Gradual Domain Adaptation via Normalizing Flows

Shogo Sagawa
sagawa@ism.ac.jp
Department of Statistical Science, Graduate University for Advanced Studies,
Hayama, Kanagawa 240-0193, Japan, and Konica Minolta, Hachioji,
Tokyo 192-8505, Japan

Hideitsu Hino
hino@ism.ac.jp
Department of Advanced Data Science, Institute of Statistical Mathematics,
Tachikawa, Tokyo 190-8562, Japan, and Center for Advanced Intelligence Project,
RIKEN, Chuo-ku, Tokyo 103-0027, Japan

Standard 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. It involves leveraging the intermediate domain, which gradually shifts from the source domain to the target domain. In previous work, it is assumed that the number of intermediate domains is large and the distance between adjacent domains is small; hence, the gradual domain adaptation algorithm, involving self-training with unlabeled data sets, is applicable. In practice, however, gradual self-training will fail because the number of intermediate domains is limited and the distance between adjacent domains is large. We propose the use of normalizing flows to deal with this problem while maintaining the framework of unsupervised domain adaptation. The proposed method learns a transformation from the distribution of the target domains to the gaussian mixture distribution via the source domain. We evaluate our proposed method by experiments using real-world data sets and confirm that it mitigates the problem we have explained and improves the classification performance.

1 Introduction

In a standard problem of learning predictive models, it is assumed that the probability distributions of the test data and the training data are the same. The prediction performance generally deteriorates when this assumption does not hold. The simplest solution is to discard the training data and collect new samples from the distribution of test data. However, this solution is inefficient and sometimes impossible, and there is a strong demand for utilizing valuable labeled data in the source domain.
Domain adaptation (Ben-David et al., 2006) is one of the transfer learning frameworks in which the probability distributions of prediction target and training data are different. In domain adaptation, the source domain has many labeled samples, while the target domain has few or no labeled samples. The case with no labels from the target domain is called unsupervised domain adaptation and has been the subject of much research, including theoretical analysis and real-world application (Ben-David et al., 2006; Mansour et al., 2009; Cortes et al., 2010; Redko et al., 2019; Zhao et al., 2019).

In domain adaptation, the predictive performance on the target data deteriorates when the discrepancy between the source and target domains is large. Kumar et al. (2020) proposed gradual domain adaptation (GDA), in which it is assumed that the shift from the source domain to the target domain occurs gradually and that unlabeled data sets from intermediate domains are available. The key assumption of GDA (Kumar et al., 2020) is that there are many indexed intermediate domains. The intermediate domains are arranged to connect the source domain to the target domain, and their order is known or index is given, starting from the one closest to the source domain to the one closest to the target domain. These intermediate domains connect the source and target domains densely so that gradual self-training is possible without the need for labeled data. In practice, however, the number of intermediate domains is limited. Therefore, the gaps between adjacent domains are large, and gradual self-training does not work well.

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 data sets. The generation of pseudo-intermediate domains is realized by mixing the samples from the source and target domains (Zhang et al., 2021). Optimal transport (OT; He et al., 2023) is also a promising method for realizing interpolation between domains. He et al. (2023) proposed an interpolation method that calculates the optimal transport plan between the marginal distributions of inputs in adjacent domains and transforms the distribution in one domain into that of another domain. Figure 1 shows the pseudo-intermediate domains generated by the interpolation methods proposed by Zhang et al. (2021) and He et al. (2023), respectively. To confirm the effectiveness of pseudo-intermediate domains, we conduct preliminary experiments. Using the toy data set shown in Figure 1, we compare the results of gradual self-training with and without pseudo-intermediate domains. Note that only the source domain has labeled samples (see Figure 2), and this label information is used solely for gradual self-training, not for interpolation between domains. The experimental settings, such as the composition of the neural network, are the same as those shown in section 5.8. In Table 1, we see that the pseudo-intermediate domains do not contribute to improving the predictive performance of the self-trained model.

In this letter, we seek to capture the continuous change between domains. We focus on the normalizing flow (NF; Papamakarios et al., 2021),
Figure 1: Examples of interpolations between domains using pseudo-intermediate domains.

Figure 2: Overview of the proposed method. Owing to the limited number of available intermediate domains, the applicability of gradual self-training is limited. Gradual domain adaptation is possible without gradual self-training by using continuous normalizing flow.

Table 1: Results of the Preliminary Experiments.

| Without pseudo-intermediate domains (Kumar et al., 2020) | Optimal transport (He et al., 2023) | Mixed samples (Zhang et al., 2021) |
|----------------------------------------------------------|-----------------------------------|-----------------------------------|
| Accuracy                                                 | 0.465 ± 0.026                     | 0.470 ± 0.010                     | 0.441 ± 0.149                     |
which learns the transformation between two distributions. We assume 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 com-
pared 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 follow 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 nonparametric distrib-
ution in one domain into a nonparametric distribution in another domain.
To the best of our knowledge, there has been no study on NF that learns the
transformation between distributions by estimating the likelihood of sam-
ple in a nonparametric manner. Here, we describe the distinctions between
our proposed method and the OT-based method (He et al., 2023). First, our
proposed method employs a single model to learn transformations across
multiple distributions, whereas the OT-based method calculates the opti-
mal transport plans separately between each pair of distributions. Second,
our method can generate new samples from both the observed domains
and pseudo-intermediate domains. In contrast, the optimal transport-based
method cannot generate, in principle, new samples; it only transforms the
observed samples.

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 data sets. 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 ei-
ther interpolation between domains or self-training. Izmailov et al. (2020)
proposed to use an NF for semisupervised 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 2 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 work 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 nonparametric distributions, our flow-based model can do so by using a nonparametric likelihood estimator.
• We demonstrate the effectiveness of the proposed method by experiments using both toy and real-world data sets.

The rest of the letter is organized as follows. We review related work in section 2. Then we explain in detail the gradual domain adaptation algorithm, an important previous work, in section 3. We introduce our proposed method in section 4. In section 5, we present experimental results. The last section is devoted to the conclusion of our study.

2 Related Work

We tackle the gradual domain adaptation problem by using the normalizing flow. These topics have been actively researched in recent years, making it challenging to provide a comprehensive review. Here, we introduce a few closely related studies.

2.1 Gradual Domain Adaptation. In conventional domain adaptation, a model learns the direct transformation between (samples from) the source and target domains. Several methods have been proposed to transfer the source domain to the target domain sequentially (Gadermayr et al., 2018; Gong et al., 2019; Hsu et al., 2020; Choi et al., 2020; Cui et al., 2020; Dai et al., 2021). A sequential domain adaptation is realized by using data generated by mixing the data from the source and target domains.

Kumar et al. (2020) proposed gradual domain adaptation (GDA), and they showed that it is possible to adapt the method to a large domain gap by self-training with unlabeled data sets. It is assumed that the intermediate domains gradually shift from the source domain to the target domain, and the sequence of intermediate domains is given. Chen and Chao (2021) developed the method in which the intermediate domains are available whereas their indices are unknown. Zhang et al. (2021) and Abnar et al. (2021) proposed to apply the idea of GDA to conventional domain adaptation. Since in these methods, they assumed that intermediate domains are unavailable, they use pseudo-intermediate domains. Zhou, Wang et al. (2022) proposed a gradual semisupervised domain adaptation method that uses self-training and requests of queries. They also provided a new data set suitable for GDA. Kumar et al. (2020) conducted a theoretical analysis and provided a generalization error bound for gradual self-training. Wang et al. (2022) conducted a theoretical analysis under more general assumptions and derived an improved generalization error bound. Dong et al. (2022) also conducted a theoretical analysis under the condition that all the labels of the intermediate domains are given. He et al. (2023) assumed a scenario where, similar to ours, the available intermediate domains are limited. They
propose generating a pseudo-intermediate domain using OT and use self-training to propagate the label information of the source domain.

There are several problem settings similar to those in GDA. Liu et al. (2020) proposed evolving domain adaptation, demonstrating the feasibility of adapting a target domain that evolves over time through meta-learning. 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. Wang et al. (2020) assumed 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. Huang et al. (2022) proposed the application of the idea of GDA to reinforcement learning. They propose 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.

While conventional machine learning methods are predicated on capturing a static distribution, continual learning (Wang et al., 2024) attempts to adapt to sequential tasks with dynamically changing distributions. When adapting from one task to the next, performance on the previous task deteriorates. This phenomenon, known as catastrophic forgetting, is a major challenge in continuous learning. In multisource domain adaptation (Zhao et al., 2020), labeled data are collected from multiple different distributions. The standard approach in multisource domain adaptation is to approximate the target distribution using a mixture of multiple source distributions. Therefore, a combination of classifiers with weight is used for multisource domain adaptation. Ye et al. (2022) proposed a method for temporal domain generalization in online recommendation models. Zhou, Zhao et al. (2022) proposed an online learning method that uses self-training and requests of queries. Ye et al. (2022) and Zhou, Zhao et al. (2022) assumed that the labels of the intermediate domains are given, while Sagawa and Hino (2023) applied the multifidelity active learning assuming access to the labels of the intermediate and target domains at certain costs.

2.2 Normalizing Flows. Normalizing flows (NFs) are reversible generative models that use invertible neural networks to transform samples from a known distribution, such as the gaussian distribution. NFs 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. NFs with constrained architectures are called discrete NFs (DNFs), and examples include RealNVP (Dinh et al., 2017), Glow (Kingma & 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 &
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. CNFs describe the transformation between samples from the gaussian to the observed samples from a complicated distribution using ordinary differential equations. Grathwohl et al. (2019) proposed a variant of CNF called FFJORD, which exhibited improved performance over DNF. FFJORD was followed by studies to improve the computational efficiency (Huang & Yeh, 2021; Onken et al., 2021) and the representation on manifolds (Mathieu & Nickel, 2020; Rozen et al., 2021; Ben-Hamu et al., 2022). Several normalizing flow models aiming to learn a low-dimensional manifold on which data are distributed and to estimate the density on that manifold have been proposed (Brehmer & Cranmer, 2020; Caterini et al., 2021; Horvat & Pfister, 2021; Ross & Cresswell, 2021).

Normalizing flows have been applied to several specific tasks, for example, data generation (images: Lu & Huang, 2020, 3D point clouds: Pumarola et al., 2020, and chemical graphs: Kuznetsov & Polykovskiy, 2021), anomaly detection (Kirichenko et al., 2020), and semisupervised 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).

As an approach of domain adaptation, it is natural to learn domain-invariant representations between the source and target domains, and several methods that use NFs have been developed for that purpose. Grover et al. (2020) and Das et al. (2021) proposed a domain adaptation method that combines adversarial training and NFs. These methods separately train two NFs for the source and target domains and execute domain alignment in a common latent space using adversarial discriminators. 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 space of the source domain is forced to be gaussian. 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 methods explored here 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 nonparametric distributions.

3 Formulation of Gradual Domain Adaptation

In this section, we introduce the concept and formulation of the gradual domain adaptation proposed by Kumar et al. (2020), which uses gradual
self-training, and we confirm that a gradual self-training-based method is unsuitable when the discrepancies between adjacent domains are large.

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. The source data set has labels $S = \{(x_i^{(j)}, y_i^{(j)})\}_{i=1}^{n_j}$, whereas both the intermediate and target data set are unlabeled $U^{(j)} = \{x_i^{(j)}\}_{i=1}^{n_j}$. The subscript $i$, $1 \leq i \leq n_j$ indicates the $i$th observed datum, and the superscript $(j)$, $1 \leq j \leq K$ indicates the $j$th domain. We note that $\{U^{(j)}\}_{j=1}^{K-1}$ are the intermediate data sets and $U^{(K)}$ is the target data set. The source domain corresponds to $j = 1$, and the target domain corresponds to $j = K$. 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. Let $p_j$ be the probability density function of the $j$th domain. The Wasserstein metrics (Villani, 2009) are used to measure the distance between domains. Kumar et al. (2020) defined the distance between adjacent domains as the per class $\infty$-Wasserstein distance. Wang et al. (2022) defined the distance between adjacent domains as the $p$-Wasserstein distance as a more general metric. Following Wang et al. (2022), we define the average $p$-Wasserstein distance between consecutive domains as $\rho = \frac{1}{K-1} \sum_{j=2}^{K} W_p(p_{j-1}(x, y), p_j(x, y))$, where $W_p(\cdot, \cdot)$ denotes the $p$-Wasserstein distance.

Kumar et al. (2020) proposed a GDA algorithm that consists of two steps. In the first step, the predictive model for the source domain is trained with the source data set. Then, by sequential application of the self-training, labels of the adjacent domains are predicted. Let $\mathcal{H} = \{h | h : \mathcal{X} \rightarrow \mathcal{Y}\}$ and $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_{\geq 0}$ be a hypothesis space and a loss function, respectively. We consider training the model $h^{(1)}$ by minimizing the loss on the source data set

$$h^{(1)} = \arg\min_{h \in \mathcal{H}} \frac{1}{n_1} \sum_{i=1}^{n_1} \ell(h(x_i^{(1)}), y_i^{(1)}).$$

For the joint distribution $p_j(x, y)$, the expected loss with the classifier $h^{(j)}$ is defined as $\epsilon^{(j)}(h^{(j)}) = \mathbb{E}_{x, y \sim p_j}(x, y) [\ell(h^{(j)}(x), y)]$.

The classifier of the current domain $h^{(j)}$ is used to make predictions on the unlabeled data set $U^{(j+1)} = \{x_i^{(j+1)}\}_{i=1}^{n_{j+1}}$ in the next domain. Let $ST(h^{(j)}, U^{(j+1)})$ be a function that returns the self-trained model for $x_i^{(j+1)} \in U^{(j+1)}$ by inputting the current model $h^{(j)}$ and an unlabeled data set $U^{(j+1)}$:

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

Self-training is only applied between adjacent domains. The output of GDA is the classifier for the target domain $h^{(K)}$. The classifier $h^{(K)}$ is obtained
by applying sequential self-training to the model of the source domain \( h^{(1)} \) along the sequence of unlabeled data sets \( U^{(2)}, \ldots, U^{(K)} \) denoted as \( h^{(2)} = \text{ST}(h^{(1)}, U^{(2)}), \ldots, h^{(K)} = \text{ST}(h^{(K−1)}, U^{(K)}) \).

Kumar et al. (2020) provided the first generalization error bound for the gradual self-training of \( \epsilon^{O(K−1)}\left( \epsilon^{(1)}(h^{(1)}) + \mathcal{O}\left(\sqrt{\frac{\log K−1}{n}}\right)\right) \), where the sample size in each domain is assumed to be the same: \( n_2 = \cdots = n_K = n \) without loss of generality. Wang et al. (2022) conducted a theoretical analysis under more general assumptions and derived an improved generalization bound:

\[
\epsilon^{(1)}(h^{(1)}) + \mathcal{O}\left(\rho \cdot (K − 1) + \frac{K − 1}{\sqrt{n}} + \frac{1}{\sqrt{n}(K−1)}\right). \tag{3.1}
\]

When we consider the problem where the number of accessible intermediate domains is limited, it is natural that the distance between adjacent domains \( \rho \) becomes large. When this occurs, equation 3.1 suggests that the bound for the expected loss of the target classifier will become loose. To tackle the problem of a large distance between adjacent domains, we propose a method using NFs.

4 Proposed Method

We consider a GDA problem with large discrepancies between adjacent domains. As discussed in section 3, the applicability of a gradual self-training-based method (Kumar et al., 2020; Wang et al., 2022; He et al., 2023) is limited in this situation. Our proposed GDA method mitigates the problem without gradual self-training. Our key idea is to use NFs to capture the continuous change between domains, as detailed in section 4.1. Conventional NFs learn only the transformation between the distribution that the given data follow and a parametric distribution, such as the Gaussian distribution. To realize GDA without gradual self-training, we develop a nonparametric transformation method between distributions using NFs. Section 4.2 introduces a nonparametric 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 postprocessing 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. We discuss the theoretical aspects and the scalability of the proposed method in sections 4.4 and 4.5, respectively.

4.1 Capturing Continuous Change with Normalizing Flows. An NF uses an invertible function \( f : \mathbb{R}^d \rightarrow \mathbb{R}^d \) to transform a sample \( x \in \mathbb{R}^d \) from the complicated distribution \( p(x) \) to a sample \( z \in \mathbb{R}^d \) from the standard
gaussian $p_0(z)$. The log density of $x = f(z)$ satisfies $\log p(x) = \log p_0(f^{-1}(x)) + \log |\operatorname{det} \nabla f^{-1}(x)|$, where $\nabla f^{-1}(x)$ is the Jacobian of $f^{-1}$.

Our aim is to capture the continuous change $x^{(k)} \mapsto \cdots \mapsto x^{(1)} \mapsto z$ by using NFs. We consider a continuous transformation, $x^{(k)} = f^{(k)}(x^{(k-1)}), \ldots, x^{(1)} = f^{(1)}(z)$, by using multiple NFs $f^{(k)}, \ldots, f^{(1)}$. Our preliminary experiments indicate that continuous normalizing flows (CNFs) are better suited for capturing continuous transitions between domains than discrete normalizing flows. Further details of these experiments are elaborated on in section 5.3.

To represent continuous changes between domains, we introduce a time index $t \in \mathbb{R}_+$. We regard the index of each domain $j$ as a continuous variable and consider it as a particular time point. We set $t = 0$ and $t = j$ for $z$ and $x^{(j)}$, respectively. The probability density function $p_j$ is a special case of $p_t$ when $t = j$. The CNF $g: \mathbb{R}^d \times \mathbb{R}_+ \rightarrow \mathbb{R}^d$ outputs a transformed variable dependent on time $t$, and we consider $f^{(t)}(\cdot)$ as $g(\cdot, t)$ following the standard notation of CNF. The function $g(x^{(j)}, t)$ takes as inputs both a sample $x^{(j)}$ from the $j$th domain and the time $t$ and transforms the input sample $x^{(j)}$ into the state at time $t$. When $t = j$, the function $g$ outputs the sample that is used as its argument: $g(x^{(j)}, j) = x^{(j)}$. Let $v$ be a neural network parameterized by $\omega$ that represents the change in $g$ along $t$. Following Chen et al. (2018) and Grathwohl et al. (2019), we express the ordinary differential equation (ODE) with respect to $g$ using the neural network $v$ as $\partial g/\partial t = v(g(\cdot, t), t; \omega)$. The parameter $\omega$ of the neural network $v$ implicitly defines the CNF $g$. When we specifically refer to the parameter of $g$, we denote it as $g_\omega$.

An NF 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 $\partial \log p(g)/\partial t = -\text{Tr}(\partial v/\partial g)$ (Chen et al., 2018, theorem 1). The outputs of the CNF are acquired by solving an initial value problem. In order to represent continuous changes between domains, the initial value problem is sequentially solved. To formulate the problem, we introduce the function $\tau$ that relates the time index $t$ to the input variable as follows:

$$
\tau(t) = \begin{cases} 
  x^{(j)}, & \text{if } t \geq 1, \\
  z, & \text{otherwise}.
\end{cases}
$$

We assign $t_0 = j - 1$ and $t_1 = j$, and the output of the CNF for the input $x^{(j)}$ is obtained by solving the following initial value problem:

$$
\begin{bmatrix}
\tau(t_0) - \tau(t_1) \\
\Delta_1
\end{bmatrix} = \int_{t_1}^{t_0} \begin{bmatrix}
v(g(x^{(j)}, t), t; \omega) \\
\text{Tr}(\partial v/\partial g)
\end{bmatrix} dt,
$$

$$
\begin{bmatrix}
\tau_0 \\
\Delta_0
\end{bmatrix} = \begin{bmatrix}
x^{(j)} \\
0
\end{bmatrix},
$$

(4.1)
where $\Delta_1 = \log p_t(\tau(t_1)) - \log p_{t_0}(\tau(t_0))$ and $\Delta_0 = \log p_{t_0}(\tau(t_1)) - \log p_t(g(x^{(i)}, t_1))$. To calculate the log-likelihood of the CNF, we need both the transformed input $g(x, t)$ and the value of the integral of the trace term $\text{Tr}(\partial v/\partial g)$ that corresponds to the Jacobian of $g$. 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. 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 equation 4.1 and retain the solutions. In the next iteration, we decrease the values of $t_1$ and $t_0$ and use 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 $\omega$:

$$
\log p_j(g_\omega(x^{(i)}, j)) = \sum_{t=1}^{j} \log p_{t-1}(g_\omega(x^{(i)}, t-1)) - \int_{0}^{t} \text{Tr}\left(\frac{\partial v}{\partial g_\omega}\right) dt. \quad (4.2)
$$

### 4.2 Nonparametric Estimation of Log-Likelihood.

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

$$
V(\delta) = \int_{b(a, \delta)} \{p(a) + (x - a)^T \nabla p(a) + O(\delta^2)\} dx
$$

$$
= |b(a, \delta)|(p(a) + O(\delta^2)) \simeq p(a)\delta^d \pi^{d/2} / \Gamma(1 + d/2),
$$

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

$$
\log \hat{p}(a) = \log \frac{k}{n} + \log \Gamma\left(1 + \frac{d}{2}\right) - \frac{d}{2} \log \pi - d \log \delta_k. \quad (4.3)
$$

The transformation of a sample from the $j$th domain to a sample from the adjacent domain using a CNF requires the computation of
log-likelihood $\log p_{t-1}(g_\omega(x^{(t)}, t-1))$, where $t = j$. We use the $k$NN estimators for the log-likelihood. We compute the Euclidean distances among all samples in $\{x_i^{(t-1)}\}_{i=1}^{n_{t-1}}$ and $g_\omega(x^{(t)}, t-1)$, with $g_\omega(x^{(t)}, t-1)$ kept fixed. Let $\delta_{k,t-1}(g_\omega(x^{(t)}, t-1))$ be the Euclidean distance between $g_\omega(x^{(t)}, t-1)$ and its $k$th nearest neighbor in $\{x_i^{(t-1)}\}_{i=1}^{n_{t-1}}$. We obtain the $k$NN estimator of the log-likelihood of the sample $g_\omega(x^{(t)}, t-1)$ as follows:

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

(4.4)

When training our flow-based model, we estimate the log-likelihood as equation 4.4 for all samples in the $t$th domain $\{x_i^{(t)}\}_{i=1}^{n_t}$ and maximize its sample average,

$$
- \frac{d}{n_t} \sum_{i=1}^{n_t} \log \delta_{k,t-1}(g_\omega(x_i^{(t)}, t-1)),
$$

with respect to $\omega$.

For simplicity, we consider the case $n_t = n_{t-1} = n$. The cost of computing the log-likelihood by the $k$NN estimators is $O(n^2)$. During the training of CNF $g_\omega$, the computation of log-likelihood by the $k$NN estimators is required each time CNF $g_\omega$ is updated. We use the nearest-neighbor descent algorithm (Dong et al., 2011) to reduce the computational cost of the $k$NN estimators. The algorithm can be used to construct $k$NN graphs efficiently, and the computational cost is empirically evaluated to be $O(n^{1.14})$. While we use the $k$NN estimators based on the distance between an inspection point and its $k$th nearest neighbor for simplicity, an estimation method for the log-likelihood that leverages the distances to the top-$k$ nearest neighbors of an inspection point, such as differentiable top-$k$ classification (Blondel et al., 2020), is also worth considering.

Another way to estimate the log-likelihood $\log p_{t-1}(g(x^{(t)}, t-1))$ is to approximate $p_{t-1}$ using a surrogate function. Our preliminary experiments show that the $k$NN estimators are suitable for capturing continuous changes between domains, and the details of the experiments are described in section 5.4.

4.3 Gaussian Mixture Model. In sections 4.1 and 4.2, we discussed a method that captures continuous changes between domains using NFs. Normalizing flows trained with a log-likelihood based on $k$NN 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 data sets, this two-step approach is complicated. We develop a method that predicts the labels of target data without any postprocessing 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 previous work (Izmailov et al., 2020) that proposed a semi-supervised learning method using DNFs and a Gaussian mixture distribution. We now 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 $\mu_s$ and the covariance matrix $\Sigma_s$, $p_0(z|y=s) = \mathcal{N}(z|\mu_s, \Sigma_s)$.
- The classes $\{1, 2, \ldots, C\}$ are balanced, that is, $\forall s \in \{1, 2, \ldots, C\}$, $p(y=s) = 1/C$, and the Gaussian mixture distribution is $p_0(z) = \frac{1}{C} \sum_{s=1}^{C} \mathcal{N}(z|\mu_s, \Sigma_s)$.

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 $\mu_s$ is assigned to each Gaussian distribution. We propose to assign the mean vector $\mu_s = (\mu_{s_1}, \ldots, \mu_{s_d})^T$ using the polar coordinates system. Each component of the mean vector is given by

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

(4.5)

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 3 shows an intuitive description of $r$. If $r$ is too small, it is difficult to identify the class labels of samples. As shown in equation 4.5 and Figure 3d, the mean vectors are set at equal intervals.

Since the source domain has labeled data, by using equation 4.2, we can obtain the class conditional log-likelihood of a labeled sample as

$$
\log p_1(g_\omega(x^{(1)}, 1)|y=s) = \log \mathcal{N}(g_\omega(x^{(1)}, 0)|\mu_s, \Sigma_s) - \int_0^1 \text{Tr} \left( \frac{\partial v}{\partial g_\omega} \right) dt. 
$$

(4.6)
Figure 3: Schematic of hyperparameter $r$. (Left) Two Moon. (Right) Block.
The intermediate domains and the target domain have no labeled data. The log-likelihood of an unlabeled sample is given by

\[
\log p_j(g_\omega(x^{(i)}, j)) = \log \left\{ \frac{1}{C} \sum_{s=1}^C \mathcal{N}(g_\omega(x^{(i)}, 0) | \mu_s, \Sigma_s) \right\}
\]

\[- \sum_{t=2}^j d \log \delta_{k,t-1}(g_\omega(x^{(i)}, t - 1)) - \int_0^j \text{Tr} \left( \frac{\partial v}{\partial g_\omega} \right) dt. \quad (4.7)
\]

To capture the continuous change between domains, we maximize the log-likelihood of our flow-based model \(g_\omega\) on all the data from the initially given domains. The algorithm minimizes the following objective function with respect to the flow-based model \(g_\omega\):

\[
L(\omega; S, \{U^{(j)}\}_{j=2}^K) = -\frac{1}{n_1} \sum_{i=1}^{n_1} \log p_1(g_\omega(x^{(1)}_i, 1) | y_i)
\]

\[- \sum_{j=2}^K \frac{1}{n_j} \sum_{i=1}^{n_j} \log p_j(g_\omega(x^{(i)}_j, j)).
\]

We show a pseudocode for our proposed method in algorithm 1.

Our method has two hyperparameters, \(k\) and \(r\): \(k\) affects the computation of log-likelihood by \(k\)NN estimators, whereas \(r\) controls the distance between the Gaussian corresponding to each class. We discuss how to tune these hyperparameters in section 5.5.

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 \(x\) being \(s\) is

\[
p(y = s | x) = \frac{p(x | y = s)p(y = s)}{\sum_{s'}^{C} p(x | y = s')p(y = s')} = \frac{\mathcal{N}(g_\omega(x, 0) | \mu_s, \Sigma_s)}{\sum_{s'=1}^C \mathcal{N}(g_\omega(x, 0) | \mu_{s'}, \Sigma_{s'})}. \quad (4.8)
\]

Therefore, the class label of a new sample \(x\) is predicted by

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

4.4 Theoretical Aspects of a Flow-Based Model. Our proposed method maximizes the log-likelihood of labeled data with respect to \(\omega\) given by equation 4.6. We use NFs to model the distribution of inputs \(p(x | y)\), and via equation 4.8, our proposed method also implicitly models the distribution of outputs \(p(y | x)\). We denote the expected loss on the source domain as follows:
Algorithm 1: Gradual Domain Adaptation with CNF.

Input: labeled data set \( S \) and unlabeled data sets \( U^{(2)}, \ldots, U^{(K)} \)

Output: trained CNF \( g_\omega \)

1: \( j \leftarrow K \) \hspace{1cm} \( \triangleright \) start training from the target domain

2: while \( j > 0 \) do

3: \( t_0 \leftarrow j - 1, \ t_1 \leftarrow j \).

4: initial values are set to \( x^{(j)} \) and 0.

5: while \( t_0 > 0 \) do \hspace{1cm} \( \triangleright \) the initial value problem is solved sequentially

6: solve the initial value problem equation 4.1

7: retain the solutions. \( \triangleright \) these values will be used as initial values in the next iteration

8: \( t_0 \leftarrow t_0 - 1, \ t_1 \leftarrow t_1 - 1. \)

9: end while

10: if \( j = 1 \) then \hspace{1cm} \( \triangleright \) update CNF \( g_\omega \)

11: maximize the log-likelihood of labeled data as equation 4.6 with respect to \( \omega \).

12: else

13: maximize the log-likelihood of unlabeled data as equation 4.7 with respect to \( \omega \).

14: end if

15: \( j \leftarrow j - 1 \) \hspace{1cm} \( \triangleright \) training on the adjacent domain

16: end while

\[
L_1(g_\omega) = \mathbb{E}_{x,y \sim p_1(x,y)}[-\log p(y|x)]
= \mathbb{E}_{x,y \sim p_1(x,y)} \left[-\log \frac{\mathcal{N}(g_\omega(x, 0)|\mu_y, \Sigma_y)}{\sum_{s=1}^{C} \mathcal{N}(g_\omega(x, 0)|\mu_s, \Sigma_s)} \right].
\]

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

Assumption 1 (Nguyen et al., 2022). For some \( M \in \mathbb{R}_{\geq 0} \), the loss satisfies \( 0 \leq -\log p(y|x) \leq M \), where \( \forall x \in \mathcal{X}, \forall y \in \mathcal{Y} \).
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 1** (Nguyen et al., 2022). If assumption 1 holds, we have

$$L_2 \leq L_1 + \frac{M}{\sqrt{2}} \sqrt{KL[p_2(x, y)|p_1(x, y)]}$$

$$= L_1 + \frac{M}{\sqrt{2}} \sqrt{KL[p_2(x)|p_1(x)] + \mathbb{E}_{p_2(x)}[KL[p_2(y|x)|p_1(y|x)]]}$$  \hspace{1cm} (4.9)

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

All proofs are provided in appendix B. 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 1 depends on the KL divergence, which is easier to compute than the total variation. In equation 4.9, we note that it is impossible to calculate the conditional misalignment term $\mathbb{E}_{p_2(x)}[KL[p_2(y|x)|p_1(y|x)]]$ since the labels from the second domain are not given. We make the following covariate shift assumption:

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

Using proposition 1, we introduce the following corollary that gives an upper bound of the target loss $L_K$:

**Corollary 1.** If assumptions 1 and 2 hold, we have

$$L_K \leq L_1 + \frac{M}{\sqrt{2}} \sum_{t=2}^{K} \sqrt{KL[p_t(x, y)|p_{t-1}(x, y)]}$$

$$= L_1 + \frac{M}{\sqrt{2}} \sum_{t=2}^{K} \sqrt{KL[p_t(x)|p_{t-1}(x)] + \mathbb{E}_{p_t(x)}[KL[p_t(y|x)|p_{t-1}(y|x)]]}$$

$$= L_1 + \frac{M}{\sqrt{2}} \sum_{t=2}^{K} \sqrt{KL[p_t(x)|p_{t-1}(x)]}. \hspace{1cm} (4.10)$$

---

1This assumption can be slightly relaxed by setting $KL[p_{t+1}(y|x)|p_t(y|x)] \leq \varepsilon_t$ with small constants $\varepsilon_t \geq 0$, $\forall t \in \{1, \ldots, K - 1\}$. 
We consider reducing the marginal misalignment term $KL[p_t(x)\|p_{t-1}(x)]$ in equation 4.10. A CNF transforms a sample $x \sim p_{t+1}(x)$ to a sample from the probability distribution $p_t(x)$ of the adjacent domain, and the likelihood of the model satisfies

$$\log p_{t+1}(g_\omega(x, t+1)) = \log p_t(g_\omega(x, t)) - \int_t^{t+1} \text{Tr} \left( \frac{\partial v}{\partial g_\omega} \right) dt.$$ 

The expectation of the minus of the log-likelihood function to be minimized is given by

$$E_{p_{t+1}}[- \log p_{t+1}(g_\omega(x, t+1))]. \quad (4.11)$$

Onken et al. (2021) derived the following proposition:

**Proposition 2** (Onken et al., 2021). The minimization of equation 4.11 is equivalent to the minimization of the KL divergence between $p_t(x)$ and $p_{t+1}(x)$ transformed by $g_\omega$.

By proposition 2, we can reduce the marginal misalignment term $KL[p_t(x)\|p_{t-1}(x)]$ in equation 4.10 by transforming $p_t$ to $p_{t-1}$ with NFs. Compared to the upper bound for the self-training-based GDA, equation 3.1, 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.

### 4.5 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 $g$ in the ODE solver. In general, the training cost of CNF $g$ is high because the number of evaluations $N$ of $g$ 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 in 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.

### 5 Experiments

We use the implementation of the CNF provided by Huang & Yeh (2021).² PyNNDescent provides a Python implementation of the nearest-neighbor

---

² https://github.com/hanhsienhuang/CNF-TPR
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. The details of the experiment, such as the composition of the neural network, are presented in appendix A. We use WILDS (Koh et al., 2021) and MoleculeNet (Wu et al., 2018) to load preprocessed data sets. All experiments are conducted on our server with Intel Xeon Gold 6354 processors and NVIDIA A100 GPU. Source code to reproduce the experimental results is available at https://github.com/ISMHinoLab/gda_via_cnf.

Our aim is to capture the continuous change between domains. In our experiments using toy data sets, 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 sample.

5.1 Data Sets. We use benchmark data sets with modifications for GDA. Since we are considering the situation that the distances of the adjacent domains are large, we prepare only one or two intermediate domains. We summarize the information on the data sets used in our experiments in Table 2. The details of the data sets are shown in appendix A.

5.2 Experimental Settings. The manifold hypothesis (Fefferman et al., 2016) posits that real high-dimensional data lie 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 data set as preprocessing. As in semisupervised learning, UMAP can use labeled and unlabeled data simultaneously. To determine the appropriate embedding dimension, we train a CNF \( g_\omega \) on the dimension-reduced source data set by maximizing equation 4.6 with respect to \( \omega \). After the training, we evaluate the accuracy on the dimension-reduced source data set. We select the embedding dimension with which the result of mean accuracy of three-fold cross-validation in the dimension-reduced source domain is the best. All parameters in UMAP are set to their default values.

In this study, we attempt to capture the continuous change between domains using CNFs. Wang et al. (2022) mentioned that the sequence of intermediate domains should be placed along the geodesic between the

---

3 https://github.com/lmcinnes/pynndescent/tree/master
Table 2: Summary of Data Sets.

| Name               | Number of Dimensions | Number of Classes | Number of Samples | Source | Intermediate | Target |
|--------------------|----------------------|-------------------|-------------------|--------|--------------|--------|
| Two Moon           | 2                    | 2                 | 1788              | 1833   | 1818         |
| Block              | 2                    | 5                 | 1810              | 1840/1845 | 1840       |
| Rotating MNIST     | $28 \times 28$      | 10                | 2000              | 2000   | 2000         |
| (Kumar et al., 2020)|                     |                   |                   |        |              |
| Portraits          | $32 \times 32$      | 2                 | 2000              | 2000   | 2000         |
| (Ginosar et al., 2015)|                 |                   |                   |        |              |
| SHIFT15M           | 4096                 | 7                 | 5000              | 5000   | 5000         |
| (Kimura et al., 2021)|                 |                   |                   |        |              |
| RxRx1              | $3 \times 32 \times 32$ | 4             | 9856              | 9856   | 6856         |
| (Taylor et al., 2019)|                 |                   |                   |        |              |
| Tox21 NH0HCount    | 108                  | 2                 | 3284              | 1898   | 807          |
| (Thomas et al., 2018)|                 |                   |                   |        |              |
| Tox21 RingCount    | 108                  | 2                 | 1781              | 2308   | 1131         |
| (Thomas et al., 2018)|                 |                   |                   |        |              |
| Tox21 NumHDonors   | 108                  | 2                 | 3246              | 2285   | 835          |
| (Thomas et al., 2018)|                 |                   |                   |        |              |

Notes: Rotating MNIST, Portraits, and RxRx1 are image data sets. Two Moon and Block are toy data sets. Block has two intermediate domains.

Source and target domains. We argue that the sequence of intermediate domains, placed along the geodesic between the source and target domains, is preferable for capturing the continuous change between domains using CNFs. Consequently, the predictive performance of the proposed method improves. To verify that $p_j$ is appropriately positioned to connect $p_{j-1}$ and $p_{j+1}$, we introduce an evaluation metric. This metric based on the KL divergence since corollary 1 shows that the target loss $L_K$ is bounded by $\sum_{t=2}^{K} \sqrt{\text{KL}[p_t(x,y)\|p_{t-1}(x,y)]}$. However, we note that our simple evaluation metric cannot explain all the reasons why the proposed method works since real-world data sets include various factors not considered in this study (e.g., class imbalance). As shown in Figure 4, we suppose that the intermediate domain $p_j$, which lies near the midpoint of the geodesic between $p_{j-1}$ and $p_{j+1}$, is preferable. We evaluate the discrepancy between $p_j$ and the midpoint of the geodesic between $p_{j-1}$ and $p_{j+1}$. Here, we introduce the JS divergence (Lin, 1991) as

$$JS[p_{j-1}(x,y)\|p_{j+1}(x,y)] = 0.5\text{KL}[p_{j-1}(x,y)\|	ilde{p}_j(x,y)]$$

$$+ 0.5\text{KL}[p_{j+1}(x,y)\|	ilde{p}_j(x,y)],$$  \hspace{1cm} (5.1)

where $\tilde{p}_j(x,y) = 0.5p_{j-1}(x,y) + 0.5p_{j+1}(x,y)$. We call $\tilde{p}_j$ in equation 5.1 is an ideal intermediate domain connecting $p_{j-1}$ and $p_{j+1}$. Note that $\tilde{p}_j$ is obtained using $p_{j-1}$ and $p_{j+1}$. If $p_j$ is an ideal intermediate domain, we can...
use $p_j$ instead of $\tilde{p}_j$ in equation 5.1. We define the computation method that uses $p_j$ instead of $\tilde{p}_j$ as

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

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

$$
E(j) = \frac{\beta(j) - JS[p_{j-1}(x, y) | p_{j+1}(x, y)]}{\text{KL}[p_1(x, y) | p_K(x, y)]}.
$$

(5.2)

To make comparisons between different data sets, the numerator is divided by the KL divergence between the source and the target domains. 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 only a sequence of intermediate domains is given.

Using the toy data set shown in Figure 5, 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 3, 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 5: A schematic of the toy data set evaluated in Table 3.

Table 3: Experimental Results on the Toy Data Set with Various Intermediate Domains.

| 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|

Note: The column labeled “mean vector” refers to the absolute value of the first component of the mean vector in the intermediate domain.

5.3 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 $k$NN as described in section 4.2, can convert a nonparametric distribution into another nonparametric 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...
Figure 6: 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.

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 data sets on the Two-Moon data set. 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 labeled information when transforming samples from the intermediate domain to samples from the source domain. Figure 6 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. 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.

5.4 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 \( \log p_{t-1}(g_\omega(x^{(t)}, t - 1)) \), where \( t = j \). We proposed a computation method for the log-likelihood by using the \( k \)NN
estimators in section 4.2. Here, as another way of estimating the loglikelihood, we consider the approximation of \( p_{t-1} \) by a gaussian mixture distribution. Let \( Q \) and \( w^{t-1}_q \) be the number of mixture components and a mixture weight, respectively. The subscript \( q \) indicates the \( q \)th component of a gaussian mixture distribution, and the superscript indicates the domain. We approximate the adjacent domain with the gaussian mixture distribution,

\[
\hat{p}_{t-1}(x^{t-1}) = \sum_{q=1}^{Q} w^{t-1}_q \mathcal{N}(x^{t-1} | \mu^{t-1}_q, \Sigma^{t-1}_q), \quad \sum_{q=1}^{Q} w^{t-1}_q = 1,
\]

where \( \mu^{t-1}_q \) and \( \Sigma^{t-1}_q \) are the mean vector and the covariance matrix, respectively. We fit a gaussian mixture distribution for each domain. Therefore, we distinguish the mixture weight, the mean vector, and the covariance matrix with superscripts. We assign a sufficiently large value to \( Q \) since our aim is an estimation of the log-likelihood \( \log p_{t-1}(g(x^{t}), t - 1)) \).

We compare the log-likelihood estimation by fitted gaussian mixture distributions with that by \( k \)NN estimators on the Two Moon data set, and conclude that the \( k \)NN estimators are suitable for our proposed method. Since our toy data set is a simple two moon forms, \( Q = 30 \) should be enough for modeling its distribution with high precision. Figure 7 shows the comparison of the transformation by the CNF trained with \( k \)NN estimators and that trained with the gaussian mixture distributions for evaluating the likelihood. Whereas the CNF trained with \( k \)NN estimators transforms the target data to the source data as expected, the CNF trained with fitted gaussian mixture distributions fails to do so.

Figure 7 suggests that we can generate pseudo-intermediate domains using the trained CNF and apply gradual self-training to the sequence of unlabeled data sets. However, as described in section 1, this two-step method is complicated due to the need to adjust the hyperparameters of both NFs and self-training. Moreover, the training cost of gradual self-training increases in proportion to the number of domains to be adapted.

5.5 Hyperparameters. The proposed method has two hyperparameters, \( k \) and \( r \), which we introduced in sections 4.2 and 4.3, respectively. The hyperparameter \( k \) affects the result of the computation of log-likelihood by \( k \)NN estimators, and the hyperparameter \( r \) controls the distance between the gaussian distributions corresponding to each class. In this section, from a practical viewpoint, we discuss how to tune these hyperparameters.

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. Since we aim to transform
Figure 7: Comparison of the methods of estimating $\log p_{t-1}(g(x^{(t)}, t-1))$. The CNF trained with kNN estimators transforms the target data to the source data as expected.

a sample from the $j$th domain into a sample from the $(j-1)$th domain, we propose determining the appropriate value of $k$ using the samples from the $(j-1)$th domain. The log-likelihood of an inspection point in the data set $\{x_i\}_{i=1}^n$ is calculated as equation 4.3 and the sample average is denoted as $\hat{\theta} = \frac{1}{n} \sum_{i=1}^n \log \hat{p}(x_i)$. We estimate the bias and variance of $\hat{\theta}$ using the jackknife technique (Efron, 1982). The jackknife subsample consists of all but the $i$th observation, and the jackknife replicate is calculated as $\hat{\theta}_i = \frac{1}{n-1} \sum_{i'=1}^{n-1} \log \hat{p}(x_{i'})$. We then take the average of these $n$ jackknife replicates to obtain $\hat{\theta}(\cdot) = \frac{1}{n} \sum_{i=1}^n \hat{\theta}_i$. The bias and variance of $\hat{\theta}$ are given by $(n-1)(\hat{\theta}(\cdot) - \hat{\theta})$ and $\frac{n-1}{n} \sum_{i=1}^n (\hat{\theta}_i - \hat{\theta}(\cdot))^2$, respectively. Here, we consider the mean squared error (MSE) of the estimator $\hat{\theta}$ with respect to $\theta = \frac{1}{n} \sum_{i=1}^n \log p(x_i)$. It is known that the MSE can be decomposed as the sum of the variance of $\hat{\theta}$ and the squared bias of $\hat{\theta}$ (Bishop, 2006). We determine the value of the hyperparameter $k$ for each domain that minimizes the MSE calculated as

$$(n-1)(\hat{\theta}(\cdot) - \hat{\theta})^2 + \frac{n-1}{n} \sum_{i=1}^n (\hat{\theta}_i - \hat{\theta}(\cdot))^2.$$ 

Table 4 shows the results of searching for the hyperparameter $k$ that minimizes the MSE on each data set, where $k \in \{5, 10, 15, 20, 30\}$. For instance, in
Table 4: Summary of Selected Hyperparameter $k$.

| Name                                      | Source | Intermediate | Target |
|-------------------------------------------|--------|--------------|--------|
| Rotating MNIST (Kumar et al., 2020)       | 5      | 5            | 5      |
| Portraits (Ginosar et al., 2015)          | 5      | 5            | 20     |
| SHIFT15M (Kimura et al., 2021)            | 5      | 5            | 15     |
| RxRx1 (Taylor et al., 2019)               | 5      | 10           | 5      |
| Tox21 NHOHCount (Thomas et al., 2018)     | 15     | 15           | 30     |
| Tox21 RingCount (Thomas et al., 2018)     | 15     | 30           | 20     |
| Tox21 NumHDonors (Thomas et al., 2018)    | 10     | 30           | 30     |

Figure 8: Experimental results of the training of our flow-based model with various hyperparameter $k$ values. Note that we use the same $k$ across all domains to conduct the experiment within a realistic amount of time.

In Figure 8, we demonstrate that the hyperparameter $k$ affects the prediction performance of the proposed method. While we propose estimating the log-likelihood using a different $k$ for each domain, we use the same $k$ across all domains to conduct the experiment shown in Figure 8 within a realistic time. We vary the hyperparameter $k$ and train our flow-based model $g_{\omega}$. After the training, we evaluate the accuracy on the target data set. The evaluation of $g_{\omega}$ was repeated 10 times using different initial weights of neural networks.

Next, we discuss a tuning method for $r$. In equation 4.8, 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 Figures 3e and 3f, 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 Figures 3c and 3d. Let $m(r)$ be the midpoint vector of the mean vectors $\mu_s$ and $\mu'_s$ of two adjacent gaussian distributions. Since the mean vectors are set at equal intervals (see equation 4.5 and Figure 10d), any pair is acceptable when they are adjacent. We propose a method to determine $r$ by calculating $\max(N(m(r)|\mu_s, \Sigma_s), N(m(r)|\mu'_s, \Sigma'_s))$, 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 should 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 9, 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 data set. In Figure 9a, 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 9b, 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.

We vary the hyperparameter $r$ and train the proposed flow-based model $g_\omega$ with only the source data set, that is, we maximize the log-likelihood
Figure 10: Experimental results of the training of our flow-based model with various hyperparameter $r$ values on the source data set only. When the hyperparameter $r$ is sufficiently large, the accuracy on the source data set does not change significantly. The embedding dimension for all data sets is four.

given by equation 4.6 with respect to $\omega$. The performance of our flow-based model is evaluated using three-fold cross-validation on the source data set. Figure 10 shows the result of mean accuracy of three-fold cross-validation. The red dashed line in Figure 10 represents the smallest $r$, which induces $U(r) < 0.001$. We see that the accuracy on the source data set 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 should determine the hyperparameter $r$ by calculating $U(r)$.

5.6 Necessity of Intermediate Domains. Our key idea is to use a CNF to capture continuous change between domains. We show the necessity of the intermediate domains for the training of the CNF. Figure 11 shows the results of CNF trained with and without the intermediate data sets on the Block data set. In Figure 11, we show the transformation from the target data to the source data by the trained CNF. Visually, the CNF trained with the intermediate data set transforms the target data to the source data as expected. The accuracy on the target data set is 0.999 when CNF is trained with the intermediate data sets. In contrast, the accuracy on the target data set is 0.181 when CNF is trained without intermediate data sets. From these results, we conclude that it is important to train a CNF with data sets from the source, intermediate, and target domains to capture continuous change.

Wang et al. (2022) mentioned that the sequence of intermediate domains should be uniformly placed along the geodesic between the source and target domains. When only one intermediate domain is given, intuitively, a
Figure 11: Necessity of intermediate domains. The total number of domains with and without the intermediate data sets 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 data set transforms the target data to the source data as expected.

A preferable intermediate domain lies near the midpoint of the geodesic between the source and target domains. In practice, we cannot select arbitrary intermediate domains for domain adaptation, and the sequence of only intermediate domains is given.

In the Rotating MNIST data set, the source data set consists of samples from the MNIST data set without any rotation, and the target data set is prepared by rotating images of the source data set 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
Table 5: Summary of the Training of CNF with Various Intermediate Domains.

| Rotation Angle | $E(j = 2)$ | Accuracy       |
|----------------|------------|----------------|
| $\pi/21.0$     | 0.311      | 0.879 ± 0.045  |
| $\pi/10.5$     | 0.298      | 0.895 ± 0.010  |
| $\pi/7.0$      | 0.283      | 0.898 ± 0.011  |
| $\pi/6.0$      | 0.301      | 0.895 ± 0.013  |
| $\pi/5.25$     | 0.315      | 0.887 ± 0.030  |
| $\pi/4.2$      | 0.331      | 0.882 ± 0.061  |
| $\pi/3.5$      | 0.338      | 0.876 ± 0.085  |
| Without intermediate | NA | 0.828 ± 0.175 |

Figure 12: Experimental results of the training of CNF with various intermediate domains.

equation 5.2 and train our flow-based model. After the training, we evaluate the accuracy on the target data set. The evaluation of our flow-based model was repeated 10 times using different initial weights of neural networks. Table 5 and Figure 12 show the results of the experiments. We confirm that $E(j = 2)$ tends to decrease for rotation angles near $\pi/6$ and increase when the rotation angle is 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.

5.7 By-Product 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 generate pseudo-intermediate domains when predicting the label of a sample. In principle, the proposed method is applicable to 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 use the trained CNF and VAE for generating artificial intermediate images such as morphing. We show a demonstration of morphing on Rotating MNIST in Figure 13. The details of the experiment are shown in appendix A.

5.8 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 5.5, we assign different values to the hyperparameter $k$ of the proposed method for each data set. The hyperparameter $r$ is set to $r = 3$ and $r = 10$ for binary and multiclass classification, respectively. The appropriateness of these settings is discussed in section 5.5.

The primary baseline methods are self-training-based GDA methods, as introduced in section 3. Recall that these methods update the hypothesis $h: X \rightarrow Y$ trained on the source data set by applying sequential self-training. The key idea of GDA is that the hypothesis $h$ should be updated gradually. The GIFT (Abnar et al., 2021) and the AuxSelfTrain (Zhang et al., 2021) are methods that apply the idea of GDA to conventional domain adaptation. While conventional (nongradual) domain adaptation is beyond the scope of this study, we limit our comparisons to methods inspired by GDA. Although GIFT and AuxSelfTrain do not use intermediate domains, we also show experimental results using intermediate domains for updating the hypothesis 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. The details of the experiment, such as the composition of the neural network, are described in appendix A. We provide a brief description of the baseline methods as follows:

- **SourceOnly**: Train the classifier with the source data set only.
- **GradualSelfTrain** (Kumar et al., 2020): Apply gradual self-training with the initially given domains.
- **GOAT** (He et al., 2023): Interpolate the initially given domains with OT and apply gradual self-training.
- **GIFT** (Abnar et al., 2021): Update 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.
- **AuxSelfTrain** (Zhang et al., 2021): 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.
- **EAML** (Liu et al., 2020): Apply the meta-learning algorithm to the initially given domains.

First, we describe the performance of the proposed method on the real-world data sets. We train our flow-based model with and without the intermediate data sets and evaluate the accuracy on the target data set. These accuracies are denoted as $A$ and $A'$, respectively. The contribution of the intermediate domains is computed as

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

When $\frac{A - A'}{A'}$ is large, it indicates that the proposed method is suitable for the data set. Furthermore, we compute $E(j = 2)$ as equation 5.2 for each data set. Note that all real-world data sets have only one intermediate domain, $K = 3$. Each evaluation was repeated 10 times using different initial weights of neural networks. In Table 6, we see that $\frac{A - A'}{A'}$ tends to be higher when the value of $E(j = 2)$ is small. However, as mentioned in section 5.2, it is difficult to explain all the reasons why the proposed method works with a simple metric.

Next, we show the result of the comparative experiment in Figure 14. Our proposed method has comparable or superior accuracy to the
Table 6: Evaluation Results of the Proposed Method on Real-World Data Sets.

| Name                                         | With Intermediate (A) | Without Intermediate (A') | $\frac{A - A'}{A}$ | $E(j = 2)$ |
|-----------------------------------------------|------------------------|----------------------------|---------------------|------------|
| Rotating MNIST (Kumar et al., 2020)           | 0.895 ± 0.013          | 0.828 ± 0.175              | 0.082               | 0.301      |
| Portraits (Ginosar et al., 2015)              | 0.730 ± 0.016          | 0.720 ± 0.030              | 0.013               | 0.581      |
| SHIFT15M (Kimura et al., 2021)                | 0.875 ± 0.014          | 0.866 ± 0.023              | 0.011               | 0.640      |
| RxRx1 (Taylor et al., 2019)                   | 0.721 ± 0.009          | 0.694 ± 0.080              | 0.039               | 0.589      |
| Tox21 NHOHCount (Thomas et al., 2018)         | 0.666 ± 0.011          | 0.675 ± 0.012              | -0.012              | 1.897      |
| Tox21 RingCount (Thomas et al., 2018)         | 0.557 ± 0.032          | 0.545 ± 0.014              | 0.021               | 0.668      |
| Tox21 NumHDonors (Thomas et al., 2018)        | 0.635 ± 0.030          | 0.651 ± 0.023              | -0.025              | 4.359      |

Figure 14: Comparison of accuracy on five real-world data sets.

baseline methods on all data sets. 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 data set, but it does not appear to be suitable for the SHIFT15M data set. The EAML learns meta-representations from the sequence of unlabeled data
sets. In our problem setting, the number of given intermediate domains is limited, which may be insufficient for learning meta-representations.

6 Discussion and Conclusion

Gradual domain adaptation is one of the promising approaches to addressing the problem of a large domain gap by leveraging the intermediate domain. Kumar et al. (2020) assumed in their previous work that the intermediate domain gradually shifts from the source domain to the target domain and that the distance between adjacent domains is small. In this study, we consider the problem of a large distance between adjacent domains. The proposed method mitigates the problem by utilizing normalizing flows, with a theoretical guarantee on the prediction error in the target domain. We evaluate the effectiveness of our proposed method on five real-world data sets. The proposed method mitigates the limit of the applicability of GDA.

In this work, we assume that there is no noisy intermediate domain. A noisy intermediate domain deteriorates the predictive performance of the proposed method. Our future work is to develop a method to select several appropriate intermediate domains from the given noisy intermediate domains.

Appendix A: Experimental Details

A.1 Networks. We propose a gradual domain adaptation method that uses normalizing flows. The baseline methods include self-training-based methods and a method that uses meta-learning. The composition of the neural network for each method is shown as follows.

A.1.1 Our Proposed Method. One CNF block consists of two fully connected layers with 64 nodes in each layer. Our flow-based model $g$ consists of one CNF block.

A.1.2 Self-Training-Based Method. Recall that these methods update hypothesis $h : \mathcal{X} \to \mathcal{Y}$ trained on the source data set by applying sequential self-training. We follow the hypothesis that He et al. (2023) used since they consider the same problem settings as ours. The hypothesis $h$ 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 data sets as preprocessing, we modify the convolutional layer of the model to fully connected layers. We use this hypothesis in all baseline methods except for the proposed method and EAML.

A.1.3 EAML. EAML requires a feature extractor and a meta-adapter. In Liu et al. (2020), the feature extractor consists of two convolutional layers,
and the meta-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 data sets as preprocessing. Other necessary parameters during training are set as specified in Liu et al. (2020).

A.2 Data Sets. We use benchmark data sets with modifications for gradual domain adaptation. Since we are considering the situation that the distances of the adjacent domains are large, we prepare only one or two intermediate domains. We describe the details of the data sets.

A.2.1 Two Moon. This is a toy data set. We use the two-moon data set as the source domain. The intermediate and target domains are prepared by rotating the source data set by \( \pi /4 \) and \( \pi /2 \), respectively.

A.2.2 Block. This is a toy data set. 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 data set has two intermediate domains.

A.2.3 Rotating MNIST (Kumar et al., 2020). We add rotations to the MNIST data. 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.

A.2.4 Portraits (Ginosar et al., 2015). This data set includes photographs of U.S. high school students from 1905 to 2013, and the task is gender classification. We sort the data set in ascending order by year and split the data set. The source data set contains data from the 1900s to the 1930s. The intermediate data set contains data from the 1940s and the 1950s. The target data set contains data from the 1960s. We resize the original image to 32 x 32 and normalize the image intensity to the range between 0 and 1 by dividing by 255.

A.2.5 Tox21 (Thomas et al., 2018). The Tox21 data set contains the results of measuring the toxicity of compounds. The data set presents 12 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 12 evaluations. Since Tox21 has no domain indicator such as year, we introduce an indicator for splitting the entire data set into domains. It is a reasonable method from the chemical viewpoint to divide the entire data set 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.

A.2.6 SHIFT15M (Kimura et al., 2021). SHIFT15M consists of 15 million fashion images collected from real fashion e-commerce sites. We estimate seven categories of clothes from image features. SHIFT15M does not provide images but provides VGG16 (Simonyan & Zisserman, 2015) features consisting of 4096 dimensions. The data set contains fashion images from 2010 to 2020, and the passage of years causes a domain shift. 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 data sets 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 5000 samples from each domain.

A.2.7 RxRx1 (Taylor et al., 2019). RxRx1 consists of three channels of cell images obtained by a fluorescence microscope. We resize the original image to $32 \times 32$ and normalize the image intensity to the range between 0 and 1 by dividing by 255. Domain shifts occur in the execution of each batch due to slight changes in temperature, humidity, and reagent concentration. We estimate the cell type used in the experiment from image features. We consider batch numbers one, two, and three as the source domain, the intermediate domain, and the target domain, respectively.

A.3 Image Generation. Our main purpose is gradual domain adaptation, but the trained CNFs can generate pseudo-intermediate domains such as morphing as a by-product of the use of normalizing flow. It is difficult to handle high-dimensional data directly due to the high computational cost of off-the-shelf CNF implementation. Therefore, we showed the demonstration of image generation by combining the proposed method and variational autoencoder (VAE) in Figure 13. Here, we denote the details of the experiment.

We assign 60,000 samples drawn from the MNIST data set without any rotation to the source domain, and the target domain is prepared by rotating images of the source data set by angle $\pi/3$. Following Kumar et al. (2020), the intermediate domains are prepared by adding gradual rotations, angles
from $\pi/84$ to $\pi/3$. The number of intermediate domains is 27 in total. To obtain the latent variables that shift the source domain to the target domain gradually, we use all the intermediate domains for the training of VAE. After the training of VAE, we extract the latent variables whose rotation angles correspond to 0, $\pi/6$, and $\pi/3$. Following Kumar et al. (2020), we randomly select 2000 samples from each domain. We train our flow-based model with the latent variables and generate pseudo-intermediate domains by using trained CNF. Figure 13 shows images from decoded pseudo-intermediate domains.

Appendix B: Proofs

Here, we show the details of proofs. Note that propositions 1 and 2 were originally derived by Nguyen et al. (2022) and Onken et al. (2021), respectively. For completeness, we provide proofs using the notations consistent with those used in this letter.

B.1 Proof of Proposition 1 (Nguyen et al., 2022).

**Proof.** Recall that the expected losses on the source and $j$th domains are defined as $L_1 = \mathbb{E}_{x,y \sim p_{1}(x,y)}[-\log p(y|x)]$ and $L_j = \mathbb{E}_{x,y \sim p_{j}(x,y)}[-\log p(y|x)]$, respectively. In the standard domain adaptation, there is no intermediate domain and $K = 2$. We have

$$L_2 = \mathbb{E}_{p_2(x,y)}[-\log p(y|x)]$$

$$= \int -[\log p(y|x)]p_2(x, y) dx dy$$

$$= \int -[\log p(y|x)]p_1(x, y) dx dy + \int - \log p(y|x)[p_2(x, y) - p_1(x, y)] dx dy$$

$$= L_1 + \int - \log p(y|x)[p_2(x, y) - p_1(x, y)] dx dy.$$

We define sets $A$ and $B$ as

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

If assumption 1 holds, we have

$$\int - \log p(y|x)[p_2(x, y) - p_1(x, y)] dx dy,$$

$$= \int_A - \log p(y|x)[p_2(x, y) - p_1(x, y)] dx dy$$

$$+ \int_B - \log p(y|x)[p_2(x, y) - p_1(x, y)] dx dy.$$
\[
\begin{align*}
\leq & \int_{A} - \log p(y|x)[p_2(x, y) - p_1(x, y)]dxdy, \\
= & \int_{A} - \log p(y|x)[p_2(x, y) - p_1(x, y)]dxdy, \\
\leq & M \int_{A} |p_2(x, y) - p_1(x, y)|dxdy \quad (\because - \log p(y|x) \leq M),
\end{align*}
\]

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

\[
\int_{A} p_2(x, y) - p_1(x, y)dxdy + \int_{B} p_2(x, y) - p_1(x, y)dxdy = 0
\]

$\Leftrightarrow \int_{A} p_2(x, y) - p_1(x, y)dxdy = \int_{B} p_1(x, y) - p_2(x, y)dxdy$

$\Leftrightarrow \int_{A} |p_2(x, y) - p_1(x, y)|dxdy = \int_{B} |p_2(x, y) - p_1(x, y)|dxdy$

$\Leftrightarrow \int_{A} |p_2(x, y) - p_1(x, y)|dxdy = \frac{1}{2} \int |p_2(x, y) - p_1(x, y)|dxdy.$

Therefore,

\[
L_2 = L_1 + \int - \log p(y|x)[p_2(x, y) - p_1(x, y)]dxdy
\]

\[
\leq L_1 + M \int_{A} |p_2(x, y) - p_1(x, y)|dxdy
\]

\[
= L_1 + \frac{M}{2} \int |p_2(x, y) - p_1(x, y)|dxdy.
\]

Using Pinsker’s inequality, we have

\[
\left( \int |p_2(x, y) - p_1(x, y)|dxdy \right)^2 \leq 2 \int p_2(x, y) \log \frac{p_2(x, y)}{p_1(x, y)}dxdy.
\]

Therefore,

\[
L_2 \leq L_1 + \frac{M}{2} \sqrt{2 \int p_2(x, y) \log \frac{p_2(x, y)}{p_1(x, y)}dxdy}
\]

\[
= L_1 + \frac{M}{\sqrt{2}} \sqrt{KL[p_2(x, y)|p_1(x, y)].}
\]
We decompose the KL divergence between \( p_2(x, y) \) and \( p_1(x, y) \) into the marginal and conditional misalignment terms as follows:

\[
KL[p_2(x, y) \| p_1(x, y)]
\]

\[
= \mathbb{E}_{p_2(x, y)}[\log p_2(x, y) - \log p_1(x, y)]
\]

\[
= \mathbb{E}_{p_2(x, y)}[\log p_2(x) + \log p_2(y|x) - \log p_1(x) - \log p_1(y|x)]
\]

\[
= \mathbb{E}_{p_2(x, y)}[\log p_2(x) - \log p_1(x)] + \mathbb{E}_{p_2(x, y)}[\log p_2(y|x) - \log p_1(y|x)]
\]

\[
= KL[p_2(x) \| p_1(x)] + \mathbb{E}_{p_2(x)}[KL[p_2(y|x) \| p_1(y|x)]]
\]

Therefore, we have

\[
L_2 \leq L_1 + \frac{M}{\sqrt{2}} \sqrt{KL[p_2(x) \| p_1(x)] + \mathbb{E}_{p_2(x)}[KL[p_2(y|x) \| p_1(y|x)]]},
\]

which completes the proof. □

**B.2 Proof of Proposition 2 (Onken et al., 2021).**

**Proof.** Let \( p_{t+1} \) be the initial density of the samples \( x \in \mathbb{R}^d \) and \( g : \mathbb{R}^d \times \mathbb{R}_+ \rightarrow \mathbb{R}^d \) be the trajectories that transform samples from \( p_{t+1} \) to \( p_t \). The change in density when \( p_{t+1} \) is transformed from time \( t+1 \) to \( t \) is given by the change of variables formula,

\[
p_{t+1}(x) = p_{t+1}^*(g(x, t))|\det \nabla g(x, t)|,
\]

where \( p_{t+1}^* \) and \( \nabla g(x, t) \) are the transformed density and the Jacobian of \( g \), respectively. Normalizing flows aim to learn a function \( g \) that transforms \( p_{t+1} \) to \( p_t \). 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:

\[
KL[p_{t+1}^*(x) \| p_t(x)] = \int_{\mathbb{R}^d} \log \left( \frac{p_{t+1}^*(x)}{p_t(x)} \right) p_{t+1}^*(x)dx.
\]

We transform a sample \( x \) using \( g \). Using the change of variable formula, we rewrite equation B.2 as follows:

\[
KL[p_{t+1}^*(x) \| p_t(x)] = \int_{\mathbb{R}^d} \log \left( \frac{p_{t+1}^*(x)}{p_t^*(x)} \right) p_{t+1}^*(x)dx
\]
\begin{align*}
&= \int_{\mathbb{R}^d} \log \left( \frac{p_{t+1}^* (g(x, t)) |\det \nabla g(x, t)|}{p_t(g(x, t)) |\det \nabla g(x, t)|} \right) p_{t+1}^*(g(x, t)) |\det \nabla g(x, t)| dx \\
&= \int_{\mathbb{R}^d} \log \left( \frac{p_{t+1}^* (g(x, t))}{p_t(g(x, t))} \right) p_{t+1}^*(g(x, t)) |\det \nabla g(x, t)| dx \\
&= \int_{\mathbb{R}^d} \log \left( \frac{p_{t+1}^* (g(x, t))}{p_t(g(x, t))} \right) p_{t+1}^*(x) dx \quad (\because \text{equation B.1}).
\end{align*}

Using equation B.1, we have

\[ \text{KL}[p_{t+1}^*(x) | p_t(x)] = \int_{\mathbb{R}^d} \log \left( \frac{p_{t+1}^* (g(x, t))}{p_t(g(x, t))} \right) p_{t+1}^*(x) dx \]

\[ = \int_{\mathbb{R}^d} \log \left( \frac{p_{t+1}(x)}{p_t(g(x, t))} \right) p_{t+1}(x) dx \]

\[ = \mathbb{E}_{p_{t+1}(x)} \left[ \log p_{t+1}(x) - \left\{ \log p_t(g(x, t)) + \log |\det \nabla g(x, t)| \right\} \right]. \quad (\text{B.3}) \]

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

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

Moreover, they showed that the instantaneous change of the density can be computed as follows:

\[ \frac{\partial \log p(g)}{\partial t} = -\text{Tr} \left( \frac{\partial v}{\partial g} \right). \]

Namely, the term \( \log |\det \nabla g(x, t)| \) in equation B.3 is equivalent to \( -\int_t^{t+1} \text{Tr}(\partial v/\partial g) dt \). Therefore, we rewrite equation B.3 as follows:

\[ \text{KL}[p_{t+1}^*(x) | p_t(x)] = \mathbb{E}_{p_{t+1}(x)} \left[ \log p_{t+1}(x) - \left\{ \log p_t(g(x, t)) - \int_t^{t+1} \text{Tr} \left( \frac{\partial v}{\partial g} \right) dt \right\} \right] \]

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

\[ \log p_{t+1}(g(x, t + 1)) = \log p_t(g(x, t)) - \int_t^{t+1} \text{Tr} \left( \frac{\partial v}{\partial g} \right) dt. \]
Therefore, we have

$$\text{KL}[p_{t+1}(x)|p_t(x)] = \mathbb{E}_{p_{t+1}(x)}[\log p_{t+1}(x) - \log p_{t+1}(g(x, t + 1))].$$ (B.4)

The first term of equation B.4 does not depend on CNF $g$, and we can ignore it during the training of the CNF. Therefore, the minimization of equation 4.11 is equivalent to the minimization of the KL divergence between $p_t$ and $p_{t+1}$ transformed by CNF $g$. □

**B.3 Proof of Corollary 1.**

**Proof.** In gradual domain adaptation, since the ordered sequence of the intermediate domains is given, we extend proposition 1 until the target loss $L_K$ as follows:

\[
\begin{align*}
L_2 &\leq L_1 + \frac{M}{\sqrt{2}} \sqrt{\text{KL}[p_2(x, y)|p_1(x, y)]} \\
L_3 &\leq L_2 + \frac{M}{\sqrt{2}} \sqrt{\text{KL}[p_3(x, y)|p_2(x, y)]} \\
&\vdots \\
L_K &\leq L_{K-1} + \frac{M}{\sqrt{2}} \sqrt{\text{KL}[p_K(x, y)|p_{K-1}(x, y)].}
\end{align*}
\]

Summing up both sides of the above inequalities, we have

\[
L_K \leq L_1 + \frac{M}{\sqrt{2}} \sum_{t=2}^{K} \sqrt{\text{KL}[p_t(x, y)|p_{t-1}(x, y)]}
= L_1 + \frac{M}{\sqrt{2}} \sum_{t=2}^{K} \sqrt{\text{KL}[p_t(x)|p_{t-1}(x)]} + \mathbb{E}_{p_t(x)}[\text{KL}[p_t(y|x)|p_{t-1}(y|x)]]
\]

(∵ equation 4.9).

If assumption 2 holds, we have

\[
L_K \leq L_1 + \frac{M}{\sqrt{2}} \sum_{t=2}^{K} \sqrt{\text{KL}[p_t(x)|p_{t-1}(x)].}
\]

□

**Acknowledgments**

We thank the anonymous reviewers for insightful comments and suggestions. Part of this work is supported by JSPS KAKENHI, grant JP22H03653,
References

Abdal, R., Zhu, P., Mitra, N. J., & Wonka, P. (2021). StyleFlow: Attribute-conditioned exploration of StyleGAN-generated images using conditional continuous normalizing flows. *ACM Transactions on Graphics, 40*(3). 10.1145/3447648

Abnar, S., van den Berg, R., Ghiasi, G., Dehghani, M., Kalchbrenner, N., & 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., & Williams, M. (2019). TimeCluster: Dimension reduction applied to temporal data for visual analytics. *Visual Computer, 35*(6), 1013–1026. 10.1007/s00371-019-01673-y

Askari, H., Latif, Y., & Sun, H. (2023). MapFlow: Latent transition via normalizing flow for unsupervised domain adaptation. *Machine Learning, 112*(8), 2953–2974. 10.1007/s10994-023-06357-2

Becht, E., McInnes, L., Healy, J., Dutertre, C.-A., Kwok, I. W. H., Ng, L. G., . . . Newell, E. W. (2019). Dimensionality reduction for visualizing single-cell data using UMAP. *Nature Biotechnology, 37*(1), 38–44. 10.1038/s41587-018-0210-z.

Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., & Vaughan, J. W. (2010). A theory of learning from different domains. *Machine Learning, 79*(1), 151–75. 10.1007/s10994-009-5152-4

Ben-David, S., Blitzer, J., Crammer, K., & Pereira, F. (2006). Analysis of representations for domain adaptation. In B. Schölkopf, J. Platt, & T. Hofmann (Eds.), *Advances in neural information processing systems, 19*. Curran.

Ben-Hamu, H., Cohen, S., Bose, J., Amos, B., Nickel, M., Grover, A., . . . Lipman, Y. (2022). Matching normalizing flows and probability paths on manifolds. In *Proceedings of the 39th International Conference on Machine Learning*, 162 (pp. 1749–1763).

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

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Brendel, M., Teboul, O., Berthet, Q., & Djolonga, J. (2020). Fast differentiable sorting and ranking. In *Proceedings of the 37th International Conference on Machine Learning, 119* (pp. 950–959).

Brehmer, J., & Cranmer, K. (2020). Flows for simultaneous manifold learning and density estimation. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems, 33* (pp. 442–453). Curran.

Caterini, A. L., Loaiza-Ganem, G., Pleiss, G., & Cunningham, J. P. (2021). Rectangular flows for manifold learning. In M. A. Beygelzimer, Y. Dauphin, P. S. Liang, & J. Wortman Vaughan (Eds.), *Advances in neural information processing systems, 34* (pp. 30228–30241). Curran.

Chen, H.-Y., & Chao, W.-L. (2021). Gradual domain adaptation without indexed in intermediate domains. In M. A. Beygelzimer, Y. Dauphin, P. S. Liang, &
Chen, R. T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. K. (2018). Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), Advances in neural information processing systems, 31. Curran.

Choi, J., Choi, Y., Kim, J., Chang, J., Kwon, I., Gwon, Y., & Min, S. (2020). Visual domain adaptation by consensus-based transfer to intermediate domain. In Proceedings of the AAAI Conference on Artificial Intelligence, 34 (pp. 10655–10662). 10.1609/aaai.v34i07.6692

Cortes, C., Mansour, Y., & Mohri, M. (2010). Learning bounds for importance weighting. In J. D. Lafferty, C. K. I. Williams, L. Shawe-Taylor, R. S. Zemel, & A. Culotta (Eds.), Advances in neural information processing systems, 23. Curran.

Dong, J., Zhou, S., Wang, B., & Zhao, H. (2022). Algorithms and theory for supervised gradual domain adaptation. Transactions on Machine Learning Research, 11.

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

Drwal, M., Siramshetty, V., Banerjee, P., Goede, A., Preissner, R., & Dunkel, M. (2015). Molecular similarity-based predictions of the Tox21 screening outcome. Frontiers in Environmental Science, 3. 10.3389/fenvs.2015.00054

Efron, B. (1982). The jackknife, the bootstrap and other resampling plans. Society for Industrial and Applied Mathematics.

Fefferman, C., Mitter, S., & Narayanan, H. (2016). Testing the manifold hypothesis. Journal of the American Mathematical Society, 29(4), 983–1049. 10.1090/jams/852

Gadermayr, M., Eschweiler, D., Klinkhammer, B. M., Boor, P., & Merhof, D. (2018). Gradual domain adaptation for segmenting whole slide images showing pathological variability. In Lecture Notes in Computer Science and Image and Signal Processing (pp. 461–469). Springer.

Ginosar, S., Rakelly, K., Sachs, S., Yin, B., & Efros, A. A. (2015). A century of portraits: A visual historical record of American high school yearbooks. In Proceedings of the 2015 IEEE International Conference on Computer Vision Workshop (pp. 652–658).

Gong, R., Li, W., Chen, Y., & Gool, L. V. (2019). DLOW: Domain flow for adaptation and generalization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 2477–2486).
Gradual Domain Adaptation via Normalizing Flows

Goria, M., Leonenko, N. N., Mergel, V. V., & Novi Inverdi, P. L. (2005). A new class of random vector entropy estimators and its applications in testing statistical hypotheses. *Journal of Nonparametric Statistics, 17*(3), 277–297. 10.1080/104852504200026815

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

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

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

Hino, H., Koshijima, K., & Murata, N. (2015). Non-parametric entropy estimators based on simple linear regression. *Computational Statistics and Data Analysis, 89*, 72–84. 10.1016/j.csda.2015.03.011

Ho, J., Chen, X., Srinivas, A., Duan, Y., & 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*, 97 (pp. 2722–2730).

Horvat, C., & Pfister, J.-P. (2021). Denoising normalizing flow. In M. A. Beygelzimer, Y. Dauphin, P. S. Liang, & J. Wortman Vaughan (Eds.), *Advances in neural information processing systems, 34* (pp. 9099–9111). Curran.

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

Huang, H.-H., & Yeh, M.-Y. (2021). Accelerating continuous normalizing flow with trajectory polynomial regularization. In *Proceedings of the AAAI Conference on Artificial Intelligence, 35* (pp. 7832–7839). 10.1609/aaai.v35i9.16956

Huang, P., Xu, M., Zhu, J., Shi, L., Fang, F., & Zhao, D. (2022). Curriculum reinforcement learning using optimal transport via gradual domain adaptation. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh (Eds.), *Advances in neural information processing systems, 35* (pp. 10656–10670). Curran.

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

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

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

Kingma, D. P., & Dhariwal, P. (2018). Glow: Generative flow with invertible 1 x 1 convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems, 31*. Curran.

Kirichenko, P., Izmailov, P., & Wilson, A. G. (2020). Why normalizing flows fail to detect out-of-distribution data. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems, 33* (pp. 20578–20589). Curran.
Koh, P. W., Sagawa, S., Marklund, H., Xie, S. M., Zhang, M., Balsubramani, A., . . .
Liang, P. (2021). WILDS: A benchmark of in-the-wild distribution shifts. In Proceedings of the 38th International Conference on Machine Learning, 139 (pp. 5637–5664).

Kong, Z., & Chaudhuri, K. (2020). The expressive power of a class of normalizing flow models. In Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics, 108 (pp. 3599–3609).

Kozachenko, L., & Leonenko, N. (1987). Sample estimate of the entropy of a random vector. Problems of Information Transmission, 23(2), 95–101.

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

Kuznetsov, M., & Polykovskiy, D. (2021). MolGrow: A graph normalizing flow for hierarchical molecular generation. In Proceedings of the AAAI Conference on Artificial Intelligence.

Lin, J. (1991). Divergence measures based on the Shannon entropy. IEEE Transactions on Information Theory, 37(1), 145–151. 10.1109/18.61115

Liu, H., Long, M., Wang, J., & Wang, Y. (2020). Learning to adapt to evolving domains. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), Advances in neural information processing systems, 33 (pp. 22338–22348). Curran.

Lu, Y., & Huang, B. (2020). Structured output learning with conditional generative flows. In Proceedings of the AAAI Conference on Artificial Intelligence, 34(4) (pp. 5005–5012). 10.1609/aaai.v34i04.5940

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

Mansour, Y., Mohri, M., & Rostamizadeh, A. (2009). Domain adaptation: Learning bounds and algorithms. In Proceedings of the 22nd Conference on Learning Theory.

Mathieu, E., & Nickel, M. (2020). Riemannian continuous normalizing flows. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), Advances in neural information processing systems, 33 (pp. 2503–2515). Curran.

McInnes, L., Healy, J., Saul, N., & Grossberger, L. (2018). UMAP: Uniform manifold approximation and projection. Journal of Open Source Software, 3(29), 861. 10.21105/joss.00861

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

Onken, D., Wu Fung, S., Li, X., & Ruthotto, L. (2021). OT-Flow: Fast and accurate continuous normalizing flows via optimal transport. In Proceedings of the AAAI Conference on Artificial Intelligence, 35(10), 9223–9232. 10.1609/aaai.v35i10.17113

Pan, Z., Yu, W., Yi, X., Khan, A., Yuan, F., & Zheng, Y. (2019). Recent progress on generative adversarial networks (GANs): A survey. IEEE Access, 7, 36322–36333. 10.1109/ACCESS.2019.2905015

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

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, & R. Garnett (Eds.), Advances in neural information processing systems, 32. Curran.

Pumarola, A., Popov, S., Moreno-Noguer, F., & 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 (pp. 7949–7958).

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

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

Ross, B., & Cresswell, J. (2021). Tractable density estimation on learned manifolds with conformal embedding flows. In M. A. Beygelzimer, Y. Dauphin, P. S. Liang, & J. Wortman Vaughan (Eds.), Advances in neural information processing systems, 34 (pp. 26635–26648). Curran.

Rozen, N., Grover, A., Nickel, M., & Lipman, Y. (2021). Moser flow: Divergence-based generative modeling on manifolds. In M. A. Beygelzimer, Y. Dauphin, P. S. Liang, & J. Wortman Vaughan (Eds.), Advances in neural information processing systems, 34 (pp. 17669–17680). Curran.

Sagawa, S., & Hino, H. (2023). Cost-effective framework for gradual domain adaptation with multifidelity. Neural Networks, 164, 731–741. 10.1016/j.neunet.2023.03.035

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. 10.1016/S0378-3758(00)00115-4

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

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

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

Teshima, T., Ishikawa, I., Tojo, K., Oono, K., Ikeda, M., & Sugiyama, M. (2020). Coupling-based invertible neural networks are universal diffeomorphism approximators. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), Advances in neural information processing systems, 33 (pp. 3362–3373). Curran.

Thomas, R. S., Paules, R. S., Simeonov, A., Fitzpatrick, S. C., Crofton, K. M., Casey, W. M., & 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.

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

Wang, H., He, H., & Katabi, D. (2020). Continuously indexed domain adaptation. In Proceedings of the 37th International Conference on Machine Learning, 119 (pp. 9898–9907).
Wang, H., Li, B., & Zhao, H. (2022). Understanding gradual domain adaptation: Improved analysis, optimal path and beyond. In *Proceedings of the 39th International Conference on Machine Learning*, 162 (pp. 22784–22801).

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

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

Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., . . . Pande, V. (2018). MoleculeNet. *Chemical Science*, 9, 513–530. 10.1039/C7SC02664A

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

Ye, M., Jiang, R., Wang, H., Choudhary, D., Du, X., Bhushanam, B., . . . Liu, Q. (2022). Future gradient descent for adapting the temporal shifting data distribution in online recommendation systems. In *Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence*, 180 (pp. 2256–2266).

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

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

Zhao, H., Combes, R. T. D., Zhang, K., & Gordon, G. (2019). On learning invariant representations for domain adaptation. In *Proceedings of the 36th International Conference on Machine Learning*, 97 (pp. 7523–7532).

Zhao, S., Li, B., Reed, C., Xu, P., & 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., & Zhu, W. (2022). Active gradual domain adaptation: Dataset and approach. *IEEE Transactions on Multimedia*, 24, 1210–1220. 10.1109/TMM.2022.3142524

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

Received January 25, 2024; accepted October 9, 2024.