Enlarging Discriminative Power by Adding an Extra Class in Unsupervised Domain Adaptation

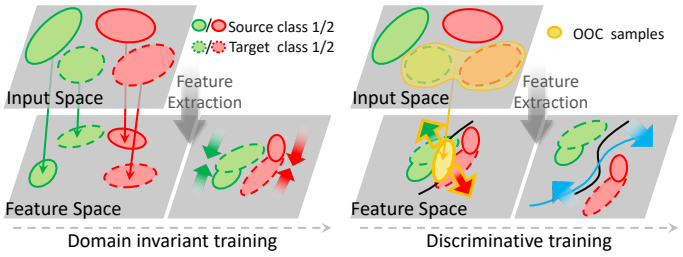
Hai H. Tran*, Sumyeong Ahn†, Taeyoung Lee‡ and Yung Yi§
School of Electrical Engineering
Korea Advanced Institute of Science and Technology, Daejeon, Republic of Korea
Email: haihongtran@kaist.ac.kr*, sumyeongahn@kaist.ac.kr†, ty.lee@kaist.ac.kr‡, yiyung@kaist.edu§

***Abstract*—We study the problem of unsupervised domain adaptation that aims at obtaining a prediction model for the target domain using labeled data from the source domain and unlabeled data from the target domain. There exists an array of recent research based on the idea of extracting features that are not only invariant for both domains but also provide high discriminative power for the target domain. In this paper, we propose an idea of improving the discriminativeness: Adding an extra artificial class and training the model on the given data together with the GAN-generated samples of the new class. The trained model based on the new class samples is capable of extracting the features that are more discriminative by repositioning data of current classes in the target domain and therefore increasing the distances among the target clusters in the feature space. Our idea is highly generic so that it is compatible with many existing methods such as DANN, VADA, and DIRT-T. We conduct various experiments for the standard data commonly used for the evaluation of unsupervised domain adaptations and demonstrate that our algorithm achieves the SOTA performance for many scenarios.**

I. Introduction

Deep neural networks have recently been used as a major way of achieving superb performance on various machine learning tasks, e.g., image classification [1], image generation [2], and speech recognition [3], just to name a few. However, it still leaves much to be desired when a network trained on a dataset from a specific data source is used for dataset from another data source. This domain shift and thus distribution mismatch frequently occurs in practice, and has been studied in the area of domain adaptation. The crucial ingredient in domain adaptation lies in transferring the knowledge from the source domain to the model used in the target domain.

In this paper, we consider the classification problem of *unsupervised* domain adaptation, where the trained model has no access to any label from the target domain. What a good domain adapation model has to have is two-fold. First, it is able to extract domain-invariant features that are present in both source and target domains, thereby aligning the feature space distributions between two different domains, e.g., [4], [5], [6], [7], [8], [9], [10]. Second, it has to have high discriminative power for the target domain task, which becomes possible by smartly mixing the following two operations: (i) extracting task-specific, discriminative features [11], [12], [13] and (ii) calibrating the extracted feature space so as to have a clearer separation among classes, e.g., moving the decision boundaries [14] (see Section II for more details).



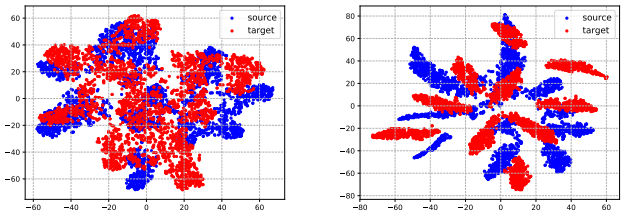
(a) Domain-invariant feature extraction

(b) Discriminative feature extraction: $(K+1)^{\text{th}}$ class and OOC samples.

Fig. 1: Illustration on how GADA works. Each arrow in the feature space corresponds to the force that moves the extracted features or tweaks the decision boundary. (a) Domain-invariant features are learned by dragging source samples and target samples to each other in the feature space. (b) Discriminative features are extracted by classifying out-of-class (OOC) samples into the extra class $(K+1)^{\text{th}}$, which creates an OOC cluster in-between the real clusters. This OOC cluster increases the distance between real clusters, thereby improving the quality of the extracted features in terms of discriminativeness.

Despite recent advances in unsupervised domain adaptation, there still exists non-negligible performance gap between domain adapted classifiers and fully-supervised classifiers, hinting a room for further improvement. In this paper, we focus on the second part of empowering the predictive model with more discriminativeness, whose key idea is as follows: Assuming that there are K classes in the target data, we equip the model with an extra $(K+1)^{\text{th}}$ class. This extra class is constructed so as to contain the target-like samples, which we call out-of-class (OOC) samples throughout this paper, that fail to belong to any of K classes. Feeding such OOC samples and classifying them into its own $(K+1)^{\text{th}}$ class create an OOC cluster in-between the real clusters, thereby increasing the distances among these real clusters and improving the extraction power in terms of discrimination. Figure 1 illustrates our idea, where to obtain the OOC samples, we train a generator based on a feature matching GAN [15]. We call our idea Generative Adversarial Domain Adaptation (GADA).

The power of an extra class has already been verified in the area of semi-supervised learning [15], [16], [17]. Our contribution is to utilize this idea in conjunction with a combination of different objective functions and techniques to unsupervised domain adaptation problem to achieve higher performance. To the best of our knowledge, this paper is the first to successfully integrate the idea of adding an extra class with unsupervised



(a) DANN [9] (74.9%) (b) GADA* (99.0%)

Fig. 2: Feature space comparison for the domain adaptation task SVHN → MNIST. The number in parenthesis corresponds to the classification accuracy.

domain adaptation. We comment that, compared to the case of semi-supervised learning, it is necessary to learn both the domain-invariant and the discriminative features, requiring to strike a good balance between those two for successful adaptation.

We highlight that our method is highly generic so as to be compatible with many existing methods. Figure 2 shows the feature space illustration, demonstrating the power of GADA when used together with the notorious method DANN [9]. As Figure 2 shows, we achieve a significant improvement in terms of accuracy and separability among the classes. We also show our integration power with two recent methods, VADA and DIRT-T [14], which improve the model's discriminative power. VADA aims to extract better discriminative features by employing smart loss functions in training, whereas DIRT-T refines the decision boundary for given extracted features. As shown later in Section IV, we achieve the best performance in the most difficult task MNIST → SVHN after the integration. This implies that (i) simply adding a new, fictitious class and training with generated samples as in GADA outperforms the VADA algorithm, and (ii) our idea is significantly synergic with a refining-based method DIRT-T.

We empirically prove the effects of our method by carrying out an extensive set of experiments where we observe that our method outperforms all other state-of-the-art methods in five among six standard domain adaptation tasks in different scenarios. Although the task SVHN → MNIST had a very high accuracy achieved by the existing methods, GADA is demonstrated to surpass all of them. As for MNIST → SVHN, which is known to be extremely challenging, our algorithm yields an improvement of 13% in terms of accuracy over VADA, setting a new state-of-the-art benchmark.

II. Related Work

Extracing domain-invariant features A collection of works [4], [5], [6], [7] aimed at aligning the feature space distributions of the source and target domains by minimizing the statistical discrepancy between their two distributions using different metrics. In [4], [5], maximum mean discrepancy (MMD) was used to align the high layer feature space. In [6], Joint MMD (JMMD) was used by defining the distance between the joint distributions of feature space for each layer one by one. In [7], the covariances of feature space were used as the discrepancy to be minimized. Different approaches include [8] and [18]. The authors in [18] proposed a method of minimizing the regularization loss between the source and target feature network parameters so as to have similar feature embeddings. DANN [8] used a domain adversarial neural network, where the feature extractor is trained to generate domain-invariant features using a gradient reversal layer, which inverses the sign of gradients from a domain discriminator.

Improving discriminativeness The idea in DANN has been used as a key component in many subsequent studies [11], [12], [14], [13], which essentially modified the adversarial training architecture to acquire more discriminative power. Different from the end-to-end training in DANN, ADDA (Adversarial Discriminative Domain Adaptation) [11] divided the training into two stages: (a) normal supervised learning on a feature extractor and a feature classifier on the source domain, and (b) training the target domain's feature extractor to output the features similar to the source domain's. In [12], a semantic loss function is used to measure the distance between the centroids of the same class from different domains. Then, minimizing the semantic loss function ensures that the features in the same class from different domains will be mapped nearby. VADA (Virtual Adversarial Domain Adaptation) [14] add two loss functions to DANN to move the decision boundaries to low-density regions. DIRT-T [14] solves the non-conservative domain adaptation problem by applying an additional refinement process to the model trained by VADA.

We summarize another array of work designed for improving discriminativeness. Tri-training method [19] used high-quality pseudo-labeled samples to train an exclusive classifier for the target domains via ensemble neural networks. Co-regularized Domain Adaptation (CoDA) [13] increases the search space by introducing multiple feature embeddings using multiple networks, aligning the target distribution into each space and co-regularizing them to make the networks agree on their predictions. In Generative Adversarial Guided Learning (GAGL) [20], the authors used a generator trained with Central Moment Discrepancy (CMD) [21], similar to what we propose in this paper, in order to boost the classifier performance. However, their experiment results are far from the state-of-the-art performance.

Pixel-level approach We have focused on the feature-level domain adaptation. There exist pixel-level approaches: In [22], the authors proposed to adapt the two domains in the pixel level. The works in [23] and [24] used Cycle GAN [25] to perform the pixel-level adaptation and integrate it with the feature-level domain adaptation in the same model to extract better domain-invariant features.

Bad GAN The idea of using a $(K+1)^{\text{th}}$ output to improve the model performance was widely used in the semi-supervised learning problem [15], [16], [17]. The work in [15] was the first that introduces the $(K+1)^{\text{th}}$ output and apply it to the semi-supervised learning problems. Bad GAN [16] first theoretically and empirically proved the effectiveness of a

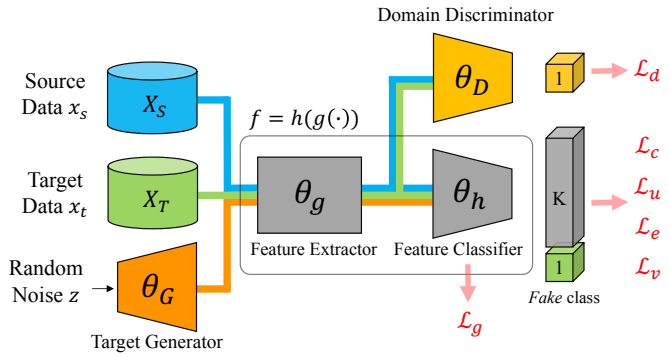


Fig. 3: Network architecture of GADA. Colored solid lines show the flows of source, target and generated data. Six different loss functions are used: (i) $\mathcal{L}_d$ updates θ_D and θ_g for domain-invariance; (ii) $\mathcal{L}_c$, $\mathcal{L}_u$, $\mathcal{L}_e$, and $\mathcal{L}_v$ update θ_g and θ_h to extract discriminative features, and (iii) $\mathcal{L}_g$ updates the generator parameters θ_G. The red arrows show the positions where the losses are computed.

bad generator in helping the classifier to learn, and then designed several loss functions as an attempt to generate bad samples. While the additional output proved its effects in semi-supervised learning, we utilize it in conjunction with other techniques to solve the problem of unsupervised domain adaptation in this paper. Note that in the semi-supervised learning setting, there is only one domain involved in the problem, and the given labels have the same domain as the target data's. Therefore, the model only needs to learn to discriminate input features for high performance. In the domain adaptation problem, the given labels are from the source domain, while target dataset is from the target domain. This domain shift makes the problem more difficult, as the model needs to achieve a good balance between extracting domain-invariant features and discriminative features.

III. Method

A. Unsupervised Domain Adaptation

The problem of unsupervised domain adaptation is formulated as follows. We are given the source dataset with labels $(\mathcal{X}_S, \mathcal{Y}_S)$ from the source domain $\mathcal{D}_S$ and the target dataset $\mathcal{X}_T$ from the target domain $\mathcal{D}_T$, but the target data has no labels. A domain shift between the two domains is assumed, i.e., $\mathcal{D}_S \neq \mathcal{D}_T$. The ultimate goal of unsupervised domain adaptation is to learn a good inference function on the target domain $f : \mathcal{X}_T \to \mathcal{Y}_T$ using the labeled source data $(\mathcal{X}_S, \mathcal{Y}_S)$ and the unlabeled target data $\mathcal{X}_T$.

B. GADA

In this section, we present our method, called GADA (Generative Adversarial Domain Adaptation), ranging from the overall network architecture to the detailed algorithm description.

1) Network Structure: We illustrate the network structure of GADA in Figure 3, which consists of four major components **C1-C4** as follows:

C1. a feature extractor g with parameters θ_g
C2. a feature classifier h with parameters θ_h
C3. a domain discriminator D with parameters θ_D
C4. a generator G with parameters θ_G

The feature extractor g extracts the common features of the source and target inputs, while the feature classifier h classifies the extracted features from g and outputs the classification scores. The domain discriminator D has binary output indicating whether an input is from the source domain or the target domain. The generator G generates the out-of-class (OOC) samples which differs from the target data distribution.

The $(K+1)^{\text{th}}$ class is added to the output layer of the main network $f = h \circ g$, whose parameter is denoted by $\theta = (\theta_g, \theta_h)$. f becomes a discriminator to train G, while the extra class becomes the *fake* class to contain the OOC samples. The classifier f must distinguish the real and the generated OOC samples to improve discriminative power. As mentioned in the introduction (Figure 1), separating real and OOC samples will create an OOC cluster *in-between* the real clusters, which increases the distances among the real clusters, thereby improving the discriminative quality of the features. OOC samples include target-like samples resembling many classes, so the OOC cluster should be placed at the position *near* the real clusters with *similar distances* to all of them in the feature space. Therefore, the OOC cluster will be placed in-between the real clusters instead of being dragged far away.

Remarks In terms of network structure, there are two differences from DANN [9]: (i) the generator G and (ii) the additional $(K+1)^{\text{th}}$ class output. In addition, our method is generic so it can be used with many other approaches, such as VADA and DIRT-T [14], as long as they have their own method of extracting domain-invariant features.

In the remainder of this section, we elaborate GADA by separately presenting the parts that contribute to the extraction of domain-invariant and discriminative features, followed by the whole algorithm description.

2) Domain-invariance via adversarial training: In this subsection, we describe the part of GADA which extracts the features that are invariant for both domains. This job involves the following three components: **(C1)** feature extractor g, **(C2)** feature classifier h and **(C3)** domain discriminator D, where domain-invariant features are extracted by adversarial training. The key idea is that if we are able to fool a smart discriminator D, i.e., leading D to fail to distinguish the input domains, the extracted features $g(X)$ turn out to be domain-invariant.

The loss functions[1] used to train the model are given by:

$$\mathcal{L}_c(\theta; \mathcal{D}_S) = \mathbb{E}_{x,y\sim\mathcal{D}_S}\left[\log P_\theta(\hat{y} = y | x, y \leq K)\right], \quad (1)$$

$$\mathcal{L}_d(\theta_g, \theta_D; \mathcal{D}_S, \mathcal{D}_T) = \mathbb{E}_{x\sim\mathcal{X}_S}\left[\log D(g(x))\right] + \mathbb{E}_{x\sim\mathcal{X}_T}\left[\log(1 - D(g(x)))\right], \quad (2)$$

where $\hat{y}$ indicates the prediction of the network, $\mathcal{L}_d$ is the cross-entropy for the domain discriminator, and $\mathcal{L}_c$ is the negative cross-entropy for the main task.

We note that this is similar to the adversarial training in DANN [9], which we also inherit in GADA, as done by

[1] In this paper, we use the notation $\mathcal{L}_x(\theta_y; \mathcal{D}_z)$ for all loss functions to mean that the loss $\mathcal{L}_x$ uses samples from domain $\mathcal{D}_z$ to update the parameters θ_y.

other related work [11], [12], [14], [13]. The difference is that we replace the gradient reversal layer by an alternating minimization method, which is known to be probably more stable [14]. This alternating training scheme is referred to as Domain Adversarial Training, and is performed as follows:

$$\max_{\theta} \min_{\theta_D} \Big[\mathcal{L}_c(\theta; \mathcal{D}_S) + \lambda_d \mathcal{L}_d(\theta_g, \theta_D; \mathcal{X}_S, \mathcal{X}_T)\Big], \quad (3)$$

where λ_d is the weight of domain discriminator loss $\mathcal{L}_d$. However, the domain discriminator does not consider the class labels while being trained, so the extracted features are not ensured to have sufficient classification capability. Therefore, more optimizations are necessary to extract discriminative features, thereby boosting the performance, which is the key contribution of this paper, as presented in the next section.

3) Discriminativeness by adding an extra class: We now present how we improve the power of discriminativeness in GADA. The three components are associated with this process: **(C1)** feature extractor g, **(C2)** feature classifier h, and **(C4)** generator G (see Figure 3).

Adding extra class and out-of-class generator As presented previously, an out-of-class (OOC) generator generates the samples whose distribution differs from the target data distribution, which provides the power of extracting discriminative features from both domains. In addition, the classifier f must be able to distinguish between the real and generated samples to have better performance, where when real and OOC samples are separated, the distance between the clusters of real samples are increased, thereby improving the discriminative quality of the features.

In order to help the classifier to distinguish the real and OOC samples, we introduce an *unsupervised* objective function as follows:

$$\mathcal{L}_u(\theta; \mathcal{X}_T, P_z) = \mathbb{E}_{x\sim\mathcal{X}_T}\left[\log P_\theta(\hat{y} \le K|x)\right] + \mathbb{E}_{z\sim P_z}\left[\log P_\theta(\hat{y} = K+1|G(z))\right], \quad (4)$$

where P_z is a random noise distribution from which the noise vector z comes. The function $\mathcal{L}_u$ has two terms: (i) the first term is used to train the network with the unlabeled target data, and (ii) the second term is to train the network with the generated samples. By maximizing the first term, we maximize the probability that an unlabeled target sample belongs to one of the first K classes. By maximizing the second term, we maximize the probability that a generated sample belongs to the fictitious $(K+1)^{\text{th}}$ class.

In addition to the objective function used in training the discriminator, we need a loss function to train the OOC generator. In [16], the authors claim that a complementary generator, which generates no in-indistribution samples, is essential to improve the performance. They proposed a bad generator (BadGAN) as an attempt to mimic the complementary generator, but it is too costly to be implemented. In our model, we use an imperfect complementary generator to reduce the implementation complexity using Feature Matching (FM) objective [15]. Using only FM objective function puts less constraints to the generator compared to BadGAN. The function is defined as follows:

$$\mathcal{L}_g(\theta_G; \mathcal{X}_T, P_z) = \left\|\mathbb{E}_{x\sim\mathcal{X}_T}\left[\phi(x)\right] - \mathbb{E}_{z\sim P_z}\left[\phi(G(z))\right]\right\|, \quad (5)$$

where ϕ is an immediate layer in the network. In our implementation, we choose ϕ to be the last hidden layer of the feature classifier h. FM matches the statistics (in this case, the mean) of each minibatch, which leads to a less constrained loss function that helps the generator to generate OOC samples [15], [16]. Note that we apply (5) to generate the target domain samples only, because the source samples are provided with the labels, which are more adequate for training. In addition, training the network with the generated source samples might hurt the performance because of non-conservativeness of domain adaptation [14] considered in this paper.

Entropy minimization and virtual adversarial training (VAT) We also minimize the entropy of the model's output in order to make the model more confident about its prediction using the following objective:

$$\mathcal{L}_e(\theta; \mathcal{D}_T) = -\mathbb{E}_{x\sim\mathcal{D}_T}\left[f(x)^\top \ln f(x)\right]. \quad (6)$$

This loss function prevents the target data from being located near the decision boundary, or in other words, places the decision boundary in the low-density area. In fact, our extra class method boosts the effect of this loss function. Our method increases the distances between the real clusters, which creates more low-density areas for the decision boundary to pass through. Therefore, a solution to minimize (6) can be found easier.

Adversarial training has been proposed to increase the robustness of the classifier to the adversarial attack which intentionally perturbes samples to degrade the prediction accuracy. Virtual Adversarial Training (VAT) was proposed for the same purpose: it ensures consistent predictions for all samples that are slightly perturbed from the original sample, where the following loss function is used:

$$\mathcal{L}_v(\theta; \mathcal{D}) = \mathbb{E}_{x\sim\mathcal{D}}\left[\max_{\|r\|\le\epsilon} \mathrm{D}_{\mathrm{KL}}(f(x) \parallel f(x+r))\right]. \quad (7)$$

This loss regularizes the classifier so that it does not change its prediction abruptly due to the perturbation of inputs, which helps to learn a robust classifier. Note that entropy minimization and VAT are popularly used in domain adaptation, as in [13] and [14].

Aggregation To extract the discriminative features, using the loss functions introduced earlier, we perform alternating optimization between the following two:

$$\max_{\theta} \mathcal{L}_c(\theta; \mathcal{D}_S) + \lambda_u \mathcal{L}_u(\theta; \mathcal{X}_T, P_z) + \lambda_s \mathcal{L}_v(\theta; \mathcal{D}_S) + \lambda_t\left[\mathcal{L}_v(\theta; \mathcal{D}_T) + \mathcal{L}_e(\theta; \mathcal{D}_T)\right],$$
$$\min_{\theta_G} \mathcal{L}_g(\theta_G; \mathcal{X}_T, P_z),$$

Algorithm 1 GADA

The following three steps are sequentially repeated until convergence.

S1. ***Update the classifier.*** Sample M source samples with the corresponding labels (x_S, y_S), M unlabeled target samples x_T, and M random noise vectors z, to update the feature extractor g and the feature classifier h:

$$\begin{aligned}\max_{\theta} \mathcal{L}_c(\theta; \mathcal{D}_S) + \lambda_d \mathcal{L}_d(\theta_g, \theta_D; \mathcal{X}_S, \mathcal{X}_T) \\ + \lambda_u \mathcal{L}_u(\theta; \mathcal{X}_T, P_z) + \lambda_s \mathcal{L}_v(\theta; \mathcal{D}_S) \\ + \lambda_t \left[\mathcal{L}_v(\theta; \mathcal{D}_T) + \mathcal{L}_e(\theta; \mathcal{D}_T)\right].\end{aligned}$$

S2. ***Update the domain discriminator.*** Sample M source samples x_S and M target samples x_T to update the domain discriminator D by minimizing $\mathcal{L}_d$:

$$\min_{\theta_D} \mathcal{L}_d(\theta_g, \theta_D; \mathcal{X}_S, \mathcal{X}_T).$$

S3. ***Update the generator.*** Sample M random noise vectors z and M target samples x_T, update the generator G by minimizing $\mathcal{L}_g$:

$$\min_{\theta_G} \mathcal{L}_g(\theta_G; \mathcal{X}_T, P_z).$$

where $\mathcal{L}_c$ is the negative cross-entropy function defined in (1), while λ_u, λ_s, and λ_t are the hyperparameters to control the impact of each loss function. Note that the VAT objective function is applied to both the source and target domains, as suggested by [14].

4) GADA Algorithm: Combining the contents in the previous two subsections, GADA solves the following optimization in training based on the network structure in Figure 3:

$$\begin{aligned}\max_{\theta} \min_{\theta_D} \min_{\theta_G} \underbrace{\mathcal{L}_c(\theta; \mathcal{D}_S)}_{\text{(a)}} + \underbrace{\lambda_d \mathcal{L}_d(\theta_g, \theta_D; \mathcal{X}_S, \mathcal{X}_T)}_{\text{(b)}} \\ + \underbrace{\lambda_s \mathcal{L}_v(\theta; \mathcal{D}_S) + \lambda_t \left[\mathcal{L}_v(\theta; \mathcal{D}_T) + \mathcal{L}_e(\theta; \mathcal{D}_T)\right]}_{\text{(c)}} \\ + \underbrace{\lambda_u \mathcal{L}_u(\theta; \mathcal{X}_T, P_z) + \mathcal{L}_g(\theta_G; \mathcal{X}_T, P_z)}_{\text{(c)}}. \quad (8)\end{aligned}$$

The above function is interpreted as follows. Maximizing (a) guides the network to achieve the classification power from the source data and labels. Updating θ_D to minimize (b), while updating θ_g to maximize it, helps the network to extract domain-invariant features. (c) improves discriminativeness by generating OOC samples and classifying them into the fictitious class $K+1$, as well as regularizing the model with entropy minimization and VAT objective. The complete training algorithm is presented in Algorithm 1. Since the algorithm monotonically decreases the objective function value, the convergence is guaranteed.

IV. Experimental Results

A. Baselines and Domain Adaptation Tasks

We mostly compare our algorithm GADA against the two recent state-of-the-art methods, VADA+DIRT-T [14] and CoDA [13]. We also include other strong algorithms as the baselines, such as DSN [10], ATT [19], MCD [26], DA Assoc. [27], and GAGL [20]. We evaluate the algorithms with the standard datasets, which include four digit datasets (MNIST, SVHN, MNIST-M, and SynthDigits) and two object datasets (CIFAR-10 and STL-10).

MNIST $\leftrightarrow$ SVHN Both MNIST and SVHN are digit data sets different in style. MNIST consists of gray-scale handwritten images, while SVHN includes images of RGB house numbers. Due to the lower input dimension in MNIST, we upscale MNIST images to have the same dimension as SVHN with three same color channels. The task MNIST $\rightarrow$ SVHN is known to be highly challenging. We observe that this task has been omitted in many related papers, possibly due to the adaptation hardness. The task of the opposite direction SVHN $\rightarrow$ MNIST is relatively easy, because the classifier is trained with the labels from SVHN, the more complex source domain.

MNIST $\rightarrow$ MNIST-M MNIST-M is constructed by blending the gray-scale MNIST images with colored backgrounds in BSDS500 dataset [28]. The resulting color images in MNIST-M increase the domain shift between the two datasets, thus this adaptation task has been widely used to compare the performance of various models [9], [10], [19], [14], [13].

SynthDigits (DIGITS) $\rightarrow$ SVHN SynthDigits is a synthetic digit dataset consisting of 500,000 images generated from Windows fonts by varying the text, positioning, orientation, background, stroke color, and the amount of blur. This task reflects a common adaptation task from synthetic images (synthesized images) to real images (house number pictures).

CIFAR-10 $\leftrightarrow$ STL-10 Both CIFAR-10 and STL-10 are RGB object images, each with 10 different classes. We remove the non-overlapping classes in each data set (*frog* in CIFAR-10 and *monkey* in STL-10) and perform the training and evaluation on the 9 leftover classes. Because of the difference in dimensions, we downscale all STL images to match the dimension of CIFAR-10's. Since CIFAR-10 has more labeled data than STL-10, it is easier to adapt from CIFAR-10 to STL-10 than the opposite direction.

B. Implementation Details

Network architecture We use a small convolutional neural network (CNN) for the digit datasets, and a larger one for the object datasets. We apply batch normalization to all fully-connected and CNN layers, while dropout and additive Gaussian noise are used in several layers. As for the generator, we use transposed convolution layers to upsample the feature maps.

Hyperparameters In all the experiments, we train the network using Adam Optimizer. Following Shu et al. [14], we randomly select 1000 labeled target samples to do a grid

TABLE I: Comparison of state-of-the-art methods in terms of classification accuracy (%). Values in bold indicate the best result. To see the performance of $\mathcal{L}_u + \mathcal{L}_g$ of GADA versus $\mathcal{L}_e + \mathcal{L}_v$ of VADA, we include the results of GADA*, a model trained with $\lambda_s = \lambda_t = 0$.

Source Target	MNIST SVHN	SVHN MNIST	MNIST MNIST-M	DIGITS SVHN	CIFAR STL	STL CIFAR
DANN[8]	35.7	71.1	81.5	90.3	-	-
DSN[10]	-	82.7	83.2	91.2	-	-
ATT[19]	52.8	86.2	94.2	92.9	-	-
MCD[26]	-	96.2	-	-	-	-
DA Assoc.[27]	-	97.6	89.5	91.9	-	-
GAGL[20]	74.6	96.7	94.9	93.1	77.0	61.5
Without instance-normalized input						
VADA[14]	47.5	97.9	97.7	94.8	80.0	73.5
CoDA[13]	55.3	98.8	99.0	**96.1**	**81.4**	**76.4**
GADA* (ours)	69.9	98.2	98.2	94.9	79.7	73.0
GADA (ours)	**72.3**	**98.9**	**99.1**	95.7	80.5	75.1
VADA+DIRT-T[14]	54.5	99.4	98.9	96.1	-	75.3
CoDA+DIRT-T[13]	63.0	99.4	99.1	96.5	-	**77.6**
GADA*+DIRT-T (ours)	82.4	99.4	98.9	96.3	-	75.3
GADA+DIRT-T (ours)	**83.7**	**99.6**	**99.3**	**96.6**	-	76.5
With instance-normalized input						
VADA[14]	73.3	94.5	95.7	94.9	78.3	71.4
CoDA[13]	81.7	98.7	98.0	96.0	**80.6**	74.7
GADA* (ours)	78.7	**99.0**	97.2	95.3	78.9	72.1
GADA (ours)	**83.6**	98.8	**98.2**	**96.1**	80.1	**74.9**
VADA+DIRT-T[14]	76.5	99.4	98.7	96.2	-	73.3
CoDA+DIRT-T[13]	88.0	99.4	98.8	96.5	-	75.9
GADA*+DIRT-T (ours)	84.5	99.4	98.8	96.5	-	74.7
GADA+DIRT-T (ours)	**90.0**	**99.6**	**99.0**	**96.8**	-	**76.2**

(a) Original MNIST images (b) Generated MNIST images (c) Original SVHN images (d) Generated SVHN images

Fig. 4: Comparison between original and generated images in the tasks SVHN → MNIST (Figures (a) and (b)) and MNIST → SVHN (Figures (c) and (d)). Bad samples of images are generated after training.

search of hyperparameter, with the learning rate restricted to $\{2\times10^{-4}, 10^{-3}\}$, while λ_d is either 10^{-2} or 0. We also restrict other hyperparameters to $\lambda_s = \{0, 1\}$, $\lambda_t = \{10^{-1}, 10^{-2}\}$ and $\lambda_u = \{10^{-1}, 10^{-2}\}$.

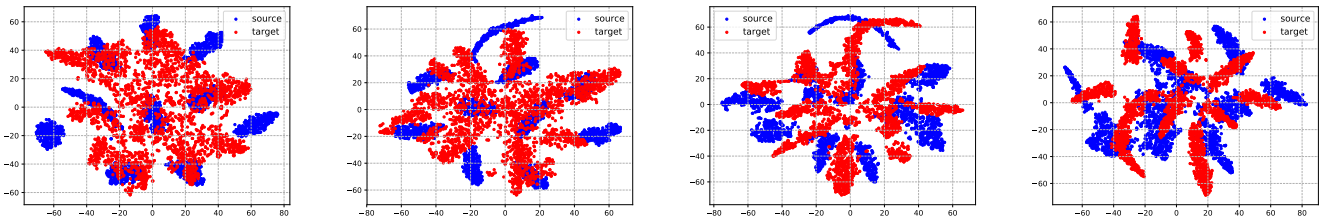
Instance normalization As suggested in [14], we apply the instance normalization to the rescaled input images. This procedure renders the classifier invariant to channel-wide shifts and rescaling of pixel intensities. The results for using and non-using instance normalization are both presented.

DIRT-T integration For a fair comparison with VADA and CoDA, after training a model using GADA, we refine it using DIRT-T[14], which proves to be effective in improving the performance. In all the experiments, we refine the model with $\beta = 10^{-2}$, except for STL-10 → CIFAR-10, where β is set to 10^{-1}. Note that we do not apply DIRT-T to CIFAR-10 → STL-10 because the number of target samples in the task is low (450 samples), which provides unreliable estimation of the entropy for minimization.

Generator pretraining When we train all networks from scratch, the noisy gradients at the beginning of the training process hurt the training of the generator. Therefore, we pretrain the generator before using it to generate training samples for the classifier. When the main classifier is trained with the pretrained generator, we keep finetuning the generator with a small learning rate of 2×10^{-5}. We find that the pretrained generator strongly improves the performance of the classifier, especially in the task MNIST → SVHN with no instance normalization. Without a pretrained generator, GADA only scores approximately 20% of accuracy in this task. However, when we pretrain the generator, the accuracy rises up to 72.3%, as presented in Table I.

C. Evaluation and Analysis

Overall comparison All the results on comparison with other baselines are presented in Table I. To summarize, we achieve state-of-the-art results across *five* tasks, MNIST ↔ SVHN, MNIST → MNIST-M, DIGITS → SVHN, and STL-



(a) VADA (acc: 70.6%) (b) GADA (acc: 83.6%) (c) VADA+DIRT-T (acc: 75.75%) (d) GADA+DIRT-T (acc: 90%)

Fig. 5: Feature space comparison between VADA [14] and GADA in MNIST → SVHN. Combining DIRT-T with GADA significantly improves the performance, which shows that our GADA module could be efficiently used to boost other techniques.

10 → CIFAR-10 (with instance normalization). For the highly challenging adaptation task of MNIST → SVHN, we gained a significant improvement of approximately 21% accuracy compared to the state-of-the-art algorithm CoDA [13] when no instance normalization is applied. With instance normalization, we achieved a considerable improvement of approximately 2% over CoDA. In the task CIFAR-10 → STL-10, GADA outperformed VADA [14] but underperformed CoDA by a small margin, because STL contains a very small number of samples in the training set (50 images per class), which seems to hurt the generator training process. For STL → CIFAR, our performance is the best when instance normalization is applied. However, without this preprocessing step, we outperformed VADA [14] but lost to CoDA [13] by about 1%. Also note that GADA*, a model trained with $\lambda_s = \lambda_t = 0$, outperformed VADA [14] in almost all the scenarios, which shows that our extra class module is more effective than the loss functions $\mathcal{L}_e$ and $\mathcal{L}_v$ used in VADA. More details are in the below ablation study.

Generated images Generated images are shown in Figure 4 for MNIST → SVHN and SVHN → MNIST. We can see that in both tasks, the digits in the generated images are recognizable, but their shapes, styles or colors were changed. This causes them to look different from the original training images, or simply *bad*. This analysis empirically shows that the distribution of the generated images is different from the training data's, while preserving meaningful features for the network to learn from.

Feature space visualization In Figure 5, we compare the t-SNE plots of the last hidden layer of VADA models (Figures 5(a) and 5(c)), and GADA models (Figures 5(b) and 5(d)). We observe that the feature space of GADA is more organized with more separate clusters, compared to those of VADA. GADA increases the distances among the clusters, which follows our intuition in the beginning. This results in a much higher accuracy (83.6% compared to 70.6%). When integrated with DIRT-T [14], our performance becomes boosted further from 83.6% to 90%. This experiment shows the power of GADA when integrated with other methods, which proves the generic characteristic of our module.

Ablation study We evaluate the contributions of the loss functions in terms of accuracy on the adaptation task MNIST → SVHN in Table II. Instance normalization is applied to all the cases for a fair comparison. Our results show that adding one of the terms $\mathcal{L}_e$, $\mathcal{L}_v$, or the extra class $(\mathcal{L}_u, \mathcal{L}_g)$ into DANN improves the performance in a stable manner. Among the three, our proposed extra class method provide the highest improvement, at 12.4% compared to 1.8% and 3.6%. We merge both $\mathcal{L}_e$ and $\mathcal{L}_v$ into DANN to have our implementation of VADA [14] to achieve a higher improvement, but the performance gain is still much less than that of GADA*. The best result is achieved when we add the extra class into VADA (last row), which creates an improvement of 13% over VADA in terms of accuracy and surpasses the state-of-the-art result in CoDA [13]. This ablation analysis shows that our module could be integrated into other methods, such as DANN or VADA, for higher performance.

TABLE II: Accuracy (%) on test set of the task MNIST → SVHN for ablation analysis. The numbers in brackets show the relative improvement compared to DANN's result.

	$\mathcal{L}_c$	$\mathcal{L}_d$	$\mathcal{L}_e$	$\mathcal{L}_v$	$\mathcal{L}_u, \mathcal{L}_g$	MNIST → SVHN
DANN(ours)	✓	✓				66.3
DANN+$\mathcal{L}_e$	✓	✓	✓			68.1 (+1.8)
DANN+$\mathcal{L}_v$	✓	✓		✓		69.9 (+3.6)
GADA*	✓	✓			✓	78.7 (+12.4)
VADA(ours)	✓	✓	✓	✓		70.6 (+4.3)
GADA	✓	✓	✓	✓	✓	**83.6 (+17.3)**

Confusion matrix In Figure 6, we present a confusion matrix that shows the prediction accuracy for each of the nine different classes in the task STL-10 → CIFAR-10. We observe that our model works very well with several classes, such as *automobile, ship*, and *truck*, each of them achieves accuracy of approximately 90%. The class that degrades our performance the most is *bird* with only 51% of accuracy. Our model misclassifies the bird images as *cat, deer*, and *dog*. We suspect that it is because of the noisy learning in the beginning of the training.

V. Conclusion

We proposed the Generative Adversarial Domain Adaptation (GADA) algorithm, which significantly improves the discriminative feature extraction by injecting an extra class and training with generated samples. The extra class along with the loss functions increases the distances among the real target clusters, thereby improving the discriminative quality of the extracted features. Through extensive experiments on different

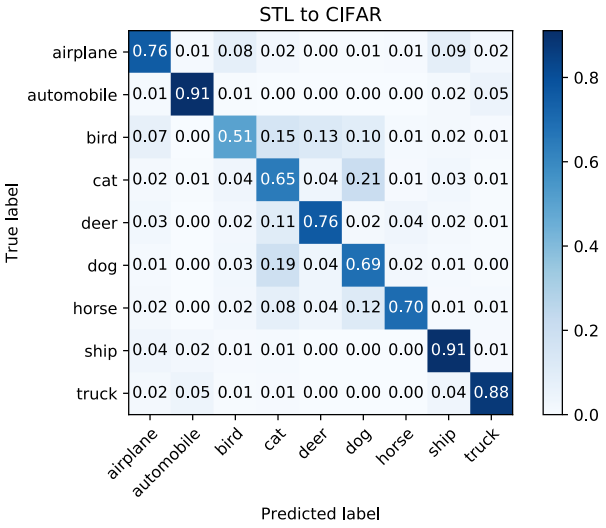


Fig. 6: Confusion matrix for STL-10 → CIFAR-10.

standard datasets, we showed the performance of our method, which outperformed the other state-of-the-art algorithms in many cases, especially on the highly challenging adaptation task MNIST → SVHN. In addition, our module is shown to be extremely effective when integrated into other methods.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1

[2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 1

[3] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pages 173–182, 2016. 1

[4] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *CoRR*, abs/1412.3474, 2014. 1, 2

[5] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I. Jordan. Learning transferable features with deep adaptation networks. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, pages 97–105, 2015. 1, 2

[6] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I. Jordan. Deep transfer learning with joint adaptation networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, pages 2208–2217, 2017. 1, 2

[7] Baochen Sun and Kate Saenko. Deep coral: Correlation alignment for deep domain adaptation. In *Computer Vision – ECCV 2016 Workshops*, pages 443–450, 2016. 1, 2

[8] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1180–1189, 2015. 1, 2, 6

[9] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016. 1, 2, 3, 5

[10] Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan, and Dumitru Erhan. Domain separation networks. In *Advances in Neural Information Processing Systems*, pages 343–351, 2016. 1, 5, 6

[11] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, pages 2962–2971, 2017. 1, 2, 4

[12] Shaoan Xie, Zibin Zheng, Liang Chen, and Chuan Chen. Learning semantic representations for unsupervised domain adaptation. In *Proceedings of the 35th International Conference on Machine Learning*, pages 5423–5432, 2018. 1, 2, 4

[13] Abhishek Kumar, Prasanna Sattigeri, Kahini Wadhawan, Leonid Karlinsky, Rogerio Feris, Bill Freeman, and Gregory Wornell. Co-regularized alignment for unsupervised domain adaptation. In *Advances in Neural Information Processing Systems*, pages 9345–9356, 2018. 1, 2, 4, 5, 6, 7

[14] Rui Shu, Hung Bui, Hirokazu Narui, and Stefano Ermon. A DIRT-t approach to unsupervised domain adaptation. In *International Conference on Learning Representations*, 2018. 1, 2, 3, 4, 5, 6, 7

[15] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems 29*, pages 2234–2242, 2016. 1, 2, 4

[16] Zihang Dai, Zhilin Yang, Fan Yang, William W Cohen, and Ruslan R Salakhutdinov. Good semi-supervised learning that requires a bad gan. In *Advances in Neural Information Processing Systems 30*, pages 6510–6520, 2017. 1, 2, 4

[17] Guo-Jun Qi, Liheng Zhang, Hao Hu, Marzieh Edraki, Jingdong Wang, and Xian-Sheng Hua. Global versus localized generative adversarial nets. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 1, 2

[18] Artem Rozantsev, Mathieu Salzmann, and Pascal Fua. Beyond sharing weights for deep domain adaptation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41:801–814, 2018. 2

[19] Kuniaki Saito, Yoshitaka Ushiku, and Tatsuya Harada. Asymmetric tri-training for unsupervised domain adaptation. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2988–2997, 2017. 2, 5, 6

[20] Kai-Ya Wei and Chiou-Ting Hsu. Generative adversarial guided learning for domain adaptation. In *British Machine Vision Conference (BMVC)*, 2018. 2, 5, 6

[21] Werner Zellinger, Thomas Grubinger, Edwin Lughofer, Thomas Natschläger, and Susanne Saminger-Platz. Central moment discrepancy (cmd) for domain-invariant representation learning. In *International Conference on Learning Representations*, 2017. 2

[22] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 95–104, 2017. 2

[23] Zak Murez, Soheil Kolouri, David J. Kriegman, Ravi Ramamoorthi, and Kyungnam Kim. Image to image translation for domain adaptation. In *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4500–4509, 2018. 2

[24] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. CyCADA: Cycle-consistent adversarial domain adaptation. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1989–1998, 2018. 2

[25] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2242–2251, 2017. 2

[26] Kuniaki Saito, Kohei Watanabe, Yoshitaka Ushiku, and Tatsuya Harada. Maximum classifier discrepancy for unsupervised domain adaptation. In *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3723–3732, 2018. 5, 6

[27] Philip Häusser, Thomas Frerix, Alexander Mordvintsev, and Daniel Cremers. Associative domain adaptation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2784–2792, 2017. 5, 6

[28] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 33:898–916, 05 2011. 5