Unsupervised Domain Adaptation with Deep Neural-Networks

MSIAM M2, 2022-2023

Artem Bitiutskii

Supervisors: Massih-Reza Amini and Marianne Clausel

Contents

1 | Introduction

Articles can be divided into two groups: practical and theoretical. We start with necessary theory to understand some essential definitions and ideas associated with domain adaptation. Then we continue with papers that cover some practical implementations of various approaches.

1.1 | Theoretical part

Domain adaptation is a subfield of machine learning that deals with the problem of transferring knowledge learned from one domain to another related but different domain. In real-world scenarios, it is common to encounter situations where the data distribution of the target domain differs significantly from the source domain used to train a model. This can lead to a significant drop in the performance of the model on the target domain.

To understand clearly what domain adaptation is about, we should start with transfer learning. For this purpose, we should dig deeper into the theory. In these articles [9] and [15] you can find a high level overview of the theory that connects with domain adaptation. Let's start with the transfer learning definition and types that it consists of.

Definition 1.1. (Transfer learning) We consider a source data distribution S called the source domain, and a target data distribution T called the target domain. Let $X_S \times Y_S$ be the source input and output spaces associated to S, and $X_T \times Y_T$ be the target input and output spaces associated to T. We use S_X and T_X to denote the marginal distributions of X_S and X_T , t_S and t_T to denote the source and target learning tasks depending on Y_S and Y_T, respectively. Then, transfer learning aims to help to improve the learning of the target predictive function $f_T : X_T \longrightarrow Y_T$ for t_T using knowledge gained from S and t_S , where $S = T$.

According to these papers ([9], [15]), transfer learning algorithms can be classified into three categories based on the differences between the source and target tasks and domains: inductive, transductive, and unsupervised transfer learning.

- **Inductive transfer learning** involves using labeled data from the source domain to train a model for a different, but related, target task in the target domain. In this case, some labeled data from the target domain is required to fine-tune the model.
- **Transductive transfer learning**, on the other hand, refers to using both labeled data from the source domain and unlabeled data from the target domain to improve the model's performance on the target domain. In this case, the tasks remain the same while the domains are different.
- **Unsupervised transfer learning** involves adapting a model trained on the source task to perform well on a related, but different target task in the target domain, without any labeled data in either the source or target domains.

Domain adaptation is a type of transfer learning where the target task remains the same as the source task, but the domain differs (the second type – transductive transfer learning). Depending on whether the feature spaces remain the same or differ, domain adaptation is

categorized into homogeneous and heterogeneous domain adaptation. Machine learning techniques are commonly categorized based on the availability of labeled training data, such as supervised, semi-supervised, and unsupervised learning. However, domain adaptation assumes the availability of data from both the source and target domains, making it ambiguous to append one of these three terms to "domain adaptation". There are different ways how these terms can be applied to domain adaptation, but we use the same as in [15].

- **Unsupervised domain adaptation** refers to the case where both labeled source data and unlabeled target data are available
- **Semi-supervised domain adaptation** refers to the case where labeled source data and some labeled target data are available
- **Supervised domain adaptation** refers to the case where both labeled source and target data are available.

This paper is more focused on studying unsupervised domain adaptation with using deep neural-networks. Before move on to the practical part, it is important to discuss theoretical analysis and guarantees that can be used in fields associated with transfer learning. Thus, there are several methods that allow you to analyze the generalization gap in machine learning [14]. One of the most popular approaches is the model complexity approach, which estimates the generalization bound by measuring the complexity of the hypothesis set, such as Vapnik-Chervonenkis (VC) dimension and Rademacher complexity. Another approach is to use the stability of the supervised learning algorithm in relation to the datasets. Stability is a measure of how much a change in a data point in the training set can affect the output of the algorithm. Both of these approaches have been used to analyze the generalization bounds of transfer learning algorithms.

It is equally important to discuss distributions and what experts mean by shift when analyzing transfer learning algorithms. Distribution refers to the set of all possible values of a random variable, and a shift refers to a change in the distribution of the data between the source and target domains. Understanding the shift in the distribution of the data is crucial in developing effective transfer learning algorithms, as it enables the selection of appropriate techniques for adapting the model to the target domain.

Unsupervised domain adaptation (UDA) is a type of supervised learning that involves training a model using labeled source data and applying it to unlabeled target data, where the distributions of the two domains differ. Let the source domain be represented by $(x^S, y^S) = (x_k^S, y_k^S)_{k=1}^{m_S}$, and the target domain be represented by $x^T = (x_k^T)_{k=1}^{m_T}$. The number of observations in the source and target domains are denoted by m_S and m_T respectively. The main challenge of domain adaptation is to develop a predictor that performs well in the target domain by leveraging the similarities between the two domains. One way to accomplish this is by making assumptions about how the joint distribution $P(X, Y)$ changes across the domains. In the case of *covariate shift*, the marginal distribution $P(X)$ changes while the conditional distribution $P(Y|X)$ remains the same. However, in real-world scenarios, $P(Y|X)$ may also change, requiring further assumptions. One such assumption is that the joint distribution can be factored into $P(Y)$ and $P(X|Y)$, allowing changes in $P(Y)$ and $P(X|Y)$ to be addressed independently. The problem is then broken down into three types of shifts [12]:

- **Target shift**, where $P(Y)$ changes while $P(X|Y)$ remains the same
- **Conditional shift**, where $P(X|Y)$ changes while $P(Y)$ stays the same
- **Conditional-target shift**, where both $P(X|Y)$ and $P(Y)$ change independently

This research paper aims to explore the application of deep neural networks in unsupervised domain adaptation for learning latent representations in scenarios where there is a "conditional shift" or "conditional target-shift" in the joint distribution of features and labels.

1.2 | Practical part

In this section, we discuss main purposes, approaches and algorithms that specialists in the field of domain adaptation use in their research. In this section (1.2) all figures are taken from the articles.

1.2.1 | First article

The purpose of the article "Unsupervised Domain Adaptation by Backpropagation" written by Yaroslav Ganin and Victor Lempitsky [2] is to tackle the problem of domain shift in machine learning and to propose a solution to this problem using a neural-network model with few standard layers and gradient reversal layer (GRL). The GRL makes the network to learn domain-invariant features by minimizing the difference between the distributions of the source and target domains. The architecture of the model is shown below (see Figure 1)

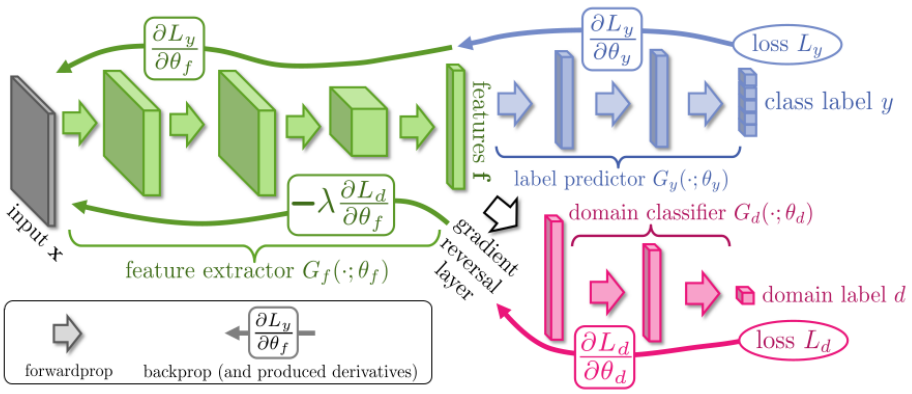

Figure 1: The proposed architecture includes a deep feature extractor (shown in green), a deep label predictor (shown in blue) and a domain classifier (shown in red). The domain classifier is connected to the feature extractor via a gradient reversal layer, which multiplies the incoming gradient by a negative constant during backpropagation-based training. The gradient reversal layer ensures that the feature distributions over the two domains become as similar as possible (i.e., indistinguishable by the domain classifier), resulting in domain-invariant features.

The authors introduce an architecture that predicts both the label $y \in Y$ and the domain label $d \in \{0, 1\}$ for each input $\mathbf{x}$. The architecture consists of three parts: feature extractor

$\mathbf{f} = G_f(\mathbf{x}, \theta_f)$, where θ_f is a vector that represents the parameters of all its layers; label predictor G_y that maps the features obtained after feature extractor to the label y, with θ_y representing its parameters; domain classifier G_d maps the same feature vector $\mathbf{f}$ to the domain label d, with θ_d representing its parameters. The purpose to minimize the label prediction loss for the source domain and simultaneously make the features $\mathbf{f}$ invariant to the domain. To achieve this, the authors optimize the parameters θ_f of the feature mapping to maximize the loss of the domain classifier, however the parameters θ_d are optimized to minimize the loss of the domain classifier. The authors consider the loss

$$\begin{aligned} E(\theta_f, \theta_y, \theta_d) &= \sum_{\substack{i=1..N \\ d_i=0}} L_y(G_y(G_f(\mathbf{x}_i; \theta_f); \theta_y), y_i) - \lambda \sum_{i=1..N} L_d(G_d(G_f(\mathbf{x}_i; \theta_f); \theta_d), y_i) \\ &= \sum_{\substack{i=0..N \\ d_i=0}} L_y^i(\theta_f, \theta_y) - \lambda \sum_{i=1..N} L_d^i(\theta_f, \theta_d) \end{aligned}$$

where L_y and L_d are label prediction and domain classification losses, respectively. (index i means the i-th example). It is considered the parameters $\hat{\theta}_f, \hat{\theta}_y, \hat{\theta}_d$ to gain a saddle point

$$(\hat{\theta}_f, \hat{\theta}_y) = \arg\min_{\theta_f, \theta_y} E(\theta_f, \theta_y, \hat{\theta}_d)$$

$$\hat{\theta}_d = \arg\max_{\theta_d} E(\hat{\theta}_f, \hat{\theta}_y, \theta_d)$$

During learning, the trade-off between the two objectives that shape the features is controlled by the parameter λ. The following stochastic updates can find a saddle point

$$\theta_f \leftarrow \theta_f - \mu \left(\frac{\partial L_y^i}{\partial \theta_f} - \lambda \frac{\partial L_d^i}{\partial \theta_f} \right)$$

$$\theta_y \leftarrow \theta_y - \mu \frac{\partial L_y^i}{\partial \theta_y}$$

$$\theta_d \leftarrow \theta_d - \mu \frac{\partial L_d^i}{\partial \theta_d}$$

where μ is a learning rate. These updates are similar to SGD but with a $-\lambda$ factor in the first update to prevent dissimilar features across domains. Therefore, the authors introduce a GRL that acts as an identity transform during forward propagation but multiplies the gradient by $-\lambda$ during backpropagation.

1.2.2 | Second article

Next, we continue with the article *"Learning Semantic Representations for Unsupervised Domain Adaptation"* written by Xie, Shaoan, et al. [16] The main purpose of the article is to propose a new method for unsupervised domain adaptation that utilizes semantic information to learn domain-invariant representations. The authors propose a domain adaptation algorithm which is based on the idea of using an adversarial learning to learn a feature representation that is invariant to domain shifts.

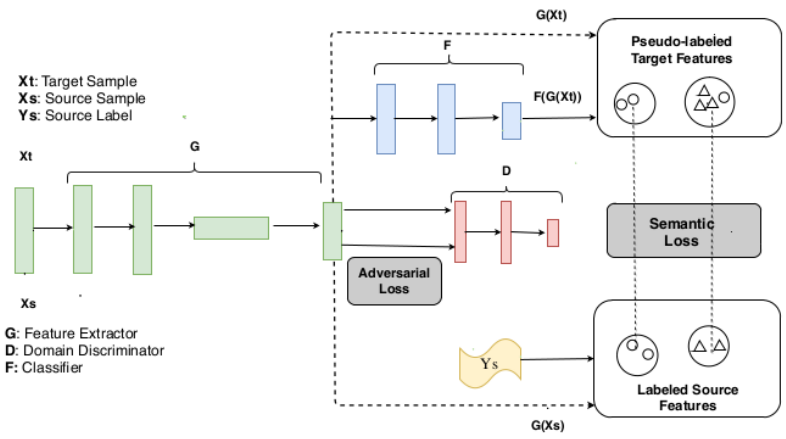

Figure 2: The authors use standard source classification loss with the domain adversarial loss to align distribution for two domains. It is showed that the performance of the domain adaptation method improved significantly on several benchmark datasets by aligning the centroids. Global centroids C_S^k and C_T^k is maintained for each class at feature level.

The authors train a feature extractor and then use it to map the input data to a high-dimensional feature space, and a domain classifier that predicts the domain label of the input data (see Figure 2). The feature extractor **G** is trained to confuse the domain classifier **D**, while the domain classifier is trained to correctly predict the domain label. In this way, the feature extractor is encouraged to learn features that are invariant to domain shifts, while still being discriminative for the task.

First, the authors denote the cross entropy loss for the source domain as $L_C(X_S, Y_S)$. Then, the discrepancy between source domain and target domain is supposed to be

$$L_{DC}(X_S, X_T) = d(X_S, X_T) = \mathbb{E}_{x\sim D_S}[\log(1 - D \circ G(x))] + \mathbb{E}_{x\sim D_T}[\log(D \circ G(x))]$$

Moreover, the authors introduce one more loss, which targets the semantic representation. Centroid alignment is used for this purpose. By computing the centroid for each class, both correct and incorrect pseudo-labeled samples are utilized together:

$$L_{SM}(X_S, Y_S, X_T) = \sum_{k=1}^{K} \Phi(C_S^k, C_T^k)$$

where C_S^k, C_T^k are centroids for each class and $\Phi(x, x') = \|x - x'\|^2$. This approach aims to cancel out the negative effects caused by inaccurate pseudo labels with accurate ones. Thus, the authors get the following total loss

$$L(X_S, Y_S, X_T) = L_C(X_S, Y_S) + \lambda L_{DC}(X_S, X_T) + \gamma L_{SM}(X_S, Y_S, X_T)$$

where λ and γ are responsible for the balance between the classification loss, domain confusion loss and semantic loss. In the article, algorithm of **moving average centroid alignment** is presented that allows to align the centroids in same class but different domains to achieve semantic transfer for UDA.

1.2.3 | Third article

The purpose of the article "Fixbi: Bridging domain spaces for unsupervised domain adaptation" written by Jaemin Na, Heechul Jung et al. [6] is to propose a fixed ratio-based mixup method to address the problem of large domain discrepancies. The authors mix up images and then fed them into neural networks to achieve greater reliability in learning from corrupted labels. It is proposed to use two predetermined mixup ratios $\lambda_s d$ and $\lambda_t d$ for the source and target domain respectively. Denote input samples and their labels for source and target domain as (x_i^s, y_i^s) and $(x_i^t, \hat{y}_i^t)$, the authors define mixup configurations in the following way:

$$\tilde{x}_i^{st} = \lambda x_i^s + (1-\lambda) x_i^t$$
$$\tilde{y}_i^{st} = \lambda y_i^s + (1-\lambda) \hat{y}_i^t,$$

where $\lambda \in \{\lambda_{sd}, \lambda_{td}\}$ and $\lambda_{sd} + \lambda_{td} = 1$, $\hat{y}_i^t$ is the pseudo-labels for the target samples. By leveraging the fixed ratio-based mixup, it is constructed two neural networks with different perspectives: the "source-dominant model" (SDM) and the "target-dominant model" (TDM) (see Figure 3).

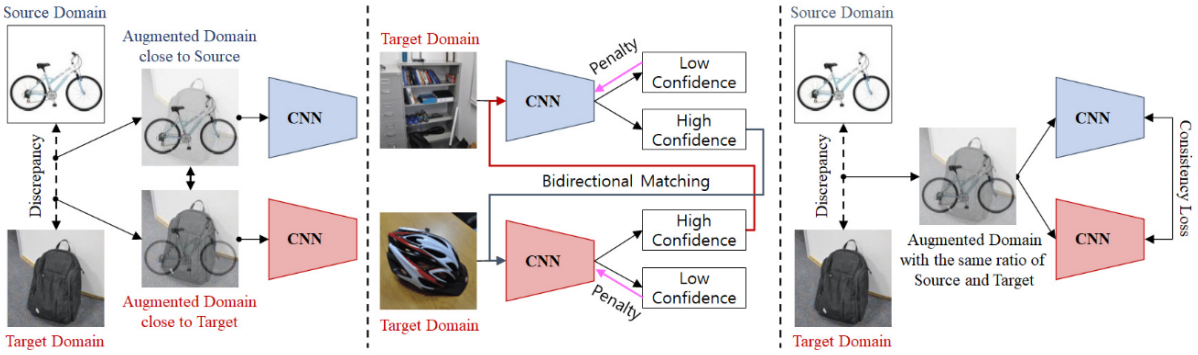

Figure 3: All parts of FixBi training model: (a) fixed ratio-based mixup, (b) confidence-based learning and (c) consistency regularization.

The SDM provides robust supervision for the source domain but relatively weak supervision for the target domain, while the TDM has strong supervision for the target domain but weaker supervision for the source domain. Thus, denoting $p(y|\tilde{x}_i^{st})$ as a predicted class distribution, it is defined fixed ratio-based mixup function

$$L_{fm} = \frac{1}{B} \sum_{i=1}^{B} \hat{y}_i^{st} \log(p(y|\tilde{x}_i^{st})),$$

where $\hat{y}_i^{st} = \arg\max p(y|\tilde{x}_i^{st})$ and B is the size of a mini-batch. In order to have connections between source and target domains, it is suggested to use a confidence-based learning approach whereby one model educates the other using positive pseudo-labels, or penalties itself using negative pseudo-labels. Positive pseudo-labels means labels which predictions are above a specific threshold, then the authors use them in training the second model by utilizing a conventional cross-entropy loss. Thus, denote p and q as distributions of two models, the authors get the following loss function

$$L_{bim} = \frac{1}{B}\sum_{i=1}^{B} \mathbb{1}(\max(p(y|x_i^t) > \tau)\hat{y}_i^t \log(q(y|x_i^t)),$$

where $\hat{y}_i^t = \arg\max p(y|x_i^t)$. In contrast, a negative pseudo-label refers to the top-1 label predicted by the network with a confidence below the threshold τ. The function of self-penalization is defined as follows:

$$L_{sp} = \frac{1}{B}\sum_{i=1}^{B} \mathbb{1}(\max(p(y|x_i^t) < \tau)\hat{y}_i^t \log(1 - p(y|x_i^t))$$

Furthermore, the threshold is changed adaptively during training. In addition, it is introduced the following expression:

$$L_{cr} = \frac{1}{B}\sum_{i=1}^{B} \|p(y|\tilde{x}_i^{st}) - q(y|\tilde{x}_i^{st})\|_2^2$$

that represents consistency regularization to guarantee a stable convergence during the training of both models.

1.2.4 | Fourth article

Now we are going to the next article *"Spherical Space Domain Adaptation with Robust Pseudo-label Loss"* written by Xiang Gu, Jian Sun, and Zongben Xu. [3] The authors propose a spherical space representation of data, which allows them to get more effective feature extraction and better adaptation across domains. One approach associated with increasing performance with differences in data distribution between source and target domain is to use pseudo-labels. However, the use of pseudo-labels can be problematic in the presence of noisy or incorrect labels. To tackle this problem, the authors map the data to a high-dimensional sphere and introduce a new loss function, called the robust pseudo-label loss, which is designed to address the problem of noisy or incorrect labels in the target domain.

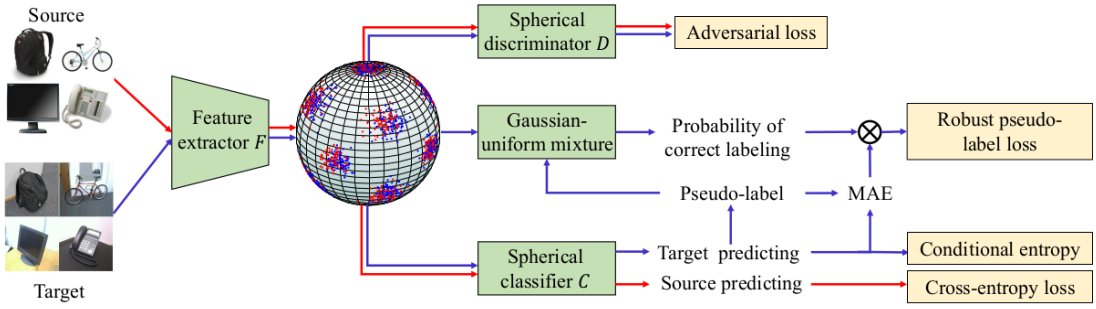

Figure 4: Robust Spherical Domain Adaptation (RSDA) method proposed by the authors consists of a feature extractor F, which is a deep convolutional network, used to extract features that are then embedded onto a sphere, a spherical classifier and a discriminator that predict class labels and domain labels, respectively. Target pseudo-labels and features passing through the Gaussian-uniform mixture model are used to estimate the posterior probability of correct labeling.

In the Figure 4 we can see spherical domain adaptation method. Domain invariant features are learned by adversarial training, entirely in the spherical feature space. The feature

extractor F is utilized to normalize the features to map onto a sphere. The classifier C and discriminator D are defined in the spherical feature space, consisting of spherical perceptron layers and a spherical logistic regression layer (see Figure 5).

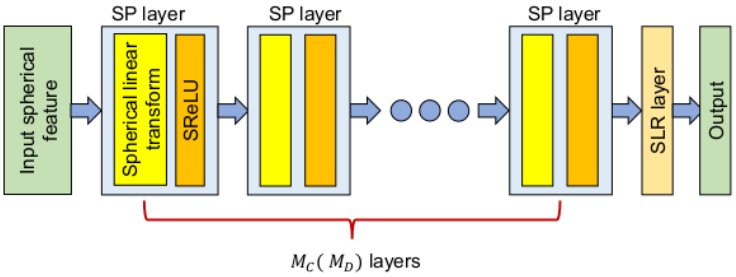

Figure 5: Spherical neural network structure

Although the use of a spherical element reduces feature dimension by one, it simplifies the domain adaptation process by eliminating differences in norms. The authors define spherical adversarial training loss as follows:

$$L = L_{bas}(F, C, D) + L_{rob}(F, C, \phi) + \gamma L_{ent}(F)$$

This lost consists of three parts: basic loss, robust pseudo-label loss and conditional entropy loss. Let's start with the first one. To align features, the authors utilize basic loss which is defined as an adversarial domain adaptation loss:

$$L_{bas}(F, C, D) = L_{src}(F, C) + \lambda L_{adv}(F, D) + \lambda' L_{sm}(F),$$

where L_{src} is a cross-entropy loss for the source domain, L_{adv} is an adversarial training loss and L_{sm} is a semantic loss. The second one is conditional entropy loss which is used to keep the learned features away from the classification boundary:

$$L_{ent}(F) = \frac{1}{N_t} \sum_{j=1}^{N_t} H(C(F(x_j^t))),$$

where $H(\cdot)$ denotes the entropy of a distribution. Additionally, the authors propose robust pseudo-label loss to increase robustness of the model. Denote $\tilde{y}_j^t = \arg\max_k [C(F(x_i^s))]_k$ as a pseudo-lebel of x_j^t where $[\cdot]_k$ means the k-th element. To be ensured in precision of pseudo-labels, it is assumed to use new random variable $z_j \in \{0, 1\}$ for each pair $(x_j^t, \tilde{y}_j^t)$ that specify the correctness of the data (1 is correct, 0 is not). Let the probability of correct labeling be $P_\phi(z_j = 1 | x_j^t, \tilde{y}_j^t)$ and ϕ to its parameters, then robust loss is defined as follows:

$$L_{rob}(F, C, \phi) = \frac{1}{N_0} \sum_{j=1}^{N_t} w_\phi(x_j^t) \mathcal{J}(C(F(x_j^t)), \tilde{y}_j^t),$$

where $N_0 = \sum_{j=1}^{N_t} w_\phi(x_j^t)$ and $\mathcal{J}(\cdot, \cdot)$ is mean absolute error (MAE). The function $w_\phi(x_j^t)$ is defined using the posterior probability of correct labeling

$$w_\phi(x_j^t) = \begin{cases} \gamma_j, & \text{if } \gamma_j \geq 0.5, \\ 0, & \text{otherwise,} \end{cases}$$

where $\gamma_j = P_\phi(z_j = 1|x_j^t, \tilde{y}_j^t)$. By utilizing a Gaussian-uniform mixture model in spherical space based on pseudo-labels, the authors model the probability $P_\phi(z_j = 1|x_j^t, \tilde{y}_j^t)$ as a function of the feature distance between the data and the center of the corresponding class. Thus, samples from target domain with a probability of correct labeling below 0.5 can be discarded. For further details of computing posterior probability, please refer to the article [3].

1.2.5 | Fifth article

Next, we move on to the article *"Domain Adaptation with Invariant Representation Learning: What Transformations to Learn?"* written by Stojanov, Petar, et al. [12] The researchers focus on the conditional shift scenario, where the data-generating process is utilized to (**i**) explain why two distinct encoding functions are required to infer the latent representation, (**ii**) improve an implementation of these functions, and (**iii**) impose meaningful structure on the latent representation Z to increase prediction accuracy in the target domain.

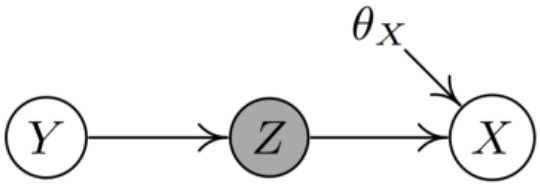

Figure 6: Data-generating process under conditional shift

Let's consider the data-generating process shown in the Figure 6 to understand what information is required for learning. The label Y is generated first from its prior distribution $P(Y)$. Then, the invariant representation Z is generated from Y through $P(Z|Y)$, and X is generated from $P(X|Z;\theta_X)$, where θ_X represents the changing parameters of $P(X|Y)$ across domains. We can consider Z as a latent representation of our data. The variable θ_X may correspond to environment-specific changes that are irrelevant for predicting the class Y. Generally speaking, Z is conditionally dependent on θ_X given X, although they may be marginally independent. Therefore, to recover Z given X, the information of θ_X should also be considered in the transformation (see detailed in the article to understand clearly how authors measure the influence of θ_X). The authors made two key observation associated with the data-generating process. Firstly, the encoder function ϕ requires θ_X as an input in addition to X. Secondly, assuming that θ_X has minimal influence on the relationship between X and Z, allowing us to use a single encoder $\phi(X, \theta_X)$ instead of two separate encoders. A decoder function $\tilde{\phi}$ that restricts the influence of θ_X, acting as a regularizer on the encoder ϕ, in order to retain important semantic information.

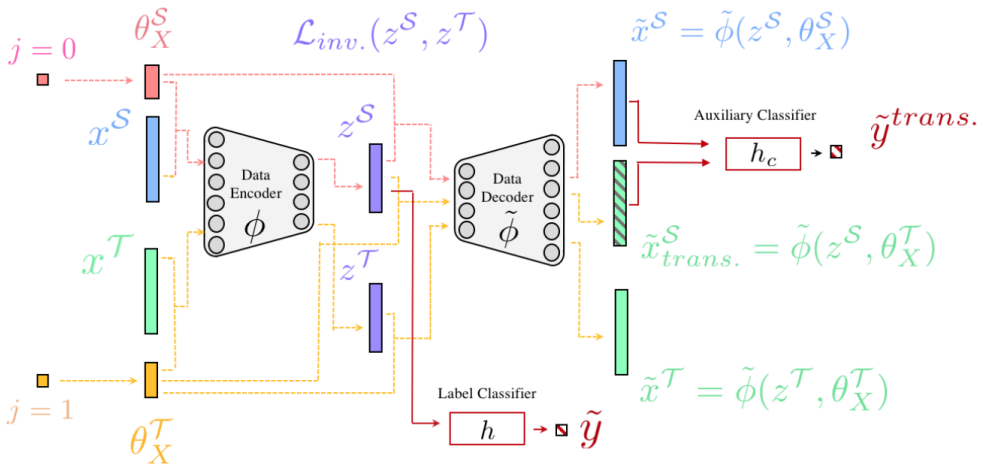

Figure 7: Autoencoder framework used in this article

Thus, the authors proposed a domain-adaptation network, which is shown in the Figure 7, where $\theta_X \in \{\theta_X^S, \theta_X^T\}$ parameters for source and target domains respectively.

1.2.6 | Sixth article

The main purpose of the paper *"Domain Adaptation for Semantic Segmentation via Class-Balanced Self-Training"* written by Zou, Yang, et al. [17] propose a new UDA framework for semantic segmentation based on iterative self-training procedure. A novel technique, referred to as Class-Balanced Self-Training (CBST), has been suggested by the authors, which aims to adapt the segmentation model from the source domain to the target domain by leveraging unlabeled target data. In the Figure 8, the authors present a structure and the results of their deep self-training framework using two datasets: GTA 5 [10] and Cityscapes [1].

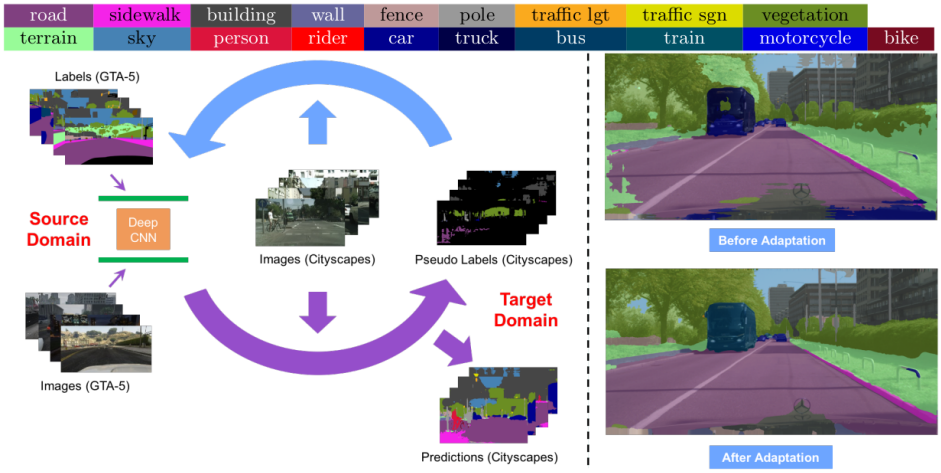

Figure 8: On the left side, the self-training framework for UDA is presented. On the right side, obtained results before and after adaptation for the Cityscapes dataset.

The CBST approach is based on two main components: a class-balancing strategy and a self-training algorithm. The class-balancing strategy aims to address the problem of

class imbalance between the source and target domains, which can negatively impact the performance of the segmentation model. The authors change the loss function using parameters that determine the proportion of selected pseudo-labels due to balance the class distribution during the self-training process. Furthermore, when the images in the source and target domains are similar, spatial prior knowledge can be effectively utilized to adapt models. For this purpose, the authors count the class frequencies in the source domain using Gaussian kernel. The experimental results show that the CBST approach outperforms several state-of-the-art unsupervised domain adaptation methods for semantic segmentation.

2 | Second part

In this part, we start with description of different datasets that are commonly used in transfer learning. Then, we will continue with implementation details and experiments that have been conducted.

2.1 | Datasets

The most popular datasets are **Office-31**, **ImageCLEF-DA**, **Office-Home**, **DomainNet** and **VisDA-2017**. Detailed discussion of each of them is given below:

- **Office-31** [11] consists of 4,110 images categorized into 31 classes, which are distributed across three separate domains: Amazon (A), Webcam (W), and Dslr (D).
- **ImageCLEF-DA**, utilized in [5], includes three distinct domains: Caltech-256 (C), ImageNet ILSVRC 2012 (I), and Pascal VOC 2012 (P). There are 600 images in each domain and 50 images for each category.
- **Office-Home** [13], includes four absolutely different domains: Artistic images (Ar), Clip Art (Cl), Product images (Pr) and Real-World images (Rw). This dataset contains 15 500 images in 65 object classes, which makes it more complex than Office-31.
- **VisDA-2017** [8] consists of 12 classes shared between two very different domains: Synthetic and Real. It contains synthetic images (training set) and real-world images (test set). The dataset was designed to have a large domain gap, which makes it a challenging benchmark for domain adaptation methods.
- **DomainNet** [7] is a large-scale visual recognition dataset designed to evaluate domain adaptation algorithms, which consists of almost 600 thousand images and includes 345 classes.

2.2 | Implementation details

At the beginning, it was necessary to start with some approaches to check their performance and have a possibility to compare results. Four different methods described in the papers have been chosen for the study:

- **Source only** is a method where a model is trained solely on the source domain data without any adaptation to the target domain.
- **Domain-Adversarial Neural Network (DANN)** is domain adaptation technique that aims to learn a domain-invariant feature representation by aligning the feature distributions of the source and target domains. The architecture of DANN consists of three components: a feature extractor network, a label predictor network, and a domain classifier network, and is described in more details in section 1.2.1.
- **Moving Semantic Transfer Network (MSTN)** The key idea behind MSTN is to add semantic transfer loss to the DANN approach. In the section 1.2.2, it is proposed to use average centroid alignment for aligning the feature distributions of the source and target domains. The architecture is the same as in the DANN method.

- **FixBi** is the approach described in details in the section 1.2.3. The main idea is to train two neural networks, allowing models to learn from each other or on their own results. For this purpose, the authors add bidirectional matching and self-penalization losses.

CNN architectures. For all approaches, pretrained Resnet50 [4] is utilized as the backbone network. The weights for the neural network can be downloaded from this link. Resnet50 has been pretrained on large image datasets such as ImageNet, which means that the network has already learned to recognize a wide range of features in images. Resnet50 is a convolutional neural network architecture consisting of 50 layers. This is a variant of the Resnet family of neural networks, which are designed to solve the vanishing gradient problem in deep neural networks. Resnet networks achieve this by using short connections between layers, which allow gradients to move more easily during backpropagation. Resnet50 is a widely used architecture in many articles, which makes it a good choice for research.

Resnet50 is used as a *Feature extractor* in all considering methods. *Label predictor* is a simple network that consists of two fully connected layers ($2048 \to 256 \to$ *number of classes*). *Domain classifier* architecture represents four fully connected layers ($2048 \to 1024 \to 256 \to 32 \to$ *number of domain* $= 2$) with a ReLU activation function and dropouts between each two fully connected layers. Using dropouts can reduce the sensitivity of the model to specific features in the input data and encourage the model to learn more generalizable features. This can lead to better performance on new, unseen data and can prevent overfitting.

Learning rate schedulers. Learning rate is an important hyperparameter that determines the step size at which the optimizer updates the model's parameters during training. There are many of them and it can be challenging to find the optimal learning rate, as setting it too high can cause the model to diverge, while setting it too low can slow down the learning process. Thus, in this study it is utilized two different learning rate schedulers: *CustomLRScheduler* and *CosineAnnealingLR*. The implementation of the first one follows the rules that are described in [2]

$$\eta_p = \frac{\eta_0}{(1 + \alpha \cdot p)^\beta},$$

where p linearly increases from 0 to 1, and the values η_0, α, and β are set to 0.01, 10, and 0.75, respectively. The second one, *CosineAnnealingLR* is a popular learning rate scheduler utilized in deep learning. It systematically reduces the learning rate over multiple epochs in a cyclical manner. Initially, the learning rate starts at its maximum value and then gradually decreases until it reaches the minimum value. Upon reaching the minimum value, the cycle restarts, and the learning rate returns to its maximum value. This process continues until the end of the training, which is usually determined by the total number of epochs or a predefined stop criterion. By starting with a higher learning rate and gradually decreasing it, the model can avoid getting stuck in local minima and converge to a better global minimum. The formula for the *CosineAnnealingLR* scheduler is:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min}) \left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right),$$

where η_{max} is your initial learning rate, η_{min} – minimum learning rate value, T_{cur} is the number of epochs since the last start, T_{max} – the total number of epochs. More detailed information about *CosineAnnealingLR* can be found here.

Optimizers. This study uses two popular optimization algorithms - stochastic gradient descent (*SGD*) and adaptive moment estimation (*Adam*). Both algorithms are commonly employed in deep learning to optimize the parameters of a neural network and improve its performance.

SGD is a simple and popular optimization algorithm that updates the weights of a model in the direction of the negative gradient of the loss function. One limitation of *SGD* is that it can get stuck in local minima and struggle with noisy or sparse gradients. To tackle this problem, several modifications can be used. In PyTorch, the *SGD* optimizer has several hyperparameters that can be tuned to improve its performance. The following parameters (except learning rate) are considered in this study:

- **Momentum** is a hyperparameter that determines how much past gradients affect the current gradient update. It helps to minimize the impact of the noise and fluctuations in the gradient updates. However, setting the momentum too high can also lead to slower convergence.
- **Weight decay** is a form of L2 regularization that adds a penalty term to the loss function during training. This penalty term is proportional to the square of the weights in the network, which encourages the model to use smaller weights and reduce overfitting.
- **Nesterov momentum** is a variant of momentum that takes into account the momentum term in the calculation of the gradient. This can help to reduce oscillations and improve convergence rates, especially in high-dimensional optimization problems.

Adam is another optimization algorithm that is commonly used in deep learning. It is an extension of SGD. The key idea behind Adam is to maintain a separate adaptive learning rate for each parameter in the network, based on estimates of the first and second moments of the gradients. This makes Adam more effective than SGD for optimization problems with noisy or sparse gradients. However, it may not always be the best choice for every task and model architecture, so it's important to experiment with different optimization algorithms and settings to find the best approach for your specific problem. *Adam* is considered in this study with default parameters, more information about the implementation and usage can be found at this link.

Pytorch Lightning. PyTorch Lightning is a lightweight PyTorch wrapper that allows users to focus on the high-level design of their experiments and models, instead of dealing with the low-level implementation details. It provides a structured way to organize PyTorch code, making it easier to read and maintain.

PyTorch Lightning offers a range of benefits that make it a good choice for deep learning researches. Firstly, it offers a modular design that makes it easy to organize code. It gives you a convenient and user-friendly interface to manage and run experiments. Moreover, all these benefits can help to improve your productivity. Secondly, PyTorch Lightning makes

it easier to scale models to multiple GPUs, which can significantly reduce training times for large models. Finally, it is flexible and can be easily integrated with other PyTorch libraries. Overall, PyTorch Lightning is an excellent choice for researchers who want to focus on the research aspect of deep learning and leave the engineering components to the library. More information can be found on the official website.

Weights and Biases. Weights and Biases (WandB) is a platform that provides a suite of tools to help developers and data scientists track and visualize their machine learning experiments. WandB makes it easy to log, keep track of your progress and compare different experiments, visualize model performance, and collaborate with team members. One of the main advantages of WandB is its integration with popular machine learning frameworks such as TensorFlow, PyTorch, and Keras. This means that you can easily log and track your model's hyperparameters and performance metrics during training and evaluation. Moreover, WandB is a cloud-based platform, which means that users can access their experiments and data from anywhere with an internet connection and also share them with colleagues and co-workers. For more detailed information, it is recommended to visit the official website.

Batch size. Different domains in your dataset can contain different number of images that makes your training process more complicated. To tackle this problem, it is proposed two approaches.

The first one is to find the ratio of the smaller dataset size to the larger one and concatenate the smaller dataset multiple times to ensure that the number of batches is aligned during the training loop. However, it is important to emphasize that with this approach, overfitting can occur if the appropriate number of epochs is not established. This is because the smaller dataset will be fed into the model more times than the larger one (depends on the ratio).

The second approach involves varying the number of images taken per batch for each domain. Applying this approach, it becomes possible to avoid concatenating the smaller dataset multiple times, which effectively reduces the amount of memory consumed. It is crucial to carefully consider the number of images per batch, as choosing a value that is either too high or too low can have negative consequences.

OmegaConf. OmegaConf is a library for Python that provides a convenient and flexible way to manage complex configurations in machine learning projects. It is designed to provides a number of features that can help to simplify the configuration process. Here are some reasons why OmegaConf can be a good choice:

- It is easy to use and allows developers to define nested configurations and easily access and modify configuration values.
- OmegaConf supports a wide range of configuration formats, including YAML and JSON. This makes it flexible and easy to integrate in your project.
- It supports type checking, which can help to catch configuration errors and improve code quality.

To sum up, OmegaConf can be a good choice for Python developers who work on large and complex projects and want a flexible and powerful configuration system for their applications. Additional details can be found here.

2.3 | Experiments

The dataset **Office-31** is used to test the approaches. This dataset consists of three domains: Amazon (A) - 2817 images, Dslr (D) - 498 images and Webcam (W) - 795 images (see Figure 9).

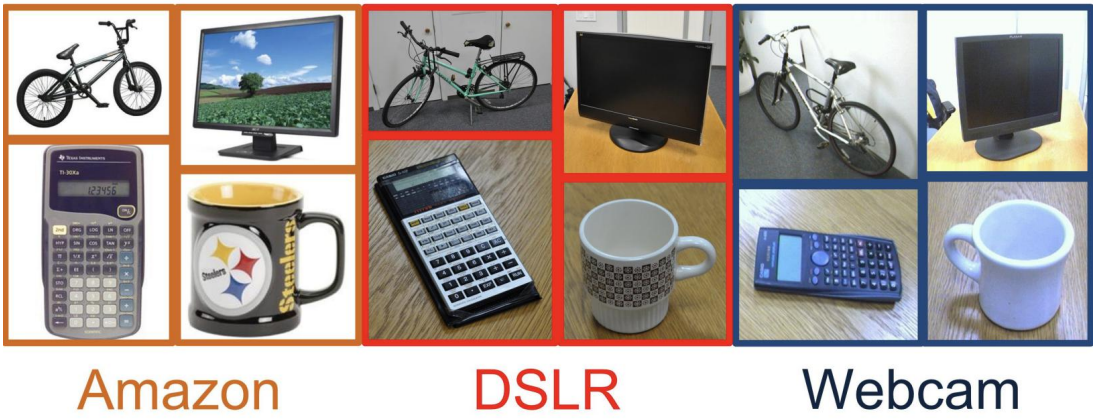

Figure 9: Different domains in dataset **Office-31**.

The existence of dissimilar image quantities ensures us in the importance of utilizing one of the approaches discussed in the previous section in order to avoid any information loss. In the first approach, where the smaller domain is concatenated, a batch size of 32 or 64 is utilized for all experiments. The second approach takes into account the size of each domain, and as a result, the batch sizes are utilized in the experiments according to the Table 1:

Table 1: The table of batch sizes is organized such that the numerical values in the first column correspond to the first letter, the second one to the second letter.

	Batch size	
AD	45	8
AW	45	13
DA	8	45
DW	20	32
WA	13	45
WD	32	20

For the all methods, two kinds of optimizers are used: SGD and Adam. However, the second one shows worse results with default parameters than SGD with lr $= 0.001$, momentum $= 0.9$, weight decay $= 0.0005$. The CustomLRScheduler and CosineAnnealingLR are both used as schedulers, but it has been found that the model performs better when using

the second one. Thus, all the following results have been obtained using the CosineAnnealingLR scheduler. Furthermore, all methods give the best results with an approach using a different number of images in each batch. As a result, this approach will be assumed by default, unless otherwise stated. For each two domains, at least three experiments were conducted for all methods, and the best results were selected.

Let's start with the first method – **Source only**. Here, the model is trained on the source domain and then tested on the target domain. The obtained results are shown in Table 2 (at the end of the 60th epoch).

Table 2: Accuracy on Office-31 for the **Source only** method.

Source	A→D	A→W	D→W	D→A	W→D	W→A
1	81.46	76.43	95.96	60.33	99.58	64.45
2	81.04	76.04	95.7	60.09	99.37	64.17
3	80.62	76.04	95.7	59.38	99.37	64.13
average	**81.04**	**76.17**	**95.79**	**59.93**	**99.44**	**64.25**

DANN is an architecture that consists not only of a feature extractor and label predictor, but also a domain classifier. This domain classifier helps to identify the domain of the input data and allows the model to learn domain-invariant features. The Table 3 below clearly demonstrates that the results for each of the two domains are superior to those obtained using the simple **Source only** method. The results are obtained at the end of the 60th epoch.

Table 3: Accuracy on Office-31 for the **DANN** method.

DANN	A→D	A→W	D→W	D→A	W→D	W→A
1	83.13	78.91	96.35	64.35	100	65.38
2	82.92	78.65	95.96	63.88	100	64.91
3	82.71	78.65	95.83	63.92	99.79	64.74
average	**82.92**	**78.74**	**96.05**	**64.05**	**99.93**	**65.01**

MSTN method is a complication of **DANN** by adding a semantic loss. To get this loss, we add centroids for each class and utilize the algorithm described in the section 1.2.2. In the Table 4, you can see the results that are acquired at the end of the 60th epoch. The quality of the results tends to suffer due to the significant influence of randomness. The selection of pictures that are included in a batch determines the movements of the centroids, ultimately influencing the overall quality to a significant extent.

Table 4: Accuracy on Office-31 for the **MSTN** method.

MSTN	A→D	A→W	D→W	D→A	W→D	W→A
1	77.71	72.53	92.06	33.38	99.79	51.07
2	76.88	72.4	91.8	32.1	99.79	48.58
3	76.67	71.48	91.41	34.09	99.79	48.65
average	**77.09**	**72.14**	**91.76**	**33.19**	**99.79**	**49.43**

FixBi is a method that addresses the domain adaptation problem by training two neural networks that can help each other. As it is described in the article [6], we define $\lambda_{sd} = 0.7$

and $\lambda_{td} = 0.3$. In this method, we cannot use the approach with different batch sizes due to the need to mix up images from source and target domain. Therefore, the second approach with concatenation is utilized. The model is trained for a combined duration of 150 epochs, with the first 100 epochs designated as the warm-up period. It is important to note that 150 epochs are used, not 200, because after the warm-up period the validation score stabilizes and almost does not change. After the warm-up period, L_{bim} starts to be applied, which leads to a critical changing in the total accuracy. The sudden improvement in accuracy can occur in either a positive or negative direction, and is often heavily influenced by randomness. One possible explanation for this phenomenon is that the model may have already found a local minimum prior to the introduction of L_{bim}, and the application of L_{bim} causes a sudden shift in gradients that propels the model out of the current minimum and into a new one. Depending on the new minimum, this can result in either an improvement or a deterioration in the model's performance. As we can see in the Figure 10, for Amazon (source) and Webcam (target) domains this method gives significant increase in accuracy, while for DSLR (source) and Amazon (target) it shows the worst results.

Figure 10: The total accuracy of **FixBi** model on Amazon and Webcam domains.

In the Figure 11, you can see the separate accuracy of "source-dominant model" (SDM) and "target-dominant model" (TDM) in case of Amazon (source) and Webcam (target) domains.

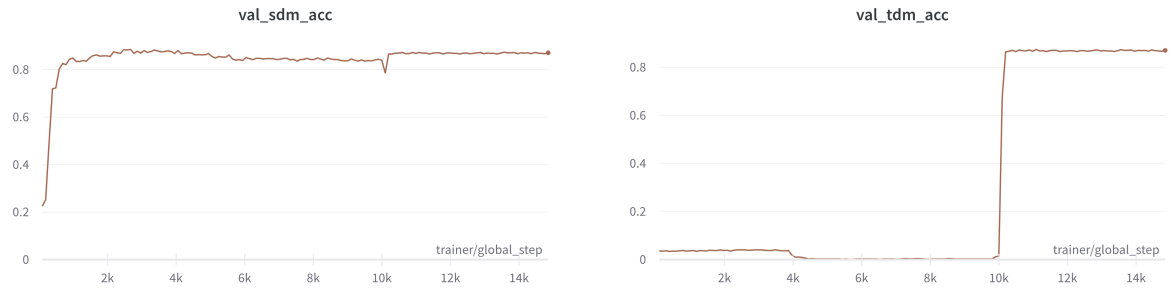

Figure 11: The separate accuracy of **FixBi** "source-dominant model" (SDM) on the left side and "target-dominant model" (TDM) on the right side.

The results for each two domains of Office-31 dataset for **FixBi** method are shown in Table 5.

Table 5: Accuracy on Office-31 for the **FixBi** method.

FixBi	A→D	A→W	D→W	D→A	W→D	W→A
1	76,88	87,11	94,66	23,19	94,38	30,15
2	70,1	86,72	90,89	15,23	94,38	26,03
3	77,08	81,77	88,54	16,01	92,92	20,63
4	73,96	81,38	92,78	17,8	92,88	23,45
average	**74.51**	**84.25**	**91.72**	**18.06**	**93.64**	**25.07**

The **FixBi** method was selected for modification, wherein the ratios for SDM and TDM were adjusted, and a domain classifier was added for mixup images. This modified method is named **DannFixBi**. λ_{sd} and λ_{td} are set as 0.9 and 0.6, respectively. As it was mentioned before, we use domain classifier for mixup images, hence our loss consists of two summands, where the first one is for the source domain and multiplied by λ_{sd} and the second one is for the target domain multiplied by λ_{td}. The Table 6 indicates that certain domains exhibit an increase in accuracy as a result of these changes.

Table 6: Accuracy on Office-31 for the **DannFixBi** method.

DannFixbi	A→D	A→W	D→W	D→A	W→D	W→A
1	86.87	85.81	97.53	39.7	99.37	46.95
2	86.46	86.07	97.35	39.45	98.98	46.95
3	87.29	86.07	97.44	39.35	99.17	46.95
average	**86.87**	**85.98**	**97.44**	**39.15**	**99.17**	**46.95**

In summary, Table 7 presents all the obtained results. The **DannFixBi** method yielded the highest accuracy for the A→D, A→W, and D→W tasks, while the **DANN** method achieves the best results for the D→A, W→D, and W→A tasks.

Table 7: Accuracy on Office-31 for all methods.

	A→D	A→W	D→W	D→A	W→D	W→A
Source	81.04	76.17	95.79	59.93	99.44	64.25
DANN	82.92	78.74	96.05	**64.05**	**99.93**	**65.01**
MSTN	77.09	72.14	91,76	33.19	99.79	49.43
FixBi	74.51	84.25	91.72	18.06	93.64	25.07
DannFixBi	**86.87**	**85.98**	**97.44**	39.15	99.17	46.95

3 | References

[1] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[2] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *International conference on machine learning*, pages 1180–1189. PMLR, 2015.

[3] Xiang Gu, Jian Sun, and Zongben Xu. Spherical space domain adaptation with robust pseudo-label loss. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9101–9110, 2020.

[4] 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.

[5] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Deep transfer learning with joint adaptation networks. In *International conference on machine learning*, pages 2208–2217. PMLR, 2017.

[6] Jaemin Na, Heechul Jung, Hyung Jin Chang, and Wonjun Hwang. Fixbi: Bridging domain spaces for unsupervised domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1094–1103, 2021.

[7] Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. Moment matching for multi-source domain adaptation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1406–1415, 2019.

[8] Xingchao Peng, Ben Usman, Neela Kaushik, Judy Hoffman, Dequan Wang, and Kate Saenko. Visda: The visual domain adaptation challenge. *arXiv preprint arXiv:1710.06924*, 2017.

[9] Ievgen Redko, Emilie Morvant, Amaury Habrard, Marc Sebban, and Younès Bennani. A survey on domain adaptation theory: learning bounds and theoretical guarantees. *arXiv preprint arXiv:2004.11829*, 2020.

[10] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14*, pages 102–118. Springer, 2016.

[11] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11*, pages 213–226. Springer, 2010.

[12] Petar Stojanov, Zijian Li, Mingming Gong, Ruichu Cai, Jaime Carbonell, and Kun Zhang. Domain adaptation with invariant representation learning: What transformations to learn? *Advances in Neural Information Processing Systems*, 34:24791–24803, 2021.

[13] Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5018–5027, 2017.

[14] Zirui Wang. Theoretical guarantees of transfer learning. *arXiv preprint arXiv:1810.05986*, 2018.

[15] Garrett Wilson and Diane J Cook. A survey of unsupervised deep domain adaptation. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(5):1–46, 2020.

[16] Shaoan Xie, Zibin Zheng, Liang Chen, and Chuan Chen. Learning semantic representations for unsupervised domain adaptation. In *International conference on machine learning*, pages 5423–5432. PMLR, 2018.

[17] Yang Zou, Zhiding Yu, BVK Kumar, and Jinsong Wang. Unsupervised domain adaptation for semantic segmentation via class-balanced self-training. In *Proceedings of the European conference on computer vision (ECCV)*, pages 289–305, 2018.