# Project 4 Report: Domain Adaptation

**Mingquan Feng** 517030910373
**Mingjie Li** 517030910344
**Shangning Xu** 517030910384

## Abstract

## 1 Introduction

In this project, we survey traditional and deep learning-based domain adaptation methods, evaluate and analyze their performance on the Office-Home dataset [9]. We select BDA [11], CORAL [7], TCA [5], KMM [3], SA [1], JDA [4] for traditional methods and Deep CORAL [6], DaNN [2] and DDC [8] for deep learning-based methods. The report is organized as follows: Section 2 introduces the domain adaptation methods surveyed in the project, organized into two categories: traditional and deep learning-based; Section 3 introduces the Office-Home dataset and present the performance (with our analysis) of each method on the dataset.

## 2 Methods

### 2.1 Traditional Methods

#### 2.1.1 BDA

Balanced Distribution Adaptation (BDA)[11] solves the transfer learning problem by adaptively minimizing the marginal and conditional distribution discrepancy between domains, and handle the class imbalance problem.

The mathematical formulation of BDA is given as follows. Given a labeled source domain $\{\mathbf{x}_{s_i}, y_{s_i}\}_{i=1}^{n}$, an unlabeled target domain $\{\mathbf{x}_{t_j}\}_{j=1}^{m}$, and assume the feature space $\mathcal{X}_s = \mathcal{X}_t$, label space $\mathcal{Y}_s = \mathcal{Y}_t$ but marginal distributions $P_s(\mathbf{x}_s) \neq P_t(\mathbf{x}_t)$ with conditional distributions $P_s(y_s|\mathbf{x}_s) \neq P_t(y_t|\mathbf{x}_t)$. BDA aims to minimize the discrepancies between: (1) $P_s(\mathbf{x}_s)$ and $P_t(\mathbf{x}_t)$, (2) $P_s(y_s|\mathbf{x}_s)$ and $P_t(y_t|\mathbf{x}_t)$. Specifically, this refers to minimizing the distance

$$D(\mathcal{D}_s, \mathcal{D}_t) \approx (1-\mu) D\Big(P_s(\mathbf{x}_s), P_t(\mathbf{x}_t)\Big) + \mu D\Big(P_s(y_s|\mathbf{x}_s), P_t(y_t|\mathbf{x}_t)\Big) \tag{1}$$

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.

In order to compute the marginal and conditional distribution divergences, BDA adopts maximum mean discrepancy (MMD) to empirically estimate both distribution discrepancies.

$$D(\mathcal{D}_s, \mathcal{D}_t) \approx (1-\mu) \left\| \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_{s_i} - \frac{1}{m} \sum_{j=1}^{n} \mathbf{x}_{t_j} \right\|^2 + \mu \sum_{c=1}^{C} \left\| \frac{1}{n_c} \sum_{\mathbf{x}_{s_i} \in \mathcal{D}_s^{(c)}} \mathbf{x}_{s_i} - \frac{1}{m_c} \sum_{\mathbf{x}_{t_j} \in \mathcal{D}_t^{(c)}} \mathbf{x}_{t_j} \right\|^2 \tag{2}$$

### 2.1.2 CORAL

CORrelation ALignment (CORAL)[7] performs second-order feature alignment on the source and target domain. It transforms the data distribution in the source domain in an unsupervised manner. It is a simple yet effect method for unsupervised domain adaptation.

The mathematical formulation of CORAL is given as follows. Suppose we are given source-domain training samples $D_S = \{\vec{x}_i\}, \vec{x} \in \mathbb{R}^D$ with labels $L_S = \{y_i\}, y \in \{1, \ldots, L\}$, and the target data $D_T = \{\vec{u}_i\}, \vec{u} \in \mathbb{R}^D$. Here both $\vec{x}$ and $\vec{u}$ are $D$-dimensional feature representations $\phi(I)$ of input $I$. 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:

$$\min_A \|C_{\hat{S}} - C_T\|_F^2 = \min_A \|A^\top C_S A - C_T\|_F^2 \tag{3}$$

where $C_{\hat{S}}$ os the covariance of the transformed source features $D_S A$ and $\|\cdot\|_F^2$ denotes the matrix Frobenius norm. The intermediate derivations and proofs are omitted in this report. We only show a compact version of the whole algorithm in Alg. 1.

**Algorithm 1:** CORAL for Unsupervised Domain Adaptation

**Input** Source Data $D_S$, Target Data $D_T$;
**Output** Adjusted Source Data $D_S^\star$;
$C_S = cov(D_S) + eye(size(D_S, 2))$;
$C_T = cov(D_T) + eye(size(D_T, 2))$;
$D_S = D_S * C_S^{-0.5}$;
$D_S^\star = D_S * C_T^{0.5}$

### 2.1.3 Transfer Component Analysis (TCA)

TCA [5] tries to learn some transfer components across domains in a Reproducing Kernel Hilbert Space (RKHS) using Maximum Mean Discrepancy (MMD). In the subspace spanned by these transfer components, data distributions in different domains are close to each other.

$$\begin{array}{ll} \min_W & \operatorname{tr}\left(W^\top W\right) + \mu \operatorname{tr}\left(W^\top KLKW\right) \\ \text{s.t.} & W^\top KHKW = I \end{array}$$

where the first term $\operatorname{tr}\left(W^\top W\right)$ is a regularization term to control the complexity of $W$, $\mu$ is given coefficient and:

$$\begin{aligned} K &= \begin{bmatrix} K_{S,S} & K_{S,T} \\ K_{T,S} & K_{T,T} \end{bmatrix} \\ L_{i,j} &= \begin{cases} \frac{1}{n_s^2}, & \text{if } i,j \in \mathcal{S} \\ \frac{1}{n_t^2}, & \text{if } i,j \in \mathcal{T} \\ -\frac{1}{n_s n_t}, & \text{otherwise} \end{cases} \\ H &= I_{n_1+n_2} - \frac{1}{n_1+n_2}\mathbf{1}\mathbf{1}^\top \end{aligned}$$

The second term $\operatorname{tr}\left(W^{\top} K L K W\right)$ is distance between the empirical means of the two domains:

$$\begin{aligned}
\operatorname{tr}\left(W^{\top} K L K W\right) &= \operatorname{tr}\left(\left(K W W^{\top} K\right) L\right) \\
&= \operatorname{tr}\left(\left(K W W^{\top} K\right) L\right) \\
&= \operatorname{tr}\left(\left(K K^{-1/2} \widetilde{W}\right)\left(\widetilde{W}^{\top} K^{-1/2} K\right) L\right) \\
&= \left\| \frac{1}{n_1} \sum_{i=1}^{n_1} \widetilde{\phi}\left(x_{S_i}\right) - \frac{1}{n_2} \sum_{i=1}^{n_2} \widetilde{\phi}\left(x_{T_i}\right) \right\|_{\mathcal{H}}^2
\end{aligned}$$

Similar to kernel Fisher discriminant (KFD), the solution of W is the eigenvectors corresponding to the m leading eigenvalues of $(I + \mu K L K)^{-1} K H K$, where at most $n_1 + n_2 - 1$ eigenvectors can be extracted.

### 2.1.4 Kernel Mean Matching (KMM)

Unlike previous work, KMM [3] infers the resampling weight directly by distribution matching between training and testing sets in feature space in a non-parametric way. account for the difference between $\Pr(x, y)$ and $\Pr'(x, y)$ by reweighting the training points such that the means of the training and test points in a reproducing kernel Hilbert space (RKHS) are close.

Specifically, reweighting coefficients $\beta$ can be obtained by solving following QP optimization problem:

$$\begin{aligned}
&\underset{\beta}{\text{minimize}} \frac{1}{2} \beta^{\top} K \beta - \kappa^{\top} \beta \\
&\text{subject to } \beta_i \in [0, B] \text{ and } \left| \sum_{i=1}^{m} \beta_i - m \right| \leq m \epsilon
\end{aligned}$$

where $\epsilon$ is given parameter and

$$\begin{aligned}
K_{ij} &= k\left(x_i, x_j\right) \\
\kappa_i &= \frac{m}{m'} \sum_{j=1}^{m'} k\left(x_i, x'_j\right)
\end{aligned}$$

Once we estimate the reweighting coefficients, we can directly use them in learning algorithms such as SVM:

$$\begin{aligned}
&\text{minimize}_{\theta, \xi} \frac{1}{2} \|\theta\|^2 + C \sum_{i=1}^{m} \beta_i \xi_i \\
&\text{subject to } \left\langle \phi\left(x_i, y_i\right) - \phi\left(x_i, y\right), \theta \right\rangle \geq 1 - \xi_i / \Delta\left(y_i, y\right) \text{ for all } y \in y, \text{ and } \xi_i \geq 0
\end{aligned}$$

### 2.1.5 SA

SA (subspace alignment) [1] tries to "align" the subspaces of the source and target domains. The subspaces are formed by dimensionality reduction with PCA and denoted by the projection matrices $X_S$ (for source domain) and $X_T$. SA wishes to find a matrix $M^*$ that minimzes the difference between $X_S$ and $X_T$, represented by the Bregman matrix divergence:

$$M^* = \underset{M}{\arg\min} \left|\left| X_S M - X_T \right|\right|_F^2$$

which has a trivial solution $M = X_S^T X_T$. Given $M^*$, the source domain data are projected into the *target aligned source coordinate system*, denoted by the projection matrix $X_a = X_S M^*$, which has been "aligned" with the target domain subspace. The target domain data is projectde into the target domain subspace with $X_T$, after which an SVM can be directly learned. The algorithm is given in Algorithm 8.

A similarity $A$ matrix can also be constructed to facilitate the use of KNN:

$$Sim(y_S, y_T) = (y S X_S M^*)(y_T X_T)^T = y_S X_S M^* X_T^T y_T^T = y_S A y_T^T$$

**Algorithm 2:** SA algorithm

**Input:** $S$ source domain data, $T$ target domain data, $L_S$ source domain label, $d$ subspace dimension
**Output:** predicted target labels $L_T$
$X_S \leftarrow \text{PCA}(S, d)$
$X_T \leftarrow \text{PCA}(T, d)$
$X_a \leftarrow X_S X_S^T X_T$
$S_a \leftarrow S X_a$
$T_T \leftarrow T X_T$
$L_T \leftarrow \text{Classifier}(S_a, T_T, L_S)$

#### 2.1.6 JDA

JDA (Joint Distribution Adaptation) [4] can be viewed as a generalized version of TCA. Let the set of all labels be $\{1, 2, \ldots, C\}$ and $\mathcal{D}_s^{(c)}$ 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 $\mathcal{D}_t^{(c)}$ for target domain. The optimization target is the MMD distance between the class-conditional distributions $Q_s(x_s|y_s = c)$ and $Q_t(x_t|y_t = c)$.

$$\left\| \frac{1}{|\mathcal{D}_s^{(c)}|} \sum_{x \in \mathcal{D}_s^{(c)}} A^T x - \frac{1}{|\mathcal{D}_t^{(c)}|} \sum_{x \in \mathcal{D}_t^{(c)}} A^T x \right\|^2 = \text{tr}(A^T X M_c X^T A)$$

where $A$ is the projection matrix in PCA. Let $n_s^{(c)}$ be $|\mathcal{D}_s^{(c)}|$ and $n_t^{(c)}$ be $|\mathcal{D}_t^{(c)}|$, $M_c$ is given by

$$(M_c)_{ij} = \begin{cases} \frac{1}{n_s^{(c)} n_s^{(c)}}, & x_i, x_j \in \mathcal{D}_s^{(c)} \\ \frac{1}{n_t^{(c)} n_t^{(c)}}, & x_i, x_j \in \mathcal{D}_t^{(c)} \\ \frac{-1}{n_s^{(c)} n_t^{(c)}}, & x_i \in \mathcal{D}_s^{(c)}, x_j \in \mathcal{D}_t^{(c)} \text{ or vice versa} \\ 0, & \text{otherwise} \end{cases}$$

The final optimization problem is

$$\begin{aligned} \text{minimize} \quad & \sum_{c=0}^{C} \text{tr}(A^T X M_c X^T A) + \lambda \|A\|_F^2 \\ \text{subject to} \quad & A^T X H X^T A = I \end{aligned}$$

Borrowing notation from Section 2.1.3, $M_0 = L$ in TCA. It is obvious that TCA is a special case of JDA when $C = 0$.

### 2.2 Deep Learning Methods

#### 2.2.1 Deep CORAL

Deep CORAL[6] is an extension of CORAL which aims to learn a nonlinear transformation that aligns correlations of layer activations in deep neural networks. The mathematical formulation of Deep CORAL is given as follows.

Suppose we are given source-domain training examples $D_S = \{\mathbf{x}_i\}, \mathbf{x} \in \mathbb{R}^d$ with labels $L_S = \{y_i\}, y \in \{1, \ldots, L\}$, and unlabeled target data $D_T = \{\mathbf{u}_i\}, \mathbf{u} \in \mathbb{R}^d$. Suppose the number of source and target data are $n_S$ and $n_T$ respectively. Here both $\mathbf{x}$ and $\mathbf{u}$ are the $d$-dimensional deep layer activations $\phi(I)$ of input $I$ that we are trying to learn. Suppose $C_S$ and $C_T$ are the feature vector covariance matrices. The author defined the CORALloss as the distance between the second-order statistics (covariances) of the source and target features:

$$\mathcal{L}_{CORAL} = \frac{1}{4d^2} \left\| C_S - C_T \right\|_F^2 \tag{4}$$

where $\|\cdot\|_F^2$ denotes the matrix Frobenius norm. The covariance matrices of the source and target data are given by

$$C_S = \frac{1}{n_S - 1}\left(D_S^\top D_S - \frac{1}{n_S}(\mathbf{1}^\top D_S)^\top(\mathbf{1}^\top D_S)\right) \tag{5}$$

$$C_T = \frac{1}{n_T - 1}\left(D_T^\top D_T - \frac{1}{n_T}(\mathbf{1}^\top D_T)^\top(\mathbf{1}^\top D_T)\right) \tag{6}$$

To enable end-to-end learning, the overall loss can be divided into two parts. One is the classification loss, and the other is the CORAL loss.

$$\mathcal{L} = \mathcal{L}_{CLASS} + \lambda \cdot \mathcal{L}_{CORAL} \tag{7}$$

Figure 1 shows a sample Deep CORAL architecture based on a CNN with a classifier layer.

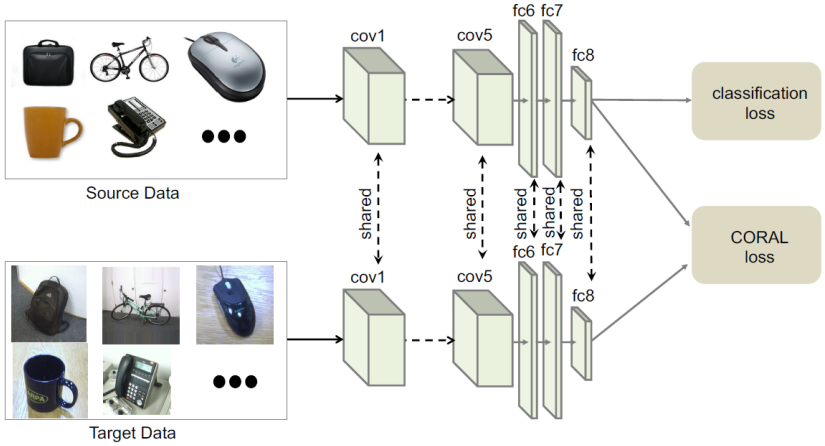



Figure 1: Sample Deep CORAL architecture based on a CNN with a classifier layer.

### 2.2.2 Domain Adaptive Neural Networks (DaNN)

DaNN [2] is a variant of the standard feed forward neural network. This model incorporates the Maximum Mean Discrepancy (MMD) measure as a regularization in the supervised learning to reduce the distribution mismatch between the source and target domains in the latent space. By using such a regularization, they aim to train the network parameters such that the supervised criterion is optimized and the hidden layer representations are encouraged to be invariant across different domains.

Given the labeled source data $\left\{\mathbf{x}_s^{(i)}, \mathbf{y}_s^{(i)}\right\}_{i=1,\ldots,n_s}$, and the unlabeled target data $\left\{\mathbf{x}_t^{(j)}\right\}_{j=1,\ldots,n_t}$, the loss function of a single layer DaNN is given by:

$$\begin{aligned}
J_{\text{DaNN}} &= J_{\text{NNs}} + \gamma \mathcal{MMD}_e^2\left(\mathbf{q}_s, \overline{\mathbf{q}}_t\right) \\
J_{\text{NNs}} &= -\frac{1}{n_s}\sum_{i=1}^{n_s}\sum_{k=1}^{l}\left(\left[\mathbf{y}_s^{(i)}\right]_k \log\left(\left[f\left(\mathbf{x}_s^{(i)}\right)\right]_k\right)\right) \\
\mathbf{q}_s &= \mathbf{W}_1^\top \mathbf{x}_s + \mathbf{b} \\
\overline{\mathbf{q}}_t &= \mathbf{W}_1^\top \mathbf{x}_t + \mathbf{b} \\
\mathcal{MMD}_{\text{e}}\left(\mathbf{x}_s, \mathbf{x}_t\right) &= \left\|\frac{1}{n_s}\sum_{i=1}^{n_s}\phi\left(\mathbf{x}_s^{(i)}\right) - \frac{1}{n_t}\sum_{j=1}^{n_t}\phi\left(\mathbf{x}_t^{(j)}\right)\right\|_{\mathcal{H}}
\end{aligned}$$

Notice that MMD is a non-parametric probability distribution distance measure, and this work is the first study to use MMD in neural networks. The idea of DaNN is very similar with TCA, but with neural network instead of traditional optimization methods.

### 2.2.3 DDC

DDC (Deep Domain Confusion) [8] works within the framework of Deep Coral (Section 2.2.1) and, instead of coral loss, MMD loss is used. "A lower dimensional, 'bottleneck,' adaptation layer" is added before the final FC layer, because "a lower dimensional layer can be used to regularize the training of the source classifier and prevent overfitting to the particular nuances of the source distribution." The MMD loss is placed on top of the "bottleneck" layer "to directly regularize the representation to be invariant to the source and target domains." The DNN architecture is shown in Figure 2.

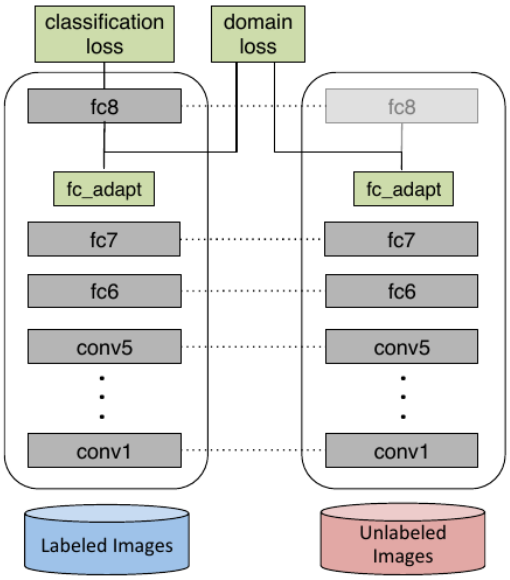



Figure 2: DDC DNN architecture

The final loss is

$$\mathcal{L} = \mathcal{L}_C(X_L, y) + \lambda \text{MMD}^2(X_S, X_T)$$

where the MMD loss is given by

$$\text{MMD}(X_S, X_T) = \left\| \frac{1}{|X_S|} \sum_{x_s \in X_S} \phi(x_s) - \frac{1}{|X_T|} \sum_{x_t \in X_T} \phi(x_t) \right\|$$

Only the labeled examples are used to compute the classification loss, while all data is used from both domains to compute the domain confusion loss. The network is jointly trained on all available source and target data.

The objective outlined in is easily represented by this convolutional neural network where MMD is computed over minibatches of source and target data. We simply use a fork at the top of the network, after the adaptation layer. One branch uses the labeled data and trains a classifier, and the other branch uses all the data and computes MMD between source and target.

After fine-tuning this architecture, owing to the two terms in the joint loss, the adaptation layer learns a representation that can effectively discriminate between the classes in question due to the classification loss term, while still remaining invariant to domain shift due the MMD term. Such a representation will thus enable increased adaptation performance.

## 3 Experiments

In this section, we fill first introduce the dataset we used, and the baseline we adopted. We will explain the experimental procedure, results and analysis of different domain adaptations in this section. Some of our codes are adapted from the Github repository transferlearning.

### 3.1 Dataset

The Office-Home dataset[10] is created to evaluate domain adaptation algorithms for object recognition using deep learning. It consists of images from 4 different domains: Artistic images, Clip Art, Product images and Real-World images. For each domain, the dataset contains images of 65 object categories found typically in Office and Home settings.

In experiments, we used the raw image version[1] of the dataset in deep learning methods. However, we used the 2048-dim ResNet-50 deep learning features of all images[2] in traditional domain adaptation methods. The number of images in each domain is shown in Table 1.

Table 1: Number of images in each domain.

| **domain** | Art (A) | Clipart (C) | Product (P) | RealWorld (R) |
|---|---|---|---|---|
| **#images** | 2426 | 4346 | 4438 | 4356 |

As is mentioned in the requirements of this project, we only conducted experiments in the following three settings – *i.e.* (1) Art→RealWorld, (2) Clipart→RealWorld, and (3) Product→RealWorld. For simplicity, we abbreviated them as A→R, C→R, and P→R.

### 3.2 Baseline

Before using domain adaptation methods, we first applied SVM with rbf kernel directly, and used the results as a baseline. More specifically, we used deep features in domain X (X∈{A,C,P}) as the training data, and tested the well-trained SVM on domain R. The accuracy of the baseline method SVM is shown in 2. The numbers in the table corresponds to the accuracy on the RealWorld domain. We tried different $C$ and selected the best results as the baseline (bold numbers in the table). We used the deep features for training and testing without any preprocessing.

Table 2: The accuracy of baseline method SVM on the target domain. We tried different $C$ and selected the best results as the baseline. We used the deep features without any preprocessing.

| C | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|---|---|---|---|---|---|---|---|---|---|---|
| A→R | 27.2% | 46.2% | 57.8% | 64.1% | 67.5% | 69.9% | 71.7% | 73.1% | 73.9% | **74.4%** |
| C→R | 41.7% | 56.0% | 59.8% | 61.4% | 62.6% | 63.3% | 63.9% | 64.3% | 64.6% | **64.9%** |
| P→R | 61.2% | 70.3% | 71.7% | 72.3% | 72.8% | 72.8% | 72.9% | 73.0% | 73.0% | **73.0%** |

### 3.3 CORAL

In this part, we reported the experimental results of CORAL for domain adaptation. One advantage is that CORAL needs no hyper-parameters. Since CORAL only transforms the source-domain data using a linear transformation, the transformed data has the same dimension as the raw data. Table 3 shows the classification accuracy on the target domain using the transformed data using CORAL. Since there is not any hyper-parameter for tuning, we only tested on different $C$ in SVM. Note that the SVM used the rbf kernel, which kept the same as the setting of the baseline.

As is shown in Table 3, the improvement is slight. In other words, from the classification accuracy, we cannot find major improvements. We then analyzed the covariance of the transformed source data and the target data. Table 4 reports the sum of squared error of the covariance matrix on the source

[1] http://hemanthdv.org/OfficeHome-Dataset/

[2] https://pan.baidu.com/s/1qvcWJCXVG8JkZnoM4BVoGg

Table 3: Classification accuracy on the target domain using the transformed data using CORAL.

| | C | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| A→R | test acc | 26.6% | **46.6%** | **58.3%** | **64.6%** | 67.2% | 69.5% | 71.3% | 72.7% | 73.6% | 73.9% |
| | baseline | 27.2% | 46.2% | 57.8% | 64.1% | 67.5% | 69.9% | 71.7% | 73.1% | 73.9% | 74.4% |
| | improvement | -0.6% | **0.4%** | **0.5%** | **0.5%** | -0.3% | -0.4% | -0.4% | -0.4% | -0.3% | -0.5% |
| C→R | test acc | **42.7%** | **57.2%** | **60.7%** | **62.6%** | **63.8%** | **64.1%** | **64.2%** | **64.6%** | **65.0%** | **65.1%** |
| | baseline | 41.7% | 56.0% | 59.8% | 61.4% | 62.6% | 63.3% | 63.9% | 64.3% | 64.6% | 64.9% |
| | improvement | **1.0%** | **1.2%** | **0.9%** | **1.2%** | **1.2%** | **0.8%** | **0.3%** | **0.3%** | **0.4%** | **0.2%** |
| P→R | test acc | 60.1% | **70.5%** | **72.0%** | **72.5%** | 72.8% | 72.7% | 72.8% | 73.0% | 73.0% | **73.1%** |
| | baseline | 61.2% | 70.3% | 71.7% | 72.3% | 72.8% | 72.8% | 73.1% | 73.0% | 73.0% | 73.0% |
| | improvement | -1.1% | **0.2%** | **0.3%** | **0.2%** | 0.0% | -0.1% | -0.3% | 0.0% | 0.0% | **0.1%** |

domain and that on the target domain. We reported $\|C_S - C_T\|_F^2$ for comparison. We could find that the covariance matrices becomes similar after the CORAL transformation.

Table 4: Sum of square error $\|C_S - C_T\|_F^2$

| | A→R | C→R | P→R |
|---|---|---|---|
| raw | 774.06 | 980.41 | 880.95 |
| CORAL | 49.33 | 87.59 | 67.77 |

To further demonstrate the similarity in an instinctive way, we visualized the covariance matrix, which is a 2048 by 2048 square matrix. Figure 3 shows that the covariance matrix of the transformed data becomes similar to the covariance matrix of the target-domain data. This illustrates the effectiveness of this algorithm.

### 3.4 BDA

In this part, we reported the experimental results of BDA for domain adaptation. Since the transformation in BDA is put on both the source and the target domain, we may choose different transformed dimensions for comparison. In our experiment, we choose from {32, 64, 128, 256, 512, 1024, 2048} for comparison with the baseline. We set the hyper-parameter $\mu$ in this method to be 0.5.

Table 5 shows the result. We found that for each task, there is at least one setting of transformed dimensionality that surpassed the baseline. This shows that BDA performs quite well on this dataset.

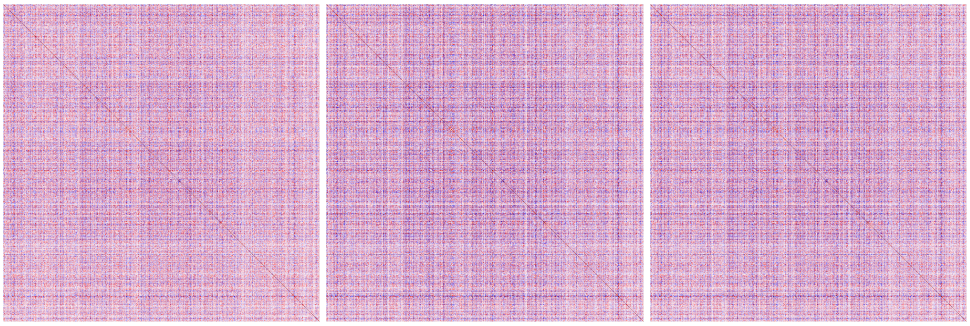
Table 5: Classification accuracy on the target domain using the transformed data using BDA.

| dimension | 32 | 64 | 128 | 256 | 512 | 1024 | 2048 | baseline |
|---|---|---|---|---|---|---|---|---|
| A→R | 70.3% | 74.1% | **74.8%** | 74.3% | 74.2% | 74.2% | 74.2% | 74.4% |
| C→R | 63.6% | **65.3%** | **65.7%** | **65.7%** | **65.5%** | **65.5%** | **65.4%** | 64.9% |
| P→R | 71.70% | **73.80%** | **73.50%** | **74.10%** | **74.20%** | **74.20%** | **74.20%** | 73.0% |

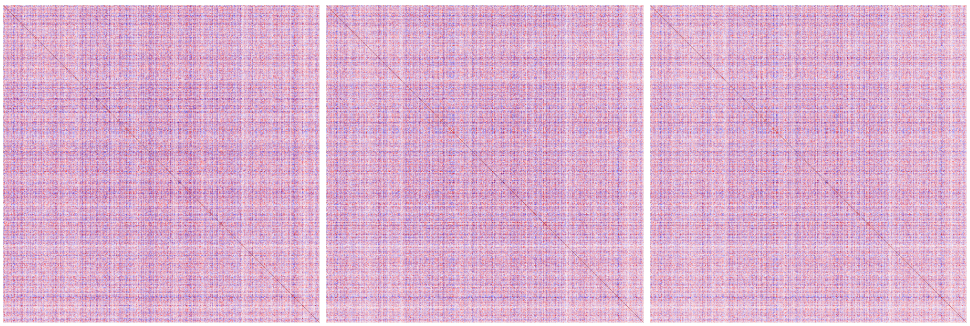
### 3.5 Deep CORAL

In this part, we reported the experimental results of Deep CORAL for domain adaptation. In experiments, we found that the numerical order of $\mathcal{L}_{CLASS}$ and $\mathcal{L}_{CORAL}$ is significantly different, with $\mathcal{L}_{CORAL} << \mathcal{L}_{CLASS}$. Therefore, the coefficient $\lambda$ before the coral loss should be large to ensure the decreasing of $\mathcal{L}_{CORAL}$. We also found that when we trained the network from scratch, the performance was bad. Therefore, in experiments, we set all the parameters in the backbone as that pretrained on ImageNet. We finetuned the network for 100 epoch after that.

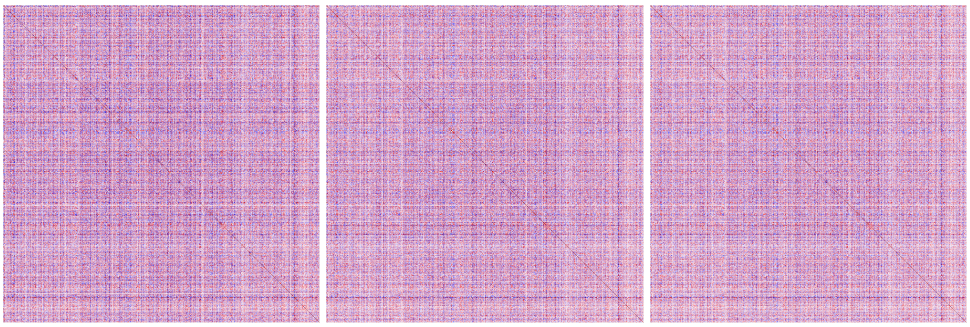
As is mentioned before, the setting of $\lambda$ is tricky. We observed that in the later period of training, the numerical order of $\mathcal{L}_{CORAL}$ is very, very small (less than $10^{-6}$). So we designed an exponentially increasing setting of $\lambda$. This means $\lambda^{(e)}$ is a exponentially increasing sequence, where $e$ denotes the number of epoch. Table 6 shows the experimental results with different $\lambda$ settings. As we can see, the exponentially increasing $\lambda$ will lead to slightly better classification results. The experiments in Table 6 all used the ResNet-50 as the backbone network.



(a) A→R source (b) A→R target (c) A→R transformed



(d) C→R source (e) C→R target (f) C→R transformed



(g) P→R source (h) P→R target (i) P→R transformed

Figure 3: Visualization of covariance matrices of the source-domain data, target-domain data, and the transformed data.

Table 6: Classification accuracy on the target domain using Deep CORAL with ResNet-50 as the backbone. We tuned $\lambda$ in an exponentially increasing manner.

| $\lambda$ | $10^3$ to $10^3$ | $10^3$ to $10^4$ | $10^3$ to $10^5$ | baseline |
|---|---|---|---|---|
| A→R | 72.29% | 71.26% | 72.04% | **74.4%** |
| C→R | 62.29% | 62.47% | 62.47% | **64.9%** |
| P→R | 71.01% | 71.52% | 71.56% | **73.0%** |

However, we found that the results is even worse than the baseline. We thought the reason is that the CORAL loss also affects the training of classification loss, because we found the training accuracy is also lower than that in the baseline experiment.

We then tried different backbones, including Alexnet, ResNet-18/34/50/101 for comparison. We found that networks with better expressiveness led to better classification results.

Table 7: Classification accuracy on the target domain using Deep CORAL with different backbones. We set $\lambda$ in an exponentially increasing manner from $10^3$ to $10^4$.

| backbone | AlexNet | ResNet-18 | ResNet-34 | ResNet-50 | ResNet-101 | baseline |
|---|---|---|---|---|---|---|
| A→R | 43.92% | 66.90% | 70.66% | **71.26%** | 74.13% | **74.4%** |
| C→R | 36.56% | 56.09% | 59.67% | **62.47%** | 64.14% | **64.9%** |
| P→R | 44.22% | 65.29% | 69.54% | **71.52%** | **73.62%** | 73.0% |

### 3.6 JDA

We use the primal kernel and set the dimensions after PCA to 512. $\lambda$ in the JDA equation is set to 1. Based on our observation, JDA converges quickly, and sometimes the accuracy on the validation set plateaus after the first iteration. Therefore, the number of iterations is set to 4, and the accuracies of JDA are reported in Table 8. With feature vectors of only 30 dimensions after domain adaptation, JDA achieves comparable performance compared with the baseline.

Table 8: Accuracy of JDA compared to baseline

| Dataset | A → R | C → R | P → R |
|---|---|---|---|
| JDA/% | 74.48 | 65.84 | 74.67 |
| Baseline/% | 74.4 | 64.9 | 73.0 |
| Improvement/% | **0.11** | **1.44** | **2.29** |

JDA is also noted for its insensitivity to the setting of $\lambda$, which we replicate in our experiment. With exponentially-increasing $\lambda$, the accuracy of JDA is stable, as shown in Figure 4.

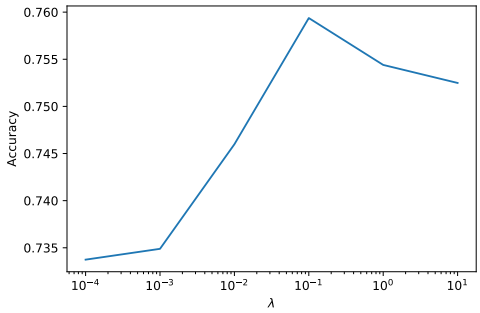



Figure 4: Accuracy with respect to JDA's hyperparameters. The left plot shows the accuracy w.r.t $\lambda$.

Table 9 shows the accuracy of JDA with respect to the number of dimensions after PCA to give a comprehensive overview of JDA's performance.

### 3.7 SA

The implementation from knnedyCzar[3] is used as the reference implementation. This implementation has two hyperparameters: the number of dimensions after PCA and the kernel used for kernel PCA. We use the linear kernel and 128 dimensions and report accuracies on the domain adaptation tasks in Table 10. Overall, positive but minimal improvements can be observed.

[3] https://github.com/kennedyCzar/TRANSFER-LEARNING-AND-OPTIMAL-TRANSPORT

Table 9: Accuracies of JDA w.r.t PCA dimensions

| PCA Dimensions | 2 | 8 | 16 | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|---|---|---|
| Accuracy/% | 7.42 | 49.12 | 63.89 | 76.13 | **76.67** | 76.05 | 75.94 | 75.86 |

Table 10: Accuracy of SA compared to baseline

| Dataset | A $\rightarrow$ R | C $\rightarrow$ R | P $\rightarrow$ R |
|---|---|---|---|
| SA/% | 75.21 | 66.22 | 73.83 |
| Baseline/% | 74.4 | 64.9 | 73.0 |
| Improvement/% | **1.09** | **0.32** | **1.14** |

To characterize the influence of PCA dimensions, Table 11 shows the accuracy with respect to PCA dimensions. The linear kernel is still used for PCA. The accuracy peaks at 256, signaling an ideal trade-off between useful information and noise.

Table 11: Accuracy of SA with respect to different PCA dimensions

| Dimensions | 2 | 8 | 16 | 64 | 128 | 256 | 512 | 1024 | 2048 |
|---|---|---|---|---|---|---|---|---|---|
| Accuracy/% | 2.87 | 46.82 | 58.80 | 74.06 | 75.21 | **75.48** | 74.45 | 74.25 | 74.56 |

### 3.8 DDC

The backbone network used for DDC is pretrained ResNet50 from PyTorch's model zoo. To fine-tune the model on our data set with MMD loss, the initial learning rate is set to $10^{-2}$ for the bottleneck and classification layers and $10^{-3}$ for the rest. The batch size is 32. $\lambda$ is set to 10. Table 12 shows the performance of DDC compared with the baseline. In our case, DDC shows improvement over the baseline SVM with ResNet features only on the Product $\rightarrow$ Real World task.

In this case, SVM with ResNet features may not be an ideal because it can't be compared with the end-to-end approach adopted by DDC. We turn off the MMD loss so that gradients are solely from the classification loss, which forms a new baseline. On the Product $\rightarrow$ Real World task, the new baseline achieves 72.69% accuracy, which implies that the MMD loss may have small influence on training.

The role of the MMD loss may be related to its loss curves. Figure 5 shows the loss curves when the MMD loss is turned off/on, respectively. It is suprising that in both cases the loss curves look identical. When the MMD loss is turned off i.e. no gradients from MMD loss, but the loss itself is still computed, the MMD loss is stable. This stability may be explained by the canceling of two factors: The MMD loss may increase due to fitting domain-specific features and may decrease due to shared features.

When MMD loss is turned on, with $\lambda = 10$, the MMD loss dominates the total loss in later stages. Even in this case, the MMD loss can't be reduced further, which may imply that the role of MMD loss is to act as a regularizer, which prevents the distributions of the source and target domain to grow too far apart during training, but can't push them together.

### 3.9 TCA

Our implementation of TCA is referred from website[4]. We choose to tune on kernels and dimensions, and the result is in Table 13 and Figure 6. Form the result, one can observe that primal kernel is better than other two kernels. We also compare the result with baseline accuracy in Table 14, and notice that Linear kernel and RBF kernel is worse than baseline result, while primal kernel is slightly better than baseline.

Data visualization result is displayed in Figure 7.We randomly select $10\%$ data points from A_R setting to perform TCA with default setting (dim=30, kernel=Primal). Then we first perform PCA to reduce data dimension to 32, and perform TSNE to reduce data dimension to 2. From the result we

[4]https://github.com/jindongwang/transferlearning/blob/master/code/traditional/TCA/TCA.py

Table 12: Accuracy of DDC compared to baseline

| Dataset | A → R | C → R | P → R |
|---|---|---|---|
| DDC/% | 74.04 | 64.13 | 73.31 |
| Baseline/% | 74.4 | 64.9 | 73.0 |
| Improvement/% | -0.48 | -1.19 | 0.42 |

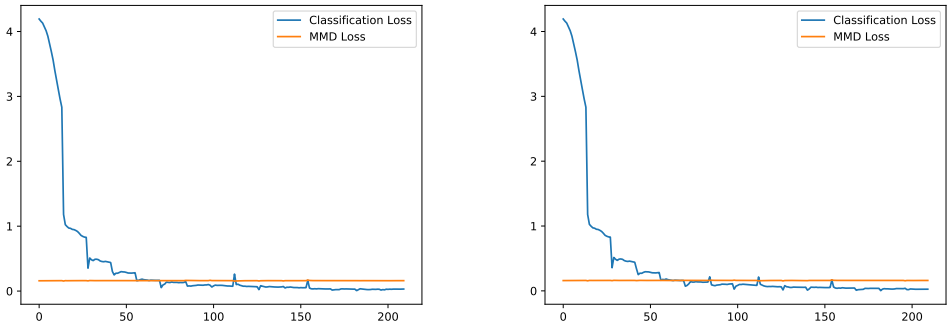



Figure 5: Loss curves during training for DDC

can observe that before TCA, the source and target (blue and green points) distributed differently, but after TCA the source and target (orange and red points) distributed slightly more similar.

### 3.10 KMM

Our implementation of KMM is referred from website[5]. The experiment result on the 3 datasets is in Table15, which illustrates that linear kernel has similar performance with RBF kernel, and both of kernels are outperformed by baseline.

Data visualization is displayed in Figure 8. We randomly select $10\%$ data points from A_R setting to perform KMM with linear kernel. Then we first perform PCA to reduce data dimension to 32, and perform TSNE to reduce data dimension to 2. We draw the histogram of learned beta in Figure 8(a), and display data distribution in Figure 8(b), where source data is scaled by weight beta. From Figure 8(a) we can observe that KMM does not effectively learn data distributions, instead, it simply detect several outliers, and most of data points have same beta. In Figure 8(b), we can intuitively see that source points are not effectively weighted, since the weights are visualized by scales, but most of points have same scales. And therefore, the accuracy of KMM is slightly worse than baseline.

### 3.11 DaNN

Our implementation of DaNN is referred from website[6]. The detailed network architecture is displayed in Table 16, and some important parameters are listed in 17. Notice that this is a very simple neural network, since it only has 2 linear layers for source data, and 1 linear layer for target data. It is possible to add more layers if input dataset is larger and more complex, but our accuracy result shows that this simple structure is effective enough for our dataset.

The training and testing accuracy is compared with baseline in Table 18, which indicates that DaNN greatly improves accuracy. We also draw accuracy curves in Figure 9, and one can observe that testing accuracy stops to increase at about 100 epoch, while training accuracy continues to grow until 200 epochs. This means the model overfits the training set, which may be solved by decreasing hidden layer dimensions.

[5]https://github.com/jindongwang/transferlearning/blob/master/code/traditional/KMM.py
[6]https://github.com/jindongwang/transferlearning/tree/master/code/deep/DaNN

Table 13: TCA accuracy of the 3 datasets, with different kernels and different dimensions.

| Dataset | A_R | | | C_R | | | P_R | | |
|---|---|---|---|---|---|---|---|---|---|
| Dimension\Kernel | Linear | Primal | RBF | Linear | Primal | RBF | Linear | Primal | RBF |
| 32 | **0.194** | 0.713 | 0.2457 | 0.1435 | 0.62 | 0.163 | 0.2239 | 0.7072 | 0.3398 |
| 64 | 0.1906 | 0.7451 | **0.2526** | 0.1435 | **0.6418** | **0.1665** | **0.2342** | 0.7394 | 0.3731 |
| 128 | 0.1883 | **0.7509** | 0.248 | **0.1447** | 0.6361 | 0.163 | 0.2296 | **0.744** | **0.3754** |
| 256 | 0.1871 | 0.7451 | 0.248 | 0.1435 | 0.6257 | 0.163 | 0.2296 | 0.7382 | 0.3743 |
| 512 | 0.1871 | 0.7394 | 0.2468 | 0.1435 | 0.6269 | 0.163 | 0.2296 | 0.7371 | 0.3731 |

Table 14: TCA best accuracy in the 3 datasets, compared with baseline result.

| Dataset\Kernel | Linear | Primal | RBF | baseline |
|---|---|---|---|---|
| A_R | 0.194 | **0.7509** | 0.2526 | 0.744 |
| C_R | 0.1447 | 0.6418 | 0.1665 | **0.649** |
| P_R | 0.2342 | **0.7394** | 0.3754 | 0.73 |

## 4 Conclusion

In this report, we survey and analyze popular domain adaptation methods. With the exception of DaNN, and despite decreased metrics on domain distance like sum of squared error or MMD, domain adaptation methods we surveyed only achieve minor improvements over baseline. Therefore, more effective methods are required to justiy the time and computation required by domain adaptation.

## References


[1] Basura Fernando, Amaury Habrard, Marc Sebban, and Tinne Tuytelaars. Unsupervised visual domain adaptation using subspace alignment. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2013.

[2] Muhammad Ghifary, W. Bastiaan Kleijn, and Mengjie Zhang. Domain adaptive neural networks for object recognition. In Duc-Nghia Pham and Seong-Bae Park, editors, *PRICAI 2014: Trends in Artificial Intelligence*, pages 898–904, Cham, 2014. Springer International Publishing. ISBN 978-3-319-13560-1.

[3] Jiayuan Huang, Arthur Gretton, Karsten Borgwardt, Bernhard Schölkopf, and Alex J Smola. Correcting sample selection bias by unlabeled data. In *Advances in neural information processing systems*, pages 601–608, 2007.

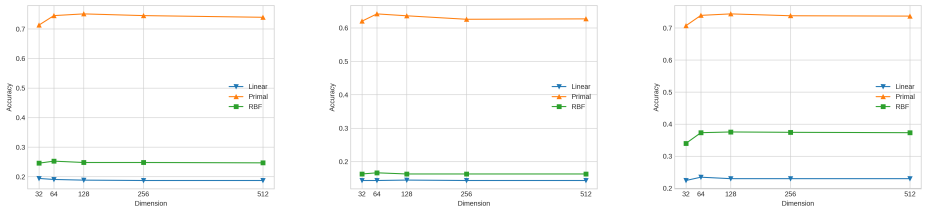
[4] Mingsheng Long, Jianmin Wang, Guiguang Ding, Jiaguang Sun, and Philip S. Yu. Transfer feature learning with joint distribution adaptation. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2013.

[5] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22(2):199–210, 2010.

[6] Baochen Sun and Kate Saenko. Deep coral: Correlation alignment for deep domain adaptation. In *European conference on computer vision*, pages 443–450. Springer, 2016.

[7] Baochen Sun, Jiashi Feng, and Kate Saenko. Correlation alignment for unsupervised domain adaptation. In *Domain Adaptation in Computer Vision Applications*, pages 153–171. Springer, 2017.

[8] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance, 2014.




(a) A_R (b) C_R (c) P_R

Figure 6: TCA accuracy of the 3 datasets, with different kernels and different dimensions.

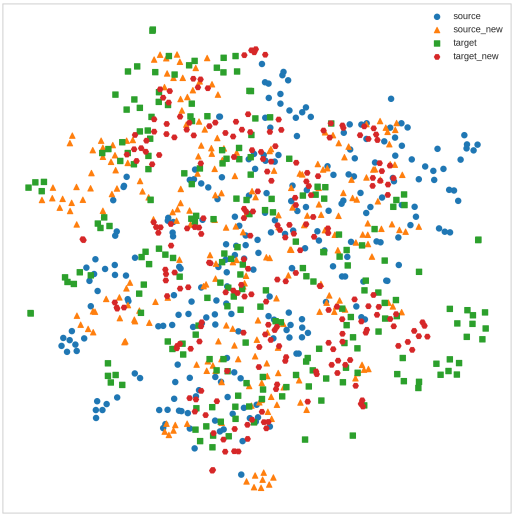



Figure 7: TCA visualization of A_R dataset, with TCA default setting (dim=30, kernel=Primal). The blue circle and green square denote original source and target, and the orange triangle and red square hexagon denote projected source and target after TCA.

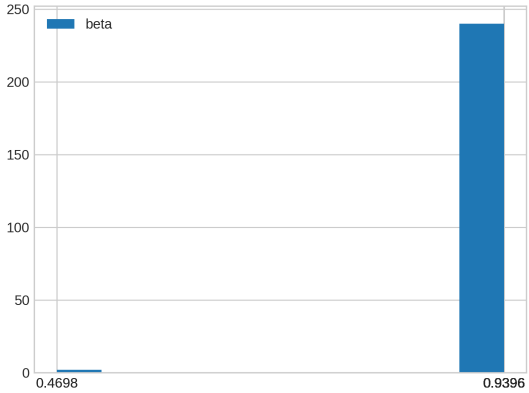
[9] Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[10] Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. In *(IEEE) Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

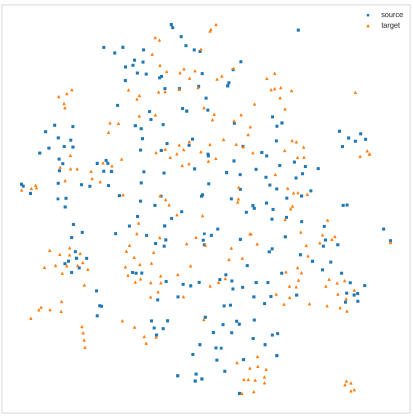
[11] Jindong Wang, Yiqiang Chen, Shuji Hao, Wenjie Feng, and Zhiqi Shen. Balanced distribution adaptation for transfer learning. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 1129–1134. IEEE, 2017.

Table 15: KMM accuracy in the 3 datasets, compared with baseline result.

| Dataset\Kernel | Linear | RBF | baseline |
|---|---|---|---|
| A_R | 0.7313 | 0.7279 | **0.744** |
| C_R | 0.6165 | 0.6154 | **0.649** |
| P_R | 0.7187 | 0.7187 | **0.73** |




(a) Histogram of beta




(b) Data distribution

Figure 8: KMM visualization of A_R dataset, with linear kernel. (a) is histogram of learned weights beta, (b) is data distribution, where the blue square denotes source samples, and orange triangle denotes target samples. Notice that source samples is scaled by its corresponding beta.

Table 16: DaNN network structure.

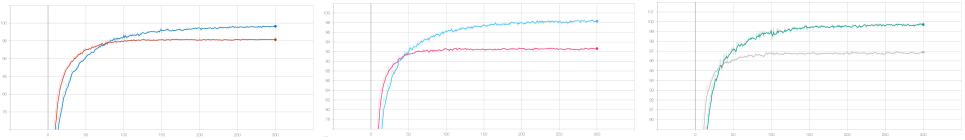
| | Source | | | Target | | |
|---|---|---|---|---|---|---|
| Index | Layer | Output Shape | #Param | Layer | Output Shape | #Param |
| 0 | Input | [-1,2048] | 0 | Input | [-1,2048] | 0 |
| 1 | Linear | [-1, 256] | 524,544 | Linear | [-1, 256] | 524,544 |
| 2 | Dropout | [-1, 256] | 0 | Dropout | [-1, 256] | 0 |
| 3 | ReLU | [-1, 256] | 0 | ReLU | [-1, 256] | 0 |
| 4 | Linear | [-1, 65] | 16,705 | - | - | - |

Table 17: The standard parameter setting of the DaNN.

| Name | Value |
|---|---|
| Learning rate | 0.02 |
| Batch size | 64 |
| Epoch number | 300 |
| Momentum | 0.05 |
| L2 weight regularization | 0.003 |
| Dropout fraction | 0.5 |
| Lambda (MMD loss fraction) | 0.25 |
| Gamma (MMD RBF bandwidth) | 1000 |

Table 18: DaNN training and testing accuracy in the 3 datasets, only testing accuracy is compared with baseline.

| Dataset\Acc | Train Acc | Test Acc | baseline |
|---|---|---|---|
| A_R | 0.9923 | **0.9531** | 0.744 |
| C_R | 0.9831 | **0.9263** | 0.649 |
| P_R | 0.9972 | **0.9688** | 0.73 |



(a) A_R (b) C_R (c) P_R

Figure 9: DaNN training and testing accuracy of the 3 datasets. In (a), training is blue and testing is red, in (b), training is cyan and testing is pink, in (c), training is green and testing is gray.