# Project 4 of CS245: Domain Adaptation for Image Classification

BoWen Zhang
*517021910797*

Xinran Chen
*517030910099*

Shuyuan Fu
*517030910124*

***Abstract*—Domain adaptation is one of the most common problems in transfer learning. The purpose is to map the data from source domain and target domain to a feature space, so that their distance in the space is minimized. In this report, we use *Office-Home dataset* [1] as our dataset, and try traditional methods (DIP, TCA, SA, CORAL, SGF, GFK, KMM) and deep learning methods (DAN, Deep Coral, DANN, CDAN) to do domain adaptation and evaluate the performances of them.**

## 1. Introduction

In this project, we implement several domain adaptation methods, covering traditional methods and deep learning methods. The traditional methods we tried are **DIP, TCA, SA, CORAL, SGF, GFK, KMM**, and the deep learning methods we tried are **DAN, Deep Coral, DAN, CDAN**, 11 methods in total. For each method, we apply the obtained domain adaptation results into the SVM classifier to evaluate the performance. For some traditional methods, we also use t-SNE to do the visualization to see the data distribution. Based on the results, we give our analyses and make some comparisons.

The remainder of this paper starts with the illustration of the principles of domain adaptation methods, followed by the statement of the procedures and results of our experiment. At the end of this paper, we give the conclusion of our experiment.

## 2. Traditional methods

### 2.1. Domain Invariant Projection (DIP)

The purpose of DIP [2] is to achieve invariance, which is that classifier trained in the source domain performs equally well in the target domain. We try to find a projection that minimizes a distance metric between the two distributions.

Suppose $\boldsymbol{X_s} = \left[\boldsymbol{x}_s^1, \cdots, \boldsymbol{x}_s^n\right]$ to be the $D \times n$ matrix containing $n$ samples from the source domain and $\boldsymbol{X_t} = \left[\boldsymbol{x}_t^1, \cdots, \boldsymbol{x}_t^m\right]$ to be the $D \times m$ matrix containing $m$ samples from the target domain. We want to find a $D \times d$ projection matrix $W$, such that the distributions of the source and target samples in the resulting $d$-dimensional subspace are as similar as possible. DIP measures the distance between these two distributions with the MMD.

$$\begin{aligned} & D\left(\boldsymbol{W}^T \boldsymbol{X}_s, \boldsymbol{W}^T \boldsymbol{X}_t\right) \\ & = \left\| \frac{1}{n} \sum_{i=1}^{n} \phi\left(\boldsymbol{W}^T \boldsymbol{x}_s^i\right) - \frac{1}{m} \sum_{j=1}^{m} \phi\left(\boldsymbol{W}^T \boldsymbol{x}_t^j\right) \right\|_{\mathcal{H}} \end{aligned} \tag{1}$$

where $\phi(\cdot)$ maps samples from $\mathbb{R}^D$ to the high-dimensional RKHS $\mathcal{H}$.

Then we get the optimization problem:

$$\begin{aligned} \min_{\boldsymbol{W}} \quad & D^2\left(\boldsymbol{W}^T \boldsymbol{X_s}, \boldsymbol{W}^T \boldsymbol{X_t}\right) \\ \text{s.t.} \quad & \boldsymbol{W}^T \boldsymbol{W} = \boldsymbol{I}_d \end{aligned} \tag{2}$$

where the constraints enforce $W$ to be orthogonal. Such constraints prevent our model from wrongly matching the two distributions by distorting the data.

The MMD in the RKHS $\mathcal{H}$ can be expressed in terms of a kernel function $k(\cdot, \cdot)$. In particular here, we exploit the Gaussian kernel function, thus, the objective function is:

$$\begin{aligned} & D^2\left(\boldsymbol{W}^T \boldsymbol{X_s}, \boldsymbol{W}^T \boldsymbol{X_t}\right) \\ & = \frac{1}{n^2} \sum_{i,j=1}^{n} \exp\left(-\frac{\left(\boldsymbol{x}_s^i - \boldsymbol{x}_s^j\right)^T \boldsymbol{W}\boldsymbol{W}^T \left(\boldsymbol{x}_s^i - \boldsymbol{x}_s^j\right)}{\sigma}\right) \\ & + \frac{1}{m^2} \sum_{i,j=1}^{m} \exp\left(-\frac{\left(\boldsymbol{x}_t^i - \boldsymbol{x}_t^j\right)^T \boldsymbol{W}\boldsymbol{W}^T \left(\boldsymbol{x}_t^i - \boldsymbol{x}_t^j\right)}{\sigma}\right) \\ & - \frac{2}{mn} \sum_{i,j=1}^{n,m} \exp\left(-\frac{\left(\boldsymbol{x}_s^i - \boldsymbol{x}_t^j\right)^T \boldsymbol{W}\boldsymbol{W}^T \left(\boldsymbol{x}_s^i - \boldsymbol{x}_t^j\right)}{\sigma}\right) \end{aligned} \tag{3}$$

The Eqs. 3 can be computed efficiently in matrix form as

$$D^2\left(\boldsymbol{W}^T \boldsymbol{X}_s, \boldsymbol{W}^T \boldsymbol{X}_t\right) = \operatorname{Tr}\left(\boldsymbol{K_W} \boldsymbol{L}\right) \tag{4}$$

where $\boldsymbol{K_W} = \begin{bmatrix} \boldsymbol{K}_{s,s} & \boldsymbol{K}_{s,t} \\ \boldsymbol{K}_{t,s} & \boldsymbol{K}_{t,t} \end{bmatrix} \in \mathbb{R}^{(n+m)\times(n+m)}$ , and

$$L_{ij} = \begin{cases} 1/n^2 & i,j \in \mathcal{S} \\ 1/m^2 & i,j \in \mathcal{T} \\ -1/(nm) & \text{otherwise.} \end{cases}$$

Thus the optimization problem Eq.1 can be rewritten as:

$$\begin{aligned} \min_{\boldsymbol{W}} \quad & \operatorname{tr}\left(\boldsymbol{K_W} \boldsymbol{L}\right) \\ \text{s.t.} \quad & \boldsymbol{W}^T \boldsymbol{W} = \boldsymbol{I}_d \end{aligned} \tag{5}$$

## 2.2. Transfer Component Analysis (TCA)

Based on the inputs $\{x_{S_i}\}$ and outputs $\{y_{S_i}\}$ from the source domain, and the inputs $\{x_{T_i}\}$ from the target domain, our task is to predict the unknown outputs $\{y_{T_i}\}$ in the target domain. We attempt to find a common latent representation for both $X_S$ and $X_T$ that preserves the data conguration of the two domains after transformation. Let the desired nonlinear transformation be $\phi : \mathcal{X} \to \mathcal{H}$. Let

$$X'_S = \left\{x'_{S_i}\right\} = \left\{\phi\left(x_{S_i}\right)\right\}, X'_T = \left\{x'_{T_i}\right\} = \left\{\phi\left(x_{T_i}\right)\right\}$$

and $X' = X'_S \cup X'_T$ be the transformed input sets from the source, target and combined domains, respectively. Then, we desire that $\mathcal{P}'\left(X'_S\right) = \mathcal{Q}'\left(X'_T\right)$.

Assume that $\phi$ is the feature map induced by a universal kernel. Using MMD shown in Eq. 2, the distance between the two distributions $\mathcal{P}$ and $\mathcal{Q}$ can be empirically measured by the (squared) distance between the empirical means of the two domains:

$$\text{Dist}\left(X'_S, X'_T\right) = \left\| \frac{1}{n_1} \sum_{i=1}^{n_1} \phi\left(x_{S_i}\right) - \frac{1}{n_2} \sum_{i=1}^{n_2} \phi\left(x_{T_i}\right) \right\|^2_{\mathcal{H}} \tag{6}$$

Assume that $\mathcal{P} \neq \mathcal{Q}$, but $P\left(Y_S \mid \phi\left(X_S\right)\right) = P\left(Y_T \mid \phi\left(X_T\right)\right)$ under a transformation mapping $\phi$ on the input.

Decompose $K$ as $\left(KK^{-1/2}\right)\left(K^{-1/2}K\right)$. $(n_1 + n_2)\times m$ matrix $\widetilde{W}$ transforms the corresponding feature vectors to a $m$-dimensional space. In general, $m \ll n_1 + n_2$. Then the resultant kernel matrix [1] is

$$\widetilde{K} = \left(KK^{-1/2}\widetilde{W}\right)\left(\widetilde{W}^\top K^{-1/2}K\right) = KWW^\top K \tag{7}$$

where $W = K^{-1/2}\widetilde{W} \in \mathbb{R}^{(n_1+n_2)\times m}$. In particular, the corresponding kernel evaluation of $k$ between any two patterns $x_i$ and $x_j$ is given by

$$\widetilde{k}\left(x_i, x_j\right) = k_{x_i}^\top WW^\top k_{x_j} \tag{8}$$

where $k_x = \left[k\left(x_1, x\right), \ldots, k\left(x_{n_1+n_2}, x\right)\right]^\top \in \mathbb{R}^{n_1+n_2}$.

Hence, the kernel $k$ in (6) facilitates a readily parametric form for out-of-sample kernel evaluations.

Moreover, using the definition of $\widetilde{K}$ in (5), the distance between the empirical means of the two domains can be rewritten as

$$\begin{aligned} \text{Dist}\left(X'_S, X'_T\right) &= \text{tr}\left(\left(KWW^\top K\right)L\right) \\ &= \text{tr}\left(W^\top KLKW\right) \end{aligned} \tag{9}$$

The kernel learning problem for domain adaptation then reduces to:

$$\begin{aligned} \min_W \quad & \text{tr}\left(W^\top W\right) + \mu\,\text{tr}\left(W^\top KLKW\right) \\ \text{s.t.} \quad & W^\top KHKW = I \end{aligned} \tag{10}$$

## 2.3. Subspace Alignment (SA)

Rather than working on the original data themselves, SA handles more robust representations of the source and target domains and learns the shift between these two domains. First, transform every source and target data in the form of a $D$-dimensional $z$-normalized vector. Then, select for each domain $d$ eigenvectors corresponding to the $d$ largest eigenvalues using PCA. These eigenvectors are used as bases of the source and target subspaces, respectively denoted by $X_S$ and $X_T$ $\left(X_S, X_T \in \mathbb{R}^{D\times d}\right)$. $X'_S$ and $X'_T$ are orthonormal and $X_S$ and $X_T$ are used to learn the shift between the two domains.

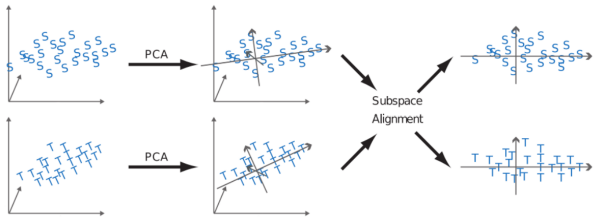



Figure 1: SA

SA suggests to project each source $(\mathbf{y_S})$ and target $(\mathbf{y_T})$ data (where $\mathbf{y_S}, \mathbf{y_T} \in \mathbb{R}^{1\times D}$ ) to its respective subspace $X_S$ and $X_T$. Then, learn a linear transformation function that aligns the source subspace coordinate system to the target one using a subspace alignment approach, so that we can compare source and target samples in their respective subspaces without unnecessary data projections. Align basis vectors by using a transformation matrix $M$ from $X_S$ to $X_T$. $M$ is learned by minimizing the following Bregman matrix divergence:

$$F(M) = \left\|X_S M - X_T\right\|_F^2 \tag{11}$$

$$M^* = \text{argmin}_M(F(M)) \tag{12}$$

where $\|\cdot\|_F^2$ is the Frobenius norm. Then obtain a simple solution of Eq. 12 in closed form. Because the Frobenius norm is invariant to orthonormal operations, we can rewrite Eq. 11 to:

$$F(M) = \left\|X'_S X_S M - X'_S X_T\right\|_F^2 = \left\|M - X'_S X_T\right\|_F^2 \tag{13}$$

In order to compare a source data $\mathbf{y_S}$ with a target data $\mathbf{y_T}$, we need a similarity function $\text{Sim}\left(\mathbf{y_S}, \mathbf{y_T}\right)$ defined as follows:

$$\begin{aligned} \text{Sim}\left(\mathbf{y_S}, \mathbf{y_T}\right) &= \left(\mathbf{y_S} X_S M^*\right)\left(\mathbf{y_T} X_T\right)' = \mathbf{y_S} X_S M^* X'_T \mathbf{y'_T} \\ &= \mathbf{y_S} A \mathbf{y'_T} \end{aligned} \tag{14}$$

where $A = X_S X'_S X_T X'_T$. We use $\text{Sim}\left(\mathbf{y_S}, \mathbf{y_T}\right)$ directly to perform a $k$-nearest neighbor classification task.

## 2.4. Correlation Alignment (CORAL)

The difference between DIP and CORAL is that, DIP aims at aligning first-order information and CORAL

aims at aligning second-order information. More concretely, CORAL aligns the distributions by re-coloring whitened source features with the covariance of the target distribution. The only computations it needs are (1) computing covariance statistics in each domain and (2) applying the whitening and re-coloring linear transformation to the source features.

Suppose we are given source domain training samples $\mathbf{X}_s = \left[\mathbf{x}_1^s, \mathbf{x}_2^s, \ldots, \mathbf{x}_{n_s}^s\right]$, and target data $\mathbf{X}_t = \left[\mathbf{x}_1^t, \mathbf{x}_2^t, \ldots, \mathbf{x}_{n_t}^t\right]$. Suppose $\mathbf{C}_s, \mathbf{C}_t$ are the feature vector covariance matrices. To minimize the distance between the second-order statistics (covariance) of the source and target features, a linear transformation $\mathbf{A}$ is applied to the original source features and the Frobenius norm can be used as the matrix distance metric:

$$\min_{\mathbf{A}} \left\| \hat{\mathbf{C}}_s - \mathbf{C}_t \right\|_F^2 = \min_{\mathbf{A}} \left\| \mathbf{A}^T \mathbf{C}_s \mathbf{A} - \mathbf{C}_t \right\|_F^2 \qquad (15)$$

where $\hat{\mathbf{C}}_s$ is the covariance matrix of $\mathbf{A}^T \mathbf{X}_s$, and $\mathbf{C}_s = \mathbf{U}_s \mathbf{\Sigma}_s \mathbf{U}_s^T, \mathbf{C}_t = \mathbf{U}_t \mathbf{\Sigma}_t \mathbf{U}_t^T$.

The optimal solution of $\mathbf{A}$ is

$$\mathbf{A}^* = \mathbf{U}_s \mathbf{\Sigma}_s^{-\frac{1}{2}} \mathbf{U}_s^T \mathbf{U}_t[1:r] \mathbf{\Sigma}_t[1:r]^{\frac{1}{2}} \mathbf{U}_t[1:r]^T, \qquad (16)$$

where $r = \min\left(\operatorname{rank}\left(\mathbf{C}_s\right), \operatorname{rank}\left(\mathbf{C}_t\right)\right)$. After obtaining $\mathbf{A}^*$, use $\mathbf{A}^{*T}\mathbf{X}_s$ as the source domain training samples and target data remains the same.

## 2.5. Geodesic Flow Kernel (SGF)

SGF attempts to identify *potential* intermediate domains between the source and target and learn the information they convey about domain changes.

Let $X = \{x_i\}_{i=1}^{N_1} \in \mathbb{R}^N$ denote data from the source domain pertaining to $M$ categories or classes and $y_i \in \{1, 2, 3, \ldots M\}$ denote the label of $x_i$. Assume that the source domain is mostly labeled. Further assume that all categories have some labeled data. Let $\tilde{X} = \{\tilde{x}_i\}_{i=1}^{N_2} \in \mathbb{R}^N$ denote unlabeled data from the target domain corresponding to the same $M$ categories. Let $S_1$ and $S_2$ denote generative subspaces of dim$^2$ $N \times d$ obtained by performing principal component analysis PCA on $X$ and $\tilde{X}$ respectively, where $d < N$.

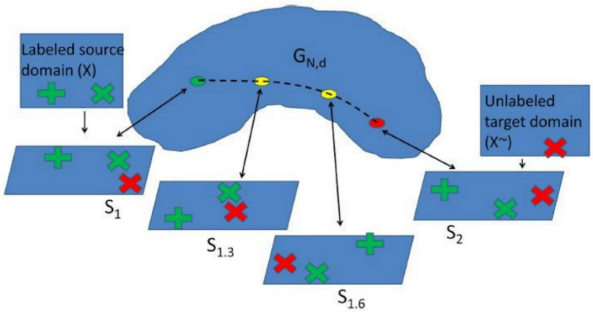



Figure 2: SGF

By viewing $\mathbb{G}_{N,d}$ as a quotient space of $SO(N)$, the geodesic path in $\mathbb{G}_{N,d}$ starting from $S_1$ is given by a one-parameter exponential flow: $\Psi(t') = Q \exp(t'B) J$, where exp refers to the matrix exponential, and $Q \in SO(N)$ such that $Q^T S_1 = J$ and $J = \begin{bmatrix} I_d \\ 0_{N-d,d} \end{bmatrix}$. $I_d$ is a $d \times d$ identity matrix, and $B$ is a skew-symmetric, block-diagonal matrix of the form $B = \begin{pmatrix} 0 & A^T \\ -A & 0 \end{pmatrix}$, $A \in \mathbb{R}^{(N-d)\times d}$, where the superscript $T$ denotes matrix transpose, and the submatrix $A$ specifies the direction and the speed of geodesic flow.

The key problem in SGF is to obtain meaningful intermediate subspaces on the geodesic path between $S_1$ and $S_2$. Compute the direction matrix $A$ using inverse exponential mapping and then use $\Psi(t')$ to obtain intermediate subspaces between $S_1$ and $S_2$. We now model the information conveyed by $S'$ on $X$ and $\tilde{X}$ to perform recognition across domain change. The details are shown in Alg. 1.

**Algorithm 1:** Algorithm for computing the exponential map, and sampling along the geodesic

1 Given a point on the Grassmann manifold $S_1$ and a tangent vector $B = \begin{pmatrix} 0 & A^T \\ -A & 0 \end{pmatrix}$
2 Compute the $N \times N$ orthogonal completion $Q$ of $S_1$
3 Compute the compact SVD of the direction matrix $A = \tilde{V}_2 \Theta V_1$
4 Compute the diagonal matrices $\Gamma(t')$ and $\Sigma(t')$ such that $\gamma_i(t') = \cos(t'\theta_i)$ and $\sigma_i(t') = \sin(t'\theta_i)$, where $\theta'$ s are the diagonal elements of $\Theta$
5 Compute $\Psi(t') = Q \begin{pmatrix} V_1 \Gamma(t') \\ -\tilde{V}_2 \Sigma(t') \end{pmatrix}$, for various values of $t' \in [0, 1]$

## 2.6. Geodesic Flow Kernel (GFK)

The main idea behind GFK is to explicitly construct an infinite-dimensional feature space $\mathcal{H}^\infty$ that assembles information on the source domain $\mathcal{D}_\mathcal{S}$, on the target domain $\mathcal{D}_\mathcal{T}$, and on "phantom"domains interpolating between those two. Inner products in $\mathcal{H}^\infty$ give rise to a kernel function that can be computed efficiently in closed form. Therefore, GFK can be readily used to construct any kernelized classifiers.

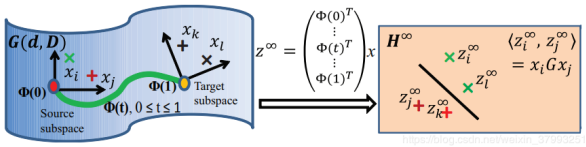



Figure 3: GFK

GFK consists of following 4 steps:

- determine the optimal dimensionality of the subspaces to embedded domains;
- construct the geodesic curve;

- compute the geodesic flow kernel;
- use the kernel to construct a classifier with labeled data.

Let $P_{\mathcal{S}}, P_{\mathcal{T}} \in \mathbb{R}^{\mathrm{D}\times\mathrm{d}}$ denote the two sets of basis of the subspaces for the source and target domains. Let $\boldsymbol{R}_{\mathcal{S}} \in \mathbb{R}^{\mathrm{D}\times(\mathrm{D}-\mathrm{d})}$ denote the orthogonal complement to $\boldsymbol{P}_{\mathcal{S}}$, $\boldsymbol{R}_{\mathcal{S}}^{\mathrm{T}}\boldsymbol{P}_{\mathcal{S}} = \mathbf{0}$. Using the canonical Euclidean metric for the Riemannian manifold, the geodesic flow is parameterized as $\boldsymbol{\Phi} : t \in [0,1] \rightarrow \boldsymbol{\Phi}(t) \in G(\mathrm{d},\mathrm{D})$ under the constraints $\boldsymbol{\Phi}(0) = \boldsymbol{P}_{\mathcal{S}}$ and $\boldsymbol{\Phi}(1) = \boldsymbol{P}_{\mathcal{T}}$. For other $t$

$$\boldsymbol{\Phi}(t) = \boldsymbol{P}_{\mathcal{S}}\boldsymbol{U}_1\boldsymbol{\Gamma}(t) - \boldsymbol{R}_{\mathcal{S}}\boldsymbol{U}_2\boldsymbol{\Sigma}(t) \tag{17}$$

where $\boldsymbol{U}_1 \in \mathbb{R}^{\mathrm{d}\times\mathrm{d}}$ and $\boldsymbol{U}_2 \in \mathbb{R}^{(\mathrm{D}-\mathrm{d})\times\mathrm{d}}$ are orthonormal matrices.

For two original $D$-dimensional feature vectors $\boldsymbol{x}_i$ and$\boldsymbol{x}_j$, we compute their projections into $\boldsymbol{\Phi}(t)$ for a continuous $t$ from 0 to 1 and concatenate all the projections into infinite-dimensional feature vectors $\boldsymbol{z}_i^{\infty}$ and $\boldsymbol{z}_j^{\infty}$. The inner product between them defines our geodesic-flow kernel

$$\left\langle \boldsymbol{z}_i^{\infty}, \boldsymbol{z}_j^{\infty} \right\rangle = \int_0^1 \left(\boldsymbol{\Phi}(t)^{\mathrm{T}}\boldsymbol{x}_i\right)^{\mathrm{T}} \left(\boldsymbol{\Phi}(t)^{\mathrm{T}}\boldsymbol{x}_j\right) dt = \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{G}\boldsymbol{x}_j \tag{18}$$

where $G \in \mathbb{R}^{\mathrm{D}\times\mathrm{D}}$ is a positive semidefinite matrix and can be computed in a closed-form from previously defined matrices:

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{P}_{\mathcal{S}}\boldsymbol{U}_1 & \boldsymbol{R}_{\mathcal{S}}\boldsymbol{U}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{\Lambda}_1 & \boldsymbol{\Lambda}_2 \\ \boldsymbol{\Lambda}_2 & \boldsymbol{\Lambda}_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{U}_1^{\mathrm{T}}\boldsymbol{P}_{\mathcal{S}}^{\mathrm{T}} \\ \boldsymbol{U}_2^{\mathrm{T}}\boldsymbol{R}_{\mathcal{S}}^{\mathrm{T}} \end{bmatrix} \tag{19}$$

where $\boldsymbol{\Lambda}_1$ to $\boldsymbol{\Lambda}_3$ are diagonal matrices, whose diagonal elements are

$$\lambda_{1i} = 1 + \frac{\sin(2\theta_i)}{2\theta_i}, \lambda_{2i} = \frac{\cos(2\theta_i) - 1}{2\theta_i}, \lambda_{3i} = 1 - \frac{\sin(2\theta_i)}{2\theta_i} \tag{20}$$

### 2.7. Kernel Mean Matching (KMM)

KMM utilizes the availability of unlabeled data to direct a sample selection de-biasing procedure for various learning methods. It accounts for the differences between source domain and target domain by reweighting the source domain samples such that the means of the training and test points in a reproducing kernel Hilbert space (RKHS) are close. The intuitive effect of KMM is shown in the Figure 4.

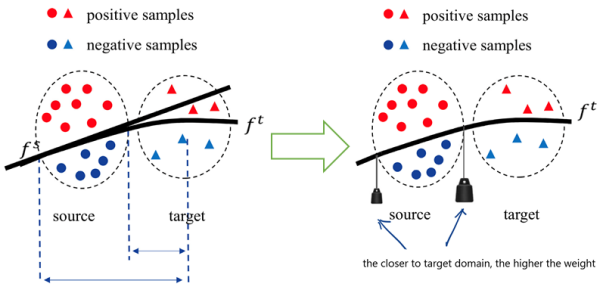



Figure 4: KMM

Suppose we are given source domain training samples $\mathbf{X}_s = \left[\mathbf{x}_1^s, \mathbf{x}_2^s, \ldots, \mathbf{x}_{n_s}^s\right]$, and target data $\mathbf{X}_t = \left[\mathbf{x}_1^t, \mathbf{x}_2^t, \ldots, \mathbf{x}_{n_t}^t\right]$, the reweighting process can be defined as:

$$d\left(\mathbf{X}_s, \mathbf{X}_t\right)^2 = \left\| \frac{1}{n_s}\sum_{i=1}^{n_s}\phi\left(\mathbf{x}_i^s\right) - \frac{1}{n_t}\sum_{i=1}^{n_t}\phi\left(\mathbf{x}_i^t\right) \right\|^2 \tag{21}$$

1) Step 1:

$$\min_{\beta_i} \| \frac{1}{n_s}\sum_{i=1}^{n_s}\beta_i\phi\left(\mathbf{x}_i^s\right) - \frac{1}{n_t}\sum_{i=1}^{n_t}\phi\left(\mathbf{x}_i^t\right) \|^2 \tag{22}$$

2) Step 2:

$$\begin{array}{ll} \min_{\mathbf{w},b,\xi_i} & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i^1 \bar{\beta}_i^k\xi_i \\ \text{s.t.} & y_i\left(\mathbf{w}^T\phi\left(\mathbf{x}_i\right) + b\right) \geq 1 - \xi_i, \quad \forall i \\ & \xi_i \geq 0, \quad \forall i \end{array} \tag{23}$$

## 3. Deep Learning

### 3.1. Deep Adaptation Networks (DAN)

DAN is a domain transfer learning method which combines the deep learning and traditional ideas and Figure 5 shows the architecture of DAN. As we can see, DAN not only needs to minimize the classification loss based on the source domain training data, but minimizes the distance between the feature of source domain data and target domain data by using MK-MMD loss.

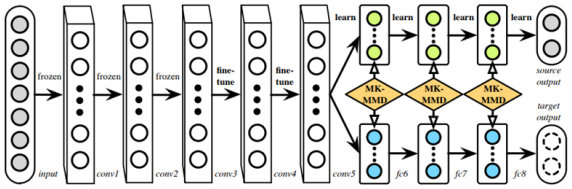



Figure 5: Deep Adaptation Networks

The multiple kernel variant of maximum mean discrepancies (MK-MMD) between probability distribution $p$ and $q$ is the reproducing kernel Hilbert space (RKHS) distance between the mean embedding of $p$ and $q$. It can be formalized as Equation 24. In our cases, $\phi$ can be a feature map.

$$d_k^2(p,q) = \left\| E_p[\phi(x^s)] - E_q[\phi(x^t)] \right\|_{\mathcal{H}_k} \tag{24}$$

Therefore, the loss function of DAN can be formalized as Equation 25. The first term refers to classification loss while the second term is the MK-MMD loss. $D_*^l$ is the $l^{th}$ layer hidden representation for the source and target examples. $\lambda > 0$ is a penalty term, while the layers between $l_1$ and $l_2$ are affected by the regularizer.

$$\min_{\theta} \frac{1}{n_a}\sum_{i=1}^{n_a} J(\theta(x_i^a), y_i^a) + \lambda\sum_{l=l_1}^{l_2} d_k^2(D_s^l, D_t^l) \tag{25}$$

### 3.2. Deep Coral

Deep Coral is another domain adaptation method which combines deep learning and traditional ideas. Coral is traditional domain adaptation method, whose motivation is aligning the covariance matrix between source samples and target samples. Deep coral draws on the idea of coral and try to align the second order information between feature maps of source samples and target samples. The architecture of Deep Coral is shown in Figure 6.

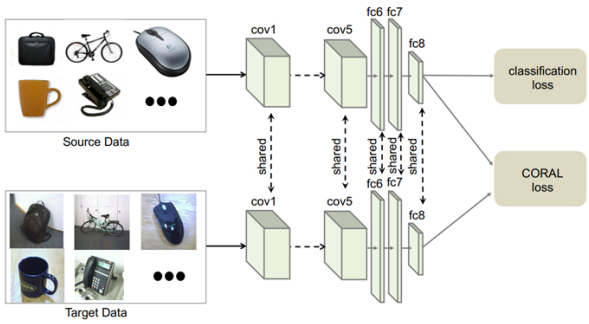



Figure 6: Deep Coral

Therefore, firstly, Deep Coral defines CORAL loss which describes the distance of the second order information between the source domain and target domain in a certain feature map. Suppose $C_S$ and $C_T$ are the covariance matrices of source domain feature maps and target domain feature maps. CORAL loss can be formalized as Equation 26, where $\|X\|_F^2$ denotes the squared matrix Frobenius norm.

$$l_{CORAL} = \frac{1}{4d^2} \|C_S - C_T\|_F^2 \tag{26}$$

Suppose the feature map of source domain training samples is $D_S$, while the feature map of target domain unlabeled samples is $D_T$. $n_S$ and $n_T$ denotes the number of source data and target data samples. The covariance matrices can be further formalized by Equation 27.

$$\begin{aligned} C_S &= \frac{1}{n_S - 1}(D_S^T D_S - \frac{1}{n_S}(\mathbf{1}^T D_S^T)(\mathbf{1}^T D_S)) \\ C_T &= \frac{1}{n_T - 1}(D_T^T D_T - \frac{1}{n_T}(\mathbf{1}^T D_T^T)(\mathbf{1}^T D_T)) \end{aligned} \tag{27}$$

As we mentioned above, Deep Coral needs to minimize the classification loss of source domain samples and the Coral loss between the feature maps of source domain and target domain, thus, the total loss function can be formalized as Equation 28, where $\lambda$ is a weight that trades off the classification performance and the ability of adaptation. Too small $\lambda$ will lead to the overfitting of source domain and poor performance on target domain while too large $\lambda$ will project both source domain and target domain data into a single point.

$$l = l_{CLASS} + \sum_{i=1}^{t} \lambda_i l_{CORAL} \tag{28}$$

### 3.3. Domain Adversarial Neural Network (DANN)

DANN is a domain adaptation learning algorithm which applies the idea of adversarial learning. The basic idea of DANN is to train a domain classification model as well as a label classification model. As Figure 7 shows, the green part is the feature extractor, the blue part is the label classifier, and the red part is a domain classifier. The goal of DANN is to train a powerful feature extractor which can not only do label classification but also fool the domain classification and make it can't distinguish the image from source domain and target domain. Therefore, there are two terms in loss function, which are label prediction loss and domain classification loss. And three parts of the model have different optimization goals. Specifically, the feature extractor wants to minimize the label classification loss and maximize the domain classification loss, the domain classifier wants to minimize the domain classification loss while the label prediction part wants to minimize the label prediction loss. Therefore, DANN designs a gradient reversal layer shown in Figure 7 to meet the require of different optimization direction between feature extractor part and the domain classification part.

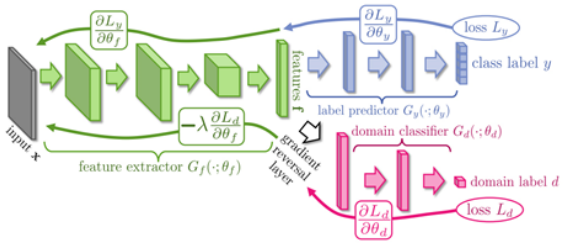



Figure 7: Deep Adversarial Neural Network

### 3.4. Conditional Domain Adversarial Adaptation (CDAN)

Adversarial learning is widely used in domain adaptation area. The basic idea of adversarial learning is to learn a discriminator to separate the data from source domain and target domain, while the feature extractor tries to fool the discriminator, which is equivalent to make the feature maps of source data and target data undistinguished. Therefore, it is actually a minimax problem.

However, CDAN claims that there are three problems in modern adversarial domain adaptation methods as follow:

1) It's insufficient for us to align the feature maps without aligning the labels.
2) Discriminator will be fooled and can't distinguish the data domain even after converging if the data distribution is a complex multi-modal structure.
3) Assign the same weight to different samples for discriminator may cause some problems due to some samples are difficult to transfer.

Regarding the issues mentioned above, CDAN firstly proposes that feeding the information combined with the

feature map and the label to the discriminator as Figure 8 shows. And there are two competitive terms in the minimax problem of CDAN:

$$\mathcal{E}(G) = \mathbb{E}_{(\mathbf{x}_i^s, \mathbf{y}_i^s) \sim \mathcal{D}_s} L\left(G\left(\mathbf{x}_i^s\right), \mathbf{y}_i^s\right) \tag{29}$$

$$\begin{aligned} \mathcal{E}(D, G) = & -\mathbb{E}_{\mathbf{x}_i^s \sim \mathcal{D}_s} \log\left[D\left(\mathbf{f}_i^s, \mathbf{g}_i^s\right)\right] \\ & - \mathbb{E}_{\mathbf{x}_j^t \sim \mathcal{D}_t} \log\left[1 - D\left(\mathbf{f}_j^t, \mathbf{g}_j^t\right)\right] \end{aligned} \tag{30}$$

where $G$ is the source classifier, $D$ is the domain discriminator, $f$ is the feature map and $g$ is the label prediction. The minimax game of CDAN is

$$\begin{aligned} & \min_G \mathcal{E}(G) - \lambda \mathcal{E}(D, G) \\ & \min_D \mathcal{E}(D, G) \end{aligned} \tag{31}$$

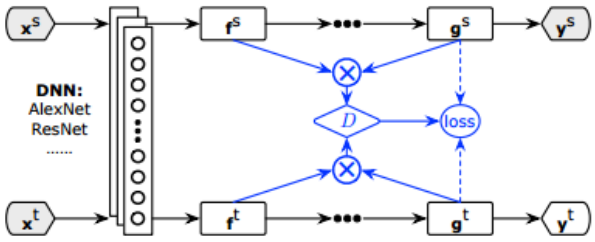



Figure 8: Conditional Domain Adversarial Adaptation

And then, CDAN uses the multilinear map $f \bigotimes g$ instead of linear map $f \bigoplus g$ to deal with complex data distribution, as multilinear map enables manipulation of the multiplicative interaction across multiple random variables. Therefore, we can reformalized $\mathcal{E}(D, G)$ with:

$$\begin{aligned} \mathcal{E}(D, G) = & -\mathbb{E}_{\mathbf{x}_i^s \sim \mathcal{D}_s} \log\left[D\left(T(\mathbf{h}_i^s)\right)\right] \\ & - \mathbb{E}_{\mathbf{x}_j^t \sim \mathcal{D}_t} \log\left[1 - D\left(T(\mathbf{h}_i^s)\right)\right] \end{aligned} \tag{32}$$

where $\mathbf{h} = (\mathbf{f}, \mathbf{g})$, and $T(\mathbf{g}) = \mathbf{f} \bigotimes \mathbf{g}$.

Moreover, CDAN uses entropy

$$H(g) = -\sum_{c=1}^{C} g_c log(g_c)$$

to represent the uncertainty of prediction, which can be used to adjust the weight between different samples. Therefore, finally, the minimax optimization goal of CDAN can be formalized as Equation 33.

$$\begin{aligned} & \min_G \mathbb{E}_{(\mathbf{x}_i^s, \mathbf{y}_i^s) \sim \mathcal{D}_s} L\left(G\left(\mathbf{x}_i^s\right), \mathbf{y}_i^s\right) \\ & + \lambda \mathbb{E}_{\mathbf{x}_i^s \sim \mathcal{D}_s} w\left(H\left(\mathbf{g}_i^s\right)\right) \log\left[D\left(T\left(\mathbf{h}_i^s\right)\right)\right] \\ & + \mathbb{E}_{\mathbf{x}_j^t \sim \mathcal{D}_t} w\left(H\left(\mathbf{g}_j^t\right)\right) \log\left[1 - D\left(T\left(\mathbf{h}_j^t\right)\right)\right] \\ & \max_D \mathbb{E}_{\mathbf{x}_i^s \sim \mathcal{D}_s} w\left(H\left(\mathbf{g}_i^s\right)\right) \log\left[D\left(T\left(\mathbf{h}_i^s\right)\right)\right] \\ & + \mathbb{E}_{\mathbf{x}_j^t \sim \mathcal{D}_t} w\left(H\left(\mathbf{g}_j^t\right)\right) \log\left[1 - D\left(T\left(\mathbf{h}_j^t\right)\right)\right] \end{aligned} \tag{33}$$

# 4. Experiments and Results

In this section, we explain our experimental process and detail the methods we adopted. We also show the results of different methods and make some analyses and comparisons.

## 4.1. Dataset

In this project, we use *Office-Home dataset* [1] as our domain adaptation dataset, which consists of 65 categories of office depot from four domains: Artistic images, Clip Art, Product images, and Real-World images. We conduct domain adaptation experiments using several algorithms in the following three settings: Art to Real-world (A $\rightarrow$ R), Clipart to Real-world (C $\rightarrow$ R), and Product to Real-world (P $\rightarrow$ R).

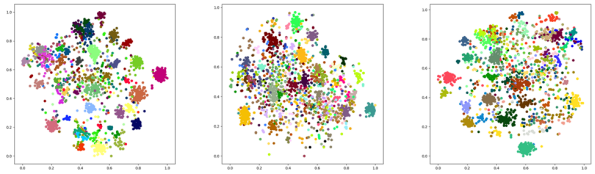
## 4.2. Baseline

In order to observe the effect of domain adaptation on image classification better, we first perform SVM on the dataset without domain adaptation, with source domain as the training set and target domain as the testing set. We tune the hyperparameter $C$ to 0.0001, 0.001, 0.01, 0.1 and 1, and the results are shown in Table 1.

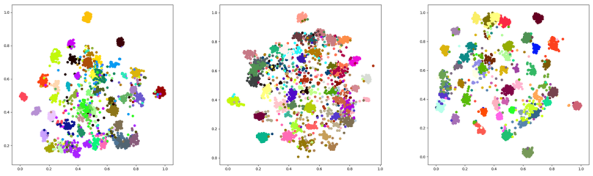
Table 1: Performance of Baseline

| | **A→R** | **C→R** | **P→R** |
|---|---|---|---|
| C=0.0001 | 43.18% | 52.64% | 63.66% |
| C=0.001 | 65.54% | 64.39% | 73.30% |
| C=0.01 | **74.59%** | **66.92%** | **74.10%** |
| C=0.1 | 74.47% | 65.08% | 71.81% |
| C=1 | 72.89% | 61.98% | 70.29% |

We also perform t-SNE on the dataset without domain adaptation to visualize the data distribution. The results are shown in Figure 9.



(a) source domain (from left to right: A $\rightarrow$ R, C $\rightarrow$ R, P $\rightarrow$ R)



(b) target domain (from left to right: A $\rightarrow$ R, C $\rightarrow$ R, P $\rightarrow$ R)

Figure 9: The visualization of the original data distribution (from left to right: A $\rightarrow$ R, C $\rightarrow$ R, P $\rightarrow$ R).

## 4.3. Traditional Methods

**4.3.1. DIP.** In this experiment, we set the $d$ of $D \times d$ projected matrix $\boldsymbol{W}$ to 200 and 1000 so that features can

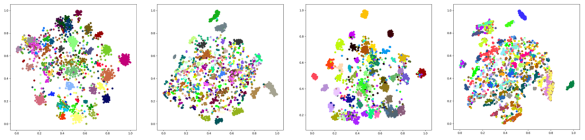
be projected into 200-dimensional and 1000-dimensional subspaces respectively. To test the performance of DIP, we choose linear SVM and train the classifier on the obtained projected source domain and test the classification accuracy on the projected target domain. We tune the hyperparameter $C$ of SVM to 0.0001, 0.001, 0.01, 0.1 and 1 in order to achieve the best accuracy. The results are shown in Table 2.

From the results, we can see that it is not the higher the dimension of the subspace the better the classification performance. The SVM results shows that when the source domain is Art, the best accuracy appears in 1000-dim when $C = 1$. When the source domain is Clipart, the best accuracy appears in 1000-dim when $C = 0.1$. When the source domain is Product, the best accuracy appears in 200-dim when $C = 0.01$. The best accuracy appears in higher dimension is understandable, for the higher dimension saves more information, which is helpful to improve the ability of the classifier. However, since the source and target domains are different, the distance between the source and target domains may not be necessarily smaller if dimension is higher. In this case, the best accuracy may appear in lower dimension.

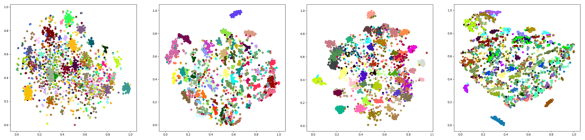
In order to observe the impact of domain adaptation on data distribution more intuitively, we use t-SNE to visualize the projected source and target domains, and the dimension of the subspace is the the one with the highest classification accuracy. The visualization results are shown in Figure 10. The first row is the source domain is Art and the target domain is Real-world, the second row is the source is Clipart and the target is Real-world, and the last row is the source is Product and the target in Real-world. The first column is the visualization of the original source domain, the second column is DIP source domain, the third column in the original target domain, and the last column is DIP target domain. It can be seen that after domain adaptation, the distributions of the source and target domains are closer than the original. However, the classification performance is not better than baseline, which may be because the projection process is projecting the original data into a lower dimensional subspace, which may lose some more detailed information.

Table 2: Experiment results of DIP

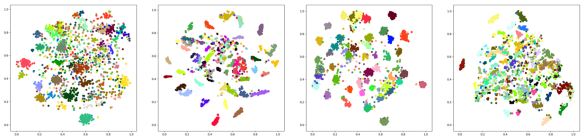
| | **A→R** | **C→R** | **P→R** |
|---|---|---|---|
| **dim=200** | | | |
| C=0.001 | 31.70% | 55.96% | 65.83% |
| C=0.01 | 56.90% | 64.61% | **73.44%** |
| C=0.1 | 68.58% | 65.85% | 73.12% |
| C=1 | 70.78% | 62.84% | 70.92% |
| **dim=1000** | | | |
| C=0.001 | 35.39% | 53.98% | 60.52% |
| C=0.01 | 61.76% | 64.54% | 69.08% |
| C=0.1 | 70.37% | **66.05%** | 70.97% |
| C=1 | **72.07%** | 64.24% | 69.38% |



(a) A → R



(b) C → R



(c) P → R

Figure 10: The visualization of the DIP transformed data distribution (from left to right: original source domain, DIP source domain, original target domain, DIP target domain).

**4.3.2. TCA.** In this experiment, we get kernel from source domain and target domain data, then based on the kernel data, we train the $D \times d$ projected matrix $\boldsymbol{W}$ where $d = 200$ and $d = 1000$. With $\boldsymbol{W}$, according to Eq. 7, we get $\widetilde{K}$ and do Z-score normalization.

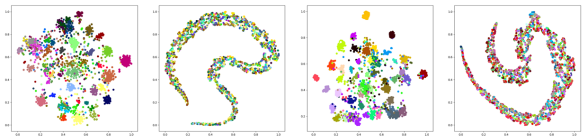
We choose linear SVM to evaluate the efficiency of TCA domain adaptation. The results are shown in Table 3. We find that TCA performs better when $d = 1000$. Observe the best accuracy of these 3 different source domains, we find that when Product domain is source domain and Real-World domain is target domain, SVM works best; when Cilpart domain is source domain, SVM works worst and this confirms that Product domain is the most closest domain to Real-World domain.

In order to observe the impact of domain adaptation on data distribution more intuitively, we use t-SNE to visualize the projected source and target domains, and the dimension of the subspace is the the one with the highest classification accuracy. The visualization results are shown in Figure 11. The result looks different from others. We think maybe it is because we get a high dimensional kernel first and when we project the data into a low dimensional subspace, it may lose much information and lead to special shapes when we do visualization. From Table 13, we find that the result of TCA is not as good as other methods. Nor the visualization result looks well neither.

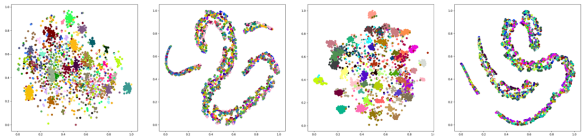
**4.3.3. SA.** In this experiment, we use linear SVM to do the classification and the results are shown in Table 4. It shows that when the source domain is Product the best accuracy appears at 200-dim, and the other two domain adaptation settings appear at 1000-dim. The reason is mentioned as above. We can see that the performance of SA is better than both DIP and TCA, but does not exceed the baseline.

Table 3: Experiment results of TCA

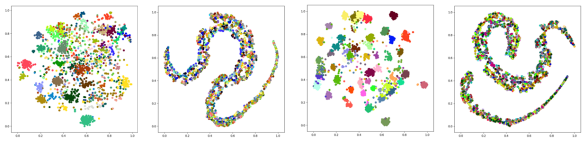
| | A→R | C→R | P→R |
|---|---|---|---|
| **dim=200** | | | |
| C=0.0001 | 12.52% | 14.14% | 22.91% |
| C=0.001 | 29.73% | 35.13% | 42.46% |
| C=0.01 | 45.93% | 42.87% | 58.96% |
| C=0.1 | 61.58% | 57.08% | 62.58% |
| C=1 | 46.77% | 41.93% | 53.29% |
| **dim=1000** | | | |
| C=0.0001 | 12.78% | 14.41% | 23.34% |
| C=0.001 | 29.85% | 35.71% | 49.41% |
| C=0.01 | 47.48% | 53.19% | 60.76% |
| C=0.1 | **62.24%** | **61.50%** | **67.07%** |
| C=1 | 49.93% | 51.53% | 56.03% |



(a) A → R



(b) C → R

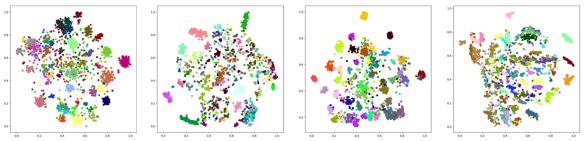


(c) P → R

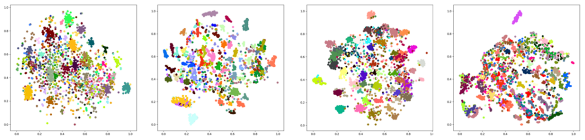
Figure 11: The visualization of the TCA transformed data distribution (from left to right: original source domain, TCA source domain, original target domain, TCA target domain).

Table 4: Experiment results of SA

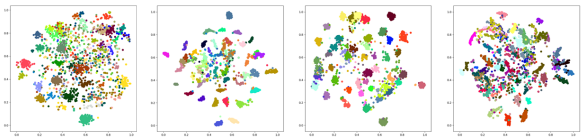
| | A→R | C→R | P→R |
|---|---|---|---|
| **dim=200** | | | |
| C=0.0001 | 19.23% | 35.56% | 49.21% |
| C=0.001 | 41.50% | 56.85% | 65.99% |
| C=0.01 | 64.91% | 65.30% | **73.67%** |
| C=0.1 | 73.33% | 66.33% | 73.35% |
| C=1 | 74.27% | 63.19% | 71.45% |
| **dim=1000** | | | |
| C=0.0001 | 18.09% | 33.88% | 47.12% |
| C=0.001 | 37.85% | 53.38% | 64.86% |
| C=0.01 | 64.43% | 64.70% | 73.24% |
| C=0.1 | 72.71% | **66.49%** | 73.08% |
| C=1 | **74.61%** | 64.13% | 71.56% |



(a) A → R



(b) C → R



(c) P → R

Figure 12: The visualization of the SA transformed data distribution (from left to right: original source domain, SA source domain, original target domain, SA target domain).

We also use t-SNE to visualize the data distribution of source and target domains and the results are shown in Figure 12. It shows that the data distribution of source and target domains becomes more similar after domain adaptation.

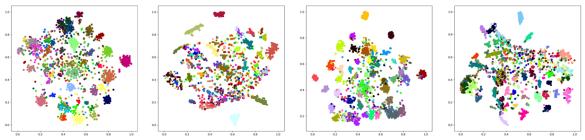
**4.3.4. CORAL.** In this experiment, we first use CORAL to fit the source domain data, and then train the SVM classifier on it. Then we feed the target domain data into the classifier to see the performance. The results are shown in Table 5. It shows that when the source domain is **A**, the best accuracy is 72.98% in 1000-dim when $C = 1$. When the source domain is **C**, the best accuracy is 67.44% in 1000-dim when $C = 0.1$. When the source domain is **P**, the best accuracy is 74.03% in 1000-dim when $C = 0.01$. The reason why low dimension can achieve better accuracy is same as above.

From the results, we can see that the performance of CORAL is better than DIP. This is because CORAL uses second-order information so it can express more concretely. However, the accuracy still does not exceed the baseline. We also use t-SNE to visualize the data distribution of source and target domains and the results are shown in Figure 13. It shows that the data distribution of source and target domains becomes more similar after domain adaptation.

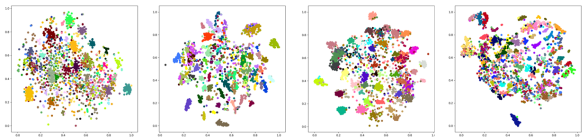
**4.3.5. SGF.** Based on the previous experiments, we set the dimension of the feature subspaces as 1000, and the hyperparameter $C$ as 1 in order to save time. The results are shown in Table 4. It shows that when the source domain is **A**, the best accuracy is 70.28%. When the source domain is **C**, the best accuracy is 66.72% in 1000-dim when $C = 0.1$. When the source domain is **P**, the best accuracy is 71.80%. From the results, we can see that the accuracy still does not exceed the baseline. We also use t-SNE to visualize the data distribution of source and target domains and the results

Table 5: Experiment results of CORAL

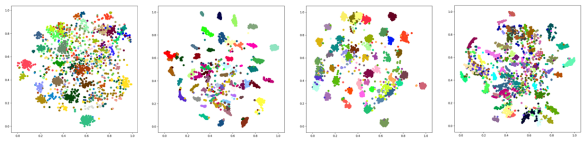
| | A→R | C→R | P→R |
|---|---|---|---|
| **dim=200** | | | |
| C=0.0001 | 15.72% | 33.27% | 48.49% |
| C=0.001 | 33.42% | 58.75% | 67.92% |
| C=0.01 | 58.66% | 66.17% | **74.03%** |
| C=0.1 | 69.92% | 66.59% | 73.98% |
| C=1 | 71.03% | 64.40% | 71.44% |
| **dim=1000** | | | |
| C=0.0001 | 17.82% | 33.35% | 46.74% |
| C=0.001 | 37.67% | 60.56% | 69.37% |
| C=0.01 | 64.73% | 65.81% | 71.69% |
| C=0.1 | 71.01% | **67.44%** | 71.99% |
| C=1 | **72.98%** | 65.87% | 70.28% |



(a) A → R



(b) C → R



(c) P → R

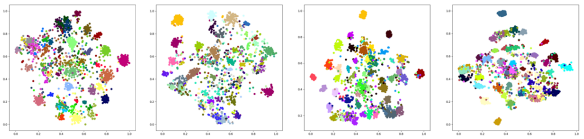
Figure 13: The visualization of the CORAL transformed data distribution (from left to right: original source domain, CORAL source domain, original target domain, CORAL target domain).

are shown in Figure 12. It shows that the data distribution of source and target domains becomes more similar after domain adaptation.

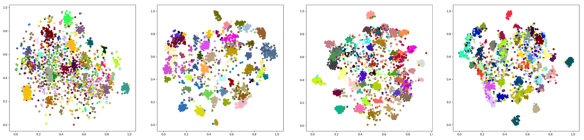
Table 6: Experiment results of SGF

| | A→R | C→R | P→R |
|---|---|---|---|
| **Accuracy** | 70.28% | 66.72% | 71.80% |

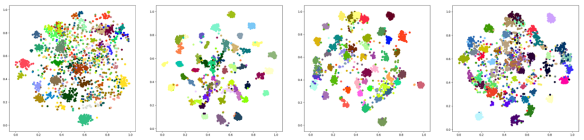
**4.3.6. GFK.** In this experiment, we obtain the domain adaptation data by GFK and then feed them into the linear SVM to do classification. In GFK, we first train a $G$ and calculate $\hat{\mathbf{X}}_s$ and $\hat{\mathbf{X}}_t$. We also set dimension as 200 and 1000, and tune hyperparameter $C$. The results are shown in Table 7. It shows that **A** → **R** reaches its best accuracy



(a) A → R



(b) C → R



(c) P → R

Figure 14: The visualization of the SGF transformed data distribution (from left to right: original source domain, SGF source domain, original target domain, SGF target domain).

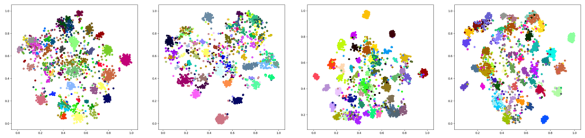
when the dimension is 1000, **C** → **R** is 200, and **P** → **R** is 1000. Only the accuracy of **C** → **R** can exceed the baseline.

We also use t-SNE to do visualization, and the visualized data distribution of the source and target domains are shown in Figure 15. It shows that the data distribution of source and target domains becomes more similar after domain adaptation.

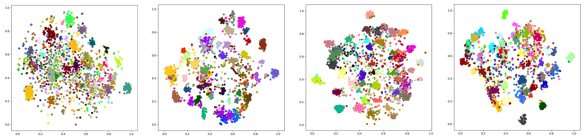
Table 7: Experiment results of GFK

| | A→R | C→R | P→R |
|---|---|---|---|
| **dim=200** | | | |
| C=0.0001 | 62.93% | 62.20% | 70.80% |
| C=0.001 | 68.23% | 64.36% | 72.20% |
| C=0.01 | 72.37% | **72.32%** | 70.18% |
| C=0.1 | 63.92% | 64.59% | 50.03% |
| C=1 | 57.03% | 61.40% | 42.92% |
| **dim=1000** | | | |
| C=0.0001 | 62.82% | 62.18% | 70.74% |
| C=0.001 | 68.67% | 64.56% | **72.36%** |
| C=0.01 | **72.73%** | 62.80% | 71.88% |
| C=0.1 | 65.41% | 51.09% | 51.34% |
| C=1 | 58.37% | 46.89% | 46.29% |

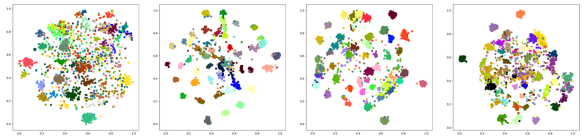
**4.3.7. KMM.** In this experiment, we first use KMM to search the weight for each source domain sample and then use the obtained weights to adjust the SVM classifier on target domain. During the SVM process, we found the performance of classification is poor when the hyperparameter $C < 1$, so we tune $C$ into a series of values based on $2^3$ and found that the performance becomes better when $C = 512$. The results are shown in Table 8. It shows that **A** → **R** reaches the accuracy as $74.38\%$, **C** → **R** reaches



(a) A → R



(b) C → R



(c) P → R

Figure 15: The visualization of the GFK transformed data distribution (from left to right: original source domain, GFK source domain, original target domain, GFK target domain).

65.22%, and **P** → **R** reaches 73.85%. None of them exceed the baseline.

Table 8: Experiment results of KMM

| | A→R | C→R | P→R |
|---|---|---|---|
| **Accuracy** | 74.38% | 65.22% | 73.85% |

**4.3.8. Summary.** In this experiment, we try 7 traditional domain adaptation methods in total, and apply the obtained results into SVM to do classification. The results shows that the classification results of the data obtained by most traditional methods are not better than the baseline. We think this may be because the projection process may lost some detailed information due to the dimensionality reduction. Also, traditional methods firstly extract deep learning features and then apply transfer learning, which makes two related tasks separated. To improve this, we try several deep learning methods.

## 4.4. Deep Learning

**4.4.1. DAN.** In thin experiment, we finetune the pre-trained ResNet50 by SGD with momentum of 0.9, weight decay of $5e-4$, initial learning rate of 0.01, batch size of 32, and iteration of 10000. In the training process, we decay the learning rate after each iteration, which can allow us to quickly find the good solution in the initial stage and carefully update the parameters later, and Equation 34 shows the update equation. The penalty term is increasing after each iteration followed the Equation 35, which allows the model to firstly learn a powerful extractor and then align the source domain features and target domain features.The results are shown in Table 9.

$$lr = \frac{init\_lr}{(1 + 10 * \frac{iteration+1}{total\_iteration})^{0.75}} \tag{34}$$

$$\lambda = \frac{2}{1 + \exp^{-10 * \frac{iteration}{total\_iteration}}} - 1 \tag{35}$$

Table 9: Experiment results of DAN

| | A→R | C→R | P→R |
|---|---|---|---|
| **Accuracy** | 76.06% | 68.02% | 74.97% |

We can see that DAN outperforms all traditional methods we tried. We assumed that the traditional methods firstly extract deep learning feature and then apply transfer learning, which separated two related tasks, while DAN incorporates the MK-MMD loss into basic classification model which aligns the feature maps of source domain and target domain to build an end-to-end model, and this allows the model to finetune the feature extractor to get better adaptation ability.

**4.4.2. Deep Coral.** In the experiment, we use pre-trained ResNet50 as our backbone, after which, we add an additonal bottleneck layer and a classifier layer. We train the model by SGD with momentum of 0.9, learning rate of $1e-3$ for backbone layer, $1e-2$ for bottleneck layer and classifier layer, batch size of 32, epoch of 100, and $\lambda$ of 10. The results are shown in Table 10.

Table 10: Experiment results of Deep Coral

| | A→R | C→R | P→R |
|---|---|---|---|
| **Accuracy** | 76.17% | 68.22% | 75.52% |

We can see that the performance of Deep Coral is much better than Coral, and is comparable to DAN. Therefore, similar to DAN, we can conclude that incorporate the classification loss with CORAL loss that aligns the second order information can guide the model to finetune the feature extractor to get better adaptability. Combining the results of DAN and Deep Coral, we can conclude a general idea to do domain adaptation, which is combining the classification loss with another loss that aligns the feature maps between source domain and target domain.

**4.4.3. DANN.** In the experiment, we finetune the pre-trained ResNet50 by SGD with momentum of 0.9, weight decay of $5e-4$, initial learning rate of 0.001, batch size of 32, and iteration of 10000. And the results are shown in Table 11.

Table 11: Experiment results of DANN

| | A→R | C→R | P→R |
|---|---|---|---|
| **Accuracy** | 75.96% | 68.10% | 75.24% |

We can find that the performance of DANN is comparable to Deep Coral and DAN, which indicates that the idea of adversarial learning works for domain adaptation learning, which provides another promising ideas for further study on domain adaptation areas.

**4.4.4. CDAN.** In this experiment, we finetune the pretrained ResNet50 by SGD with momentum of $0.9$, weight decay of $5e-4$, learning rate of $0.001$, batch size of 36, and iteration of 20000. The results are shown in Table 12.

Table 12: Experiment results of CDAN

| | A→R | C→R | P→R |
|---|---|---|---|
| **Accuracy** | 76.97% | 71.15% | 77.46% |

The performance of CDAN is better than DANN, which shows that aligning the label prediction of source domain and target domain is meaningful. Moreover, the result of task $C \rightarrow R$ of CDAN is much better than which of DANN, which possibly due to the multilinear map that can better deal with the complex task. The result also shows that assigning different weights for different examples is beneficial to the training of adversarial learning.

## 5. Conclusions

In this project, we try 7 traditional methods and 4 deep learning methods to perform domain adaptation on *Office-Home dataset*. The methods we use are **DIP, TCA, SA, CORAL, SGF, GFK, KMM** for traditional methods and **DAN, Deep Coral, CDAN** for deep learning methods. The best performance of them in the three domain adaptation settings are shown in Table 13. We can see that most of the traditional methods don't get better performance than baseline, it might because our dataset is very complex, and the traditional methods are not powerful enough to do adaptation. Besides, the largest drawback of traditional methods is separating two related tasks, that is feature extractor and the alignment of source and target domains. The deep learning methods based on traditional ideas such as DAN and Deep Coral are designed to deal with that problem, which build end-to-end models to learn a feature extractor that not only correctly classifies the source domain images but aligns the feature maps between source domain and target domain. Furthermore, models using the adversarial learning ideas such as CDAN and DANN are even more efficient, and CDAN performs best on domain adaptation tasks in our experiment.

Table 13: The best performance of all methods in the three settings.

| | A→R | C→R | P→R |
|---|---|---|---|
| Baseline | 74.59% | 66.92% | 74.10% |
| **Traditional** | | | |
| DIP | 72.07% | 66.05% | 73.44% |
| TCA | 62.24% | 61.50% | 67.07% |
| SA | 74.61% | 66.49% | 73.67% |
| CORAL | 72.98% | 67.44% | 74.03% |
| SGF | 70.28% | 66.72% | 71.80% |
| GFK | 72.73% | 72.32% | 72.36% |
| KMM | 74.38% | 65.22% | 73.85% |
| **Deep Learning** | | | |
| DAN | 76.06% | 68.02% | 74.97% |
| Deep Coral | 76.17% | 68.22% | 75.52% |
| DANN | 75.96% | 68.10% | 75.24% |
| CDAN | 76.97% | 71.15% | 77.46% |

## Acknowledgments

This project is successfully completed under the guidance of Li Niu.

## References

[1] 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.

[2] Mahsa Baktashmotlagh, Mehrtash Harandi, Brian Lovell, and Mathieu Salzmann. Unsupervised domain adaptation by domain invariant projection. 12 2013.