Project 4: Domain Adaptation for Image Classification

Xu Jiayi , Gao Yifeng, Tang Qidong, Bi Wendong

I. Domain Adaptation Methods

Domain adaptation, which adapts the model trained on a source domain to recognize instances from a new target domain, is an important problem receiving recent attention. In this section, we will introduce the principle of some domain adaptation methods. The methods we introduce can be roughly divided into following categories: projection to common space, interpolation on the manifold, sample selection and deep learning.

A. Projection to common space

The source and target domain in domain adaptation may be different but related. Therefore, a major problem in domain adaptation is how to reduce the difference between the distributions of source domain data and target domain data. Intuitively, we should discover a good feature representation across domains. The motivation of DIP and TCA is that we should learn a shared latent space underlying the domains where distance between distributions can be reduced. The shared latent space is the so called common space.

1) **DIP**

For domain adaptation, DIP [1] tries to extract the information that is invariant across the source and the target domain. Therefore, the projection that DIP learns should account for the potential distortions induced by the domain shift.

To achieve the invariance, DIP searches for a projection to a low-dimensional subspace where the source and target distributions are similar, in other words, a projection that minimizes a distance measure between the two distributions.

Suppose the source domain data is denoted as $\mathcal{D}_S = \{(x_{S_1}, y_{S_1}), \cdots, (x_{S_{n_1}}, y_{S_{n_1}})\}$, where $x_{S_{n_i}} \in \mathcal{X}$ is the input and $y_{S_{n_i}} \in \mathcal{Y}$ is the corresponding output; the target domain data is denoted as $\mathcal{D}_T = \{(x_{T_1}, y_{T_1}), \cdots, (x_{T_{n_2}}, y_{T_{n_2}})\}$. Suppose the dimension of source domain and target domain are both D, DIP needs to search 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 two distributions with MMD, the distance can be expressed as:

$$D(W^T X_S, W^T X_T) = ||\frac{1}{n_1}\sum_{i=1}^{n}\phi(W^T x_{S_i}) - \frac{1}{n2}\sum_{j=1}^{m}\phi(W^T x_{T_j})||_{\mathcal{H}},$$

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

According to MMD, learning W can be expressed as the optimization problem:

$$\min_{W} \quad D^2(W^T X_S, W^T X_T) \tag{1}$$

$$s.t. \quad W_T W = I, \tag{2}$$

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 $k(\cdot,\cdot)$. For example, if we exploit the Gaussian kernel function, the objective function can be rewritten as:

$$\begin{aligned} &D^2(W^T X_S, W^T X_T) \\ &= \frac{1}{n_1^2}\sum_{i,j=1}^{n_1} exp\left(-\frac{(x_{S_i}-x_{S_j})^T W W^T (x_{S_i}-x_{S_j})}{\sigma}\right) \\ &+ \frac{1}{n_2^2}\sum_{i,j=1}^{n_2} exp\left(-\frac{(x_{T_i}-x_{T_j})^T W W^T (x_{T_i}-x_{T_j})}{\sigma}\right) \\ &- \frac{2}{n_1 n_2}\sum_{i,j=1}^{n_1,n_2} exp\left(-\frac{(x_{S_i}-x_{T_j})^T W W^T (x_{S_i}-x_{T_j})}{\sigma}\right) \end{aligned}$$

Therefore, Eq. 1 can be computed efficiently in matrix form as:

$$D^2(W^T X_S, W^T X_T) = tr(K_W L), \tag{3}$$

where $K_W = [K_{S,S}, K_{S,T}; K_{T,S}, K_{T,T}]$; $L = [L_{i,j}]$ with $L_{ij} = 1/n_1^2$ if $x_i, x_j \in X_S$, $L_{ij} = 1/n_2^2$ if $x_i, x_j \in X_T$, otherwise, $-1/n_1 n_2$.

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

$$\min_{W} \quad tr(K_W L) \tag{4}$$

$$s.t. \quad W_T W = I, \tag{5}$$

2) **TCA**

For domain adaptation, TCA [2] tries to learn some transfer components across domains in a Reproducing Kernel Hilbert Space (RKHS) using Mean Discrepancy (MMD). It minimizes the distance between domain distributions by projecting data onto the learned transfer components.

Suppose the source domain data is denoted as $\mathcal{D}_S = \{(x_{S_1}, y_{S_1}), \cdots, (x_{S_{n_1}}, y_{S_{n_1}})\}$, where $x_{S_{n_i}} \in \mathcal{X}$ is the input and $y_{S_{n_i}} \in \mathcal{Y}$ is the corresponding output; the target domain data is denoted as $\mathcal{D}_T = \{(x_{T_1}, y_{T_1}), \cdots, (x_{T_{n_2}}, y_{T_{n_2}})\}$. Let $\mathcal{P}(X_S)$ and $\mathcal{Q}(X_T$ be the marginal distributions of X_S and X_T respectively. In general, $\mathcal{P}$ and $\mathcal{Q}$ can be different. The task of TCA is to predict the labels y_{T_i} according to the inputs x_{T_i} in the target domain. The key assumption is that $\mathcal{P} \neq \mathcal{Q}$, but $P(Y_S|X_S) = P(Y_T|X_T)$.

To estimate the distance between distributions, TCA adopts Maximum Mean Discrepancy (MMD) to compare the distributions based on Reproducing Kernel Hilbert Space (RKHS). The empirical estimate of the distance between $\mathcal{P}$ and $\mathcal{Q}$, as defined by MMD, is:

$$Dist(X, Y) = ||\frac{1}{n_1}\sum_{i=1}^{n_1}\phi(x_i) - \frac{1}{n_2}\sum_{i=1}^{n_2}\phi(y_i)||_{\mathcal{H}}. \quad (6)$$

where $\mathcal{H}$ is a universal RKHS, and $\phi : \mathcal{X} \to \mathcal{H}$.

Since TCA attempts to find a common latent representation for both X_S and X_Y that preserves the data configuration of the two domains after transformation, let $X'_S = \{x_{S_i}\} = \{\phi(x_{S_i})\}, X'_T = \{x_{T_i}\} = \{\phi(x_{T_i})\}$ and $X' = X'_S \cup X'_T$ be the transformed input sets from the source, target and combined domains respectively. Then we desire that $\mathcal{P}'(X'_S) = \mathcal{Q}'(X'_T)$. According to MMMD, the distance between two distributions can be measured as Eq. 6.

Instead of finding the nonlinear transformation ϕ explicitly, TCA transform this problem as a kernel learning problem. Therefore, the distance between the empirical means of the two domains in Eq. 6 can be written as:

$$Dist(X'_S, X'_T) = tr(KL) \quad (7)$$

where $k = [K_{S,S}, K_{S,T}; K_{T,S}, K_{T,T}]$, which is a $((n_1 + n_2) \times (n_1 + n_2))$ kernel matrix, $K_{S,S}, K_{T,T}$ and $K_{S,T}$ respectively are the kernel matrices defined by k on the data in the source domain, target domain and cross domains; and $L = [L_{i,j}]$ with $L_{ij} = 1/n_1^2$ if $x_i, x_j \in X_S$; $L_{ij} = 1/n_2^2$ if $x_i, x_j \in X_T$; otherwise, $-1/n_1 n_2$.

The kernel matrix K can be decomposed as $K = (KK^{-1/2})(K^{-1/2}K)$. Considering that we use a $(n_1 + n_2) \times m$ matrix $\widetilde{W}$ to transform the corresponding feature vectors to a m-dimensional space, the resultant kernel matrix is then:

$$\widetilde{K} = (KK^{-1/2}\widetilde{W})(\widetilde{W}^T K - 1/2K) = KWW^T K, \quad (8)$$

where $W = k^{-1/2}\widetilde{W}$.

Therefore, Eq. 7 can be further rewritten as:

$$Dist(X'_S, X'_T) = tr((KWW^T K)L) \quad (9)$$
$$= tr(W^T KLKW) \quad (10)$$

In minimizing Eq. 7, a regularization term tr($W^T W$) is usually needed to control the complexity of W. Then the kernel learning problem for domain adaptation is reduced to:

$$\max_W \quad tr(W^T W) + \mu tr(W^T KLKW) \quad (11)$$
$$s.t. \quad W^T KHKW = I, \quad (12)$$
$$(13)$$

where μ is a trade-off parameter, I is the identity matrix.

3) **SA**

Even though both the source and target data lie in the same D-dimensional space, they have been drawn according to different marginal distributions. Consequently, SA (Subspace Align) [3] selects for each domain d eigenvectors corresponding to the d largest eigenvalues derived from PCA. These eigenvectors are used as bases of the source and target subspaces, respectively denoted by X_S and X_T ($X_S, S_T \in R^{D \times d}$).

Subspace alignment approach is used here to directly compare source and target samples in their respective subspaces. M is used to align X_S to X_T, learned by minimizing the following Bregman matrix divergence:

$$F(M) = ||X_S M - X_T||_F^2 \quad (14)$$
$$M^* = argmin_M(F(M))$$

We can re-write Eq(14) as follows, where X'_S is the transpose of X_S.

$$F(M) = ||X'_S X_S M - X'_S X_T||_F^2 = ||M - X'_S X_T||_F^2$$

From the result, we can conclude that the optimal M^* is obtained as $M^* = X'_S X_T$. This implies that the new coordinate system is equivalent to $X_a = X_S X'_S X_T$. X_a is called the target aligned source coordinate system.

Through the new coordinate system X_a, we can define the similarity of y_s and y_t (y_s and y_t are respectively the instance in source and target domain).

$$Sim(y_s, y_t) = (y_s X_S M^*)(y_t X_T)' = y_s X_S M^* X'_T y'_t$$
$$= y_s A y'_t,$$

where $A = X_S X'_S X_T X'_T$.

A encodes the relative contributions of the different components of the vectors in their original space. We can use $Sim(y_s, y_t)$ directly to perform a k-nearest neighbor classification task. On the other hand, we can also project the source data via X_a into the target aligned source subspace and the target data into the target subspace (using X_T) to learn a SVM classifier.

4) **Correlation Alignment(CORAL)**

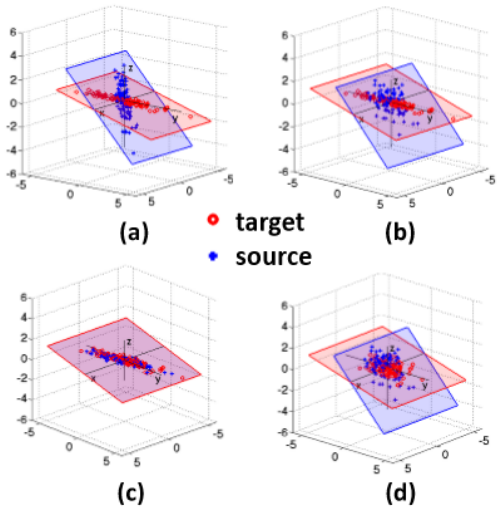

Fig. 1. CORAL

CORelation ALignment (CORAL) [4] works by aligning the distributions of the source and target features in an unsupervised manner. We propose to match the distributions by aligning the second-order statistics, namely, the covariance.

Formulation and Derivation:

C_s and C_t: covariance matrices of X_s and X_t

$\hat{C}_s$: covariance matrix of $A^T X_s$

$$min_A||\hat{C}_s - \hat{C}_t||_F^2 \;=\; min_A||A^T C_s A - C_t||_F^2 \quad (15)$$

$$C_s = U_s \Sigma U_s^T \;\; C_t = U_t \Sigma_t U_t^T$$

The optimal solution

$$A^* = U_s \Sigma_s^{-\frac{1}{2}} U_s^T U_t[1:r] \Sigma_t[1:t]^{\frac{1}{2}} U_t[1:r]^T \quad (16)$$

$$r = min(rank(C_s), rankC(C_t))$$

After obtaining $A*$, use $A^{*T} X_s$ and X_t

B. Interpolation on the manifold

1) **SGF**

SGF (Sampling Geodesic Flow) is motivated by incremental learning. Intermediate representations of data between the two domains are created by viewing the generative subspaces created from these domains as points on the Grassmann manifold. Then the projections of labeled source domain data onto these subspaces are obtained, from which a discriminative classifier is learnt to classify projected data from the target domain [5]. The schematic diagram of SGF is shown in Fig. 1.

Let S_1 and S_2 respectively denote the subspaces by performing PCA on source domain and target domain. The key problem in SGF is to obtain meaningful intermediate subspaces on the geodesic path between S_1 and S_2. After getting the collection of subspaces, denoted by S', we can model the information conveyed by S' on the source information and target information to perform recognition across domain change. The details are shown in Alg. 5.

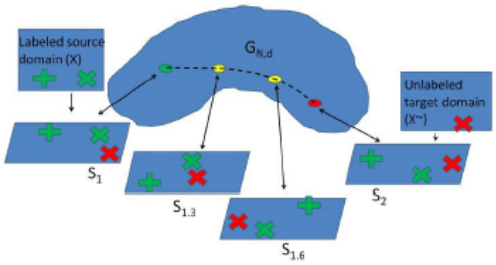

Fig. 2. Schematic diagram of SGF

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 $\tau(t')$ and $\Sigma(t')$ such that $\gamma_i(t') = cos(t'\theta_i)$ and $\sigma(t') = sin(t'\theta)$, where $\theta' s$ are the diagonal elements of Θ.
5: Compute $\Psi(t') = Q \begin{pmatrix} V_1 \tau(t'1) \\ -\tilde{V}_2 \Sigma(t') \end{pmatrix}$, for various values of $t' \in [0, 1]$.

2) **GFK** SGF is the inaugurator of using geodesic flow to derive subspace and achieves an encouraging results. However, it is not clear about the number of subspaces to sample or the dimensionality of the subspaces. GFK (Geodesic Flow Kernel) [**?**]ddresses these limitations in a simple kernel-based framework. In GFK, the subspaces are not discrete any more. An infinite number of subspaces along the geodesic flow between the source and target points are integrated. The schematic diagram of GFK is shown in Fig. 2.

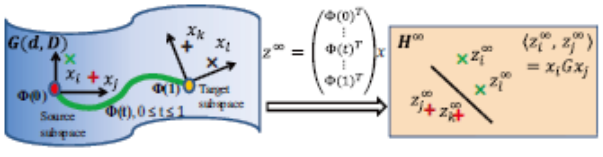

Fig. 3. Schematic diagram of GFK

Let the geodesic flow is parameterized as $\Phi : t \in [0; 1] \rightarrow \Phi(t) \in G(d; D)$ under the constraints $\Phi(0) = P_S$ and $\Phi(1) = P_T$. P_S and P_T denote the two sets of basis of the subspaces for the source and target domains. The details of GFK are summarized in Alg. 2.

Algorithm 2 Algorithm for computing the geodesic flow kernel

1: Derive the subspaces along geodesic flow : $\Phi(t) = P_S U_1 \tau(t) - R_S U_2 \Sigma(t)$, where $P_S^T P_T = U_1 \tau V_T$, $R_S^T P_T = -U_2 \Sigma V^T$.

2: Derive the closed-form of G : $G = \begin{bmatrix} P_S U_1 & R_S U_2 \end{bmatrix} \begin{bmatrix} \Lambda_1 & \Lambda_2 \\ \Lambda_2 & \Lambda_3 \end{bmatrix} \begin{bmatrix} U_1^T P_S^T & U_2^T R_S^T \end{bmatrix}$

3: Compute the projections into $\Phi(t)$ of original feature vector x_i and x_j: $< z_i^\infty, z_j^\infty >= \int_0^1 (\Phi(t)^T x_i)^T (\Phi(t)^T x_j)^T dt = x_i^T G x_j$.

As shown in Alg. 2, Once we get the matrix G, the distance between x_i and x_j can be calculated as follows in GFK:

$$< z_i^\infty, z_j^\infty >= \int_0^1 (\Phi(t)^T x_i)^T (\Phi(t)^T x_j)^T dt = x_i^T G x_j. \tag{17}$$

C. Sample selection

1) **Kernel Mean Matching (KMM)** We donate the distribution of source domain date and target domain data by $P(X_s, Y_s)$ and $P(X_t, Y_t)$. And KMM [6] is the method that accounts for the difference between $P(X_s, Y_s)$ and $P(X_s, Y_s)$ by re-weighting the training points such that the means of the training and test points in a reproducing kernel Hilbert space (RKHS) are close.

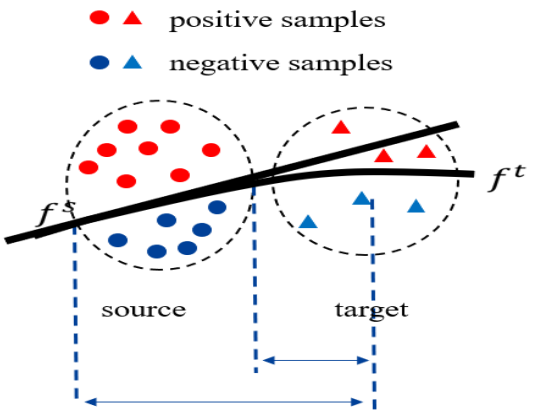

Fig. 4. KMM

Formulation and Derivation

$d(X_s, X_t)^2 = ||\frac{1}{n_s} \sum_{i=1}^{n_2} \phi(X_i^s) - \frac{1}{n_t} \sum_{i=1}^{n_t} \phi(x_i)^t||^2$

a) step1:

$$min_{\beta_i} ||\frac{1}{n_s} \sum_{i=1}^{n_2} \beta_i \phi(X_i^s) - \frac{1}{n_t} \sum_{i=1}^{n_t} \phi(x_i)^t||^2$$

b) step2:

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

s.t.

$$y_i(w^T \phi(x_i) + b) > 1 - \xi_i \quad \forall i,$$
$$x_i \geq 0$$

D. Deep Learning

1) **Deep Coral** CORAL is a "frustratingly easy" unsupervised domain adaptation method that aligns the second-order statistics of the source and target distributions with a linear transformation. So CORAL approach can be extended to learn a nonlinear transformation that aligns correlations of layer activations in deep neural networks (Deep CORAL).

We address the unsupervised domain adaptation scenario where there are no labelled training data in the target domain, and propose to leverage both the deep features pre-trained on a large generic domain (such as Imagenet) and the labelled source data. In the meantime, we also want the final learned features to work well on the target domain.

The first goal can be achieved by initializing the network parameters from the generic pre-trained network and fine-tuning it on the labelled source data. For the second goal, we propose to minimize the difference in second-order statistics between the source and target feature activations, i.e. the CORAL loss.

Deep CORAL can be any deep network incorporating the CORAL loss for domain adaptation.

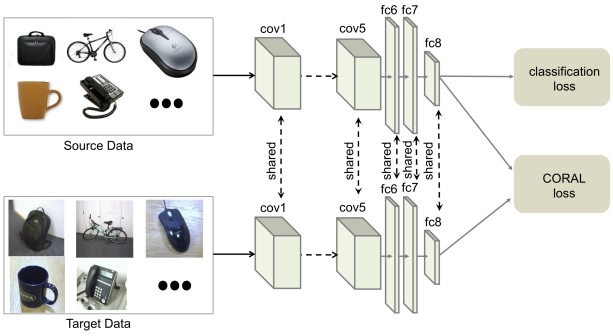

Fig. 5. Deep CORAL Model Structure

E. Coral Loss

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

$$l_{CORAL} = \frac{1}{4d^2} \|C_S - C_T\|_F^2 \tag{18}$$

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

$$C_S = \frac{1}{n_S - 1} (D_S^\intercal D_S - \frac{1}{n_S} (\mathbf{1}^\intercal D_S)^\intercal (\mathbf{1}^\intercal D_S)) \tag{19}$$

$$C_T = \frac{1}{n_T - 1} (D_T^\intercal D_T - \frac{1}{n_T} (\mathbf{1}^\intercal D_T)^\intercal (\mathbf{1}^\intercal D_T)) \tag{20}$$

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:

$$\frac{\partial l_{CORAL}}{\partial D_T^{ij}} = -\frac{1}{d^2(n_T-1)}(D_T^\intercal - \frac{1}{n_T}((\mathbf{1}^\intercal D_T)^\intercal \mathbf{1}^\intercal)^\intercal (C_S - C_T))^{ij} \tag{21}$$

$$\frac{\partial l_{CORAL}}{\partial D_S^{ij}} = \frac{1}{d^2(n_S-1)}(D_S^\intercal - \frac{1}{n_S}((\mathbf{1}^\intercal D_S)^\intercal \mathbf{1}^\intercal)^\intercal (C_S - C_T))^{ij} \tag{22}$$

F. Loss Function

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. For example, the network could project all of the source and target data to a single point, making the CORAL loss trivially zero. However, no strong classifier can be constructed on these features. Joint training with both the classification loss and CORAL loss is likely to learn features that work well on the target domain:

$$loss = l_{CLASS} + \lambda l_{CORAL} \tag{23}$$

2) **Deep Adaptation Networks** In unsupervised domain adaptation, we are given a source domain $S = \{(x_i^s, y_i^s)\}_{i=1}^{n_s}$ with n_s labeled examples, and a target domain $T = \{(x_i^t, y_i^t)\}_{i=1}^{n_t}$ with n_t unlabeled examples. The source domain and target domain are characterized by probability distributions p and q, respectively. The DNN aims to construct a deep neural network which is able to learn transferable features that bridge the cross-domain discrepancy, and build a classifier $y = \theta(x)$ which can minimize target risk $\epsilon_t(\theta) = Pr_{(x,y)\ q}[\theta(x) \neq y]$ using source supervision. In semi-supervised adaptation where the target has a small number of labeled examples, we denote by $D_a\{(x_i^a, y_i^a)\}$, the n_a annotated examples of source and target domains.

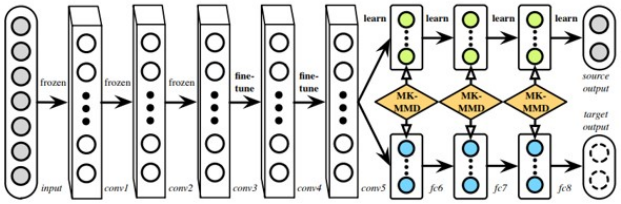

Fig. 6. The DAN architecture for learning transferable features.Since deep features eventually transition from general to specic along the network. The features extracted by convolutional layers conv1conv3 are general, hence these layers are frozen, the features extracted by layers conv4conv5 are slightly less transferable, hence these layers are learned via ne-tuning, and fully connected layers fc6fc8 are tailored to t specic tasks, hence they are not transferable and should be adapted with MK-MMD.

The loss of training part is :

$$min_\theta \frac{1}{n_a}\sum_{i=1}^{n_a} J(\theta(x_i^a), y_i^a) + \lambda \sum_{l=l_1}^{l2} d_k^2(D_s^l, D_t^l), \tag{24}$$

Where J is the Cross Entropy Loss for classification and the second term is the MK-MMD-based multi-layer adaptation regularizer to the CNN-risk. Then $\lambda > 0$ is a penalty parameter, l_1 and l_2 are layer indices between which the regularizor is effective. In our training process, we set $l1 = 6$ and $l2 = 8$, although different configurations are also possible, depending on the size of the labeled source dataset and the number of parameters in the layers that to be fin-tuned.

The squared formulation of MK-MMD is defined as:

$$d_k^2(p,q) \equiv ||E_p[\phi(x^s)] - E_q[\phi(x^t)]||_{H_k}^2 \tag{25}$$

For the finally classifier and feature extractor, here we select Resnet50 which is different from the paper which used Alexnet. And we think this can get a better results. However, the number of raw image in our dataset is so small that we can't train a CNN model from the beginning, so we use model pretrained on ImageNet 2012 and then fine-tune it.

II. Experiments

In this section, we will introduce the details about the experiment procedure, and the results of different methods will be displayed and compared.

A. About the Dataset

The dataset we use is Office-Home dataset, which 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. We conduct domain adaptation experiments in the following three settings: Art $\rightarrow$ Real, Clip $\rightarrow$ Real and Product $\rightarrow$ Real.

B. DIP

1) KNN as classifier: To test the efficiency of DIP, we first train KNN on DIP projected source domain data, and then use the trained KNN to classify the target domain data. The classification accuracy on Art$\rightarrow$RealWorld, Clipart$\rightarrow$RealWorld, product$\rightarrow$RealWorld are shown as Fig. 7. This result image is composed of three sections: the target domain are all RealWorld pictures, the source domain are Art, Clipart and Product respectively. In each section, projected dimension 200 and 1000 is tested. In each projected dimension, we vary the number of neighbors for KNN, from 1, 3, 5, 10, 20 to 50. The blue bar in the picture indicates the number of neighbors for KNN, the orange lines represent the classification accuracy of projected dimension 200 on different # neighbors, and the gray lines represent the classification accuracy of projected dimension 1000 on different # neighbors.

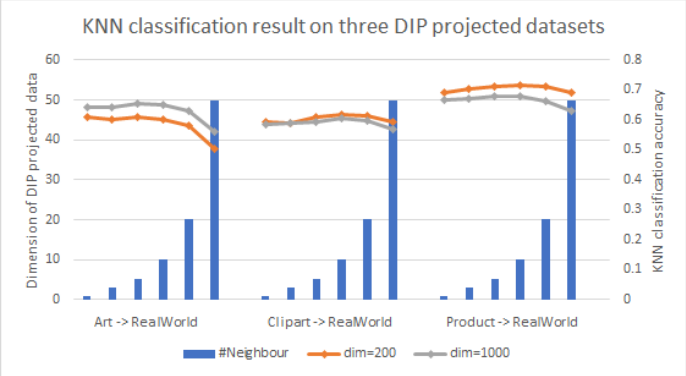

Fig. 7. KNN classification result on three DIP projected datasets

As we can see from Fig. 7, larger dimension and larger number of neighbors do not guarantee a higher accuracy. And trading off efficiency and accuracy, taking the number of neighbors = 5, and the projected dimension = 200 is a good choice.

2) SVM as classifier: Similar to KNN as a classifier, to test the efficiency of DIP, we first train SVM on DIP projected source domain data, and then use the trained SVM to classify the target domain data. The classification accuracy on Art→RealWorld, Clipart→RealWorld, product→RealWorld are shown as Fig. 8. This result image is composed of three sections: the target domain are all RealWorld pictures, the source domain are Art, Clipart and Product respectively. In each section, SVM with linear kernel, polynomial kernel, rbf kernel and sigmoid kernel is tested. The orange lines represent the classification accuracy of projected dimension 200 on different kernels, and the gray lines represent the classification accuracy of projected dimension 1000 on different kernels.

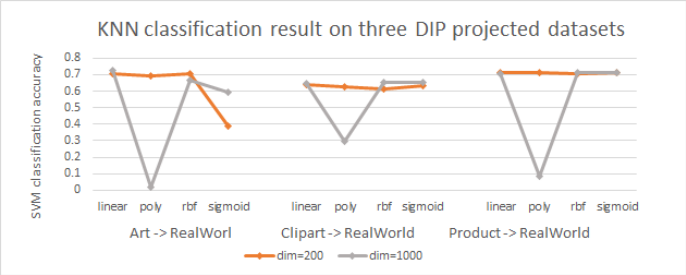

Fig. 8. SVM classification result on three DIP projected datasets

As we can see from Fig. 8, higher projected dimension (1000) cannot guarantee higher classification accuracy, and it is less robust than lower dimension (200) because dimension 1000 performs much worse than dimension 200 using polynomial kernel. Compared to the results of KNN (around 0.5), the accuracy of KNN is much better (around 0.7 and 0.6). Therefore, we should take SVM as our classifier when we use DIP to do domain adaptation. In addition, considering the time and computation consumption, we should use linear SVM and take the projected dimension = 200.

C. TCA

1) KNN as classifier: To test the efficiency of TCA, we first train KNN on TCA projected source domain data, and then use the trained KNN to classify the target domain data. The classification accuracy on Art→RealWorld, Clipart→RealWorld, product→RealWorld are shown as Fig. 9. This result image is composed of three sections: the target domain are all RealWorld pictures, the source domain are Art, Clipart and Product respectively. In each section, projected dimension from 10, 30, 50, 100, 200, 500, 1000, 1500 to 2000 is tested. In each projected dimension, we vary the number of neighbors for KNN, from 1, 3, 5, 10, 20 to 50. The blue bar in the picture indicates the TCA projected dimension, each orange bar indicates the number of neighbors for KNN in the specific dimension, and the yellow line represents the classification accuracy.

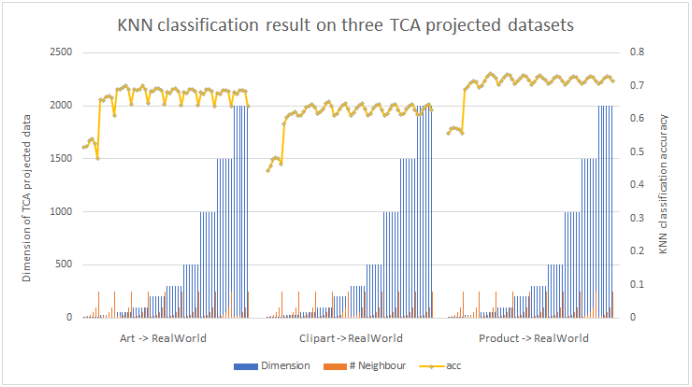

Fig. 9. KNN classification result on three TCA projected datasets

As we can see from Fig. 9, when the projected dimension is small, no matter what the number of neighbors for KNN is, the classification accuracy is low. This is reasonable because such a low dimension space may not preserve the feature information. However, when the projected dimension is greater than 100, the highest classification has no big difference among different dimensions. That maybe because the transfer components' dimension is around 100. In addition, we can find that too small and too large number of neighbors for KNN will not lead to high classification accuracy, #neighbors = 5 or 10 is a good choice.

Furthermore, we compare the classification results of TCA with different kernels. The KNN classification result with different kernels are shown as Tab. I. The 'base' represents the classification accuracy using KNN without TCA projection, we use PCA to do dimension reduction. 'tca-primal' represents classification accuracy using KNN on TCA projection without kernel function. 'tca-linear' and 'tca-rbf' represents classification accuracy using KNN on TCA projection with linear kernel and rbf kernel respectively.

2) SVM as classifier: Similar to KNN as a classifier, to test the efficiency of TCA, we first train SVM on TCA projected source domain data, and then use the trained SVM to classify the target domain data. The clas-

TABLE I
KNN CLASSIFICATION ACCURACY OF TCA ON DIFFERENT PROJECTION DIMENSIONS AND KERNELS (NEIGHBOR = 5, SOURCE = ART)

Dim	base	tca-primal	tca-linear	tca-rbf
10	0.221253	0.534083	0.499196	0.524213
30	0.447785	0.666972	0.622905	0.636676
50	0.534542	0.695432	0.645398	0.524213
100	0.599265	0.690153	0.652742	0.661693
200	0.644480	0.691071	0.651365	0.636676
300	0.647693	0.690153	0.652283	0.672022
500	0.662841	0.689465	0.652283	0.661693
1000	0.658710	0.6887766	0.652283	0.672940
1500	0.658021	0.687170	0.652283	0.672022
2000	0.656873	0.687170	0.652283	0.672251

sification accuracy on Art→RealWorld, Clipart→RealWorld, product→RealWorld are shown as Fig. 10. This result image is composed of three sections: the target domain are all Real-World pictures, the source domain are Art, Clipart and Product respectively. In each section, projected dimension from 10, 30, 50, 100, 200, 500, 1000, 1500 to 2000 is tested. The blue bar in the picture indicates the TCA projected dimension, and the yellow line represents the classification accuracy.

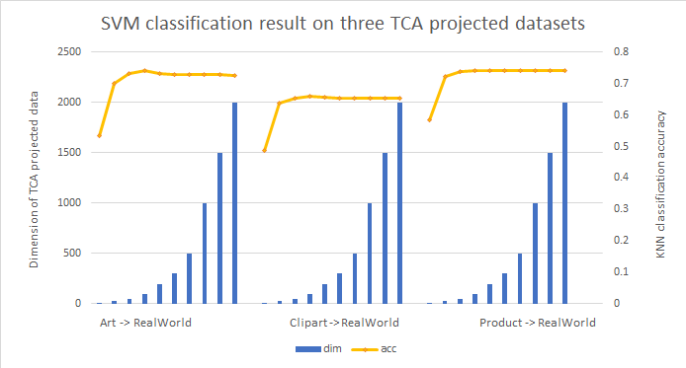

Fig. 10. SVM classification result on three TCA projected datasets

As we can see from Fig. 10, when the projected dimension is small, the classification accuracy is low. This is understandable because such a low dimension space may not preserve the necessary information for SVM to do classification. However, when the projected dimension is greater than 100, there is no accuracy improvement with larger dimension, sometimes the accuracy even drops with larger dimension. That maybe because the transfer components' dimension is around 100, which is consistent with the results of KNN on TCA. Therefore, considering accuracy and efficiency, we should project the dataset to dimension 100.

Furthermore, we compare the SVM classification results of TCA with different kernels. The SVM classification result with different kernels are shown as Tab. II. The 'base' represents the classification accuracy using SVM without TCA projection, we use PCA to do dimension reduction. 'tca-primal' represents classification accuracy using SVM on TCA projection without kernel function. 'tca-linear' and 'tca-rbf' represents classification accuracy using SVM on TCA projection with linear kernel and rbf kernel respectively.

TABLE II
SVM CLASSIFICATION ACCURACY OF TCA ON DIFFERENT PROJECTION DIMENSIONS AND KERNELS (NEIGHBOR = 5, SOURCE = ART)

Dim	base	tca-primal	tca-linear	tca-rbf
10	0.221253	0.534083	0.102364	0.535230
30	0.447785	0.666972	0.1374799	0.700481
50	0.534542	0.695432	0.138627	0.731007
100	0.599265	0.690153	0.139086	0.742712
200	0.644480	0.691071	0.139086	0.731696
300	0.647693	0.690153	0.139086	0.729859
500	0.662841	0.689465	0.139086	0.728712
1000	0.658710	0.6887766	0.139086	:0.729400
1500	0.658021	0.687170	0.139086	0.728712
2000	0.656873	0.687170	0.139086	0.727335

D. SA

To test the effect of SA, we first get the target aligned source coordinate system X_a. Then, we project the source data via X_a into the target aligned source subspace and meanwhile the target data are projected into the target subspace(using X_T). Based on the projected data, we train a SVM classifier and a linear classifier to complete the image classification task. The result is shown in Table III.

In the three domain settings, we use SVM as classifier and explore the effect of the value of C. The results are shown in the following figures. The purple bar shows the cross-validation accuracy while training, and the red bar shows the test accuracy while testing.

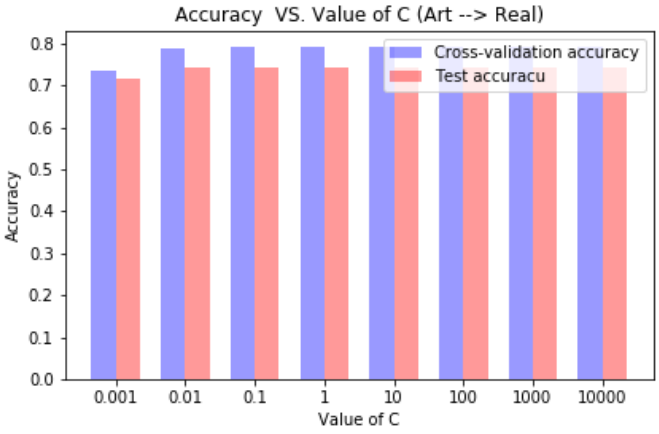

Fig. 11. The Effect of Value C in SA (Art → Real)

As we can see, when source domains are Clipart and Product, the cross-validation accuracy is relatively high but the test accuracy is around 70%. When source domain is Art, the cross-validation accuracy and the test accuracy are almost equal. We think this phenomenon may be caused by overfitting and underfitting issue. To be specific, the transformation matrix X_a in the former two problems is "soft". As a result, the source domain data X_S does not change too much thus the training accuracy can reach high. In Art domain, the

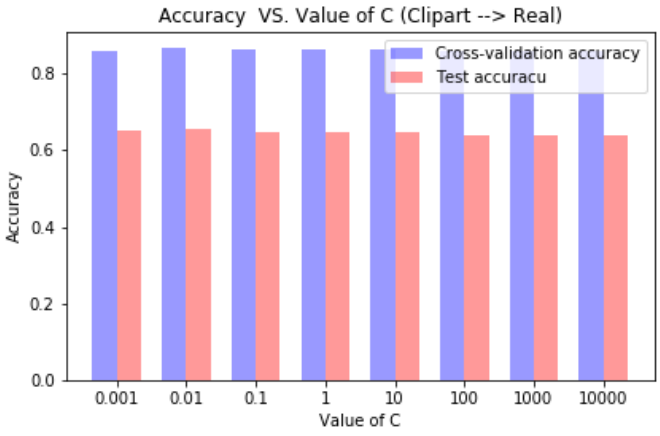

Fig. 12. The Effect of Value C in SA (Clipart → Real)

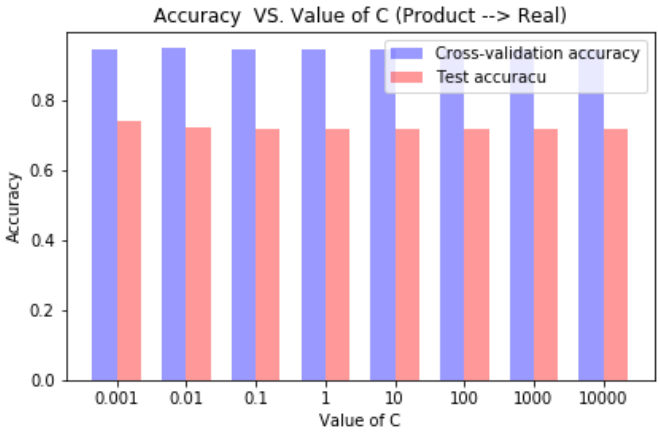

Fig. 13. The Effect of Value C in SA (Product → Real)

transformation matrix X_a changes the source data much more. The source domain's structure is destroyed in some extent so that the training accuracy is comparatively low. However, this greater change makes the projected data are closer to target data, so the test accuracy in Art domain is comparatively high.

Except SVM classifier, we also use Linear classifier, the result is shown in Table III.

TABLE III
BEST CLASSIFICATION ACCURACY OF SA ON SVM AND LINEAR CLASSIFIER

Method \ Domain	Art → Real	Clipart → Real	Product → Real
SVM	0.7408	0.6529	0.7241
Linear	0.7475	0.6648	0.7306

From the result, we are sure that linear classifier is better than SVM classifier in SA method.

E. GFK

As shown in Eq (17), $\mathbf{x}_i^T \mathbf{G} \mathbf{x}_j$ represents the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. Therefore,It is natural to choose KNN (K-nearest neighbouring) as the classifier.

To prove that GFK is helpful for domain adaptation learning, we compare its effect with baseline. The dimension of geodesic flow kernel is set as 200, and the K in KNN is set as 1. By contrast, the baseline is a simple KNN with $K = 1$ without any special processing. The results are shown in Table IV. We can see that, the classification accuracy of all 3 groups of experiment with different domains increase by about 1%.

TABLE IV
GFK (K=1, DIM=100) CLASSIFICATION ACCURACY COMPARED WITH BASELINE

Method \ Domain	Art → Real	Clipart → Real	Product → Real
Baseline	0.6580	0.5873	0.6957
GFK	0.6690	0.6025	0.7046

The Effect of Geodesic Flow Kernel Dimension

At first, we explore the effect of the dimension of geodesic flow kernel. As we can see from Fig. 14, with different domains (A→R, C→R, P→R) and different K in KNN ($K = 1$, $K = 10$), the most proper dimension is between 60 and 100 (marked by a dot on the curve). When the dimension is below 40, the accuracy is obviously low. When the dimension continues to increase after 100, the accuracy gradually goes down. Therefore, we think $dim = 100$ is a good choice for all time.

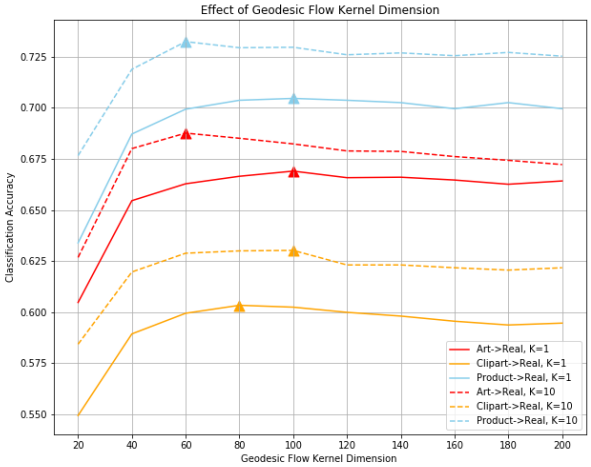

Fig. 14. The Effect of Geodesic Flow Kernel Dimension

The Effect of K in KNN

The magnitude of K in KNN also has a significant effect on the performance of GFK. In Fig. 15, K is successively set as $[1, 3, 5, 7, 9]$. As the value of K increases, the accuracy shows an upward trend. By contrast, K is successively set as $[10, 30, 50, 70, 90]$ in Fig. 16. With the increasing of K, the accuracy shows a decline trend. Therefore, we are confident that $K = 10$ is nearly the optimal choice.

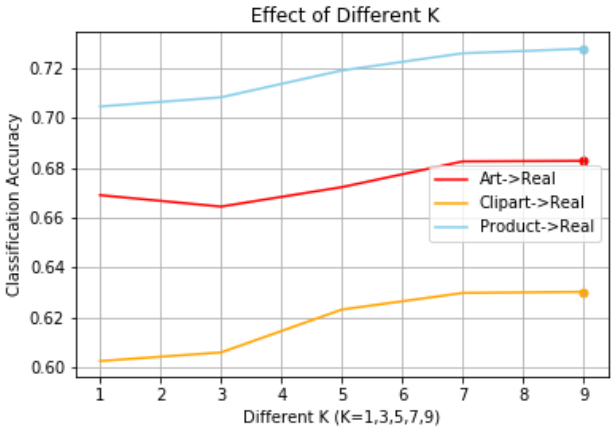

Fig. 15. The Effect of K in KNN (K=[1,3,5,7,9])

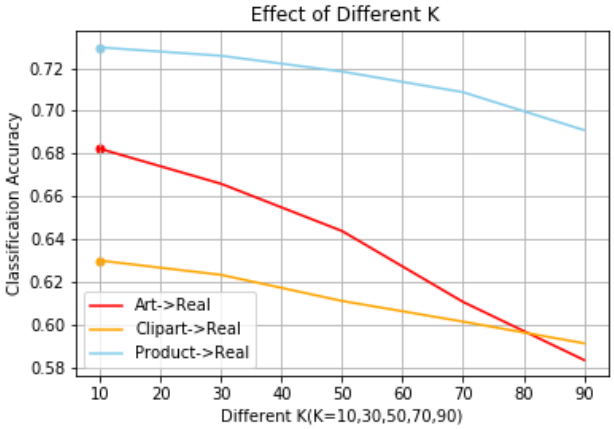

Fig. 16. The Effect of K in KNN (K=[10,30,50,70,90])

F. Correlation Alignment(CORAL)

In this part, we first use CORAL method to fit the source domain data, and then apply the policy learned on source domain data to target domain data. After domain adaptation part, we test the classification accuracy on the target domain data processed on target domain by our classifier.

Here we use **three** kinds of classifier to test the performance of CORAL method: KNN, Linear SVM, rbf kernel SVM. And for each, we use K-ford cross validation to find the best parameter such as C for SVM or N-neighbours for KNN.

Then the results are as follows.

1) K-ford for model selection:
Here we use data from domain A to domain R as our data for model selection.

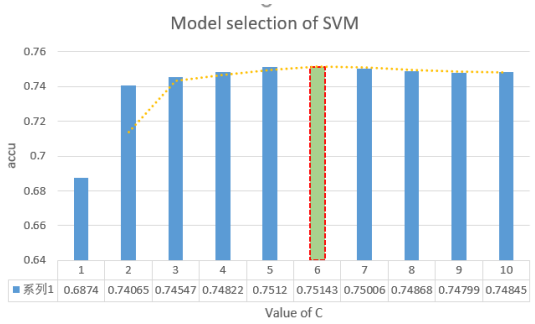

Fig. 17. Model selection for SVM

And the best value of parameter C is : 6.

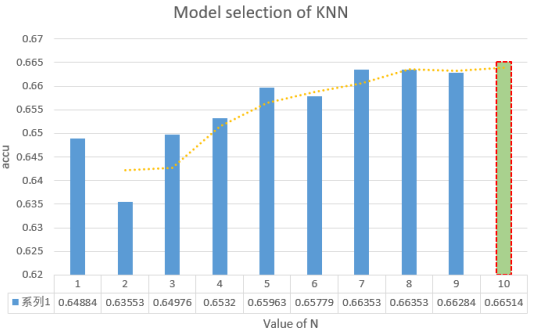

Fig. 18. Model selection for KNN

And the best value of parameter N is : 10.

2) The best result of each classifiers:

TABLE V
BASELINE BEFORE DOMAIN ADAPTATION

Domain	Basic rbf SVM	Basic linear SVM	Basic KNN
$A \rightarrow R$	0.7514	0.7434	0.6669
$C \rightarrow R$	0.6628	0.6501	0.6179
$P \rightarrow R$	0.7328	0.7198	0.7191

TABLE VI
RESULTS AFTER DOMAIN ADAPTATION

Data domain trans	Linear SVM	rbf SVM	KNN
$A \rightarrow R$	0.7315	0.7427	0.6580
$C \rightarrow R$	0.6479	0.6576	0.5979
$P \rightarrow R$	0.7152	0.7269	0.6931

G. Kernel Mean Matching (KMM)

In this part, we first use KMM to find the fittable weight on each sample of source domain, here we donate this weight matrix by β and then we apply this weights matrix β on target domain data we classified by SVM.

Here we can't use KNN for we have to use beta on the SVM. Similarly, We test the performance of KMM on Linear

kernel SVM and rbf kernel SVM after using K-ford to help us select the best model parameters.

The results of experiments are as follows:

1) K-ford of SVM to find best C:
   Here we also use the validation results of CORAL, we select C = 6 as our best model.
2) Test best accuracy with best C:

TABLE VII
BASELINE BEFORE DOMAIN ADAPTATION

Data domain trans	Basic rbf SVM	Basic linear SVM
$A \rightarrow R$	0.7514	0.7434
$C \rightarrow R$	0.6628	0.6501
$P \rightarrow R$	0.7328	0.7198

TABLE VIII
RESULTS AFTER DOMAIN ADAPTATION

Data domain trans	Linear SVM	rbf SVM
$A \rightarrow R$	0.7434	0.7507
$C \rightarrow R$	0.6495	0.6638
$P \rightarrow R$	0.7198	0.7331

H. Deep CORAL

We make the experiment according the the paper "Deep CORAL: Correlation Alignment for Deep Domain Adaptation". The CORAL model in modified from the original AlexNet. We set the batch size to 128, base learning rate to 10^3, weight decay to 5×10^4, and momentum to 0.9.

The dataset is Office Home dataset, and we use the original images as input (resized), and train from scratch.

As is mentioned in the paper, our λ value in the loss function is set in such way that at the end of training, the classification loss and the CORAL loss is roughly the same. It seems be a reasonable choice as we want to have a feature representation that is both discriminative and also minimizes the distance between the source and target domains.

In the experiment, we can see that the coral loss is extremely small at the beginning of the training phase, for the output layers to calculate the coral loss haven't leant a valid representation for the feature space. Therefore the represented source space and the target space are relatively random, as well as the covariance matrix, causing the coral loss to be trivially zero.

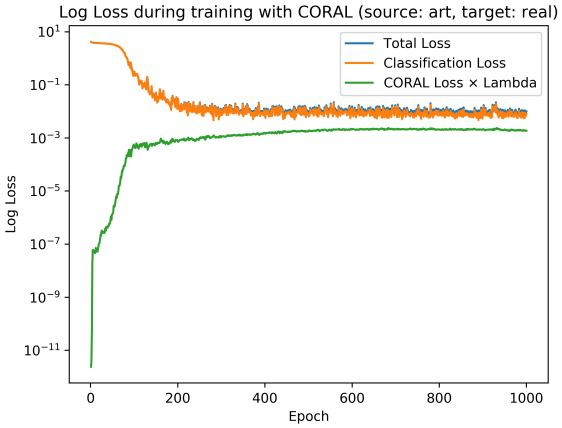

Fig. 19. Log Loss with CORAL (Art to Real)

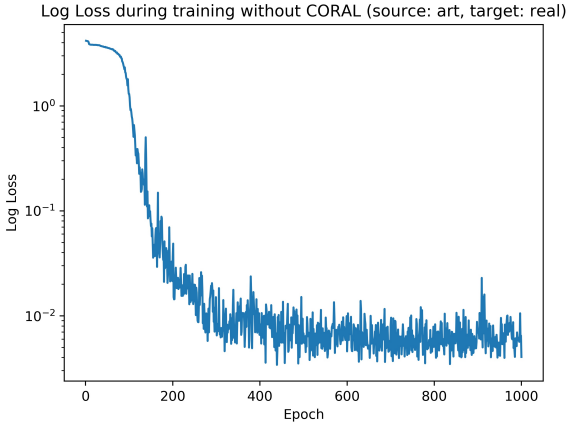

Fig. 20. Log Loss without CORAL (Art to Real)

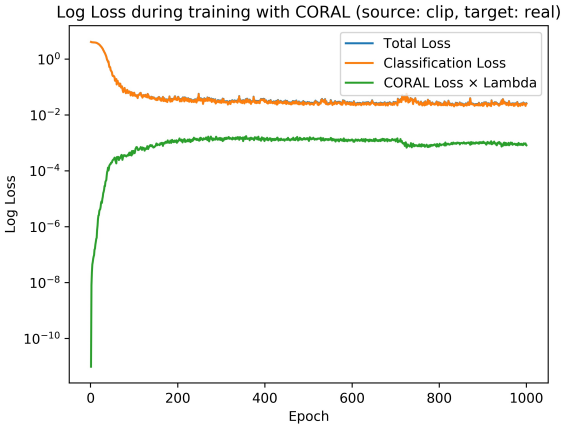

Fig. 21. Log Loss with CORAL (Clip Art to Real)

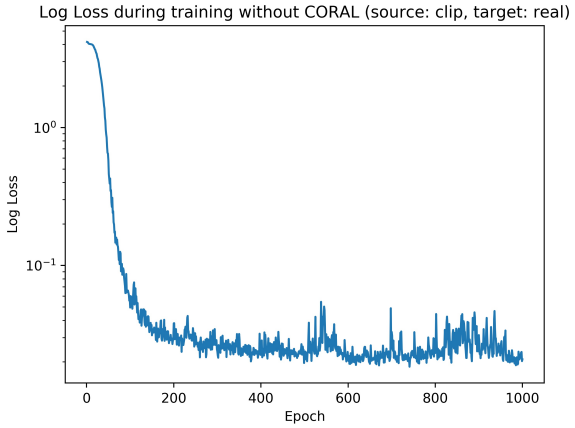

Fig. 22. Log Loss without CORAL (Clip Art to Real)

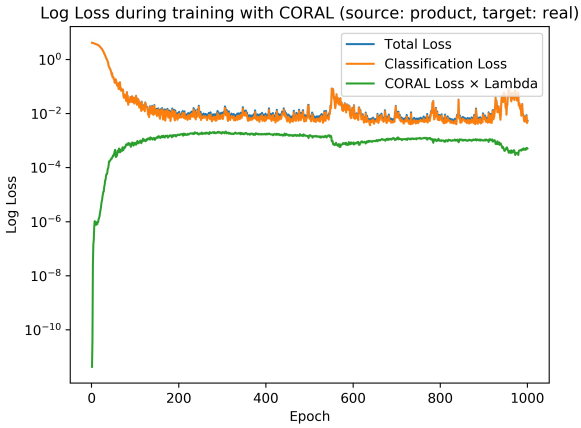

Fig. 23. Log Loss with CORAL (Product to Real)

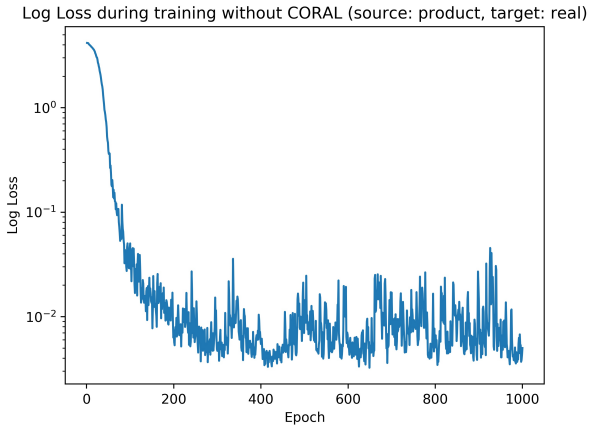

Fig. 24. Log Loss without CORAL (Product to Real)

As the classification loss drops, the output layer gradually learns to represent the feature space. And the difference representation between the source space and target space is more obvious. Hence the coral loss begin to grow. The training process then deals with the classification loss and the coral loss at the same time.

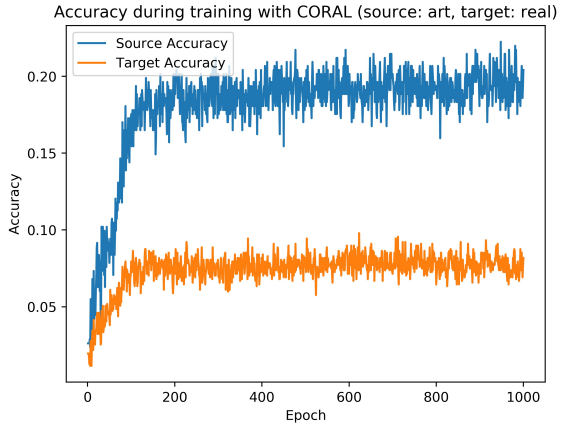

Fig. 25. Accuracy with CORAL (Art to Real)

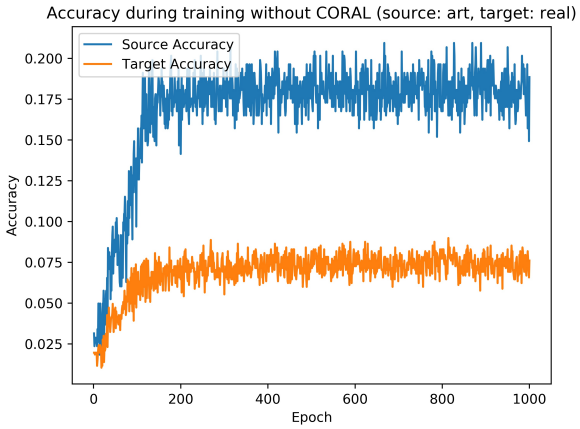

Fig. 26. Accuracy without CORAL (Art to Real)

TABLE IX
TARGET ACCURACY WITH/WITHOUT CORAL (TARGET DOMAIN: REAL)

Source Domain	Art	Clip Art	Product
With CORAL	0.0758	0.1257	0.1397
Without CORAL	0.0733	0.1081	0.1418

From the result graph, we can see that the basic model (AlexNet) and the model with CORAL loss layer does not show much difference in performance.

To give a reasonable explanation for the result, we recall the value of λ. In the original paper, the source domain is basically a general feature domain, such as ImageNet. That means, in the original paper, the source domain will learn a feature representation in large data scale, to reduce the randomness

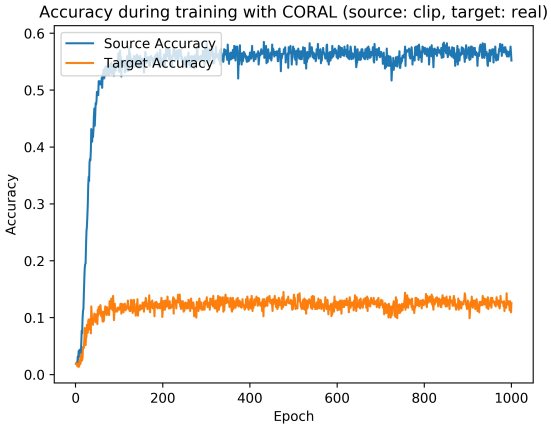

Fig. 27. Accuracy with CORAL (Clip Art to Real)

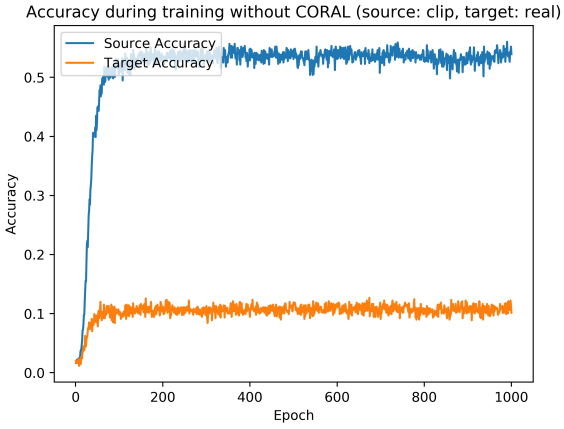

Fig. 28. Accuracy without CORAL (Clip Art to Real)

of the representation. Practically the value of λ is set to 0.5 to make the CORAL loss and the classification loss roughly the same at the end of the training process.

However, in our experiment, the value of λ need to be set to 1000 to guarantee the same magnitude between the CORAL loss and the classification loss. The absolute CORAL loss in much smaller than what we expected. That means, in our dataset Office Home dataset (the input is the resized raw images), the scale of data do not convince a reliable representation even in the source domain with training labels. The result that the poor performance in the source dataset also approves the explanation.

I. Domain Adaptation Networks(DAN)

1) **Use raw image as the input of network.**
For the reason of few num of images in each class used for training, we failed to train the model very well and only get a baseline of $acc = 22.21365\%$ on the model trained on source domain and testing on the target domain data.

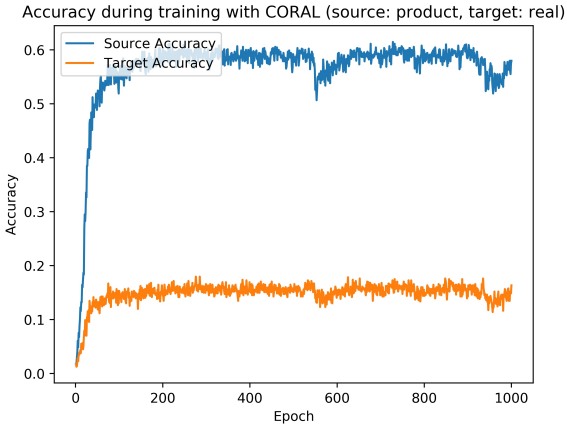

Fig. 29. Accuracy with CORAL (Product to Real)

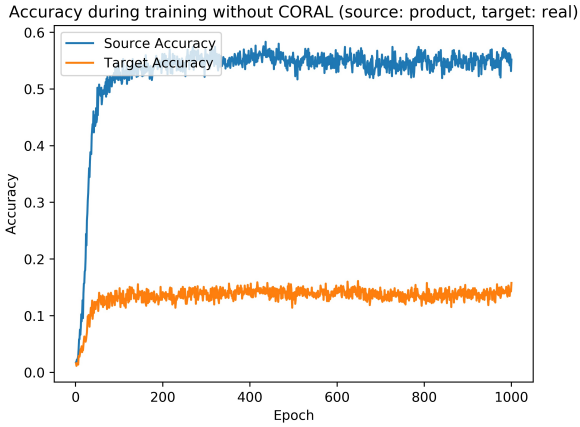

Fig. 30. Accuracy without CORAL (Product to Real)

TABLE X
DAN RESULTS WITH IMAGE INPUT

Data domain trans	DAN
$A \rightarrow R$	0.1653
$C \rightarrow R$	0.1285
$P \rightarrow R$	0.1438

2) **Use extracted feature as the input of network.**
Though using the raw image we can't easily trained a DAN model, in this part, we use the extracted feature vector of each sample as the input this time to train our model and test our domain adaptation method of DAN. And here we get a base line of classification accuracy of 79.3%. Also the DAN model's accuracy is better than that of using image as input.
The result of this part can be seen at 2

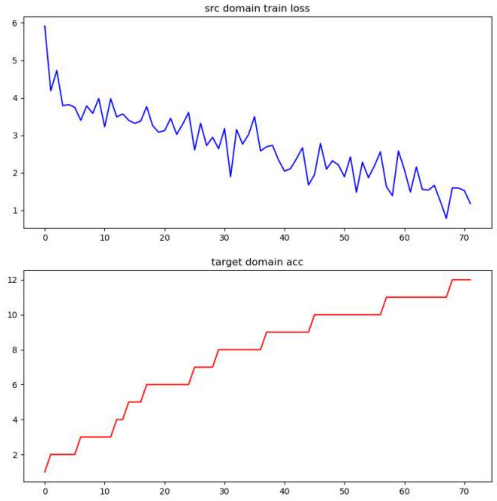

Fig. 31. DAN results with image inputs.

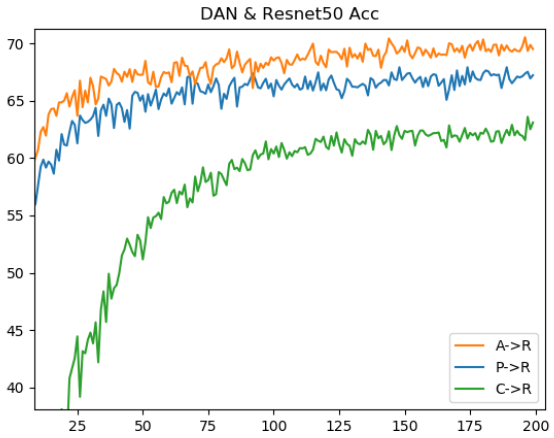

Fig. 32. DAN results with feature inputs.

TABLE XI
DAN RESULTS WITH FEATURE INPUT

Data domain trans	DAN
$A \rightarrow R$	0.7081
$C \rightarrow R$	0.6295
$P \rightarrow R$	0.6718

3) **Analysis:** And we think the reason why image-input model can not work well is that the number of image is few and the CNN neural network can not easily extract feature from the raw image. And when training the model, it's quite easily to be over -fitting.But the extracted feature input can avoid this case to some extent. However, we can still not get a result better than baseline, so we think the model parameter we use may not very reasonable. Because one experiment needs to run 200 epochs which takes around 10 hour on GPU, we do not have much time to do more experiments. And we believe we can get result higher than baseline after more experiments.

III. Conclusion

In this project, we are required to use Office-Home dataset to do domain adaptation. We should train a classifier on Art dataset, Clipart dataset, and Product dataset respectively, and then transfer it to RealWorld dataset. We implement 4 kinds of domain adaptation methods, 8 methods altogether. The best performance of them on three domain adaptation tasks are shown as Tab. XII and Tab. XIII.

TABLE XII
BEST CLASSIFICATION ACCURACY ON THREE DOMAIN ADAPTATION TASKS (METHODS USING EXTRACTED FEATURES)

Method	Art → Real	Clipart → Real	Product → Real
DIP	0.7245	0.6529	0.7151
TCA	0.7427	0.6605	0.7417
SA	0.7475	0.6648	0.7306
CORAL	0.7427	0.6576	0.7269
GFK	0.6976	0.6404	0.7423
KMM	0.7507	0.6638	0.7331

TABLE XIII
BEST CLASSIFICATION ACCURACY ON THREE DOMAIN ADAPTATION TASKS (METHODS USING RAW IMAGES)

Method	Art → Real	Clipart → Real	Product → Real
Deep Coral	0.0758	0.1257	0.1397
DAN	0.1653	0.1285	0.1438

REFERENCES

[1] M. Baktashmotlagh, M. T. Harandi, B. C. Lovell, and M. Salzmann, "Unsupervised domain adaptation by domain invariant projection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 769–776.

[2] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *IEEE Transactions on Neural Networks*, vol. 22, no. 2, pp. 199–210, 2010.

[3] B. Fernando, A. Habrard, M. Sebban, and T. Tuytelaars, "Unsupervised visual domain adaptation using subspace alignment," in *Proceedings of the IEEE international conference on computer vision*, 2013, pp. 2960–2967.

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

[5] B. Gong, Y. Shi, F. Sha, and K. Grauman, "Geodesic flow kernel for unsupervised domain adaptation," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 2066–2073.

[6] J. Huang, A. Gretton, K. Borgwardt, B. Schölkopf, and A. J. Smola, "Correcting sample selection bias by unlabeled data," in *Advances in neural information processing systems*, 2007, pp. 601–608.