Master Thesis

Adversarial Instance Re-weighting for Unsupervised Domain Adaptation

Itzel Belderbos

Master Thesis DKE-22-05

Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science of Data Science for Decision Making
at the Department of Data Science and Knowledge Engineering
of the Maastricht University

Thesis Committee:

Drs. M. Popa
Dr. C. Seiler
External: Dr. F. Jansen

Maastricht University
Faculty of Science and Engineering
Department of Data Science and Knowledge Engineering

October 22, 2021

Contents

Abstract

In classical machine learning, we assume a model trained on the source domain generalizes well to the unlabeled target domain. However, oftentimes we encounter a distribution shift, causing the trained model to be unable to generalize well to the new task. Unsupervised domain adaptation is focused on this transfer of domain knowledge utilizing the available information in the target data. Recent advances in this field have addressed the issue in a feature-based manner, but the integration with instance re-weighting has not yet been explored. This work introduces a non-generative adversarial framework which is inspired by the Wasserstein GAN, and aims at matching the two distributions sample-wise. We first explore the potential and limits of the method on small domain shifts in several proof-of-concepts. Subsequently, we combine the concept with a feature-based network and compare its performance with the state-of-the-art. We conclude that the framework is able to match the distributions under small domain shift, as well as being advantageous for larger domain shifts in imbalanced cases.

Chapter 1

Introduction

In the ideal situation, we are able to train a machine learning classifier over a source dataset which generalizes well to other target sets. However, this generalization does not perform accurately when the train samples are not a suitable representation of the underlying distribution. In this case, there is a shift between source and target distribution, leading to poor performance on the target dataset. A possible solution could be to fine-tune these classification models on the target datasets; however, generally it is remarkably challenging and expensive to obtain sufficient labeled data to accurately fine-tune the model [1].

An example setting is when we would like to predict whether patients develop heart disease on the basis of their cholesterol and age. Figure 1.1 shows two scatter plots of a subset of a heart disease dataset [2]. The two independent variables age and cholesterol are shown on the two axes. The source data is drawn from patients in a hospital in Hungary, while the target data is taken from a hospital in the United States. Nonetheless, as the Hungarian patients in the data are much younger than the American patients, a classification model trained on the source set does not generalize well to the target data.

Obviously, the above example consists of two variables and is thus not very complex. In the field of computer vision, the data is extremely high-dimensional, as the samples consists of multi-dimensional matrices. For image data, an example could be when we aim to train a model on real-world images, while testing on synthetic images. Another example is when two datasets are drawn from two different weather conditions. In these cases, the classes to be predicted are present in both datasets, but the features originate from a different part in the feature space.

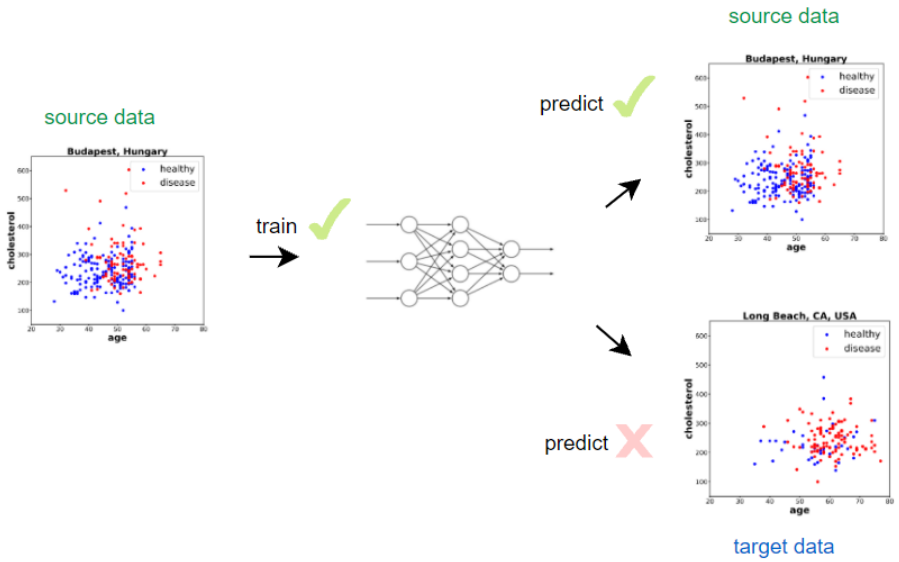

Figure 1.1: Classifier on source and target data

1.1 Problem statement

As explained above, a shift in distribution between source and target dataset can harm the prediction performance on the unlabeled target dataset. In the optimal situation, we would like to have a source dataset which is representative for all possible distributions in the feature space, such that the trained source model can be applied to every possible target dataset. However, in some situations it might not be necessary to generalize the model to all possible samples in the feature space, but solely to *one* specific distribution within the complete feature space, called a *domain* [3]. Domain adaptation refers to the transformations imposed to improve this domain transfer. In our research, we focus on *unsupervised* domain adaptation, meaning that no labels are available for the target dataset.

Let $\mathcal{D}^s$ be the source domain, which consists of the feature space $\mathcal{X}^s$, and $\mathcal{D}^t$ be the target domain with feature space $\mathcal{X}^t$ without any labeled data. The marginal distribution of the data is denoted as $P(X)^s$ and $P(X)^t$, with $X^t = \{x_1, \ldots, x_m\} \in \mathcal{X}^t$ and $X^s = \{x_1, \ldots, x_n\} \in \mathcal{X}^s$.

From the source domain $\mathcal{D}^s = \{\mathcal{X}^s, P(X)^s\}$, we aim to learn the conditional probability $P(Y^s|X^s)$ from the labeled data x_i, y_i, with $x_i \in \mathcal{X}^s$ and $y_i \in \mathcal{Y}^s$. Eventually, we would like to learn the conditional probability for the unlabeled target dataset $P(Y^t|X^t)$: the output of the classification prediction.

In classical machine learning, we assume that $\mathcal{D}^s = \mathcal{D}^t$, implying that the source

and target domain are the same. However, a *covariate shift* is a specific case of domain adaptation which implies that the posteriors and the feature space are similar in both domains, but the feature distributions are different [4]:

$$\begin{aligned} \mathcal{X}^s &= \mathcal{X}^t \\ P(Y|X)^s &= P(Y|X)^t \\ P(X)^s &\neq P(X)^t \end{aligned} \tag{1.1}$$

The difference with transfer learning is that the latter involves two different feature spaces, while in case of covariate shift we assume the distributions are over the same feature space ($\mathcal{X}^s = \mathcal{X}^t$). Transfer learning is part of heterogeneous domain adaptation, which implies that both the feature space and distributions between two datasets are different. In contrary, when the feature spaces are similar (as for covariate shift), we speak of homogeneous domain adaptation [5].

Two other forms of domain adaptation are *prior probability shift* and *concept shift*. Prior probability shift assumes there is a difference in the distribution on the classes, while the likelihood of the data remains equal, as shown in Equation 1.2. It can be noted that in case of covariate shift, we make use of causal learning, implying we derive effects from its causes, while prior shift complies with anti-causal learning, where the opposite is the case [3].

$$\begin{aligned} \mathcal{X}^s &= \mathcal{X}^t \\ P(X|Y)^s &= P(X|Y)^t \\ P(Y)^s &\neq P(Y)^t \end{aligned} \tag{1.2}$$

Concept shift means there is a change in the relation between features and class label. This implies that the distributions of the mappings from the features to the label is different for both datasets:

$$\begin{aligned} \mathcal{X}^s &= \mathcal{X}^t \\ P(Y|X)^s &\neq P(Y|X)^t \\ P(X)^s &= P(X)^t \end{aligned} \tag{1.3}$$

In this research, we will focus on the unsupervised covariate shift problem, implying that the feature distribution is different between two datasets.

1.2 Related work

According to the surveys by Kouw et al. [3] and Wang et al. [5], domain adaptation (DA) techniques can generally be subdivided into two approaches:

sample-based DA and feature-based DA. Sample-based adaptation implies re-weighting the source samples from source domain, such that the source domain resembles the target domain. In contrary, feature-based techniques aim to transform the features in such a way that a domain-invariant representation is learned.

In this adaptation procedure, there are generally two steps followed. First, the domain discrepancy is minimized, either in a feature-based or sample-based manner. Secondly, a classifier is applied to the transformed data, in order to perform the final classification on the target dataset [6]. Hence, these adaptation methods attempt to learn an adjusted version of the classification model with a low *target risk*, implying that we minimize the probability that the classifier makes a false prediction in the target domain [7].

These DA techniques can further be subdivided into two groups: divergence-based and reconstruction-based models. *Divergence-based* DA tries to minimize the discrepancy between the two distributions by means of a predefined distance metric. *Reconstruction-based* DA creates new data to generate features invariant to the distribution. Generally, both of these methods are combined with an *adversarial* component, minimizing the distribution distance by means of an opposed objective. Adversarial models makes use of a discriminator network which tries to minimize the distribution discrepancy, while a counteracting network aims to maximize this discrepancy. In our work, we will focus on divergence-based and adversarial techniques.

Reconstruction-based techniques aim to rebuild the data in such a way that there is not difference with the source data, such as Stacked Autoencoders (SAEs) or Generative Adversarial Networks (GANs). A popular work is the Coupled Generative Adversarial Network (CoGAN) [8], which generates target samples which are paired with generated source samples and share labels. Additionally, the CycleGAN [9] translates the features of one domain to another domain, without the requirement that the samples in the domains should be paired. Bousmalis et al. [10] introduced PixelDA, a GAN which is conditioned on both the source samples and a noise vector. A classifier predicts the labels of both the synthetic and source samples, whilst the discriminator makes predictions about the domain of the both the synthetic and target samples. A very recent method is DRAnet [11], which disentangles the representations of the features into multiple factors, and performs the adaptation by transferring the style features of the source domain.

In contrast with reconstruction-based techniques, divergence-based techniques do not generate new samples, but rather learn transformed features. As a covariate shift implies that the marginal distributions of the two domains are different $P(X^s) \neq P(X^t)$, feature-based methods aim to find a feature representation by means of an encoder (or extractor) τ which aligns these distributions: $P(\tau(X^s)) = P(\tau(X^t))$. These representations, also called embeddings or feature vectors, are optimized with respect to a certain divergence measure. Subsequently, the target encoder can be used for the final classification of the target

dataset, which should lead to a better performance. Hence, the goal is to to learn a mapping from the the data to these feature embeddings, such that this representation is domain-invariant [12]. However, a constraint of the these techniques is that the match of the marginal distributions does not automatically lead to a match of the posterior distributions $P(Y^s|\tau(X^s))$ and $P(Y^t|\tau(X^t))$, which is required for covariate shift [3][13].

For these divergence-based techniques, there are several divergence metrics utilized for domain adaptation. Sun et al. [14] used the difference of the mean and variance of the distributions in the Deep Correlation Alignment framework. Other frameworks [15] use the Maximum Mean Discrepancy (MMD) metric, which maps the features to a Reproducing Kernel Hilbert Space. The distances between distributions are represented as distances between the means of the two feature mappings. The Deep Domain Confusion by Tzeng et al. [16] also uses the MMD metric to learn these aligned representations.

A frequently used metric is the binary cross-entropy (BCE) loss, in which a discriminator acts as a classifier, which predict whether the representation originates from the source or target domain. This metric is used by Tzeng et al. [17] in the Domain Confusion Loss framework. However, instead of optimizing the original BCE loss, the cross entropy between a uniform distribution ($P^s(Y) = P^t(Y) = 0.5$) and the predicted domain is computed. This leads to $-\sum \frac{1}{2} \log q_s - \sum \frac{1}{2} \log q_t$, where q_d is the prediction over the domain, based on the learned feature representations. In this way, the feature representations are optimized such that the discriminator predicts a probability of 0.5. Another work that used the BCE loss in an adversarial way is the Domain Adversarial Neural Network (DANN) by Ganin et al. [7]. The framework enforces features which are beneficial for the classification task, but also present in both distributions. This is performed by optimizing the domain loss as well as the classification loss. It performs the minimax game by means of a gradient reversal layer, which uses an adjusted version of backpropagation by multiplying the gradients with a negative number. On the other hand, Tzeng et al. [1] proposed the Adversarial Discriminative Domain Adaptation (ADDA), which uses an adversarial optimization scheme which is relatively similar a Generative Adversarial Network (GAN) framework. It pre-trains the source representations with a source encoder. The parameters of this source encoder network are fixed, after which the target representations are learned in an adversarial game between the discriminator and target encoder. Hence, instead of learning to generate 'fake' images (as in a GAN), the target encoder learns to generate 'fake' feature vectors which are similar to the source representations. A more recent method is Discriminative Adversarial Domain Adaptation by Tang et al. [18], which uses an interaction between the class predictions and feature representation such that the discriminator conscious of the classification boundary.

Other recent methods make use of the Wasserstein metric, such as the Wasserstein Distance Guided Representation Learning for Domain Adaptation (WDGRL), introduced by Shen at al. [6]. The method works in a comparable way as

the DANN [7] and ADDA [1], except that it substitutes the BCE loss for the Wasserstein distance for the minimization of the domain discrepancy. The discriminator is now referred to as a critic, which aims giving values instead of a classification to its input. While methods such as ADDA and Deep Domain Confusion make use of two separate encoders (*asymmetric learning*) for the source and target representations, the WDGRL learns one feature encoder for both domains (*symmetric learning*). However, to avoid that the encoder learns embeddings which are not discriminative with respect to the classes, the classification loss on the source samples is also included in the feature optimization. Another method that uses the Wasserstein loss is the Re-weighted Adversarial Adaptation Network (RAAN) by Chen et al. [13]. The method makes use of a class weight for the source data in order to align the source and target class distributions. The idea behind this alignment is that simply learning feature representations which are domain-invariant (as mainly performed in feature-based DA), does not ensure that the posterior distributions are also aligned. Hence, $P(\tau(X^s)) = P(\tau(X^t))$ does not automatically imply that $P(Y^s|\tau(X^s)) = P(Y^t|\tau(X^t))$. They aim to estimate the distribution of the target labels $P^t(Y^t)$ by a softmax, and use this distribution to compute the weight to align the class distributions between source and target: $\beta(Y^s) = \frac{P^{re}(Y^s=Y^s)}{P^s(Y^s=Y^s)}$.

1.3 Research goal and research questions

In this work, we propose a novel framework which combines the concepts behind sample-based and feature-based unsupervised domain adaptation. We introduce the addition of a *weight network*, which is able to re-weight every instance of the source distribution such that it matches the target distribution (sample-based). The weight network opposes a *critic network*, which aims to estimate whether its input originates from the source or target dataset. Hence, the weight and critic networks are trained in an adversarial manner: the weight aims to make the two distributions more similar, while the critic tries to make them more divergent. We make use of the Wasserstein distance as a divergence metric between the domains. We also integrate the weight network with the learning of feature representations (feature-based), and endeavour to come up with our own framework.

The goal of this work is to investigate to which extent this weight network is able to benefit the domain adaptation, particularly in case of covariate shift. We will explain the methodology in more detail in the next chapters.

The research questions can be formulated as following:

1. Can we use non-generative adversarial networks, inspired by the Wasserstein GAN, to improve covariate shift adaptation?
    (a) How can we train a weight function, which converts source into target distribution?

(b) What is the effect of the critic?

2. How can we measure the performance of the reconstructed target distribution and the adversarial network?
3. Are there other ways than a weight function to express source data into target data?

1.4 Contributions

First of all, we come up with a novel concept, which is able to transform the source distribution into the target distribution under small domain shift. To our knowledge, learning a weight in this manner has not been done before. For a larger domain shift, we also propose a framework which combines feature-based and sample-based adversarial learning, which integrates the current methods ADDA [1] and WDGRL [6]. Our approach is an addition to the WDGRL [6], as their work does not involve a weight network to re-weight the source distribution and thus does not entail the sample-based element. Moreover, instead of learning two separate encoders we use the GAN-based optimization scheme, where we force the target embeddings to mimic the pre-trained source embeddings. Even though the work in [13] also combines learning a weight, their weight is not an independent network and is estimated based on the density ratio.

1.5 Research structure

This work consists of two parts: a proof-of-concept phase where we gain a more thorough understanding of the capacities and limitations of the weighted adversarial model, followed by a comparison with the state-of-the-art on benchmark datasets. The methodology for the proof-of-concept is explained in Section 2, and the results are shown in Section 3. The results from the proof-of-concept are used as input for the development of the methodology for the state-of-the-art comparison (Section 4). The results for the comparison are shown in Section 5. Finally, the discussion (Section 6) gives a thorough evaluation of the used methods, and the conclusion (Section 7) gives answers to the research questions.

1.5.1 Proof-of-concept experiments

First, we perform three types of experiments as a proof-of-concept. Experiment 1 considers two Gaussian one-dimensional datasets with slightly different means. This experiment is aimed at showing whether the proposed concept works and whether there are limitations. Experiment 2 and 3 make use of the MNIST digit recognition dataset [19]. In Experiment 2, a subset with 2 classes is drawn, having an imbalanced class distribution. This is the simplest form of verifying whether the weight is applicable in a high-dimensional setting. In Experiment 3, an actual covariate shift is artificially created in a subset of the dataset. This is

performed by applying KMeans clustering on extracted feature embeddings, and creating an imbalance within the classes by means of these clusters.

1.5.2 State-of-the-art comparison

The second part of this work considers a comparison with other state-of-the-art frameworks, using the MNIST-USPS [19][20] and Office-31 [21] datasets. Since the benchmark datasets used in state-of-the-art research consist of a large domain shift, solely a weight network might not be sufficient to transform the distributions. Therefore, we analyze whether we can integrate the weight function with feature-based methods. In Experiment 4, we test whether we can come up with our own unweighted feature-based model. In Experiment 5, we integrate the feature-based model from the previous experiment with the weight function. Moreover, in Experiment 6 we modify the benchmark datasets such that a severe class imbalance emerges, and test the benefits of our method in case of imbalanced domain adaptation.

Chapter 2

Methodology for proof-of-concept

2.1 Background: adversarial learning

As our algorithm is inspired by the Wasserstein GAN, we will briefly discuss its background.

2.1.1 Generative Adversarial Networks

Generative Adversarial Network (GAN) is a deep learning based technique that consists of two antagonistic neural networks which have opposite objectives. Eventually, the goal of the GAN is to generate images which are indistinguishable from the real images, as the fake distribution gradually simulates the real distribution. In the original GAN, the discriminator is a classifier which determines whether a given image originated from the real or generative distribution, by evaluating the conditional probability of the class (real/fake) Y, given the features X. In contrary, the generator aims to produce samples which are close to the real data by modeling its feature distribution, evaluating the probability of the features/data X. It effectuates this imitation by learning a mapping between a latent input vector and the space of the data.

Since the discriminator makes a distinction between two classes, the binary cross-entropy (BCE) cost function is used to determine the GAN loss. This loss function is defined as:

$$\min_d \max_g -\mathbb{E}_{x\sim p_{data}(x)} \left[\log d(x)\right] - \mathbb{E}_{z\sim p_z(z)} \left[\log(1 - d(g(z)))\right] \tag{2.1}$$

In Equation 2.1, d is the discriminator network, g is the generator network, x is the real data, z is the noise vector, $p_{data}(x)$ is the true distribution and $p_z(z)$

is the latent distribution, from which the generator samples to generate fake images. The discriminator d is optimized such that the probability of classifying a sample correctly is maximized, minimizing the above BCE loss. In contrary, the generator attempts to maximize this cost. This leads to a minimax game, where both networks advance in an alternating fashion.

2.1.2 Wasserstein loss

One of the main disadvantages of the BCE loss is that it approaches zero when the discriminator outputs probabilities close to 0 or 1, which leads to vanishing gradients. This could induce mode collapse, which implies that the generator is stuck at generating one type of sample (mode) which has shown to trick the discriminator, as it has stopped learning. To combat this inconvenience, Wasserstein GANs are introduced by Arjovsky, Chintala and Bottou [22] [23]. The WGAN considers a loss function based on the Earth Mover's Distance (EMD) [24], also called Wasserstein-1 distance, which computes the minimum effort required to transform the real distribution to the generative distribution. The main benefit of the EMD compared to BCE loss is that its output values are unbounded, which diminishes its susceptibility to vanishing gradient problems. In this setting, the discriminator is known as a 'critic', as it evaluates a distance rather than probabilities of classes. The Wasserstein loss gives an approximation of the Earth Mover's Distance, which is shown below:

$$\min_g \max_c \mathbb{E}_{x \sim p_{data}(x)} \left[c(x) \right] - \mathbb{E}_{\sim p_z(z)} \left[c(g(z)) \right] \tag{2.2}$$

Likewise, the critic attempts to maximize the distance between the real and generative distribution, while the generator endeavors to minimize the same distance. However, the Wasserstein GAN has the stability requirement that the critic's gradient norm is at most 1 for every possible input, which is controlled by 1-Lipschitz continuity. This is mathematically expressed as $\|\nabla c(x)\|_2 \leq 1$.

A method to enforce 1-L continuity on the Wasserstein GAN is adding a gradient penalty, a form of regularization of the critic's gradient introduced by Gulrajani et al. [25]. Since it is unfeasible to compute the gradient of the critic function with respect to all possible inputs, different points in the feature space are sampled by means of interpolation between real and fake images. Eventually, the new Wasserstein loss function with gradient penalty and penalty weight λ is defined in Equation 2.3, where $\hat{x}$ is the interpolation.

$$\min_g \max_c \mathbb{E}(c(x)) - \mathbb{E}(c(g(z))) + \lambda \mathbb{E}(\|\nabla c(\hat{x})\|_2 - 1)^2 \tag{2.3}$$

2.2 Wasserstein distance for covariate shift

2.2.1 Addition of a weight function

As the Wasserstein loss aims to decrease the distance between two distributions, we can also apply this loss to decrease the divergence between source and target distribution. Instead of making use of a generator which learns a mapping between the noise vector and a generated image, we will train a weight function which is able to map the input samples to an instance weight. This weight reflects the occurrence of the source sample in the target dataset. Multiplying the probability of the source sample with the learned weight enforces that the source distribution is 'transformed' into the target distribution as much as possible. As we learn a weight for every instance, the method can be referred to as *sample-based* unsupervised domain adaptation, being a combination divergence-based and adversarial-based adaptation.

The loss for the adversarial network is defined as:

$$\min_{w} \max_{c} \mathbb{E}_{x \sim s(x)}[c(x)w(x)] - \mathbb{E}_{x \sim t(x)}[c(x)] \tag{2.4}$$

We can observe that the loss function is very similar to the WGAN loss in Equation 2.2. Similar to the WGAN, the critic aims to maximize the distance between the two distributions, providing high values for samples from the source distribution and low values for the target samples. In contrary, the weight network aims to minimize the distance, providing weight outputs which enhance this minimization.

If we rewrite the loss for the critic, the critic loss is defined as:

$$\min \mathbb{E}_{x \sim t(x)}[c(x)] - \mathbb{E}_{x \sim s(x)}[c(x)w(x)] \tag{2.5}$$

Similar to the WGAN, to ensure the the stability of the critic loss and to assure that the loss is an approximation of the Wasserstein distance, the addition of a gradient penalty function is required. The penalty avoids that the critic outputs do not diverge excessively. However, not only do we want to control the gradient of the outputs, we also want to stabilize the actual output values of the critic. Therefore, we introduce a second penalty, which is similar to L2-regularization, but in function space instead of parameter space. In this way, we can control that the critic does not provide overly large or small outputs. As already explained in Section 2.1.2, it is infeasible to compute the output values of the critic for every possible input. Likewise, we make use of an auxiliary distribution $e(x)$ to draw samples from.

Eventually, this leads to:

$$\min \mathbb{E}_{x \sim t(x)}[c(x)] - \mathbb{E}_{x \sim s(x)}[c(x)w(x)] + \frac{\mu}{2}\mathbb{E}_{x \sim e(x)}[c(x)]^2 + \frac{\nu}{2}\mathbb{E}_{x \sim e(x)}\left[\frac{dc}{dx}(x)\right]^2 \tag{2.6}$$

As observed in the loss function in Equation 2.4, the weight network only evaluates the source data and has no access to the target data. The weight network has no influence on the term where the critic evaluates the target data, and therefore it is not able to optimize its parameters with respect to this term. Hence, the loss function for the weight only includes the source data term. With the addition of an L2-penalty, this leads to:

$$\min \mathbb{E}_{x\sim s(x)}[c(x)w(x)] + \frac{\alpha}{2}\mathbb{E}_{x\sim e(x)}[w(x)]^2 \tag{2.7}$$

2.2.2 Intermezzo: theoretical solution

Under the assumption that the underlying distributions $s(x)$ and $t(x)$ are known, we could statistically approximate the expected values in the loss with averages over distributions. We can now rewrite the loss functions in Equation 2.6 and 2.7:

$$\begin{aligned} L^c(\theta^c, \theta^w) = &\int t(x)c(x;\theta^c)dx - \int s(x)w(x;\theta^w)c(x;\theta^c)dx \\ &+\frac{\mu}{2}\int e(x)\left[c(x;\theta^c)\right]^2 dx + \frac{\nu}{2}\int e(x)\left[\frac{dc}{dx}(x;\theta^c)\right]^2 dx \end{aligned} \tag{2.8}$$

$$L^w(\theta^c, \theta^w) = \int s(x)w(x;\theta^w)c(x;\theta^w)dx + \frac{\alpha}{2}\int e(x)\left[c(x;\theta^w)\right]^2 dx \tag{2.9}$$

Furthermore, we can approximate the loss as a function of functions $c(x)$ and $w(x)$ when we assume that the network parameters θ^c and θ^w can be anything they need to be:

$$\begin{aligned} L^c[c, w] = &\int t(x)c(x)dx - \int s(x)w(x)c(x)dx \\ &+\frac{\mu}{2}\int e(x)\left[c(x)\right]^2 dx + \frac{\nu}{2}\int e(x)\left[\frac{dc}{dx}(x)\right]^2 dx \end{aligned} \tag{2.10}$$

$$L^w[c, w] = \int s(x)w(x)c(x)dx + \frac{\alpha}{2}\int e(x)\left[w(x)\right]^2 dx \tag{2.11}$$

Gradient descent in parameter space will aim to find the global minimum of the loss function(s). However, with the derived functional approximation in Equation 2.10 and 2.11, we could compute the theoretical optimal solution by deriving the minimum in function space. Instead of computing the minimum

with respect to parameters θ^c and θ^w, we compute the minimum with respect to functions $c(x)$ and $w(x)$. Two important calculus rules to compute the gradients of functions are shown in Equation 2.12 and 2.13:

$$\frac{\delta}{\delta g}\int_{-\infty}^{\infty} F(g(x))dx = \frac{\partial F}{\partial g}(g(x)) \tag{2.12}$$

$$\frac{\delta}{\delta g}\int_{-\infty}^{\infty} F(g'(x))dx = -\frac{d}{dx}\frac{\partial F}{\partial g'}(g'(x)) \tag{2.13}$$

Applying Equation 2.12 and 2.13 to compute the partial derivatives of the loss function with respect to $c(x)$ and $w(x)$ leads to:

$$\frac{\delta L}{\delta c}[c,w] = t(x) - s(x)w(x) + \mu e(x)c(x) - \nu\frac{d}{dx}\left(e(x)\frac{dc}{dx}(x)\right) \tag{2.14}$$

$$\frac{\delta L}{\delta w}[c,w] = -s(x)c(x) + \alpha e(x)w(x) \tag{2.15}$$

In case we gradually decrease α to 0, for $\frac{\delta L}{\delta c}[c,w] = 0$ and $\frac{\delta L}{\delta w}[c,w] = 0$, we will obtain:

$$t^*(x) = s(x)w(x) - \mu e(x)c(x) + \nu\frac{d}{dx}\left(e(x)\frac{dc}{dx}(x)\right) \tag{2.16}$$

$$c^*(x) = \alpha\frac{e(x)w(x)}{s(x)} = 0 \text{ (if } \alpha = 0) \tag{2.17}$$

If $c^*(x)$ will become zero, the penalties for the critic will also cancel out. In that case, the theoretical optimum will become:

$$t^*(x) = s(x)w(x) \tag{2.18}$$

$$c^*(x) = 0 \tag{2.19}$$

We can make use of the transformed source distribution t^* in a controlled experiment, using synthetic data of which we know the probability distributions.

2.2.3 Practical application

In practice, the underlying source and target distribution $s(x)$ and $t(x)$ are not known and we only have access to a sample drawn from this distribution. Therefore, we make use of averages over the datasets. For the critic and weight loss, the loss can be rewritten from Equation 2.8 and 2.9 to:

$$L^c(\theta^c, \theta^w) = \frac{1}{|T|}\sum_{x\in T} c(x;\theta^c) - \frac{1}{|S|}\sum_{x\in S} w(x;\theta^w)c(x;\theta^c) \tag{2.20}$$
$$+\frac{\mu}{2\,|E|}\sum_{x\in E}\left[c(x;\theta^c)\right]^2 + \frac{\nu}{2\,|E|}\sum_{x\in E}\left[\frac{dc}{dx}(x;\theta^c)\right]^2$$

$$L^w(\theta^c, \theta^w) = \frac{1}{|S|}\sum_{x\in S} w(x;\theta^w)c(x;\theta^c) + \frac{\alpha}{2\,|E|}\sum_{x\in E}\left[c(x;\theta^w)\right]^2 \tag{2.21}$$

In the above equations, the source and target datasets are denoted as S and T, while the data samples drawn from these datasets are given as x. An auxiliary dataset to compute the penalties is noted as E. The networks themselves are noted as c and w, while its weights and biases are defined as θ^c respectively θ^w.

2.2.4 Constraints

The optimization procedure has to comply with several constraints. First of all, the computed weights should be non-negative, such than we do not obtain negative probabilities or negative frequencies of occurrence. Moreover, the obtained weighted source distribution is required to be a valid probability distribution, which implies that the probabilities should sum up to 1:

$$1 = \int t^*(x)dx = \int s(x)w(x)dx \approx \frac{1}{|S|}\sum_{x\in S} w(x) \tag{2.22}$$

This also implies that the sum of the weights should sum up to the number of samples $|S|$, such that $\frac{\sum_{x\in S} w(x)}{|S|} = \frac{|S|}{|S|} = 1$. We can ensure the validity of the transformed distribution by means of normalizing the weights. In case of taking averages over datasets, we can divide the weights by the average weight: $\omega = \frac{1}{|S|}\sum_{x\in S} w(x)$. In case of taking averages over probability distributions, this is equivalent to: $\omega = \int s(x)w(x)$. Applying this normalization to the weights should guarantee that the weighted distribution is valid:

$$\frac{1}{|S|}\sum_{x\in S}\frac{w(x)}{\omega} = \frac{1}{|S|}\sum_{x\in S}\frac{w(x)}{\frac{1}{|S|}\sum_{x\in S} w(x)} = 1 \tag{2.23}$$

$$\int s(x)\frac{w(x)}{\omega}dx = \int s(x)\frac{w(x)}{\int s(x)w(x)dx}dx = 1 \tag{2.24}$$

2.3 Datasets

2.3.1 One-dimensional Gaussian distributions

The efficacy of the proposed method can first be examined in a controlled experiment with simple synthetic data. The advantage of this experiment is that

we know the underlying marginal distributions from which the samples are drawn. We choose all three distributions to be one-dimensional Gaussian distributions with a different mean and variance. It can be noted that for simplicity, we are not dealing with a classification problem, but rather a regression problem.

$$t(x) = \mathcal{N}(-2, 1)$$
$$s(x) = \mathcal{N}(-1, 2)$$
$$e(x) = \mathcal{N}(4, 10)$$

The three distributions and 1000 drawn samples are shown in Figure 2.1. Note that p refers to the target distribution, q refers to the source distribution. The auxiliary distribution $e(x)$ is chosen to have a relatively high variance, such that it contains wide tails. In this way, almost all values in the domain have an almost equal probability of occurrence. As we use the auxiliary distribution to determine the inputs of the function which are checked for penalization, we want to make sure that all values in the domain are covered.

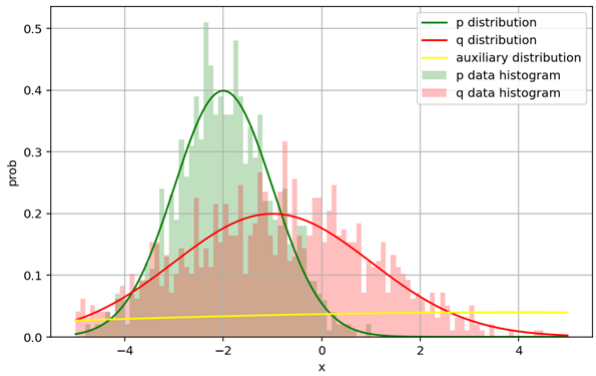

Figure 2.1: Gaussian source (q), target (p) and auxiliary (e) distributions

2.3.2 Modified MNIST dataset

The MNIST dataset is a digit recognition image dataset consisting of 60,000 train samples and 10,000 test samples. It contains 10 classes, having a resolution of 28×28×1. We will modify the MNIST dataset such that we can test the efficacy of our method.

Class imbalance. The simplest dataset modification is to artificially create a class imbalance within the dataset. Theoretically, this does not comply with the definition of covariate shift but rather with prior probability shift, as $P(Y)^s \neq P(Y)^t$ instead of $P(X)^s \neq P(X)^t$. However, this experiment will give more insights into the efficacy of the weight function to transform the samples on an instance level, such that the datasets become more similar.

Covariate shift. The second dataset modification consists of an actual artificial covariate shift. In this dataset, the aim is to divide a class into two subclasses. For example, the class with digit 1 is subdivided into a group with oblique 1-digits, while the other group consists of straight 1-digits. For the source dataset, there is created an imbalance between the oblique and straight 1-digits, e.g. 80% oblique 1-digits and 20% straight 1-digits. The target dataset has an opposite subclass distribution: 20% of oblique 1-digits and 80% of straight 1-digits. Meanwhile, the number of samples per class is the same for the source and target dataset. In this way, $P(Y)^s = P(Y)^t$, but $P(X)^s \neq P(X)^t$. The subclasses are obtained by extracting feature embeddings with a CNN. On these embeddings, KMeans clustering is performed on these embeddings, obtaining 20 subclasses (2 per class). Afterwards, the subclasses are used to create an imbalance within the class between the source and target dataset.

2.4 Evaluation metrics

Binary domain classifier. Most domain adaptation methods evaluate their results by means of the classification accuracy of the trained adaptation model on the target dataset. However, in the controlled experiments we will perform in the proof-of-concept phase, the domain shift is rather small. Therefore, a source classifier trained on the source dataset and tested on the target dataset will still have a relatively high accuracy. Moreover, in the first proof-of-concept experiment, we test the algorithm on a regression problem instead of a classification problem, which implies that we do not have labels. During this phase, we simply want to investigate whether the algorithm is able to make the two distributions indistinguishable. Therefore, we will employ a binary classifier which classifies from which domain the data samples originate from.

The input of the classifier consists of the merged source and target data, which is labeled as 0 for the target data and 1 for the source data. First, we train the classifier on the unweighted merged data and note its performance. Secondly, we train the classifier on the weighted merged data. Since the weights should be able to convert the source distribution into the target distribution, the classifier should have more difficulty with discriminating between the domains in the weighted setup. In this way, we would expect a higher accuracy for the unweighted classifier than for the weighted classifier. In the ideal situation, the weights will render a completely domain-invariant representation, such that the classifier is not able to discriminate between the domains and gives an accuracy close to 50%. This approach is aligned with the Domain Confusion Loss proposed by Tzeng et al. [17], which also makes use of the BCE loss. In their work, the samples have the domain as their label, and the parameters in the network are optimized such that the sigmoid outputs a probability of 0.5 for each domain. In our case, we do not use this method for optimization, but simply as an evaluation method.

There are various options for the choice of the domain classifier. In this research,

we make use of three types of classifiers: Logistic Regression, the modified LeNet architecture [26] and a custom made CNN architecture. The modified LeNet architecture is used by most papers as a label classifier for the digit recognition domain adaptation problem. We adapt this architecture such that it gives a binary output instead of a multi-class output. We assert the discriminative ability of the architectures by first training them on a dataset with the two domains being two different classes. Furthermore, we can test whether the accuracy of the classifiers actually decreases towards 50% in case we input two target datasets with the same distribution. In this way, we can confirm that in the best possible scenario, we are able to reach an accuracy of 50%, implying that the domain classifier is not able to distinguish the datasets from each other.

Average pixel image. As we are applying the proposed method on image datasets, we are able to construct the average pixel image. For every pixel value i, j, we compute the average pixel value over all samples in source dataset Q. We re-weight the average value with the average weight over all source samples. In case we are also using penalties for the critic loss, we also add the computed values to the average pixels.

$$\frac{1}{|T|}\sum_{x\in T} x_{ij}^{*} = \frac{1}{|S|}\sum_{x\in S} x_{ij}w(x) + \mu\frac{1}{|E|}\sum_{x\in E} x_{ij}c(x) + \nu\frac{1}{|E|}\sum_{x\in E}\frac{\partial c}{\partial x_{ij}(x)} \quad (2.25)$$

t-SNE. T-Distributed Stochastic Neighbor Embedding (t-SNE) [27] is a stochastic method which focuses on reducing the dimensionality of data into lower dimensions, while preserving the information stored in the data. To measure the discrepancy between the high-dimensional and lower dimensional data, it uses the the Kullback-Leibner (KL) divergence, minimizes the difference between the resemblance between instances in the high-dimensional data and the resemblance between instances in the reduced data. In the t-SNE visualization, the sample size of the marker is related to the weight value, giving larger markers if the same has a higher weight. In this way, we can check whether a higher weight is assigned to the samples which are artificially under-represented in the source dataset.

Additionally, we can use the extracted t-SNE dimensions to construct a 2D histogram image. The color of the pixel shows the frequency of occurrence of the low-dimensional mapping of the sample. In the weighted case, we would expect the frequencies of occurrence from the mapping of the target dataset and the re-weighted source dataset be almost similar.

Histogram of function outputs. We can also visualize the frequencies of the output values of the weight and critic network. In this way, we can check whether the weight does output non-negative weights, as well as whether it gives outputs which are close to the expected outputs. We can also check whether the outputs of the critic do not have too extreme values.

Chapter 3

Results for proof-of-concept

3.1 Experiment 1: Gaussian distributions

In the first proof-of-concept experiment, we will test whether our theoretical solution is valid in practice. We will analyze whether the model converges to:

$$t^*(x) \approx s(x)w(x) \tag{3.1}$$
$$c^*(x) \approx 0 \tag{3.2}$$

3.1.1 Loss with distribution averages

When we apply the ideal loss function, we compute the loss by means of distribution averages, as shown in Equation 2.8 and 2.9. Instead of using sampled data points, we use the complete distribution as input for the adversarial network. Since the input is low-dimensional, we use a relatively simple architecture for the critic and weight. Both consist of three fully connected (FC) layers with 1024, 512 and 1 neuron. After the first and second FC layer, we use a Sigmoid Linear Unit (SiLU) activation layer, which is defined as $x * \sigma(x)$, with $\sigma(x)$ being the logistic sigmoid. This activation function resembles the ReLU function, except that it does not have the kink at $x = 0$. For the weight function, we also experiment with squaring the output after the last FC layer to enforce non-negativity. It can be observed that both networks do not include an activation layer to compute their outputs, to avoid that we restrict their outputs.

The authors of the vanilla WGAN paper [22] state that a momentum-based optimizer such as Adam may destabilize training. However, the authors of the WGAN-GP paper [25] found that for the WGAN-GP optimizer Adam

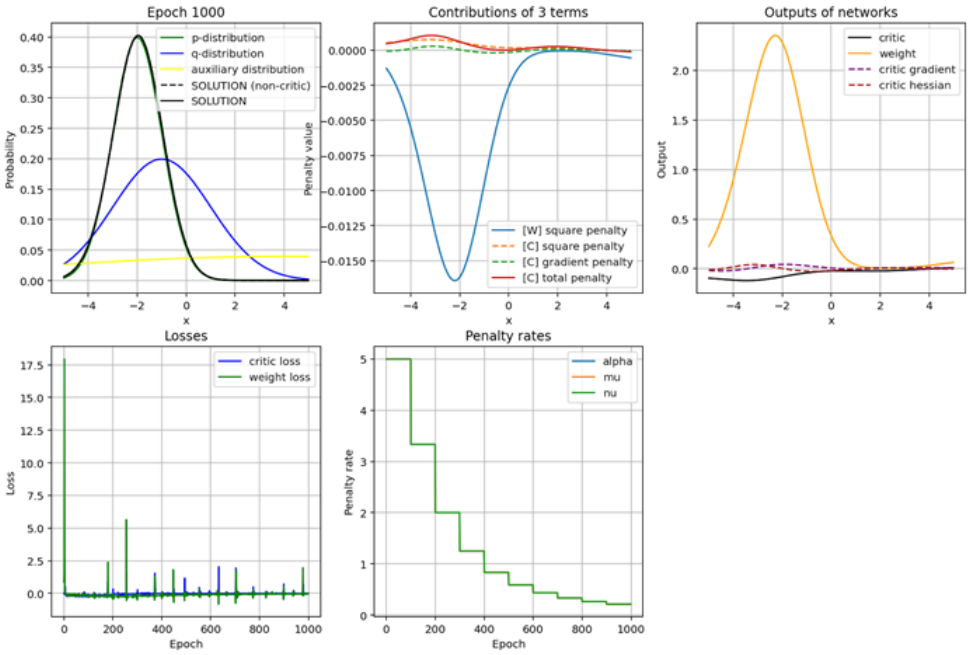

Figure 3.1: Results for ideal loss function
source (q), target (p), target* (solution)

outperforms RMSprop, therefore we experiment with both. Figure 3.1 shows the results after 1000 epochs with optimizer RMSprop and a learning rate of 0.005. The upper left figure shows the source $s(x)$, target $t(x)$ and auxiliary $e(x)$ distributions in blue, green respectively yellow. The black continuous line represents the obtained re-weighted source distribution $t^*(x)$, while the black dotted line called *non_critic* represents the solution $t^*(x)$ without penalties. The second image shows the contributions of the penalty terms to the loss functions, while the third image shows the output of the functions. The last two images represent the losses and the course of the penalty rates. Analogous to the derived solution in Equation 3.1, the critics has become close to zero, which can be observed in the third figure. This results in cancelling out the penalties, such that $t^*(x) = s(x)w(x)$. This can be observed in the first figure, as the black dotted line, continuous black line and green line are identical, implying that $t^*(x) \approx s(x)w(x)$.

3.1.2 Loss with sample averages

In case we cannot use the distributions $t(x)$ and $s(x)$ as input for the networks, we make use of a sample drawn from the distribution. Therefore, the loss is computed by taking averages over the data points, as shown in Equation 2.20 and 2.21 in Section 2.2.3. In Figure 3.2, the results are shown for two drawn

samples consisting of 1000 data points. Since this is a controlled experiment where we know the underlying distributions, we can still compute $t^*(x)$ with $s(x)$ and compare it with $t(x)$. The results show that even when we use two samples drawn from the distributions, the adversarial network is still able to reconstruct the target distribution. However, from the third figure it can be derived that the critic has not (yet) become zero, leading to very small non-zero critic penalty terms. This might indicate that the model has not converged yet.

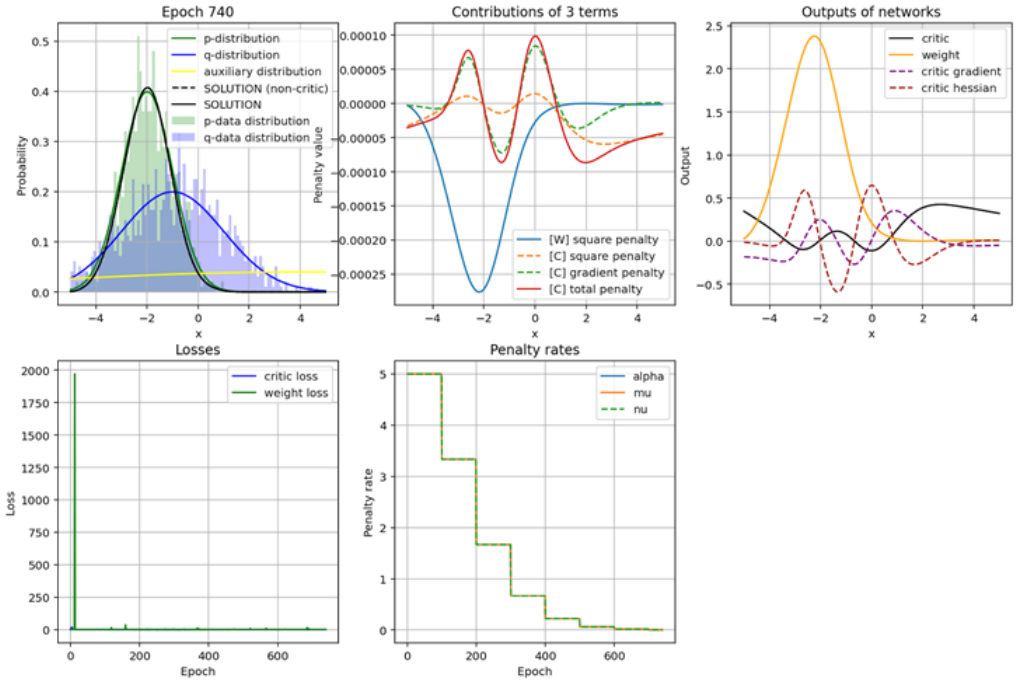

Figure 3.2: Results for approximated loss function

3.1.3 Evaluation

We will evaluate the domain invariance of the transformed distribution with a domain classifier. Since we are not dealing with complex data, we will use a simple Multi-Layer Perceptron (MLP) for the domain classification. The MLP consists of three FC layers (64, 128, 1 neurons) with ReLU activation in-between. The prediction is performed with a Sigmoid function. We use the Adam optimizer and a learning rate of 0.0002. The results for the average over sample experiments are shown in Table 3.1. The results indicate that weighting the source data decreases the classifier's ability to distinguish between the two domains, as the accuracy decreases from 69.25% to 58.10% after 2000 epochs. Applying the normalized weights shows an even better resemblance between the datasets, decreasing the accuracy to 52.90%.

model	accuracy	recall	precision
unweighted	0.6925	0.5570	0.7641
weighted	0.5810	0.5110	0.5942
weighted$_{norm}$	0.5290	0.5670	**0.5269**

Table 3.1: Domain classifier performance metrics

We can also visualize the output probabilities of the MLP and compare these with the two histograms of the datasets. For the unweighted and weighted (normalized) datasets, the histograms and MLP outputs are shown in Figure 3.3 and 3.4. It must be recollected that the target dataset T (p) is labeled as 0, while the source dataset S (q) is labeled as 1. Therefore, an output probability higher than 0.5 will be classified as source dataset. Figure 3.3 shows that the data points where $t(x) > s(x)$, the classifier predicts that the value originates from dataset T, giving an output probability lower than 0.5. This applies (approximately) to the values in the range $[-4, 0]$. The larger the difference between $t(x)$ and $s(x)$, the more confident the classifier becomes in its predictions, as for values close to 2 the probability is the closest to 0. Moreover, for the data points where $s(x) > t(x)$, the classifier outputs a probability higher than 0.5, predicting that the sample belongs to source dataset S. Figure 3.4 compares the outputs probabilities to the weighted histogram of the source data. It is evident that the classifier has difficulty assigning a clear-cut class to the data; it has learned that all values in its range have nearly equal probability of belonging to one of the two domains.

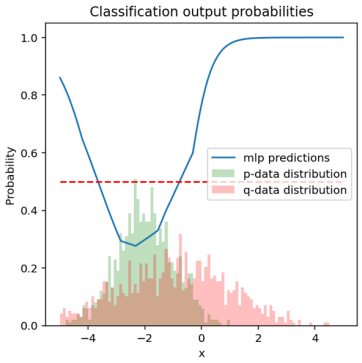

Figure 3.3: Unweighted results

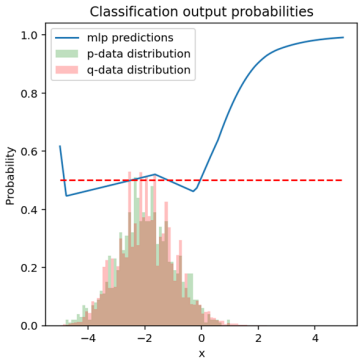

Figure 3.4: Weighted results

3.1.4 Limitations

In some cases, the adversarial network has more difficulty reconstructing the target distribution. This is particularly the case when $s(x) = 0$ and $t(x) > 0$. This means that the target dataset contains features which are not present in the source data. Hence, the model cannot assign weights to features which do not

exist in the source data, and thus is not able to reproduce the target distribution. This is shown in Figure 3.6 and 3.6.

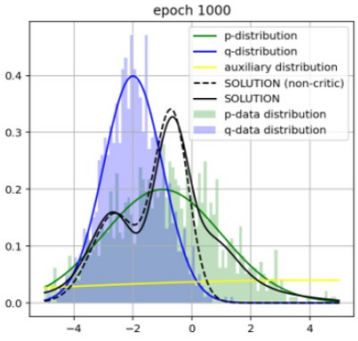

Figure 3.5: Non-overlap example

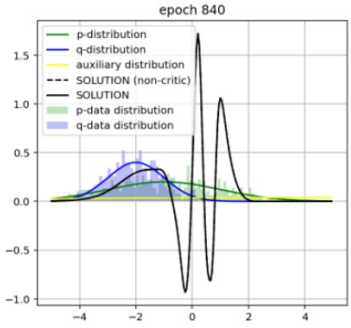

Figure 3.6: Non-overlap example

3.2 Modified MNIST dataset

The second proof-of-concept experiments are performed on modified MNIST datasets. Obviously, the MNIST data itself does not contain two different domains; however, we can aim to artificially create two domains within the dataset.

3.2.1 Experiment 2: class prior problem

The most simplistic experiments are performed on a subset of 1000 samples from 2 classes: 0-digits and 1-digits. The class distribution of the two datasets is shown in Table 3.2. Source dataset S consists of 20% 0-digits and 80% 1-digits. Furthermore, target dataset T consists of 80% 1-digits and 20% 0-digits. Therefore, we would expect that the weight network learns to give a weight of 4 to the 0-digits and a weight of 1/4 to the 1-digits, such that the class distribution is similar to the target dataset. Note that in this case, we are adjusting prior probabilities the $P(Y^s)$ and $P(Y^t)$ rather than the marginal probabilities $P(X^s)$ and $P(X^t)$. However, this use-case is simply a toy experiment where we exactly know which weights to expect for the samples; in this way, we will discover whether the weight is applicable for image data.

k	$P(Y=k)^s$	$P(Y=k)^t$	$\mathbb{E}[w(x)]$
0	0.20	0.80	4
1	0.80	0.20	$\frac{1}{4}$

Table 3.2: Class distributions and expected source weight

For these experiments, we used a critic consisting of two convolutional blocks.

The first block consists of a convolutional layer with 32 filters and a kernel size of 9, followed by Scaled Exponential Linear Unit (SELU). The last block also consists of the same number of filters and kernel size. The weight network has a similar architecture. The penalty rate for the critic L2 penalty μ is set to 8, the rate for the critic gradient penalty ν is set to 5, the rate for the weight L2 penalty α is set to 0. The optimizer is RMSprop with a momentum of 0.00005, having a learning rate of 0.001. The auxiliary data is computed by taking the union of the source and target data in the batch. The batch size is 100, and the adversarial model is run for 1000 epochs.

As explained in the methodology, the weight should have non-negative outputs, such that we do not obtain negative probabilities. Therefore, we could square the outputs of the weight in its last network layer. However, this means the weight network can produce a minimum output of 0, and thus has an output range $[0, \infty]$. In contrary, the critic does not have this restriction and can output values in the range $[-\infty, \infty]$, although the penalties control these outputs. If we again consider the weight loss $\mathbb{E}_{x\sim s(x)}[c(x)w(x)]$, we can observe that the minimization of the weight loss is strongly influenced by the output of the critic. The weight will desire that the critic outputs values close to 0, such that the weight loss also approaches 0. However, the critic aims to minimize $\mathbb{E}_{x\sim t(x)}[c(x)] - \mathbb{E}_{x\sim s(x)}[c(x)w(x)]$, and thus aims to maximize $\mathbb{E}_{x\sim s(x)}[c(x)w(x)]$. As the weight will give outputs close to 0, the critic will give very high outputs for the source data. Therefore, the critic has a much easier task minimizing its loss.

Accordingly, we experiment with an unbounded weight function. The results for the binary domain classifier for the train (dataset 1) and test set (dataset 2) are shown in Figure 3.7 and 3.8. The figure shows that especially for the validation set, the classifier which includes the weights of the source domain has difficulty separating the two datasets and gives an accuracy close to 0.50. Moreover, Figure 3.9 shows the results of the classifier on two target datasets with the same class distribution. This plot is added as a baseline; on this dataset, the classifier is supposed to not be able to see a difference.

Furthermore, a visual comparison between the target dataset T and the (un)weighted source dataset S is shown in Figure 3.10 and 3.11. Figure 3.10 shows the original case, where there is a class imbalance. Figure 3.11 shows the weighted version of the source dataset compared to the target dataset. We can observe the average pixel image resembles the average image of the target dataset, implying that the distributions have become similar.

We can also perform t-SNE with the feature embeddings extracted with the classifier, where the marker size is relative to the assigned weight. The output is shown in Figure 3.12. It is evident that the 0-digits from the source dataset are assigned to a high weight, while the 1-digits in the source set are assigned to a low weight. Moreover, Figure 3.13 and 3.14 the 2D histograms of the two t-SNE output dimensions are shown. Note that the 2D histograms are flipped on the horizontal and vertical axis compared to the t-SNE in Figure 3.12. Thus,

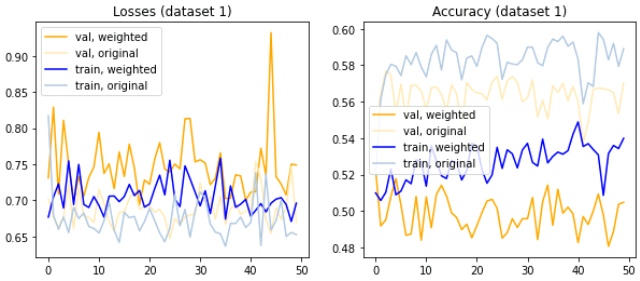

Figure 3.7: Binary domain classifier dataset 1 (train)

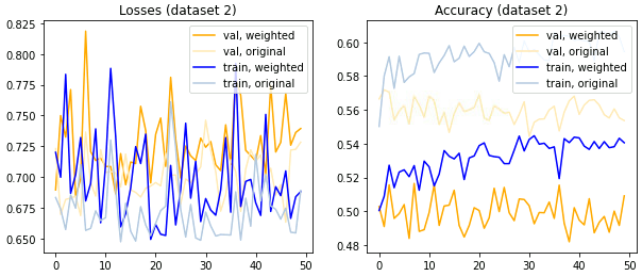

Figure 3.8: Binary domain classifier dataset 2 (test)

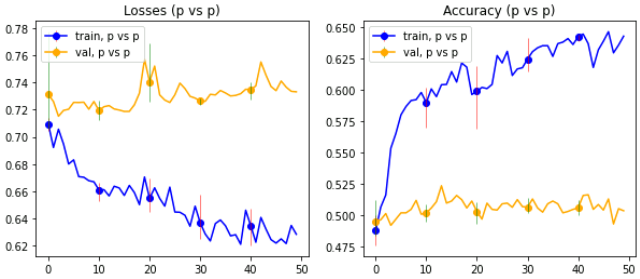

Figure 3.9: Binary domain classifier target-only

the upper left corner in the 2D histogram represents the lower right corner in the original t-SNE. The color of the pixel represents the frequency of sample occurrence in that part of the image. The first image shows a large difference between the area in the low-dimensional mapping where target dataset P and source dataset Q have the most samples. This makes sense, as the target dataset contains most 0-digits, which in the 2D histogram are grouped in the upper left corner, while the source dataset consists of mainly 1-digits. However, the second image shows that in the weighted source dataset, the transformed Q also consists of most 0-digits, while fewer 1-digits.

Figure 3.15 shows the histograms of the weight outputs for the two classes. As we expect an average weight of 4 for the 0-digits and an average weight of 0.25

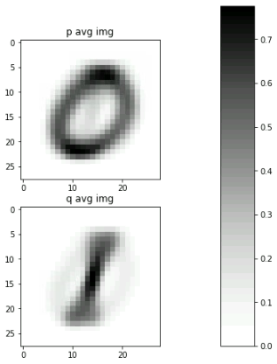

Figure 3.10: Average pixel image unweighted

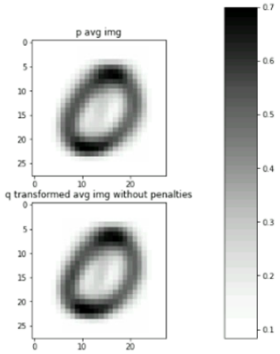

Figure 3.11: Average pixel image weighted

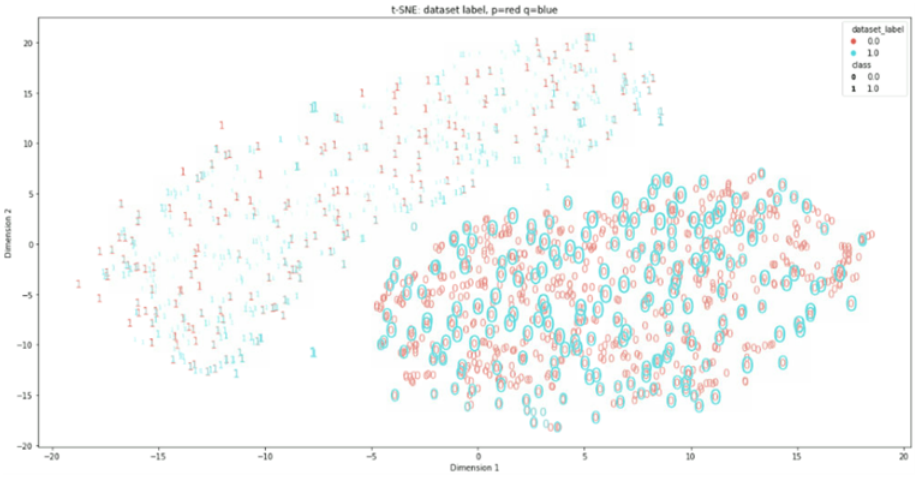

Figure 3.12: t-SNE results

for the 1-digits, we can observe that the weight outputs are consistent with this expectation. Nonetheless, the weight output for the 1-digits show that a few samples are assigned to negative weights. Moreover, Figure 3.16 shows the outputs of the critic, and shows that the critic tends to provide and output around 0.

3.2.2 Experiment 3: covariate shift problem

The second proof-of-concept consists of dividing the classes into subclasses, such that the weight is able to learn weights for more specific features within a class. During this subdivision, it is important that $P(Y)^s = P(Y)^t$, but $P(X)^s \neq P(X)^t$. By creating an imbalance between the subclasses within a

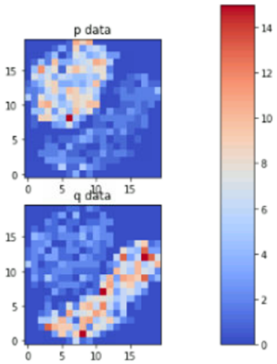

Figure 3.13: 2D histogram, unweighted

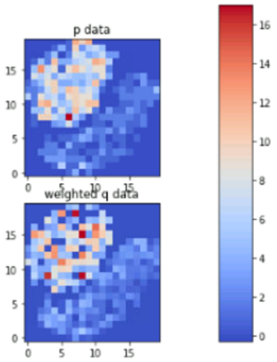

Figure 3.14: 2D histogram, weighted

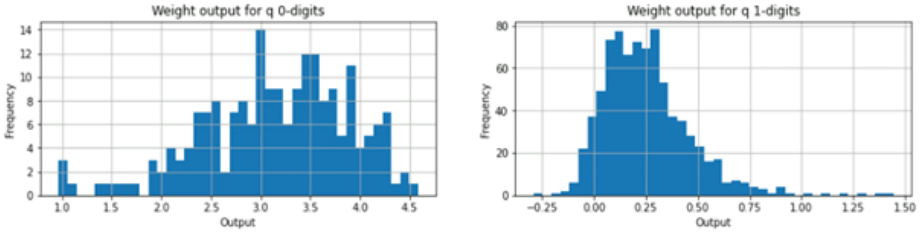

Figure 3.15: Histogram of weight outputs

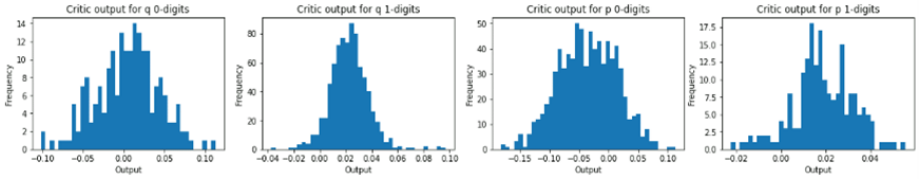

Figure 3.16: Histogram of critic outputs

class, while the number of samples per class stays equal, we can enable that $P(X)^s \neq P(X)^t$.

For this experiment, we select the first 5 classes (class 0 until 4). We first extract feature embeddings with a CNN, which will be the input for the KMeans analysis. We choose $k = 10$ clusters, such that every class consists of 2 subclasses. The results for the t-SNE is shown in Figure 3.17. The samples which are clustered in the wrong class are omitted, such that every class consists of 2 clusters.

With these results, we create four new datasets: a source and target train set (dataset 1), and a source and target test set (dataset 2). In all four datasets, the number of samples is 10,000, while all five classes are balanced. Thus, for both

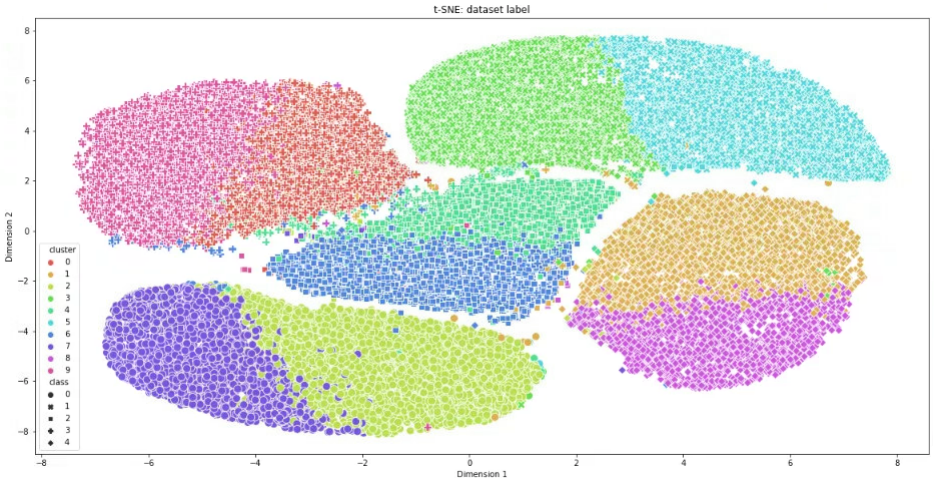

Figure 3.17: t-SNE of subset with 5 classes

class	**cluster**	**expected weight**	**actual weight**
0	2	1	0.97
0	7	1	0.98
1	3	1	0.98
1	5	1	0.99
2	4	1	0.90
2	6	1	0.99
3	0	0.33	0.55
3	9	3	1.92
4	1	0.33	0.56
4	8	3	2.03

Table 3.3: Expected weight vs actual weight per cluster

target and source dataset:

$$P(Y = 0) = P(Y = 1) = P(Y = 2) = P(Y = 3) = P(Y = 4) \tag{3.3}$$

In Table 3.3, the mapping between the cluster number and class is shown. Classes 0, 1 and 2 contain a balanced proportion of their two subclasses. However, class 3 and 4 consist of 25% of one cluster, while 75% of the other cluster in the source dataset, while the target dataset contains the opposite proportions of the clusters within the class. Therefore, we expect an average weight of $\frac{1}{3}$ and 3 for the two clusters within the class, such that source and target dataset have the same proportions of the subclasses.

We run the adversarial re-weighting model on the source and target dataset with the same architecture for the weight and critic network as the previous experiments. The loss over epochs for the critic and weight are shown in Figure 3.18.

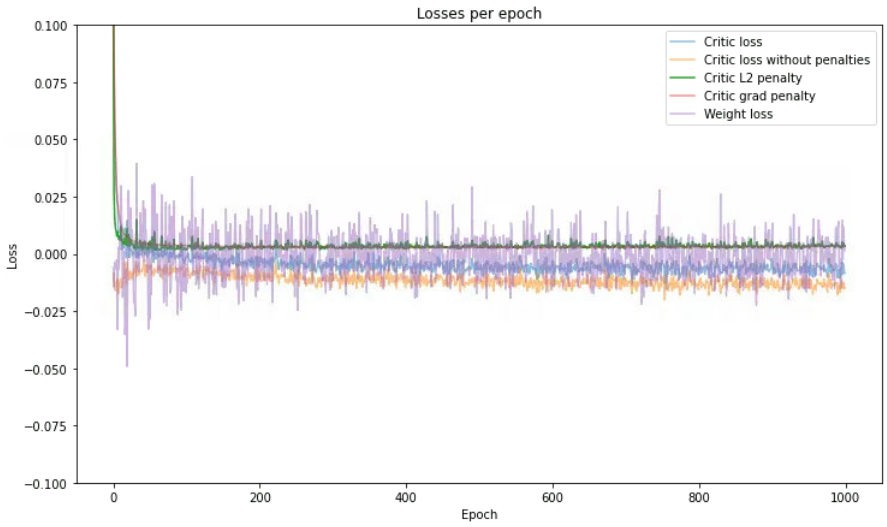

Figure 3.18: Losses during adversarial optimization

After training, we can evaluate the domain invariance of the re-weighted source dataset with the domain classifier. We run the classifier on the merged train source and target dataset (dataset 1), and also test the adversarial model with the test source and target dataset with the same distribution (dataset 2). Hence, dataset 1 and dataset 2 both consist of 20,000 samples. Furthermore, we divide merged datasets into train and validation set in ratio 70:30.

The results for the binary domain classifier are shown in Figure 3.19, 3.20 and 3.21. The visualizations clearly show that the learned weights are able to transform the source datasets in such a way that the domain classifier is unable to see the difference between the target and re-weighted source dataset. This can be observed since the bold lines (weighted) have an accuracy which is closer to 0.50 than the accuracies of the transparent lines (original). For both dataset 1 and 2, the validation accuracy drops from 0.56 to around 0.50.

Figure 3.21 shows the results for the domain classifier on the merged target datasets from dataset 1 and dataset 2. This is used as a baseline, as the two target datasets have the same subclass distribution.

We can also visualize the results by means of performing a t-SNE analysis on the feature embeddings. Figure 3.22 shows the results for the source dataset (test set). The size of the marker represents the size of the weight. Comparing the plot to the t-SNE in Figure 3.17 shows that the clusters which are under-represented

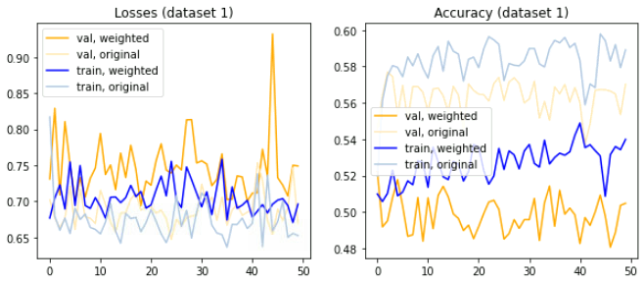

Figure 3.19: Binary domain classifier dataset 1 (train)

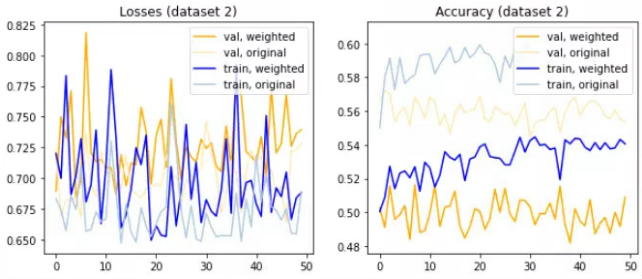

Figure 3.20: Binary domain classifier dataset 2 (test)

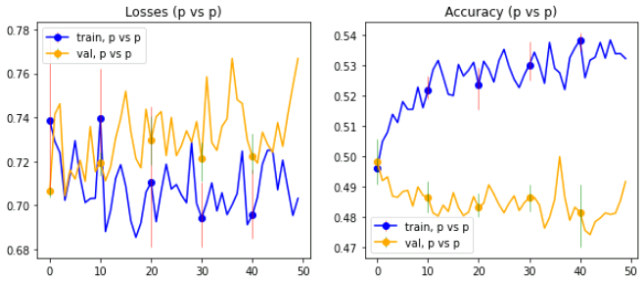

Figure 3.21: Binary domain classifier target-only

have the highest weight (cluster 8 of class 4 and cluster 9 of class 3). The t-SNE plots for class 3 and 4 from the source dataset are shown in Appendix A.

We can also show with plotting the average image per class, that the model has learned to align the distributions. This is shown in Figure 3.23 and 3.24. We observe that the weight has learned that the target dataset contains more digits with straight instead of oblique features.

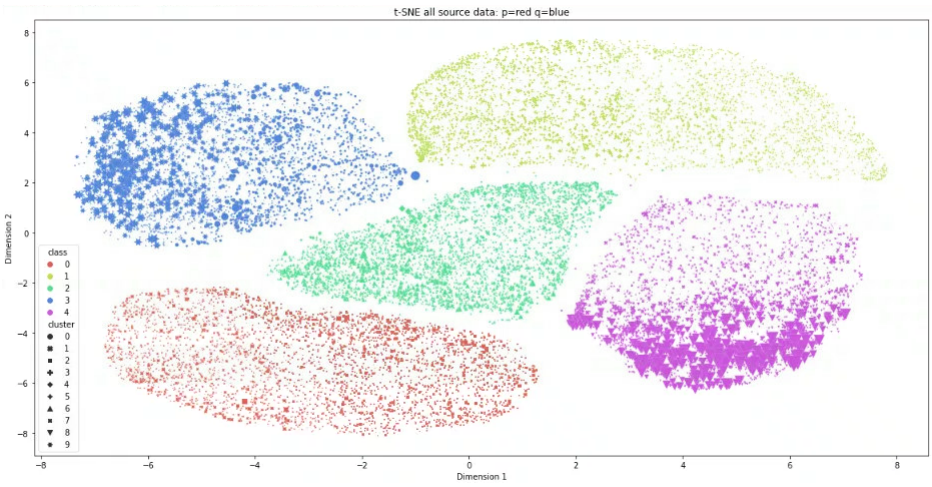

Figure 3.22

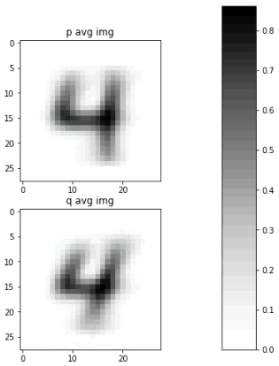

Figure 3.23: Average pixel image class 4, unweighted

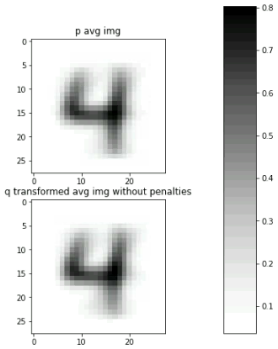

Figure 3.24: Average pixel image class 4, weighted

Chapter 4

Methodology for state-of-the-art comparison

In the experiments with the one-dimensional Gaussians (Experiment 1), we observed that when specific features are not present in the source data while being present in the target data, a weight might not be suitable. It is not possible to assign a weight to a feature which does not exist. During the modified MNIST experiments (Experiment 2 and 3), the shift between the features in the samples was very small, and thus a weight was able to transform the source into target domain. Hence, the above methods for the proof-of-concept experiments have an important constraint: they assume that there is a lot of overlap between the source and target samples, as the features between the datasets are similar. However, in most cases this assumption will not be met, implying that there will be a larger domain shift. For example, when we want to adapt from one weather condition to another, simply learning a weight for the samples might not be sufficiently effective, as we also need a transformation of the features. Therefore, we will adapt our method used for the proof-of-concept in order to employ the method for a larger domain shift.

Our methodology consists of two parts. First, we will develop a novel unweighted feature-based framework, based on previous models. We will first describe the two existing models which we will use as inspiration for our own feature-based model. We compare our unweighted feature-based method with other approaches in Experiment 4. Afterwards, we can use this feature-based model to combine it with a weight function in Experiment 5 and 6. In Experiment 5, we will test whether and under which conditions our weighted feature-based model works best. In Experiment 6, we will test whether our weighted feature-based model works better than an unweighted feature-based model under severe class

imbalance.

4.1 Baseline methods

As explained in the Related Work, *feature-based* methods aim to learn a feature transformation that maps the data from both domains into a shared feature space. This is generally achieved with adversarial learning, where a discriminator or a critic aims to differ between the two domains, while a feature extractor (also called encoder) aims to learn a domain-invariant feature representation. This learned feature extractor is merged with a classifier, such that an adapted *classification model* is formed, as shown in Figure 4.1. Note that there is a difference in meaning between *classifier* and *classification model*. The difference between these feature models and our weighted model is that instead of learning a weight for every instance, these techniques learn a mapping between the target data and an encoding in lower dimensional space.

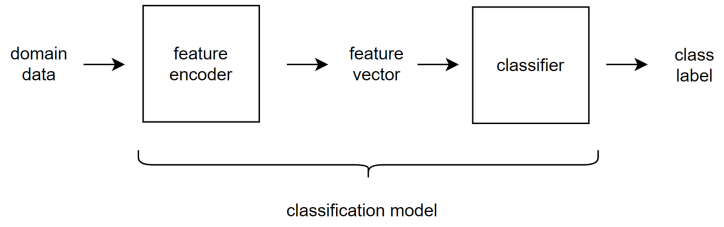

Figure 4.1: Splitting the classification model

However, there are several design choices for the algorithm, e.g. whether to use two separate feature encoders for the source and target domain, which loss function to use and whether to include the label classification in the adversarial process. These choices are visualized in Figure 4.2. The two methods we will experiment with are Adversarial Discriminative Domain Adaptation (ADDA) [1] and Wasserstein Distance Guided Representation Learning (WDGRL) [6], both use different design choices and optimization schemes.

4.1.1 Adversarial Discriminative Domain Adaptation

The Adversarial Discriminative Domain Adaptation (ADDA) framework was introduced by Tzeng et al. [1] and makes use of two separate encoders τ_s and τ_t for the source and target domain. The ADDA framework is optimized in three phases:

(i) First, a classification model is pre-trained on the source data and labels. The last FC layer in the classification model is referred to as the *classifier*,

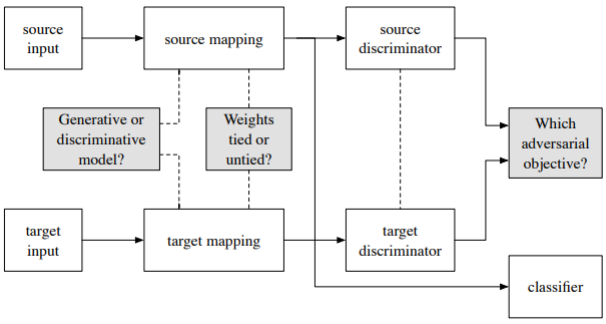

Figure 4.2: Design choices for adversarial domain adaptation [1]

while other layers are referred to as the *feature extractor* or *encoder*. The source classification model is optimized by means of the cross-entropy loss.

(ii) In the second phase, the target encoder is trained with both the source and target data. This is performed by using the source encoder from the pre-trained classification model and fixing its parameters. Meanwhile, the target encoder is initialized by the source parameters and trained such that the target feature representations are identical to the source representations. The target feature encoder opposes the discriminator, which aims to predict whether the sample originates from the source or target domain. During this adversarial procedure, the target feature encoder tries to maximize the BCE loss, while the discriminator tries to minimize this loss.

(iii) In the last phase, the pre-trained source classifier from phase 1 is put on top of the trained target encoder from phase 2, and a classification on the test set is performed to evaluate the classification performance on the target domain.

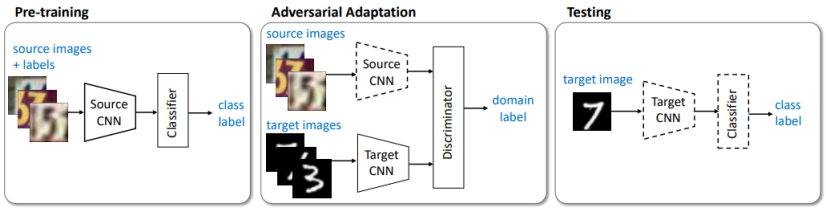

Figure 4.3: The ADDA optimization scheme [1]

Figure 4.3 shows the three phases of the optimization procedure. In phase 1, the source classification model is optimized with the cross-entropy loss, which is

defined as:

$$\min_{\tau^s, c} L^{ce,adda} = -\mathbb{E} \sum_{k=1}^{K} 1(k = y_s) \cdot \log c^s(\tau^s(x^s)) \tag{4.1}$$

In the above equation, c is the classifier network, τ^s is the source feature encoder, x^s is the source data, y_s is the class of the source sample and K is the total number of classes.

In phase 2, the discriminator classifies whether a feature encoding originates from the source or target domain and follows the standard binary cross-entropy (BCE) loss:

$$\min_{d} L^d = -\mathbb{E}[\log d(\tau^s(x^s))] - \mathbb{E}[\log(1 - d(\tau^t(x^t))] \tag{4.2}$$

Simultaneously, the feature encoder τ_t aims to learn a target representation such that the discriminator cannot separate the target samples from the source samples. As the target feature encoder is not able to influence the parameters of the source encoder, the target encoder loss consists of the term of the discriminator loss function which includes the target encoder:

$$\min_{\tau^t} L^{\tau_t} = -\mathbb{E}[\log d(\tau^t(x^t))] \tag{4.3}$$

Hence, the target encoder has a similar role as the generator in a general GAN optimization scheme. Instead of generating fake images that imitate the real images, the target encoder aims to generate target representations which are similar to source representations.

4.1.2 Wasserstein Distance Guided Representation Learning

The Wasserstein Distance Guided Representation Learning framework by Shen et al. [6] has two main differences from the ADDA framework. First of all, instead of using the BCE loss to measure the divergence between source and target domain, it makes use of the Wasserstein distance between the domains. Moreover, it does not learn two separate encoders, but trains one single encoder for both domains. However, using this symmetric mapping for the domains requires that the source classifier loss is also included in the feature encoder optimization. If the classifier would not be included, the feature encoder would learn representations which are not discriminative with respect to the classes.

Since the discriminator in the WDGRL is not a classifying discriminator but rather outputs a score, the discriminator will now be referred to as *critic*. Since the letter c is already used to represent the classifier, we will still refer to d as the critic. The critic's Wasserstein loss including gradient penalty is defined as:

$$\min_d L^d = \frac{1}{T}\sum_{x\in X^t} d(\tau^{s,t}(x^t)) - \frac{1}{S}\sum_{x\in X^s} d(\tau^{s,t}(x^s)) + \nu\frac{1}{E}\sum_{x\in X^s}(\parallel \nabla_e d\parallel_2 - 1)^2 (4.4)$$

In contrary, the feature encoder τ aims to maximize the above function. In addition, the cross-entropy loss over the source samples is included in the feature encoder loss. The cross-entropy loss is defined as:

$$\min_c L^{ce,wdgrl} = -\mathbb{E}\sum_{k=1}^{K} 1(k = y_s)\cdot \log c^{s,t}(\tau^{s,t}(x^s)) \tag{4.5}$$

Including the final classification of the source samples in the optimization of the feature encoder ensures that the feature representations can also be segregated by label. Note that in the ADDA framework this addition of the classification was not required, as the pre-trained source encoder already secured this discriminativeness. The gradient penalty is not included in the Wasserstein loss function, as the feature extractor is aimed at learning representations which are as similar as possible. The Wasserstein loss is weighted with a fixed balancing parameter λ. Eventually, this leads to:

$$\min_{\tau^{s,t},c} L^{ce,wdgrl} L^{\tau} = \lambda\left[\frac{1}{S}\sum_{x\in X^s} d(\tau^{s,t}(x^s)) - \frac{1}{T}\sum_{x\in X^t} d(\tau^{s,t}(x^t))\right] + L^{ce,wdgrl} (4.6)$$

4.2 Proposed method

4.2.1 Choice of optimization scheme

In our feature-based model, we propose of a combination of ADDA and WDGRL. We use the Wasserstein distance as a divergence metric (WDGRL) and we apply the classification loss in the feature loss (WDGRL). However, we employ two separate encoders for the source and target domain (ADDA). Hence, after pre-training the source encoder, we learn the target feature representations adversarially. As the source encoder is static during adversarial optimization, the classification loss in the feature loss is computed with the target encoder instead of the source encoder. The algorithm design choices of our method compared to the other methods is shown in Table 4.1.

Including the classification of the source samples in the adversarial optimization gives an adapted version of the optimized cross-entropy loss:

$$\min_{\tau^t,c} L^{ce,ours} = -\mathbb{E}\sum_{k=1}^{K} 1(k = y_s)\cdot \log c^{s}(\tau^{t}(x^s)) \tag{4.7}$$

Note that the only difference with the cross-entropy used in the WDGRL is that instead of using the single encoder $\tau^{s,t}$ and symmetric classifier $c^{s,t}$ to

model	feature mappings	divergence metric	source classification	instance reweighting
ADDA	2	BCE loss	not included	no
WGRL	1	wasserstein	general encoder + general classifier	no
ours	2	wasserstein	target encoder + source classifier	yes

Table 4.1: Design choices of the methods

compute the source classification loss, we use the target encoder τ^t and source classifier c^s. Our final loss function for the critic includes this weight for the source samples:

$$\min_c L^c = \frac{1}{n^t} \sum_{x \in X^t} c(\tau^t(x^t)) - \frac{1}{n^s} \sum_{x \in X^s} c(\tau(x^s)) + \nu \frac{1}{E} \sum_{x \in X^e} (\| \nabla_e c(e) \|_2 - 1)^2 \quad (4.8)$$

The loss function for the feature extractor is shown as:

$$\min_{\tau^t} L^\tau = -\frac{1}{n^t} \sum_{x \in X^t} c(\tau^t(x^t)) + L^{ce,ours} \quad (4.9)$$

4.2.2 Introduction of a weight network

Although domain adaptation has shown to work properly when the class distributions of source and target are similar, these methods might become unreliable under an imbalance. Under this class imbalance, there is a risk that the target feature encoder learns to map its samples to the wrong class. This is due to the divergence function of the feature encoder loss, which aims to make the two feature mapping distributions as similar as possible. Simultaneously, the class discriminativeness of the representations can only be taken into consideration for the source labels, as during adversarial learning we assume we do not have access to the target labels. Therefore, the model can start to overfit towards the source labels, as the class segregation of the target embeddings cannot be included in the optimization process.

A simple example is shown in Figure 4.4, where the source (blue) and target (green) data consists of 5 samples with labels 0 and 1. When the feature encoder learns a mapping of the data, the divergence metric will aim to align the mappings in this domain-invariant space. However, since we have a severe class imbalance, three out of the four target samples with label 1 will be mapped close to the zeros, in order to be aligned with the source mapping. For the source mapping this is not a problem, as the class discriminativeness can be taken into account during optimization.

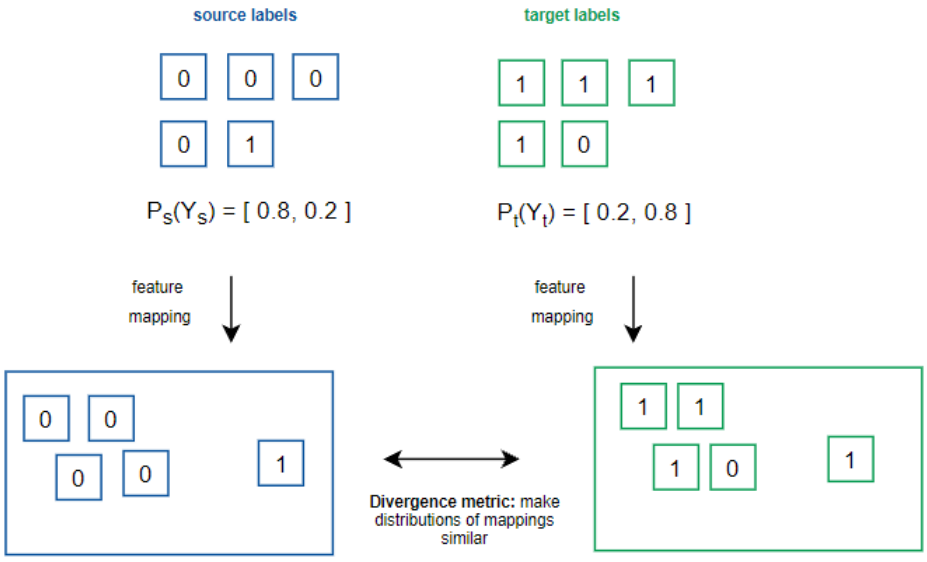

Figure 4.4: Wrong feature mapping example

In our proposed method, we introduce the weight network, which is able to align the distributions on an instance-based level. As shown in Experiment 2, the weight is able to equalize the class distributions of two datasets. It is also able to re-weight specific features within a class, as shown in Experiment 3: the weight learned a higher weight for the 4-digits with vertical lines, which was under-represented in the source dataset. The aim of the weight is to improve the mapping between the data and the feature embeddings when (features within) the classes are imbalanced.

The introduction of the weight into the framework leads to the following critic loss:

$$\min_c L^c = \frac{1}{T}\sum_{x\in X^t} c(\tau^t(x^t)) - \frac{1}{S}\sum_{x\in X^s} c(\tau(x^s))w(\tau^s(x^s)) \tag{4.10}$$

$$+\nu\frac{1}{E}\sum_{x\in X^e} (\| \nabla_e c(e)\|_2 - 1)^2 \tag{4.11}$$

The loss function for the feature extractor is shown as:

$$\min_{\tau^t} L^\tau = \lambda^{\tau^t}\left[-\frac{1}{T}\sum_{x\in X^t} c(\tau^t(x^t))\right] + L^{ce,ours} \tag{4.12}$$

The loss function for the weight network is shown as:

$$\min_{w} L^{w} = \lambda^{w} \left[\frac{1}{S} \sum_{x \in X^{s}} c(\tau^{s}(x^{s})) w(\tau^{s}(x^{s})) \right] + L^{ce,ours} \tag{4.13}$$

As shown in the Equations 4.12 and 4.13, the target feature encoder and weight network have a similar objective: transforming the target representations such that they resemble the source representations. However, there are several variations possible for this optimization scheme. We can choose whether or not to include the balancing parameter $\lambda^{\tau^{t}}$ and λ^{w}, in order to control the influence of the Wasserstein loss on the feature and weight optimization. Moreover, we can choose if we add the classification loss to the $L^{ce,ours}$ to the weight and/or feature loss. We will experiment with several of these configurations in Experiment 5.

However, adding an extra network to the adversarial game can cause stability issues, as three networks are optimized at the same time. This imposes an extra challenge. Therefore, we divide the experiments in two parts. First, we experiment with our proposed method, but without including the weight network. We compare the results with several state-of-the-art frameworks in order to test the efficiency of our method. Afterwards, we add the weight network to our proposed framework, and test this framework on an imbalanced version of the dataset. The complete framework is shown in Appendix D.

4.3 Datasets

4.3.1 Digit recognition

We use two datasets within the hand-written digit recognition field: MNIST [19] and USPS [20]. We construct two transfer tasks: MNIST → USPS and USPS → MNIST. Both datasets consists of 10 classes, from digit 0 to digit 9. MNIST consists of 60,000 train samples and 10,000 test samples and has a resolution of $28 \times 28 \times 1$, while USPS consists of 7,291 training images and 2,007 test images and a resolution of $16 \times 16 \times 1$. An example is shown in Figure 4.5.

Figure 4.5: MNIST and USPS dataset

4.3.2 Office-31 object recognition

The benchmark Office dataset [21] includes a total of 4,110 images with 31 categories. The classes are several objects which are prevalent in offices, e.g. laptops, rulers or keyboards. There are 3 available domains: webcam, amazon and dslr. Amazon consists of 2,955 samples and has images of size $300 \times 300 \times 3$; dslr contains 572 samples of resolution $1000 \times 1000 \times 3$; webcam has 796 samples of resolution $423 \times 423 \times 3$. Due to the small size of the datasets, it is common to train the domain adaptation methods transductively, which implies that the training is performed on the complete datasets. Example images are shown in Figure 4.6.

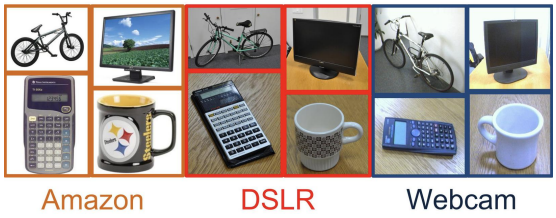

Figure 4.6: Office-31 datasets

4.4 Evaluation metrics

During training of a source classification model, the training set is split between train and validation set. Moreover, during adversarial training, the optimization is performed on the train set. For testing, the performance of the domain adaptation methods is evaluated by means of the accuracy on the test set of the target domain. Additionally, we can visualize the feature embeddings of the source and target domain, which shows us whether the representations per class are close to each other.

Chapter 5

Results for state-of-the-art comparison

As mentioned in the methodology section, we perform two types of experiments. First, we perform experiments where we aim to obtain the optimal unweighted framework (Experiment 4). Hence, in this part, we do not include the weight network. This is simply to check whether our adapted framework is able to give comparable results as the state-of-the-art. We will perform these tests on both the digit recognition datasets and Office-31 datasets. Afterwards, we introduce the weight network in the optimal unweighted network (Experiment 5). We test several configurations of the architecture and loss functions for MNIST-USPS. Lastly, we experiment with various levels of class imbalances between source and target domain, to check whether our weighted method improves the performance under severe class imbalance.

5.1 Architectures

Digit recognition. As the MNIST and USPS datasets consist of low-dimensional images, the feature extraction for ADDA and WDGRL is performed with the modified LeNet architecture [19]. The architecture is shown in Appendix B. The output of the feature encoder network is a 500-dimensional vector for ADDA. The classifier takes this 500-dimensional vector as an input and consists of 1 fully-connected (FC) layer. The critic/discriminator and weight network both consist of 3 FC layers with 500 neurons. The activation function is ReLU. The last layer is not followed by an activation function.

Office-31. The domain adaptation for the Office-31 datasets is performed differently for ADDA and WDGRL. The WDGRL makes use of Decaf features [28], which are 4096-dimensional feature vectors extracted by AlexNet. In contrary to the original data, which consists of 31 classes, these features capture

a subset of 10 classes. In contrary, ADDA uses the original images as an input. The ResNet-50 architecture [29] is deployed, requiring an input resized to $224 \times 224 \times 3$ pixels. A visualization of the architecture is shown in Appendix B. The feature encoder is the network up to and including conv4, proving a 2048-dimensional feature vector. The classifier is one FC layer. The parameters are initialized by the pre-trained weights. The critic and weight have a similar architecture as for the digit recognition experiments.

5.2 Experiment 4: unweighted method

5.2.1 Digit recognition

The first unweighted experiments are conducted for the MNIST $\rightarrow$ USPS and USPS $\rightarrow$ MNIST transfer tasks. We implement the ADDA, WDGRL and our own method and compare the results. Since the code for ADDA and WDGRL is written in Tensorflow, we decide to develop it ourselves in PyTorch.

For our own method, we use a batch size of 1024, gradient penalty weight ν of 20 and balancing parameter λ of 0.01. The optimizers is Adam with β_1 0.5 and β_2 of 0.999. The learning rates for both the feature extractor and the critic are set to 0.0001. For the critic, we used a similar architecture as used in ADDA, but added 1 extra FC layer with 500 neurons. For the implementation of the WDGRL, we used different parameters and architectures than the settings given by the authors. Eventually, we used the critic and feature extractor architectures from ADDA, as these showed better results than the original architectures. For the ADDA implementation, we did mostly use the predefined parameter settings. The results of our own implemented versions of the models is shown in Table 5.1. We observe that our method outperforms ADDA and WDGRL. However, it does not outperform the state-of-the-art. The DRAnet [11], published in 2021, shows the highest accuracy for MNIST $\rightarrow$ USPS. However, for this domain transfer task, our method comes second.

Method		$M \rightarrow U$	$U \rightarrow M$
	Source only	0.761	0.588
Previous work	DANN [7]	0.771	0.750
	DDC [16]	0.791	0.698
	ADDA [1]	0.896	0.909
	WDGRL [6]	-	-
	CoGAN [8]	0.912	0.899
	DRAnet [11]	**0.978**	**0.991**
This work	WDGRL^{ours}	0.920	0.898
	ADDA^{ours}	0.901	0.923
	ours	**0.958**	**0.936**

Table 5.1: Results for domain adaptation on MNIST-USPS transfer tasks

Figure 5.1 and Figure 5.2 shows the accuracy and loss function of the best model.

We can observe that the source accuracy is constant and stays close to 1, while the target accuracy shows a peak at approximately 1000 epochs. The plots with the losses shows that classifier loss is stable, while the target feature loss and critic loss are more divergent.

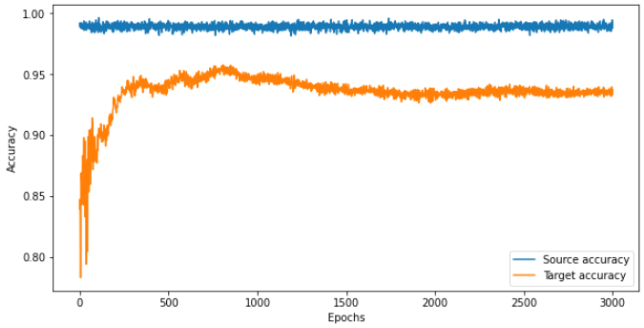

Figure 5.1: Course of accuracy over epochs

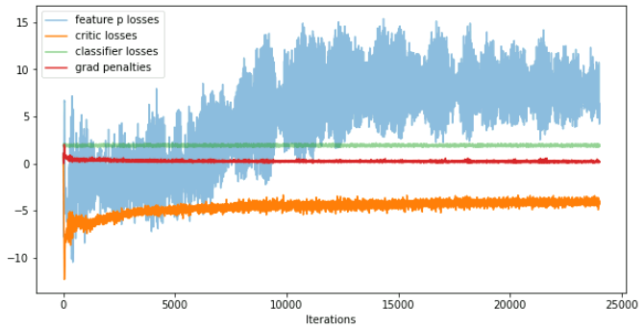

Figure 5.2: Course of losses over epochs

In order to gain a better understanding of the domain adaptation, we can visualize the feature vectors with t-SNE. Figure 5.4, 5.3 and 5.5 zoom in on the t-SNE of one specific class. It is interesting to note that although the WDGRL model provides better accuracies than ADDA, the feature vectors of ADDA shows the most domain-invariant representations. As our method is an integration of WDGRL and ADDA, we can also note that the level of domain-invariance is in-between ADDA and WDGRL: the embeddings are more consistent than WDGRL, but have more noise than ADDA. This pattern applies for every class: the WDGRL method learns target feature vectors which are quite far from the source vectors, but somehow still sufficiently discriminative and domain-invariant. This difference could explained by the different optimization schemes: ADDA uses pre-trained source feature representations, and forces the target embeddings to imitate these. In contrary, in the WDGRL, the feature loss does not only consist of the divergence between the distributions, but is also defined by the source classification. Due to the balancing parameter λ, the classification loss is

more dominant than the domain-invariance. Appendix C shows the complete t-SNEs from the models.

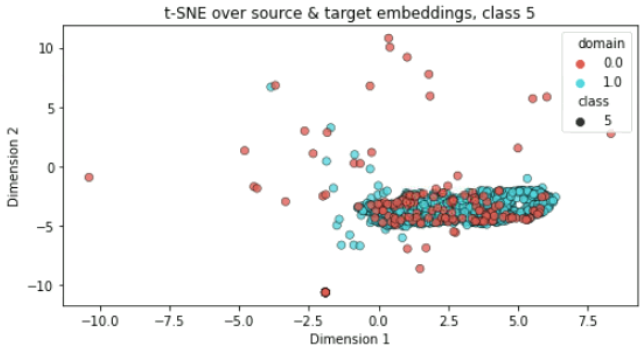

Figure 5.3: Example feature representation for ADDA

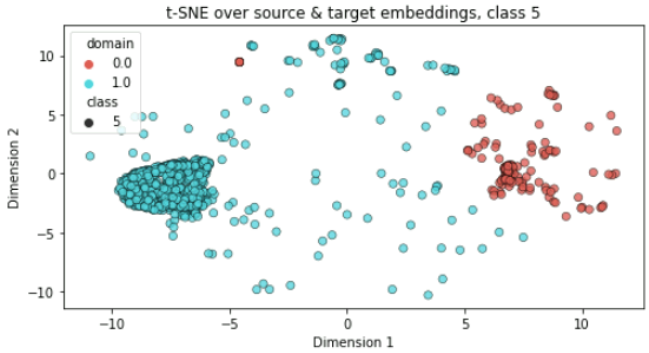

Figure 5.4: Example feature representation for WDGRL

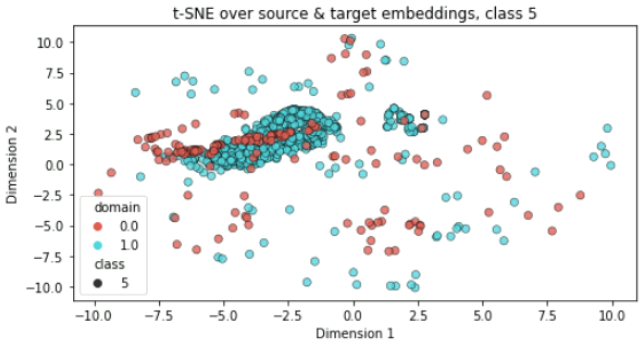

Figure 5.5: Example feature representation for our method

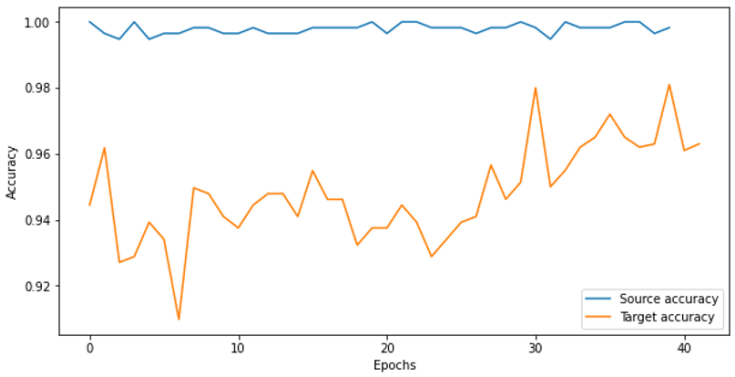

Figure 5.6: Accuracy over D → W transfer

5.2.2 Office-31

We also experimented with our method on the Office-31 datasets. As mentioned in the methodology section, ADDA makes use of the ResNet-50 architecture as a backbone. In contrary, the WDGRL makes use of Decaf features, which implies that the feature extractor does not require a complex architecture. However, these features are pre-trained on the target labels and contain a subset of the dataset. Therefore, we decide to use the ResNet-50 network for the feature extraction, consistent with other recent works [30] [18]. For our own ADDA implementation, we initialize the parameters with the pre-trained source parameters, but re-train the complete network during adversarial training. This is different from the original ADDA framework, where the authors state that they freeze the layers up to conv4. However, this did not work in our case, and thus we decided to re-train the complete backbone. Moreover, for our own model, the batch size is set to 16, critic learning rate of 0.0001 and feature learning rate of 0.005 optimizer Adam, and a penalty weight of 20.

Method		$A \rightarrow W$	$D \rightarrow W$	$W \rightarrow A$
	Source only	0.626	0.961	0.610
Previous work	DANN [7]	0.730	0.964	0.675
	ADDA [1]	0.751	0.970	-
	DDC [16]	0.618	0.950	-
	WDGRL [6]	0.895	0.979	0.937
	DADA [18]	**0.924**	**0.993**	**0.743**
This work	ADDA^{ours}	0.765	0.972	0.697
	ours	**0.803**	**0.980**	**0.709**

Table 5.2: Results for domain adaptation on Office-31 transfer tasks

Table 5.2 shows the results for three transfer tasks: $A \rightarrow W$, $D \rightarrow W$ and $W \rightarrow A$. The WDGRL is also shown in the table, although the results are not completely comparable due to the usage of a subset of pre-trained features.

Moreover, in our own implementations we did not include our results for WDGRL, as it did not converge. The results for $D \rightarrow W$ and $W \rightarrow A$ for the DDC [16] and ADDA [16] framework are not published, hence are not included in the table. The table shows that our method improves the implementation of ADDA as well as the official implementation of ADDA. However, the DADA framework [18] still has higher performance for every transfer task.

Figure 5.6 shows the accuracy from our best model, from dslr to webcam. We can observe that the target accuracy already starts above 94%, as the source-only model already gives an accuracy of 96.1%. The accuracy seems to slightly increase towards 98%. Nonetheless, Figure 5.7 and 5.8 show the the t-SNE results before and after domain adaptation. It is noticeable that even though the source-only and adaptation accuracy are relatively close, there is an enormous improvement in class segregation after adaptation. Before adaptation, the embeddings of the different classes were scattered in the space, while afterwards the classes are easily separated.

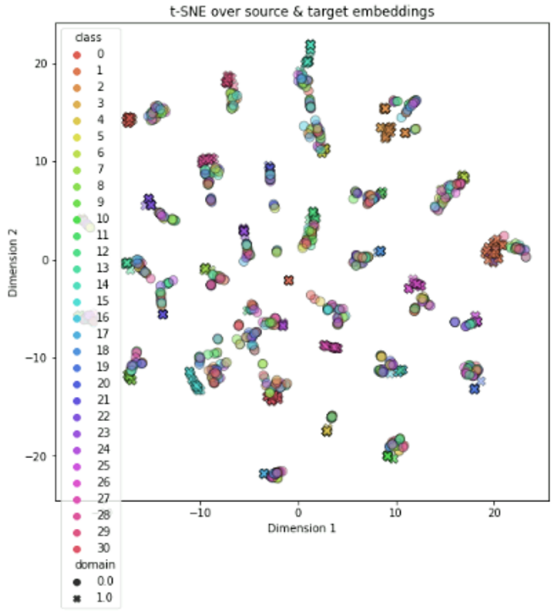

Figure 5.7: Feature representations with source-only

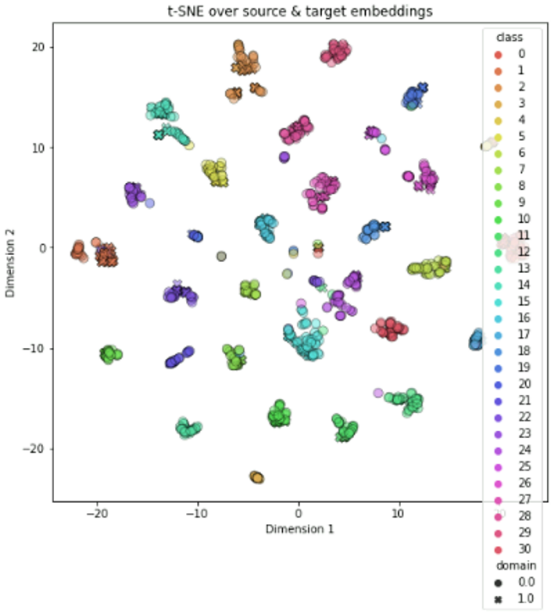

Figure 5.8: Feature representations after our method

5.3 Integration of weight function

5.3.1 Experiment 5: combining feature-based adaptation with the weight

In Experiment 4, we aimed at finding the optimal feature-based baseline model. In this section, the focus lies on analyzing whether an integration of the feature-based framework and sample-based weight is suitable.

Adding an additional network to the framework implies that there is a large choice of parameters and loss functions. First of all, we can choose whether we want to add the source classification loss $L^{ce,ours}$ to the weight and/or feature loss. Adding this classification to the loss enforces learning weights or transformations which enable class segregation. However, it might be the case that this addition is unnecessary, or only necessary for one loss function. Another choice is whether to include the learned weights in the computation of the cross-entropy loss for classification. Including weights in the classification loss could form an extra enforcement for class segregation.

In the unweighted framework in the previous section, we used a balancing parameter λ to balance the influence of the Wasserstein distance in the feature loss. We can also choose to add this parameter to the feature or weight loss. We have experimented with λ-values of 0.5, 0.1, and 0.01.

Also, we can choose what activation function the last layer of the weight includes. In order to enforce non-negativity, we experiment with two types of activation functions: ReLU and Softmax. During adversarial training, the output of the softmax is multiplied by the batch size, such that the sum of the weights sum up to the number of samples. In this way, the transformed source distribution is a valid probability distribution, as explained in Section 2.2.4.

A sample of results for several types of experiments with the above mentioned choices is shown in Table 5.3. The column λ_τ and λ_w indicate whether or not a balancing parameter is used for balancing the Wasserstein distance in the total loss of the feature respectively weight loss. The columns $L^{ce,ours}$ *for* τ and $L^{ce,ours}$ *for* w indicate whether the classification loss is included in the feature or weight loss function. The column w *in* $L^{ce,ours}$ defines whether the computed weights are included in the computation of the classification loss, while *activation weight* defines which activation function is used. Also, w_{pass} indicates whether we skip updating the weight network for a few epochs. In our case, we used 5 epochs.

The table shows that using a softmax leads to better performance than the usage of ReLU. Also, including the classification loss for both the weight and feature loss slightly increases the performance. Not using a balancing parameter λ, as in trial 4, lowers the performance. Also, updating the weight every 5 epochs improves the performance.

Method	λ_τ	λ_w	$L^{ce,ours}$ for τ	$L^{ce,ours}$ for w	w in $L^{ce,ours}$	w_{pass}	activation	acc_{target}
Source-only								0.761
Unweighted$_{ours}$								0.958
trial 1	✓	✓	✓	×	✓	×	softmax	0.834
trial 2	✓	×	✓	×	✓	×	softmax	0.818
trial 3	×	×	✓	✓	✓	×	softmax	0.792
trial 4	✓	×	✓	✓	✓	×	relu	0.801
trial 5	✓	×	✓	×	✓	×	relu	0.799
trial 6	✓	✓	✓	✓	✓	×	softmax	**0.842**
trial 7	✓	✓	✓	✓	✓	✓	softmax	**0.862**

Table 5.3: Results for weighted domain adaptation on MNIST-USPS transfer

Nonetheless, these differences are relatively small. Although all trials are able to improve the domain transfer performance, the addition of a weight function does not improve the unweighted framework used in the previous section.

In Experiment 3 of the proof-of-concept, we have observed that a weight network is able to detect differences in features such a rotation, as shown on Figure 3.23 and 3.24 in Section 3.2.2. However, it might be possible that the feature encoder is already capable op capturing these transformations, and thus the usage of both a weight network and feature encoder might be redundant.

5.3.2 Experiment 6: weighted method for imbalanced adaptation

Nonetheless, it can be noted that the class distributions of the source and target dataset of both MNIST-USPS and Office-31 are mostly balanced. In real-world cases, this balance between source and target is not guaranteed. As explained in Section 4.2.2, we suppose an imbalance between the two domains forces the target feature representations to be mapped to the wrong class. The results from Experiment 2 in Section 3.2.1 showed that the weight is also able to improve the class balance between the two distributions. Hence, we set up an additional experiment on the MNIST $\rightarrow$ USPS transfer, where we test whether the weight function is able to improve the performance under a class imbalance. We create class imbalances of three levels, where the class imbalance is expressed with the Kullback-Leibner divergence between the two class distributions, consistent with previous research [31].

The original class distributions from MNIST and USPS have a KL-divergence of 0.04. We create two imbalanced versions, having a KL-divergence of 0.40 and 0.74. The original distribution and the imbalanced distribution are shown in Figure 5.9, 5.10 and 5.11. For these experiments, we use a critic architecture with 3 layers instead of 4 layers, which is slightly different from Experiment 4, as Experiment 5 has shown that this architecture provides higher results.

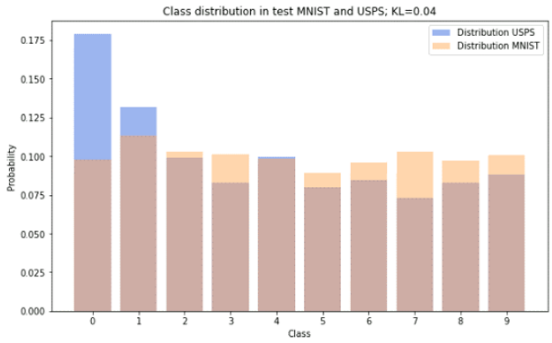

Figure 5.9: Class distribution original MNIST-USPS

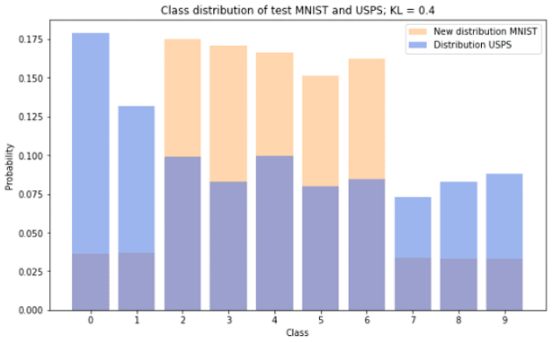

Figure 5.10: Class distribution imbalanced MNIST-USPS

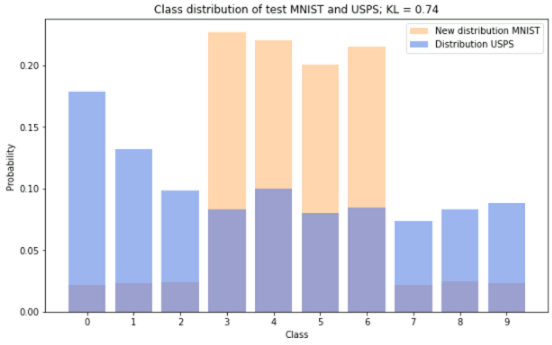

Figure 5.11: Class distribution imbalanced MNIST-USPS

Aligned with the results from Table 5.3, we pass updating the weight network for 4 epochs, and update the weight every 5th epoch. We use a batch size of

1024, balancing parameter λ of 0.5, β_1 of 0.5 and β_2 of 0.999, penalty weight ν of 50, and learning rate for all networks is 0.001. The results are shown in Table 5.4. Note that the accuracy for the unweighted model with with a KL of 0.04 is lower than our optimal accuracy, as we used different parameter settings. The table shows that for a negligible class imbalance of 0.04, the weighted model does not outperform the unweighted model. The same applies for a severe class imbalance of 0.74. However, for a class imbalance of 0.40, the weighted model does outperform the unweighted model with more than 10%.

Method		$M \rightarrow U$	
KL(S,T)	0.04	0.40	0.74
Unweighted	0.872	0.609	0.640
Weighted	0.865	**0.714**	0.643

Table 5.4: Results for weighted domain adaptation on imbalanced MNIST-USPS transfer

Figure 5.12 and 5.13 show the accuracy of the the unweighted versus weighted version of the model with a KL-divergence of 0.40. We can observe that the unweighted method is unstable and has an optimum at around 0.60. In contrary, the weighted method appears more stable and grows to above 0.70.

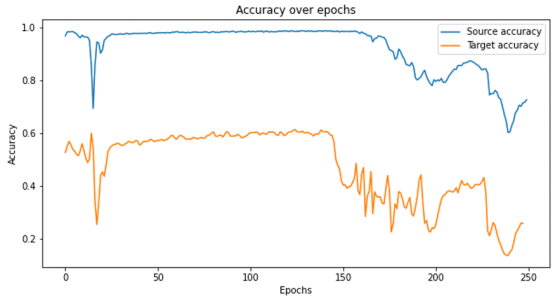

Figure 5.12: Accuracy for unweighted method

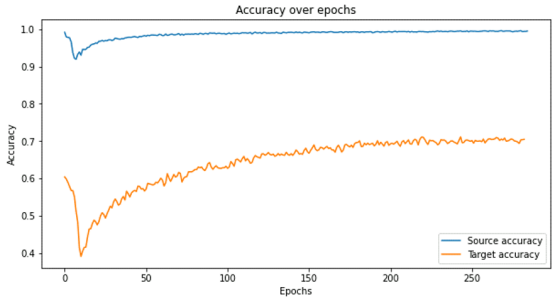

Figure 5.13: Accuracy for weighted method

Chapter 6

Discussion

This research was focused on investigating whether an adversarial instance re-weighting framework, inspired by the Wasserstein GAN, is able to enhance the unsupervised domain adaptation problem. This problem occurs when there is a shift between the source and target distribution, leading to a decrease in prediction performance on the unlabeled target dataset. We introduced the addition of a weight network as a sample-based transformation of the source samples, such that the source distribution matches the target distribution.

This work was divided into two parts. The first part (Experiment 1—3) consisted of several proof-of-concept experiments, where we tested whether the learning of a weight function is able to align two different distributions in controlled environments. The second part (Experiment 4—6) addressed an integration of the weight function with existing feature-based methods, in order to compare the method with the state-of-the-art.

Experiment 1 considered two one-dimensional Gaussian distributions with slightly different means. The results showed that the weight is able to transform the source distribution into the target distribution, as well as converging to the hypothesized critic output. However, we noted an important limitation: the method requires that the features in the target domain also exist in the source domain. If this requirement is not met, the weight has to learn a weight for features which do not exist, and is thus unable to align the distributions.

Experiment 2 and Experiment 3 dealt with a modified version of the MNIST dataset. In Experiment 2, we created two small subsets with only 0-digits and 1-digits. We sampled two classes from the dataset and created a class imbalance between the domains. The results showed that the weight function is capable of learning the expected weights, such that a balanced dataset emerges. Experiment 3 dealt with an actual fabricated covariate shift within the MNIST dataset for class 0 to 4. The data was pre-processed with the LeNet architecture, in order to extract feature vectors from the samples. Afterwards, KMeans clustering was

performed on the vectors, such that every class was subdivided into two groups. From these two groups within a class, one generally consisted of a rotated version of digits compared to the other group. In two out of the five classes, an imbalance within the class was created by means of the subclasses. The results of the experiments showed that the weight recognized the shift in features and was able to correct for them. However, we note that the shift between the two domains is rather small; as the two domains are extracted from the same dataset, there is not a large difference between the features. Hence, there is still a lot of overlap between the features, similar to Experiment 1. In real-world applications, we do not have the guarantee that this is the case. Moreover, if the domain shift is this small, there might not be a need for domain adaptation, as mostly the source-only network is able to lead to high performance.

In order to make the model applicable to larger domain shifts, we aimed at combining the weight with a feature-based transformation. Before integrating the weight network with a feature-based framework, in Experiment 4 we aimed at finding the optimal unweighted feature-based adaptation technique as a baseline method. We tested these methods on two benchmark datasets, the object recognition datasets (MNIST-USPS) and the Office-31 object recognition datasets. We experimented with two current methods, WDGRL [6] and ADDA [1]. We observed that ADDA has the advantage of delivering domain-invariant feature embeddings, as well as being stable due to the support of the pre-trained source embeddings. In contrary, the WDGRL has the advantage of using the Wasserstein distance (analogous to our method) and including the classification into the adversarial training, which generally leads to high performance. The combination of both methods showed to lead to higher performance than the original methods, as well as preserving the domain-invariance of the target representations. For the MNIST-USPS transfer tasks, we observe that our combined method leads to a target accuracy of 95.8% on the MNIST $\rightarrow$ USPS transfer, and 93.6% on the USPS $\rightarrow$ MNIST transfer. This improves both our own implementations of WDGRL and ADDA. However, both do not beat the state-of-the-art, as the DRAnet [11], published in 2021, achieved an accuracy of 97.8% respectively 99.1%. For the Office-31 adaptation tasks, our method improved ADDA, but also did not outperform the DADA framework [18].

In Experiment 5, it was tested whether the addition of a weight function enhanced the performance on the MNIST $\rightarrow$ USPS transfer task. This extension leads to a large choice space for the architectures and loss functions. Experimenting with several configurations showed that all of the compositions led to a target accuracy increase between 3-10% compared to the source-only, but not beating the optimal unweighted method. We state that the addition of the weight might be redundant when we also perform a feature transformation, as the latter might already address sample-wise conversions. Moreover, due to the large choice space in loss functions, architectures and parameters, it might be more time-consuming to arrive at the optimal solution.

However, we reckon that when there is a class imbalance, the weight addition

might benefit the feature encoding, as it could avoid that the encoder is forced to map the features to the wrong class. Therefore, we perform another set of experiments (Experiment 6), where we create a class imbalance between the source and target samples, and measure this imbalance with the KL-divergence. We observe that for a balanced case (KL-divergence of 0.04), the weighted method does not improve the unweighted method. However, for an imbalanced case (KL-divergence of 0.40), the weighted method increases the target accuracy from 60.9% to 71.4%, and also shows more stable optimization than the unweighted method. However, in a very high imbalance, the unweighted and weighted method seem to perform identically. This could be due to the feature encoder not being able to learn accurate feature representations, and thus also influencing the functioning of the weight, as it is an adversarial process.

There were several limitations during this work. First of all, unsupervised domain adaptation is a highly competitive field. As our computational power is limited, it is time-consuming to run a lot of model configurations, and thus achieve the highest possible accuracy. Meanwhile, the the weight function was initially not designed with the motive to combat the large domain shifts present in state-of-the-art domain adaptation literature. Therefore, it was challenging to develop a framework which compares with current methods. Moreover, the integration of the weighted method with a feature-based technique led to an abundance of design choices, which was also time-consuming.

For future work, we suppose further examination in which ways the weight function is able to enhance imbalanced domain adaptation. We presume that the weight network can be adapted to estimate the class distribution rather than the sample distribution, and in this way learn a class-based instead of instance-based weight. Although the RAAN framework [13] has worked with a similar concept, this weight is not learned adversarially and in their work it is unclear what the effect is of this weight. Additionally, the experiments could be conducted on the VisDA-2017 dataset [32], although this dataset consists of more than 200,000 RGB images and is thus extremely expensive to train.

Chapter 7

Conclusion

The first research question denotes: *'Can we use non-generative adversarial networks, inspired by the Wasserstein GAN, to improve covariate shift adaptation?'*. This question consists of two subquestions: *'How can we train a weight function, which converts source into target distribution?* and *'What is the effect of the critic?'*. The results of the proof-of-concept demonstrated that the weight alone is able to match the distributions when there is a lot of overlap between the features. For larger domain shifts, the weight must be combined with a feature-based transformation. The integration between the weight and feature encoder is capable to advance the adaptation compared to the source-only, but is not able to improve the state-of-the-art. However, in imbalanced domain adaptation, the weight and feature integration might be advantageous compared to only feature-based methods. The critic is able to oppose the weight and feature extractor to learn these domain-invariant representations.

Another question states *'How can we measure the performance of the reconstructed target distribution and the adversarial network?'* Generally, domain adaptation techniques measure the performance with applying the adapted target network on the target data and observe its accuracy. However, in the controlled experiments 1—3, the shift in features was very small. Therefore, a source-only model applied on the target data would also give high accuracy performance, and thus this would not be a good metric. Hence, for the proof-of-concept experiments, we measure the domain-invariance with a binary domain classifier as well as a visualization of the weights with t-SNE. In contrary, in Experiment 4—6, where the domain shift was large, we did use the target accuracy as the evaluation metric.

The last question is *'Are there other ways than a weight function to express source data into target data?'* We showed that feature-based adaptation, which learns a domain-invariance feature encoding of the target distribution, is better at large domain shifts that the weight network.

In short, we have showed that the weight function enhances the distribution shift in small adaptation problems, while solely feature-based methods are better at handling larger domain shifts. However, when there is a severe imbalance between the two distributions, the integration of a weight function and a feature-based technique could outperform the feature-based technique.

Bibliography

[1] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7167–7176.

[2] D. Dua and C. Graff. (2017). "Uci machine learning repository," [Online]. Available: `http://archive.ics.uci.edu/ml` (visited on 10/09/2021).

[3] W. M. Kouw and M. Loog, "A review of domain adaptation without target labels," *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[4] M. Sugiyama, S. Nakajima, H. Kashima, P. Von Buenau, and M. Kawanabe, "Direct importance estimation with model selection and its application to covariate shift adaptation.," in *NIPS*, Citeseer, vol. 7, 2007, pp. 1433–1440.

[5] M. Wang and W. Deng, "Deep visual domain adaptation: A survey," *Neurocomputing*, vol. 312, pp. 135–153, 2018.

[6] J. Shen, Y. Qu, W. Zhang, and Y. Yu, "Wasserstein distance guided representation learning for domain adaptation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[7] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *The journal of machine learning research*, vol. 17, no. 1, pp. 2096–2030, 2016.

[8] M.-Y. Liu and O. Tuzel, "Coupled generative adversarial networks," *Advances in neural information processing systems*, vol. 29, pp. 469–477, 2016.

[9] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[10] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan, "Unsupervised pixel-level domain adaptation with generative adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3722–3731.

[11] S. Lee, S. Cho, and S. Im, "Dranet: Disentangling representation and adaptation networks for unsupervised cross-domain adaptation," in *Pro-*

ceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 15 252–15 261.

[12] R. Gopalan, R. Li, V. M. Patel, and R. Chellappa, "Domain adaptation for visual recognition," *Foundations and Trends® in Computer Graphics and Vision*, vol. 8, no. 4, pp. 285–378, 2015.

[13] Q. Chen, Y. Liu, Z. Wang, I. Wassell, and K. Chetty, "Re-weighted adversarial adaptation network for unsupervised domain adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7976–7985.

[14] B. Sun and K. Saenko, "Deep coral: Correlation alignment for deep domain adaptation," in *European conference on computer vision*, Springer, 2016, pp. 443–450.

[15] A. Rozantsev, M. Salzmann, and P. Fua, "Beyond sharing weights for deep domain adaptation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 4, pp. 801–814, 2018.

[16] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell, "Deep domain confusion: Maximizing for domain invariance," *arXiv preprint arXiv:1412.3474*, 2014.

[17] E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko, "Simultaneous deep transfer across domains and tasks," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4068–4076.

[18] H. Tang and K. Jia, "Discriminative adversarial domain adaptation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 5940–5947.

[19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[20] J. S. Denker, W. Gardner, H. P. Graf, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, H. S. Baird, and I. Guyon, "Neural network recognizer for hand-written zip code digits," in *Advances in neural information processing systems*, Citeseer, 1989, pp. 323–331.

[21] K. Saenko, B. Kulis, M. Fritz, and T. Darrell, "Adapting visual category models to new domains," in *European conference on computer vision*, Springer, 2010, pp. 213–226.

[22] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[23] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," *arXiv preprint arXiv:1701.04862*, 2017.

[24] C. Villani, *Optimal transport: old and new*. Springer, 2009, vol. 338.

[25] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in neural information processing systems*, 2017, pp. 5767–5777.

[26] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 675–678.

[27] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, no. 11, 2008.

[28] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *International conference on machine learning*, PMLR, 2014, pp. 647–655.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[30] Y. Zhang, H. Tang, K. Jia, and M. Tan, "Domain-symmetric networks for adversarial domain adaptation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5031–5040.

[31] B. Chidlovskii, "Using latent codes for class imbalance problem in unsupervised domain adaptation," *arXiv preprint arXiv:1909.08962*, 2019.

[32] X. Peng, B. Usman, N. Kaushik, J. Hoffman, D. Wang, and K. Saenko, "Visda: The visual domain adaptation challenge," *arXiv preprint arXiv:1710.06924*, 2017.

[33] M. H. Yap, G. Pons, J. Marti, S. Ganau, M. Sentís, R. Zwiggelaar, A. Davison, and R. Martí, "Automated breast ultrasound lesions detection using convolutional neural networks," *IEEE Journal of Biomedical and Health Informatics*, vol. PP, pp. 1–1, Jul. 2017. DOI: 10.1109/JBHI.2017.2731873.

[34] S. Wang, X. Xia, L. Ye, and B. Yang, "Automatic detection and classification of steel surface defect using deep convolutional neural networks," *Metals*, vol. 11, p. 388, Feb. 2021. DOI: 10.3390/met11030388.

Appendix A

Modified MNIST dataset

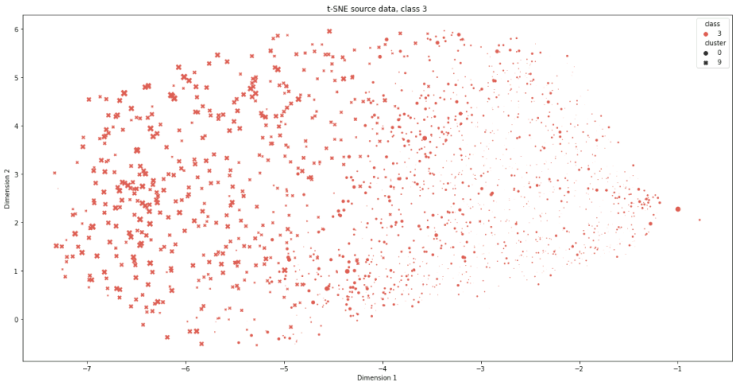

Figure A.0.1: t-SNE class 3 (Experiment 3)

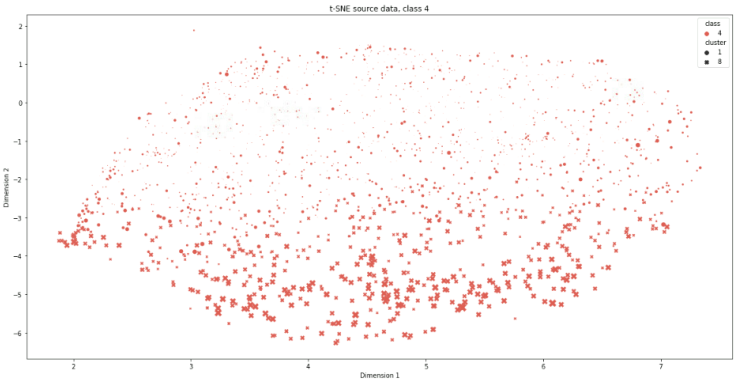

Figure A.0.2: t-SNE class 4 (Experiment 3)

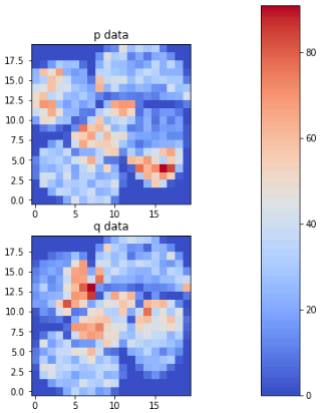

Figure A.0.3: Unweighred 2D histogram Experiment 3

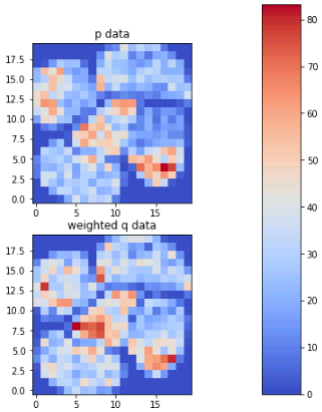

Figure A.0.4: Weighted 2D histogram Experiment 3

Appendix B

Network architectures

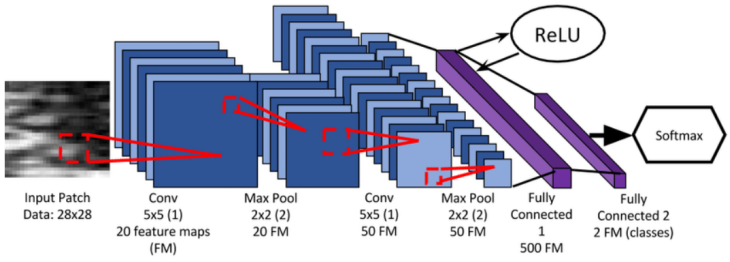

Figure B.0.1: Modified LeNet architecture [33] (Digit recognition)

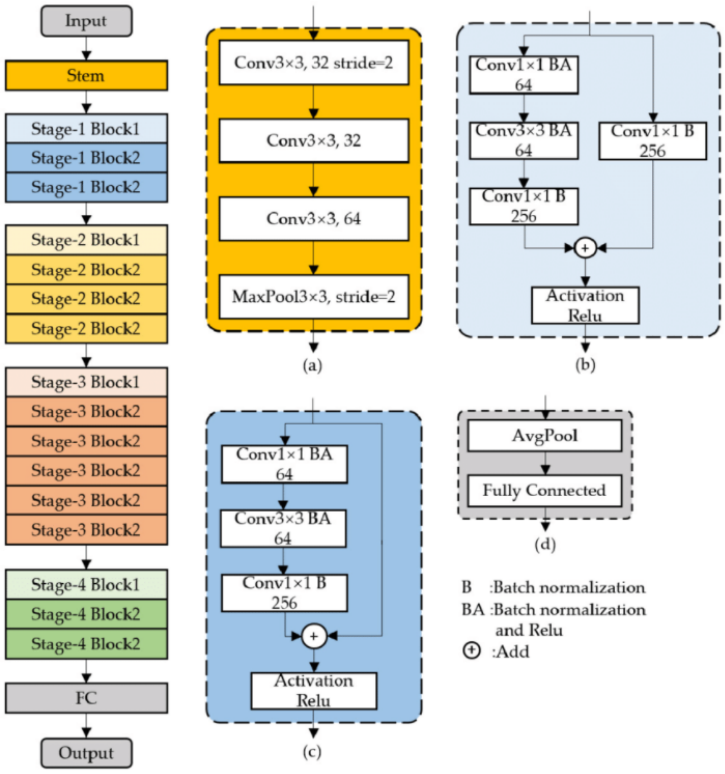

Figure B.0.2: ResNet-50 architecture [34] (Office-31)

Appendix C

t-SNE MNIST-USPS

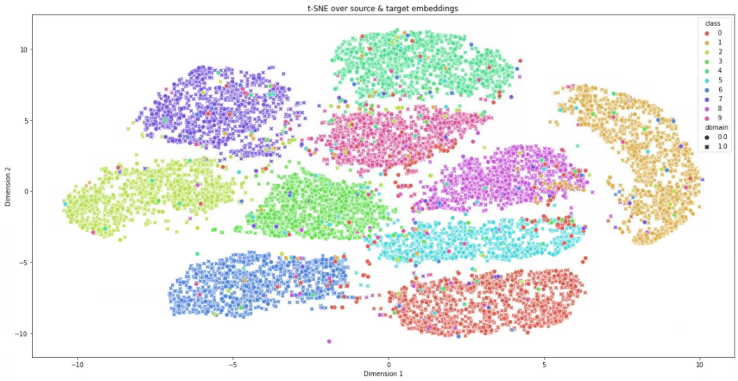

Figure C.0.1: t-SNE MNIST to USPS for ADDA

Figure C.0.2: t-SNE MNIST to USPS for WDGRL

Figure C.0.3: t-SNE MNIST to USPS for our method

Appendix D

Complete framework

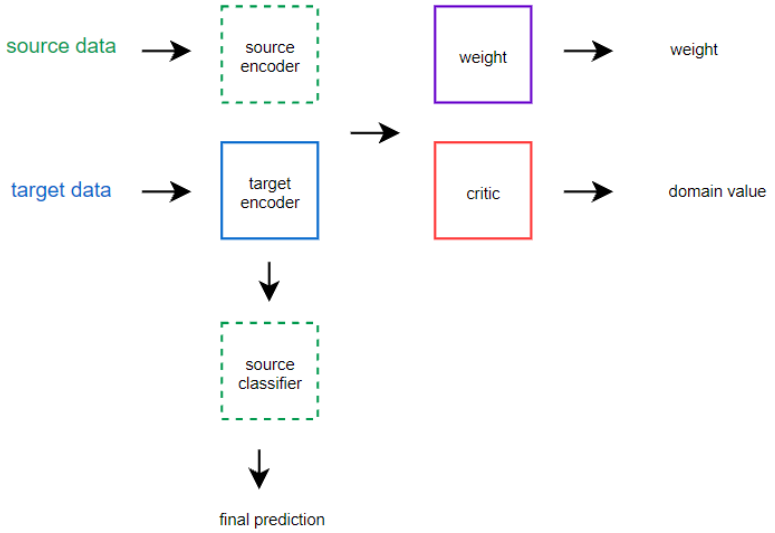

Figure D.0.1: Weighted, feature-based model