Project4 Report: Domain Adaptation

Hou Shengyuan 518021910604
Yan Binghao 518021910753
Wu Shiqu 518021910665

Abstract. Domain Adaptation is one of the branches of migration learning. Its purpose is to map differently distributed source and target domain data into a feature space, minimizing the distance between the two in that space. In this report, we use Office-Home dataset as our dataset, and try traditional methods (TCA, JDA, CORAL, BDA, WBDA) and deep learning methods (DANN, DDC DeepCORAL) to do domain adaptation and evaluate the performances of them.

Keywords: Domain Adaptation, TCA, JDA, CORAL, BDA, WBDA, DANN, DDC DeepCORAL

1 Introduction

For traditional machine learning algorithms, a very important premise is that training data and test data must be independent and identically distributed. Obviously, this premise is not always true. If the distribution of test data and training data is significantly different, then it is conceivable that traditional machine learning models may not perform well.

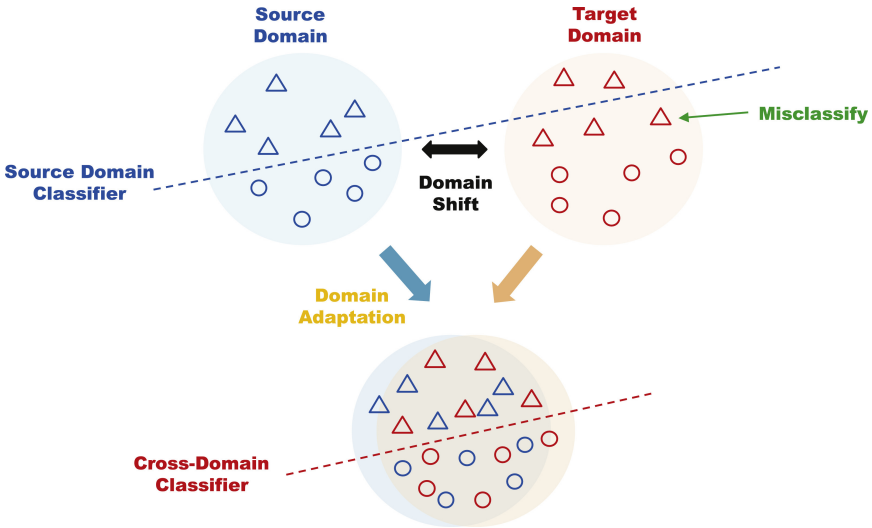

Fig. 1: Domain Adaptation

Therefore, transfer learning came into being, and in this project we will focus on a sub-category of transfer learning-domain adaptation. The basic idea of Domain Adaptation is to map the data of the source domain and the target domain to the same feature space (the source domain and the

target domain have different data distributions). Then find a measurement criterion in the feature space to make the feature distribution of the source domain and target domain data as similar as possible. Therefore, a discriminator trained based on the characteristics of the source domain data can be used on the target domain data.

In this project, we have implemented several domain adaptation methods, covering traditional methods (TCA, JDA, CORAL, BDA, WBDA) and deep learning methods (DANN, DDC DeepCORAL). The schematic diagram is as follows.

Fig. 2: Methods implemented in the project

2 Domain Adaptation Related Surveys

Domain adaptation is a field associated with machine learning and transfer learning. Generally speaking, depending on whether the target domain data is labeled, there are three types of domain adaptation as follows:

- **Unsupervised domain adaptation:** the learning sample contains a set of labeled source domain examples with only unlabeled data in the target domain.
- **Semi-supervised domain adaptation:** in this situation, we also consider a "small" set of labeled target examples.
- **Supervised domain adaptation:** all the examples considered are supposed to be labeled.

On the other hand, we also found that domain adaptation algorithms have the following categories:

- **Reweighting algorithm:** The objective is to reweight the source labeled sample such that it "looks like" the target sample (in terms of the error measure considered).
- **Iterative algorithms:** A method for adapting consists in iteratively "auto-labeling" the target examples.
- **Common representation space algorithms:** The goal is to find or construct a common representation space for the two domains. The objective is to obtain a space in which the domains are close to each other while keeping good performances on the source labeling task. This can be achieved through the use of Adversarial machine learning techniques where feature representations from samples in different domains are encouraged to be indistinguishable.A well

known model of this type is DANN, which is selected as the Deep Domain Adaptation Method for this assignment.

- **Hierarchical Bayesian Model:** The goal is to construct a Bayesian hierarchical model $p(n)$, which is essentially a factorization model for counts n, to derive domain-dependent latent representations allowing both domain-specific and globally shared latent factors.

As for the specific methods of domain adaptation, we also summarized according to the characteristics:

- **Distribution Adaptation:** Minimize the probability distribution distance so that the probability distribution of the source domain and the target domain are similar.
  - **Marginal Distribution Adaptation:**
    - TCA (Transfer Component Analysis)
    - MMD (Maximum Mean Discrepancy)
    - DDC (Deep Domain Confusion): MMD + Neural Network
    - DAN (Deep Adaptation Networks): MKK-MMD + Neural Network
    - DME (Distribution-Matching Embedding): matrix transformation + projection
    - CMD (Central Moment Discrepancy): K-order MMD
  - **Conditional Distribution Adaptation:**
    - CTC (Conditional Transferrable Components)
  - **Joint Distribution Adaptation:**
    - JDA (Joint Distribution Adaptation): TCA + Conditional distribution adaptation
    - ARTL (Adaptation Regularization): JDA + Classifier learning
    - VDA (Visual Domain Adaptation): JDA+ Inner class distance + Among class distance
    - JGSA (Joint Geometrical and Statistical Alignment): JDA + Inner class distance + Among class distance + Label adaptation
    - JAN (Joint Adaptation Networks): JDA + JMMD, used in deep network
    - BDA (Balanced Distribution Adaptation): Add the balance factor to dynamically measure the importance of the two distributions
- **Feature Selection:** Select and extract shared features from the source domain and target domain, and establish a unified model.
  - SCL (Structural Correspondence Learning)
  - TJM (Transfer Joint Matching): MMD Distribution Adapation + Source Domain Sampling
  - FSSL (Feature Selection and Structure Preservation): Feature Selection + Information Immutability
- **Subspace Learning:** Transform the source domain and target domain to the same subspace, and then build a unified model.
  - **Statistical Characteristics Alignment**
    - SA (Subspace Alignment): Directly seek a linear transformation, which transform the source into the target space
    - SDA (Subspace Distribution Alignment): SA + Prob Distribution Adaptation
    - CORAL (CORrelation Alignment): Minimize the second-order statistical characteristics of the source and target domains
    - Deep-CORAL: CORAL + DNN
  - **Manifold Learning**
    - SGF (Sample Geodesic Flow)
    - GFK (Geodesic Flow Kernel): SGF + Gaussian Kernel

- ◇ DIP (Domain-Invariant Projection)

. **Deep Learning:**

- ◇ Domain-Adversarial Neural Network (DANN): ICCV 2014
- ◇ Deep Adaptation Networks (DAN): ICML 2015
- ◇ Simultaneous feature and task transfer: ICCV 2015
- ◇ Joint Adaptation Networks (JAN): ICML 2017
- ◇ Deep Hashing Network (DHN): CVPR 2017
- ◇ Adversarial Discriminative Domain Adaptation (ADDA): arXiv 2017
- ◇ Learning to Transfer (L2T): arXiv 2017
- ◇ Label Efficient Learning of Transferable Representations across Domains and Tasks: NIPS 2017

3 Traditional Methods

3.1 Transfer Component Analysis(TCA)

TCA is a feature-based transfer learning method. From an academic point of view, TCA maps the source and target data in different data distributions to a high-dimensional Reproducing Kernel Hilbert Space (RKHS). [1] In this space, TCA minimizes the data distance between the source and target, while retaining their respective internal attributes to the greatest extent. In layman's terms, in the face of the different distributions of the source domain and the target domain, that is, $P(X_S) \neq P(X_T)$, TCA assumes that there is a mapping $\phi(\cdot)$, so that the distribution of the data after the mapping is $P(\phi(X_S) \approx P(\phi(X_T))$, or further, there is $P(Y_S|\phi(X_S) \approx P(Y_T|\phi(X_T))$. The specific steps of TCA are as follows:

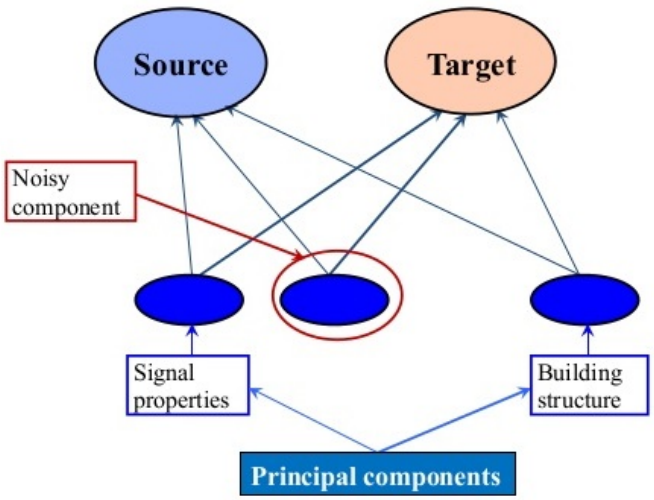

Fig. 3: Transfer Component Analysis

Step 1. Define optimization goals based on MMD

The distance metric used by TCA is the maximum mean difference (MMD), and the formula is as follows (n_1, n_2 represent the number of samples of X_S, X_T respectively):

$$DIS\left(X_S, X_T\right) = \left\| \frac{1}{n_1} \sum_{i=1}^{n_1} \phi\left(x_{S_i}\right) - \frac{1}{n_2} \sum_{i=1}^{n_2} \phi\left(x_{T_i}\right) \right\|_{\mathcal{H}}^2 \tag{1}$$

It is not difficult to find that after the square expansion of the above formula, there is a part of the product of the quadratic term. Here we contact the kernel function learned in SVM and solve it with the help of it. Therefore, TCA introduces the kernel matrix K and the function L:

$$K = \begin{bmatrix} K_{S,S} & K_{S,T} \\ K_{T,S} & K_{T,T} \end{bmatrix} \in \mathbb{R}^{(n_1+n_2)\times(n_1+n_2)}$$

$$L_{\mathrm{ij}} = \begin{cases} \frac{1}{n_1^2} & x_i, x_j \in X_s \\ \frac{1}{n_2^2} & x_i, x_j \in X_T \\ -\frac{1}{n_1 n_2} & \text{otherwise} \end{cases}$$

Therefore, each row of the matrix K is multiplied by each column of the matrix L, and then added, which is the formula (1). In other words, the distance function is equivalent to:

$$D\mathrm{IS}\left(X_s, X_t\right) = tr\left(\mathrm{KL}\right) - \lambda \mathrm{tr}\left(K\right) \tag{2}$$

Step 2. Dimensionality reduction to define a new objective function(Parametric Kernel Map for Unseen Patterns)

K is a positive definite matrix, which can be decomposed into $K = (K^{1/2})^2$. K is $(n_1 + n_2)$ dimensions. In order to reduce the kernel matrix to m dimensions ($m < n_1 + n_2$), define a matrix $\tilde{W} \in \mathbb{R}^{(n_1+n_2)\times m}$, at this time the new kernel matrix can be expressed as:

$$\tilde{K} = \left(KK^{-\frac{1}{2}}\tilde{W}\right)\left(\tilde{W}^T K^{-\frac{1}{2}} K\right) = \mathrm{KW}W^T K$$

Where $W = K^{-\frac{1}{2}}\tilde{W}$, the distance function becomes:

$$D\mathrm{IS}\left(X_s, X_t\right) = tr\left(\left(\mathrm{KW}W^T K\right)L\right) = \mathrm{tr}\left(W^T KL\mathrm{KW}\right)$$

Here, the nature of trace calculation $\mathrm{tr}(\mathbf{AB}) = \mathrm{tr}(\mathbf{BA})$ is used. And at this point, the objective function is defined as:

$$\min_W \mathrm{tr}\left(W^\top KLKW\right) + \mu\,\mathrm{tr}\left(W^\top W\right) \tag{3}$$

$$\text{s.t.} \quad W^T KH\mathrm{KW} = I_m \tag{4}$$

Here $\mathrm{tr}\left(W^T W\right)$ is used as a penalty term to reduce the complexity of W, μ is the regularization control parameter, H is the center matrix($H = \mathbf{I}_{n_1+n_2} - \frac{1}{nn_1+n_2}\mathbf{1}\mathbf{1}^T$, $\mathbf{I}_{n_1+n_2} \in \mathbb{R}^{(n_1+n_2)\times m}$), and formula (3) is the constraint condition, Keep the divergence constant.

3.2 Joint Distribution Adaptation(JDA)

Joint Distribution Adaptation(JDA) [2] can be viewed as a generalized version of TCA. Let the set of all labels be $\{1, 2, \cdots, C\}$ and $D_s^{(c)}$ be the set of samples of class c in the source data. We generate pseudo labels for samples from the target domain data with some base classifiers like SVM or even TCA to form the sets $D_t^{(c)}$ for target domain. The optimization target is the MMD distance between the class-conditional distributions $Q_s(x_s|y_s = c)$ and $Q_t(x_t|y_t = c)$.

$$||\frac{1}{|D_s^{(c)}|}\sum_{x\in D_s^{(c)}} A^T x - \frac{1}{|D_t^{(c)}|}\sum_{x\in D_t^{(c)}} A^T x||^2 = tr(A^T X M_c X^T A)$$

where A is the projection matrix in PCA. Let $n_s^{(c)}$ be $|D_s^{(c)}|$ and $n_t^{(c)}$ be $|D_t^{(c)}|$, M_c is given by

$$(M_c)_{\mathrm{ij}} = \begin{cases} \frac{1}{n_s^{(c)} n_s^{(c)}} & x_i, x_j \in D_s^{(c)} \\ \frac{1}{n_t^{(c)} n_t^{(c)}} & x_i, x_j \in D_t^{(c)} \\ -\frac{1}{n_s^{(c)} n_t^{(c)}} & x_i \in D_s^{(c)}, x_j \in D_t^{(c)} \\ 0 & \text{otherwise} \end{cases}$$

The final optimization problem is

$$\begin{aligned} &\min\ \mathrm{tr}\left(A^T X M_c X^T A\right) + \lambda||A||_F^2 \\ &\text{s.t.}\ \ A^T X M_c X^T A = I \end{aligned}$$

Borrowing notation from **Section 2.1**, $M_0 = L$ in TCA. It is obvious that TCA is a special case of JDA when $C = 0$.

3.3 Balanced Distribution Adaptation(BDA)

Balanced Distribution Adaptation (BDA) [3] mainly completes domain adaptation by adaptively minimizing the marginal and conditional distribution differences between domains. Similar to TCA, facing the different distribution of source domain and target domain ($P(X_S) \neq P(X_T)$), BDA hopes to accomplish two goals: (1)$P(X_S) \approx P(X_T)$, (2)$P(Y_S|X_S) \approx P(Y_T|X_T)$.And the above objective is expressed by the formula as

$$D(D_S, D_T) \approx (1-\mu)D(P(X_S), P(X_T)) + \mu D(P(Y_S|X_S), P(Y_T|X_T))$$

where $\mu \in [0, 1]$ is a balance factor. When $\mu \to 0$, it means the datasets are more dissimilar, so the marginal distribution is more dominant; when $\mu \to 1$, it reveals the datasets are similar, so the conditional distribution is more important to adapt. D_S, D_T represent data marked as different labels in the source domain and target domain (similar to **Section 2.1**)

In addition, BDA also uses maximum mean discrepancy (MMD) as the distance metric. Therefore, the loss function is as follows:

$$D(D_S, D_T) \approx (1-\mu)||\frac{1}{n}\sum_{i=1}^{n} x_{s_i} - \frac{1}{m}\sum_{j=1}^{m} x_{t_j}||^2 + \mu||\frac{1}{n_c}\sum_{x_{s_i}\in D_s^{(c)}}^{n} x_{s_i} - \frac{1}{m_c}\sum_{x_{t_j}\in D_t^{(c)}}^{n} x_{t_j}||^2$$

3.4 CORrelation Alignment(CORAL)

CORAL [4] uses a linear transformation method to align the second-order statistical features of the source domain and target domain distribution. It is very effective for unsupervised domain adaptation.

Let us set the training sample of the source domain as $D_s = \{x_1^s, x_2^s, \cdots, x_n^s\}, x_i^s \in R^D$, with lables $L_s = \{y_1^s, y_2^s, \cdots, y_n^s\}, y_i^s \in \{1, \cdots, L\}$. Similarly, there are target domain samples $D_t = \{x_1^t, x_2^t, \cdots, x_n^t\}, x_i^t \in R^D$. In addition, we suppose C_S and C_T are the feature vector covariance matrices.

To minimize the distance between the second-order statistics (covariance) of the source and target features, we apply a linear transformation A to the original source features and use the Frobenius norm as the matrix distance metric(The intermediate derivations and proofs are omitted in this report):

$$\min_A ||C_{S'} - C_T||_F^2 = \min_A ||A^T C_S A - C_T||_F^2$$

where $C_{S'}$ is the covariance of the transformed source features $D_S A$ and $||\cdot||_F^2$ denotes the matrix Frobenius norm.

4 Deep Learning Methods

4.1 Domain-Adversarial Neural Network(DANN)

DANN, Domain Adversarial Neural Network [5], also known as RevGrad. [5] It is a classic Domain Adaptation based on Adversarial Network. The main idea is to use **H-Divergence** to measure the distance between two domains, and it is blended with the idea of GAN. The frame diagram is given below:

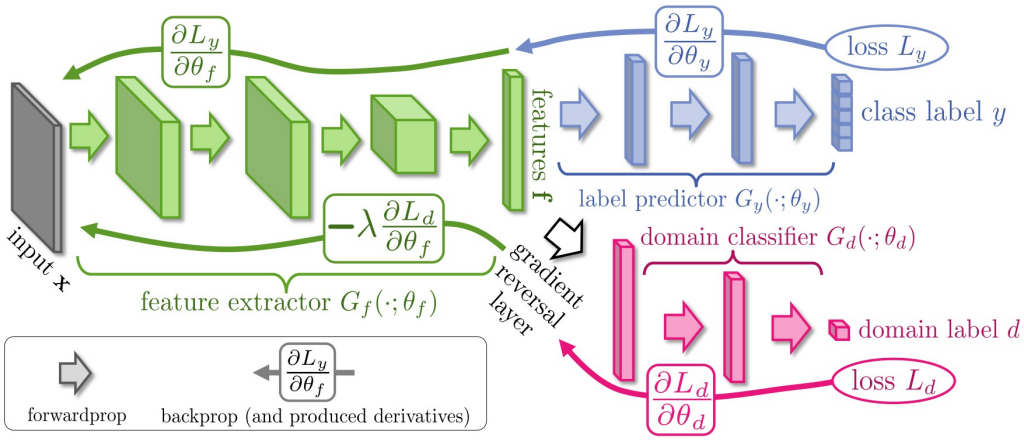

Fig. 4: DANN architecture, which includes a deep feature extractor (green), a deep label predictor (blue), and a domain classifier (red) .

There are three parts in this architecture:, **Feature Extractor**, **Label Predictor** and **Domain Classifier**. They together form a standard feed-forward network. Among them, the feature extractor and the classifier constitute the classic architecture of the traditional classification network, whose

goal is to maximize the classification performance. Introducing the Domain Classifier module is the key innovation of the paper. The following briefly explains the role of the three modules:

- **Feature Extractor** is used to extract the features of the source domain and target domain data. On the one hand, it maximizes the classification performance of Label Predictor on the source domain, and on the other hand makes the extracted source/target domain features inseparable from the Domain Classifier;
- **Label Predictor** maximizes the classification performance of source domain data based on the features extracted by Feature Extractor;
- The purpose of **Domain Classifier** is to distinguish the source domain/target domain features extracted by Feature Extractor as much as possible.

From the above introduction, it can be seen that Feature Extractor and Domain Classifier are two adversarial parts, so naturally it is a minimax problem. Record the parameters of the above three parts as $\theta_f, \theta_y, \theta_d$, so the following goals are obtained:

$$E(\theta_f, \theta_y, \theta_d) = \sum_{i \in S} L_y^i(\theta_f, \theta_y) - \lambda \sum_{i \in S \cup T} L_d^i(\theta_f, \theta_d)$$

Among them, $i \in S$ indicates that the sample comes from the source domain, L_y^i indicates the classification loss, such as Multi-Class Cross-Entropy, and L_d^i indicates the Domain Classifier Classification loss (the specific method is that the Domain Label of the source domain sample is 0, and the target domain sample is 1, training a two-classifier, using Binary Cross Entropy loss).The optimization process is the iterative process shown below:

$$\theta_f^*, \theta_y^* = arg \min_{\theta_f, \theta_y} E(\theta_f, \theta_y, \theta_d)$$
$$\theta_d^* = arg \min_{\theta_d} E(\theta_f^*, \theta_y^*, \theta_d)$$

In order to avoid iterative optimization, DANN introduces a gradient reversal (RevGrad), which is mainly to add a reverse operation in the process of the gradient return of the Domain Classifier. In this way, the optimization process can get away with iterative update, which enhances the training speed.

4.2 DeepCORAL

The CORAL method has been mentioned above, and Deep CORAL is an extension of it. The problem with CORAL is that it relies on linear transformation and is not end-to-end training. Deep CORAL [6] uses nonlinear transformation and applies it to the deep network to optimize the CORAL loss of the source and target domains to the minimum. In addition to improving performance, nonlinear transformation can also be seamlessly connected with CNN, making it more scalable.

Following the definition of CORAL parameters, we define the following CORAL Loss as the covariance distance between the source and target domain features:

$$l_{CORAL} = \frac{1}{4d^2} ||C_S - C_T||_F^2$$

where $|| \cdot ||_F^2$ represents the Frobenius norm, from which the data covariance of the source and target domains is further obtained as

$$C_S = \frac{1}{n_S - 1}(D_S D_S^T - \frac{1}{n_S}(1^T D_S)(1^T D_S))$$
$$C_T = \frac{1}{n_T - 1}(D_T D_T^T - \frac{1}{n_T}(1^T D_T)(1^T D_T))$$

In the two covariance expressions, the term after the minus sign can be understood as the mean. The shape of the final two covariance matrices is $(d \cdot d)$.

The final loss function is

$$l = l_{CLASS} + \sum_{i=1}^{t} \lambda_i l_{CORAL}$$

Where t is the number of CORAL layers, and the reason why the loss function consists of two parts is because the two parts need to be optimized at the same time:

1. Minimizing the classification loss itself will cause the model to overfit the source domain, and the performance on the target domain is very poor.
2. Optimizing only the CORAL loss will deteriorate the characteristics. The network will map the source and target data. If they are mapped to the same point, the CORAL loss will become 0. Such features cannot build a powerful classifier.

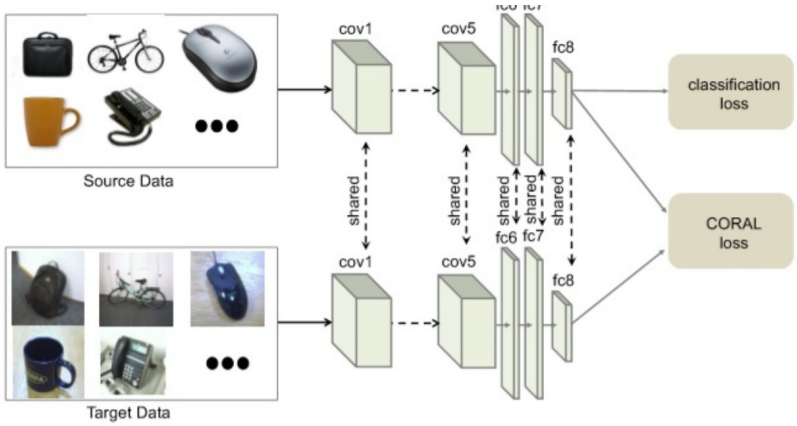

Fig. 5: Sample Deep CORAL architecture based on a CNN with a classifier layer.

5 Experiment

5.1 Dataset

Office-Home dataset is a domain adaptation dataset which consists of 65 categories of office depot from four domains (i.e., A: Art, C:Clipart, P:Product, R: Real-world). In experiments, we used the raw image version of the dataset in deep learning methods. However, we used the 2048-dim

ResNet-50 deep learning features of all images2 in traditional domain adaptation methods. The number of images in each domain is shown as follows in table 1:

Art (A)	**Clipart (C)**	**Product (P)**	**RealWorld (R)**
2426	4346	4438	4356

Table 1: Number of images in each domain.

The following experiments will be conducted in the following three settings (source domain → target domain):

- a) A→R;
- b) C→R;
- c) P→R.

In $X \to Y$ setting, use the deep learning features $X\_X.csv$ as source domain features and $X\_Y.csv$ as target domain features.

5.2 Transfer Component Analysis(TCA)

In this part, we use Python to design our TCA function for experimentation. According to section 3.1, the major parameters are the type of kernel function, and the number of high-dimensional spatial dimensions that the source domain and target domain map together. When the kernel functions are *'primal'*, *'linear'*, and *'rbf'*, we fix the dimension as 30 to carry out the experiment. Considering the amount of calculation, in the experiment to explore the influence of parameters, we randomly select 10% of the data in each domain for calculation, and the experimental results are as follows in table 2 and fig 6:

Kernel Function	A→R	C→R	P→R
linear	**0.463**	**0.495**	**0.578**
primal	0.461	0.493	0.562
rbf	0.442	0.492	0.564

Table 2: Experimental results of TCA under different kernel functions with domain dim=30

In order to eliminate the contingency of the experiment, we repeatedly sampled 10% data from each domain for experiments without replacement. In other words, the above table records the results after average, and the data in each experiment does not overlap. The experimental results show that *'linear'* performs better in the three sets of domain migration, and the training speed also has obvious advantages.

Regarding the dimensions of the public projection space, we conducted repeated experiments similar to the above, and obtained the results shown in the figure 7. Among them, a more interesting phenomenon is that the accuracy rate changes stepwise with the dimensions of the public projection space. We speculate that this is because some features of the experimental data can be recognized

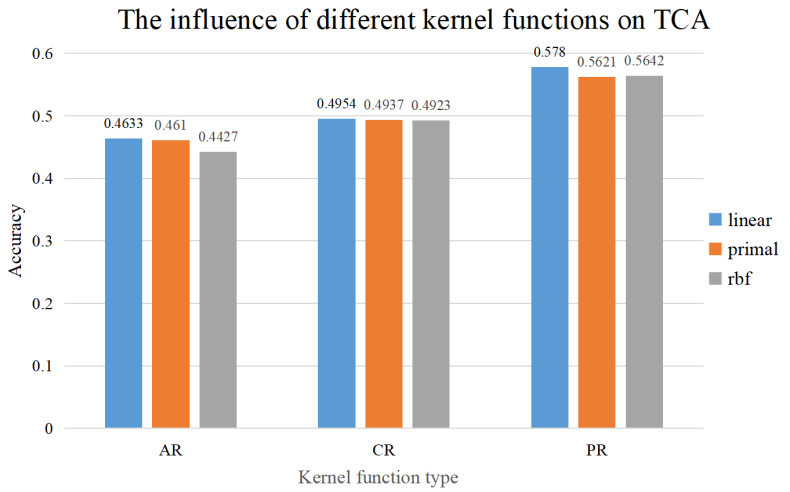

Fig. 6: The influence of different kernel functions on TCA.

by the public space that reaches a certain threshold, and the performance cannot be improved due to the inability to extract new features between the two feature thresholds.

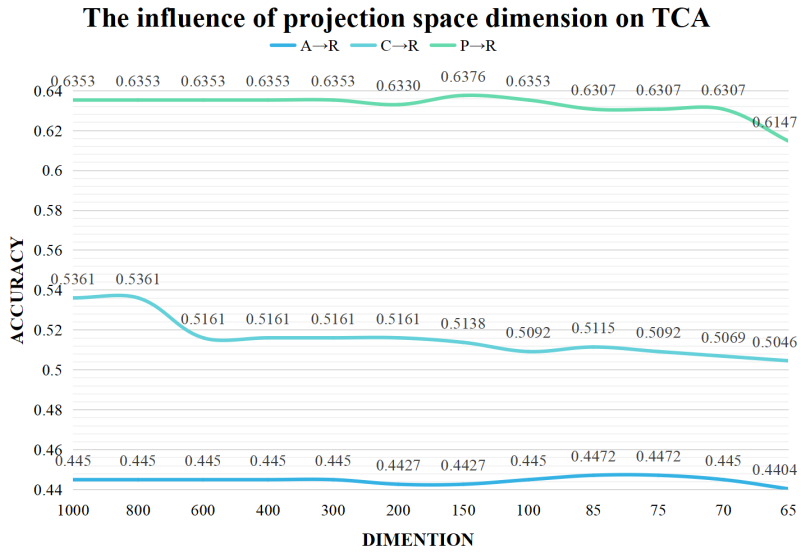

Fig. 7: The influence of projection space dimension on TCA.

In the end, when experimenting with all the data, we found that the projection dimension of 30 is more appropriate. (This is also the reason why we selected 30 for the dimensionality of the nuclear function laboratory)

5.3 Joint Distribution Adaptation(JDA)

Similar to TCA, we implemented our own JDA function using Python. The main parameters of JDA are the kernel function type, the number of high-dimensional space dimensions, and the number of iteration rounds. The experimental method of the kernel function part is similar to TCA, and results are shown in table 3 and fig 8. Finally, the *'primal'* core has the best overall effect.

Domain = 30	A→R	C→R	P→R
linear	0.623	0.569	**0.667**
primal	**0.653**	**0.582**	0.663
rbf	0.618	0.524	0.637

Table 3: Experimental results of JDA under different kernel functions.

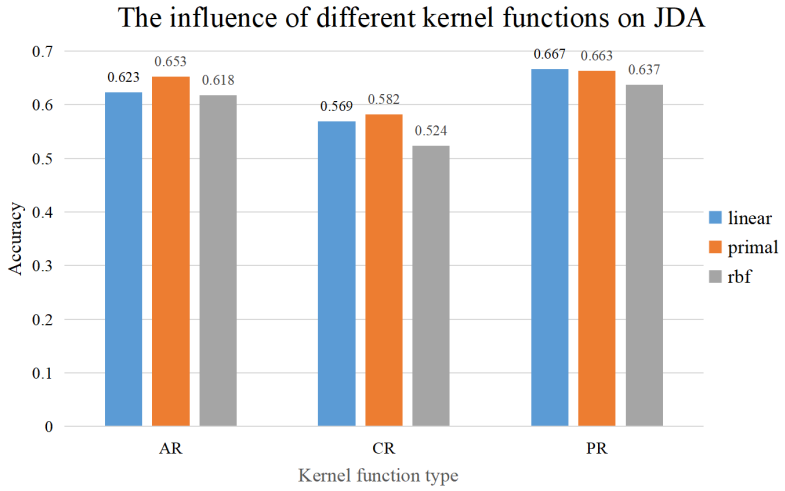

Fig. 8: The influence of different kernel functions on JDA.

For the dimensions of the public projection space, the phenomenon is similar to TCA. In the end, we found that the final experimental performance performed well when the public projection space of JDA was set to 70. In addition, a big change of JDA compared to TCA is that it introduces iterations. We also conducted the experiment shown in figure 11 for the number of iterations.

It is easy to find that in addition to the effect of the regular iteration rounds in machine learning on performance, a very obvious phenomenon is that the second round of the three sets of experiments has the best effect. By repeating the experiment, we also ruled out the possibility of experimental chance. After consulting the information, we were unable to make reasonable guesses about this phenomenon. Due to time constraints, we did not conduct further exploration, but this can be candidated for future work. In addition, after the experiment, it is found that JDA basically converges after 5 iterations (the same phenomenon occurs when all data is selected for the experiment), so we finally choose $Iteration = 5$ for the experiment.

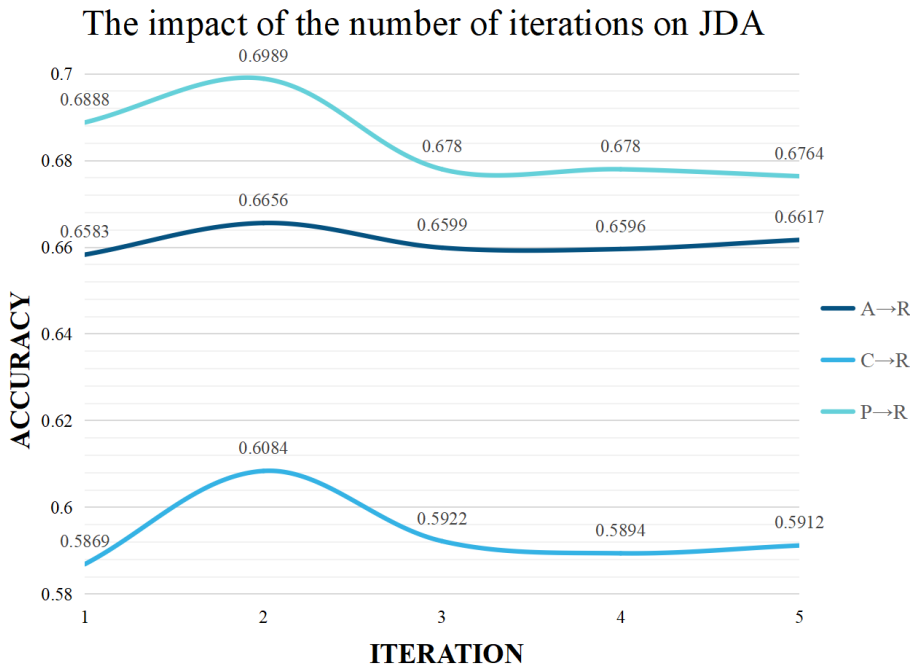

Fig. 9: The impact of the number of iterations on JDA.

5.4 Balanced Distribution Adaptation(BDA)

We have implemented our own BDA/WBDA functions using Python. The main parameters of BDA are the type of kernel function, the number of high-dimensional space dimensions, and the number of iteration rounds. Balance parameters are added to WBDA. The experiment of the kernel function part is similar to the former two, and the experimental results are as follows in table4 and fig10.

Domain = 30	A→R	C→R	P→R
linear	0.625	0.567	0.670
primal	**0.673**	**0.597**	**0.681**
rbf	0.618	0.574	0.647

Table 4: Experimental results of BDA under different kernel functions.

In addition, after comprehensively comparing the experiments that TCA, JDA and BDA are affected by the common projection space dimension, we found that in TCA, the projection space dimension increases and the accuracy rate decreases, but this phenomenon is not found in JDA and BDA. We speculate that this may be due to the introduction of iterations in the latter two, and the learning of features is more comprehensive.

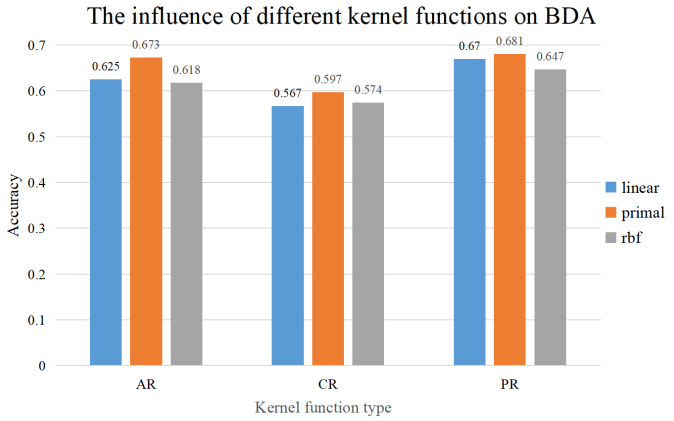

Fig. 10: The influence of different kernel functions on BDA.

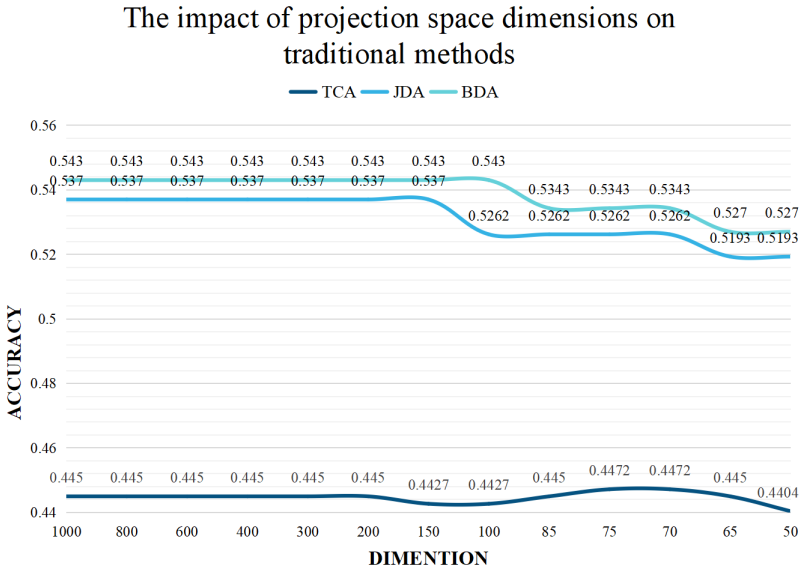

Fig. 11: The impact of projection space dimensions on traditional methods.

5.5 Comparison between traditional methods

In order to test the effect of each traditional method, we will use the experiment of KNN classification directly without domain adaptation as the control group. The experiment result is as follows in table 5.

From the result we can see that all TCA results are worse than the original KNN. It means that this domain adaptation method cause **negative transfer**. Negative transfer means that the knowledge learned in the source domain has a negative effect on the learning in the target domain. This is not a special case in our experiment. We found that other methods also have this phenomenon in certain pair or even pairs of domains.

We speculate that the reason for the negative transfer may be:

Domain	A→R	C→R	P→R
KNN	0.658	0.587	0.695
TCA	0.622	0.569	0.678
JDA	0.661	0.591	0.676
BDA	0.661	0.591	0.676
WBDA	**0.670**	**0.614**	**0.698**
CORAL	0.647	0.583	0.685

Table 5: Experimental results of traditional methods.

1. The transfer learning method is not suitable for current dataset. (the problem of method)
2. There is no/weak relation between source domain and target domain.(the problem of dataset)

Reason 1 is easy to understand. Each method has its limitations to some extent. This is in line with the phenomenon that TCA is far inferior to JDA in the first two pairs of domains, but even slightly better in the last pair of domains.

For reason 2, by consulting the literature, we learned that when the two fields are not similar, we can use several intermediate fields between the two fields to complete the transfer of knowledge. One of the most famous anti-negative transfer learning methods is **Transfer transfer learning**, as shown in figure 13 below. Limited by time and the subject of the experiment, we did not do further exploration.

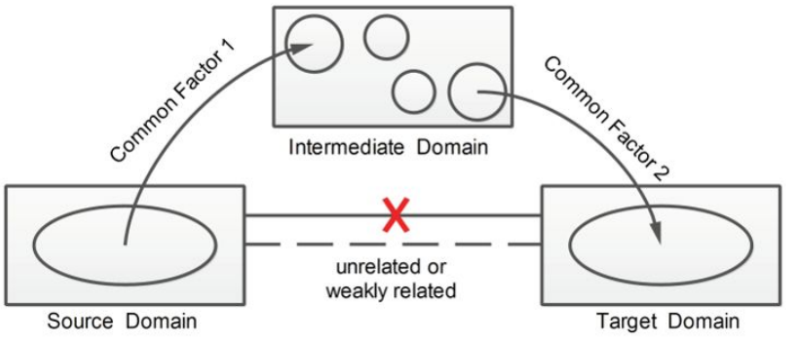

Fig. 12: Transfer transfer learning.

As shown in table 5, the overall performance of JDA is better than TCA. This is because JDA is considered to have added Conditional distribution adaptation on the basis of TCA. On the other hand, JDA is a supervised method, which has certain advantages over unsupervised TCA. And the following picture 13 vividly reflects the role of JDA in domain adaptation

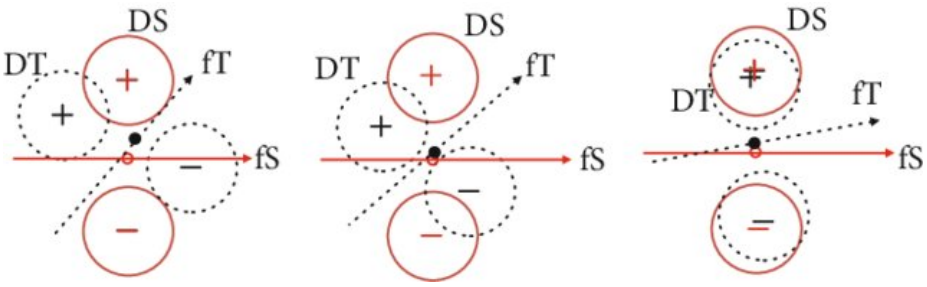

Fig. 13: The function of JDA in domain adaptation.

JDA adapts both the marginal distribution and the conditional distribution, but for different tasks, the marginal distribution and the conditional distribution are not equally important. BDA is better than JDA because it introduces an adaptive mechanism on the basis of JDA: the balance factor. The BDA method can effectively measure (through the balance factor) the weight of these two distributions to achieve the best results. As for WBDA, it goes one step further and adds weight to each category in the target domain. This also explains why WBDA is superior to the first two.

For CORAL,its results are slightly worse than pure KNN, but better than TCA. By consulting the literature, we found that the Fig.14 illustrates why CORAL has a domain adaptation effect but its effect is poor

The biggest advantage of CORAL is that its training speed is very fast, because CORAL only performs simple matrix operations, does not require eigen decomposition like TCA, and does not have an iterative process like JDA, BDA. Considering the correctness rate and time cost comprehensively, it is reasonable to think that CORAL's performance on this task is better than TCA and JDA.

5.6 Deep learning Methods

Our experimental results of the deep learning part is shown in table6 and figure 15:

Domain	A→R	C→R	P→R
DNN	0.665	0.573	0.705
DeepCORAL	0.720	0.627	0.707
DANN	**0.757**	**0.639**	**0.730**

Table 6: Experimental results of deep learning methods.

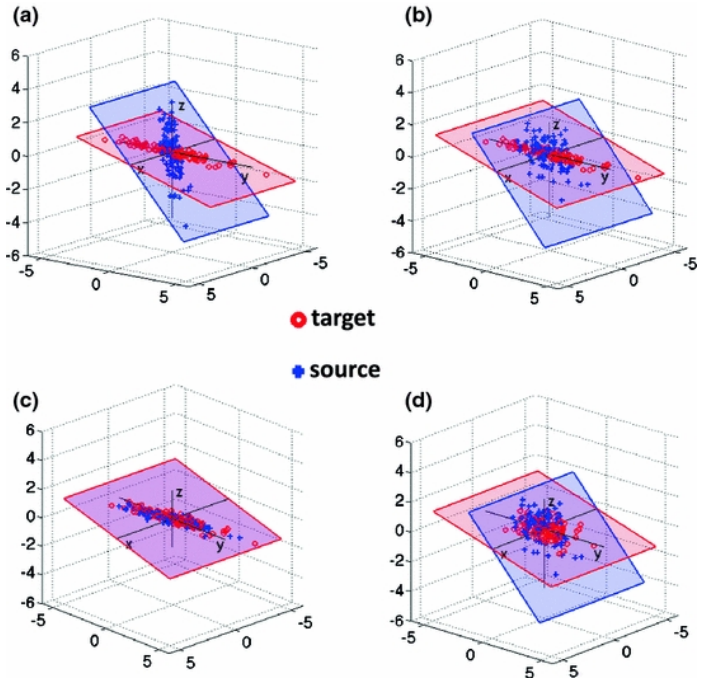

Fig. 14: Illustration of CORrelation ALignment (CORAL) for Domain Adaptation:**a.** The original source and target domains have different distribution covariances. **b.** The same two domains after source decorrelation, i.e., removing the feature correlations of the source domain. **c.** Target re-correlation, adding the correlation of the target domain to the source features. **d.** It is also possible that CORAL might instead attempt to align the distributions by whitening both source and target, which will undoubtedly cause trouble.

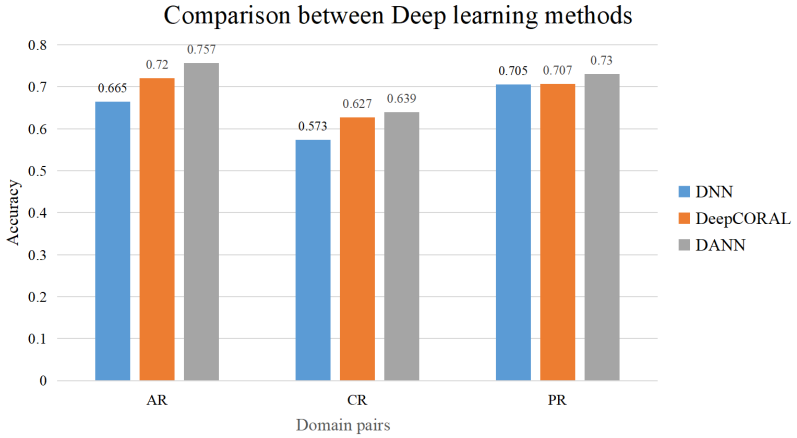

Fig. 15: Comparison between Deep learning methods.

As far as Deep CORAL is concerned, its performance in the horizontal direction is better than that of a simple DNN, and in the vertical direction, it also has an improvement of about 3% compared

to CORAL. The former obviously embodies the role of domain adaptation. The latter is because Deep CORAL can use nonlinear transformations and apply them to deep networks. (The problem with CORAL is that it only relies on linear transformations)

DANN has the best performance among all methods, and it has an improvement of about 3% on the basis of Deep CORAL. After discussion, we speculate that the advantages of DANN are mainly in the following two points:

1. The introduction of the idea of Generative Adversarial Network (GAN) has a higher utilization rate of target domain data.
2. Optimizing only the CORAL loss will deteriorate the characteristics. The network will map the source and target data. If they are mapped to the same point, the CORAL loss will become 0. Such features cannot build a powerful classifier.

The following is the DANN pseudo code we designed in the experiment.

Algorithm 1 DANN with K-fold

Input: The raw dataset
Output: Test set accuracy list for each fold
$acc\_list \leftarrow [\,]$
Set hyper-parameters as hp
Read the data and preprocess
for $train\_indices, test\_indices$ **in** $K - fold\ split$ **do**
    $train\_data \leftarrow dataset[train\_indices]$
    $test\_data \leftarrow dataset[test\_indices]$
    $combined\_model, source\_classification\_model, domain\_classification\_model \leftarrow build\_models(hp)$
    $combined\_model.train(train\_data)$
    $predicts \leftarrow source\_classification\_model.predict(test\_data.X)$
    $acc \leftarrow accuracy\_score(test\_data.Y, predicts)$
    $acc\_list.append(acc)$
return acc_list

Fig. 16: Pseudo code of DANN.

6 Conclusion

In this project, we tried different domain adaptation methods. In terms of traditional methods, we tried TCA, JDA, BDA (WBDA) and CORAL; in terms of deep learning methods, we tried DNN, DeepCORAl and DANN. All methods have designed their own functions and conducted experiments and reasonable horizontal and vertical comparisons. From the experimental results, we also have the following findings:

1. For traditional methods such as TCA, parameters such as kernel function type and projection space dimension will all have an impact on performance, but overall, the improvement of the method will greatly improve the performance than the adjustment of the parameters.

2. After setting up the control group, we found that the traditional method will have worse results than the original KNN. This means that the domain adaptive method will lead to "negative transfer". Negative transfer means that the knowledge learned in the source domain has a negative impact on the learning in the target domain.
3. The overall performance of the traditional method is $WBDA > BDA > JDA > CORAL > TCA$. Detailed analysis is shown in experimental part.
4. DANN is the best performance of all methods. The advantages of DANN mainly lie in the following two points: 1. The introduction of the idea of generative adversarial network (GAN), which has a higher utilization rate of target domain data 2. The loss function is clear In terms of domain label classification loss and loss, the former concentrates on the classification task, while the latter is responsible for measuring the effect of domain adaptation.

7 Contribution

Hou Shengyuan
1.Do experiment on Traditional Methods part.
2.Analyze and discuss the experimental part, and provide materials for report writing.
3.Do vidualization for some of experimental part.

Wu Shiqu
1.Program most of code.
2.Do experiment on Deep learning Methods part.
3.Preprocessing of the dataset and typesetting of paper.

Yan Binghao
1.Write all introduction,abstract and related work part.
2.Write most of experimental part.
3.Most of Visualization work.

References

1. Pan S J, Tsang I W, Kwok J T, et al. Domain adaptation via transfer component analysis[J]. IEEE Transactions on Neural Networks, 2010, 22(2): 199-210.
2. Long M, Wang J, Ding G, et al. Transfer feature learning with joint distribution adaptation[C]//Proceedings of the IEEE international conference on computer vision. 2013: 2200-2207.
3. Wang J, Chen Y, Hao S, et al. Balanced distribution adaptation for transfer learning[C]//2017 IEEE international conference on data mining (ICDM). IEEE, 2017: 1129-1134.
4. Sun B, Feng J, Saenko K. Correlation alignment for unsupervised domain adaptation[M]//Domain Adaptation in Computer Vision Applications. Springer, Cham, 2017: 153-171.
5. Ajakan H, Germain P, Larochelle H, et al. Domain-adversarial neural networks[J]. arXiv preprint arXiv:1412.4446, 2014.
6. Sun B, Saenko K. Deep coral: Correlation alignment for deep domain adaptation[C]//European conference on computer vision. Springer, Cham, 2016: 443-450.