Shanghai Jiao Tong University
CS245 – Fundamentals of Data Science
2nd Semester, 2019-2020

Domain Adaptation for Image Classification

Project 4 for Principle of Data Science (CS245)

Zijie Zhu (朱子杰) 517030910389
Mengtian Zhang (张孟天) 517030910387

June 23, 2020

Abstract

In this experiment, we tried many domain adaptation methods, which improved the performance of classification tasks on different domains. The dataset we use is Office-Home, which contains 65 categories of pictures in three different domains(that is "A" for art, "C" for clipart and "P" for product). We have studied six traditional adaptation methods(KMM, CORAL, TCA, SA, GFK and MEDA) and two depth methods(Deep CORAL and DAN). Our report is mainly divided into two parts: first, we analyze the various methods to be studied at the principle level, and then we carry out various experiments to get the results and analyze them. Combined with all the experimental results, we have discovered many interesting findings.

1 Introduction

In the real world, there is an endless variety of related objects. Similar object types may appear in different scene or environments, such as a real picture of an apple and a painting of an apple. In this case, we call the samples in different domains.

In this experiment, we still carry out the task of classification. However, the difference between the labeled training source domain and the target domain is different. It means that only using the model trained in the source domain, the spatial distribution of the sample points in the target domain can not be well divided, which makes the performance of the model poor.

Can machine models draw inferences from one instance like humans? This requires that the model has the ability to treat the past knowledge as experience and transform it into the ability to recognize new things, which we called transfer learning or domain adaption. As human beings cannot collect all the sample data and labels of the world, it is necessary to learn how to make the model have the ability of migration.

In this experiment, we try many traditional adaptation and deep adaptation methods. The characteristics and advantages and disadvantages of each method are analyzed from the perspective of

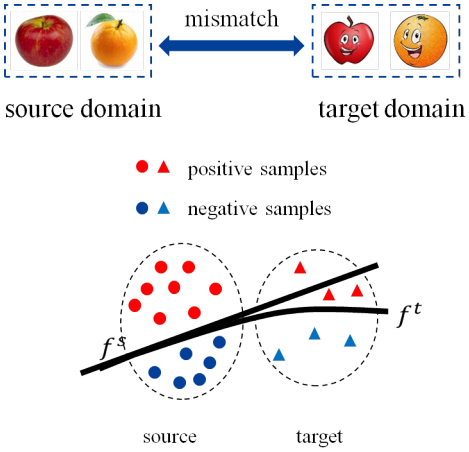

Figure 1: Illustration of different domains.

principle and experimental results. Using this method, we try to deepen the understanding of domain adaptation in turn

2 Brief Introduction for Principal

2.1 KMM

The full name of KMM algorithm is kernel mean matching, which was put forward in [1] in 2007. The basic idea of KMM is to "softly" select the source domain samples in a way that train a vector giving each sample a weight. And then use this weight to training model with different punishment extent.

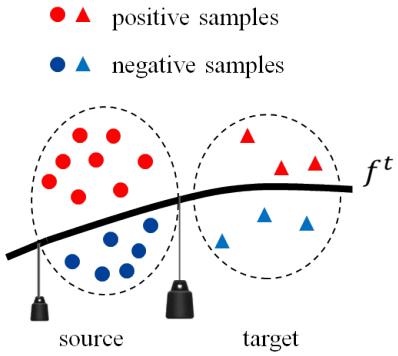

Figure 2: Illustration of KMM algorithm.

Take using SVM model as example, the first step is to solve a optimization problem, which makes the difference of mean between target domain with the kernel projections of source domain be minimized.

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

The second step is to use the obtained weight vector β to weight the penalty term of SVM model.

$$\begin{aligned} \text{Step 2}: \quad \min_{\mathbf{w},b,\xi_i} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \beta_i \xi_i \\ \text{s.t.} \quad & y_i\left(\mathbf{w}^T\phi\left(\mathbf{x}_i\right)+b\right) \geq 1-\xi_i, \quad \forall i \\ & \xi_i \geq 0, \quad \forall i \end{aligned}$$

This method can be understood as using the distribution of the target domain to make the model trained on the source domain incline with the weight of the target domain, so as to achieve the goal of the model matching the target domain.

2.2 CORAL & Deep CORAL

CORAL is a projection method put forward in [2], of which destination is to learning a projection matrix A^* aligning the source domain and the target domain with the minimum difference of covariance matrix. It can be represented in the form of:

$$\min_A ||A^T C_s A - C_t||_F^2$$

After getting the optimal solution A^*, use $A^{*T}X_s$ as the training data and X_t as the testing data.

And deep CORAL is a deep adaptation method using deep network and the traditional idea of CORAL [3]. Deep CORAL framework constructs two neural networks sharing the same parameters and uses CORAL loss as a assist loss.

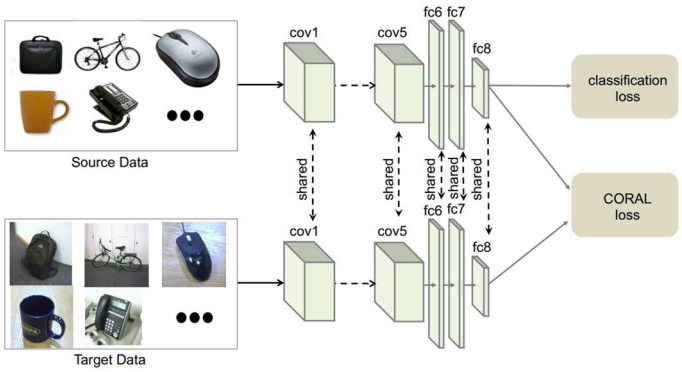

Figure 3: Illustration of deep CORAL framework.

2.3 Transfer Component Analysis (TCA)

For domain adaptation, TCA [4] 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.

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. The general assumption in domain adaptation is that the marginal densities, $P(X_S)$ and $Q(X_T)$, are very different. In this method, we attempt to find a common latent representation for both X_S and X_T that

preserves the data configuration of the two domains after transformation. Let the desired nonlinear transformation be $\phi : \mathcal{X} \to \mathcal{H}$. 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 $P'(X'_S) = Q'(X'_T)$.

Assuming that ϕ is the feature map induced by a universal kernel. The distance between two distributions P and Q can be empirically measured by the (squared) distance between the empirical means of the two domains:

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

Therefore, a desired nonlinear mapping ϕ can be found by minimizing this quantity. However, ϕ is usually highly nonlinear and a direct optimization of 1 can get stuck in poor local minima. We thus need to find a new approach, based on the following assumption.

The key assumption in the proposed domain adaptation setting is that $P \neq Q$, but $P(Y_S|\phi(X_S)) = P(Y_T|\phi(X_T))$ **under a transformation mapping ϕ on the input.**

2.4 Subspace Alignment (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) [5] 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 ($X_S, X_T \in R^{D \times d}$). In the method, it suggests to project each source ($\mathbf{y_S}$) and target ($\mathbf{y_T}$) data (where $\mathbf{y_S}, \mathbf{y_T} \in R^{1 \times D}$) to its respective subspace X_S and X_T by the operations $\mathbf{y_S}X_S$ and $\mathbf{y_T}X_T$, respectively. Then, we learn a linear transformation function that align the source subspace coordinate system to the target one. This step allows us to directly compare source and target samples in their respective subspaces without unnecessary data projections. To achieve this task, we use a subspace alignment approach. We align basis vectors by using a transformation matrix M from X_S and to X_T. M is learned by minimizing the following Bregman matrix divergence:

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

$$M^* = \arg\min_M (F(M)) \tag{3}$$

where $\|\cdot\|^2_F$ is the Frobenius norm. Since X_S and X_T are generated from the first d eigenvectors, it turns out that they tend to be intrinsically regularized. Therefore, we do not add a regularization term in the Equation 2. It is thus possible to obtain a simple solution of Equation 3 in closed form. Because the Frobenius norm is invariant to orthonormal operations, we can re-write Equation 2 as follows:

$$F(M) = \|X'_S X_S M - X'_S X_T\|^2_F = \|M - X'_S X_T\|^2_F \tag{4}$$

From this 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 $X_a = X_S X'_S X_T$. We call X_a the *target aligned source coordinate system.* It is worth noting that if the source and target domains are the same, then $X_S = X_T$

and M^* is the identity matrix.

Matrix M transforms the source subspace coordinate system into the target subspace coordinate system by aligning the source basis vectors with the target ones. If a source basis vector is orthogonal to all target basis vectors, it is ignored. On the other hand, a high weight is given to a source basis vector that is well aligned with the target basis vectors.

2.5 Geodesic Flow Kernel (GFK)

Geodesic Flow Kernel was proposed by [6], The approach consists of the following steps:

1. determine the optimal dimensionality of the subspaces to embed domains.

2. construct the geodesic curve. Let $\mathbf{P_S}, \mathbf{P_T} \in R^{D\times d}$ denote the two sets of basis of the subsapces for the source and target domains. Let $\mathbf{R_S} \in R^{D\times(D-d)}$ denote the orthogonal complement to $\mathbf{P_S}$, namely $\mathbf{R_S}^T\mathbf{P_S} = 0$. Using the canonical Euclidean metric for the Riemannian manifold, the grodesic flow is parameterized as $\Phi : t \in [0, 1] \rightarrow \Phi(t) \in G(d, D)$ under the constraints $\Phi(0) = \mathbf{P_S}$ and $\Phi(1) = \mathbf{P_T}$. For other t,

$$\Phi(t) = \mathbf{P_S}\mathbf{U_1}\Gamma(t) - \mathbf{R_S}\mathbf{U_2}\Sigma(t) \tag{5}$$

   where $U_1 \in R^{d\times d}$ and $U_2 \in R^{(D-d)\times d}$ are orthonormal matrices.

3. compute the geodesic flow kernel. For two original D-dimensional feature vectors x_i and x_j, we compute their projections into $\Phi(t)$ for a continuous t from 0 to 1 and concatenate all the projections into infinite-dimensional feature vectors z_i^∞ and z_j^∞. The inner product between them defines the geodesic-flow kernel,

$$\left\langle z_i^\infty, z_j^\infty \right\rangle = \int_0^1 (\Phi(t)^T x_i)^T (\Phi(t)^T x_j) dt = x_i^T \mathbf{G} x_j \tag{6}$$

4. use the kernel to construct a classifier with labeled data.

2.6 Manifold Embedded Distribution Alignment (MEDA)

There are two significant challenges in existing methods, i.e. degenerated feature transformation and unevaluated distribution alignment. there had been no previous work that tackle these two challenges together until [7]. MEDA learns a domain-invariant classifier in Grassmann manifold with structural risk minimization, while performing dynamic distribution alignment by considering the different importance of marginal and conditional distributions. Also, MEDA is the first attempt to reveal the relative importance of marginal and conditional distributions in domain adaptation.

MEDA consists of two fundamental steps. Firstly, MEDA performs manifold feature learning to address the challenge of degenerated feature transformation. Secondly, MEDA performs dynamic distribution alignment to quantitatively account for the relative importance of marginal and conditional distributions to address the challenge of unevaluated distribution alignment. Eventually, a domaininvariant classifier f can be learned by summarizing these two steps with the principle of SRM. Figure 4 presents the main idea of the proposed MEDA approach.

Formally, if we denote $g(\cdot)$ the manifold feature learning functional, then f can be represented as

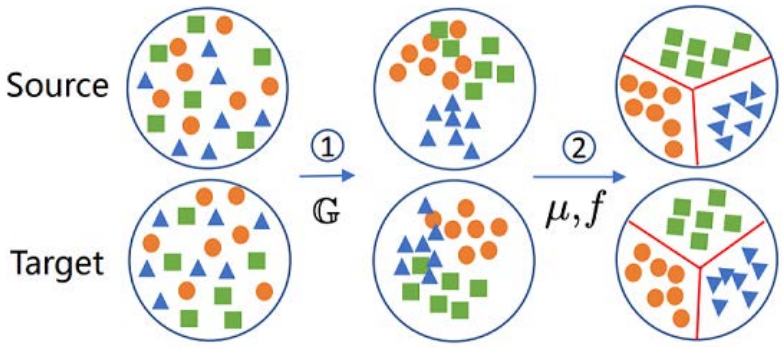

Figure 4. The main idea of MEDA. ① Features in the original space are transformed into manifold space by learning the manifold kernel G. ② Dynamic distribution alignment (by learning μ) with SRM is performed in manifold to learn the final domaininvariant classifier f. .

$$f = \underset{f \in \sum_{i=1}^{n} \mathcal{H}_K}{arg\,min} \; l(f(g(x_i)), y_i) + \eta \|f\|_K^2 + \lambda \bar{D}_f(D_s, D_t) + \rho R_f(D_s, D_t) \tag{7}$$

where $\|f\|_K^2$ is the squared norm of f. The term $\bar{D}_f(\cdot, \cdot)$ represents the proposed dynamic distribution alignment. Additionally, we introduce $R_f(\cdot, \cdot)$ as a Laplacian regularization to further exploit the similar geometrical property of nearest points in manifold G. η, λ, and ρ are regularization parameters accordingly.

2.7 Deep Adaptation Network (DAN)

In the paper [8], it proposex a new Deep Adaptation Network (DAN) architecture, which generalizes deep convolutional neural network to the domain adaptation scenario. In DAN, hidden representations of all task-specific layers are embedded in a reproducing kernel Hilbert space where the mean embeddings of different domain distributions can be explicitly matched. The domain discrepancy is further reduced using an optimal multi-kernel selection method for mean embedding matching. DAN can learn transferable features with statistical guarantees, and can scale linearly by unbiased estimate of kernel embedding.

In unsupervised domain adaptation, we are given a source domain $D_s = \{(x_i^s, y_i^s)\}_{i=1}^{n_s}$ with n_s labeled examples, and a target domain $D_t = \{x_j^t\}_{j=1}^{n_t}$ with n_t unlabeled examples. The source domain and target domain are characterized by probability distributions p and q, respectively. We aim 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 $\epsilon_t(\theta) = Pr_{(x,y)\sim q}[\theta(x) \neq y]$ 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.

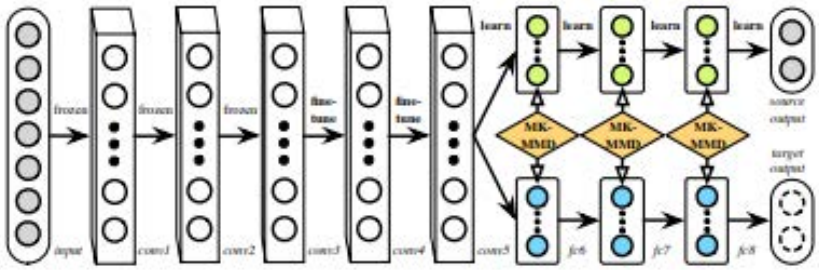

Figure 5. The DAN architecture for learning transferable features. Since deep features eventually transition from general to specific along the network, (1) the features extracted by convolutional layers conv1–conv3 are general, hence these layers are frozen, (2) the features extracted by layers conv4–conv5 are slightly less transferable, hence these layers are learned via fine-tuning, and (3) fully connected layers fc6–fc8 are tailored to fit specific tasks, hence they are not transferable and should be adapted with MK-MMD.
DAN [8] adopts the idea of MK-MMD-based adaptation for learning transferable features in deep networks. The network structure is shown in Figure 5.
.

3 Experiment

3.1 Dataset

Office-Home dataset is a domain adaptation dataset. It contains four different domains: ” Art” , ” Clipart” , ” Product” and ” RealWorld” . Each domain evolves 65 same categories of objects with different style and more than 1000 images.

Instead of using the images as the dataset, we use a extracted 2048-dimension resnet feature as the raw data. It contains the features extracted by the model trained on its own domain and the feature extracted by the model trained on other domains. Each line in the files consists a 2048-dimension feature and a label.

In this project, we only conduct the adaptations of: Art→Real World, Clipart→Real World and Product→Real World. Therefore we use files: *Art_Art.csv*, *Clipart_Clipart.csv* and *Product_Product.csv* as train set; *Art_RealWorld.csv*, *Clipart_RealWorld.csv* and *Product_RealWorld.csv* as test set.

3.2 Baseline Experiment

In order to be a baseline comparing with the following experiments, we conduct SVM and 1NN(KNN algorithm with K=1) algorithms on the original datasets. Take the *”Art”→”RealWorld”* domain as example, we use the features and labels in file *Art_Art.csv* as the training data and use the features and labels in file *Art_RealWorld.csv* as the testing data to examine the trained model. We repeat this experiment three times for different domains. The results are list in table 1.

In our baseline experiment, the time difference between the two methods is not much. However, in terms of accuracy, SVM is about 3-5 percentage points higher than 1NN in all cases. However, considering the difference between different algorithms and the nature of the classification model, we will still try to use two classification methods in the later experiments.

Table 1: Baseline for different cases for domain adaptation.

	Accuracy(%)			times(s)		
Method	A_R	C_R	P_R	A_R	C_R	P_R
SVM	**0.68687**	**0.64371**	**0.74219**	48.9	**89.8**	**79.2**
1NN	0.65817	0.58770	0.69582	**48.8**	90.1	90.0

3.3 KMM

First of all, we will show our accuracy and the result and training time(table 2). We use SVM with "rbf" kernel and 1.0 as the penalty parameter C. And the kernel in KMM algorithm is 'rbf'.

Table 2: Comparison between KMM and baseline.

	Accuracy(%)			times(s)		
Method	A_R	C_R	P_R	A_R	C_R	P_R
KMM	0.66575	**0.64738**	0.73669	50.3	107.2	91.2
Baseline	**0.68687**	0.64371	**0.74219**	**48.9**	**89.8**	**79.2**

It can be seen that the running time of KMM is relatively longer, but its accuracy performance decreases, at least not increases significantly. In order to explore the reasons, we show the spatial distribution of source domain and target domain before and after KMM algorithm by using t-sne, and the reasons are immediately obvious.(figure 6 & 7)

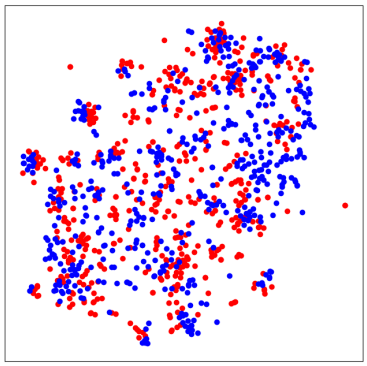

Figure 6: Before KMM.

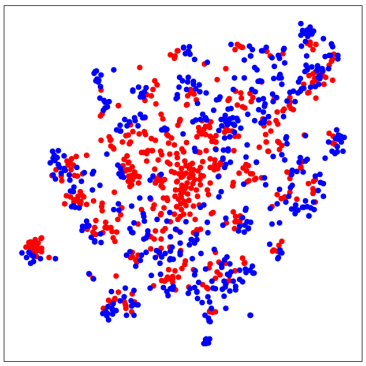

Figure 7: After KMM.

In the figure, the red points represent the distribution of the source domain, and the blue points represent the distribution of the target domain. According to the image, we try to put forward several reasons for the phenomenon that KMM algorithm does not improve performance.

- KMM algorithm uses kernel function to shorten the distance between the mean values of the two

distributions in high-dimensional space. But this does not necessarily have a positive impact on the coincidence of the entire distribution.

- The weight we give to each source domain vector is one-dimensional, that is, each dimension of the vector is multiplied by the same coefficient, which does not necessarily lead to better optimization results.
- After weighting, the whole envelope of data is distorted, which makes the original overlapped sample points separate, so the accuracy is reduced.

Generally speaking, KMM, as a kind of traditional transfer learning, takes too few distribution characteristics into account, which results in poor fitting results.

Next, we hope to explore the shape of the weight parameter β calculated in KMM. Take the *beta* trained by "Art" domain as example. "Art" domain has 2426 samples, therefore the trained β is a 2426-dimension vector. We use *matplotlib* library to plot the vector into a bar chart. (figure 8)

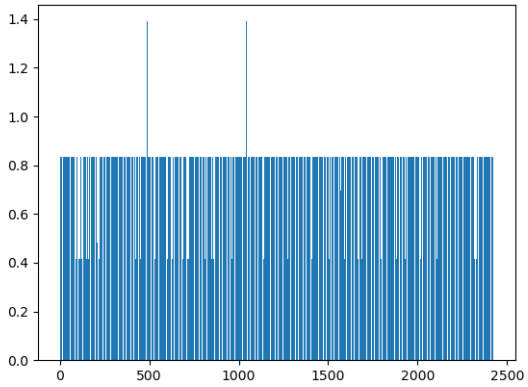

Figure 8: Schematic diagram for the values of β.

The result is quite unexpected. From the graph we can see that almost all dimensions have values of about 0.4 and 0.82. Very few values exceed 1 and reach about 1.4. Considering that the β value will be used as the penalty weight of SVM, most of the source domain points are consistent with the distribution in the target domain, so they do not need to be punished more. This corresponds to the fact that most values in β are less than 1.

3.4 CORAL

We then have a test on CORAL algorithm. This is a projection method that consider the second-order information. By aligning the covariance matrix of two different domain, the spatial distribution of data will be closer. Using the projected matrix to train the model can make the model more adaptable in the target domain. The results are listed in table 3.

In this experiment, we do not record the time needed for model training and testing, but test the accuracy of the algorithm with SVM and 1NN as models respectively. The results in the table show that for SVM algorithm, except the slightly reduced migration from the product to the real world, in

Table 3: Comparison between CORAL and baseline.

	SVM(%)			1NN(%)		
Method	A_R	C_R	P_R	A_R	C_R	P_R
CORAL	**0.73669**	**0.68251**	0.74018	0.64853	0.58150	0.68503
Baseline	0.68687	0.64371	**0.74219**	**0.65817**	**0.58770**	**0.69582**

other cases, CORAL algorithm has greatly improved the original accuracy. However, in the 1NN model, all the accuracies have a small decline.

Our interpretation of this result is as follows:

- When projecting the data, CORAL algorithm only aims at the overall distribution and ignores the fine-grained inter class distribution.
- When all kinds of fine-grained samples deform and shift along with the whole, the trend of gap between classes will also change, which affects the generation of training model.
- As a model with high robustness and error tolerance, SVM has the ability to maintain good performance and give full play to the ability of CORAL algorithm.

Using the similar methods used before, we show the spatial distribution of the two domains before and after CORAL. The art domain is used here, and the result is processed after LDA and t-SNE. The reason for using LDA is that in the previous experiments we found that the data visualization directly is too scattered. After the dimension reduction of LDA, the fine-grained inter class features of data can be distinguished to achieve the purpose of more detailed observation. (figure 9-10)

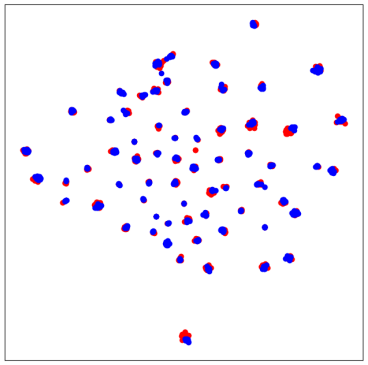

Figure 9: Before CORAL.

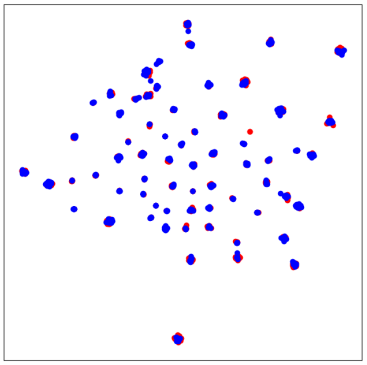

Figure 10: After CORAL.

The red dot in the figure is the source domain ("Art"), and the red dot is the target domain ("RealWorld"). It can be seen that there are 65 clusters in both images, which exactly correspond to 65 object categories in the dataset. This is exactly the effect of LDA, and it can be proved that after CORAL, the inter-class relationship of data is not lost.

Although before CORAL, all kinds of data in different fields have a high degree of coincidence, but after the implementation of the algorithm, the degree of data coincidence is higher. However, this degree of coincidence is difficult to represent the real distribution of data, because many high-dimensional data information is lost in visualization, so the accuracy of our model prediction is still very low.

3.5 Deep CORAL

In order to compare with the CORAL algorithm, we use the same idea of deep CORAL algorithm to carry out a depth model test. We use ResNet18 as the network and use the original images as the dataset. The basic idea of deep CORAL is to train two networks with shared parameters and use CORAL loss as training loss. After 24 hours of 100 epoch training in three domains, we finally get the results.(table 4)

Table 4: Comparison between Deep CORAL, CORAL and baseline.

	Accuracy(%)		
Method	A_R	C_R	P_R
Deep CORAL	0.5472	0.2203	0.6128
CORAL	**0.73669**	**0.68251**	0.74018
Baseline	0.68687	0.64371	**0.74219**

As a deep learning algorithm, deep CORAL should be much more accurate than the traditional algorithm. However, our results are not improved on the basis of CORAL, but lower than baseline. Even on the "Clipart" dataset, the final accuracy is extremely low. Fortunately, we have recorded three indexes of training loss, transfer loss(test loss when transfer the model to different domain) and test accuracy in the process of network training, which gives us a chance to find out what is happening.

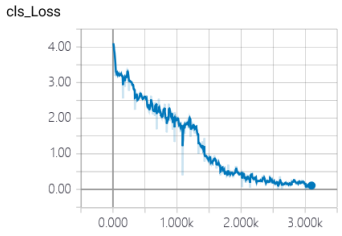

Figure 11: Training loss.

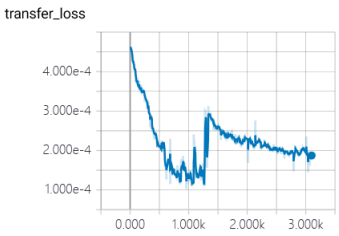

Figure 12: Testing loss.

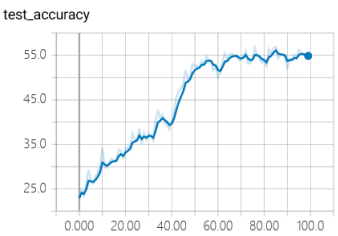

Figure 13: Testing accuracy.

Figure 11-13 is the training process for transfer from "Art" domain to "RealWorld". It can be seen that:

- With the training, the training loss gradually decreases, and the test accuracy generally improves.
- However, when the training loss gradually approaches to zero, the test accuracy does not approach 100%, but remains at about 55%.
- When the training reaches the middle range, the transfer loss suddenly increases in a cliff style, which almost corresponds to the rising stagnation of test accuracy.

From the above phenomena, we analyze that: The training parameters close to 0 indicates that the training is steady and normal. However, the method using CORAL loss as training loss can not make the model perform better in the test domain. This shows that although deep CORAL adopts the new neural network technology, its core idea is still not perfect. Covariance information as a statistical feature does not fit the spatial distribution of data perfectly. As the training goes on, the training with errors finally breaks out, which makes the transfer loss suddenly increase. The over fitting characteristic of neural network makes the performance of depth algorithm even worse than that of traditional CORAL on the basis of traditional ideas.

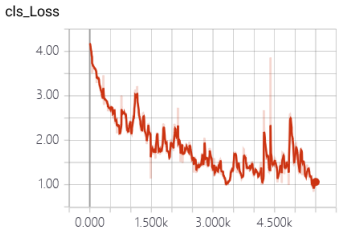

Figure 14: Training loss.

Figure 15: Testing loss.

Figure 16: Testing accuracy.

However, the experiment on "Art" domain still has an accuracy of 54%. The experiment on "Clipart" even much worse. Similarly we plot the training part to see the training process. The results are displayed in figure 14-15.

In this case, the volatility of training loss becomes larger, which makes it difficult to carry out training steadily. The reason may be that the setting of super parameters such as learning rate is not reasonable, but we think that the poor design of training loss is still the main reason for the poor performance of the model.

3.6 Transfer Component Analysis (TCA)

In the experiment part of TCA, we first try to use different classifier and compare their accuracy, see Table 5.

We can find that although in baseline method svm classifier perform a bit better than knn classifier, svm performs rather poor in TCA. We think the reason may be that after TCA, although same label points are still near but they can not be separated by svm classifier. So in our next experiments of TCA, we will use TCA as our classifier.

Table 5: Classification accuracy with different classifier

classifier	$A \to R$	$C \to R$	$P \to R$
svm(kernel='rbf')	0.2318	0.3314	0.5531
knn(k=1)	0.6268	0.5609	0.6553

For knn classifier, we first work to find the optimal number of nearest neighbor k. So we do an experiment for knn with different k and the results is shown in Figure 17. And finally we think $k = 10$ is a well-performed value.

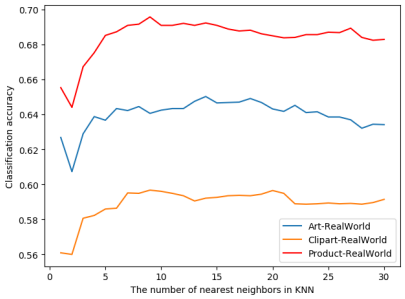

Figure 17: The Classification accuracy with different number of nearest neighbors in KNN (using 'rbf' kernel).

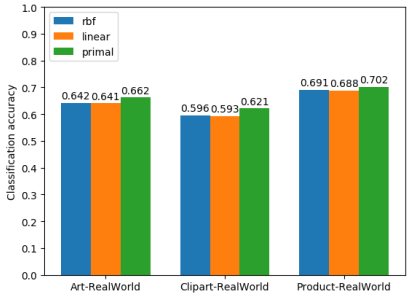

Figure 18: The Classification accuracy with different TCA kernel (using $knn = 10$).

As there are different kernels can be selected for TCA, we also try different kernels for domain adaption ($A \rightarrow R$, $C \rightarrow R$, and $P \rightarrow R$). And the result can be seen in Figure 18.

To check the performance of TCA in domain adaption, we do some result visualization. In Figure 19, we just do PCA to project the data to 30 dimensions and then do t-SNE to 2-d visualization without TCA. And in Figure 20, we do TCA to project the data to 30 dimensions and then do t-SNE to 2-d visualization.

Compared with Figure 19 and Figure 20, we can get that TCA can really reduce the distance between source domain and target domain in some degree and improve the classification accuracy.

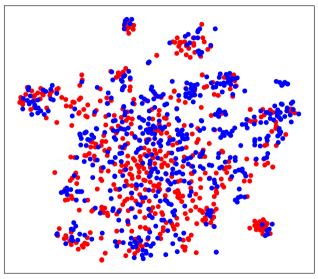

Figure 19: t-SNE 2-d visualization result with PCA to 30 dimensions and no TCA

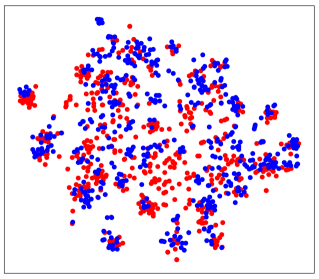

Figure 20: t-SNE 2-d visualization result with TCA to 30 dimensions directly

Our final TCA result is shown in Table 6. But due to the reason of TCA running time, we do not do more experiment for higher dimensions in TCA and so our result of TCA is not very high but acceptable.

3.7 Subspace Alignment (SA)

For the Subspace Alignment (SA), we try to use svm and knn classifier to do domain adaptation on ($A \rightarrow R$, $C \rightarrow R$, and $P \rightarrow R$). We can find that 1-nn classifier can still perform better than svm classifier.

Table 6: Our best domain adaptation result using TCA

	$A \to R$	$C \to R$	$P \to R$
accuracy	0.6622	0.6214	0.7021
time(s)	8756	19648	20645

* In this table, we use $svm(C = 1, kernel =' rbf')$ and $TCA(kernel\_type =' rbf', dim = 30, lamb = 1, gamma = 1)$.

Table 7: Classification accuracy with different classifier using SA

classifier	$A \to R$	$C \to R$	$P \to R$
svm(kernel='rbf')	0.6622	0.5687	0.6383
knn(k=1)	0.6644	0.5938	0.7021

3.8 Geodesic Flow Kernel (GFK)

For Geodesic Flow Kernel (GFK), we just try to use 1-nn classifier for domain adaptation on ($A \to R$, $C \to R$, and $P \to R$). We can find that GFK can get both good and fast running result, see Table 8.

Table 8: Classification accuracy for GFK

	$A \to R$	$C \to R$	$P \to R$
knn(k=1) accuracy	0.6691	0.6024	0.7046
time(s)	8.73	15.24	13.37

3.9 Manifold Embedded Distribution Alignment (MEDA)

For Manifold Embedded Distribution Alignment (MEDA), we use 1-nn classifier for domain adaptation on ($A \to R$, $C \to R$, and $P \to R$).

For MEDA, we do 10 iterations for each adaptation and the iteration result is shown in Figure 21.

And the final optimal classification accuracy for MEDA is shwn in Table 9. From the table, we can see that the classification result is pretty good compared with other methods.

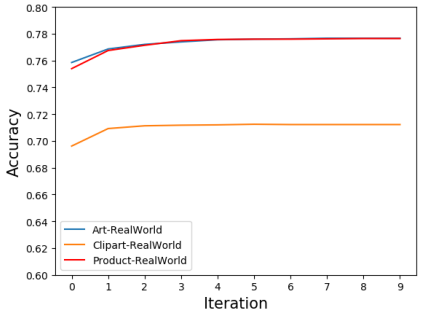

Figure 21: Iteration accuracy for MEDA

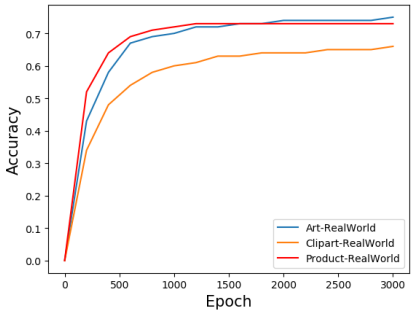

Figure 22: Accuracy along with epoch for DAN

Table 9: Classification accuracy for MEDA

Domain adaptation	Optimal accuracy
$A \rightarrow R$	77.67%
$C \rightarrow R$	71.22%
$P \rightarrow R$	77.65%

3.10 Deep Adaptation Network (DAN)

For Deep Adaptation Network (DAN), we use original images as the model input, and use pre-trained ResNet50 and a DanNet for domain adaptation on ($A \rightarrow R$, $C \rightarrow R$, and $P \rightarrow R$). Figure 22 shows the classification accuracy along with epoch. And Table 10 shows our optimal accuracy for the three domain adaptation tasks. We can find that Deep Adaptation Network can really achieve good result but also need lots of time and computing power.

Table 10: Classification accuracy for DAN

Domain adaptation	Optimal accuracy
$A \rightarrow R$	75.00%
$C \rightarrow R$	66.00%
$P \rightarrow R$	73.00%

4 Summary

In this project, we conduct domain adaptation for image classification in the following three settings ($A \rightarrow R$, $C \rightarrow R$, and $P \rightarrow R$).

For our project, we implement baseline and additional 8 domain adaptation methods (traditional: KMM, CORAL, TCA, SA, GFK, MEDA and deep: Deep CORAL, DAN). And the optimal result is shown in Figure 23.

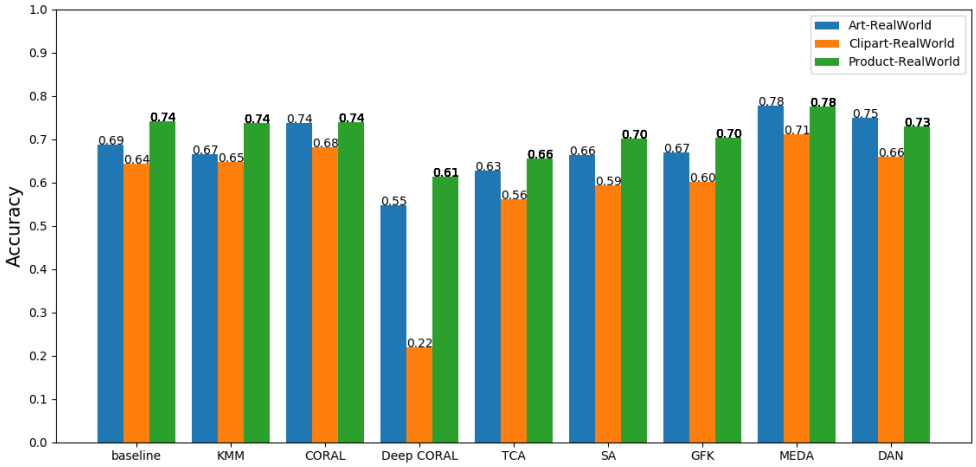

Figure 23. The optimal classification accuracy for our baseline and 8 domain adaptation methods (traditional: KMM, CORAL, TCA, SA, GFK, MEDA and deep: Deep CORAL, DAN) on ($A \rightarrow R$, $C \rightarrow R$, and $P \rightarrow R$) .

From the experiment, we can get that different methods need different running time and computing power and also have different final result. Totally speaking, MEDA performs best among our methods, with highest accuracy and less running time. Other traditional methods can also have not bad result can less running time. And deep methods need more time and computing power in general but higher accuracy. But due to the limitation of time and computing power, we just do 2 deep methods, Deep CORAL and DAN , and not get very high result.

Reference

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

[2] Baochen Sun, Jiashi Feng, and Kate Saenko. Return of frustratingly easy domain adaptation. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

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

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

[5] Basura Fernando, Amaury Habrard, Marc Sebban, and Tinne Tuytelaars. Unsupervised visual domain adaptation using subspace alignment. In *Proceedings of the IEEE international conference on computer vision*, pages 2960–2967, 2013.

[6] Boqing Gong, Yuan Shi, Fei Sha, and Kristen Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2066–2073. IEEE, 2012.

[7] Jindong Wang, Wenjie Feng, Yiqiang Chen, Han Yu, Meiyu Huang, and Philip S Yu. Visual domain adaptation with manifold embedded distribution alignment. In *Proceedings of the 26th ACM international conference on Multimedia*, pages 402–410, 2018.

[8] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *International conference on machine learning*, pages 97–105, 2015.