Domain Adaptation for Image Classification

Group:
Yang Zhou
Yiming Liu
Yakai Wang
Hongkai Zheng

Abstract

In this project, we compared six different traditional domain adaptation methods on SVM-based and 1-NN classifiers. We then investigated three deep learning methods including Deep CORAL, Deep Domain Confusion(DDC),and Deep Adaptation Networks. Further more, we conduct further analysis into Balanced Distribution Adaptation(BDA), and CORrelation ALignment(CORAL). Besides, we adopted 2-D visualization to visualize how traditional domain adaptation algorithms works.

1 Traditional Domain Adaptation Methods

1.1 Subspace Alignment

Subspace Alignment [1] seeks a domain invariant feature space by learning a mapping function which aligns the source subspace with the target one. We assume that both the source and target data $\mathcal{D}_S, \mathcal{D}_T$ lie in the same D-dimensional space and have been drawn according to different distributions. Table 1 shows the notions we use in this section.

Table 1: Notations

Notation	Concept
$\mathcal{D}_S$	source domain data$\{(x_{S_1}, y_{S_1}), \ldots, (x_{S_{n_1}}, y_{S_{n_1}})\}$.
$\mathcal{D}_T$	target domain data$\{x_{T_1}, \ldots, x_{T_{n_2}}\}$.
L_i	class label
$P(X_d)$	the marginal distributions of $X_d, d \in \{S, T\}$
$E_S \in \mathbb{R}^{D \times d}$	basis vectors of the source subspace
$E_T \in \mathbb{R}^{D \times d}$	basis vectors of the target subspace

First, we project source data $\mathcal{D}_S$ and target data $\mathcal{D}_T$ to its respective subspace E_S and E_T, which are generated from the first d eigenvectors. Then, a linear transformation M is learned to map the source subspace to the target one. $M \in \mathbb{R}^{d \times d}$ is learned by minimizing the following Bregman matrix divergence:

$$F(M) = \|E_S M - E_T\|_F^2 \tag{1}$$

$$M^* = \text{argmin}_M(F(M)) \tag{2}$$

where $||.||_F^2$ is the Frobenius norm. Because the Frobenius norm is invariant to orthonormal operations, we can re-write the objective function in Eq.1 as follows:

$$\begin{aligned} M^* &= \operatorname{argmin}_M \left\| E_S^T E_S M - E_S^T E_T \right\|_F^2 \\ &= \operatorname{argmin}_M \left\| M - E_S^T E_T \right\|_F^2 \\ &= E_S^T E_T \end{aligned} \tag{3}$$

Matrix M transforms the source subspace coordinate system into the target subspace coordinate system by aligning the source basis vectors with the target ones.

To compare source data with target data, we define $Sim(\mathbf{x}_S, \mathbf{x}_T)$ as follows:

$$\begin{aligned} \operatorname{sim}\left(\mathbf{x}_S, \mathbf{x}_T\right) &= \left(\mathbf{x}_{\mathrm{S}} E_S M^*\right)\left(\mathbf{x}_{\mathbf{T}} E_T\right)^T \\ &= \mathbf{x}_S A \mathbf{x}_T^T \end{aligned} \tag{4}$$

where $A = E_S X_S^T E_T X_T^T$. We use the matrix A to construct the kernel matrices via $Sim(\curvearrowleft_S, \curvearrowleft_T)$ and perform SVM-based classification. To use with nearest neighbor classifier, we project the source data via Xa into the target aligned source subspace and the target data into the target subspace (using E_T), and perform the classification in this d-dimensional space. The pseudo-code of this algorithm is presented in Algorithm 1.

Algorithm 1: Subspace alignment DA algorithm in the lower d-dimensional space

Input: Source data S, Target data T, Source labels L_S, Subspace dimension d
Output: Predicted target labels L_T

Initialize cluster centroids $\boldsymbol{\mu_k}$, covariances $\boldsymbol{\Sigma_k}$, π_k
$E_S \leftarrow PCA(S, d)$;
$E_T \leftarrow PCA(T, d)$;
$E_a \leftarrow E_S E_S^T E_T$;
$S_a \leftarrow SE_a$;
$T_T \leftarrow TE_T$;
$L_T \leftarrow Classifier(S_a, T_T, L_S)$;

1.2 TCA

TCA(Transfer Component Analysis)[5]is a transfer learning method based on features. For the domain adaptation problem, when the source domain and the target domain are in different data distributions, TCA maps the data of the two domains together to a high-dimensional regenerative kernel Hilbert space. In this space, TCA can minimize the data distance between the source and the target while maximizing their respective internal properties.

In TCA, we focus on the setting where the target domain has no labeled training data, but has plenty of unlabeled data. We also assume that some labeled data $\mathcal{D}_S$ are available in a source domain, while only unlabeled data $\mathcal{D}_T$ are available in the target domain. We denote the source domain data as $\mathcal{D}_S = \left\{\left(x_{S_1}, y_{S_1}\right), \ldots, \left(x_{S_{n_1}}, y_{S_{n_1}}\right)\right\}$, the target domain data as $\mathcal{D}_T = \left\{x_{T_1}, \ldots, x_{T_{n_2}}\right\}$ Let $\mathcal{P}\left(X_S\right)$ and $\mathcal{Q}\left(X_T\right)$ be the marginal distributions of X_S and X_T.

The edge distribution of the source and target domains is different, so we can't directly use traditional machine learning methods. TCA assumes that there is a feature map ϕ such that $P\left(\phi\left(X_S\right)\right) \approx P\left(\phi\left(X_T\right)\right)$, namely $P\left(Y_S|\phi\left(X_S\right)\right) \approx P\left(Y_T|\phi\left(X_T\right)\right)$.

TCA use MMD(Maximum Mean Discrepancy) to describe the distance between source domain and target domain: $\operatorname{Dist}\left(X_S', X_T'\right) = \left\| \frac{1}{n_1} \sum_{i=1}^{n_1} \phi\left(x_{S_i}\right) - \frac{1}{n_2} \sum_{i=1}^{n_2} \phi\left(x_{T_i}\right) \right\|_{\mathcal{H}}^2$.

TCA use a kernel matric K:

$$K = \begin{bmatrix} K_{src,src} & K_{src,tar} \\ K_{tar,src} & K_{tar,tar} \end{bmatrix}$$

and L:

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

to demonstrate MDD: $\operatorname{trace}(KL) - \lambda \operatorname{trace}(K)$.

All in all, the final optimization objective is:

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

, where μ is a trade-off parameter, $I \in \mathbb{R}^{m \times m}$ is the identity matrix, and $H = I_{n_1+n_2} - \frac{1}{n_1+n_2} 11^\top$ is the centering matrix. where $1 \in \mathbb{R}^{n_1+n_2}$ is the column vector with all ones, and $I_{n_1+n_2} \in \mathbb{R}^{(n_1+n_2)\times(n_1+n_2)}$ is the identity matrix. Finally, we get the dimensionally reduced data of the source and target domains, and we can use the traditional machine learning method above.

1.3 BDA

Balanced Distribution Adaptation(BDA)[10] is a novel method to tackle dataset dissimilarity and class imbalance. Given a labeled source domain $\left\{\mathbf{x}_{s_i}, y_{s_i}\right\}_{i=1}^n$, an unlabeled target domain $\left\{\mathbf{x}_{t_j}\right\}_{j=1}^m$, and assume feature space $\mathcal{X}_s = \mathcal{X}_t$, label space $\mathcal{Y}_s = \mathcal{Y}_t$ but marginal distributions $P_s\left(\mathbf{x}_s\right) \neq P_t\left(\mathbf{x}_t\right)$ with conditional distributions $P_s\left(y_s|\mathbf{x}_s\right) \neq P_s\left(y_t|\mathbf{x}_t\right)$. Transfer learning aims to learn the labels $\mathbf{y}_t$ of $\mathcal{D}_t$ using the source domain $\mathcal{D}_s$. BDA solves the transfer learning problem by adaptively minimizing the marginal and conditional distribution discrepancy between domains, and handle the class imbalance problem, i.e. to minimize the discrepancies between 1) $P\left(\mathbf{x}_s\right)$ and $P\left(\mathbf{x}_t\right)$, 2) $P\left(y_s|\mathbf{x}_s\right)$ and $P\left(y_t|\mathbf{x}_t\right)$. BDA can not only adapt both the marginal and conditional distributions between domains, but also leverage the importance of those two distributions, thus it can be effectively adjusted to specific transfer learning tasks.

BDA exploits a balance factor μ to leverage the different importance of distributions

$$D\left(\mathcal{D}_s, \mathcal{D}_t\right) \approx (1-\mu) D\left(P\left(\mathbf{x}_s\right), P\left(\mathbf{x}_t\right)\right) \tag{5}$$

$$+ \mu D\left(P\left(y_s|\mathbf{x}_s\right), P\left(y_t|\mathbf{x}_t\right)\right) \tag{6}$$

where $\mu \in [0,1]$. When $\mu \to 0$, it means two datasets are more dissimilar, so the marginal distribution is more dominant; when $\mu \to 1$, it reveals that the datasets are similar, so the conditional distribution is more important to adapt. Therefore, the balance factor μ can adaptively leverage the importance of each distribution and lead to good results.

In order to compute the marginal and conditional distribution divergences, we adopt maximum mean discrepancy to empirically estimate both distribution discrepancies. As a nonparametric measurement, MMD has been widely applied to many existing transfer learning approaches. This equation can be represented as

$$D\left(\mathcal{D}_s, \mathcal{D}_t\right) \approx (1-\mu) \left\| \frac{1}{n} \sum_{i=1}^n \mathbf{x}_{s_i} - \frac{1}{m} \sum_{j=1}^m \mathbf{x}_{t_j} \right\|_{\mathcal{H}}^2 \tag{7}$$

$$+ \mu \sum_{c=1}^C \left\| \frac{1}{n_c} \sum_{\mathbf{x}_{s_i} \in \mathcal{D}_s^{(c)}} \mathbf{x}_{s_i} - \frac{1}{m_c} \sum_{\mathbf{x}_{t_j} \in \mathcal{D}_t^{(c)}} \mathbf{x}_{t_j} \right\|_{\mathcal{H}} \tag{8}$$

by further taking advantage of matrix tricks and regularizaion, it can be formalized as :

$$\min \operatorname{tr}\left(\mathbf{A}^\top \mathbf{X}\left((1-\mu)\mathbf{M}_0 + \mu \sum_{c=1}^C \mathbf{M}_c\right)\mathbf{X}^\top \mathbf{A}\right) + \lambda \|\mathbf{A}\|_F^2 \tag{9}$$

$$s.t. \mathbf{A}^\top \mathbf{X}\mathbf{H}\mathbf{X}^\top \mathbf{A} = \mathbf{I}, \quad 0 \leq \mu \leq 1 \tag{10}$$

Similarly, $\mathbf{M}_0$ and $\mathbf{M}_c$ are MMD matrices and can be constructed in the following ways:

$$(\mathbf{M}_0)_{ij} = \begin{cases} \frac{1}{n^2}, & \mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}_s \\ \frac{1}{m^2}, & \mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}_t \\ -\frac{1}{mn}, & \text{otherwise} \end{cases} \tag{11}$$

$$(\mathbf{M}_c)_{ij} = \begin{cases} \frac{1}{n^2}, & \mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}_s^{(c)} \\ \frac{1}{m^2}, & \mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}_t^{(c)} \\ -\frac{1}{mn}, & \begin{cases} \mathbf{x}_i \in \mathcal{D}_s^{(c)}, \mathbf{x}_j \in \mathcal{D}_t^{(c)} \\ \mathbf{x}_i \in \mathcal{D}_t^{(c)}, \mathbf{x}_j \in \mathcal{D}_s^{(c)} \end{cases} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

Learning algorithm: Denote $\Phi = (\phi_1, \phi_2, \cdots, \phi_d)$ as Lagrange multipliers, then Lagrange function is

$$L = \operatorname{tr}\left(\mathbf{A}^\top \mathbf{X}\left((1-\mu)\mathbf{M}_0 + \mu \sum_{c=1}^{C} \mathbf{M}_c\right)\mathbf{X}^\top \mathbf{A}\right) \tag{13}$$

$$+ \lambda \|\mathbf{A}\|_F^2 + \operatorname{tr}\left(\left(\mathbf{I} - \mathbf{A}^\top \mathbf{X}\mathbf{H}\mathbf{X}^\top \mathbf{A}\right)\Phi\right) \tag{14}$$

Set derivative $\partial L / \partial \mathbf{A} = 0$, the optimization can be derived as a generalized eigendecomposition problem

$$\left(\mathbf{X}\left((1-\mu)\mathbf{M}_0 + \mu \sum_{c=1}^{C} \mathbf{M}_c\right)\mathbf{X}^\top + \lambda \mathbf{I}\right)\mathbf{A} = \mathbf{X}\mathbf{H}\mathbf{X}^\top \mathbf{A}\Phi \tag{15}$$

Finally, the optimal transformation matrix $\mathbf{A}$ can be obtained by solving equation and finding its d smallest eigenvectors.

1.4 JDA

JDA(Joint Distribution Adaptation)[4] assumes that 1)The marginal distributions of source and target domain are different; 2) The conditional distributions of source and target domain are different.

The goal of JDA is to find a transformation matrix $\mathbf{A}$ such that the distance between $P\left(\mathbf{A}^\top \mathbf{x}_s\right)$ and $P\left(\mathbf{A}^\top \mathbf{x}_t\right)$ is as small as possible while the distance between $P\left(y_s | \mathbf{A}^\top \mathbf{x}_s\right)$ and $P\left(y_t | \mathbf{A}^\top \mathbf{x}_t\right)$ is as small as possible. So there are totally 2 steps in JDA.

Marginal distribution adaptation. We want to make the distance between $P\left(\mathbf{A}^\top \mathbf{x}_s\right)$ and $P\left(\mathbf{A}^\top \mathbf{x}_t\right)$ is as small as possible. We use TCA to do this: $D\left(\mathcal{D}_s, \mathcal{D}_t\right) = \operatorname{tr}\left(\mathbf{A}^\top \mathbf{X}\mathbf{M}_0\mathbf{X}^\top \mathbf{A}\right)$ where $\mathbf{M}_0$ is a MMD matrix:

$$(\mathbf{M}_0)_{ij} = \begin{cases} \frac{1}{n^2}, & \mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}_s \\ \frac{1}{m^2}, & \mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}_t \\ -\frac{1}{mn}, & \text{otherwise} \end{cases}$$

where n, m is the sample numbers in source and target domain respectively.

Conditional distribution adaptation. In this part, our goal is to make the distance between $P\left(y_s | \mathbf{A}^\top \mathbf{x}_s\right)$ and $P\left(y_t | \mathbf{A}^\top \mathbf{x}_t\right)$ as small as possible. However, in our target domain, there is no y_t, so we can't find the conditional distribution of the target domain. We use $(\mathbf{x}_s, y_s)$ to train a simple calssifier, and predict directly on $\mathbf{x}_t$to get the pseudo-label $\hat{y}_t$. The MMD distance between classes can be defined as:

$$\sum_{c=1}^{C} \left\| \frac{1}{n_c} \sum_{\mathbf{x}_{s_i} \in \mathcal{D}_s^{(e)}} \mathbf{A}^\top \mathbf{x}_{s_i} - \frac{1}{m_c} \sum_{\mathbf{x}_{t_i} \in \mathcal{D}_t^{(c)}} \mathbf{A}^\top \mathbf{x}_{t_i} \right\|_{\mathcal{H}}^2$$

where n_c, m_c is the sample numbers of source and target domain in the c^{th} class. We use the same method in TCA, and de distance can be written as follows:

$$\sum_{c=1}^{C} \operatorname{tr}\left(\mathbf{A}^\top \mathbf{X}\mathbf{M}_c\mathbf{X}^\top \mathbf{A}\right)$$

,where

$$(\mathbf{M}_c)_{ij} = \begin{cases} \frac{1}{n_c^2}, & \mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}_s^{(c)} \\ \frac{1}{m^2}, & \mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}_t^{(c)} \\ -\frac{1}{m_c n_c}, & \mathbf{x}_i \in \mathcal{D}_s^{(c)}, \mathbf{x}_j \in \mathcal{D}_t^{(c)} or \mathbf{x}_i \in \mathcal{D}_t^{(c)}, \mathbf{x}_j \in \mathcal{D}_s^{(c)} \\ 0 & \text{otherwise} \end{cases}$$

Finally, we combine the two distances and the final optimization objective

$$\min \sum_{c=0}^{C} \operatorname{tr}\left(\mathbf{A}^\top \mathbf{X} \mathbf{M}_c \mathbf{X}^\top \mathbf{A}\right) + \lambda \|\mathbf{A}\|_F^2$$

Similarly to TCA, there is a restriction: s.t. $\mathbf{A}^\top \mathbf{X} \mathbf{H} \mathbf{X}^\top \mathbf{A} = \mathbf{I}$.

However, the accuracy of pseudo-labels is not high, so we have to implement iretations on it. finally, the pseudo-labels are more and more accurate.

1.5 CORAL

CORrelation ALignment (CORAL)[6] is a "frustratingly easy" unsupervised domain adaptation method. CORAL is simple and efficient, as the only computations it needs is (1)computing covariance statistics in each domain and (2)applying the whitening and re-coloring linear transformation to the source features. Then supervised learning proceeds as usual-training a classifier on the transformed source features.

Suppose we are given $D_S = \{\vec{x}_i\}, \vec{x} \in \mathbb{R}^D$ with labels $L_S = \{y_i\}, y \in \{1, \ldots, L\}$, and target data $D_T = \{\vec{u}_i\}, \vec{u} \in \mathbb{R}^D$. Here both $\vec{x}$ and $\vec{u}$ are the D-dimensional feature representations $\phi(I)$ of input I. Suppose μ_s,μ_t and C_S, C_T are the feature vector means and covariance matrices. $\mu_t = \mu_s = 0$ after feature normalization while $C_S \neq C_T$. 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:

$$\min_A \left\|C_{\hat{S}} - C_T\right\|_F^2 = \min_A \left\|A^\top C_S A - C_T\right\|_F^2 \tag{16}$$

where $C_{\hat{S}}$ is covariance of the transformed source features $D_s A$ and $\|\cdot\|_F^2$ denotes the matrix Frobenius norm. source-domain training examples. Combining the results in the above two cases(1)$r_{C_S} > r_{C_T}$(2) $r_{C_S} \leq r_{C_T}$ yields that $C_{\hat{S}} = U_{T[1:r]} \Sigma_{T[1:r]} U_{T[1:r]}^\top$ is the optimal solution to Equation with $r = \min(r_{C_S}, r_{C_T})$. We then proceed to solve for A based on the above result. Let $C_{\hat{S}} = A^\top C_S A$, and we get:

$$A^\top C_S A = U_{T[1:r]} \Sigma_{T[1:r]} U_{T[1:r]}^\top \tag{17}$$

since $C_S = U_S \Sigma_S U_S^\top$,we have

$$A^\top U_S \Sigma_S U_S^\top A = U_{T[1:r]} \Sigma_{T[1:r]} U_{T[1:r]}^\top \tag{18}$$

This gives:

$$\left(U_S^\top A\right)^\top \Sigma_S \left(U_S^\top A\right) = U_{T[1:r]} \Sigma_{T[1:r]} U_{T[1:r]}^\top \tag{19}$$

Let $E = \Sigma_S^{+\frac{1}{2}} U_S^\top U_{T[1:r]} \Sigma_{T[1:r]}^{\frac{1}{2}} U_{T[1:r]}^\top$, then the right hand side of the above equation can be re-written as $E^\top \Sigma_S E$. This gives

$$\left(U_S^\top A\right)^\top \Sigma_S \left(U_S^\top A\right) = E^\top \Sigma_S E \tag{20}$$

By setting $U_S^\top A$ to E, we get the optimal solution of A as

$$A^* = U_S E \tag{21}$$

$$= \left(U_S \Sigma_S^{+\frac{1}{2}} U_S^\top\right) \left(U_{T[1;r]} \Sigma_{T[1:r]}^{\frac{1}{2}} U_{T[1:r]}^\top\right) \tag{22}$$

1.6 KMM

Kernel mean matching(KMM)[2] accounts for difference between source domain and target domain by reweighting the training points such that the means of the training and test points in a reproducing kernel Hilbert space (RKHS) are close. KMM tries to minimizes the expected risk of a loss function. Firstly, it estimates the reweighting coefficients β_i.

$$\min_{\beta_i} \left\| \frac{1}{n_s} \sum_{i=1}^{n_s} \beta_i \phi\left(\mathbf{x}_i^s\right) - \frac{1}{n_t} \sum_{i=1}^{n_t} \phi\left(\mathbf{x}_i^t\right) \right\|^2 \tag{23}$$

For original SVM minimization problem, sample reweighting can be utilized as following:

$$\min_{\mathbf{w},b,\xi_i} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \beta_i \xi_i \tag{24}$$

$$s.t. \quad y_i\left(\mathbf{w}^T\phi\left(\mathbf{x}_i\right) + b\right) \geq 1 - \xi_i, \quad \forall i \tag{25}$$

$$\xi_i \geq 0, \quad \forall i \tag{26}$$

2 Deep Domain Adaption Method

2.1 Deep CORAL

CORAL relies on a linear transformation and is not end-to-end: it needs to first extract features, apply the transformation, and then train an SVM classifier in a separate step. Deep CORAL[7] can be directly utilized into deep networks by constructing a differentiable loss function that minimizes the difference between source and target correlations–the CORAL loss. Deep CORAL approach learns a non-linear transformation that is more powerful and also works seamlessly with deep CNNs as it is shown in 1.

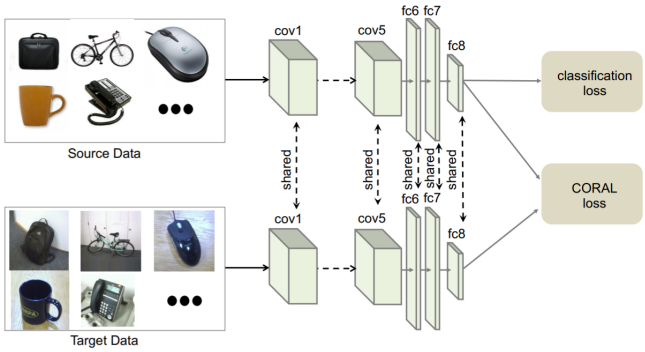


Figure 1: Sample Deep CORAL architecture based on a CNN with a classifier layer.

2.1.1 CORAL Loss

Suppose we are given source-domain training examples $D_S = \{\mathbf{x}_i\}, \mathbf{x} \in \mathbb{R}^d$ with labels $L_S = \{y_i\}, i \in \{1, \ldots, L\}$, 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 D_S^{ij} $\left(D_T^{ij}\right)$ indicates the j-th dimension of the i-th source (target) data example and C_S (C_T) denote the feature covariance matrices.

CORAL loss is defined as the distance between the second-order statistics (covariances) of the source and target features:

$$\ell_{CORAL} = \frac{1}{4d^2}\|C_S - C_T\|_F^2 \tag{27}$$

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

$$C_S = \frac{1}{n_S - 1}\left(D_S^\top D_S - \frac{1}{n_S}\left(\mathbf{1}^\top D_S\right)^\top \left(\mathbf{1}^\top D_S\right)\right) \tag{28}$$

$$C_T = \frac{1}{n_T - 1}\left(D_T^\top D_T - \frac{1}{n_T}\left(\mathbf{1}^\top D_T\right)^\top \left(\mathbf{1}^\top D_T\right)\right) \tag{29}$$

where $\mathbf{1}$ is a column vector with all elements equal to 1. The gradient with respect to the input features can be calculated using the chain rule:

$$\frac{\partial \ell_{CORAL}}{\partial D_S^{ij}} = \frac{1}{d^2\left(n_S - 1\right)}\left(\left(D_S^\top - \frac{1}{n_S}\left(\mathbf{1}^\top D_S\right)^\top \mathbf{1}^\top\right)^\top \left(C_S - C_T\right)\right)^{ij} \tag{30}$$

$$\frac{\partial \ell_{CORAL}}{\partial D_T^{ij}} = -\frac{1}{d^2\left(n_T - 1\right)}\left(\left(D_T^\top - \frac{1}{n_T}\left(\mathbf{1}^\top D_T\right)^\top \mathbf{1}^\top\right)^\top \left(C_S - C_T\right)\right)^{ij} \tag{31}$$

We use batch covariances and the network parameters are shared between two networks.

2.1.2 End-to-end Domain Adaptation with CORAL Loss

Minimizing the classification loss itself is likely to lead to overfitting to the source domain, causing reduced performance on the target domain. On the other hand, minimizing the CORAL loss alone might lead to degenerated features. Joint training with both the classification loss and CORAL loss is likely to learn features that work well on the target domain:

$$\ell = \ell_{CLASS.} + \sum_{i=1}^{t} \lambda_i \ell_{CORAL} \tag{32}$$

where t denotes the number of CORAL loss layers in a deep network and λ is a weight that trades off the adaptation with classification accuracy on the source domain. These two losses play counterparts and reach an equilibrium at the end of training, where the final features are expected to work well on the target domain.

2.2 Deep Domain Confusion

DDC[8] is a new CNN architecture, outlined in Figure2.This architecture (see Figure2) consists of a source and target CNN, with shared weights. 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.

A domain confusion loss is represented based on maximum mean discrepancy (MMD).In our case, we define a representation, $\phi(\cdot)$, which operates on source data points,$x_s \in X_S$, and target data points,$x_t \in X_T$.Then an empirical approximation to this distance is computed as followed:

$$MMD\left(X_S, X_T\right) = \tag{33}$$

$$\left\|\frac{1}{|X_S|}\sum_{x_s \in X_S}\phi\left(x_s\right) - \frac{1}{|X_T|}\sum_{x_t \in X_T}\phi\left(x_t\right)\right\| \tag{34}$$

One approach to minimize the distance between domains and train strong classifiers is to minimize the loss:

$$\mathcal{L} = \mathcal{L}_C\left(X_L, y\right) + \lambda \mathrm{MMD}^2\left(X_S, X_T\right) \tag{35}$$

where $\mathcal{L}_C\left(X_L, y\right)$ denotes classification loss on the available labeled data, X_L, and the ground truth labels, y, and $MMD(X_S, X_T)$ denotes the distance between the source data, X_S, and the target data, X_T . The hyperparameter λ determines how strongly we would like to confuse the domains.

This loss function 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

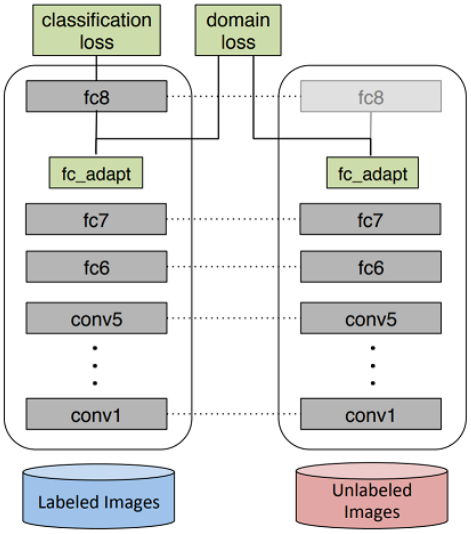


Figure 2: DDC architecture optimizes a deep CNN for both classification loss as well as domain invariance.

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. We expect that such a representation will thus enable increased adaptation performance.

2.3 DAN

DAN(Deep Adaptation Networks)[3] was a network based on DDC(Deep Domain Confusion)[8]. Briefly, for the pre-trained AlexNet (8-layer) network, DDC add the MMD distance to the 7th layer (the feature layer, the upper layer of softmax) to reduce the difference between source and target, and add an adaption layer behind the $fc7$ layer. Different from DDC, DAN adds several adaption layers of Alex-Net model. What's more, it use MK-MMD instead of the traditional MMD. The network structure3 is shown as follows:

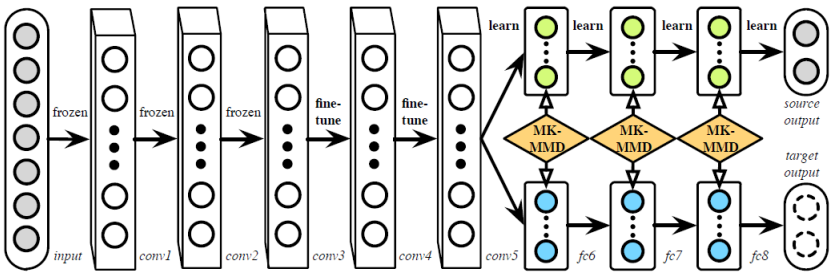


Figure 3: The DAN Network Structure

Based on the AlexNet network, DAN explores the adaptation relationship between source and target. Its optimization objective consists of two parts: the loss function and the distribution distance. The loss function measures the difference between the predicted value and the true value. The distribution

distance is the MK-MMD distance we mentioned above. Therefore, DAN's optimization goal is

$$\min_{\Theta} \frac{1}{n_a} \sum_{i=1}^{n_a} J\left(\theta\left(\mathbf{x}_i^a\right), y_i^a\right) + \lambda \sum_{l=l_1}^{l_2} d_k^2\left(\mathcal{D}_s^l, \mathcal{D}_t^l\right)$$

In this formula, Θ indicates the weight of the network and the bias parameter, which is the target for learning. $l_1 = 6, l_2 = 8$, which indicates that the network adaptation is from the 6^{th} layer to the 8^{th} layer, and the previous layers are not adapted. $\mathbf{x}_a$, n_a represents a collection of all data with labels in source and target. λ is the penalty factor. $J(\cdot)$ defines a loss function, which is generally cross-entropy in deep networks.

3 Experiment and Result

In our experiments, we first evaluate five traditional domain adaptation methods: TCA, JDA, CORAL, BDA, and SA, using 1-Nearest Neighbor-based classification. Then we use SVM-based classification to investigate five traditional domain adaptation methods: TCA, JDA, CORAL, BDA, and KMM. We also evaluate three deep learning methods including DDC, DAN, and Deep CORAL on the benchmark dataset. Further more, we conduct sensitivity analysis for CORAL, BDA, and SA.

3.1 Dataset and Implementation

To evaluate all domain adaptation methods, we use office-home dataset [9]that contains four domains altogether with each domain containing images from 65 categories of everyday objects and a total of around 15, 500 images. The domains include, Art(denoted by **A**), Clipart(denoted by **C**), Product(denoted by **P**), and Real-World(denoted by **R**).

All the experiments are conducted in three settings(source domain $\rightarrow$ target domain): 1) **A** $\rightarrow$ **R** 2) **C** $\rightarrow$ **R** 3) **P**$\rightarrow$ **R**. Training set includes source labeled data and target unlabeled data. Testing set includes target labeled data.

As for traditional methods, we use 2048-dim ResNet50 deep learning features of all images. And we use original images for deep learning methods. All data is preprocessed by z-normalization.

3.2 Traditional Methods

In this section, we first compare TCA, JDA, CORAL, BDA, and SA, using 1-NN classifier on three settings. Table 3.2 reports the classification accuracy of these methods on target data. It is evident that JDA got the first place in $P \rightarrow R$ setting, BDA performed best in $C \rightarrow R$, SA reached the best in $A \rightarrow R$, showing their ability to construct more effective and robust representation for classification task.

We then compare TCA, JDA, CORAL, BDA, and KMM with SVM-based classifier. Here we set the C in SVM to be 1. Table 3 depicts the classification performance on the target domain. We observe that KMM achieves better performance than the other methods. But JDA and BDA still stand out although at a considerable distance from KMM.

	1NN	TCA	JDA	BDA	CORAL	SA
A->R	0.6582	0.6136	0.6662	0.6641	0.6485	**0.6770**
C->R	0.5877	0.5475	0.6102	**0.6109**	0.5815	0.6095
P->R	0.6958	0.6458	**0.6995**	0.6935	0.685	0.6965

Table 2: Compare TCA, JDA, CORAL, BDA, and SA with 1-NN classifier

3.3 Deep Learning

In this section, we compare three deep learning methods including deep CORAL, DDC, DAN.

Table 4 demonstrated the classification accuracy over target data. As is shown in the table, the performance of DDC and DAN are very close while DeepCORAL seems a little worse. Allowing for the fact that DDC and DAN share a very similar architecture, it is reasonable that they have similar performance. But all of them outperform the traditional methods.

	TCA	JDA	BDA	CORAL	KMM(rbf)	KMM(linear)
A->R	0.4401	0.6986	0.6974	0.7064	0.7424	**0.7426**
C->R	0.5197	0.6377	0.637	0.5953	**0.6473**	0.649
P->R	0.6276	0.7165	0.7188	0.6903	**0.7305**	0.7213

Table 3: Compare TCA, JDA, CORAL, BDA, and KMM with SVM classifier

	DDC	DAN	DeepCORAL
A->R	**0.7645**	0.7638	0.7551
C->R	0.669	**0.6782**	0.666
P->R	0.7549	**0.7558**	0.7482

Table 4: Results of deep learning methods

3.4 Further Analysis

BDA: We run BDA with varying balance factor $\mu \in \{0, 0.2, \ldots, 0.99\}$. Figure 4 shows the results. It is obvious that optimal μ varies on different tasks, indicating the importance to balance the marginal and conditional distributions between domains. In task $A \rightarrow R$ with optimal $\mu = 0.2$, it means the marginal distributions contributes most to the discrepancy. In task $C \rightarrow R$ with optimal $\mu = 0.8$, it means the marginal distribution are almost the same so the performance mostly depends on conditional distribution.

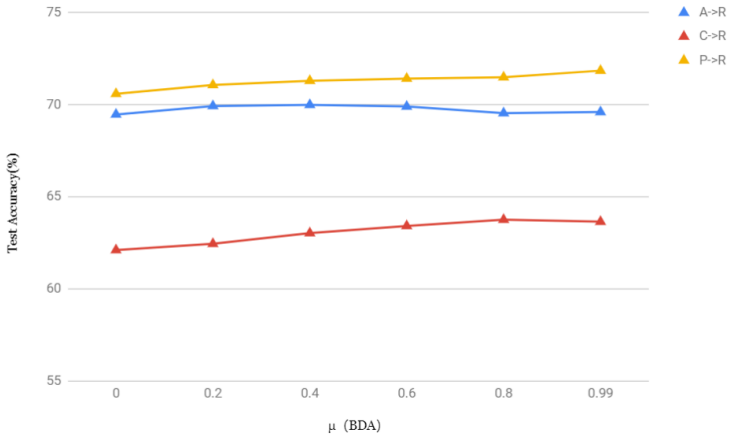


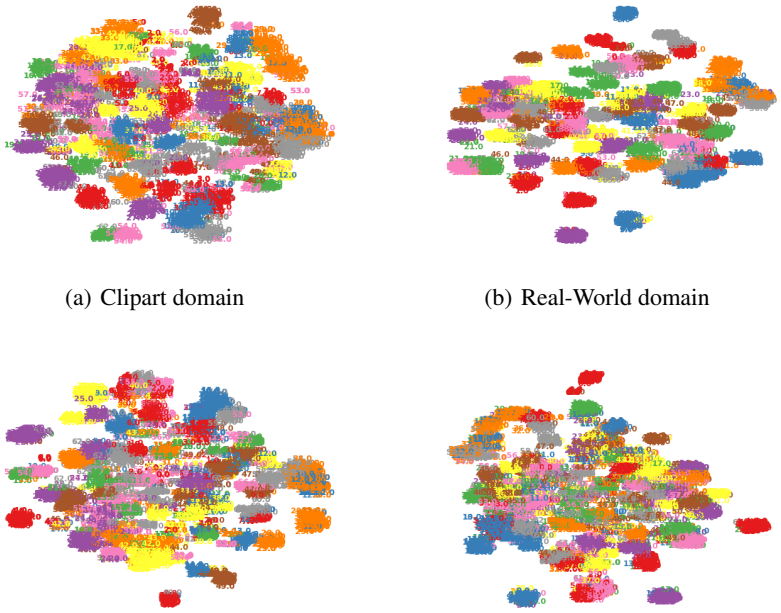
Figure 4: BDA results by varying μ

CORAL: We run CORAL with varying regularization parameter λ. Figure 6 depicts the results. As is shown in the figure, the larger the regularization parameter is, the better the performance will be, validating its ability to making the covariance matrix full rank and helping construct more effective representation.

SA: We further study SA by varying subspace dimensionality i.e the number d of eigenvectors from 16 to 512. As demonstrated in figure 7, we find that 80 is probably the intrinsic subspace dimensionality.

3.5 Visualization

To have a intuitive view into how domain adaptation algorithms works, we use t-SNE to visualize the source and target domain data, and data that transferred by these algorithms. Here, we take Clipart domain as source data and Real-World domain as target data for example. JDA algorithm is used for domain adaptation. In figure 5, (a) (b) visualize the original source domain and target domain; (c),(d) depicts the transferred source domain and target domain. It is obvious that with the help of JDA, source domain and target domain is projected to a intermediate common subspace.



(a) Clipart domain (b) Real-World domain

(c) Transfered Clipart domain (d) Transfered Real-Wrold domain

Figure 5: 2-D visualization of JDA transfer learning

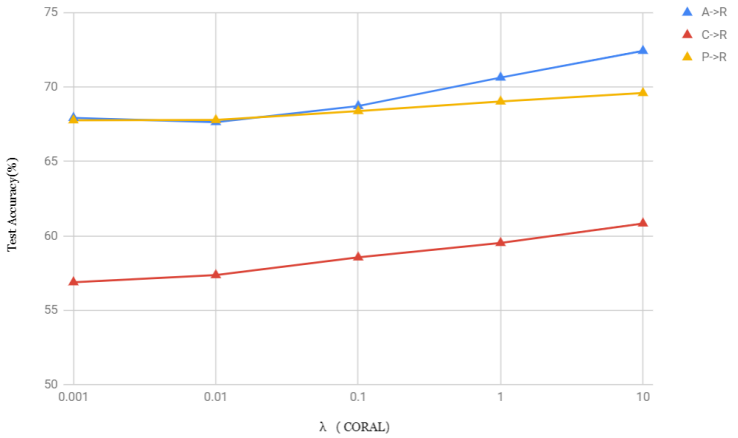


Figure 6: CORAL results by varying λ

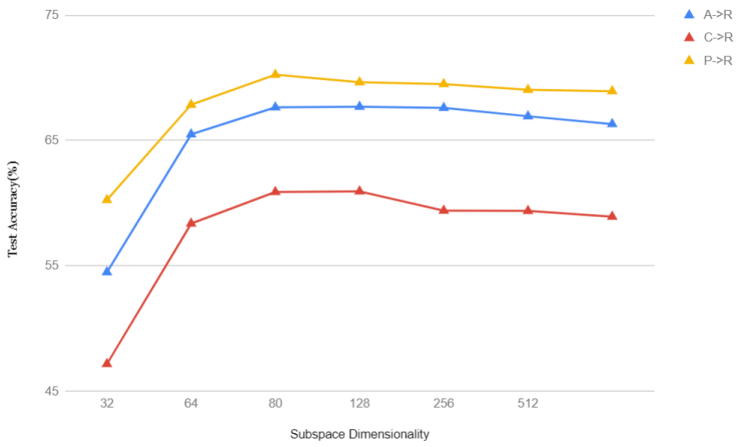


Figure 7: SA results by varying dimensionality d

4 Conclusion

In conclusion, we compared six different traditional domain adaptation methods including TCA, JDA, CORAL, BDA, KMM, and SA. Experimental results show that JDA, BDA, SA have strong ability ti construct more robust representation for classfication task. We then investigated three deep learning methods including Deep CORAL, Deep Domain Confusion(DDC),and Deep Adaptation Networks. As is summarized in table 5, results on the benchmark dataset proved that deep learning methods generally outperforms traditional methods in each task setting. Further more, we conduct further analysis into Balanced Distribution Adaptation(BDA), and CORrelation ALignment(CORAL). Besides, we adopted 2-D visualization to visualize how traditional domain adaptation algorithms works.

Setting	Method	Performance
A->R	DDC	**0.7645**
C->R	DAN	**0.6782**
P->R	DAN	**0.7558**

Table 5: Best results on each task setting

References

[1] B. Fernando, A. Habrard, M. Sebban, and T. Tuytelaars. Subspace alignment for domain adaptation. *CoRR*, abs/1409.5241, 2014.

[2] Huang Correcting sample selection bias by unlabeled data. In *NIPS'06 Proceedings of the 19th International Conference on Neural Information Processing Systems*, pages 601–608, 2006.

[3] M. Long, Y. Cao, J. Wang, and M. I. Jordan. Learning transferable features with deep adaptation networks. *arXiv preprint arXiv:1502.02791*, 2015.

[4] M. Long, J. Wang, G. Ding, J. Sun, and P. S. Yu. Transfer feature learning with joint distribution adaptation. In *Proceedings of the IEEE international conference on computer vision*, pages 2200–2207, 2013.

[5] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22(2):199–210, 2010.

[6] B. Sun, J. Feng, and K. Saenko. Return of frustratingly easy domain adaptation. In *AAAI Publications, Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[7] B. SunEmail and K. Saenko. Deep coral: Correlation alignment for deep domain adaptation. In *Hua G., Jégou H. (eds) Computer Vision – ECCV 2016 Workshops. ECCV 2016. Lecture Notes in Computer Science, vol 9915. Springer, Cham*, 2016.

[8] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell. Deep domain confusion: Maximizing for domain invariance. In *arxiv*, 2014.

[9] H. Venkateswara, J. Eusebio, S. Chakraborty, and S. Panchanathan. Deep hashing network for unsupervised domain adaptation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 5385–5394, 2017.

[10] J. Wang, Y. Chen, S. Hao, W. Feng, and Z. Shen. Balanced distribution adaptation for transfer learning. In *2017 IEEE International Conference on Data Mining,18-21 Nov. 2017,New Orleans, LA, USA*, 2017.