

Domain Adaptation with a Classifier Trained by Robust Pseudo-Labels

Yunke Zhou

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Paul E. Plassmann, Chair

Creed F. Jones III

Ruoxi Jia

December 6, 2021

Blacksburg, Virginia

Keywords: Robust pseudo-label, Gaussian-uniform mixture model, Close-set unsupervised
domain adaptation

Copyright 2022, Yunke Zhou

Domain Adaptation with a Classifier Trained by Robust Pseudo-Labels

Yunke Zhou

(ABSTRACT)

With the rapid growth of computing power, approaches based on deep learning algorithms have achieved remarkable results in solving computer vision classification problems. These performance improvements are achieved by assuming the source and target data are collected from the same probability distribution. However, this assumption is usually too strict to be satisfied in many real-world applications, such as big data analysis, natural language processing, and computer vision classification problems. Because of distribution discrepancies between these domains, directly training the model on the source domain cannot be expected to generate satisfactory results on the target domain. Therefore, the problem of minimizing these data distribution discrepancies is the main challenge with which modern machine learning is now faced. To address this problem, domain adaptation (DA) aims to identify domain-invariant features between two different but related domains. This thesis proposes a state-of-the-art DA approach that overcomes the limitations of traditional DA methods. To capture fine-grained information for each category, I deploy centroid-to-centroid alignment to perform domain adaptation. An Exponential Moving Average strategy (EMA) is used to ensure we can form robust source and target centroids. A Gaussian-uniform mixture model is trained using an Expectation-Maximization (EM) algorithm to infer the robustness of the target pseudo-labels. With the help of target pseudo-labels, I propose two novel types of classifiers: (1) a target-oriented classifier (TO); and (2) a centroid-oriented classifier (CO).

Extensive experiments show that these two classifiers exhibit superior performance on a variety of DA benchmarks when compared to standard baseline methods.

Domain Adaptation with a Classifier Trained by Robust Pseudo-Labels

Yunke Zhou

(GENERAL AUDIENCE ABSTRACT)

Approaches based on deep learning algorithms have achieved remarkable results in solving computer vision classification problems. These performance improvements are achieved by assuming the source and target data are collected from the same probability distribution; however, in many real-world applications, such as big data analysis, natural language processing, and computer vision classification problems, this assumption is usually too strict to be satisfied. For example, these two domains may have the same types of classes, but the objects in each category of these different domains can vary in shape, color, background, or even illumination. Because the probability distributions are slightly mismatched, directly training the model on one domain cannot achieve a satisfactory result on the other domain. To address this problem, domain adaptation (DA) aims to extract common features on both domains to transfer knowledge from one domain to another. In this thesis, I propose a state-of-the-art DA approach that overcomes the limitation of the traditional DA methods. To capture the low-level information of each category, I deploy centroid-to-centroid alignment to perform domain adaptation. An Exponential Moving Average (EMA) strategy is used to ensure the generation of robust centroids. A Gaussian-Uniform Mixture model is trained by using the Expectation-Maximization (EM) algorithm to infer the robustness of the target sample pseudo-labels. With the help of robust target pseudo-labels, I propose two novel

types of classifiers: (1) a target-oriented classifier (TO); and (2) a centroid-oriented classifier (CO). Extensive experiments show that the proposed method outperforms traditional baseline methods on various DA benchmarks.

Dedication

To my family

Contents

List of Figures	x
List of Tables	xiv
1 Introduction	1
1.1 Background	1
1.2 Terminology Explanation	2
1.3 Contributions	3
1.4 Thesis Structure	4
2 Review of Literature	6
2.1 Theory Foundation of Domain Adaptation	6
2.1.1 How transferable are features in deep neural networks?	6
2.2 Statistical Style Domain Adaptation approaches	10
2.2.1 A kernel two-sample test	10
2.2.2 Deep Domain Confusion: Maximizing for Domain Invariance (DDC)	12
2.2.3 Transferable Representation Learning with Deep Adaptation Networks (DAN)	13
2.2.4 Deep Subdomain Adaptation Network for Image Classification (DSAN)	15

2.2.5	Conclusion for statistical domain adaptation approaches	16
2.3	Adversarial Style Domain Adaptation approaches	17
2.3.1	Domain-adversarial training of neural networks (DANN)	18
2.3.2	Multi-adversarial domain adaptation (MADA)	19
3	Ablation Studies toward DAN Model	20
3.1	Results with AlexNet	21
3.2	Results with ResNet	24
3.3	Results with semi-supervised transfer learning	25
4	Methodology	27
4.1	Introduction	27
4.2	Workflow and architecture of the proposed method	30
4.2.1	An Overview of the Training Algorithm	30
4.2.2	Training the Initial Model	32
4.2.3	Training the Gaussian-Uniform Mixture Model	33
4.2.4	A Centroid-Oriented Classifier	36
4.2.5	A Target-Oriented Classifier	37
5	Experiments	39
5.1	DA Benchmarks	39
5.2	Baseline methods	40

5.3	Implementation details	41
5.4	Experimental Results	42
5.5	Ablation Study	44
5.6	Choice of the Filtering Threshold Parameter τ	46
6	Conclusions	48
	Bibliography	49

List of Figures

2.1	Experimental result from Yosinski et al. [25]	9
2.2	Network architecture of DDC [22]	13
2.3	Network architecture of DAN [15]	14
3.1	Samples from Office-31 Dataset	21
3.2	AlexNet-1: Single adaptation layer (FC8)	22
3.3	AlexNet-2: Double adaptation layers (FC7 FC8)	23
3.4	AlexNet-3: Triple adaptation layers (FC6 FC8)	23
3.5	Unsupervised transfer learning with ResNet	24
3.6	Unsupervised transfer learning vs. Semi-supervised transfer learning	25

4.1 In these illustrations, the large circles represent the domains (for the global alignment figure, or subdomains for the class-to-class illustrations)—either the target or source domains. The solid triangles and circles represent individual samples from two classes. Samples from the source domain are colored blue, whereas samples from the target domain are colored orange. **Top left:** global alignment might do well at aligning the domains, but by failing to capture fine-grained information and it might not do an excellent job at separating the classes. **Top right:** class-to-class alignment can exploit local information, namely pseudo-labels; however, since some of these pseudo-labels may be incorrect, using these incorrectly labeled samples to train the network results in an imperfect class-to-class alignment. **Bottom:** Before performing the adaptation, if we can filter out the target samples that might be incorrectly labeled (based on the robustness of their pseudo-labels), we can potentially improve the accuracy of the class-to-class alignment. 29

4.2	A graphical representation of the training steps of our proposed method. In the figure, the red and black arrows represent the computational flow of the target and source domain, respectively. I introduce two training models: M_{init} , and M_{train} . Each network is separated into two modules: a feature extractor $G_{init/train}$, and a classifier $F_{init/train}$. In the first step , I initialize M_{init} by performing a centroid-to-centroid alignment. Upon completing the training of M_{init} , in the second step , I form target centroids by utilizing the target features and the target pseudo-labels generated by M_{init} . Based on the target pseudo-labels and the cosine distance between target features and the corresponding target centroid, I use the Expectation-Maximization (EM) algorithm to train a Gaussian-uniform mixture model and locally store the resulting posterior probabilities of correct labeling. In the third step , I have developed two types of training models M_{train} : a target-oriented classifier and a centroid-oriented classifier. For the first training model, I treat the target pseudo-labels with robustness greater than a filtering threshold τ as the ground-truth labels and employ these labels to a weighted cross-entropy loss function. I use target samples with robustness greater than τ to form target centroids for the second training model. Based on the target feature distance to each of the target centroids, I generate a new pseudo-label for each target sample and employ these labels to the standard cross-entropy loss function.	31
5.1	Sample Images from Office-31 Dataset	40
5.2	Sample Images from Office-Home Dataset	40

5.3	The t-SNE visualization of the extracted features for target-oriented classifier with (a) and without (b) robust target pseudo-labels for the A→W scenario.	45
5.4	The observed accuracy for the Office-31 dataset using the TO classifier as function of the filtering threshold parameter τ .	47

List of Tables

5.1	ACCURACY (%) on Office-31 dataset for unsupervised domain adaptation (RESNET-50)	43
5.2	ACCURACY (%) on Office-Home dataset for unsupervised domain adaptation (RESNET-50)	43
5.3	ACCURACY (%) on the Office-31 dataset for the ablation study (RESNET-50)	45
5.4	ACCURACY (%) on the Office-Home dataset for the ablation study (RESNET-50)	45

List of Abbreviations

- δ : Normalization parameter of a uniform distribution
- $\mathcal{N}^+$: Gaussian distribution
- $\mathcal{U}$: Uniform distribution
- ϕ : Set of parameter of Gaussian-uniform mixture model
- π : Prior probability of getting correct labeling
- ρ : Trade of parameter to ensure centroid can be updated gradually
- σ : The covariance of the Gaussian distribution
- τ : The filtering threshold to determine if a pseudo-label is robust
- Υ : The posterior probability of correct labeling given the target feature distance and its pseudo-label
- $\tilde{y}^t$: Target pseudo-label
- B_t : Mini-batch size
- $C_{init_k}^{localT}$: Target local centroid. Superscript *localT* means it is a local target centroid. Subscript *init_k* means the centroid belongs to class k and it is generated by the feature extracted by the initial model M_{init}
- $C_{init_k}^T$: Global target centroid
- F_{init} : The classifier for the initial model

F_{train} : The classifier for the final model

G_{init} : The feature extractor for the initial model

G_{train} : The feature extractor for the final model

K : Number of classes

M_{init} : The initial model trained by step 1 of my algorithm

M_{train} : The final model trained by step 3 of my algorithm

$q1$: First-order centered data moment

$q2$: Second-order centered data moment

CDAN: Conditional Adversarial Network

CO: Centroid-Oriented Classifier

DA: Domain Adaptation

DAN: Deep Adaptation Network

DANN: Domain-Adversarial Training of Neural Networks

DDC: Deep Domain Confusion

DSAN: Deep Subdomain Domain Confusion

EM: Expectation-Maximization Algorithm

EMA: Exponential Moving Average Strategy

FC: Fully-Connected Layer

iCAN: Collaborative and Adversarial Network

JAN: Joint Adaptation Network

MADA: Multi-Adversarial Domain Adaptation

MMD: Maximum Mean Discrepancy

SAFN: Stepwise Adaptive Feature Norm

SGD: Stochastic Gradient Descent

TO: Target-Oriented Classifier

UDA: Unsupervised Domain Adaptation

Chapter 1

Introduction

1.1 Background

Recently, deep learning approaches have played an increasingly significant role in solving computer vision problems. As the data diversity grows, transferring and re-utilizing supervised knowledge across different domains becomes increasingly more essential and inherently more challenging. A typical example would be transferring the knowledge from a large-scale dataset into a new small-scale dataset. With the co-existence of the large and small-scale datasets, we expect our deep learning model to have a similar performance on both of them. However, this goal is hard to achieve since these two datasets usually are not collected from an identical probability distribution. For example, the large-scale dataset could be images or videos that we can easily find from the internet. This data is usually of high quality and contains label information. On the contrary, the small-scale dataset might be the images or videos captured by a security camera. This data is often of low quality and does not have label information associated with it. To train a model based on these low-quality inputs, the training process will be time-consuming, and the final result will be unsatisfactory. At that moment, the value of the large-scale dataset starts to shine. We would like to find a way to utilize the existing knowledge extracted from the large-scale dataset to compensate for the disadvantages of the small-scale dataset.

Additionally, providing sufficient labeled data for each specific task is time-consuming and

expensive in many real-world applications. To mitigate the dependency of obtaining enhanced large-scale, labeled datasets, people have come up with an idea called transfer learning/domain adaptation (DA) [17]. The goal here is to transfer the knowledge from a specific domain or task into a different but related area. The core idea can be thought of as similar to how humans try to learn through analogy. For example, the ability to roller skate can be helpful in learning skiing, and likewise, the ability to walk can be beneficial in learning running.

1.2 Terminology Explanation

There are two important concepts involved in transfer learning: (1) domain and (2) task. We give definitions of these two terms below.

A *Domain* $\mathcal{D}$ consists of a d -dimensional feature space $\mathcal{X}$ and its corresponding marginal distribution $\mathcal{P}(x)$. Suppose there are n samples in the dataset, and each of these samples can be described by a d -dimensional feature x , then $\mathcal{X} = \{x_i\}_{i=1}^n$.

A *Task*, on the other hand, consists of a mapping function $f(x)$, and a corresponding labeled space $\mathcal{Y}$. Each of the data in the label space $\mathcal{Y}$ corresponds to the label of the sample. Therefore, $\mathcal{Y} = \{y_i\}_{i=1}^n$. For each sample, we need a mapping function to map the d -dimensional feature x into its label y . Therefore, the function f can be defined as: $f : x \rightarrow y$.

In short, $\mathcal{D} = \{X, \mathcal{P}(x)\}$, and $\mathcal{T} = \{\mathcal{Y}, f(x)\}$. The goal of transfer learning is to transfer the knowledge from the source domain and source task to compensate for the limitations of the target domain and target task. The source and target dataset usually are different but also related. Based on the similarity between their feature space, label space, marginal distributions, and mapping functions, transfer learning problems can be further divided into

many sub-problems. This thesis will only focus on the scenario in which the label space and feature space are equaled across domains. However, their marginal distributions and the mapping functions are different. This specific scenario is named Domain Adaptation (DA).

For all of the traditional methods shared in Chapter 2 and the proposed method described in Chapter 4, we will focus on solving domain adaptation problems with the following problem statement: Given a labeled source domain $D_s = \{(x_i^s, y_i^s)\}_{i=1}^{N_s}$ with N_s source samples and an unlabeled target domain $D_t = \{(x_j^t)\}_{j=1}^{N_t}$ with N_t target samples, the goal is to correctly label the target samples in D_t through training the model on $D_s \cup D_t$.

1.3 Contributions

Traditional domain adaptation methods focus on doing domain-to-domain alignment or class-to-class alignment to minimize the domains discrepancy. A standard approach performs a domain-to-domain alignment; however, this approach usually leads to unsatisfactory transfer learning results because the category information is not explicitly used. To address this problem, recently proposed approaches try to perform class-to-class alignment by utilizing the pseudo-labels assigned to each target data through a classifier trained by the source data. However, this classifier is inevitably biased to the source dataset because of differences between datasets.

This thesis addresses this limitation by training the model using both the source and target data in combination with robust target pseudo-labels. A Gaussian-uniform mixture model is incorporated into the design to infer the target pseudo-labels robustness. With the help of robust pseudo-labels, I propose two types of novel classifiers: (1) A target sample oriented classifier; and (2) A centroid oriented classifier. Extensive experimental results shown in Chapter 5 demonstrate that these two classifiers exhibit superior performance on a variety

of Domain Adaptation (DA) benchmarks when compared to standard baseline methods.

The main contributions of this thesis can be summarized as follows:

- (1) Discovering and analyzing the limitations of the traditional methods;
- (2) Using Gaussian-uniform mixture model to minimize the potential negative impact of incorrect labeling;
- (3) Designing and implementing two novel types of classifier to fully leverage the target pseudo-labels;
- (4) Transferring the problem setting from unsupervised learning into semi-supervised learning by incorporating the idea of pseudo-label into the design;
- (5) Testing the features transfer-ability when transferring the knowledge from a small-scale dataset to a large-scale dataset, and vice versa; and
- (6) Testing how the choice of the kernel function can potentially affect the final result.

1.4 Thesis Structure

The thesis chapters are organized as follows.

Chapter 1: In this chapter, I provide a brief introduction and motivations toward domain adaptation. In this chapter, I clarify the meaning of domain and task and specify the problem setting.

Chapter 2: In this chapter, I provide a literature review of the traditional domain adaptation methods, analyze their limitations, and review the evolution of these methods.

Chapter 3: In this chapter, I conduct ablation studies toward the DAN model (one of the

papers shared in Chapter 2, and build the foundation for Chapter 4.

Chapter 4: Based on the results from Chapter 3, I propose a new method and provide a technical explanation toward how the proposed method can overcome the limitations illustrated in Chapter 2.

Chapter 5: In this chapter, I present experimental results and conduct an ablation study to further verify the improved performance of the proposed methods.

Chapter 6: Conclude this thesis and propose future improvement.

Chapter 2

Review of Literature

In this section, I will first present the paper that builds theory on the foundation of domain adaptation. Then, I will go through the evolution of the domain adaptation approaches and analyze their contributions, feasibility, and limitations.

2.1 Theory Foundation of Domain Adaptation

2.1.1 How transferable are features in deep neural networks?

In this paper, Yosinski et al. [25] successfully found out that the features extracted from each layer of the deep neural network will eventually transition from general to specific as the depth of the hidden layer increases. This finding establishes the foundation and possibility of domain adaptation. Because the shallow layers of the neural network focus on extracting general features and the deeper layers focus on extracting task-specific features, it is intuitive to consider that the shallower layers are safe to transfer, whereas the deeper layers are not. However, Yosinski et al. [25] also clarified that there existed an inter-dependency between the consecutive hidden layers. Suppose two successive hidden layers have very strong inter-dependency; If we only transfer the information stored in the first layer, then the transferring knowledge is not completed and consequently causes network performance degradation. Yosinski et al. [25] found out that one way to overcome the problem mentioned

above is by fine-tuning the network using the target data. However, this is usually not applicable in many real-world problems because the data collected from the target domain is usually unlabeled, and fine-tuning the network needs labeled data to perform supervised learning methods. In the following paragraph, I will demonstrate his experiment setup and explain how he got the above conclusions.

To establish the experiment, Yosinski et al. [25] firstly created two different subsets A and B, by randomly splitting 1000 non-overlapping classes collected from the ImageNet. Each subgroup contains 500 classes with approximately 645,000 images in total. He used AlexNet [10], an eight-layer neural network, as the backbone and trained six different types of image classifiers. The first type of classifier (BaseA) used subset A as the training data. The second type of classifier (BaseB) used subset B as the training data. The third type of classifier (BnB) copied the first n th layers from BaseB and only fine-tuned the remaining layers using subset B. The fourth type of classifier (AnB) copied the first n th layers from BaseA and only fine-tuned the remaining layers using subset B. The fifth type of classifier (BnB⁺) copied the first n th layers from BaseB and fine-tuned the entire network using subset B. At last, the sixth type of classifier (AnB⁺) copied the first n th layers from BaseA and fine-tuned the entire network using subset B. Except for BaseA, all the remaining network performance was tested on the corresponding training subset, subset B.

Figure 2.1 is the experimental result provided in the paper. The graph's x-axis stands for the number of copied layers, and the y-axis stands for the top-1 accuracy. Without copying any layers, baseB can achieve approximate 62.4% accuracy (Base case). By taking a close look at the AnB case (red diamond shape), we notice that as the number of copied layers gets larger and larger, the performance becomes lower and lower. If we only transfer the shallow layers (layers 1~3), the network performance can remain stable. However, if we start transferring the deeper layers (layers 4~7), the network performance drops significantly. This

observation leads to the assumption that the shallow layers are more likely to extract the general features (therefore, they are safe to transfer). In contrast, the deeper layers are more likely to extract task-specific features (thus, need to perform adaptation before transferring).

Compared with AnB, AnB⁺ can achieve noticeable performance improvement by fine-tuning on all the network layers. This observation concludes that by performing fine-tune operations, we can overcome the limitations of the task-specific layers.

Next, let's take a look at the case of BnB. We can notice that the network performance drops when the numbers of copied layers range from 4 to 5. The interesting observation is that the network performance goes back to normal once we transfer these two layers. This observation shows that inter-dependency must exist between the fourth and fifth layers and between the fifth and sixth layers. If we only transfer the first part, then the inter-dependency between these layers is damaged and consequently causes performance degradation.

At last, let's take a look at the case of BnB⁺. We can observe that by fine-tuning the entire network, the inter-dependency between these layers can be rebuilt, and the network performance can go back to normal.

Based on the observations so far, we know that the shallower layers of the network are safe to transfer, whereas the deeper layers are not. When transferring the layers, we also need to consider the inter-dependency between each layer. In the ideal case, if we know the label of the target data, we can fine-tune the entire network to overcome the limitations mentioned above. However, fine-tuning the network using the target samples in real-world applications is hard to achieve. The main reason is that the target samples are usually unlabeled and not collected from the same probability distribution as the source data. Therefore, in such circumstances, how we can still safely transfer the deeper layer is the main focus of domain adaptation.

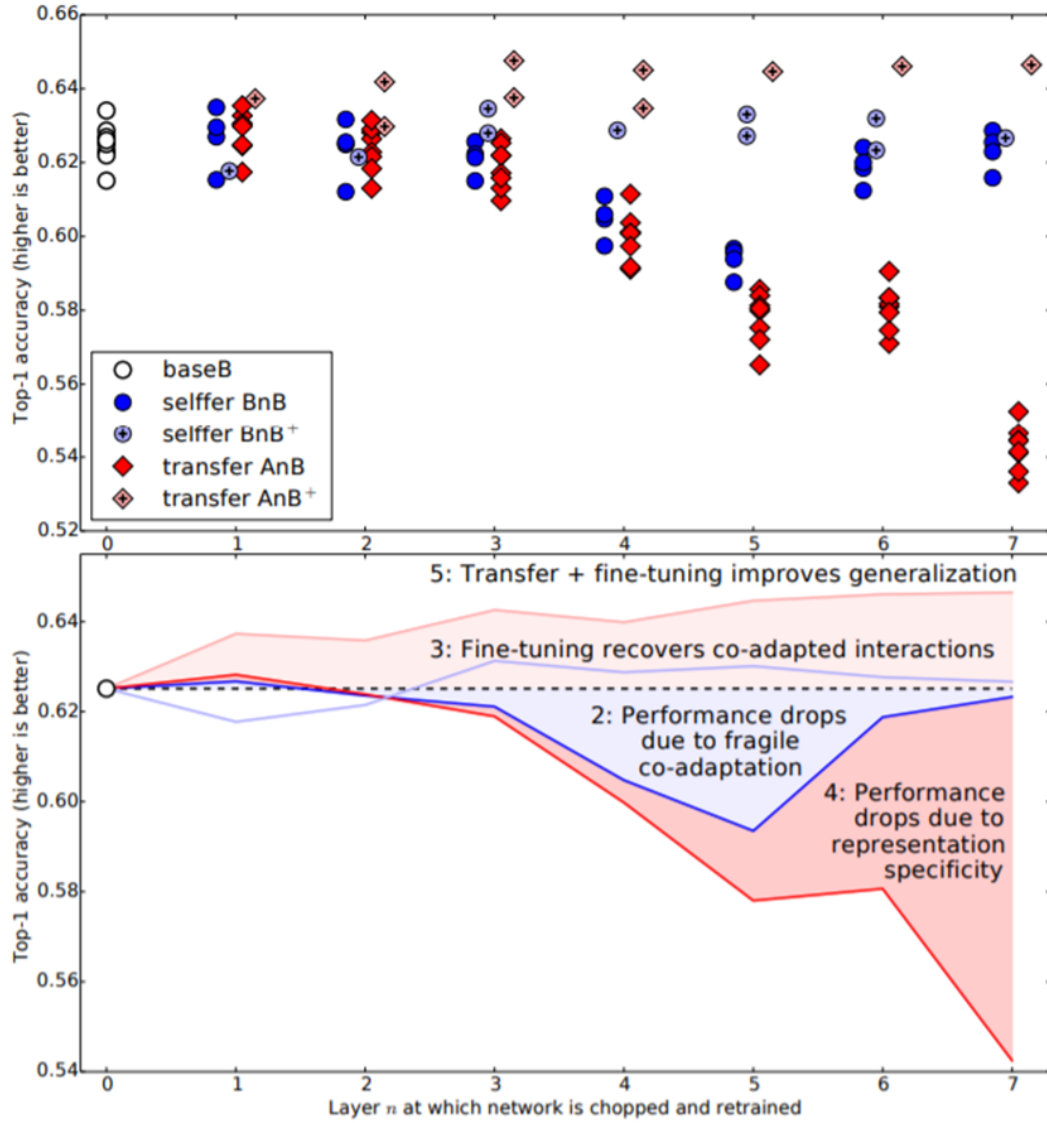


Figure 2.1: Experimental result from Yosinski et al. [25]

2.2 Statistical Style Domain Adaptation approaches

This section will walk through one of the famous types of domain adaptation methods, the statistical style domain adaptation. These methods usually use statistical approaches, such as Maximum Mean Discrepancy (MMD) [7] and $H\Delta H$ [1], to estimate the distribution discrepancy between the source and target domains. Since MMD is more widely used in this area, I will first introduce its mechanism, followed by some classic papers that used this statistic approach to solve the domain adaptation problem.

2.2.1 A kernel two-sample test

Gretton et al. [7] proposed a novel statistical approach called Maximum Mean Discrepancy (MMD) to determine if two data distributions are identical. The idea is to project the original data into a reproducing kernel Hilbert space (RKHS) and calculate the L2 distance between the kernel-mean embedding of these two data distributions.

The reason for projecting the features into the high-dimensional space is very similar to the logic illustrated in the paper of support vector machine (SVM) [3]: finding a hyper-plane so that we can linearly separate the features by half.

Suppose we have two data distributions: source data distribution (S) and target data distribution (T). We can write the formula of MMD as:

$$MMD(S, T) = \|\mu_S - \mu_T\|^2, \quad (2.1)$$

where μ_S and μ_T are the kernel mean embedding in the RKHS for the source and target data distribution respectively.

The kernel-mean embedding is calculated using the mapping function f to project the original data from a low-dimensional space into the RKHS feature space and then calculate their expectations. Therefore, we can write the formula of kernel-mean embedding as follow:

$$\mu_S = E_{x \sim S} f(x), \mu_T = E_{x \sim T} f(x), \quad (2.2)$$

where function f maps the data from the input feature space into RKHS feature space.

Back to the calculation of MMD, since calculating the L2-norm requires doing dot product which takes lots of computing power when doing it in the high-dimensional space, Gretton et al. [7] used the kernel trick to avoid these extensive computations. The idea of the kernel trick is that by using the original data as the input of the kernel function, it is equivalent to the result of doing the dot product in the high-dimensional space. In short, the kernel function expression can be viewed as follows:

$$K(x, x') = \langle f(x), f(x') \rangle, \quad (2.3)$$

where $x \in S$ and $x' \in T$.

By expanding equation 2.1 and using the kernel function to replace the dot product, we can update the formula of calculating MMD as:

$$MMD(S, T, K) = \frac{1}{n_s^2} \sum_{i=1}^{n_s} \sum_{j \neq i}^{n_s} K(x_i, x_j) + \frac{1}{n_t^2} \sum_{i=1}^{n_t} \sum_{j \neq i}^{n_t} K(x'_i, x'_j) - \frac{2}{n_s n_t} \sum_{i=1}^{n_s} \sum_{j=1}^{n_t} K(x_i, x'_j) \quad (2.4)$$

In general, the smaller the value of MMD, the more likely these two data distributions are identical. Similarly, the larger the value of MMD, the more likely these two data distributions are different from each other

2.2.2 Deep Domain Confusion: Maximizing for Domain Invariance (DDC)

Tzeng et al. [22] is the first person who applied the idea of MMD in solving domain adaptation problems. As Yosinski et al. [25] mentioned in his paper, the feature extracted by each layer will gradually transition from general to specific along with the network. To address this problem, Tzeng et al. [22] used the concept of MMD in the deep layer to estimate the difference between the extracted source and target features. The result of MMD will be used as part of the network loss function to ensure the deep layers can extract the features common on the source and target domain.

Figure 2.2 demonstrates the network architecture of DDC. The dashed line means that the left and right-hand side layers share the same architecture and hyper-parameters. DDC still used AlexNet [10] as the backbone. For both of the source domain (correspond to the labeled images in Figure 2.2) and the target domain (correspond to the unlabeled images in Figure 2.2), they are sharing the same network architecture. The only novel point is that at the end of the seventh fully connected layer, Tzeng et al. [22] treat the source and target features as the input of MMD and pass the result into the final loss function.

Since the domain adaptation loss is part of the network loss function, later on, it will participate in the process of backward propagation. Therefore, as the total loss function gets smaller and smaller, we would expect the domain adaptation loss to get smaller. As the domain adaptation loss becomes smaller, it equivalently means that the distribution discrepancy between the source and target features becomes smaller. Therefore, we successfully extract the domain invariant features and overcome the limitations of task-specific layers.

However, DDC has an obvious disadvantage. As Yosinski et al. [25] mentioned in his paper, if using AlexNet as the backbone, layers 4 to 7 are all marked as task-specific layers. In DDC,

we only perform domain adaptation at the seventh fully connected layer. To safely transfer all the task-specific layers, we should apply them to all the task-specific layers instead of only performing domain adaptation in a single layer. The reported result of DDC is not satisfactory. In the following article, let's see how we can further improve it.

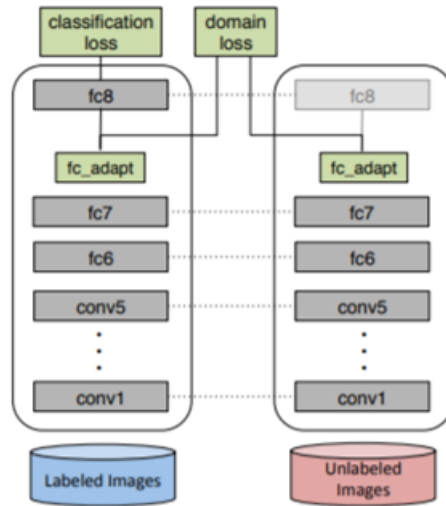


Figure 2.2: Network architecture of DDC [22]

2.2.3 Transferable Representation Learning with Deep Adaptation Networks (DAN)

Instead of only performing domain adaptation in one of the task-specific layers, Long et al. [15] performs domain adaptation on all of the fully connected layers. In DDC, Tzeng et al. [22] used a single Gaussian kernel to compute the value of MMD. In DAN, Long et al. [15] uses multiple Gaussian kernels to increase the testing power of MMD. The final kernel choice of DAN is actual a convex combination of multiple Gaussian kernels. The mathematical

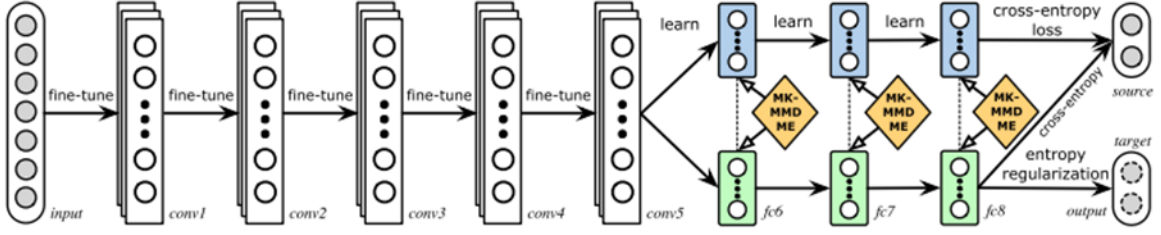


Figure 2.3: Network architecture of DAN [15]

expression of calculating the kernel function can be expressed as:

$$K = \sum_{u=1}^m B_u K_u , \quad (2.5)$$

where $\sum_{u=1}^m B_u = 1$ and $B_u \geq 0$. Figure 2.3 illustrates the workflow and architecture of DAN. The input of the network is the source and target data. Similarly, the source data is labeled, whereas the target data is unlabeled. The source and target data share the same network architecture and hyper-parameters. For the shallow layers, Long et al. [15] does not perform domain adaptation since they are more likely to capture the general information and safe to transfer. Notice that DAN keeps fine-tuning the first fifth layers instead of freezing them to maintain the inter-dependency between the consecutive hidden layers. For the sixth, seventh, and eighth fully connected layers, the blocks marked in blue mean the current layer

takes source data as the input, and the blocks marked in green mean the current layer takes the target data as the input. We can notice that instead of only performing domain adaption in the seventh fully connected layer, DAN performs domain adaption on all the fully connected layers. Moreover, instead of using a single kernel to compute MMD, DAN uses multiple kernel MMD to increase its testing power and minimize the probability of getting a type-II error.

However, in DAN, there is also a significant limitation: only performing domain-to-domain alignment without considering the category information of each sample. At the end of the training process, the discrepancy between the source and target domain distributions can be relatively small. However, the subdomains from the same class might still not align with each other accurately.

2.2.4 Deep Subdomain Adaptation Network for Image Classification (DSAN)

To alleviate the limitation of DAN, instead of performing domain-to-domain alignment, Zhu et al. [27] performs class-to-class alignment by leveraging the pseudo-label given by the classifier. Instead of using MMD to measure the discrepancy between the source and target domain, Zhu et al. [27] uses a weighted MMD to measure the discrepancy between the source and target subdomains.

The mathematical expression of weighted MMD is as follows:

$$WMMD(S, T, K) = \frac{1}{C} \sum_{c=1}^C \left\| \sum_{x \in S} P_x^c f(x) - \sum_{x' \in T} P_{x'}^c f(x') \right\|^2 \quad (2.6)$$

where C stands for the total number of classes, function f stands for the mapping function,

P_x^c and $P_{x'}^c$ stand for the probability that the deep neural network assign sample x and x' to class c respectively.

By performing weighted MMD, we are trying to align the features with the same class type. That's the reason why people call it class-to-class or subdomains alignment. Except for calculating MMD, DSAN and DAN share almost the same network architecture.

Although DSAN incorporates the pseudo-labels of the samples into the design and successfully solves the limitation of domain-to-domain alignment, the quality of the pseudo-labels is the main challenge DSAN needs to face. The weighted MMD uses the pseudo-labels of the source and target samples to perform the class-to-class alignment. If the classifier assigns a wrong pseudo-label to an instance with very high confidence, the result returned by equation 2.6 will be misleading and cause a negative impact.

Therefore, we need a way to test the robustness of the pseudo-label before using them during the training process. In Chapter 4, I will demonstrate the proposed method and explain how I overcome this limitation.

2.2.5 Conclusion for statistical domain adaptation approaches

So far, we have gone through a new statistical method, Maximum Mean Discrepancy, which can help us measure the discrepancy between two domains. We also talk about three different types of statistical domain adaptation methods.

The first one is deep domain confusion (DDC) [22]. DDC is the first paper that utilizes the idea of MMD in solving domain adaptation problems. However, since it only performs the domain adaptation on a single fully connected layer, it doesn't achieve satisfactory results.

To overcome the limitation of DDC, the deep adaptation network (DAN) [15] performs

a multi-layer domain adaption and switches from a single-kernel MMD into multi-kernel MMD. Multi-kernel MMD can increase the test power of MMD and decrease the probability of getting type-II error. DAN can help these task-specific layers to extract domain-invariant features. However, since DAN does not consider the class of the sample during the training process, it is performing a domain-to-domain alignment. The main disadvantage of domain-to-domain alignment is that the discrepancy between these two domains can be relatively small at the end of the training phase. However, the discrepancy between the subdomains can still be significant.

To overcome the limitation of DAN, Deep Subdomain Adaptation Network (DSAN) [27] incorporates the idea of pseudo-labels into the design and utilizes weighted MMD to perform a class-to-class alignment. Although it successfully takes the fine-grained information into account, it cannot check the quality of individual pseudo-labels. If the classifier mistakenly labels a sample with very high confidence, this error will further propagate into the training process and cause the weighted MMD to generate misleading results.

2.3 Adversarial Style Domain Adaptation approaches

Another type of method used in solving domain adaptation problems is adversarial training. This type of method is inspired by the idea of Generative Adversarial Network (GAN) [6]. Unlike GAN, when solving domain adaptation problems, the generator is usually responsible for extracting domain-invariant features. The discriminator is responsible for distinguishing whether the extracted feature comes from the source domain or the target domain. In this section, I will introduce two adversarial style domain adaptation methods: Domain-Adversarial training of Neural Networks (DANN) [5] and Multi-Adversarial Domain Adaptation (MADA) [18].

2.3.1 Domain-adversarial training of neural networks (DANN)

Ganin et al. [5] is the first person that incorporated the idea of adversarial training into the area of domain adaptation. The network architecture developed consists of three main parts: (1) the feature extractor, (2) the discriminator, and (3) the classifier. The assigned duty of each component is as follows:

(1) The feature extractor takes the source and target data as the inputs and outputs their extracted features. Then, the extracted features will be sent to the classifier and discriminator, respectively. Therefore, the feature extractor has two primary responsibilities. First, the extracted feature should be a domain-invariant feature so that the discriminator can not distinguish whether the feature is coming from the source or the target domain. Second, since only the source data has a label, the classifier will only take the source feature as the input to calculate the cross-entropy loss. Because of this circumstance, the extracted feature should not be trivial to ensure the classifier can be well-trained.

(2) On the other hand, the discriminator determines whether the input features are coming from the source or target domain. Since we already know the domain information of each sample in advance, we will send both the source and target extracted features to the discriminator. The goal is to force the feature extractor to extract the domain-invariant features to train the classifier.

(3) For the classifier, since only the source data contains label information, only the source features will participate in updating the classifier. With the help of the discriminator, we can expect the source features to be domain-invariant (which means the target samples should also have similar characteristics). Therefore, even though the classifier is only trained based on the source samples, it still can perform reasonably on the target samples.

One of the main limitations of DANN is that the discriminator is doing domain-to-domain

alignment but not class-to-class alignment. One may wonder why it is domain-to-domain alignment and why perhaps this is not a good choice? It is a domain-to-domain alignment because the discriminator only focuses on classifying the input features into the source domain or the target domain without considering the class of the input features. Why is this not a good choice? Suppose I have two similar types (Class A and Class B) in the source and target domain. Although the discriminator can help us extract domain-invariant features from classes A and B, these features might not help train the classifier. They are so similar that the classifier will treat them as the same class. To alleviate this problem Pei et al. [18] introduce a new network architecture, called (MADA).

2.3.2 Multi-adversarial domain adaptation (MADA)

Instead of having only one discriminator in the entire network, Pei et al. [18] solves the aforementioned problem by including K discriminators into its design, where K is the total number of classes. After the feature extractor finishes extracting the features, it will utilize the pseudo-labels given by the classifier to determine which features are needed to be sent to which discriminator. By doing this matching, the discriminator can perform class-to-class alignment and ensure that each class has only its characteristic so that the classifier can efficiently utilize the extracted feature later on.

The main limitation of MADA is similar to the restriction of DSAN [27]. Both of these methods rely on pseudo-labels to perform a class-to-class alignment. However, neither of them has included a way to test the quality of the pseudo-labels. Suppose the classifier fails to provide accurate pseudo-labels (this is very likely to happen, especially during the early training phase). In that case, the result of the class-to-class alignment will be misleading, resulting in negative impacts on the whole training process.

Chapter 3

Ablation Studies toward DAN Model

To fully understand the importance of each component in the DAN [15] model, I conduct some ablation studies in this section. The findings in this chapter build the foundation of the proposed method introduced in Chapter 4.

I compare transfer learning performance with different DAN model configurations in the ablation studies. To test the influence of multiple adaptation layers for transfer learning, I use AlexNet [10] as base network and compare the network performance for scenarios using 1) single layer (the eighth fully connected layer, FC8), 2) 2 layers(FC7-FC8) and 3) 3 layers (FC6-FC8) for domain adaptation, respectively termed as AlexNet-1, AlexNet-2, and AlexNet-3. To study the influence of MMD kernel numbers, I compare the model performance using the above three model setups with kernels as 1, 2, 5, and 10, respectively. To examine the influence of base network depth, I use ResNet [9] as the base network architecture and try different depth configurations (50,101 and 152 layers). All above comparisons are based on unsupervised transfer learning settings, i.e., the source data is labeled, and the target data is unlabeled. At last, I exam semi-supervised domain adaptation by adding eight labeled examples per category from the target domain during the training process.

I use an open-source benchmark dataset, Office-31, for domain adaptation for the dataset. This dataset contains 4,653 images in 31 categories collected from 3 domains: Amazon (A), Webcam (W), and Dslr (D). Domain A contains office images taken from amazon.com, while images in domains W and D are collected using a webcam or a DSLR camera, respectively.

Since domains W and D are very similar, I only test transfer learning performance from A to W ($A \rightarrow W$), A to D ($A \rightarrow D$), W to A ($W \rightarrow A$), and D to A ($D \rightarrow A$). Fig 3.1 illustrates how the samples look like in the Office-31 dataset.



Figure 3.1: Samples from Office-31 Dataset

3.1 Results with AlexNet

This section will discuss three factors that affect domain adaption accuracy: the number of kernels in calculating MMD, the number of layers for adaption, and the number of layers of the backbone network.

In the first experiment, I implemented a single-layer adaption for the seventh fully connected layer (FC7) with four different kernel configurations: 1, 2, 5, and 10 kernels. Figures 3.2, 3.3, and 3.4 show the experimental results. There are two main findings. One is that it is unable to build a complex model to advise the network without sufficient kernels. However, stacking too many kernels will also cause the kernel functions to become too complicated. The network, therefore, cannot learn well since the extracted features will never pass the MMD test.

The other finding is that domain adaption has better performance when the adaption happens from a large domain to a smaller domain. For example, A has larger size compared to

W . The adaption $A \rightarrow W$ also performs better than $W \rightarrow A$. I believe the main reason is that the large domain has a lower bias so that the network can learn to extract more robust features compared with the small domain.

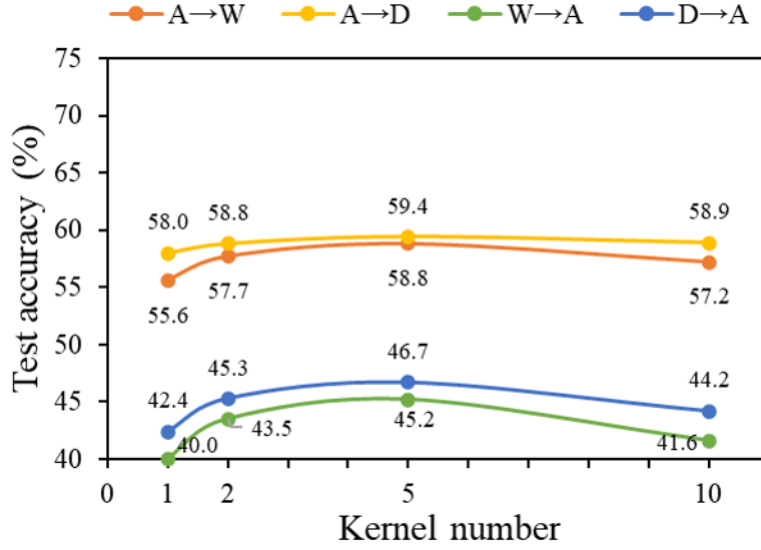


Figure 3.2: AlexNet-1: Single adaptation layer (FC8)

Then I examine multi-layer adaptation by running AlexNet with two adaptation layers and three adaptation layers. As shown in Figure 3.2 and Figure 3.3, the test accuracy of AlexNet-2 is similar to that of using a single layer. However, when three task-specific fully connected layers are used for adaptation, the model performance improves significantly. For example, in Figure 3.2 and Figure 3.4, compared to AlexNet-1, AlexNet-3 using single kernel can achieve an absolute accuracy increase of 9.6 for task $A \rightarrow W$, 7.1 for $A \rightarrow D$, 7.3 for $W \rightarrow A$ and 4.4 for $D \rightarrow A$. This observation is because different layers capture features at different abstraction levels, thus narrowing the gap between source and target underlying the marginal and conditional distribution [15].

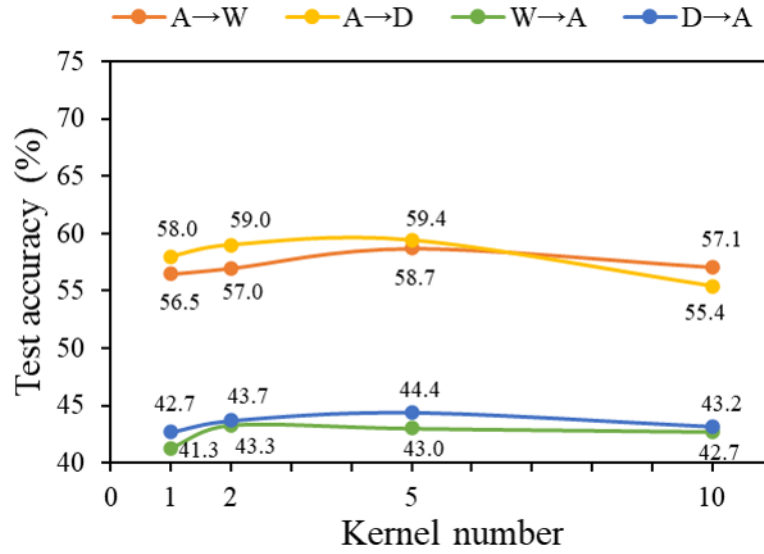


Figure 3.3: AlexNet-2: Double adaptation layers (FC7 FC8)

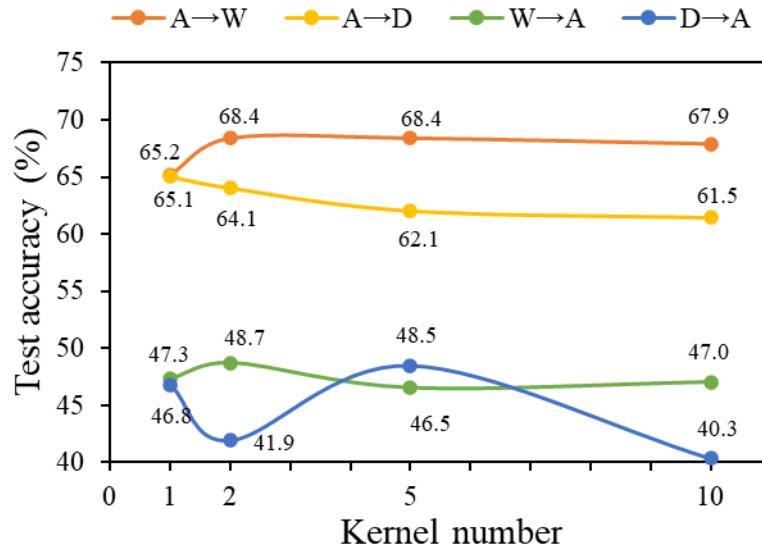


Figure 3.4: AlexNet-3: Triple adaptation layers (FC6 FC8)

3.2 Results with ResNet

To study how the depth of base architecture may affect domain adaptation performance, I use ResNet50, ResNet101, and ResNet 152 on the same transfer learning tasks. I use five kernels for MMD, and the results are shown in Figure 3.5. By switching the backbone network from 8 layers AlexNet to ResNet50, the results show around 20% improvement compared with AlexNet in Figure 3.2. This observation reveals that a deeper network structure can improve feature transferability than the shallower ones. However, it is not guaranteed that very deep networks can always show better-adapted accuracy. For all tasks, ResNet 101 offers the best model performance. Although ResNet152 extracts even deeper features, the feature transferability decreases slightly. The cross-domain discrepancy still exists in the deep features, showing that we cannot solve the transfer learning problem by using deep neural networks alone.

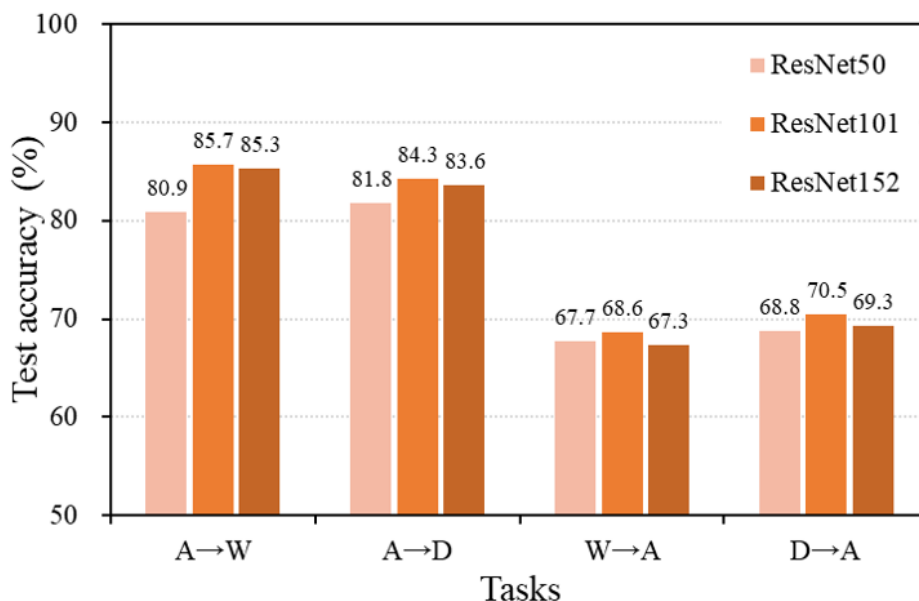


Figure 3.5: Unsupervised transfer learning with ResNet

3.3 Results with semi-supervised transfer learning

I sampled eight labeled data per category from the target domain to study semi-supervised domain adaptation. Only source samples and these target samples can be seen during the training process. I use AlexNet as our base network and three adaptation layers for this experiment. As Figure 3.6 shows, even with very few labeled target data, semi-supervised adaptation still outperforms unsupervised adaptation for all tasks. For example, using the semi-supervised method, the test accuracy reaches 83.5% for task $A \rightarrow W$, with a drastic boost of 19.7% compared to the unsupervised method. Similar behavior can also be found when using ResNet 101 as the backbone. This observation reveals that labeled target data is much more valuable than source data because it can specify where a good target classifier should reside in the source-constrained spaces.

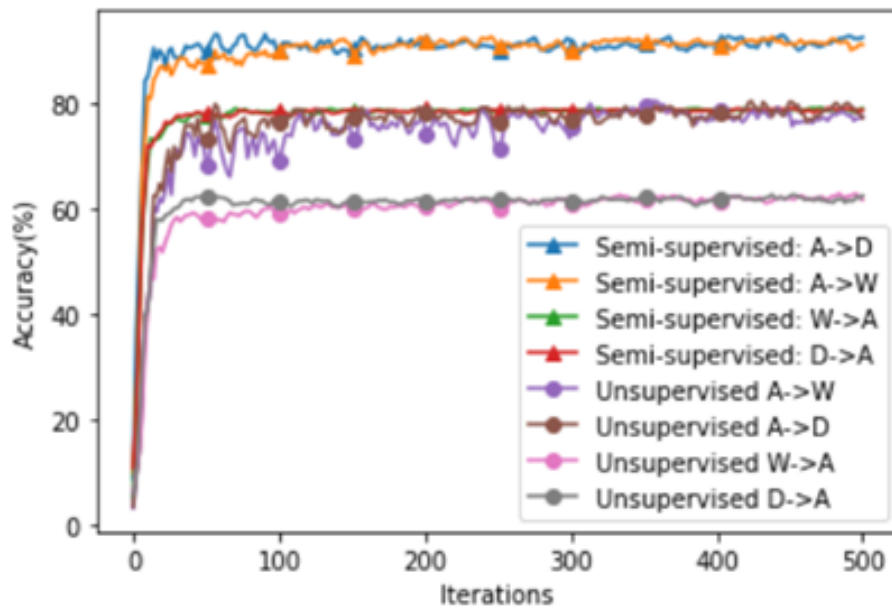


Figure 3.6: Unsupervised transfer learning vs. Semi-supervised transfer learning

This finding inspires me a lot. Suppose I can have a sufficient number of target samples with robust pseudo-labels. By treating these pseudo-labels as the ground-truth labels, I can switch the problem setting from unsupervised domain adaptation into semi-supervised domain adaptation. As Figure 3.6 shows, by just having a few correctly labeled target samples during the training process, the domain adaptation accuracy can improve significantly. Based on this idea, let's jump into my proposed method.

Chapter 4

Methodology

4.1 Introduction

In order to alleviate the distribution discrepancy, domain adaptation transfers the knowledge from a labeled source domain into an unlabeled but related target domain. The shift of data distribution across different domains is the main challenge with which DA is currently faced. An example is shown in Figure 4.1 (Top left). Before adaptation, although the source and target domain have the same objects, we can observe the shift of data distribution between the source and target domains. If we directly apply the source data to train the model, the model will not perform well on the target domain.

To solve this problem, people developed domain-to-domain alignment methods, such as DDC [22], DAN [15], and DANN [5]. The limitation of these studies is that they do not consider the fine-grained information of each category. As shown in Figure 4.1(Top left), after the global alignment, the discrepancy between the distributions of the two domains can be relatively small, but the subdomains from the same type of class might still not be able to align with each other accurately.

With regard to the challenge of global alignment, recent researches have started focusing on class-to-class alignment (also known as semantic alignment), such as MADA [18], DSAN[27], and cluster-to-cluster alignment method like [2]. An example of class-to-class alignment can

be found in Figure 4.1(Top right). To perform class-to-class alignment, we need target pseudo-labels, which are usually collected from the classifier trained by the source data. Due to the shift in datasets, the classifier will inevitably bias to the source domain [21]. Because of that, the pseudo-labels are of low quality. Using these low-quality pseudo-labels to perform class-to-class alignment will propagate errors into the future training process.

To tackle this issue, as shown in Figure 4.1 (bottom), I will filter out the target samples without robust pseudo labels before the training process. The robustness of the target pseudo-label is measured based on the cosine distance between the extracted target feature and the corresponding class centroid. A Gaussian-uniform mixture model [8, 11] is deployed to estimate the posterior probability of correct labeling. The samples with high robustness (equivalently means having a high probability of being correctly labeled) will be marked as “inliers.” In contrast, the samples with low robustness will be marked as “outliers.” Their pseudo-labels will be treated as the “true” label for the inliers. For the outliers, their pseudo-labels will be updated based on the category of the nearest class centroid. An exponential moving average strategy [2] is used to update the class centroids, and only inliers are participating in the centroids’ updating process to minimize the negative transfer and maximize the positive transfer. After updating their pseudo-labels, I use the target samples to train the classifier and feature extractor, along with the source samples, through a standard cross-entropy loss. Comprehensive experiments presented in Chapter 5 demonstrate the effectiveness of this proposed method.



Figure 4.1: In these illustrations, the large circles represent the domains (for the global alignment figure, or subdomains for the class-to-class illustrations)—either the target or source domains. The solid triangles and circles represent individual samples from two classes. Samples from the source domain are colored blue, whereas samples from the target domain are colored orange. **Top left:** global alignment might do well at aligning the domains, but by failing to capture fine-grained information and it might not do an excellent job at separating the classes. **Top right:** class-to-class alignment can exploit local information, namely pseudo-labels; however, since some of these pseudo-labels may be incorrect, using these incorrectly labeled samples to train the network results in an imperfect class-to-class alignment. **Bottom:** Before performing the adaptation, if we can filter out the target samples that might be incorrectly labeled (based on the robustness of their pseudo-labels), we can potentially improve the accuracy of the class-to-class alignment.

4.2 Workflow and architecture of the proposed method

4.2.1 An Overview of the Training Algorithm

Figure 4.2 illustrates the overall architecture and workflow of the proposed method.

The proposed method is based on the well-known architecture [4], which consists of two primary modules, a feature extractor G and a classifier F .

The workflow of this method consists of three main steps as given below.

- **First step:** Performing a cluster-to-cluster alignment as described in [2] to obtain the initial model M_{init} .
- **Second step:** Using M_{init} to compute a pseudo-label for the target data. The pseudo-labels and the extracted target features (both given by M_{init}) will subsequently be used to estimate the target centroids. Based on the cosine distance from the target feature to each class centroid, I train the Gaussian-uniform mixture model by using the Expectation-Maximization (EM) algorithm [16]. Once the training is completed, the Gaussian-uniform mixture model will output the posterior probability of the correct labeling given the feature distance and its pseudo-label.
- **Third step:** Initializing a fresh model M_{train} , and developing two types of the classifier to train it: a Target-Oriented (TO) classifier or centroid-oriented (CO) classifier. These classifiers employ the posterior probability of correct labeling (Υ). For the TO classifier, based on the pseudo-labels given from M_{init} and the probability of correct labeling given by Gaussian-uniform mixture model, I treat the robust pseudo-labels

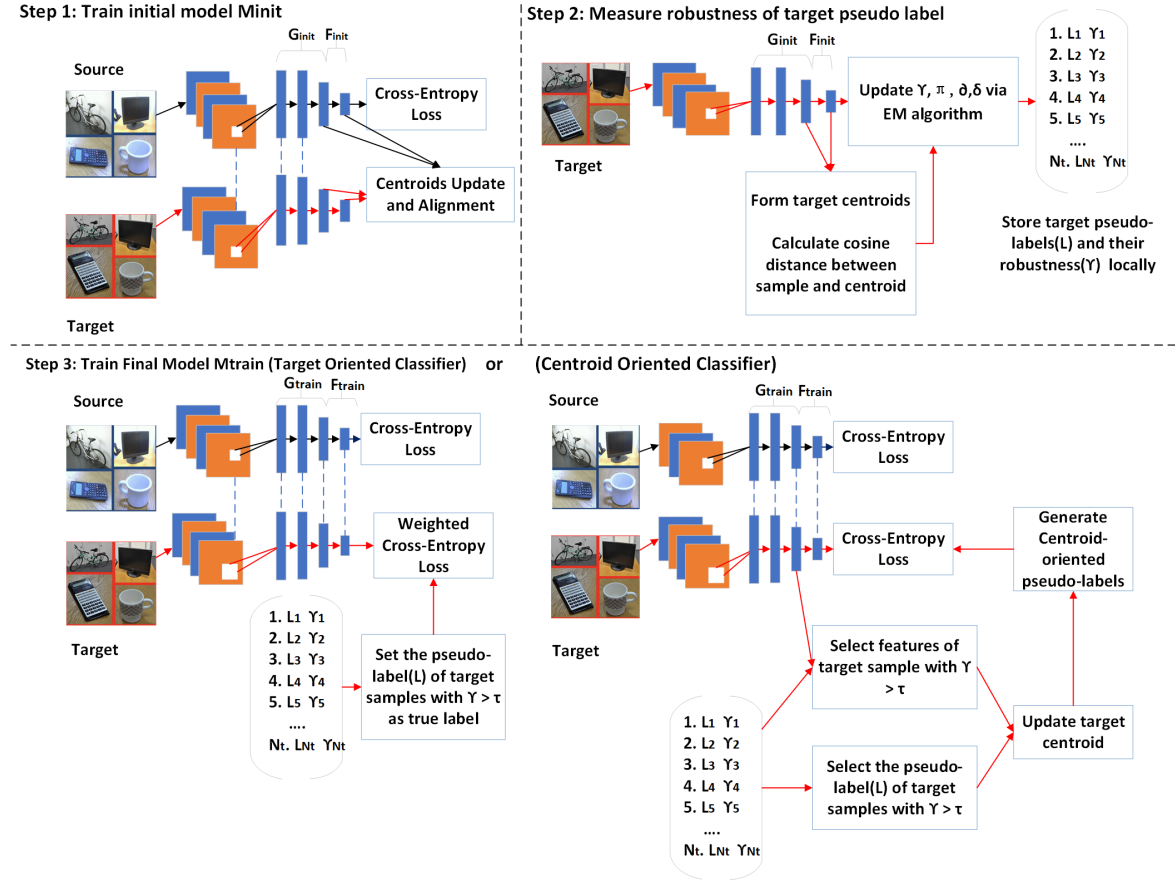


Figure 4.2: A graphical representation of the training steps of our proposed method. In the figure, the red and black arrows represent the computational flow of the target and source domain, respectively. I introduce two training models: M_{init} , and M_{train} . Each network is separated into two modules: a feature extractor $G_{init/train}$, and a classifier $F_{init/train}$. **In the first step**, I initialize M_{init} by performing a centroid-to-centroid alignment. Upon completing the training of M_{init} , **in the second step**, I form target centroids by utilizing the target features and the target pseudo-labels generated by M_{init} . Based on the target pseudo-labels and the cosine distance between target features and the corresponding target centroid, I use the Expectation-Maximization (EM) algorithm to train a Gaussian-uniform mixture model and locally store the resulting posterior probabilities of correct labeling. **In the third step**, I have developed two types of training models M_{train} : a target-oriented classifier and a centroid-oriented classifier. For the first training model, I treat the target pseudo-labels with robustness greater than a filtering threshold τ as the ground-truth labels and employ these labels to a weighted cross-entropy loss function. I use target samples with robustness greater than τ to form target centroids for the second training model. Based on the target feature distance to each of the target centroids, I generate a new pseudo-label for each target sample and employ these labels to the standard cross-entropy loss function.

(i.e., the pseudo-labels with robustness $\Upsilon \geq \tau$, where τ is the filtering threshold) as the “true” label and employ them to a weighted cross-entropy loss. For the CO classifier, I form target centroids by using the target features extracted from M_{train} and the pseudo-labels given by M_{init} . It is noteworthy that only the target samples with robust pseudo-labels will participate during the updating process of the centroids. Based on the class of the nearest centroid for each target sample, I assign a new pseudo-label to the target samples. By treating the updated pseudo-label as a “true” label, I apply both source and target samples to a standard cross-entropy loss to train the networks F_{train} and G_{train} . Once this training is completed, I replace the initial model M_{init} with M_{train} and repeat the second and third steps until the maximum number of iterations is reached. Pseudo-code of training process is provided at Algorithm 1. In the following subsections, I will give a detailed explanation of each training step.

4.2.2 Training the Initial Model

For this step, I train the initial model M_{init} by performing the centroid-to-centroid alignment as introduced in [2]. I firstly obtain the target pseudo-labels as $\tilde{y}_j^t = \text{argmax}(F_{init}(G_{init}(x_j^t)))$ for each mini-batch of target samples. Based on the extracted target features $\{G_{init}(x_j^t)\}_{j=1}^{N_t}$ and for each pseudo-label $\{\tilde{y}_j^t\}_{j=1}^{N_t}$, I form the target local centroid as follows (where the superscript *localT* means it is a local target centroid, and the subscript *init_k* means this centroid belongs to class k and I utilize the features extracted by M_{init} to form the centroid):

$$C_{init_k}^{localT} = \frac{\sum_{j=1}^{B_t} \mathbb{1}_{\{\tilde{y}_j^t=k\}} G_{init}(x_j^t)}{\sum_{j=1}^{B_t} \mathbb{1}_{\{\tilde{y}_j^t=k\}}}, k = 1, 2, 3, \dots, K, \quad (4.1)$$

where $k \in K$, and K is the number of classes. B_t is the mini-batch size. Based on the local target centroid, I update the global target centroid through the Exponential Moving Average

(EMA) strategy [2] as follows:

$$C_{init_k}^T = \rho C_{init_k}^{localT} + (1 - \rho) C_{init_k}^T, k = 1, 2, 3, \dots K, \quad (4.2)$$

where ρ is the trade-off parameter, fixed at 0.1 by default. This choice of ρ ensures that our global centroids are updated gradually. The source local and global centroids are updated in the same manner. To perform centroid-to-centroid alignment, I measure the square Euclidean distance between the source and target global centroids as in Equation 4.3. The justification of using Euclidean distance is that by calculating the centroid mean using the Bregman divergence (such as Euclidean distance), I can generate the optimal centroid representation [19] as

$$d(C_{init_k}^T, C_{init_k}^S) = \|C_{init_k}^T - C_{init_k}^S\|^2. \quad (4.3)$$

Following the calculation of the distance between the source and target global centroids, the loss function for training M_{init} is computed as:

$$\min_{\theta_{G_{init}}, \theta_{F_{init}}} \frac{1}{N_s} \sum_{i=1}^{N_s} J(F_{init}(G_{init}(x_i^s)), y_i^s) + \sum_{k=1}^K d(C_{init_k}^T, C_{init_k}^S), \quad (4.4)$$

where J is the standard cross-entropy loss function.

4.2.3 Training the Gaussian-Uniform Mixture Model

According to [8, 11], we can use a Gaussian-uniform mixture model to compute the posterior probability of correct labeling $P_\phi(Z_j = 1 | G_{init}(x_j^t), \tilde{y}_j^t)$, where $Z \in \{0, 1\}$ is a latent variable used to indicate whether or not the sample is correctly labeled, j stands for the sample's index and ϕ is the set of parameters of Gaussian-uniform mixture model.

I follow the same approach that is introduced in [8] to compute the posterior probability of

correct labeling and train the Gaussian-uniform mixture model. By assuming the correct labeling data is much closer to the class centroid, I use a Gaussian distribution to model the correctly labeled target data and use a uniform distribution to model the incorrectly labeled target data. The Gaussian-uniform mixture model is built based on the cosine distance between the target feature and corresponding class centroid as $d_j^t = \text{CosDist}(G_{init}(x_j^t), C_{\tilde{y}_{j_{init}}^t}^T)$. For the class centroids, instead of using the EMA strategy to progressively update $\{C_{init_k}^T\}_{k=1}^K$, since I have already finished training M_{init} , I can utilize all the target samples with the same pseudo-labels to form them as follows: $C_{init_k}^T = \frac{\sum_{j=1}^{N_t} \mathbb{1}_{\{\tilde{y}_j^t=k\}} G_{init}(x_j^t)}{\sum_{j=1}^{N_t} \mathbb{1}_{\{\tilde{y}_j^t=k\}}}$, $k = 1, 2, 3, \dots, K$.

Given the pseudo-labels, the posterior probabilities of getting certain feature distances can be computed as [8]:

$$P(d_j^t | \tilde{y}_j^t) = \pi_{\tilde{y}_j^t} \mathcal{N}^+(d_j^t | 0, \sigma_{\tilde{y}_j^t}) + (1 - \pi_{\tilde{y}_j^t}) \mathcal{U}(0, \delta_{\tilde{y}_j^t}) \quad , \quad (4.5)$$

where π is the prior probability of getting correct labeling, σ is the covariance of the Gaussian distribution, δ is the normalization parameter of the uniform distribution, $\mathcal{N}^+$ is the Gaussian distribution, and $\mathcal{U}$ stands for the uniform distribution. Based on Equation (5) and instructions from Gu et al. [8], I can write the posterior probability of correct labeling for a target sample j as:

$$P_\phi(Z_j = 1 | G_{init}(x_j^t), \tilde{y}_j^t) = \frac{\pi_{\tilde{y}_j^t} \mathcal{N}^+(d_j^t | 0, \sigma_{\tilde{y}_j^t})}{\pi_{\tilde{y}_j^t} \mathcal{N}^+(d_j^t | 0, \sigma_{\tilde{y}_j^t}) + (1 - \pi_{\tilde{y}_j^t}) \mathcal{U}(0, \delta_{\tilde{y}_j^t})} \quad , \quad (4.6)$$

where $\phi = \{\pi_k, \sigma_k, \delta_k\}_{k=1}^K$ is a set of parameters of Gaussian-uniform mixture model, and K is the number of classes.

For updating the Gaussian-uniform mixture model, I follow the exact same algorithm as described in [8]. To make sure the Gaussian distribution is zero-mean, I define $\tilde{d}_j^t = (-1)^{m_j} d_j^t$,

where m_j is sampled from the Bernoulli distribution $B(1, 0.5)$. By using the EM algorithm as shown below, we can iteratively update ϕ (for iteration $l = 0, \dots$) until the model converges or reaches a maximum of iterations.

$$\{\pi_k^{(0)} = 0.1, \sigma_k^{(0)} = \frac{\sum_{j=1}^{N_t} \mathbb{1}_{\{\tilde{y}_j^t=k\}} (\tilde{d}_j^t)^2}{N_t}, \delta_k^{(0)} = 0.2\}_{k=1}^K \quad (4.7)$$

$$\{\Upsilon_j^{(l+1)} = \frac{\pi_{\tilde{y}_j^t}^{(l)} \mathcal{N}(\tilde{d}_j^t | 0, \sigma_{\tilde{y}_j^t}^{(l)})}{\pi_{\tilde{y}_j^t}^{(l)} \mathcal{N}(\tilde{d}_j^t | 0, \sigma_{\tilde{y}_j^t}^{(l)}) + (1 - \pi_{\tilde{y}_j^t}^{(l)}) \mathcal{U}(-\delta_{\tilde{y}_j^t}^{(l)}, \delta_{\tilde{y}_j^t}^{(l)})}\}_{j=1}^{N_t} \quad (4.8)$$

$$\{\pi_k^{(l+1)} = \frac{1}{\sum_{j=1}^{N_t} \mathbb{1}_{\{\tilde{y}_j^t=k\}}} \sum_{j=1}^{N_t} \mathbb{1}_{\{\tilde{y}_j^t=k\}} \Upsilon_j^{(l+1)}\}_{k=1}^K \quad (4.9)$$

$$\{\sigma_k^{(l+1)} = \frac{\sum_{j=1}^{N_t} \mathbb{1}_{\{\tilde{y}_j^t=k\}} \Upsilon_j^{(l+1)} (\tilde{d}_j^t)^2}{\sum_{j=1}^{N_t} \mathbb{1}_{\{\tilde{y}_j^t=k\}} \Upsilon_j^{(l+1)}}\}_{k=1}^K \quad (4.10)$$

$$\{\delta_k^{(l+1)} = \sqrt{3(q_2 - q_1^2)}\}_{k=1}^K \quad (4.11)$$

In the above description, $\mathbb{1}$ is the indicator function, and q_1 and q_2 are the first- and second-order centered data moments which are defined as:

$$q_1 = \frac{1}{\sum_{j=1}^{N_t} \mathbb{1}_{\{\tilde{y}_j^t=k\}} \Upsilon_j^{(l+1)}} \sum_{j=1}^{N_t} \frac{1 - \Upsilon_j^{(l+1)}}{1 - \pi_k^{(l+1)}} \mathbb{1}_{\{\tilde{y}_j^t=k\}} \tilde{d}_j^t, \quad (4.12)$$

$$q_2 = \frac{1}{\sum_{j=1}^{N_t} \mathbb{1}_{\{\tilde{y}_j^t=k\}} \Upsilon_j^{(l+1)}} \sum_{j=1}^{N_t} \frac{1 - \Upsilon_j^{(l+1)}}{1 - \pi_k^{(l+1)}} \mathbb{1}_{\{\tilde{y}_j^t=k\}} (\tilde{d}_j^t)^2. \quad (4.13)$$

4.2.4 A Centorid-Oriented Classifier

This section will specify how to update the target centroids via target samples with robust pseudo-labels and use the latest target centroids to generate pseudo-labels for the target data.

I utilize the target features extracted by M_{train} and the pseudo-labels generated by M_{init} to estimate the target centroids. The idea of a centroid-oriented classifier was first introduced in [12] where the algorithm uses all the target samples to form the centroids. Our approach uses only selected target samples with posterior probability of correct labeling $P_\phi(Z_j = 1|G_{init}(x_j^t), \tilde{y}_j^t) \geq \tau$, where τ is a filtering threshold. In practice we set τ to 0.5, for reasons discussed in Section 5.6. I update the target centroids via the EMA strategy [2, 12]:

$$C_k = \rho C_k^{local} + (1 - \rho)C_k, k = 1, 2, 3, \dots K \quad . \quad (4.14)$$

where $C_k^{local} = \frac{\sum_{j=1}^{B_t} \mathbb{1}_{\{\tilde{y}_j^t=k, \Upsilon_j \geq 0.5\}} G_{train}(x_j^t)}{\sum_{j=1}^{B_t} \mathbb{1}_{\{\tilde{y}_j^t=k, \Upsilon_j \geq 0.5\}}}$ is the local class centroid of class k generated from the mini-batch target samples.

Once finishing updating the target centroids, I assign the target samples to the nearest target centroid based on the cosine distance as:

$$\hat{y}_j^t = \operatorname{argmin}_{k=1}^K \operatorname{CosDist}(G_{train}(x_j^t), C_k) \quad . \quad (4.15)$$

By treating the updated pseudo-labels as ground-truth labels, I employ them to the standard cross-entropy loss function as follows:

$$\mathcal{L}_{CO} = \frac{\lambda}{N_t} \sum_{j=1}^{N_t} \mathcal{J}(F_{train}(G_{train}(x_j^t)), \hat{y}_j^t) \quad . \quad (4.16)$$

The coefficient λ is a linear scheduler that will gradually grow from 0 to 1 to minimize the effect of incorrect target centroids during the early stages of training.

4.2.5 A Target-Oriented Classifier

Unlike the centroid-oriented classifier, the target-oriented classifier only uses target samples with $\Upsilon > \tau$ to train the classifier. By treating these robust pseudo-labels as the ground-truth labels, I employ them to a weighted cross-entropy loss:

$$\mathcal{L}_{TO} = \frac{1}{\sum_{j=1}^{N_t} \mathbb{1}_{\{\Upsilon_j \geq \tau\}} \Upsilon_j} \sum_{j=1}^{N_t} \mathbb{1}_{\{\Upsilon_j \geq \tau\}} \Upsilon_j \mathcal{J}(F_{train}(G_{train}(x_j^t)), \tilde{y}_j^t) \quad . \quad (4.17)$$

At last, by choosing one of those classifiers, the final loss function can be viewed as follow:

$$\min_{\theta_{G_{train}}, \theta_{F_{train}}} \mathcal{L}_{src} + \mathcal{L}_{CO|TO} \quad , \quad (4.18)$$

where $\mathcal{L}_{src} = \frac{1}{N_s} \sum_{i=1}^{N_s} \mathcal{J}(F_{train}(G_{train}(x_i^s)), y_i^s)$, and where $\mathcal{L}_{CO|TO}$ means one can use either of the loss functions defined above. The pseudo-code of the proposed method is given in Algorithm 1.

Algorithm 1: Auxiliary Classifier Trained by Robust Target pseudo-Labels, $m = \{0, 1, \dots\}$ denotes the training steps and $l = \{0, 1, \dots\}$ denotes the iteration times. $g_{(m)}^t = \{G_{init}(x_j^t)\}_{j=1}^{N_t}$ and $\tilde{y}_{(m)}^t = \{\tilde{y}_j^t\}_{j=1}^{N_t}$ denotes all the target features and pseudo-labels generated by $M_{init}^{(m)}$. $M_{init}^{(m)}$ and $M_{train}^{(m)}$ stand for the m th version of M_{init} and M_{train} respectively. B_s and B_t denote the mini-batch training set

Given: Labeled source samples $D_s = \{(x_i^s, y_i^s)\}_{i=1}^{N_s}$ and unlabeled target samples

$$D_t = \{x_j^t\}_{j=1}^{N_t}$$

Output: $\theta_{G_{train}}, \theta_{F_{train}}$

Step-1:

```

1 m = 0
2 Optimize  $\theta_{G_{init}}, \theta_{F_{init}}$  using Eq.4.4, output:  $M_{init}^{(m)}$ 
3 for m ← 0 to max_training_steps do
4   Generate  $g_{(m)}^t$  and  $\tilde{y}_{(m)}^t$  by using  $M_{init}^{(m)}$ 
5   Step-2:
6   l = 1
7   Initialize  $\{\pi_k^{(0)}, \sigma_k^{(0)}, \delta_k^{(0)}\}_{k=1}^K$  using Eq 4.7
8   do
9     Compute  $\{\Upsilon_j^{(l)}\}_{j=1}^{N_t}$  using Eq 4.8
10    Compute  $\{\pi_k^{(l)}\}_{k=1}^K$  using Eq 4.9
11    Compute  $\{\sigma_k^{(l)}\}_{k=1}^K$  using Eq 4.10
12    Compute  $\{\delta_k^{(l)}\}_{k=1}^K$  using Eq 4.11
13    l = l + 1
14  while  $|\Upsilon_j^{(l)} - \Upsilon_j^{(l-1)}| > 1^{-10}$  and l < max_iter
15  Step-3:
16  m = m + 1
17  Initialize a fresh model:  $M_{train}^{(m)}$ 
18  while Not Converge do
19    Derive  $B_s$  and  $B_t$  samples from  $D_s$  and  $D_t$ 
20    Extract the corresponding target pseudo-labels and their probability of being
      correctly labeled from  $\tilde{y}_{(m)}^t$  and  $\Upsilon^{(l)}$ 
21    if Centroid-oriented classifier then
22      Based on Eq 4.14, update the target centroids by using target samples with
        high-confident pseudo-label.
23      Based on Eq 4.15, use updated target centroids to assign a new pseudo-label
        to each target sample.
24      Using Eq 4.16 to get  $L_{CO}$ 
25    else
26      Using Eq 4.17 to get  $L_{TO}$ 
27    Update  $M_{train}^{(m)}$  by optimizing Eq 4.18
28   $M_{init}^{(m)} = M_{train}^{(m)}$ 

```

Chapter 5

Experiments

5.1 DA Benchmarks

I have evaluated the methods proposed above on standard DA benchmarks through computational experiments and compared these results with results from a set of widely-accepted DA methods.

In this paper, I denote all the transfer tasks in the form: **source domain** $\rightarrow$ **target domain**. The DA benchmark dataset used and a short description of each dataset are given below.

- **Office-31** dataset is a widely-used DA benchmark containing 4,110 images of 31 categories shared by 3 different domains: Amazon (**A**) which contains images collected from amazon.com, Webcam (**W**) and DSLR (**D**) which contains images taken by webcam or SLR camera respectively. In this benchmark, I will evaluate the proposed methods on all 6 transfer learning tasks: these tasks are **A** $\rightarrow$ **W**, **A** $\rightarrow$ **D**, **W** $\rightarrow$ **A**, **W** $\rightarrow$ **D**, **D** $\rightarrow$ **A**, and **D** $\rightarrow$ **W**.
- **Office-Home** dataset is a new dataset, and it is more challenging than Office-31. It consists of 15,588 images of 65 categories shared by 4 extremely different domains: Artistic images (**Ar**), Clip Art images (**Cl**), Product images (**Pr**) and Real-World images (**Rw**). Similarly, I will evaluate the proposed methods on all domain combinations

and construct 12 transfer learning tasks.

Figure 5.1 and Figure 5.2 illustrate how the samples look like in the Office-31 and Office-Home datasets.



Figure 5.1: Sample Images from Office-31 Dataset



Figure 5.2: Sample Images from Office-Home Dataset

5.2 Baseline methods

To be more specific about how I perform the comparison with baseline methods, I include the following details of the experiments. For Office-31, I compare the proposed methods with the standard transfer learning approaches and standard deep learning approaches: deep convolutional neural network (ResNet-50) [9], deep adaptation network (DAN) [15], deep domain

confusion (DDC) [22], deep CORAL (D-CORAL) [20], domain adversarial training network (DANN) [5], conditional adversarial domain adaptation (CDAN and CDAN+E) [13], Collaborative and adversarial network (iCAN) [26], Stepwise Adaptive Feature Norm (SAFN+E) [24], multi-adversarial domain adaptation (MADA) [18], joint adaptation network (JAN) [14], and Deep subdomain adaptation (DSAN) [27]. For Office-Home, I compared the proposed method with ResNet-50, DAN, DANN, CDAN, JAN, and DSAN. Except for ResNet-50, the results of all the remaining baseline methods are obtained from the original papers.

5.3 Implementation details

Here are some details for the implementation I employed to obtain the experimental results presented below. The accuracy reported is the average of three randomized trials. I use ResNet-50 as the backbone [9] and add a bottleneck layer with 256 units after the last average pooling layer. All the baseline methods also use ResNet-50 as the backbone to ensuring a fair comparison. I resize the images to be 256x256 pixels and then follow the resizing by randomly cropping the images (into 224x224 pixels), and then random flipping the images (as introduced in JAN [14]). Finally, I normalize the RGB channels of the images with the means of [0.485, 0.456, 0.406] and standard deviations of [0.229, 0.224, 0.225]. For the centroid-oriented classifier, to mitigate the effect of incorrect centroids in the early training stage, the hyperparameter λ is gradually increased from 0 to 1 as follows: $\lambda = \frac{\text{Current_iteration}}{\text{Maximum_iteration}}$. For the filtering threshold τ , I set it to 0.5 for the results presented in this paper. For an explanation as to why this choice is made, in Section 5.6, I will evaluate the performances of choosing different values of τ .

I adopt a mini-batch Stochastic Gradient Descent (SGD) algorithm with a momentum of 0.9 to train the feature extractor and the classifier for all the tasks. The feature extractor is the

ImageNet pre-trained model, and the classifier is a two-layer, randomly initialized network. The learning rate of the feature extractor is set to 0.001 (as it pre-trained), whereas the learning rate of the classifier is set to 0.01 (as it is randomly initialized). The mini-batch size is set to 36, and the learning rates gradually decrease with weight decay of 0.001 as introduced in CDAN [13].

5.4 Experimental Results

Table 5.1 and table 5.2 show the classification results for Office-31 and Office-Home. As the following results demonstrate, the proposed method outperforms the comparison methods on the majority of the transfer tasks. Additionally, there are several observations from these results that I would like to give additional insight into the performance of these algorithms.

1. Without doing any alignment or adversarial training, M_{train} still performs quite well for most transfer tasks. This fact points to the importance of the robust pseudo-labels, and it makes apparent the following observation: by treating those robust pseudo-labels as the ground-truth labels, we can change our problem setting from unsupervised learning into semi-supervised learning. In Figure 3.6, I illustrate the performance difference between unsupervised and semi-supervised learning (with eight labeled target data per category). This figure clearly shows that with the help of very few labeled target data, semi-supervised adaptation can greatly outperform unsupervised adaptation for all tasks on the Office-31 dataset.
2. Compared with non-adversarial adaptation methods [14, 15, 20, 22], the proposed method significantly outperforms the comparison methods on all of the 18 transfer tasks.

3. When compared with the subdomain adaptation methods [13, 18, 27], the proposed method is superior for most of the transfer tasks. This improvement results from the usage of the robust pseudo-labels, which help to incorporate the fine-grained information for each category into the training process.

In sum, the above-mentioned observations give additional insight, and the overall experimental results demonstrate, the effectiveness of the proposed method.

Table 5.1: ACCURACY (%) on Office-31 dataset for unsupervised domain adaptation (RESNET-50)

Method	A→W	W→A	A→D	D→A	W→D	D→W	Avg
ResNet-50[9]	68.4±0.2	60.7±0.3	68.9±0.2	62.5±0.3	99.3±0.1	96.7.4±0.5	76.1
DAN[15]	83.8±0.4	62.7±0.2	78.4±0.2	66.7±0.3	99.5±0.1	96.8±0.2	81.3
DDC[22]	75.8±0.2	64.0±0.5	77.5±0.3	67.4±0.4	98.2±0.1	95.0±0.2	79.7
D-CORAL[20]	77.7±0.3	64.0±0.4	81.1±0.4	64.6±0.3	99.7±0.1	97.6±0.2	80.8
DANN[5]	82.0±0.4	67.4±0.5	79.7±0.4	68.2±0.4	99.1±0.1	96.9±0.2	82.2
iCAN[26]	92.5	69.9	90.1	72.1	100	98.8	87.2
MADA[18]	90.0±0.1	66.4±0.3	87.8±0.2	70.3±0.3	99.6±0.2	97.4±0.1	85.2
JAN[14]	85.4±0.3	70.0±0.4	84.7±0.3	68.6±0.3	99.8±0.2	97.4±0.2	84.3
SAFN+E[24]	90.1±0.8	70.2±0.3	90.7±0.5	73.0±0.2	99.8±0.0	98.6±0.2	87.1
CDAN[13]	93.1±0.2	68.0±0.4	89.8±0.3	70.1±0.4	100.0±0.0	98.2±0.2	86.6
CDAN+E[13]	94.1±0.1	69.3±0.32	92.9±0.2	71.0±0.3	100.0±0.0	98.6±0.1	87.7
DSAN[27]	93.6±0.2	74.8±0.4	90.2±0.7	73.5±0.5	100.0±0.0	98.3±0.1	88.4
M_{init}	90.7	70.8	90.36	69.2	100	98.62	86.62
M_{train} w/ target-oriented classifier(TO)	94.84	73.59	93.46	73.09	99.6	98.87	88.91
M_{train} w/ Centroid-oriented classifier(CO)	94.47	74.36	95.78	73.87	99.6	98.74	89.43

Table 5.2: ACCURACY (%) on Office-Home dataset for unsupervised domain adaptation (RESNET-50)

Method	A→C	A→P	A→R	C→A	C→P	C→R	P→A	P→C	P→R	R→A	R→C	R→r	Avg
ResNet-50[9]	34.9	50.0	58.0	37.4	41.9	46.2	38.5	31.2	60.4	53.9	41.2	59.9	46.1
DAN[15]	43.6	57.0	67.9	45.8	56.5	60.4	44.0	43.6	67.7	63.1	51.5	74.3	56.3
DANN[5]	45.6	59.3	70.1	47.0	58.5	60.9	46.1	43.7	68.5	63.2	51.8	76.8	57.6
JAN[14]	45.9	61.2	68.9	50.4	59.7	61.0	45.8	43.4	70.3	63.9	52.4	76.8	58.3
CDAN[13]	49.0	69.3	74.5	54.4	66.0	68.4	55.6	48.3	75.9	68.4	55.4	80.5	63.8
CDAN+E[13]	50.7	70.6	76.0	57.6	70.0	70.0	57.4	50.9	77.3	70.9	56.7	81.6	65.8
DSAN[27]	54.4	70.8	75.4	60.4	67.8	68.0	62.6	55.9	78.5	73.8	60.6	83.1	67.6
M_{init}	57.3	74.31	77.83	64.2	73.4	71.7	64.3	54.6	79.4	71.6	59.2	83.83	69.30
M_{train} w/ TO	58.7	78.13	80.65	65.4	76.03	75.56	66.01	56.07	81.25	73.3	60.36	85.48	71.41
M_{train} w/ CO	56.76	77.81	80.45	67.28	77.59	77.56	66.75	54.98	81.78	73.23	60.46	84.88	71.63

5.5 Ablation Study

I conducted an ablation study on the Office-31 and Office-Home benchmarks to demonstrate the effectiveness of the usage of the robust pseudo-labels. Results presented in tables 5.3 and 5.4 compare the performance of the proposed method with and without the usage of robust pseudo-labels (in these tables (w/R) and (wo/R) respectively stand for using and not using the robust pseudo-labels during the training process). For the case of target-oriented (TO) classifier without pseudo-labels, I treat all the pseudo-labels generated by M_{init} as ground-truth labels and employ them to a standard cross-entropy loss. For the case of the centroid-oriented (CO) classifier without pseudo-labels, I use the pseudo-labels generated by M_{train} to form the target centroids. These results demonstrate that for the majority of the transfer tasks, the performance of both the centroid-oriented classifier and the target-oriented classifier is improved with the use of robust pseudo-labels. For scenarios where the test accuracy is relatively high, for example, the Office-31 A→D scenario, I observe that the centroid-oriented classifier without robust pseudo-labels can achieve a better performance than with the use of robust pseudo-labels. Intuitively, this difference may be because the prediction accuracy is already high enough. Only choosing to use the target samples with robust pseudo-labels may filter out too many candidates labeled correctly. We can observe a similar phenomenon for the D→W and W→D scenarios.

Although filtering out target samples with weak pseudo-labels might be slightly harmful when the test accuracy is already high (around 96%~100%), it does show a significant performance improvement when having a low test accuracy (around 50%~76%). In table 5.4, we can observe that both the target-oriented classifier and centroid-oriented classifier achieve a performance improvement by incorporating the robust pseudo-label mechanism into the training process. These results demonstrate the effectiveness of robust pseudo-labels for

scenarios where the test accuracy is relatively low. In real-world applications where we do not know the test accuracy (since the target domain is unlabeled), we can still develop a general rule-of-thumb for selecting the target samples' filtering threshold (τ). That is, when the source and target domains are similar, the filtering threshold should be relatively low to ensure that enough target samples participate in the training process. Additionally, the filtering threshold should be relatively high when those two domains are extremely different to ensure that only the target samples with robust pseudo-labels can participate in the training process.

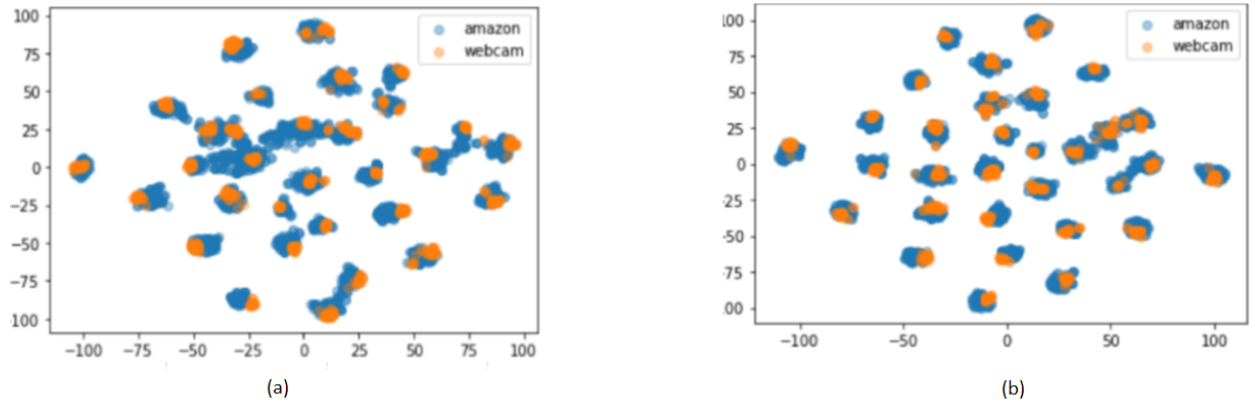
Table 5.3: ACCURACY (%) on the Office-31 dataset for the ablation study (RESNET-50)

Method	A→W	W→A	A→D	D→A	W→D	D→W	Avg
M_{train} w/ TO(w/R)	92.08	70.22	88.96	70.32	100	98.49	86.67
M_{train} w/ CO(w/R)	92.70	73.09	96.18	72.4	99.6	99.12	88.84
M_{train} w/ TO(w/R)	94.84	73.59	93.46	73.09	99.6	98.87	88.91
M_{train} w/ CO(w/R)	94.47	74.36	95.78	73.87	99.6	98.74	89.43

Table 5.4: ACCURACY (%) on the Office-Home dataset for the ablation study (RESNET-50)

Method	A→C	A→P	A→R	C→A	C→P	C→R	P→A	P→C	P→R	R→A	R→C	R→P	Avg
M_{train} w/ TO(w/R)	57.18	75.2	78.88	64.28	73.2	73.19	65.02	55.35	79.41	71.8	59.4	83.89	69.73
M_{train} w/ CO(w/R)	54.5	77.19	79.86	66.86	76.86	75.8	65.64	51.75	80.31	72.38	57.25	84.34	70.22
M_{train} w/ TO(w/R)	58.7	78.13	80.65	65.4	76.03	75.56	66.01	56.07	81.25	73.3	60.36	85.48	71.41
M_{train} w/ CO(w/R)	56.76	77.81	80.45	67.28	77.59	77.56	66.75	54.98	81.78	73.23	60.46	84.88	71.63

Figure 5.3: The t-SNE visualization of the extracted features for target-oriented classifier with (a) and without (b) robust target pseudo-labels for the A→W scenario.



To conclude our ablation study, I use t-SNE embedding [23] to visualize the extracted features learned by target-oriented classifier with and without robust pseudo-labels in task $A \rightarrow W$ scenario (Office-31). Figure 5.3(a) illustrates the result of target-oriented classifier without pseudo-labels. We can observe that the source and target domain are not well aligned, and that some of the data appear to be difficult to classify. On the contrary, Figure 5.3(b) illustrates the feature representation learned by the target-oriented classifier with pseudo-labels. We can observe that not only are the subdomains with the same class in different domains well aligned but, additionally, subdomains with different classes are also well separated. These visualizations illustrate that robust pseudo-labels can help capture more fine-grain information for each category. Also, without doing any kind of alignment, by using the target samples with robust pseudo-labels to train the network, we can still achieve a good alignment of the distributions.

5.6 Choice of the Filtering Threshold Parameter τ

For the algorithm to work as a “black box” for any application, we must make a choice for the value for the filtering threshold τ to be used. In this section, I will investigate the effect of varying τ on the algorithm’s accuracy. In Figure 5.4, I show the accuracy of the transfer tasks for the Office-31 dataset (using target-oriented classifier) as a function of τ . For the $A \rightarrow W$ and $A \rightarrow D$ scenarios, we can observe that the optimal result is achieved for $\tau = 0.5$. For the $W \rightarrow A$ and $D \rightarrow A$ scenarios, the optimal results are achieved at $\tau = 0.7$. We clearly want to avoid the steep drop-off in accuracy for choosing too large a value for τ , yet at the same time, we want to ensure that our choice of τ yields an overall improved accuracy. The steep drop-off in accuracy for large τ is because this choice will filter more and more of the target samples and ultimately degenerate the whole network to the unsupervised setting. As

we observed in the previous subsection, this would significantly decrease the overall accuracy. Regarding the choice of τ , my intuition is that when the overall accuracy is relatively low, the

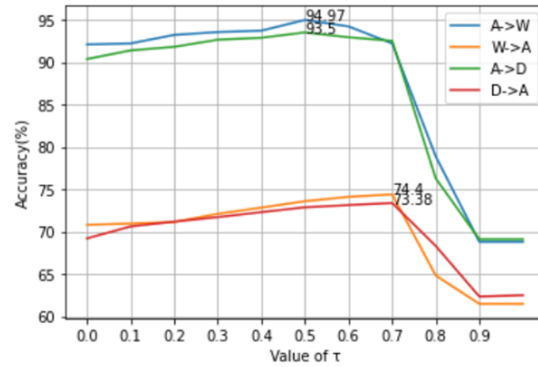


Figure 5.4: The observed accuracy for the Office-31 dataset using the TO classifier as function of the filtering threshold parameter τ .

threshold should be relatively high to ensure only the samples with robust pseudo-labels can participate in the training process. Then again, if the overall accuracy is relatively high, then the threshold can be relatively low because the majority of the pseudo-labels are accurate, and these data can safely be used in the training process. As a compromise, I recommend setting $\tau = 0.5$ for general applications since there is no way of knowing the overall accuracy, and for both low and high accuracy scenarios, this choice of τ performs well.

Chapter 6

Conclusions

In this thesis, I presented two types of novel classifiers using robust target pseudo-labels: a target-oriented classifier and a centroid-oriented classifier. These classifiers employ two algorithmic steps: (1) use the Gaussian-uniform mixture model to select the target samples with robust pseudo-labels, and (2) use these selected samples to train the network directly by employing a weighted cross-entropy loss (for the target-oriented classifier) or estimating the target centroids (for the centroid-oriented classifier).

Previously proposed methods consist of a single phase that attempts to align the source and target distributions, either by using a global or a sub-domain alignment scheme. Unlike these methods, the method proposed in this thesis only uses this alignment strategy initially, then the method switches to a robust target pseudo-label approach for the final training stages. One can motivate this approach by thinking of it as a two-stage approach that switches from: (1) the first phase of unsupervised learning to (2) the second phase of semi-supervised learning (where we treat the robust pseudo labels as the ground-truth labels). Compared with previously proposed adversarial approaches that can employ several loss functions, the method proposed in this thesis has a simple loss function and is shown to converge rapidly. Extensive experimental results demonstrate, for both the Office-31 and Office-Home datasets, the overall improved effectiveness of the proposed method.

Bibliography

- [1] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1):151–175, 2010.
- [2] Chaoqi Chen, Weiping Xie, Wenbing Huang, Yu Rong, Xinghao Ding, Yue Huang, Tingyang Xu, and Junzhou Huang. Progressive feature alignment for unsupervised domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 627–636, 2019.
- [3] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
- [4] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *International conference on machine learning*, pages 1180–1189. PMLR, 2015.
- [5] 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.
- [6] 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.
- [7] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and

- Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.
- [8] 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.
- [9] 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.
- [10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>.
- [11] Stéphane Lathuilière, Pablo Mesejo, Xavier Alameda-Pineda, and Radu Horaud. Deepgum: Learning deep robust regression with a gaussian-uniform mixture model. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 202–217, 2018.
- [12] Jian Liang, Dapeng Hu, and Jiashi Feng. Domain adaptation with auxiliary target domain-oriented classifier. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16632–16642, 2021.
- [13] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. Conditional adversarial domain adaptation. *arXiv preprint arXiv:1705.10667*, 2017.
- [14] 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.
- [15] Mingsheng Long, Yue Cao, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. Transferable representation learning with deep adaptation networks. *IEEE transactions on pattern analysis and machine intelligence*, 41(12):3071–3085, 2018.
- [16] Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.
- [17] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.
- [18] Zhongyi Pei, Zhangjie Cao, Mingsheng Long, and Jianmin Wang. Multi-adversarial domain adaptation. In *Thirty-second AAAI conference on artificial intelligence*, 2018.
- [19] Jake Snell, Kevin Swersky, and Richard S Zemel. Prototypical networks for few-shot learning. *arXiv preprint arXiv:1703.05175*, 2017.
- [20] Baochen Sun and Kate Saenko. Deep coral: Correlation alignment for deep domain adaptation. In *European conference on computer vision*, pages 443–450. Springer, 2016.
- [21] Antonio Torralba and Alexei A Efros. Unbiased look at dataset bias. In *CVPR 2011*, pages 1521–1528. IEEE, 2011.
- [22] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *arXiv preprint arXiv:1412.3474*, 2014.
- [23] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

- [24] Ruijia Xu, Guanbin Li, Jihan Yang, and Liang Lin. Larger norm more transferable: An adaptive feature norm approach for unsupervised domain adaptation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1426–1435, 2019.
- [25] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *arXiv preprint arXiv:1411.1792*, 2014.
- [26] Weichen Zhang, Wanli Ouyang, Wen Li, and Dong Xu. Collaborative and adversarial network for unsupervised domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3801–3809, 2018.
- [27] Yongchun Zhu, Fuzhen Zhuang, Jindong Wang, Guolin Ke, Jingwu Chen, Jiang Bian, Hui Xiong, and Qing He. Deep subdomain adaptation network for image classification. *IEEE transactions on neural networks and learning systems*, 32(4):1713–1722, 2020.