting only Spruce/Fir and Lodgepole Pine, which are the majority of the seven objective variables. The source, intermediate, and target datasets are created in descending order of distance from the water body. There are eight intermediate domains, and each dataset has 50,000 samples except for the target dataset. The target dataset has 30,000 samples.

`Gas Sensor` **(Vergara et al., 2012; Rodriguez-Lujan et al., 2014):** This dataset contains the measurements from the 128 sensors exposed to different concentrations of gases, and the task is to guess the six types of chemicals from the responses of the sensors. This dataset is divided into batches based on the time the gas is exposed. The degradation of the sensors causes a change in the distribution of the input. The last 3,600 samples are excluded because they are measured in very different environments.[1] In ascending order of batch, the source, intermediate and target datasets are used. Each dataset has 3,000 samples, and we set the number of intermediate domains as one.

Since these datasets are used in the previous work (Kumar et al., 2020), except the `Two Moon` and the `Gas Sensor`, the discrepancies between the adjacent domains are significantly small. Hence, we can apply self-training-based GDA method. On the other hand, we consider the problem where the discrepancies between the adjacent domains are large. Thus, we restrict the number of accessible intermediate domains in each dataset. In practice, we cannot select arbitrary intermediate domains for GDA. The sequence of intermediate domains is only given.

In this chapter, we randomly select an arbitrary number of intermediate domains from each dataset. The arbitrary number is determined by ρ. For example, when the number of accessible intermediate domains is five, we select five intermediate domains randomly and calculate ρ. We show the calculation result with various numbers of accessible intermediate domains for each dataset in Figure 3.2. Note that the distances are scaled between zero and one for each dataset. In Section 3.4.1, we already confirmed that $\rho \simeq 0.5$ is unsuitable for gradual self-training. The number of accessible intermediate domains for each dataset is selected as two or three, which leads to $\rho \simeq 0.5$. We should select the intermediate domains that satisfy Eq. (3.5). In Figure 3.3, we show the calculation result of the correlation coefficient between the outputs of models $h^{(K)}, h^{(s)}$, which are trained on the input-output pairs in each domain. In Section 3.4.3 and 3.4.4, we randomly select two or three intermediate domains which satisfy Eq. (3.5). During the training of the proposed method, sometimes the value of r given by Eq. (3.8) becomes negative. In such cases, we query one sample to all domains. Recall that the proposed method recalculates the number of query samples $m^{(2)}, \ldots, m^{(K)}$ each time the labeled datasets $D^{(2)}, \ldots, D^{(K)}$ are updated.

We set the costs $c^{(2)}, \ldots, c^{(K)}$ to satisfy Eq. (3.6) after selecting the intermediate

[1] See Vergara et al. (2012) for details.

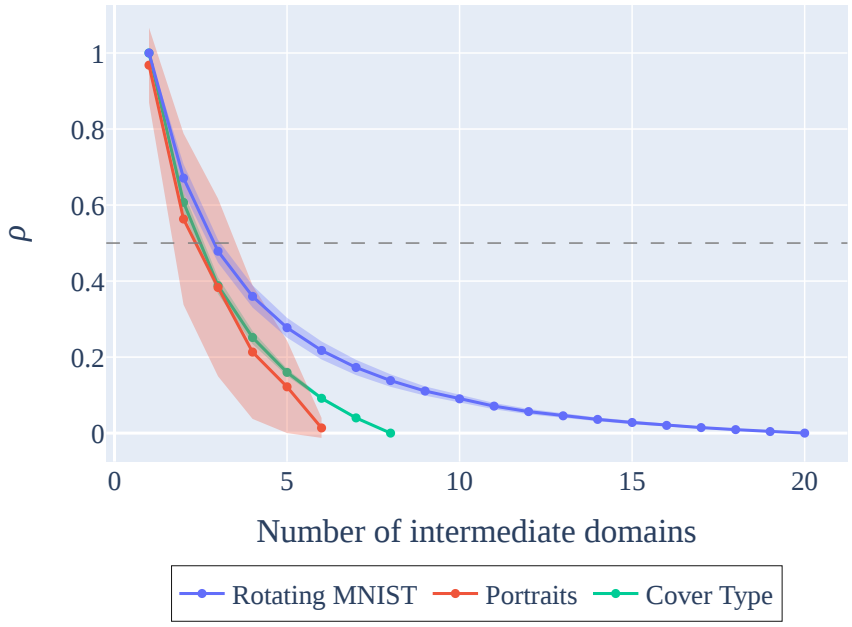


Figure 3.2: The average value of the distances between adjacent domains varies with the number of accessible intermediate domains. We randomly select an arbitrary number of intermediate domains from each dataset and calculate ρ.

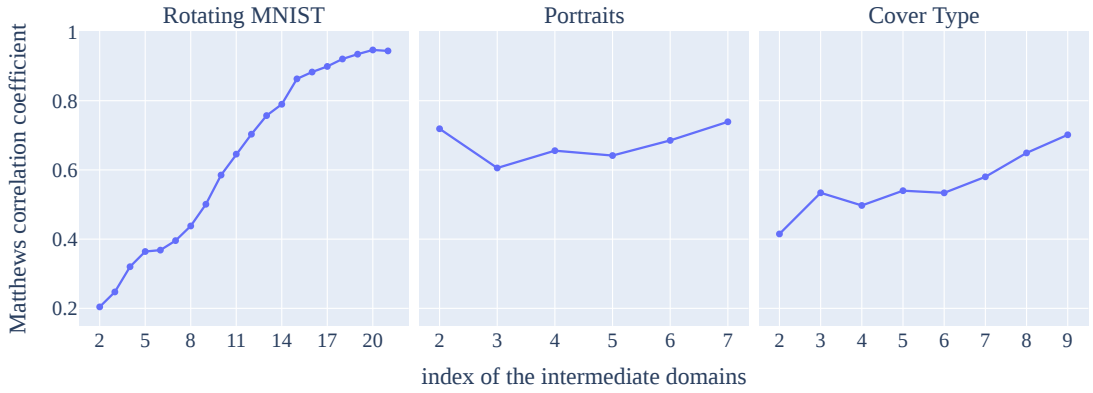


Figure 3.3: Calculation results of Matthews correlation coefficients between the outputs of models $h^{(K)}, h^{(s)}$.

domains. It is unnatural to assume that we can obtain the same number of labeled samples from the target domain as that of the source domain since we consider the scenario that the query cost is highest in the target domain. We set the maximum number of labeled samples from the target domain $\tilde{n}_K$ as 50 and set the budget B as $\tilde{n}_K \times c^{(K)}$. Apart from the budget, we randomly select initial samples from each unlabeled dataset. We can consider various situations and settings of the number of initial samples. Supposing that the number of labeled initial samples is limited, we set the number of initial samples to 1% of the size of the source dataset n_1. In `Cover Type`, we assign the number of initial samples to 0.1% of the source dataset size $n_1/1000$ since the size of the source dataset is large.

3.4.3 Ablation study

The effectiveness of our proposed method is verified by the ablation study. We describe the details of each ablation study below.

- w/o active learning (AL): The query sample is determined at random.
- w/o intermediate: Our algorithm runs only with the source and the target dataset.
- w/o AL/intermediate: Our algorithm runs without intermediate datasets, and the query sample is determined randomly. We apply the warm-starting when the training of models.
- w/o warm-starting: We do not utilize warm-starting during model training, and instead initialize the model parameters with a random value. This approach is tantamount to training the model exclusively on the target dataset without using any domain adaptation techniques.

We show the result of the ablation study in Figure 3.4. The prediction performances deteriorated in the cases of w/o AL, w/o intermediate, and w/o AL/intermediate on the `Two Moon` dataset. It suggests that the contribution of the intermediate domains is significant. In contrast, the deterioration of prediction performance of w/o warm-starting is slight. We consider that the prediction performance of w/o warm-starting on the `Two Moon` dataset does not deteriorate due to the simplicity of predicting the `Two Moon` dataset.

The prediction performances of w/o warm-start deteriorated on real-world datasets. Since real-world datasets are not easy to predict, the proposed method, which utilizes the knowledge of the source domain and the intermediate domains, achieves better prediction performance than w/o warm-starting. It is a reasonable result that the prediction performance of w/o AL is comparable or inferior to that of our proposed method on real-world datasets. Especially on the `Portraits` dataset, w/o intermediate has comparable or superior accuracy to our proposed method. The experimental result suggests that the given intermediate domains are not arranged to connect the source domain to the target domain, and the intermediate domains are unsuitable for GDA. Chapter 5 provides further discussion on the intermediate domains that are not suitable for GDA.

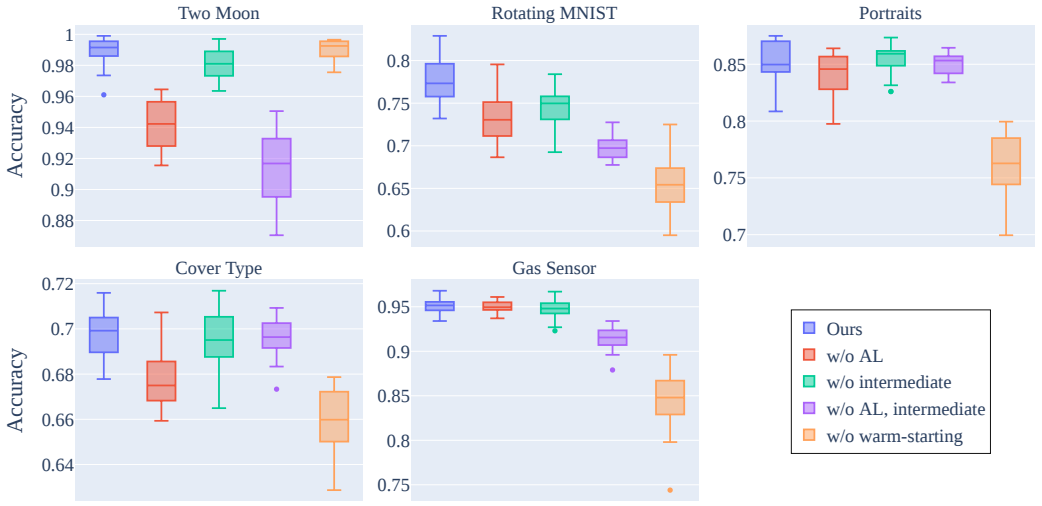


Figure 3.4: The result of the ablation study on synthetic and real-world datasets. In the `Two Moon` dataset, we confirm the degradation of prediction performance in three ablation studies. In real-world datasets, the prediction performance of w/o warm-starting degrades.

For further discussion, Figure 3.5 shows the number of queried samples from each domain on the maximum budget. As shown in Eq. (3.8), the multifidelity algorithm sets the number of queries from each domain by considering the correlation between the output of the classifier for the target domain $h^{(K)}(\boldsymbol{x}^{(K)})$ and the output of the classifier for the arbitrary intermediate domain $h^{(s)}(\boldsymbol{x}^{(K)})$. While the MFMC algorithm ensures $m^{(2)} \geq \cdots \geq m^{(K)}$, in the `Rotating MNIST`, `Portraits`, and `Cover`

`Type` datasets, $m^{(K)}$ is larger than $m^{(K-1)}$. This is because r given by Eq. (3.8) becomes negative during the training of the proposed method, and the proposed method queries one sample to all domains. Running our algorithm without warm-starting increases the number of cases where r is negative. Consequently, the number of queries from the target domain becomes large.

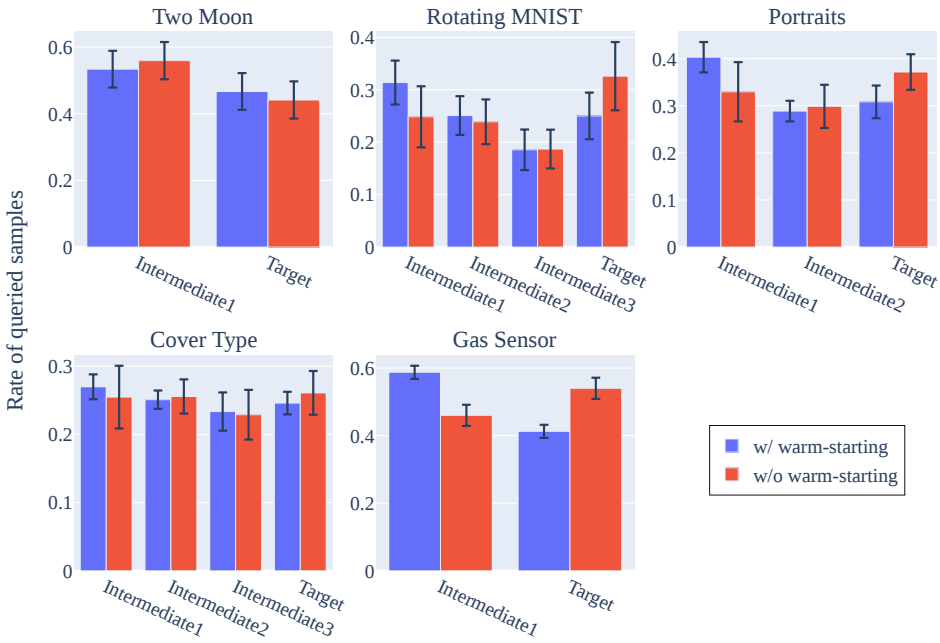


Figure 3.5: The number of queried samples from each domain with the maximum budget. When we run our algorithm without warm-starting, the number of queries from the target domain is large since the correlations between the models are small.

3.4.4 Comparison with baseline method

Finally, we show that the proposed method can make more accurate predictions than existing baseline methods. DS-AODA (Rai et al., 2010), which is a previous work on active domain adaptation, is not a method for GDA, but it is used for comparison by updating the model step by step. While GradualSelfTrain (Kumar et al., 2020) is an unsupervised domain adaptation method, we include it in the comparative study since our algorithm is inspired by this work. Zhou et al. (2022a) also proposed a GDA method. Although their proposed method requires four hyperparameters, it is not clear how those

hyperparameters should be selected.

The experimental setup for each baseline method is shown below.

- TargetOnly: We randomly select data from the target dataset and train a model.
- DS-AODA (Rai et al., 2010): Update the model by querying labels from the neighboring domains in order from the source domain to the target domain. The process ends when the budget is exhausted.
- GradualSelfTrain (Kumar et al., 2020): Apply gradual self-training with the initially given domains. It does not depend on the budget.

All the settings of experiments, such as model composition, query cost, and accessible intermediate domains, are the same as those of the proposed method. The baseline methods, except for TargetOnly, also utilize a technique of warm-starting as in the proposed method. We show the results of twenty repetitions, varying the selected intermediate domain, initial samples from each domain, and initial weights of the neural network. The DS-AODA requires the probability for querying as a hyperparameter. We evaluate three levels of different hyperparameters and show the results with the highest performance.

We show the result with various budgets for each dataset in Figure 3.6. The proposed method achieves better prediction performance than TargetOnly on all datasets, and we confirm that our proposed method is suitable for a problem with query cost.

Figure 3.7 shows the comparison results between the proposed and the baseline methods. In the case of the method that requests a query, we show the results with the maximum budget. We see that the proposed method is comparable to or superior to the baseline methods on all datasets. On the `Rotating MNIST`, `Portraits`, and `Cover Type` datasets, the predictive performance of DS-AODA is inferior to that of the proposed method. This is because the budget is depleted before a query can be issued to label a sample from the target domain, consequently compromising the dataset's accuracy. Conversely, DS-AODA achieves superior results on the `Two Moon` and `Gas Sensor` datasets, where it is able to request queries from the target domain. GradualSelfTrain produces reasonable results on the `Portraits` and `Cover Type` datasets, but does not achieve reasonable results on all datasets.

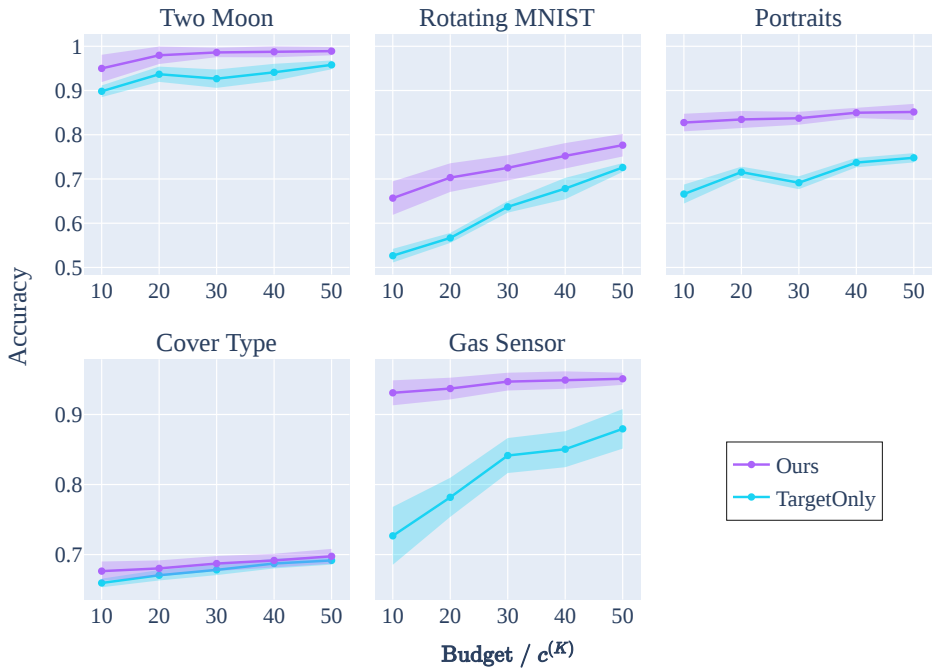


Figure 3.6: Experimental results with various budgets.

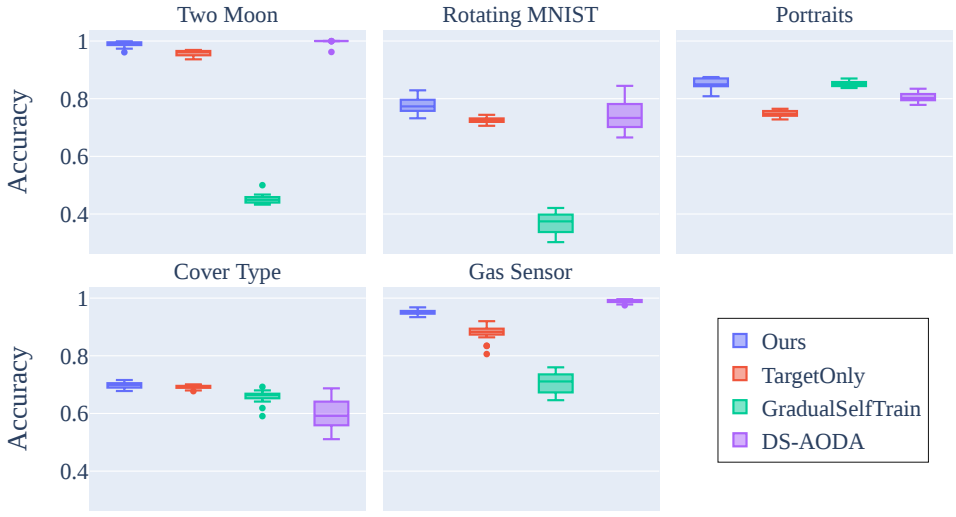


Figure 3.7: Comparison of classification accuracy on synthetics and real-world datasets.

3.5 Discussion

In this chapter, we studied a problem setting of semi-supervised GDA, in which we can query the label of a sample from the intermediate domains and the target domain. The characteristic of our problem setting is that the query cost from each domain is not uniform. The query cost is highest in the target domain, and the query cost in the intermediate domain is higher when it is closer to the target domain and lower when it is closer to the source domain. Our query strategy is determined based on the mulitifidelity algorithm. Figure 3.6 shows that our query strategy is more suitable for our problem setting than a simple query strategy, which devotes the entire budget to queries from the target domain. The proposed method has demonstrated stable and reasonable performance, making it a strong candidate for semi-supervised GDA under a multifidelity setting. In our experiments, we employed a sequential application of the DS-AODA algorithm for GDA. However, this approach risks running out of budget before requesting queries from the target domain, as it lacks control over the number of queries per domain. The proposed method can avoid the risk by distributing the budget to each domain.

4

Gradual domain adaptation with generative model

4.1 Contributions

In this chapter, we develop a method to deal with the problem of large discrepancies between adjacent domains while maintaining the framework of unsupervised domain adaptation. A simple approach to address the issue is to interpolate the large gaps between adjacent domains with pseudo-intermediate domains and then apply gradual self-training to the dense sequence of unlabeled datasets. The generation of pseudo-intermediate domains is realized by mixing the samples from the source and target domains (Zhang et al., 2021). Optimal transport is also a promising method for realizing interpolation between domains (He et al., 2023). However, as shown in Figure 4.1, the pseudo-intermediate domains generated by these methods might not be suitable for self-training.

We attempt to capture the continuous change between domains. We focus on the

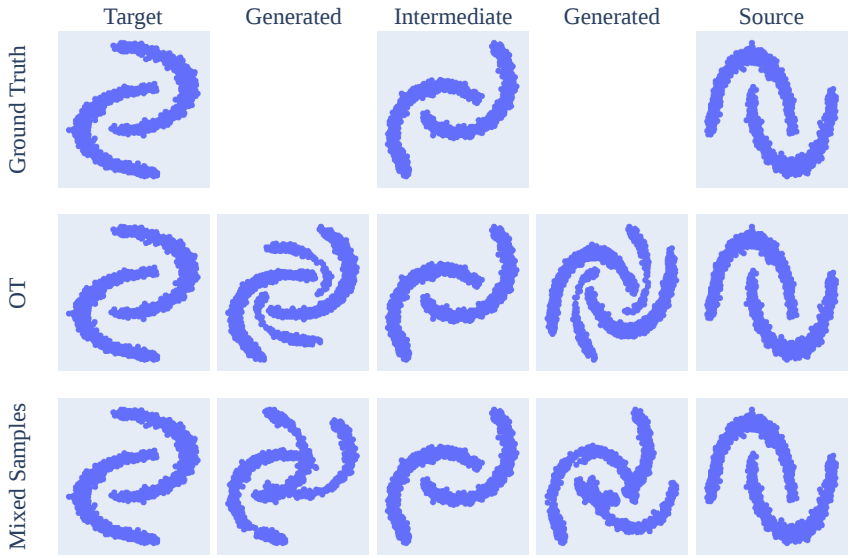


Figure 4.1: Examples of interpolations between domains utilizing pseudo-intermediate domains. Ineffective pseudo-intermediate domains deteriorate the performance of gradual self-training.

normalizing flow (NF), which learns the transformation between two distributions. Assuming that the transformation process of the distribution by trained NFs corresponds to the continuous change between the domains. Normalizing flows can achieve a more natural and direct transformation between domains compared to other generative models, such as generative adversarial networks (Pan et al., 2019).

In general, NFs learn the transformation between the distribution that the given data follows and a parametric distribution, such as the Gaussian distribution. Our flow-based model is designed to learn transformations between multiple distributions, and it can convert a non-parametric distribution in one domain into a non-parametric distribution in another domain. To the best of the authors' knowledge, there has been no study on NF designed for sequential domains and transformations between non-parametric distributions.

Since NF is a generative model, we can generate pseudo-intermediate domains using the trained NF and subsequently apply gradual self-training to the sequence of unlabeled datasets. However, this two-step method is complicated due to the need to adjust the hyperparameters of both NF and self-training. Moreover, it is not easy to determine the

appropriate number of pseudo-intermediate domains to be generated. We propose a method that can predict the label of a sample from the target domain without using either interpolation between domains or self-training. Izmailov et al. (2020) proposed to utilize an NF for semi-supervised learning. Inspired by this work, in order to predict the label of a sample with NF, our proposed method transforms the distribution of the source domain into a Gaussian mixture distribution instead of the Gaussian distribution. Figure 1.3 shows a schematic of the proposed method. Our trained NF predicts the class label of a sample from the target domain by transforming the sample to a sample from the Gaussian mixture distribution via the source domain. The transformation between the distribution of the source domain and the Gaussian mixture distribution is learned by leveraging labeled data from the source domain. The contributions of this chapter are as follows.

- We propose a GDA method, which does not use either interpolation between domains or gradual self-training, by capturing the continuous change between domains with NFs.

- While conventional normalizing flows are not designed to learn the transformation between non-parametric distributions, our flow-based model can do so by utilizing a non-parametric likelihood estimator.

- We demonstrate the effectiveness of the proposed method by experiments using both toy and real-world datasets.

4.2 Normalizing flow

Recently, there has been active research on the topic of normalizing flow. In this section, we discuss a few of these studies. A summary of the related work is shown in Table 4.1.

Normalizing flows are reversible generative models that use invertible neural networks to transform samples from a known distribution, such as the Gaussian distribution. Normalizing flows are trained by the maximum likelihood estimation, where the probability density of the transformed random variable is subject to the change of variable formula.

The architecture of the invertible neural networks is constrained (e.g., coupling-based architecture) so that its Jacobian matrix is efficiently computed. Normalizing flows with constrained architectures are called discrete NFs (DNFs), and examples include Real-valued Non-volume Preserving (RealNVP) (Dinh et al., 2017), Generative Flow (Glow) (Kingma and Dhariwal, 2018), and Flow++ (Ho et al., 2019). Several theoretical analyses of the expressive power of DNFs have also been reported. Kong and Chaudhuri (2020) studied basic flow models such as planar flows (Rezende and Mohamed, 2015) and proved the bounds of the expressive power of basic flow models. Teshima et al. (2020) conducted a more generalized theoretical analysis of coupling-based flow models.

Chen et al. (2018) proposed continuous normalizing flow (CNF), mitigating the constraints on the architecture of the invertible neural networks. Continuous normalizing flows describe the transformation between samples from the Gaussian to the observed samples from a complicated distribution using ordinary differential equations (ODEs). Grathwohl et al. (2019) proposed a variant of CNF called Free-form Jacobian of Reversible Dynamics (FFJORD), which exhibited improved performance over DNF. The FFJORD was followed by studies to improve the computational efficiency (Huang and Yeh, 2021; Onken et al., 2021) and on the representation on manifolds (Mathieu and Nickel, 2020; Rozen et al., 2021; Ben-Hamu et al., 2022). Several normalizing flow models have been proposed with the goal of learning a low-dimensional manifold on which data are distributed and estimating the density on that manifold (Brehmer and Cranmer, 2020; Caterini et al., 2021; Horvat and Pfister, 2021; Ross and Cresswell, 2021).

Normalizing flows have been applied to several specific tasks, for example, data generation [images (Lu and Huang, 2020), 3D point clouds (Pumarola et al., 2020), chemical graphs (Kuznetsov and Polykovskiy, 2021)], anomaly detection (Kirichenko et al., 2020), and semi-supervised learning (Izmailov et al., 2020). NFs are also used to compensate for other generative models (Mahajan et al., 2020; Yang et al., 2019; Abdal et al., 2021; Huang et al., 2021) such as generative adversarial networks and variational autoencoders (VAEs) (Zhai et al., 2018).

Normalizing flows have been utilized for a domain adaptation method to extract domain-invariant features. Grover et al. (2020) proposed a domain adaptation method that combines adversarial training and NFs. The method separately trains two NFs for the source and target domains and uses adversarial training to obtain domain-invariant

Table 4.1: Summary of related work of normalizing flow

Subject	Type	Reference
performance improvement	DNF	(Rezende and Mohamed, 2015; Dinh et al., 2017; Kingma and Dhariwal, 2018; Ho et al., 2019)
	CNF	(Chen et al., 2018; Grathwohl et al., 2019; Huang and Yeh, 2021; Onken et al., 2021)
theoretical analysis	DNF	(Kong and Chaudhuri, 2020; Teshima et al., 2020)
representation on manifolds	DNF	(Brehmer and Cranmer, 2020; Caterini et al., 2021; Horvat and Pfister, 2021; Ross and Cresswell, 2021)
	CNF	(Mathieu and Nickel, 2020; Rozen et al., 2021; Ben-Hamu et al., 2022)
data generation	DNF	(Lu and Huang, 2020; Pumarola et al., 2020; Kuznetsov and Polykovskiy, 2021; Huang et al., 2021)
	CNF	(Yang et al., 2019; Abdal et al., 2021)
domain adaptation	DNF	(Grover et al., 2020; Askari et al., 2023)
	CNF	Ours

features. These features, existing in a common latent variable space, make it difficult for the domain discriminator to distinguish between the source and target domains. Askari et al. (2023) proposed a domain adaptation method using NFs and VAEs. The encoder converts samples from the source and target domains into latent variables. The latent variables of the source domain are constrained to follow the Gaussian distribution. NFs transform latent variables of the target domain into latent variables of the source domain and predict the class label of the target data. While the above-explained methods only learn the transformation between the distribution that the given data follows and the Gaussian distribution, our proposed method is developed for sequential domains and is designed to transform between non-parametric distributions.

4.3 Proposed method

Our key idea is to utilize NFs to capture the continuous change between domains, as detailed in Section 4.3.1. Conventional NFs only learn the transformation between the distribution that the given data follows and a parametric distribution, such as the Gaussian distribution. To realize GDA without gradual self-training, we develop a non-parametric transformation method between distributions using NFs. Section 4.3.2

introduces a non-parametric likelihood estimator and shows how the likelihood of transformation between samples from adjacent domains is evaluated. We consider a multiclass classification problem and propose a method that can predict the labels of target data without the need for any post-processing after the training of NFs. Our proposed method learns the transformation between the distribution of the source domain and a Gaussian mixture distribution as detailed in Section 4.3.3. We discuss the theoretical aspects and the scalability of the proposed method in Sections 4.3.4 and 4.3.5, respectively.

4.3.1 Capturing continuous change with normalizing flows

To represent continuous changes between domains, we introduce a time index $t \in \mathbb{R}_{\geq 0}$. We regard the index of each domain j as a continuous variable and consider it as a particular time point. We set $t = j$ for the distribution of domain j, and $t = 0$ corresponds to the Gaussian mixture distribution, and we describe the details of the Gaussian mixture distribution in Section 4.3.3. The probability density function $p^{(j)}$ is a special case of $p^{(t)}$ when $t = j$. The DNF uses an invertible function $f : \mathbb{R}^d \to \mathbb{R}^d$ to transform a sample $\boldsymbol{x}^{(j)} \sim p^{(j)}$ to a sample $\boldsymbol{x}^{(j-1)} \sim p^{(j-1)}$. The log density of $\boldsymbol{x}^{(j)} = f(\boldsymbol{x}^{(j-1)})$ satisfies

$$\log p^{(j)}(\boldsymbol{x}^{(j)}) = \log p^{(j-1)}(f^{-1}(\boldsymbol{x}^{(j)})) + \log \left| \det \nabla f^{-1}(\boldsymbol{x}^{(j)}) \right|,$$

where $\nabla f^{-1}(\boldsymbol{x}^{(j)})$ is the Jacobian of f^{-1}. Our aim is to capture the continuous change by using NFs. We can consider a continuous transformation by using multiple DNFs. However, our preliminary experiments indicate that CNFs are better suited for capturing continuous transitions between domains than DNFs. Further details of these experiments are elaborated upon in Section 4.4.3.

The CNF $g : \mathbb{R}^d \times \mathbb{R}_{\geq 0} \to \mathbb{R}^d$ outputs a transformed variable dependent on time t, and we denote the output corresponding to t as $g(\cdot, t)$, following the standard notation of CNF. When the time assigned to the variable and the time input into g are the same, g does not perform any conversion, i.e., $\boldsymbol{x}^{(j)} = g(\boldsymbol{x}^{(j)}, j)$. Let $v : \mathbb{R}^d \times \mathbb{R}_{\geq 0} \ni (\boldsymbol{x}, t) \mapsto v(\boldsymbol{x}, t; \omega) \in \mathbb{R}^d$ be a neural network parameterized by ω that represents the change in g along t. Since the input and output of the functions g and v are both $\mathbb{R}^d$, we denote the output of each function as $\boldsymbol{g}$ and $\boldsymbol{v}$. Following Chen et al. (2018) and Grathwohl et al.

(2019), we express the ODE with respect to g using the neural network v as

$$\frac{\partial \boldsymbol{g}}{\partial t} = v(g(\cdot, t), t; \omega).$$

The parameter ω of the neural network v implicitly defines the CNF g. When we specifically refer to the parameter of g, we denote it as g_ω.

A DNF requires an explicit computation of the Jacobian, and in a CNF, it is calculated by integrating the time derivative of the log-likelihood. The time derivative of the log-likelihood is expressed as $\partial \log p(g_\omega(\boldsymbol{x}, t))/\partial t = -\mathrm{Tr}(\partial \boldsymbol{v}/\partial \boldsymbol{g})$ (Chen et al., 2018, Theorem 1), where $\partial \boldsymbol{v}/\partial \boldsymbol{g} \in \mathbb{R}^{d \times d}$. We assign $t_0 = j - 1$ and $t_1 = j$, and the outputs of the CNF for the input $\boldsymbol{x}^{(j)}$ are obtained by solving the following initial value problem:

$$\begin{aligned} \begin{bmatrix} \boldsymbol{x}^{(t_0)} - \boldsymbol{x}^{(t_1)} \\ \Delta_1 \end{bmatrix} &= \int_{t_1}^{t_0} \begin{bmatrix} v(g_\omega(\boldsymbol{x}^{(j)}, t), t; \omega) \\ \mathrm{Tr}(\partial \boldsymbol{v}/\partial \boldsymbol{g}) \end{bmatrix} dt, \\ \begin{bmatrix} g_\omega(\boldsymbol{x}^{(j)}, t_1) \\ \Delta_0 \end{bmatrix} &= \begin{bmatrix} \boldsymbol{x}^{(j)} \\ 0 \end{bmatrix}, \end{aligned} \tag{4.1}$$

where $\Delta_1 = \log p^{(t_1)}(\boldsymbol{x}^{(t_1)}) - \log p^{(t_0)}(\boldsymbol{x}^{(t_0)})$ and $\Delta_0 = \log p^{(t_1)}(\boldsymbol{x}^{(t_1)}) - \log p^{(t_1)}(g_\omega(\boldsymbol{x}^{(j)}, t_1))$. Note that the range of integration is from time t_1 to t_0 since the CNF transforms time t_1 to t_0. The initial value problem can be addressed using numerical ODE solvers. The CNF transforms a sample $\boldsymbol{x}^{(j)} \sim p^{(j)}$ to a sample $\boldsymbol{x}^{(j-1)} \sim p^{(j-1)}$, and the likelihood of the model satisfies

$$\log p^{(j)}(g_\omega(\boldsymbol{x}^{(j)}, j)) = \log p^{(j-1)}(g_\omega(\boldsymbol{x}^{(j)}, j-1)) - \int_{j-1}^{j} \mathrm{Tr}\left(\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{g}}\right) dt.$$

A CNF is formulated as a problem of minimizing the expectation of the minus of the log-likelihood function

$$\mathbb{E}_{\boldsymbol{x}^{(j)} \sim p^{(j)}}[-\log p^{(j)}(g_\omega(\boldsymbol{x}^{(j)}, j))] \tag{4.2}$$

with respect to the parameter ω.

In order to represent continuous changes between domains, the initial value problem given by Eq. (4.1) is sequentially solved. When $j > 1$, the initial value problem is solved sequentially with decreasing values of t_1 and t_0 until $t_1 = 1$ and $t_0 = 0$. For instance,

when $j=2$, we solve the initial value problem given by Eq. (4.1) and retain the solutions. In the next iteration, we decrease the values of t_1 and t_0 and utilize the retained values as initial values. We define our flow-based model as a problem of maximizing the following log-likelihood with respect to the parameter ω:

$$\log p^{(j)}(g_\omega(\boldsymbol{x}^{(j)}, j)) = \sum_{t=1}^{j} \log p^{(t-1)}(g_\omega(\boldsymbol{x}^{(j)}, t-1)) - \int_0^j \mathrm{Tr}\left(\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{g}}\right) dt. \tag{4.3}$$

4.3.2 Non-parametric estimation of log-likelihood

We introduce k nearest neighbor (kNN) estimators for the log-likelihood (Kozachenko and Leonenko, 1987; M. N. Goria and Inverardi, 2005), following the derivation in Hino et al. (2015). Our aim is to estimate $p(\boldsymbol{a})$ from the observed dataset $\{\boldsymbol{x}_i\}_{i=1}^n$, where $\boldsymbol{a} \in \mathbb{R}^d$ is called an inspection point. Let $b(\boldsymbol{a}, \delta) = \{\boldsymbol{x} \in \mathbb{R}^d \mid \|\boldsymbol{x} - \boldsymbol{a}\| < \delta\}$ be a d-dimensional ball of radius δ centered at $\boldsymbol{a}$. The volume of the ball is $|b(\boldsymbol{a}, \delta)| = \delta^d \pi^{d/2}/\Gamma(1 + d/2)$, where $\Gamma(\cdot)$ is the gamma function. We denote the probability mass of the ball centered at $\boldsymbol{a}$ as $V(\delta) = \int_{b(\boldsymbol{a},\delta)} p(\boldsymbol{x}) d\boldsymbol{x}$. Assuming that δ is sufficiently small, we obtain the following approximation formula by Taylor's expansion:

$$\begin{aligned} V(\delta) &= \int_{b(\boldsymbol{a},\delta)} \{p(\boldsymbol{a}) + (\boldsymbol{x} - \boldsymbol{a})^\top \nabla p(\boldsymbol{a}) + O(\delta^2)\} d\boldsymbol{x} \\ &= |b(\boldsymbol{a}, \delta)|(p(\boldsymbol{a}) + O(\delta^2)) \simeq p(\boldsymbol{a}) \delta^d \pi^{d/2}/\Gamma(1 + d/2), \end{aligned}$$

where the term with first derivative of the density function vanishes due to symmetry. We consider $\delta = \delta_k$, the Euclidean distance between $\boldsymbol{a}$ and its k-th nearest neighbor in $\{\boldsymbol{x}_i\}_{i=1}^n$. When the radius of the ball is δ_k, k out of n observations are included in $b(\boldsymbol{a}, \delta_k)$, and the probability mass can be approximated as $V(\delta) \simeq k/n$. Therefore, $k/n \simeq p(\boldsymbol{a}) \delta^d \pi^{d/2}/\Gamma(1 + d/2)$ and we obtain a kNN estimator for the log-likelihood as follows:

$$\log \hat{p}(\boldsymbol{a}) = \log \frac{k}{n} + \log \Gamma\left(1 + \frac{d}{2}\right) - \frac{d}{2} \log \pi - d \log \delta_k.$$

The transformation of a sample from the distribution of j-th domain to a sample from the adjacent domain using a CNF requires the computation of the log-likelihood $\log p_{j-1}(g_\omega(\boldsymbol{x}^{(j)}, j-1))$. We use the kNN estimators for the log-likelihood. We compute the Euclidean distances between among all samples in $\{\boldsymbol{x}_i^{(j-1)}\}_{i=1}^{n_{j-1}}$ and $g_\omega(\boldsymbol{x}^{(j)}, j-1)$,

with $g_\omega(\boldsymbol{x}^{(j)}, j-1)$ kept fixed. Let $\delta_{k,j-1}(g_\omega(\boldsymbol{x}^{(j)}, j-1))$ be the Euclidean distance between $g_\omega(\boldsymbol{x}^{(j)}, j-1)$ and its k-th nearest neighbor in $\{\boldsymbol{x}_i^{(j-1)}\}_{i=1}^{n_{j-1}}$. We obtain the kNN estimator of the log-likelihood of the sample $g_\omega(\boldsymbol{x}^{(j)}, j-1)$ as follows:

$$\log \hat{p}^{(j-1)}(g_\omega(\boldsymbol{x}^{(j)}, j\text{–}1)) = \log \frac{k}{n_{j-1}} + \log \Gamma\left(1 + \frac{d}{2}\right) - \frac{d}{2} \log \pi - d \log \delta_{k,j-1}(g_\omega(\boldsymbol{x}^{(j)}, j\text{–}1)).$$

We estimate the log-likelihood for all samples in the j-th domain $\{\boldsymbol{x}_i^{(j)}\}_{i=1}^{n_j}$ and maximize its sample average

$$-\frac{d}{n_j} \sum_{i=1}^{n_j} \log \delta_{k,j-1}(g_\omega(\boldsymbol{x}_i^{(j)}, j-1)))$$

with respect to ω.

For simplicity, we consider the case $n_j = n_{j-1} = n$. The cost of computing the log-likelihood by the kNN estimators is $O(n^2)$. During the training of g_ω, the computation of log-likelihood by the kNN estimators is required each time g_ω is updated. We use the nearest neighbor descent algorithm (Dong et al., 2011) to reduce the computational cost of the kNN estimators. The algorithm can be used to construct kNN graphs efficiently, and the computational cost is empirically evaluated to be $O(n^{1.14})$.

Another way to estimate the log-likelihood $\log p^{(j-1)}(g_\omega(\boldsymbol{x}^{(j)}, j\text{–}1))$ is to approximate $p^{(j-1)}$ using a surrogate function. Our preliminary experiments show that the kNN estimators are suitable for capturing continuous changes between domains, and the details of the experiments are described in Section 4.4.4.

4.3.3 Gaussian mixture model

In Sections 4.3.1 and 4.3.2, we have discussed a method that captures continuous changes between domains using NFs. Normalizing flows trained with log-likelihood based on kNN can transform samples from the target domain to samples from the source domain via the intermediate domain. While we can generate pseudo-intermediate domains using trained NFs and subsequently apply gradual self-training to the sequence of unlabeled datasets, this two-step approach is complicated. We develop a method that predicts the labels of target data without any post-processing after the training of NFs. Since we consider a multiclass classification problem, we assign one Gaussian distribution for each class label. Therefore, the proposed method learns the transformation between the

distribution of the source domain and a Gaussian mixture distribution. It is inspired by the previous work (Izmailov et al., 2020), which proposed a semi-supervised learning method using DNFs and a Gaussian mixture distribution. In this subsection, we explain the Gaussian mixture model (GMM) suitable for our proposed method and the log-likelihood of the flow-based model with respect to the GMM.

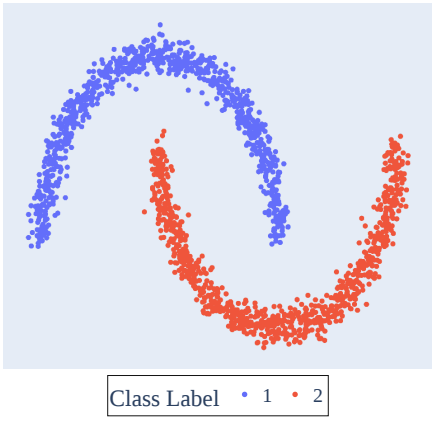
Following Izmailov et al. (2020), we make two assumptions.

- The distribution $p^{(0)}$, conditioned on the label s, is modeled by a Gaussian with the mean $\boldsymbol{\mu}_s$ and the covariance matrix Σ_s, $p^{(0)}(\boldsymbol{x}|y=s)=\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_s,\Sigma_s)$.
- The classes $\{1,2,\ldots,C\}$ are balanced, i.e., $\forall s\in\{1,2,\ldots,C\},\ p(y=s)=1/C$, and the Gaussian mixture distribution is $p^{(0)}(\boldsymbol{x})=\frac{1}{C}\sum_{s=1}^{C}\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_s,\Sigma_s)$.

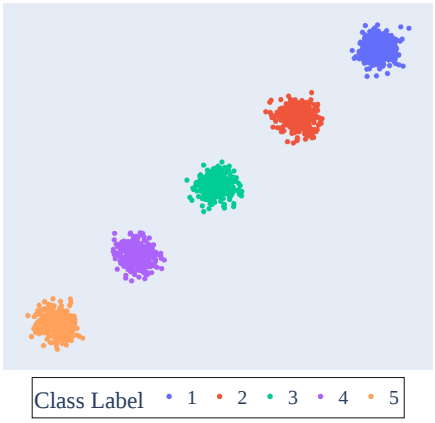
Empirically, it is known that the second assumption is effective in learning the transformation between the distribution that data follows and the Gaussian mixture distribution by NFs (Izmailov et al., 2020). We set an identity matrix as the covariance matrix for all classes, $\Sigma_s=I$. The Gaussian distributions for different labels should be distinguishable from each other, and it is desirable that the appropriate mean vector $\boldsymbol{\mu}_s$ is assigned to each Gaussian distribution. We propose to assign the mean vector $\boldsymbol{\mu}_s=(\mu_{s_1},\ldots,\mu_{s_d})^\top$ using the polar coordinates system. Each component of the mean vector is given by

$$\mu_{s_i}=\begin{cases} r\cos\theta_s(\sin\theta_s)^{i-1}, & (i=1,\ldots,d-1),\\ r(\sin\theta_s)^{d-1}, & (i=d),\end{cases} \tag{4.4}$$

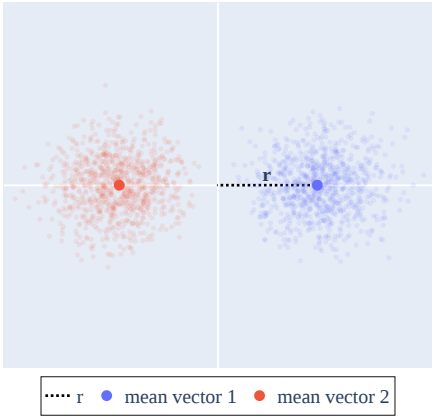
where r is the distance from the origin in the polar coordinate system and the angle $\theta_s=2\pi(s-1)/C, \forall s\in\{1,2,\ldots,C\}$. Note that r is a hyperparameter. Figure 4.2 shows an intuitive description of r. If r is too small, as shown in Figure 4.2 (e) and 4.2 (f), it is difficult to identify the class labels of samples. As shown in Eq. 4.4 and Figure 4.2 (d), the mean vectors are set at equal intervals.



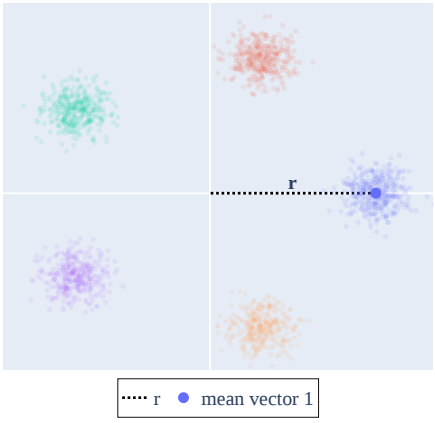
(a) `Two Moon`: source domain



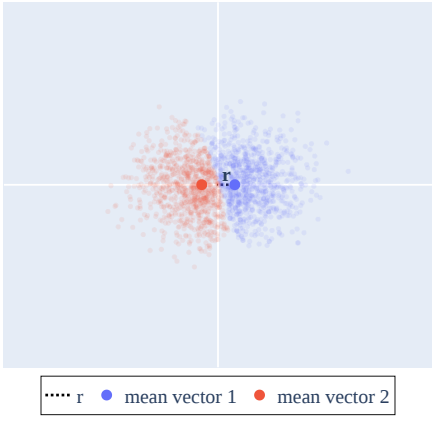
(b) `Block`: source domain



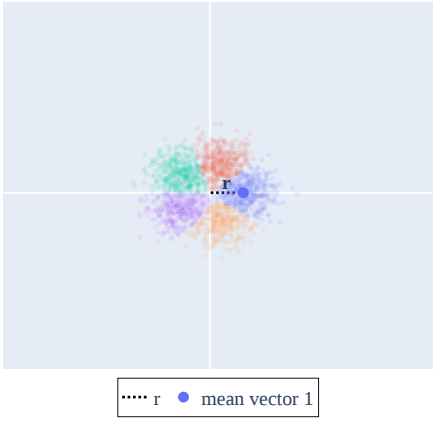
(c) Gaussian mixture ($r = 3.0$).



(d) Gaussian mixture ($r = 10.0$).



(e) Gaussian mixture ($r = 0.5$).



(f) Gaussian mixture ($r = 2.0$).

Figure 4.2: Schematic of hyperparameter r, (left): `Two Moon` and (right): `Block`.

Since the source domain has labeled data, by using Eq. (4.3), we can obtain the class conditional log-likelihood of a labeled sample as

$$\log p^{(1)}(g_\omega(\boldsymbol{x}^{(1)}, 1)|y = s) = \log \mathcal{N}(g_\omega(\boldsymbol{x}^{(1)}, 0)|\boldsymbol{\mu}_s, \Sigma_s) - \int_0^1 \mathrm{Tr}\left(\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{g}}\right) dt. \quad (4.5)$$

The intermediate domains and the target domain have no labeled data. Thus, $\log p^{(0)}(g_\omega(\boldsymbol{x}^{(j)}, 0))$ is estimated as $\log\left\{\frac{1}{C}\sum_{s=1}^{C} \mathcal{N}(g_\omega(\boldsymbol{x}^{(j)}, 0)|\boldsymbol{\mu}_s, \Sigma_s)\right\}$. As described in Section 4.3.2, we compute $\log p^{(j-1)}(g_\omega(\boldsymbol{x}^{(j)}, j-1))$ as $-d \log \delta_{k,j-1}(g_\omega(\boldsymbol{x}^{(j)}, j-1))$. Using Eq. (4.3), the log-likelihood of an unlabeled sample is given by

$$\begin{aligned}
\log p^{(j)}(g_\omega(\boldsymbol{x}^{(j)}, j)) &= \sum_{t=1}^{j} \log p^{(t-1)}(g_\omega(\boldsymbol{x}^{(j)}, t-1)) - \int_0^j \mathrm{Tr}\left(\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{g}}\right) dt \\
&= \log p^{(0)}(g_\omega(\boldsymbol{x}^{(j)}, 0)) + \sum_{t=2}^{j} \log p^{(t-1)}(g_\omega(\boldsymbol{x}^{(j)}, t-1)) - \int_0^j \mathrm{Tr}\left(\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{g}}\right) dt \\
&= \log\left\{\frac{1}{C}\sum_{s=1}^{C} \mathcal{N}(g_\omega(\boldsymbol{x}^{(j)}, 0)|\boldsymbol{\mu}_s, \Sigma_s)\right\} - \sum_{t=2}^{j} d \log \delta_{k,t-1}(g_\omega(\boldsymbol{x}^{(j)}, t-1)) \\
&\quad - \int_0^j \mathrm{Tr}\left(\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{g}}\right) dt. \quad (4.6)
\end{aligned}$$

Namely, to capture the continuous change between domains, we maximize the log-likelihood of g_ω on all the data from the given domains. The algorithm minimizes the following objective function with respect to g_ω:

$$\mathrm{L}(\omega; D^{(1)}, \{U^{(j)}\}_{j=2}^{K}) = -\frac{1}{n_1}\sum_{i=1}^{n_1} \log p^{(1)}(g_\omega(\boldsymbol{x}_i^{(1)}, 1)|y_i) - \sum_{j=2}^{K} \frac{1}{n_j}\sum_{i=1}^{n_j} \log p^{(j)}(g_\omega(\boldsymbol{x}_i^{(j)}, j)).$$

We show a pseudocode for our proposed method in Algorithm 4.1.

Our method has two hyperparameters, k and r: k affects the result of the computation of log-likelihood by the kNN estimators, whereas r controls the distance between the Gaussian corresponding to each class. We discuss how to tune these hyperparameters in Section 4.4.5.

We consider making predictions for a new sample by using our flow-based model.

The predictive probability that the class of the given test sample $\boldsymbol{x}$ being s is

$$p(y=s|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|y=s)p(y=s)}{\sum_{s'=1}^{C} p(\boldsymbol{x}|y=s')p(y=s')} = \frac{\mathcal{N}(g_\omega(\boldsymbol{x},0)|\mu_s,\Sigma_s)}{\sum_{s'=1}^{C} \mathcal{N}(g_\omega(\boldsymbol{x},0)|\mu_{s'},\Sigma_{s'})}. \tag{4.7}$$

Therefore, the class label of a new sample $\boldsymbol{x}$ is predicted by

$$\hat{y} = \arg\max_{s\in\{1,\ldots,C\}} p(y=s|\boldsymbol{x}).$$

Algorithm 4.1 Gradual domain adaptation with CNF

Input: labeled dataset $D^{(1)}$ and unlabeled datasets $U^{(2)},\ldots,U^{(K)}$
Output: trained CNF g_ω

$j \leftarrow K$ ▷ start training from the target domain
while $j > 0$ **do**
  $t_0 \leftarrow j-1,\quad t_1 \leftarrow j.$
  initial values are set to $\boldsymbol{x}^{(j)}$ and 0.
  while $t_0 > 0$ **do** ▷ the initial value problem is solved sequentially
    solve the initial value problem Eq. (4.1).
    retain the solutions. ▷ these values will be used as initial values in the next iteration
    $t_0 \leftarrow t_0 - 1,\quad t_1 \leftarrow t_1 - 1.$
  end while
  if $j = 1$ **then** ▷ update CNF g_ω
    maximize the log-likelihood of labeled data as Eq. (4.5) with respect to ω.
  else
    maximize the log-likelihood of unlabeled data as Eq. (4.6) with respect to ω.
  end if
  $j \leftarrow j-1$ ▷ training on the adjacent domain
end while

4.3.4 Theoretical aspects of flow-based model

Our proposed method maximizes the log-likelihood of labeled data with respect to ω given by Eq. (4.5). We utilize NFs to model the distribution of inputs $p(\boldsymbol{x}|y)$, and via Eq. (4.7), our proposed method also implicitly models the distribution of outputs $p(y|\boldsymbol{x})$.

We denote the expected loss on the source domain as follows:

$$\begin{aligned}\mathrm{L}_1(g_\omega) &= \mathbb{E}_{\boldsymbol{x},y\sim p^{(1)}_{\boldsymbol{x},y}}[-\log p(y|\boldsymbol{x})] \\ &= \mathbb{E}_{\boldsymbol{x},y\sim p^{(1)}_{\boldsymbol{x},y}}\left[-\log \frac{\mathcal{N}(g_\omega(\boldsymbol{x},0)|\mu_y,\Sigma_y)}{\sum_{s=1}^{C}\mathcal{N}(g_\omega(\boldsymbol{x},0)|\mu_s,\Sigma_s)}\right].\end{aligned}$$

Similarly, the expected loss on the j-th domain is denoted as $\mathrm{L_j}(g_\omega) = \mathbb{E}_{\boldsymbol{x},y\sim p^{(j)}_{\boldsymbol{x},y}}[-\log p(y|\boldsymbol{x})]$. For notational simplicity, we omit the argument of the expected loss and denote it as L_1 and $\mathrm{L_j}$. Note that $p(y|\boldsymbol{x})$ is a probability mass function, and we consider the following natural assumption.

Assumption 4.1 (Nguyen et al. (2022))**.** *For some $M \in \mathbb{R}_{\geq 0}$, the loss satisfies $0 \leq -\log p(y|\boldsymbol{x}) \leq M$, where $\forall \boldsymbol{x} \in \mathcal{X}, \forall y \in \mathcal{Y}$.*

Nguyen et al. (2022) derived an upper bound for the loss L_2 on the basis of the source loss L_1 and the Kullback-Leibler (KL) divergence between $p^{(2)}$ and $p^{(1)}$. Note that in the standard domain adaptation, there is no intermediate domain and $K = 2$.

Proposition 4.1 (Nguyen et al. (2022))**.** *If Assumption 4.1 holds, we have*

$$\begin{aligned}\mathrm{L}_2 &\leq \mathrm{L}_1 + \frac{M}{\sqrt{2}}\sqrt{\mathrm{KL}[p^{(2)}_{\boldsymbol{x},y}|p^{(1)}_{\boldsymbol{x},y}]} \\ &= \mathrm{L}_1 + \frac{M}{\sqrt{2}}\sqrt{\mathrm{KL}[p^{(2)}_{\boldsymbol{x}}|p^{(1)}_{\boldsymbol{x}}] + \mathbb{E}_{p^{(2)}_{\boldsymbol{x}}}[\mathrm{KL}[p^{(2)}_{y|\boldsymbol{x}}|p^{(1)}_{y|\boldsymbol{x}}]]},\end{aligned} \tag{4.8}$$

where $\mathrm{KL}[\cdot|\cdot]$ denotes the KL divergence.

Proof. See Appendices. □

In the theoretical analysis of conventional domain adaptation, an upper bound for the target loss is derived based on the source loss and the divergence. From a practical point of view, a divergence that is easy to compute would be preferred. Although the bound using the total variation (Ben-David et al., 2010) is well-known, evaluating the total variation might be challenging in practice since it requires the computation of a supremum. In contrast, Proposition 4.1 depends on the KL divergence, which is easier to compute than the total variation.

In Eq. (4.8), we note that it is impossible to calculate the conditional misalignment term $\mathbb{E}_{p_{\boldsymbol{x}}^{(2)}}[\mathrm{KL}[p_{y|\boldsymbol{x}}^{(2)}|p_{y|\boldsymbol{x}}^{(1)}]]$ since the labels from the second domain are not given. We make the following covariate shift assumption: [1]

Assumption 4.2 (Shimodaira (2000))**.** *For any* $t \in \{2,\ldots,K\}$, $p_{y|\boldsymbol{x}}^{(t)} = p_{y|\boldsymbol{x}}^{(t-1)}$, $\forall \boldsymbol{x} \in \mathcal{X}, \forall y \in \mathcal{Y}$.

Using Proposition 4.1, we introduce the following corollary that gives an upper bound of the target loss $\mathrm{L_K}$.

Corollary 4.1. *If Assumptions 4.1 and 4.2 hold, we have*

$$\begin{aligned}
\mathrm{L_K} &\leq \mathrm{L_1} + \frac{M}{\sqrt{2}} \sum_{j=2}^{K} \sqrt{\mathrm{KL}[p_{\boldsymbol{x},y}^{(j)}|p_{\boldsymbol{x},y}^{(j-1)}]} \\
&= \mathrm{L_1} + \frac{M}{\sqrt{2}} \sum_{j=2}^{K} \sqrt{\mathrm{KL}[p_{\boldsymbol{x}}^{(j)}|p_{\boldsymbol{x}}^{(j-1)}] + \mathbb{E}_{p_{\boldsymbol{x}}^{(j)}}[\mathrm{KL}[p_{y|\boldsymbol{x}}^{(j)}|p_{y|\boldsymbol{x}}^{(j-1)}]]} \\
&= \mathrm{L_1} + \frac{M}{\sqrt{2}} \sum_{j=2}^{K} \sqrt{\mathrm{KL}[p_{\boldsymbol{x}}^{(j)}|p_{\boldsymbol{x}}^{(j-1)}]}. \qquad (4.9)
\end{aligned}$$

Proof. Since the ordered sequence of the intermediate domains is given, we extend Proposition 4.1 until the target loss $\mathrm{L_K}$ as follows:

$$\begin{aligned}
\mathrm{L_2} &\leq \mathrm{L_1} + \frac{M}{\sqrt{2}} \sqrt{\mathrm{KL}[p_{\boldsymbol{x},y}^{(2)}|p_{\boldsymbol{x},y}^{(1)}]} \\
\mathrm{L_3} &\leq \mathrm{L_2} + \frac{M}{\sqrt{2}} \sqrt{\mathrm{KL}[p_{\boldsymbol{x},y}^{(3)}|p_{\boldsymbol{x},y}^{(2)}]} \\
&\vdots \\
\mathrm{L_K} &\leq \mathrm{L_{K-1}} + \frac{M}{\sqrt{2}} \sqrt{\mathrm{KL}[p_{\boldsymbol{x},y}^{(K)}|p_{\boldsymbol{x},y}^{(K-1)}]}.
\end{aligned}$$

[1] This assumption can be slightly relaxed by setting $\mathrm{KL}[p_{y|\boldsymbol{x}}^{(t)}|p_{y|\boldsymbol{x}}^{(t-1)}] \leq \varepsilon_t$ with small constants $\varepsilon_t \geq 0, \forall t \in \{2,\ldots,K\}$.

Summing up both sides of the above inequalities, we have

$$\begin{aligned}\mathrm{L_K} &\leq \mathrm{L}_1 + \frac{M}{\sqrt{2}} \sum_{j=2}^{K} \sqrt{\mathrm{KL}[p_{\boldsymbol{x},y}^{(j)} | p_{\boldsymbol{x},y}^{(j-1)}]} \\ &= \mathrm{L}_1 + \frac{M}{\sqrt{2}} \sum_{j=2}^{K} \sqrt{\mathrm{KL}[p_{\boldsymbol{x}}^{(j)} | p_{\boldsymbol{x}}^{(j-1)}] + \mathbb{E}_{p_{\boldsymbol{x}}^{(j)}} [\mathrm{KL}[p_{y|\boldsymbol{x}}^{(j)} | p_{y|\boldsymbol{x}}^{(j-1)}]]} \ (\because \ \textit{Eq.} \ (4.8)).\end{aligned}$$

If Assumption 4.2 holds, we have

$$\mathrm{L_K} \leq \mathrm{L}_1 + \frac{M}{\sqrt{2}} \sum_{j=2}^{K} \sqrt{\mathrm{KL}[p_{\boldsymbol{x}}^{(j)} | p_{\boldsymbol{x}}^{(j-1)}]}.$$

□

We consider reducing the marginal misalignment term $\mathrm{KL}[p_{\boldsymbol{x}}^{(j)} | p_{\boldsymbol{x}}^{(j-1)}]$ in Eq. (4.9), and introduce the following proposition derived by Onken et al. (2021).

Proposition 4.2 (Onken et al. (2021))**.** *The minimization of Eq.* (4.2) *is equivalent to the minimization of the KL divergence between* $p_{\boldsymbol{x}}^{(j-1)}$ *and* $p_{\boldsymbol{x}}^{(j)}$ *transformed by* g_ω*.*

Proof. See Appendices. □

By Proposition 4.2, we can reduce the marginal misalignment term $\mathrm{KL}[p_{\boldsymbol{x}}^{(j)} | p_{\boldsymbol{x}}^{(j-1)}]$ in Eq. (4.9) by transforming $p_{\boldsymbol{x}}^{(j)}$ to $p_{\boldsymbol{x}}^{(j-1)}$ with NFs.

Compared to the upper bound for the self-training-based GDA Eq. (2.3), which depends only on the loss in the source domain and the number of the intermediate domains, our bound is adaptive since the KL-divergence term can be reduced by the training of CNF. The self-training procedure contains hyperparameters such as the number of epochs and learning rates, and how to determine those appropriate hyperparameters based on Eq. (2.3) is non-trivial. In contrast, Corollary 4.1 is free from self-training.

4.3.5 Scalability

Grathwohl et al. (2019) discussed the details of the scalability of a CNF. Evaluating a CNF costs $O(dH)$, where d is the dimension of the input and H is the size of the

largest hidden unit in v (Grathwohl et al., 2019). They derived the cost of computing the likelihood as $O(dHN)$, where N is the number of evaluations of v in the ODE solver. In general, the training cost of CNF is high because the number of evaluations N of v in the ODE solver is large. When K domains are given, we have to solve the initial value problem $\frac{1}{2}K(K+1)$ times in our proposed method, resulting the computational cost $O(dHNK^2)$, which is computationally expensive when the number of intermediate domains is large. When we can access many intermediate domains and the distances between the intermediate domains are small, the conventional self-training-based GDA algorithm (Kumar et al., 2020) will be suitable.

The solution obtained from the ODE solver contains numerical errors. In order to reduce these errors, it is necessary to increase the number of evaluations for the function v. Chen et al. (2018) experimentally demonstrated the trade-off between error and the number of evaluations of the function v, and they also provided heuristic settings for ODE solvers. In Section 4.4, we use the settings of ODE solvers provided by Chen et al. (2018).

4.4 Experiments

We use the implementation of the CNF provided by Huang and Yeh (2021). [2] PyNNDescent provides a python implementation of the nearest neighbor descent (Dong et al., 2011). [3] PyTorch (Paszke et al., 2019) is used to implement all procedures in our proposed method except for the procedure of CNF and nearest neighbor descent. Our flow-based model g_ω consists of one CNF block. One CNF block consists of two fully connected layers with 64 nodes in each layer. We use WILDS (Koh et al., 2021) and MoleculeNet (Wu et al., 2018) to load pre-processed datasets. All experiments are conducted on our server with Intel Xeon Gold 6354 processors and NVIDIA A100 GPU.

Our aim is to capture the continuous change between domains. In our experiments using toy datasets, we generate pseudo-intermediate domains from trained NFs to verify whether the trained NFs accurately capture the continuous change between domains. Generating pseudo-intermediate domains is just a sanity check. Recall that the proposed method does not generate pseudo-intermediate domains when predicting the label of a

[2] https://github.com/hanhsienhuang/CNF-TPR
[3] https://github.com/lmcinnes/pynndescent/tree/master

sample.

4.4.1 Datasets

We use benchmark datasets with modifications for GDA. Since we are considering the situation that the discrepancies between the adjacent domains are large, we prepare only one or two intermediate domains. We describe the details of the datasets.

`Two Moon`**:** A toy dataset. We use the `two-moon` dataset as the source domain. The intermediate and target domains are prepared by rotating the source dataset by $\pi/4$ and $\pi/2$, respectively.

`Block`**:** A toy dataset. The number of dimensions of the data is two, and the number of classes is five. We prepare the intermediate and target domains by adding horizontal movement to each class. Note that only the `Block` dataset has two intermediate domains.

`Rotating MNIST` **(Kumar et al., 2020):** We add rotations to the `MNIST` dataset. The rotation angle is 0 for the source domain, $\pi/6$ for the intermediate domain, and $\pi/3$ for the target domain. We normalize the image intensity to the range between 0 and 1 by dividing by 255.

`Portraits` **(Ginosar et al., 2015):** The `Portraits` dataset includes photographs of U.S. high school students from 1905 to 2013, and the task is gender classification. We sort the dataset in ascending order by year and split the dataset. The source dataset includes data from the 1900s to the 1930s, and the intermediate and target datasets contain data from the 1940s to 1950s and from the 1960s, respectively. We resize the original image to 32×32 and normalize the image intensity to the range between 0 and 1 by dividing by 255.

`Tox21` **(Thomas et al., 2018):** The `Tox21` dataset contains the results of measuring the toxicity of compounds. The dataset contains twelve types of toxicity evaluation with a number of missing values. We merge these evaluations into a single evaluation

and consider a compound as toxic when it is determined to be harmful in any of the twelve evaluations. Because the Tox21 dataset does not have a domain indicator such as year, we introduce an indicator for splitting the entire dataset into domains. It is a reasonable method from the chemical view point to divide the entire dataset into domains by the number of arbitrary substituents in the compound. We select the following three chemically representative substituents and use the number of substituents as a domain indicator.

- NHOHCount: Number of NHOH groups in the compound.
- RingCount: Number of ring structures in the compound.
- NumHDonors: Number of positively polarized hydrogen bonds in the compound.

The zeroth, first, and second substituents are assigned to the source domain, the intermediate domain, and the target domain, respectively. We use 108-dimensional molecular descriptors as features used in the previous work (Drwal et al., 2015). Each descriptor is normalized by subtracting the mean of the samples and then dividing the result by the standard deviation of the samples.

SHIFT15M (Kimura et al., 2021): The SHIFT15M dataset consists of 15 million fashion images collected from real fashion e-commerce sites. We estimate seven categories of clothes from image features. The SHIFT15M dataset does not provide images but provides VGG16 (Simonyan and Zisserman, 2015) features consisting of 4,096 dimensions. The dataset contains fashion images from 2010 to 2020, and the passage of years causes changes in distributions. The number of samples from 2010 is significantly smaller than that from other years, and we merge samples from 2010 with those from 2011. We consider the datasets from 2011, 2015, and 2020 as the source domain, the intermediate domain, and the target domain, respectively. Owing to the significant number of samples, we randomly select 5,000 samples from each domain.

RxRx1 (Taylor et al., 2019): The RxRx1 dataset consists of three channels of cell images obtained by a fluorescence microscope. We resize the original image to 32×32 and normalize the image intensity to the range between 0 and 1 by dividing by 255. We estimate the cell type used in the experiment from image features. The change in

distributions of each batch occurs due to slight changes in temperature, humidity, and reagent concentration during experiments. We consider batch numbers one, two, and three as the source domain, the intermediate domain, and the target domain, respectively.

4.4.2 Experimental settings

The manifold hypothesis (Fefferman et al., 2016) posits that real high-dimensional data lies on a low-dimensional manifold embedded in a high-dimensional space. Brehmer and Cranmer (2020) mentioned that NFs are unsuitable for data that agree with the manifold hypothesis. UMAP (McInnes et al., 2018) is a dimensionality reduction method based on the manifold hypothesis, which embeds high-dimensional data into a low-dimensional space. Empirically, it is demonstrated that UMAP provides a suitable low-dimensional representation for each task (Ali et al., 2019; Becht et al., 2019). We apply UMAP to each dataset as preprocessing. As in semi-supervised learning, UMAP can use labeled and unlabeled data simultaneously. To determine the appropriate embedding dimension, we train a CNF g_ω on the dimension-reduced source dataset by maximizing Eq. (4.5) with respect to ω. After the training, we evaluate the accuracy on the dimension-reduced source dataset. We select the embedding dimension with which the result of mean accuracy of three-fold cross-validation in the dimension-reduced source dataset is the best. All parameters in UMAP are set to their default values.

Our proposed method is designed for datasets where the intermediate domains are arranged to connect the source domain to the target domain. To verify the configuration of the intermediate domains, we introduce an evaluation metric for each dataset. Our evaluation metric based on the KL divergence since Corollary 4.1 shows that the target loss $\mathrm{L_K}$ is bounded by $\sum_{j=2}^{K} \sqrt{\mathrm{KL}[p^{(j)}_{\boldsymbol{x},y} | p^{(j-1)}_{\boldsymbol{x},y}]}$. However, we note that our simple evaluation metric cannot explain all the reasons why the proposed method works since real-world datasets include various factors not considered in this study (e.g., class imbalance). Wang et al. (2022) mentioned that the sequence of intermediate domains should be placed evenly along the geodesic between the source and target domains. When only one intermediate domain is given, intuitively, a preferable intermediate domain lies near the midpoint of the geodesic between the source and target domains.

Here, we introduce the JS divergence (Lin, 1991) as

$$\mathrm{JS}[p_{\boldsymbol{x},y}^{(j-1)}|p_{\boldsymbol{x},y}^{(j+1)}] = 0.5\mathrm{KL}[p_{\boldsymbol{x},y}^{(j-1)}|\tilde{p}_{\boldsymbol{x},y}^{(j)}] + 0.5\mathrm{KL}[p_{\boldsymbol{x},y}^{(j+1)}|\tilde{p}_{\boldsymbol{x},y}^{(j)}], \tag{4.10}$$

where $\tilde{p}_{\boldsymbol{x},y}^{(j)} = 0.5p_{\boldsymbol{x},y}^{(j-1)} + 0.5p_{\boldsymbol{x},y}^{(j+1)}$. We suppose that $\tilde{p}^{(j)}$ in Eq. (4.10) is an ideal intermediate domain connecting $p^{(j-1)}$ and $p^{(j+1)}$. If $p^{(j)}$ is an ideal intermediate domain, we can use $p^{(j)}$ instead of $\tilde{p}^{(j)}$ in Eq. (4.10). We define the computation method that uses $p^{(j)}$ instead of $\tilde{p}^{(j)}$ as

$$\beta(j) = 0.5\mathrm{KL}[p_{\boldsymbol{x},y}^{(j-1)}|p_{\boldsymbol{x},y}^{(j)}] + 0.5\mathrm{KL}[p_{\boldsymbol{x},y}^{(j+1)}|p_{\boldsymbol{x},y}^{(j)}].$$

Finally, we define an evaluation metric for each intermediate domain as

$$E(j) = \frac{\beta(j) - \mathrm{JS}[p_{\boldsymbol{x},y}^{(j-1)}|p_{\boldsymbol{x},y}^{(j+1)}]}{\mathrm{KL}[p_{\boldsymbol{x},y}^{(1)}|p_{\boldsymbol{x},y}^{(K)}]}. \tag{4.11}$$

To make comparisons between different datasets, the numerator is divided by the KL divergence between the source and the target domains. Figure 4.3 shows a schematic of the evaluation metric given by Eq. (4.11). In practice, we cannot calculate the discrepancies between consecutive domains since the labeled data from the intermediate domains and the target domain are not available. Moreover, we cannot select arbitrary intermediate domains for domain adaptation, and a sequence of intermediate domains is only given.

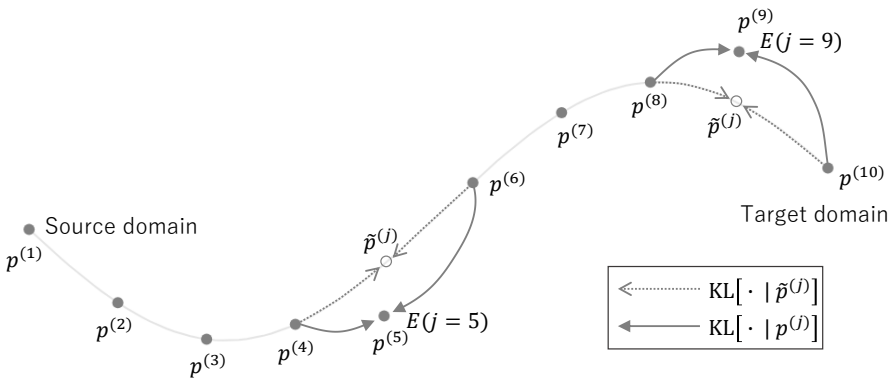


Figure 4.3: A schematic of the evaluation metric for each intermediate domain.

Using the toy dataset shown in Figure 4.4, we check the correlation between the performance of our proposed method and $E(j)$. We assign class labels 1 and 2 to the samples from the Gaussian distributions with mean vectors $(3, 1)^\top$ and $(-3, 1)^\top$, respectively, and use these samples as the source domain. The target domain consists of samples from the Gaussian distributions with mean vectors $(3, 5)^\top$ and $(-3, 5)^\top$. We use only one intermediate domain for training and evaluate the intermediate domain by computing $E(j = 2)$. The evaluation was repeated five times using different initial weights of neural networks. As shown in Table 4.2, when the absolute value of the first component of the mean vector of the intermediate domain is large, $E(j = 2)$ also increases, leading to poorer performance of the proposed method.

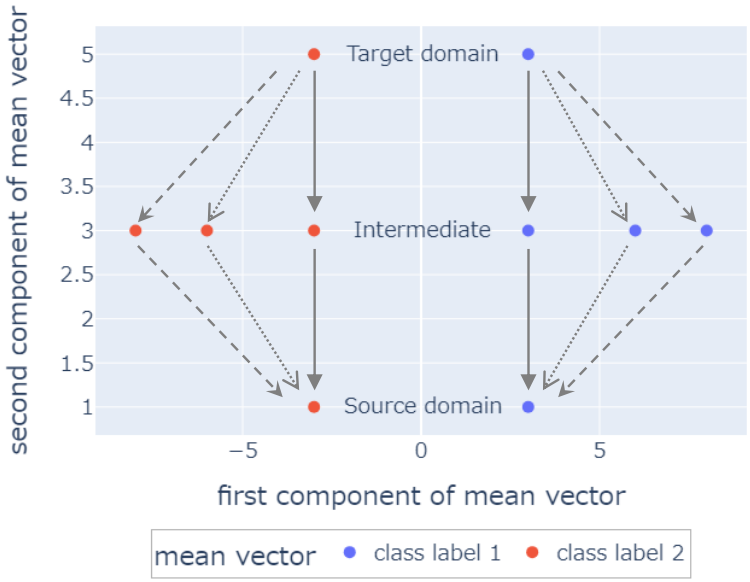


Figure 4.4: A schematic of the toy dataset evaluated in Table 4.2.

Table 4.2: Experimental results on the toy dataset with various intermediate domains. The column labeled "mean vector" refers to the absolute value of the first component of the mean vector in the intermediate domain.

mean vector	$E(j = 2)$	Accuracy
3	0.242	0.996 ± 0.0003
6	0.783	0.992 ± 0.0040
8	1.162	0.705 ± 0.2627

4.4.3 Capturing continuous change with discrete normalizing flows

We aim to capture the continuous change between domains using NFs. In principle, the DNFs, which are trained with log-likelihood based on kNN as described in Section 4.3.2, can convert a non-parametric distribution into another non-parametric distribution. However, it is empirically shown that DNFs are unsuitable for capturing the continuous change between domains.

We use RealNVP (Dinh et al., 2017) as a DNF. One DNF block consists of four fully connected layers with 64 nodes in each layer. To improve the expressive power of the flow-based model, we stack three DNF blocks. In the stacked DNF model, the input data are processed by each DNF block sequentially. Specifically, the input of the second DNF block is the output from the first DNF block. The output from each DNF block should vary continuously. We train the DNFs with the source and the intermediate datasets on the `Two Moon` dataset. Note that the CNF only uses label information when transforming samples from the source domain to samples from the Gaussian mixture distribution and does not use label information when transforming samples from the intermediate domain to samples from the source domain. Figure 4.5 shows the transformation of the intermediate data to the source data. We see that the CNF continuously transforms the intermediate data into the source data. On the other hand, in the DNFs, the path of transformation from the intermediate data into the source data is not continuous. From this preliminary experiment, we adopt CNF to realize the proposed GDA method.

4.4.4 Estimation of log-likelihood by fitting Gaussian mixture distribution

Our proposed method learns the transformation of a sample from the j-th domain to a sample from the adjacent domain, and it requires the estimation of the log-likelihood $\log p^{(j-1)}(g_\omega(\boldsymbol{x}^{(j)}, j-1))$. We proposed a computation method for the log-likelihood by using the kNN estimators in Section 4.3.2. Here, as another way of estimating the log-likelihood, we consider the approximation of $p^{(j-1)}$ by a Gaussian mixture distribution. Let Q and $w_q^{(j-1)}$ be the number of mixture components and a mixture weight, respectively. The subscript q indicates the q-th component of a Gaussian mixture

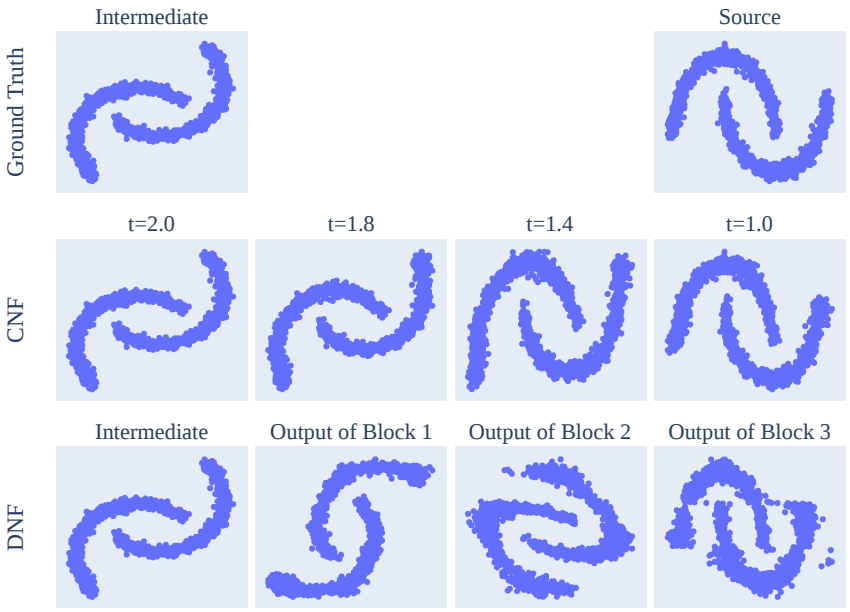


Figure 4.5: Comparison between the discrete and the continuous NFs. Whereas continuous NF is suitable for capturing continuous change, discrete NFs are unsuitable for capturing continuous change.

distribution, and the superscript indicates the domain. We approximate $p^{(j-1)}$ with the Gaussian mixture distribution

$$\hat{p}^{(j-1)}(\boldsymbol{x}) = \sum_{q=1}^{Q} w_q^{(j-1)} \mathcal{N}(\boldsymbol{x}^{(j-1)} | \boldsymbol{\mu}_q^{(j-1)}, \Sigma_q^{(j-1)}), \quad \sum_{q=1}^{Q} w_q^{(j-1)} = 1,$$

where $\boldsymbol{\mu}_q^{(j-1)}$ and $\Sigma_q^{(j-1)}$ are the mean vector and the covariance matrix, respectively. We fit the Gaussian mixture distribution for each domain and distinguish the mixture weight, mean vector, and covariance matrix with superscripts. We assign a sufficiently large value to Q since our aim is an estimation of the log-likelihood $\log p^{(j-1)}(g_\omega(\boldsymbol{x}^{(j)}, j-1))$.

We compare the log-likelihood estimation by the fitted Gaussian mixture distributions with that by the kNN estimators on the `Two Moon` dataset, and conclude that the kNN estimators are suitable for our proposed method. Since our toy dataset is a simple `two-moon` forms, $Q = 30$ should be enough for modeling its distribution with high precision. Figure 4.6 shows the comparison of the transformation by the CNF trained with kNN estimators and that trained with the Gaussian mixture distributions for

evaluating the log-likelihood. Whereas the CNF trained with the kNN estimators transforms the target data to the source data as expected, the CNF trained with the fitted Gaussian mixture distributions fails to do so.

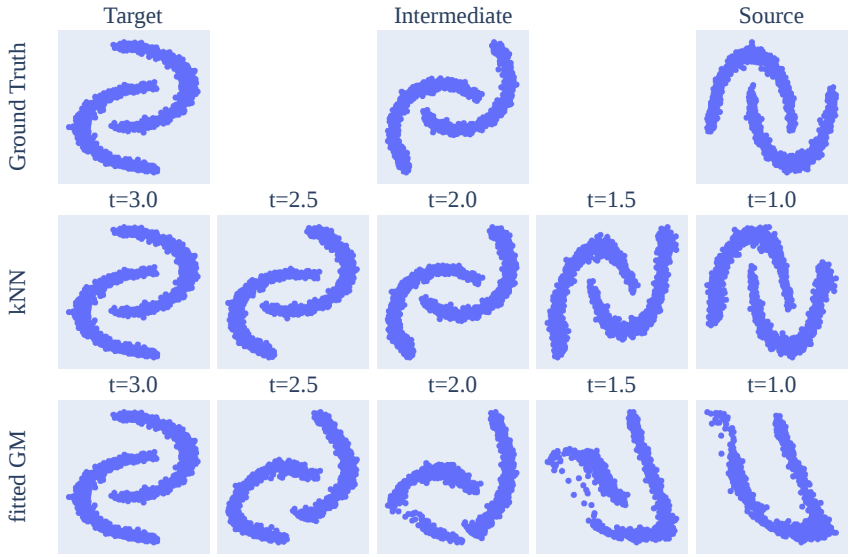


Figure 4.6: Comparison of the methods of estimating $\log p^{(j-1)}(g_\omega(\boldsymbol{x}^{(j)}, j-1))$. The CNF trained with the kNN estimators transforms the target data to the source data as expected

4.4.5 Hyperparameters

The proposed method has two hyperparameters, k and r. These hyperparameters are introduced in Sections 4.3.2 and 4.3.3, respectively. The hyperparameter k affects the result of the computation of log-likelihood by the kNN estimators, and the hyperparameter r controls the distance between the Gaussian distributions corresponding to each class. In this section, from a practical view point, we discuss how to tune these hyperparameters.

First, we discuss the tuning method of k. The hyperparameter k controls the trade-off between bias and variance of the estimate of the log-likelihood. The parameter k is similar to the kernel bandwidth parameter in the kernel density estimation (Sugiyama et al., 2012), in which large k results in larger bias and smaller variance and vice versa. We suppose that an appropriate k can be determined by fitting a kNN classifier on the

source dataset. We change the font to emphasize that the kNN estimator and the kNN classifier are different notions. We train the kNN classifier with only the source dataset and determine k from the result of three-fold cross-validation.

We vary the hyperparameter k and train our flow-based model g_ω. After the training, we evaluate the accuracy on the target dataset. The evaluation of g_ω was repeated ten times using different initial weights of neural networks. In Figure 4.7, the red dashed line denotes the k determined using the result of the kNN classifier fitting. The appropriate value of k for the kNN estimator can be determined roughly by the fitting of the kNN classifier on the source dataset. However, for the `Tox21 RingCount` dataset, the appropriate k is not determined by this method, and there is room for improvement.

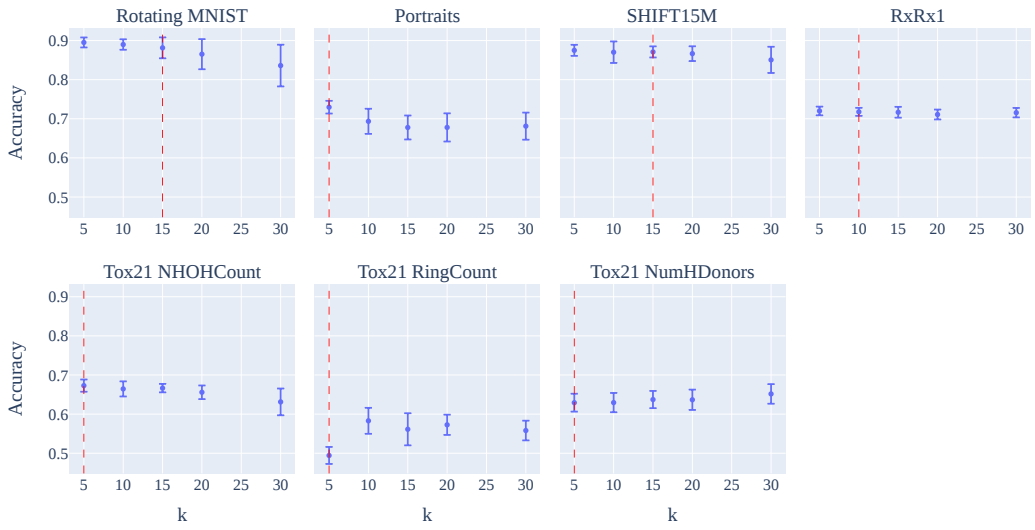
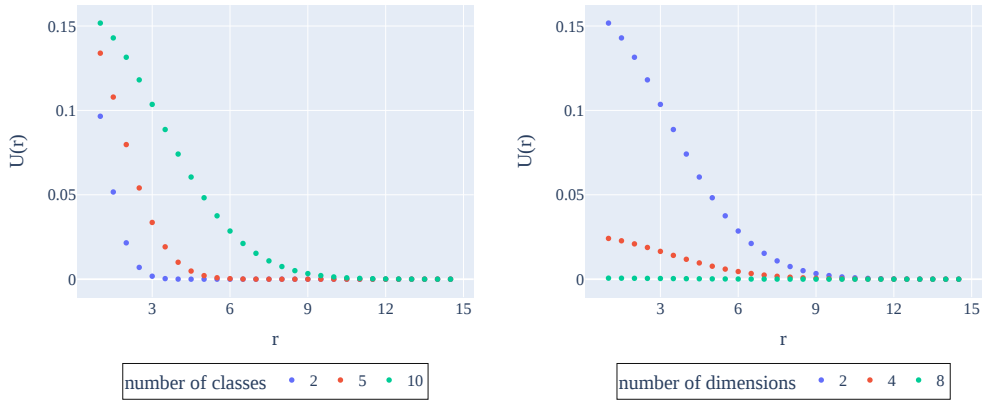


Figure 4.7: Experimental result of the training of our flow-based model with various hyperparameter k values. The appropriate k can be determined roughly by the fitting of the kNN classifier on the source dataset.

Next, we discuss a tuning method for r. In Eq. (4.7), the label of a sample from an arbitrary domain is predicted by transforming the sample into the sample from the Gaussian mixture distribution. If the setting of r is inappropriate, as shown in Figure 4.2 (e) and (f), it is hard to predict the label of a sample. The Gaussian distributions for different labels should be distinguishable from each other.

The hyperparameter r can be roughly determined by considering the number of classes and the number of dimensions of data. Our idea is simple. We evaluate the distance between the mean vectors of two adjacent Gaussian distributions. When the

distance is large, the Gaussian distributions for different labels are distinguishable from each other, as shown in Figure 4.2 (c) and (d). Let $\boldsymbol{m}(r)$ be the midpoint vector of the mean vectors $\boldsymbol{\mu}_s$ and $\boldsymbol{\mu}'_s$ of two adjacent Gaussian distributions. Since the mean vectors are set at equal intervals (see Eq. (4.4) and Figure 4.9 (d)), any pair is acceptable when they are adjacent. We propose a method to determine r by calculating $\max(\mathcal{N}(\boldsymbol{m}(r)|\boldsymbol{\mu}_s, \Sigma_s), \mathcal{N}(\boldsymbol{m}(r)|\boldsymbol{\mu}'_s, \Sigma'_s))$, and for simplicity, we denote it as $U(r)$. When the distance between the two mean vectors is sufficiently large, the likelihood of the midpoints is almost zero, regardless of which Gaussian distribution is assumed. We select a sufficiently small r such that $U(r) \simeq 0$ since the Gaussian distributions for different labels should be separable. As shown in Figure 4.8, the r that satisfies $U(r) \simeq 0$ varies depending on the number of classes and the number of dimensions of data. Note that this is not an experiment with a specific dataset. In Figure 4.8 (a), when the number of dimensions is fixed at 2, it can be seen that a larger r is required with a large number of classes. In Figure 4.8 (b), the number of classes is fixed at 10, and we see that $U(r)$ becomes sufficiently small even for a small r when the number of dimensions is large.



(a) Number of dimensions is fixed.

(b) Number of classes is fixed.

Figure 4.8: Determination method of hyperparameter r. We can roughly determine the hyperparamter r by considering the number of classes and the number of dimensions of data.

We vary the hyperparameter r and train the proposed flow-based model g_ω with only the source dataset, i.e., we maximize the log-likelihood given by Eq. (4.5) with respect to ω. The performance of our flow-based model is evaluated using three-fold

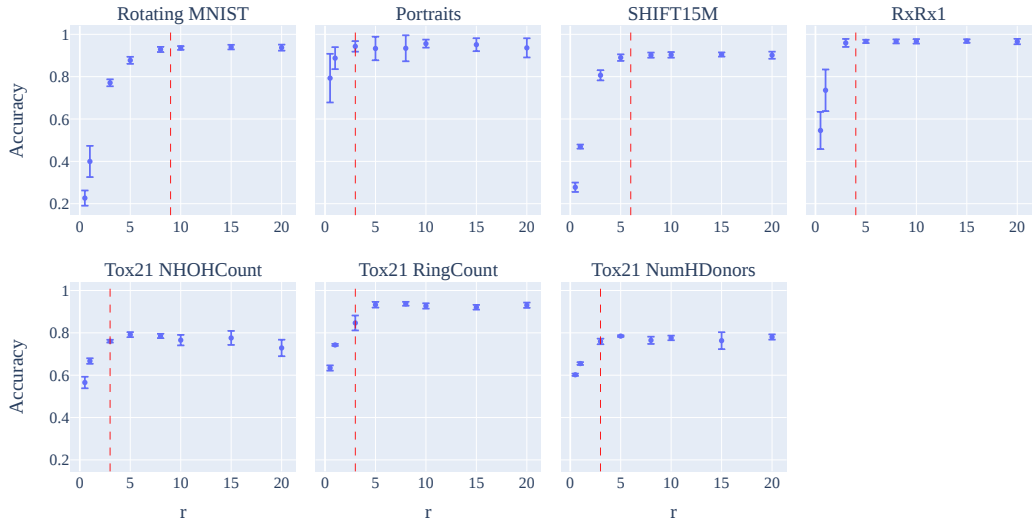


Figure 4.9: Experimental results of the training of our flow-based model with various hyperparameter r values on the source dataset only. When the hyperparameter r is sufficiently large, the accuracy on the source dataset does not change significantly. The embedding dimension for all dataset is four.

cross-validation on the source dataset. Figure 4.9 shows the result of mean accuracy of three-fold cross-validation. The red dashed line in Figure 4.9 represents the smallest r, which induces $U(r) < 0.001$. We see that the accuracy on the source dataset does not change significantly with a sufficiently large r, which means that the Gaussian distributions for different labels are distinguishable from each other. Therefore, we can determine the hyperparameter r by calculating $U(r)$.

4.4.6 Necessity of intermediate domains

Here, we show the necessity of the intermediate domains for the training of the CNF. In Figure 4.10, using the `Block` dataset, we show the transformation from the target data to the source data by the CNF trained with and without the intermediate datasets. Visually, the CNF trained with the intermediate dataset transforms the target data to the source data as expected. The accuracy on the target dataset is 0.999 when the CNF is trained with the intermediate datasets. In contrast, the accuracy on the target dataset is 0.181 when the CNF is trained without intermediate datasets. From these results, we conclude that it is important to train a CNF with datasets from the source, intermediate,

and target domains to capture continuous change between domains.

In the `Rotating MNIST` dataset, the source dataset consists of samples from the `MNIST` dataset without any rotation, and the target dataset is prepared by rotating images of the source dataset by angle $\pi/3$. We suppose that a suitable rotation angle for the intermediate domain is $\pi/6$. To study the effect of the intermediate domains, we prepare various intermediate domains with different rotation angles. Note that the number of intermediate domains used for training is always one. We compute $E(j = 2)$ as Eq. (4.11) and train our flow-based model. After the training, we evaluate the accuracy on the target dataset. The evaluation of our flow-based model was repeated ten times using different initial weights of neural networks. Table 4.3 and Figure 4.11 show the results of the experiments. We confirm that $E(j = 2)$ tends to decrease for rotation angles near $\pi/6$ and to increase when the rotation angle is either small or large. Additionally, we see that the accuracy tends to be higher when $E(j = 2)$ is small. The model trained without intermediate domains performs the worst. The arrangement of the intermediate domains between the source and target domains affects the results of GDA, but the impact is relatively small compared to the result obtained without the intermediate domain.

Table 4.3: Summary of the training of CNF with various intermediate domains.

Rotation angle	$E(j = 2)$	Accuracy
$\pi/21.0$	0.299	0.875 ± 0.040
$\pi/10.5$	0.276	0.885 ± 0.017
$\pi/7.0$	0.269	0.882 ± 0.013
$\pi/6.0$	0.278	0.881 ± 0.027
$\pi/5.25$	0.297	0.882 ± 0.020
$\pi/4.2$	0.306	0.850 ± 0.131
$\pi/3.5$	0.308	0.809 ± 0.178
w/o intermediate	NA	0.796 ± 0.140

4.4.7 Byproduct of using normalizing flows

While our primary focus lies in GDA, our proposed method can generate synthetic data from intermediate domains, even in the absence of observed samples from that domain. Recall that the proposed method does not need to generate pseudo-intermediate domains when predicting the label of a sample. In principle, the proposed method is applicable to

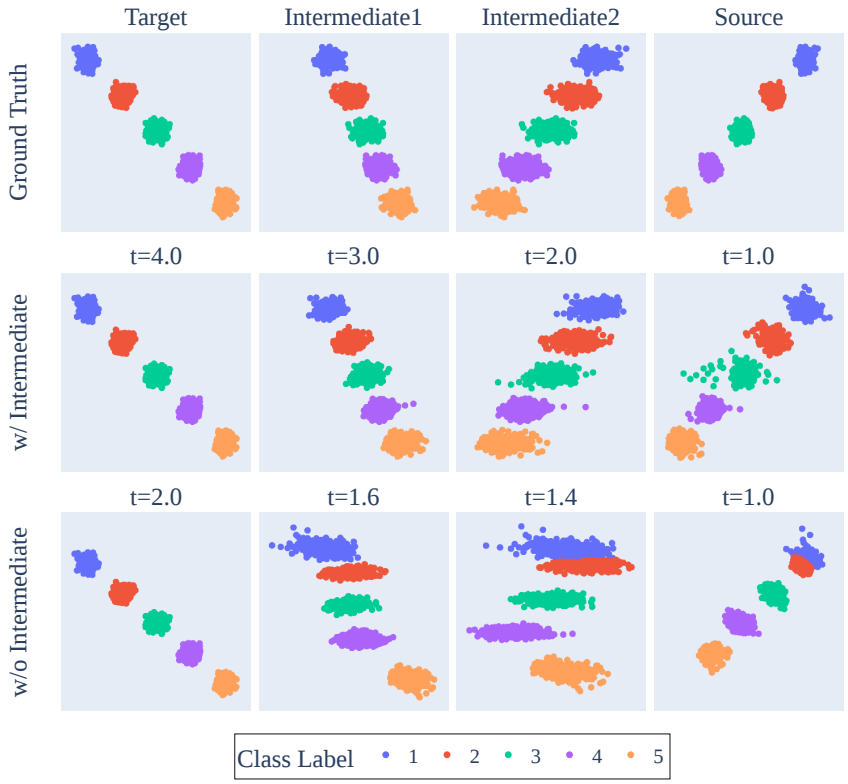


Figure 4.10: Necessity of intermediate domains. The total number of domains with and without the intermediate datasets is four and two, respectively. We consider the index of each domain j as a particular time point. Thus, the times t represented in the second and third rows of the figure are different. The CNF trained with the intermediate dataset transforms the target data to the source data as expected.

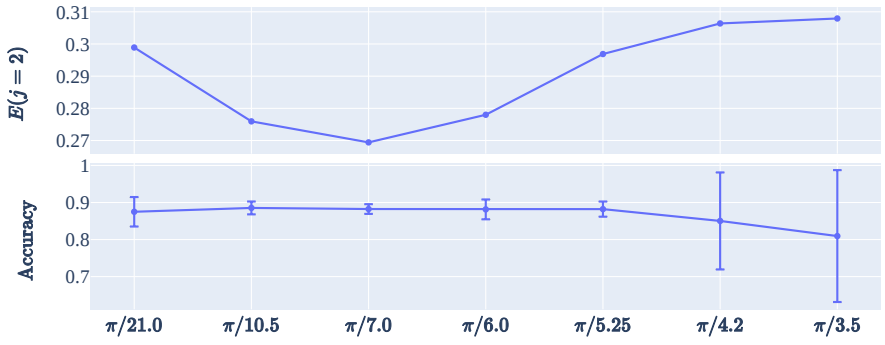


Figure 4.11: Experimental results of the training of CNF with various intermediate domains.

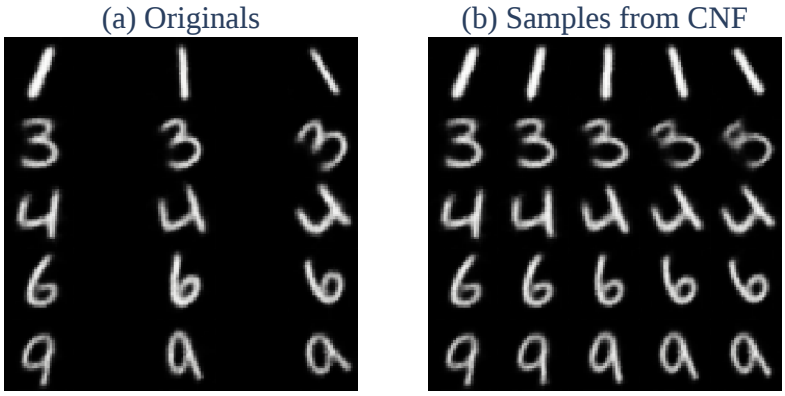


Figure 4.12: Morphing of `Rotating MNIST`.

any dimensional data as input, such as image data, but the current technology of CNF is the computational bottleneck. It is difficult to handle high-dimensional data directly due to the high-computational cost of off-the-shelf CNF implementation. Therefore, as a preprocessing, it is reasonable to reduce the dimensionality of high-dimensional data. A combination of the proposed method and VAE is applicable to image data. We can utilize the trained CNF and VAE for generating artificial intermediate images such as morphing. We show a demonstration of morphing on `Rotating MNIST` in Figure 4.12.

4.4.8 Comparison with baseline methods

To verify the effectiveness of the proposed method, we compare it with the baseline methods. Following the approach described in Section 4.4.5, we assign different values to the hyperparameter k of the proposed method for each dataset. The hyperparameter r is set to $r = 3$ and $r = 10$ for binary and multiclass classification, respectively. The appropriateness of these settings was discussed in Section 4.4.5. We summarize the baseline methods in Table 4.4 and provide a brief description of the baseline methods as follows.

- SourceOnly: Train the classifier with the source dataset only.
- GradualSelfTrain (Kumar et al., 2020): Apply gradual self-training with the initially given domains.

- Generative Gradual Domain Adaptation with Optimal Transport (GOAT) (He et al., 2023): Interpolate the initially given domains with optimal transport and apply gradual self-training.
- Gradual Interpolation of Features toward Target (GIFT) (Abnar et al., 2021): This is a conventional domain adaptation method, and this method updates the source model by gradual self-training with pseudo-intermediate domains generated by the source and target domains.
- Sequential GIFT (Abnar et al., 2021): Apply the GIFT algorithm with the initially given domains.
- Self-Training of Auxiliary models (AuxSelfTrain) (Zhang et al., 2021): This is a conventional domain adaptation method, and this method updates the source model through gradual self-training, which incorporates unsupervised learning, using pseudo-intermediate domains. The approach of generating pseudo-intermediate domains from the source and target domains differs from that of GIFT.
- Sequential AuxSelfTrain (Zhang et al., 2021): Apply the AuxSelfTrain algorithm with the initially given domains.
- Evolution Adaptive Meta-Learning (EAML) (Liu et al., 2020): Apply the meta-learning algorithm to the initially given domains.

Table 4.4: Summary of the baseline methods

Method	Domains to be used	Approach	# of domains to be adapted
Ours	all	normalizing flows	single
SourceOnly	source	NA	NA
GradualSelfTrain (Kumar et al., 2020)	all	gradual self-training	single
GOAT (He et al., 2023)	all	gradual self-training	single
GIFT (Abnar et al., 2021)	source, target	gradual self-training	single
Sequential GIFT (Abnar et al., 2021)	all	gradual self-training	single
AuxSelfTrain (Zhang et al., 2021)	source, target	gradual self-training	single
Sequential AuxSelfTrain (Zhang et al., 2021)	all	gradual self-training	single
EAML (Liu et al., 2020)	all	meta-learning	multiple

The primary baseline methods are self-training-based GDA methods, as introduced in Section 2.2. Recall that these methods update the classifier $h : \mathcal{X} \to \mathcal{Y}$ trained on the source dataset by applying gradual self-training. The key idea of self-training-based

GDA is that the classifier h should be updated gradually. We follow the classifier used by He et al. (2023) since they consider the same problem settings as ours. The classifier h consists of an encoder and a classifier. The encoder has two convolutional layers, and the classifier has two convolutional layers and two fully connected layers. Since we apply UMAP to all datasets as preprocessing, we modify the convolutional layer of the model to fully connected layers. This classifier is used in all self-training based methods and in the baseline method which is trained only on the source dataset.

The GIFT (Abnar et al., 2021) and the AuxSelfTrain (Zhang et al., 2021) are methods that apply the idea of Kumar et al. (2020) to conventional domain adaptation. While conventional (non-gradual) domain adaptation is beyond the scope of this dissertation, we limit our comparisons to those methods inspired by Kumar et al. (2020). Although GIFT and AuxSelfTrain do not utilize intermediate domains, we also show experimental results using intermediate domains for updating the classifier sequentially (Sequential GIFT and Sequential AuxSelfTrain).

The EAML (Liu et al., 2020) is a method that uses meta-learning to adapt to a target domain that evolves over time (i.e., the sequence of the target domain is given). While the goal of GDA is to achieve accurate predictions for a single target domain, EAML aims to achieve accurate predictions across a sequence of target domains. In the other study of GDA (Dong et al., 2022), EAML has been used as a baseline method. Thus, we also compare the proposed method with EAML. In EAML, the model consists of a feature extractor and an adapter that predicts the class label of a sample. It is crucial to adapt to the current domain while retaining knowledge of the previous domain. When the model adapts from the previous domain to the current domain, penalties are imposed for significant changes in the parameters of the adapter. In Liu et al. (2020), the feature extractor consists of two convolutional layers, and the adapter comprises two fully connected layers. We modify the convolutional layers of the feature extractor to fully connected layers since we apply UMAP to all datasets as preprocessing. Other necessary parameters during training are set as specified in Liu et al. (2020).

First, we discuss the performance of the proposed method on the real-world datasets. We compute $E(j = 2)$ as Eq. (4.11) for each dataset. Note that all real-world datasets have only one intermediate domain, i.e., $K = 3$. We train our flow-based model with and without the intermediate dataset and evaluate the accuracy on the target dataset. These accuracies are denoted as A and A', respectively. The contribution of the intermediate

domain is computed as

$$\frac{A - A'}{A'}.$$

When $\frac{A-A'}{A'}$ is large, the proposed method is suitable for the dataset. Each evaluation was repeated ten times using different initial weights of neural networks. In Table 4.5, we see that $\frac{A-A'}{A'}$ tends to be higher when the value of $E(j = 2)$ is small. The behavior of the `Tox21 RingCount` dataset is significantly different from the other datasets. However, as mentioned in Section 4.4.2, it is difficult to explain all the performance of the proposed method on real datasets with a simple measure.

Next, we show the result of the comparative experiment in Figure 4.13. The performance of the proposed method is not particularly high for the `Tox21 RingCount` dataset. As shown in Figure 4.7, the appropriate k for the dataset was not selected using the method proposed in Section 4.4.5. We set the hyperparameter of the proposed method to $k = 10$ and re-evaluate it on the `Tox21 RingCont` dataset. The experimental results are shown in Figure 4.14. In the `Tox21 RingCont` dataset, the performance of the proposed method is improved. Our proposed method, with the appropriate hyperparameter k, has comparable or superior accuracy to the baseline methods on all datasets. It remains as future work to develop a method for selecting the hyperparameter k with broad applicability.

The AuxSelfTrain is the only method among the baseline methods that incorporate unsupervised learning during self-training. The AuxSelfTrain seems to be suitable for the `Portraits` dataset, but it does not appear to be suitable for the `SHIFT15M` dataset. The EAML learns meta-representations from the sequence of unlabeled datasets. In our problem setting, the number of given intermediate domains is limited, which may be insufficient for learning meta-representations.

Table 4.5: Evaluation results of the proposed method on real-world datasets.

Name	Accuracy		$\frac{A-A'}{A'}$	$E(j = 2)$
	w/ intermediate (A)	w/o intermediate (A')		
Rotating MNIST (Kumar et al., 2020)	0.881 ± 0.027	0.796 ± 0.140	0.107	0.278
Portraits (Ginosar et al., 2015)	0.730 ± 0.016	0.720 ± 0.030	0.013	0.581
SHIFT15M (Kimura et al., 2021)	0.871 ± 0.014	0.868 ± 0.016	0.004	0.621
RxRx1 (Taylor et al., 2019)	0.718 ± 0.010	0.689 ± 0.076	0.041	1.166
Tox21 NHOHCount (Thomas et al., 2018)	0.673 ± 0.016	0.679 ± 0.011	−0.009	15.903
Tox21 RingCount (Thomas et al., 2018)	0.494 ± 0.022	0.552 ± 0.022	−0.105	0.756
Tox21 NumHDonors (Thomas et al., 2018)	0.629 ± 0.023	0.631 ± 0.020	−0.002	3.319

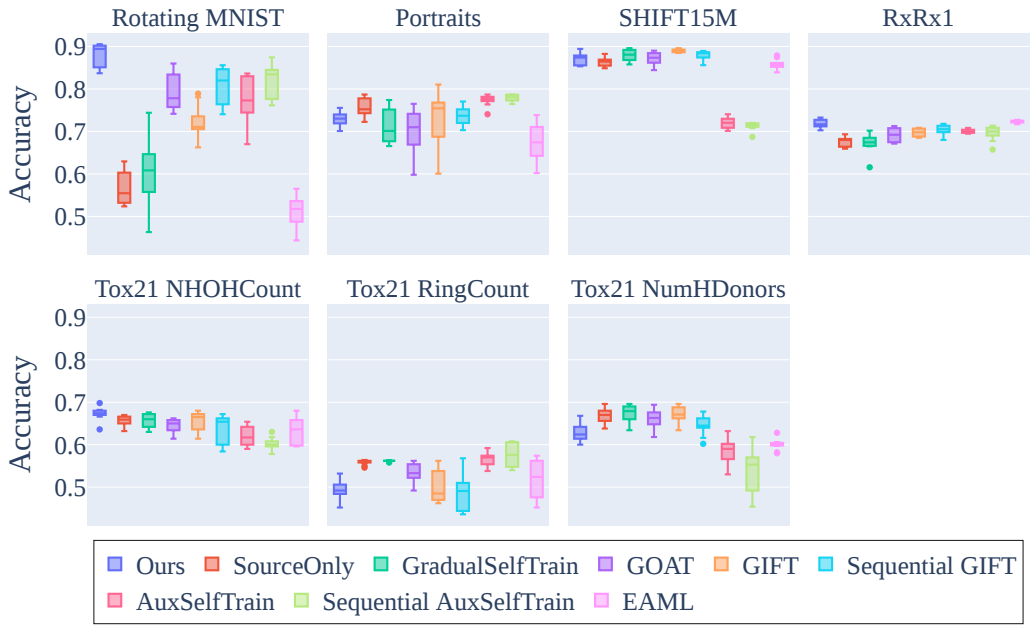


Figure 4.13: Comparison of accuracy on five real-world datasets.

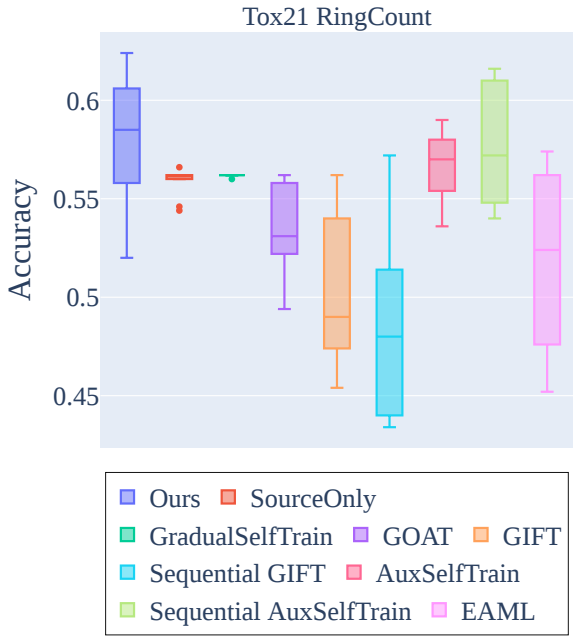


Figure 4.14: Comparison of accuracy on the `Tox21 RingCount` dataset. The setting of the hyperparameter k of the proposed method is different between Figure 4.13 and this figure.

4.5 Discussion

In this chapter, we tackle the problem of large discrepancies between adjacent domains using a framework of unsupervised domain adaptation. The self-training-based GDA methods are not effective in this situation. A naive approach to address the issue is to interpolate the large gaps between adjacent domains with pseudo-intermediate domains, and then apply gradual self-training to the dense sequence of unlabeled datasets. We attempt to capture the continuous change between domains using NFs, instead of interpolating between domains. Our flow-based model is designed for sequential domains, with a theoretical guarantee on the prediction error in the target domain. The proposed method has demonstrated stable and reliable performance, making it a strong candidate for a GDA problem in which there are large discrepancies between adjacent domains.

5 Conclusion

In this dissertation, we focused on a GDA problem in which there are large discrepancies between adjacent domains since the number of accessible intermediate domains is limited. As shown in Table 3.2 of Section 3.4, the prediction performance of self-training-based GDA method deteriorates when the discrepancies of adjacent domains is large. When the applicability of self-training-based GDA is constrained due to the limited number of available intermediate domains, one approach is to discard the intermediate domains and apply conventional domain adaptation only to the source and target domains. We argue that the intermediate domains are valuable and should be used, even when the gaps between adjacent domains are large. To address this issue, in Chapters 3 and 4, we developed two GDA methods that do not use self-training. Both of the proposed methods have demonstrated reliable performance, making them strong candidates for a GDA problem in which there are large discrepancies between adjacent domains.

In Chapter 3, we developed a method for a semi-supervised GDA problem setting, in which we can query the label of a sample from both the intermediate and target domains. In our problem setting, the query cost is highest in the target domain, and the query cost

in the intermediate domain is higher when it is closer to the target domain and lower when it is closer to the source domain. The proposed method distributes the budget to each domain and requests efficient queries by utilizing the notion of multifidelity active learning. Since the multifidelity algorithm is only provided with information about fidelity and cost, it may distribute a budget to domains with low uncertainty, potentially wasting the budget. A stopping criterion for active learning (Ishibashi and Hino, 2020) mitigates this issue.

Recall that the goal of GDA is reducing the loss in the target domain. One considerable approach is to design an acquisition function that directly reduces uncertainty in the target domain. However, it is not easy to design the acquisition function since our problem setting involves changes in distributions.

The proposed method has demonstrated stable and reasonable performance under a multifidelity setting. However, the proposed method is not theoretically guaranteed on the prediction error in the target domain. Zhou et al. (2022b) proposed a domain generalization method for online settings, which aims to reduce the loss across all domains by utilizing self-training and requests of queries. While Zhou et al. (2022b) have derived the generalization error bound for their proposed online algorithm, they have not considered the query costs. It remains as future work to derive a generalization error bound for our proposed method based on the techniques used in the proof for the theoretical guarantee of the domain generalization method proposed by Zhou et al. (2022b).

We considered a problem setting with explicit querying costs, although problem settings with unknown costs are also possible. In Section 3.2, we introduced the molecular simulation as a concrete example of semi-supervised GDA, where the number of carbons in the input compound varies across domains. In reality, the time required for simulation is different for each compound, and the exact cost is unknown. In the problem setting where the query costs are unknown, it is necessary to estimate both the label and the query cost from the input, making it more challenging than the problem setting considered in Chapter 3.

In Chapter 4, we proposed a method to capture continuous changes between domains utilizing continuous normalizing flow (CNF). As shown in Figures 4.6 and 4.10, our proposed method successfully captured the continuous changes between domains. The limitation of the proposed method is scalability as discussed in Section 4.3.5. Recall

that the CNF g describes the transformation of samples over time t using an ordinary differential equation (ODE), and the ODE is defined as

$$\frac{\partial \boldsymbol{g}}{\partial t} = v(g(\cdot, t), t; \omega),$$

where v is a neural network with the parameter ω and $\boldsymbol{g}$ is the output of $g(\cdot, t)$. Generally, the computational cost of CNF is high since the number of evaluations of the function v in the ODE solver is large. The proposed method learns transformations among multiple distributions using CNF, and the scalability of the proposed method is limited.

Accelerating the computation of CNF is an important challenge, and several approaches have been proposed. Huang and Yeh (2021) added a regularization term to the objective function of CNF to reduce the number of evaluations of the function v in the ODE solver. Onken et al. (2021) utilized optimal transport theory to accelerate the computation of CNF. Both of these methods force the trajectory of the ODE to follow a straight path. We aim to capture the continuous change between domains using CNFs, and it remains unclear whether the aforementioned accelerated CNFs are suitable for capturing such continuous changes.

Our proposed method can generate samples since it utilizes normalizing flow, one of the generative models. An example of interpolative data generation by the proposed method has been shown in Figure 4.12. When samples from new intermediate domains are added and the model needs to be retrained, the interpolative data generation is useful. Since the proposed method can generate samples from the observed domains, there is no need to store the learned data. The model can be retrained using both the samples from these new intermediate domains and the samples that are generated by the model prior to retraining. The technique of using a generative model to generate samples for retraining is known as generative replay in continuous learning (Wang et al., 2024). In addition to interpolation, extrapolation is also known as a challenging task in the field of data generation. Chen et al. (2018) demonstrated an example of extrapolative data generation using NeuralODE. For instance, a CNF that is trained with images of cars from the past to the present might suggest designs for future cars.

In GDA, it is assumed that the intermediate domains are arranged to connect the source domain to the target domain. In previous studies, it has been assumed that the Wasserstein distances between the adjacent domains are small. However, this

assumption, based on the Wasserstein distances between the adjacent domains, does not explicitly describe the configuration of intermediate domains. We argue that an assumption that describes the configuration of intermediate domains is naturally suited to the concept of GDA. In previous studies, the discrepancies between adjacent domains were measured using the Wasserstein distance, while in Chapter 4, we measured these discrepancies using KL divergence. It remains unclear which metric is suitable for GDA. Intuitively, as the classifier adapts to each intermediate domain, the accuracy on the target dataset should improve. However, using the `Portraits` dataset, Chen and Chao (2021) showed that the accuracy on the target dataset did not improve gradually when the pre-defined domain indicator (specifically, the *years*) was used. The experimental results suggest that when a sequence of intermediate domains is provided, it is crucial to either select intermediate domains that are suitable for GDA or to rearrange the sequence. An unsuitable intermediate domain, which is not arranged to connect the source domain to the target domain, deteriorates the predictive performance of both of our proposed methods. There is a demand for a method that selects several appropriate intermediate domains for GDA from the given sequence. Chen and Chao (2021) proposed a GDA method that finds a new domain indicator and rearranges the sequence of the intermediate domains. However the method is based on self-training and might not be effective in our problem setting, where the number of available intermediate domains is limited.

Appendices

Here, we provide the details of Matthews correlation coefficient and the proofs of Propositions 4.1 and 4.2. Gorodkin (2004) derived the Matthews correlation coefficient for multi-class classification. Propositions 4.1 and 4.2 were originally derived by Nguyen et al. (2022) and Onken et al. (2021), respectively. For the sake of completeness, we provide proofs using the notations consistent with those used in this dissertation.

Matthews correlation coefficient (Gorodkin, 2004)

Continuous case: We consider Pearson's correlation coefficient between the two matrices $X, Y \in \mathbb{R}^{N \times C}$. Let X_{nk} and Y_{nk} be the elements of the matrices X and Y, and X_k and Y_k be the k-th columns of X and Y, respectively. We define the covariance between X and Y as the empirical covariance between the corresponding k-th columns:

$$\begin{aligned}\mathrm{cov}(X, Y) &= \frac{1}{C}\sum_{k=1}^{C}\mathrm{cov}(X_k, Y_k) \\ &= \frac{1}{C}\sum_{k=1}^{C}\sum_{n=1}^{N}(X_{nk} - \bar{X}_k)(Y_{nk} - \bar{Y}_k),\end{aligned}$$

where $\bar{X}_k = \frac{1}{N}\sum_{n=1}^{N} X_{nk}$ and $\bar{Y}_k = \frac{1}{N}\sum_{n=1}^{N} Y_{nk}$ are the mean values of k-th columns of X and Y, respectively. The correlation coefficient between X and Y is defined as

$$\frac{\mathrm{cov}(X, Y)}{\sqrt{\mathrm{cov}(X, X)\ \mathrm{cov}(Y, Y)}}. \tag{A.1}$$

When $C = 1$, Eq. (A.1) reduces to Pearson's correlation coefficient between two datasets.

Discrete case: Consider a multiclass classification problem. The $N \times C$ matrix X consists of one-hot vectors, where each element $X_{nk} \in \{0, 1\}$. The matrix X denotes the prediction results for N samples, with each row representing the categorical prediction in a one-hot encoded format across C possible classes. The actual labels for the N samples are shown in the matrix Y, which also consists of one-hot vectors $Y_{nk} \in \{0, 1\}$. We describe the confusion matrix calculated from X and Y as

$$M = \begin{bmatrix} M_{11} & \dots & M_{1C} \\ \vdots & \ddots & \vdots \\ M_{C1} & \dots & M_{CC} \end{bmatrix},$$

where M_{ij} corresponds to the total number of samples where the actual class is the i-th class but the predicted class is the j-th class. We introduce the following variables.

- The total number of samples that are correctly predicted: $T = \sum_{i=1}^{C} M_{ii}$.
- The sum of the elements in the k-th row of M: $R_k = \sum_{i=1}^{C} M_{ki}$.
- The sum of the elements in the k-th column of M: $A_k = \sum_{i=1}^{C} M_{ik}$.

Note that the total number of elements in the confusion matrix $\sum_{i=1}^{C} \sum_{j=1}^{C} M_{ij}$ is equivalent to N, which is the number of rows in the matrices X and Y. We have

$$\begin{aligned} \mathrm{cov}(X, Y) &= \frac{1}{C} \sum_{k=1}^{C} \sum_{n=1}^{N} (X_{nk} - \bar{X}_k)(Y_{nk} - \bar{Y}_k) \\ &= \frac{1}{C} \left(\sum_{k=1}^{C} \sum_{n=1}^{N} X_{nk} Y_{nk} - \sum_{k=1}^{C} \sum_{n=1}^{N} X_{nk} \bar{Y}_k - \sum_{k=1}^{C} \sum_{n=1}^{N} Y_{nk} \bar{X}_k + \sum_{k=1}^{C} \sum_{n=1}^{N} \bar{X}_k \bar{Y}_k \right) \\ &= \frac{1}{C} \left(\sum_{k=1}^{C} \sum_{n=1}^{N} X_{nk} Y_{nk} - \sum_{k=1}^{C} \bar{Y}_k \sum_{n=1}^{N} X_{nk} - \sum_{k=1}^{C} \bar{X}_k \sum_{n=1}^{N} Y_{nk} + N \sum_{k=1}^{C} \bar{X}_k \bar{Y}_k \right) \end{aligned} \tag{A.2}$$

We simplify the notation in Eq. (A.2). The first term $\sum_{k=1}^{C} \sum_{n=1}^{N} X_{nk} Y_{nk}$ represents the total number of correctly predicted samples since the elements $X_{nk}, Y_{nk} \in \{0, 1\}$. Thus, $\sum_{k=1}^{C} \sum_{n=1}^{N} X_{nk} Y_{nk} = T$. In the second term, $\sum_{n=1}^{N} X_{nk}$ indicates the number of samples predicted as the k-th class, and it is equivalent to A_k. Moreover, $\bar{Y}_k$ can be calculated as R_k/N. Therefore, the second term $\sum_{k=1}^{C} \bar{Y}_k \sum_{n=1}^{N} X_{nk}$ becomes $\frac{1}{N} \sum_{k=1}^{C} R_k A_k$. Similarly, the third and fourth terms becomes $\frac{1}{N} \sum_{k=1}^{C} R_k A_k$ and $\frac{1}{N} \sum_{k=1}^{C} R_k A_k$, respectively. From

the above results, we have

$$\mathrm{cov}(X, Y) = \frac{T \times N - \sum_{k=1}^{C} R_k A_k}{C \times N}. \tag{A.3}$$

Furthermore, $\mathrm{cov}(X, X)$ and $\mathrm{cov}(Y, Y)$ are given by

$$\mathrm{cov}(X, X) = \frac{N^2 - \sum_{k=1}^{C} A_k^2}{C \times N}, \quad \mathrm{cov}(Y, Y) = \frac{N^2 - \sum_{k=1}^{C} R_k^2}{C \times N}. \tag{A.4}$$

We obtain the Matthews correlation coefficient for multi-class classification by substituting Eqs. (A.3) and (A.4) into Eq. (A.1):

$$\frac{T \times N - \sum_{k=1}^{C} R_k \times A_k}{\sqrt{(N^2 - \sum_{k=1}^{C} R_k^2)(N^2 - \sum_{k=1}^{C} A_k^2)}}.$$

Proof of Proposition 4.1 (Nguyen et al., 2022)

Proof. Recall that the expected losses on the source and j-th domains are defined as $\mathrm{L}_1 = \mathbb{E}_{\boldsymbol{x},y\sim p^{(1)}_{\boldsymbol{x},y}}[-\log p(y|\boldsymbol{x})]$ and $\mathrm{L}_\mathrm{j} = \mathbb{E}_{\boldsymbol{x},y\sim p^{(j)}_{\boldsymbol{x},y}}[-\log p(y|\boldsymbol{x})]$, respectively. In the standard domain adaptation, there is no intermediate domain and $K = 2$. We have

$$\begin{aligned}
\mathrm{L}_2 &= \mathbb{E}_{p^{(2)}_{\boldsymbol{x},y}}[-\log p(y|\boldsymbol{x})] \\
&= \int -[\log p(y|\boldsymbol{x})] p_2(\boldsymbol{x}, y) d\boldsymbol{x} dy \\
&= \int -[\log p(y|\boldsymbol{x})] p_1(\boldsymbol{x}, y) d\boldsymbol{x} dy + \int -\log p(y|\boldsymbol{x})[p_2(\boldsymbol{x}, y) - p_1(\boldsymbol{x}, y)] d\boldsymbol{x} dy \\
&= \mathrm{L}_1 + \int -\log p(y|\boldsymbol{x})[p_2(\boldsymbol{x}, y) - p_1(\boldsymbol{x}, y)] d\boldsymbol{x} dy.
\end{aligned}$$

We define sets $\mathcal{A}$ and $\mathcal{B}$ as

$$\mathcal{A} = \{(\boldsymbol{x}, y) | p_2(\boldsymbol{x}, y) - p_1(\boldsymbol{x}, y) \geq 0\}, \quad \mathcal{B} = \{(\boldsymbol{x}, y) | p_2(\boldsymbol{x}, y) - p_1(\boldsymbol{x}, y) < 0\}.$$

If Assumption 4.1 holds, we have

$$
\begin{aligned}
&\int -\log p(y|\boldsymbol{x})[p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)]d\boldsymbol{x}dy \\
&=\int_{\mathcal{A}} -\log p(y|\boldsymbol{x})[p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)]d\boldsymbol{x}dy+\int_{\mathcal{B}} -\log p(y|\boldsymbol{x})[p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)]d\boldsymbol{x}dy \\
&\leq \int_{\mathcal{A}} -\log p(y|\boldsymbol{x})[p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)]d\boldsymbol{x}dy \\
&=\int_{\mathcal{A}} -\log p(y|\boldsymbol{x})|p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)|d\boldsymbol{x}dy \\
&\leq M\int_{\mathcal{A}} |p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)|d\boldsymbol{x}dy \quad (\because -\log p(y|\boldsymbol{x})\leq M),
\end{aligned}
$$

where $|\cdot|$ is the absolute value. Note that $\int_{\mathcal{A}} |p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)|d\boldsymbol{x}dy$ is called the total variation of two distributions. From the identity $\int p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)d\boldsymbol{x}dy=0$, we have

$$
\begin{aligned}
&\int_{\mathcal{A}} p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)d\boldsymbol{x}dy+\int_{\mathcal{B}} p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)d\boldsymbol{x}dy=0 \\
&\Leftrightarrow \int_{\mathcal{A}} p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)d\boldsymbol{x}dy=\int_{\mathcal{B}} p_1(\boldsymbol{x},y)-p_2(\boldsymbol{x},y)d\boldsymbol{x}dy \\
&\Leftrightarrow \int_{\mathcal{A}} |p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)|d\boldsymbol{x}dy=\int_{\mathcal{B}} |p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)|d\boldsymbol{x}dy \\
&\Leftrightarrow \int_{\mathcal{A}} |p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)|d\boldsymbol{x}dy=\frac{1}{2}\int |p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)|d\boldsymbol{x}dy.
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
\mathrm{L}_2 &= \mathrm{L}_1+\int -\log p(y|\boldsymbol{x})[p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)]d\boldsymbol{x}dy \\
&\leq \mathrm{L}_1+M\int_{\mathcal{A}} |p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)|d\boldsymbol{x}dy \\
&= \mathrm{L}_1+\frac{M}{2}\int |p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)|d\boldsymbol{x}dy.
\end{aligned}
$$

Using the Pinsker's inequality, we have

$$
\left(\int |p_2(\boldsymbol{x},y)-p_1(\boldsymbol{x},y)|d\boldsymbol{x}dy\right)^2 \leq 2\int p_2(\boldsymbol{x},y)\log\frac{p_2(x,y)}{p_1(x,y)}d\boldsymbol{x}dy.
$$

Therefore,

$$\begin{aligned}\mathrm{L}_2 &\leq \mathrm{L}_1 + \frac{M}{2}\sqrt{2\int p_2(\boldsymbol{x}, y)\log\frac{p_2(x,y)}{p_1(x,y)}d\boldsymbol{x}dy} \\ &= \mathrm{L}_1 + \frac{M}{\sqrt{2}}\sqrt{\mathrm{KL}[p^{(2)}_{\boldsymbol{x},y}|p^{(1)}_{\boldsymbol{x},y}]}.\end{aligned}$$

We decompose the KL divergence between $p^{(2)}_{\boldsymbol{x},y}$ and $p^{(1)}_{\boldsymbol{x},y}$ into the marginal and conditional misalignment terms as follows:

$$\begin{aligned}&\mathrm{KL}[p^{(2)}_{\boldsymbol{x},y}|p^{(1)}_{\boldsymbol{x},y}] \\ &= \mathbb{E}_{p^{(2)}_{\boldsymbol{x},y}}[\log p_2(\boldsymbol{x}, y) - \log p_1(\boldsymbol{x}, y)] \\ &= \mathbb{E}_{p^{(2)}_{\boldsymbol{x},y}}[\log p_2(\boldsymbol{x}) + \log p_2(y|\boldsymbol{x}) - \log p_1(\boldsymbol{x}) - \log p_1(y|\boldsymbol{x})] \\ &= \mathbb{E}_{p^{(2)}_{\boldsymbol{x},y}}[\log p_2(\boldsymbol{x}) - \log p_1(\boldsymbol{x})] + \mathbb{E}_{p^{(2)}_{\boldsymbol{x},y}}[\log p_2(y|\boldsymbol{x}) - \log p_1(y|\boldsymbol{x})] \\ &= \mathrm{KL}[p^{(2)}_{\boldsymbol{x}}|p^{(1)}_{\boldsymbol{x}}] + \mathbb{E}_{p^{(2)}_{\boldsymbol{x}}}[\mathbb{E}_{p^{(2)}_{y|\boldsymbol{x}}}[\log p_2(y|\boldsymbol{x}) - \log p_1(y|\boldsymbol{x})]] \\ &= \mathrm{KL}[p^{(2)}_{\boldsymbol{x}}|p^{(1)}_{\boldsymbol{x}}] + \mathbb{E}_{p^{(2)}_{\boldsymbol{x}}}[\mathrm{KL}[p^{(2)}_{y|\boldsymbol{x}}|p^{(1)}_{y|\boldsymbol{x}}]].\end{aligned}$$

Therefore, we have

$$\mathrm{L}_2 \leq \mathrm{L}_1 + \frac{M}{\sqrt{2}}\sqrt{\mathrm{KL}[p^{(2)}_{\boldsymbol{x}}|p^{(1)}_{\boldsymbol{x}}] + \mathbb{E}_{p^{(2)}_{\boldsymbol{x}}}[\mathrm{KL}[p^{(2)}_{y|\boldsymbol{x}}|p^{(1)}_{y|\boldsymbol{x}}]]},$$

which completes the proof. □

Proof of Proposition 4.2 (Onken et al., 2021)

Proof. Let p_t be the initial density of the samples $\boldsymbol{x} \in \mathbb{R}^d$ and $g : \mathbb{R}^d \times \mathbb{R}_{\geq 0} \to \mathbb{R}^d$ be the trajectories that transform samples from p_t to p_{t-1}. The change in density when p_t is transformed from time t to $t-1$ is given by the change of variables formula

$$p_t(\boldsymbol{x}) = p^*_t(g(\boldsymbol{x}, t-1))|\det\nabla g(\boldsymbol{x}, t-1)|, \tag{A.5}$$

where p_t^* and $\nabla g(\boldsymbol{x}, t-1)$ are the transformed density and the Jacobian of g, respectively. Normalizing flows aim to learn a function g that transforms p_t to p_{t-1}. Measuring the discrepancy between the transformed and objective distributions indicates whether the trained function g is appropriate. The discrepancy between two distributions is measured using the KL divergence

$$\mathrm{KL}[p_t^* | p_{t-1}] = \int_{\mathbb{R}^d} \log \left(\frac{p_t^*(\boldsymbol{x})}{p_{t-1}(\boldsymbol{x})} \right) p_t^*(\boldsymbol{x}) d\boldsymbol{x}. \tag{A.6}$$

We transform a sample $\boldsymbol{x}$ using g. By using the change of variable formula, we rewrite Eq. (A.6) as follows

$$\begin{aligned}
\mathrm{KL}[p_t^* | p_{t-1}] &= \int_{\mathbb{R}^d} \log \left(\frac{p_t^*(\boldsymbol{x})}{p_{t-1}(\boldsymbol{x})} \right) p_t^*(\boldsymbol{x}) d\boldsymbol{x} \\
&= \int_{\mathbb{R}^d} \log \left(\frac{p_t^*(g(\boldsymbol{x}, t-1)) \cancel{|\det \nabla g(\boldsymbol{x}, t-1)|}}{p_{t-1}(g(\boldsymbol{x}, t-1)) \cancel{|\det \nabla g(\boldsymbol{x}, t-1)|}} \right) p_t^*(g(\boldsymbol{x}, t-1)) |\det \nabla g(\boldsymbol{x}, t-1)| d\boldsymbol{x} \\
&= \int_{\mathbb{R}^d} \log \left(\frac{p_t^*(g(\boldsymbol{x}, t-1))}{p_{t-1}(g(\boldsymbol{x}, t-1))} \right) p_t^*(g(\boldsymbol{x}, t-1)) |\det \nabla g(\boldsymbol{x}, t-1)| d\boldsymbol{x} \\
&= \int_{\mathbb{R}^d} \log \left(\frac{p_t^*(g(\boldsymbol{x}, t-1))}{p_{t-1}(g(\boldsymbol{x}, t-1))} \right) p_t(\boldsymbol{x}) d\boldsymbol{x} \quad (\because \textit{Eq. (A.5)}).
\end{aligned}$$

By using Eq. (A.5), we have

$$\begin{aligned}
&\mathrm{KL}[p_t^* | p_{t-1}] \\
&= \int_{\mathbb{R}^d} \log \left(\frac{p_t^*(g(\boldsymbol{x}, t-1))}{p_{t-1}(g(\boldsymbol{x}, t-1))} \right) p_t(\boldsymbol{x}) d\boldsymbol{x} \\
&= \int_{\mathbb{R}^d} \log \left(\frac{p_t(\boldsymbol{x})}{p_{t-1}(g(\boldsymbol{x}, t-1)) |\det \nabla g(\boldsymbol{x}, t-1)|} \right) p_t(\boldsymbol{x}) d\boldsymbol{x} \\
&= \mathbb{E}_{p_t} \left[\log p_t(\boldsymbol{x}) - \{ \log p_{t-1}(g(\boldsymbol{x}, t-1)) + \log |\det \nabla g(\boldsymbol{x}, t-1)| \} \right].
\end{aligned} \tag{A.7}$$

Chen et al. (2018) proposed neural ordinary differential equations, in which a neural network v parametrized by ω represents the time derivative of the function g, as follows

$$\frac{\partial \boldsymbol{g}}{\partial t} = v(g(\cdot, t), t; \omega),$$

where $\boldsymbol{g}$ denotes the output of $g(\cdot, t)$. Moreover, they showed that the instantaneous change of the density can be computed as follows

$$\frac{\partial \log p(g(\cdot, t))}{\partial t} = -\mathrm{Tr}\left(\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{g}}\right),$$

where $\boldsymbol{v}$ is the output of v and $\partial \boldsymbol{v}/\partial \boldsymbol{g} \in \mathbb{R}^{d\times d}$. Namely, the term $\log|\det \nabla g(\boldsymbol{x}, t-1)|$ in Eq. (A.7) is equivalent to $-\int_{t-1}^{t} \mathrm{Tr}(\partial \boldsymbol{v}/\partial \boldsymbol{g})dt$. Therefore, we rewrite the Eq. (A.7) as follows:

$$\mathrm{KL}[p_t^*|p_{t-1}] = \mathbb{E}_{p_t}\left[\log p_t(\boldsymbol{x}) - \left\{\log p_{t-1}(g(\boldsymbol{x}, t-1)) - \int_{t-1}^{t} \mathrm{Tr}\left(\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{g}}\right) dt\right\}\right].$$

Recall that the log-likelihood of continuous normalizing flow is given by

$$\log p_t(g(\boldsymbol{x}, t)) = \log p_{t-1}(g(\boldsymbol{x}, t-1)) - \int_{t-1}^{t} \mathrm{Tr}\left(\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{g}}\right) dt.$$

Therefore, we have

$$\mathrm{KL}[p_t^*|p_{t-1}] = \mathbb{E}_{p_t}\left[\log p_t(\boldsymbol{x}) - \log p_t(g(\boldsymbol{x}, t))\right]. \tag{A.8}$$

The first term of Eq. (A.8) does not depend on g, and we can ignore it during the training of the CNF. Therefore, the minimization of $\mathbb{E}_{p_t}[-\log p_t(g(\boldsymbol{x}, t))]$ is equivalent to the minimization of the KL divergence between p_{t-1} and p_t transformed by g. □

Acknowledgments

First and foremost, I would like to express my profound gratitude to Prof. Hideitsu Hino (ISM) my doctoral supervisor. Despite my lack of knowledge and experience, his perseverance in guiding me was invaluable. Prof. Hino gave me not only technical advice but also taught me his attitude as a researcher. I am committed to utilizing the knowledge and skills I have acquired from ISM in my role at my current company. My aspiration is to become a conduit between corporate professionals and university researchers.

I would like to express my sincere gratitude to the members of my thesis examination committee: Prof. Ryo Yoshida (ISM), Prof. Mirai Tanaka (ISM), and Prof. Kota Matsui (Nagoya University). Prof. Yoshida provided invaluable suggestions that enhanced the quality of my dissertation, particularly focusing on the novelty and specific applications of the proposed methods. Prof. Tanaka conducted a meticulous and comprehensive review of my entire dissertation. Prof. Matsui devoted considerable attention to discussing the potential future developments of the proposed methods.

I extend my heartfelt thanks to everyone at ISM. The faculty members provided me with numerous insightful advice, which has significantly advanced my research. I am also greatly indebted to the members of the Hino lab. The seminars at Hino lab exposed me to a variety of perspectives.

I would like to express my appreciation to my superiors and colleagues at KONICA MINOLTA, INC. I am deeply grateful to my superiors for readily agreeing to my enrollment in the doctoral program. My colleagues have been instrumental in adjusting the workload for me. Without the collaboration and understanding from my superiors and colleagues, my research journey would not have reached its conclusion.

Last but not least, I want to thank my family for their unwavering support throughout

my life. Research is not always a smooth journey and indeed, there were more difficult times than not. I am certain that I would not have been able to navigate through many situations without the steadfast support of my family.

Without the support and help from these individuals, it would have been impossible for me to complete my research journey. Thank you all from the bottom of my heart.

Bibliography

Abdal, R., Zhu, P., Mitra, N. J., and Wonka, P. (2021). StyleFlow: Attribute-conditioned exploration of StyleGAN-generated images using conditional continuous normalizing flows. *ACM Trans. Graph.*, 40(3).

Abnar, S., van den Berg, R., Ghiasi, G., Dehghani, M., Kalchbrenner, N., and Sedghi, H. (2021). Gradual domain adaptation in the wild: When intermediate distributions are absent. *CoRR*, abs/2106.06080.

Ali, M., Jones, M. W., Xie, X., and Williams, M. (2019). TimeCluster: Dimension reduction applied to temporal data for visual analytics. *The Visual Computer*, 35(6):1013–1026.

Amini, M., Feofanov, V., Pauletto, L., Devijver, E., and Maximov, Y. (2022). Self-training: A survey. *CoRR*, abs/2202.12040.

Askari, H., Latif, Y., and Sun, H. (2023). MapFlow: Latent transition via normalizing flow for unsupervised domain adaptation. *Machine Learning*, 112(8):2953–2974.

Becht, E., McInnes, L., Healy, J., Dutertre, C.-A., Kwok, I. W. H., Ng, L. G., Ginhoux, F., and Newell, E. W. (2019). Dimensionality reduction for visualizing single-cell data using UMAP. *Nature Biotechnology*, 37(1):38–44.

Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Vaughan, J. W. (2010). A theory of learning from different domains. *Machine Learning*, 79(1):151–175.

Ben-Hamu, H., Cohen, S., Bose, J., Amos, B., Nickel, M., Grover, A., Chen, R. T. Q., and Lipman, Y. (2022). Matching normalizing flows and probability paths on manifolds.

In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 1749–1763.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, page 41–48.

Blackard, J. A. and Dean, D. J. (1999). Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 24(3):131–151.

Brehmer, J. and Cranmer, K. (2020). Flows for simultaneous manifold learning and density estimation. In *Advances in Neural Information Processing Systems*, volume 33, pages 442–453.

Caterini, A. L., Loaiza-Ganem, G., Pleiss, G., and Cunningham, J. P. (2021). Rectangular flows for manifold learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 30228–30241.

Chen, H.-Y. and Chao, W.-L. (2021). Gradual domain adaptation without indexed intermediate domains. In *Advances in Neural Information Processing Systems*, volume 34, pages 8201–8214.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, volume 31.

Choi, J., Choi, Y., Kim, J., Chang, J., Kwon, I., Gwon, Y., and Min, S. (2020). Visual domain adaptation by consensus-based transfer to intermediate domain. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(07):10655–10662.

Cui, S., Wang, S., Zhuo, J., Su, C., Huang, Q., and Tian, Q. (2020). Gradually vanishing bridge for adversarial domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12455–12464.

Dai, Y., Liu, J., Sun, Y., Tong, Z., Zhang, C., and Duan, L.-Y. (2021). IDM: An intermediate domain module for domain adaptive person Re-ID. In *Proceedings*

of the IEEE/CVF International Conference on Computer Vision (ICCV), pages 11864–11874.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2017). Density estimation using real NVP. In *International Conference on Learning Representations*.

Dong, J., Zhou, S., Wang, B., and Zhao, H. (2022). Algorithms and theory for supervised gradual domain adaptation. *Transactions on Machine Learning Research*.

Dong, W., Moses, C., and Li, K. (2011). Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web*, page 577–586.

Drwal, M., Siramshetty, V., Banerjee, P., Goede, A., Preissner, R., and Dunkel, M. (2015). Molecular similarity-based predictions of the Tox21 screening outcome. *Frontiers in Environmental Science*, 3.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(19):625–660.

Fefferman, C., Mitter, S., and Narayanan, H. (2016). Testing the manifold hypothesis. *Journal of the American Mathematical Society*, 29(4):983–1049.

Gadermayr, M., Eschweiler, D., Klinkhammer, B. M., Boor, P., and Merhof, D. (2018). Gradual domain adaptation for segmenting whole slide images showing pathological variability. In *Image and Signal Processing*, pages 461–469.

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., March, M., and Lempitsky, V. (2016). Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(59):1–35.

Ginosar, S., Rakelly, K., Sachs, S., Yin, B., and Efros, A. A. (2015). A century of portraits: A visual historical record of american high school yearbooks. In *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, pages 652–658.

Gong, R., Li, W., Chen, Y., and Gool, L. V. (2019). DLOW: Domain flow for adaptation and generalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2477–2486.

Gong, S., Wang, S., Xie, T., Chae, W. H., Liu, R., Shao-Horn, Y., and Grossman, J. C. (2022). Calibrating DFT formation enthalpy calculations by multifidelity machine learning. *JACS Au*, 2(9):1964–1977.

Gorodkin, J. (2004). Comparing two k-category assignments by a k-category correlation coefficient. *Computational Biology and Chemistry*, 28(5):367–374.

Grathwohl, W., Chen, R. T., Bettencourt, J., Sutskever, I., and Duvenaud, D. (2019). FFJORD: Free-form continuous dynamics for scalable reversible generative models. In *International Conference on Learning Representations*.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. (2012). A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773.

Grover, A., Chute, C., Shu, R., Cao, Z., and Ermon, S. (2020). AlignFlow: Cycle consistent learning from multiple domains via normalizing flows. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):4028–4035.

He, Y., Wang, H., Li, B., and Zhao, H. (2023). Gradual domain adaptation: Theory and algorithms. *CoRR*, abs/2310.13852.

Hino, H. (2020). Active learning: Problem settings and recent developments. *CoRR*, abs/2012.04225.

Hino, H., Koshijima, K., and Murata, N. (2015). Non-parametric entropy estimators based on simple linear regression. *Computational Statistics & Data Analysis*, 89:72–84.

Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. (2019). Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 2722–2730.

Horvat, C. and Pfister, J.-P. (2021). Denoising normalizing flow. In *Advances in Neural Information Processing Systems*, volume 34, pages 9099–9111.

Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. (2022). Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5149–5169.

Hsu, H.-K., Yao, C.-H., Tsai, Y.-H., Hung, W.-C., Tseng, H.-Y., Singh, M., and Yang, M.-H. (2020). Progressive domain adaptation for object detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*.

Huang, H.-H. and Yeh, M.-Y. (2021). Accelerating continuous normalizing flow with trajectory polynomial regularization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(9):7832–7839.

Huang, P., Xu, M., Zhu, J., Shi, L., Fang, F., and Zhao, D. (2022). Curriculum reinforcement learning using optimal transport via gradual domain adaptation. In *Advances in Neural Information Processing Systems*, volume 35, pages 10656–10670.

Huang, Z., Chen, S., Zhang, J., and Shan, H. (2021). AgeFlow: Conditional age progression and regression with normalizing flows. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 743–750.

Ishibashi, H. and Hino, H. (2020). Stopping criterion for active learning based on deterministic generalization bounds. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108, pages 386–397.

Izmailov, P., Kirichenko, P., Finzi, M., and Wilson, A. G. (2020). Semi-supervised learning with normalizing flows. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4615–4630.

Kimura, M., Nakamura, T., and Saito, Y. (2021). SHIFT15M: multiobjective large-scale fashion dataset with distributional shifts. *CoRR*, abs/2108.12992.

Kingma, D. P. and Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, volume 31.

Kirichenko, P., Izmailov, P., and Wilson, A. G. (2020). Why normalizing flows fail to detect out-of-distribution data. In *Advances in Neural Information Processing Systems*, volume 33, pages 20578–20589.

Koh, P. W., Sagawa, S., Marklund, H., Xie, S. M., Zhang, M., Balsubramani, A., Hu, W., Yasunaga, M., Phillips, R. L., Gao, I., Lee, T., David, E., Stavness, I., Guo, W., Earnshaw, B., Haque, I., Beery, S. M., Leskovec, J., Kundaje, A., Pierson, E., Levine, S., Finn, C., and Liang, P. (2021). WILDS: A benchmark of in-the-wild distribution shifts. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pages 5637–5664.

Kohn, W. and Sham, L. J. (1965). Self-consistent equations including exchange and correlation effects. *Phys. Rev.*, 140:A1133–A1138.

Kong, Z. and Chaudhuri, K. (2020). The expressive power of a class of normalizing flow models. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108, pages 3599–3609.

Kozachenko, L. and Leonenko, N. (1987). Sample estimate of the entropy of a random vector. *Problems of information transmission*, 23(2):95–101.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86.

Kumar, A., Ma, T., and Liang, P. (2020). Understanding self-training for gradual domain adaptation. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 5468–5479.

Kuznetsov, M. and Polykovskiy, D. (2021). MolGrow: A graph normalizing flow for hierarchical molecular generation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(9):8226–8234.

Li, H., Pan, S. J., Wang, S., and Kot, A. C. (2018). Domain generalization with adversarial feature learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5400–5409.

Li, S., Wang, Z., Kirby, R., and Zhe, S. (2022). Deep multi-fidelity active learning of high-dimensional outputs. In *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151, pages 1694–1711.

Li, S., Xing, W., Kirby, R., and Zhe, S. (2020). Multi-fidelity Bayesian optimization via deep neural networks. In *Advances in Neural Information Processing Systems*, volume 33, pages 8521–8531.

Lin, J. (1991). Divergence measures based on the Shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151.

Liu, H., Long, M., Wang, J., and Wang, Y. (2020). Learning to adapt to evolving domains. In *Advances in Neural Information Processing Systems*, volume 33, pages 22338–22348.

Liu, H., Wang, J., and Long, M. (2021). Cycle self-training for domain adaptation. In *Advances in Neural Information Processing Systems*, volume 34, pages 22968–22981.

Lu, Y. and Huang, B. (2020). Structured output learning with conditional generative flows. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):5005–5012.

M. N. Goria, N. N. Leonenko, V. V. M. and Inverardi, P. L. N. (2005). A new class of random vector entropy estimators and its applications in testing statistical hypotheses. *Journal of Nonparametric Statistics*, 17(3):277–297.

Mahajan, S., Gurevych, I., and Roth, S. (2020). Latent normalizing flows for many-to-many cross-domain mappings. In *International Conference on Learning Representations*.

Mathieu, E. and Nickel, M. (2020). Riemannian continuous normalizing flows. In *Advances in Neural Information Processing Systems*, volume 33, pages 2503–2515.

Matthews, B. (1975). Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure*, 405(2):442–451.

McInnes, L., Healy, J., Saul, N., and Grossberger, L. (2018). UMAP: Uniform manifold approximation and projection. *The Journal of Open Source Software*, 3(29):861.

Nguyen, A. T., Tran, T., Gal, Y., Torr, P., and Baydin, A. G. (2022). KL guided domain adaptation. In *International Conference on Learning Representations*.

Onken, D., Wu Fung, S., Li, X., and Ruthotto, L. (2021). OT-Flow: Fast and accurate continuous normalizing flows via optimal transport. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(10):9223–9232.

Pan, Z., Yu, W., Yi, X., Khan, A., Yuan, F., and Zheng, Y. (2019). Recent progress on generative adversarial networks (GANs): A survey. *IEEE Access*, 7:36322–36333.

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. (2021). Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32.

Peherstorfer, B., Willcox, K., and Gunzburger, M. (2016). Optimal model management for multifidelity Monte Carlo estimation. *SIAM Journal on Scientific Computing*, 38(5):A3163–A3194.

Peherstorfer, B., Willcox, K., and Gunzburger, M. (2018). Survey of multifidelity methods in uncertainty propagation, inference, and optimization. *SIAM Review*, 60(3):550–591.

Perdew, J. P., Burke, K., and Ernzerhof, M. (1996). Generalized gradient approximation made simple. *Phys. Rev. Lett.*, 77:3865–3868.

Pumarola, A., Popov, S., Moreno-Noguer, F., and Ferrari, V. (2020). C-Flow: Conditional generative flow models for images and 3D point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7949–7958.

Rai, P., Saha, A., Daumé, H., and Venkatasubramanian, S. (2010). Domain adaptation meets active learning. In *Proceedings of the NAACL HLT 2010 Workshop on Active Learning for Natural Language Processing*, pages 27–32.

Ramirez-Loaiza, M. E., Sharma, M., Kumar, G., and Bilgic, M. (2017). Active learning: An empirical study of common baselines. *Data Mining and Knowledge Discovery*, 31(2):287–313.

Redko, I., Morvant, E., Habrard, A., Sebban, M., and Bennani, Y. (2019). *Advances in domain adaptation theory*. Elsevier.

Rezende, D. and Mohamed, S. (2015). Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 1530–1538.

Rodriguez-Lujan, I., Fonollosa, J., Vergara, A., Homer, M., and Huerta, R. (2014). On the calibration of sensor arrays for pattern recognition using the minimal number of experiments. *Chemometrics and Intelligent Laboratory Systems*, 130:123–134.

Ross, B. and Cresswell, J. (2021). Tractable density estimation on learned manifolds with conformal embedding flows. In *Advances in Neural Information Processing Systems*, volume 34, pages 26635–26648.

Rozen, N., Grover, A., Nickel, M., and Lipman, Y. (2021). Moser Flow: Divergence-based generative modeling on manifolds. In *Advances in Neural Information Processing Systems*, volume 34, pages 17669–17680.

Sagawa, S. and Hino, H. (2022). Gradual domain adaptation via normalizing flows. *CoRR*, abs/2206.11492.

Sagawa, S. and Hino, H. (2023). Cost-effective framework for gradual domain adaptation with multifidelity. *Neural Networks*, 164:731–741.

Settles, B. (2009). Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison.

Shen, J., Qu, Y., Zhang, W., and Yu, Y. (2018). Wasserstein distance guided representation learning for domain adaptation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).

Shimodaira, H. (2000). Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*.

Smola, A., Gretton, A., Song, L., and Schölkopf, B. (2007). A Hilbert space embedding for distributions. In *Algorithmic Learning Theory*, pages 13–31.

Sohn, K., Berthelot, D., Carlini, N., Zhang, Z., Zhang, H., Raffel, C. A., Cubuk, E. D., Kurakin, A., and Li, C.-L. (2020). FixMatch: Simplifying semi-supervised learning with consistency and confidence. In *Advances in Neural Information Processing Systems*, volume 33, pages 596–608.

Sugiyama, M., Suzuki, T., and Kanamori, T. (2012). *Density ratio estimation in machine learning*. Cambridge University Press.

Sun, B. and Saenko, K. (2016). Deep CORAL: Correlation alignment for deep domain adaptation. In *Computer Vision – ECCV 2016 Workshops*, pages 443–450.

Sun, J., Ruzsinszky, A., and Perdew, J. P. (2015). Strongly constrained and appropriately normed semilocal density functional. *Phys. Rev. Lett.*, 115:036402.

Takeno, S., Fukuoka, H., Tsukada, Y., Koyama, T., Shiga, M., Takeuchi, I., and Karasuyama, M. (2020). Multi-fidelity Bayesian optimization with max-value entropy search and its parallelization. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 9334–9345.

Taylor, J., Earnshaw, B., Mabey, B., Victors, M., and Yosinski, J. (2019). RxRx1: An image set for cellular morphological variation across many experimental batches. In *International Conference on Learning Representations*.

Teshima, T., Ishikawa, I., Tojo, K., Oono, K., Ikeda, M., and Sugiyama, M. (2020). Coupling-based invertible neural networks are universal diffeomorphism approximators. In *Advances in Neural Information Processing Systems*, volume 33, pages 3362–3373.

Thomas, R. S., Paules, R. S., Simeonov, A., Fitzpatrick, S. C., Crofton, K. M., Casey, W. M., and Mendrick, D. L. (2018). The US Federal Tox21 Program: A strategic and operational plan for continued leadership. *ALTEX - Alternatives to animal experimentation*, 35(2):163–168.

Vergara, A., Vembu, S., Ayhan, T., Ryan, M. A., Homer, M. L., and Huerta, R. (2012). Chemical gas sensor drift compensation using classifier ensembles. *Sensors and Actuators B: Chemical*, 166-167:320–329.

Villani, C. (2009). *Optimal transport: old and new*. Springer.

Wang, H., He, H., and Katabi, D. (2020). Continuously indexed domain adaptation. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 9898–9907.

Wang, H., Li, B., and Zhao, H. (2022). Understanding gradual domain adaptation: Improved analysis, optimal path and beyond. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 22784–22801.

Wang, L., Zhang, X., Su, H., and Zhu, J. (2024). A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–20.

Wang, M. and Deng, W. (2018). Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153.

Wilson, G. and Cook, D. J. (2020). A survey of unsupervised deep domain adaptation. *ACM Trans. Intell. Syst. Technol.*, 11(5).

Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., Leswing, K., and Pande, V. (2018). MoleculeNet: A benchmark for molecular machine learning. *Chem. Sci.*, 9:513–530.

Yang, G., Huang, X., Hao, Z., Liu, M.-Y., Belongie, S., and Hariharan, B. (2019). PointFlow: 3D point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4541–4550.

Ye, M., Jiang, R., Wang, H., Choudhary, D., Du, X., Bhushanam, B., Mokhtari, A., Kejariwal, A., and Liu, Q. (2022). Future gradient descent for adapting the temporal shifting data distribution in online recommendation systems. In *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*, volume 180, pages 2256–2266.

Zhai, J., Zhang, S., Chen, J., and He, Q. (2018). Autoencoder and its various variants. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 415–419.

Zhang, Y., Deng, B., Jia, K., and Zhang, L. (2021). Gradual domain adaptation via self-training of auxiliary models. *CoRR*, abs/2106.09890.

Zhao, S., Li, B., Reed, C., Xu, P., and Keutzer, K. (2020). Multi-source domain adaptation in the deep learning era: A systematic survey. *CoRR*, abs/2002.12169.

Zhou, S., Wang, L., Zhang, S., Wang, Z., and Zhu, W. (2022a). Active gradual domain adaptation: Dataset and approach. *IEEE Transactions on Multimedia*, 24:1210–1220.

Zhou, S., Zhao, H., Zhang, S., Wang, L., Chang, H., Wang, Z., and Zhu, W. (2022b). Online continual adaptation with active self-training. In *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151, pages 8852–8883.

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. (2021). A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76.