Project 4 Domain Adaption

Yanjun Fu 516030910354 | Zeren Huang 516030910356 | Hao Sun 516030910362 | Bohong Wu 516030910365

1. Introduction

In this project, we need to conduct the domain adaptation experiments in different source and target domains. We pick up **ten** traditional methods and **eleven** deep methods to compare and improve. Specially, we have also tried GAN based method. At last we give the accuracy rank among all listed methods and got the conclusion that MEDA performs the best.

2. Traditional methods

Let $\mathcal{P}(X_S)$ and $\mathcal{Q}(X_T)$ (or $\mathcal{P}$ and $\mathcal{Q}$ in short) be the marginal distributions of $X_S = \{x_{S_i}\}$ and $X_T = \{x_{T_i}\}$ from the source and target domains, respectively. In general, $\mathcal{P}$ and $\mathcal{Q}$ can be different. Our task is to predict the labels $y_{T_i}s$ corresponding to inputs $x_{T_i}s$ in the target domain. The key assumption in most adaptation methods i that $\mathcal{P} \neq \mathcal{Q}$, but $P(Y_S|X_S) = P(Y_T|X_T)$.

2.1. Transfer Component Analysis (TCA)

Most domain adaptation methods assume the $\mathcal{P} \neq \mathcal{Q}$ but $P(Y_S|X_S) = P(Y_T|X_T)$ until [13]. In fact, in many real-world applications, the conditional probability $P(Y|X)$ may also change across domains due to noisy or dynamic factors underlying the observed data. In this paper, we use the weaker assumption that $\mathcal{P} \neq \mathcal{Q}$, but there exists a transformation ϕ such that $P(\phi(X_S)) \approx P(\phi(X_T))$ and $P(Y_S|\phi(X_S)) \approx P(Y_T|\phi(X_T))$.

Given the source domain data set $\mathcal{D}_S = \{(x_{S_i}, y_{src_i})\}_{i=1}^{n_1}$, and target domain data set $\mathcal{D}_T = \left\{x_{T_j}\right\}_{j=1}^{n_2}$. The method steps of computing the transformation matrix W are as follows.

1. construct kernel matrix K from $\{x_{S_i}\}_{i=1}^{n_1}$ and $\{x_{T_i}\}_{j=1}^{n_2}$ based on

$$K = \begin{bmatrix} K_{S,S} & K_{S,T} \\ K_{T,S} & K_{T,T} \end{bmatrix} \in \mathbb{R}^{(n_1+n_2)\times(n_1+n_2)} \quad (1)$$

Construct the matrix L from

$$\max_{K\succeq 0} \operatorname{tr}(KL) - \lambda \operatorname{tr}(K) \text{ subject to constraints on } K \quad (2)$$

2. Eigendecompose the matrix

$$(K(L+\lambda)\mathcal{L}K + \mu I)^{-1} K H \widetilde{K}_{yy} H K$$

Then select the m leading eigenvectors to construct the transformation matrix W.

The kernel learning method requires only a simple and efficient eigenvalue decomposition. This takes only $O\left(m\left(n_1+n_2\right)^2\right)$ time when m nonzero eigenvectors are to be extracted.

2.2. Geodesic Flow Kernel (GFK)

Geodesic Flow Kernel was proposed by [5], The approach consists of the following steps:

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

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

$$\boldsymbol{\Phi}(t) = \boldsymbol{P}_S \boldsymbol{U}_1 \boldsymbol{\Gamma}(t) - \boldsymbol{R}_{\mathcal{S}} \boldsymbol{U}_2 \boldsymbol{\Sigma}(t) \quad (3)$$

where $U_1 \in \mathbb{R}^{d\times d}$ and $U_2 \in \mathbb{R}^{(D-d)\times d}$ are orthonormal matrices. They are given by the following pair of SVDs.

$$\boldsymbol{P}_{\mathcal{S}}^{\mathrm{T}}\boldsymbol{P}_{\mathcal{T}} = \boldsymbol{U}_1\boldsymbol{\Gamma}\boldsymbol{V}^{\mathrm{T}}, \quad \boldsymbol{R}_{\mathcal{S}}^{\mathrm{T}}\boldsymbol{P}_{\mathcal{T}} = -\boldsymbol{U}_2\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}} \quad (4)$$

$\boldsymbol{\Gamma}$ and $\boldsymbol{\Sigma}$ are $d \times d$ diagonal matrices. The diagonal elements are $\cos\theta_i$ and $\sin\theta_i$ for $i = 1, 2, \cdots, d$. Particularly, θ_i are called the principal angles between P_S and P_T:

$$0 \leq \theta_1 \leq \theta_2 \leq \cdots \leq \theta_{\mathrm{d}} \leq \pi/2 \quad (5)$$

3. Compute the geodesic flow kernel. For two original D-dimensional feature vectors x_i and x_j, we compute

their projections into $\mathbf{\Phi}(t)$ 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,

$$\left\langle \boldsymbol{z}_i^\infty, \boldsymbol{z}_j^\infty \right\rangle = \int_0^1 \left(\mathbf{\Phi}(t)^{\mathrm{T}} \boldsymbol{x}_i\right)^{\mathrm{T}} \left(\mathbf{\Phi}(t)^{\mathrm{T}} \boldsymbol{x}_j\right) dt = \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{G} \boldsymbol{x}_j \tag{6}$$

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

2.3. Subspace Alignment (SA)

Subspace Alignment was proposed by [3] for the situation where the source and target domains are represented by subspaces spanned by eigenvectors. SA seeks a domain invariant featre space by learning a mapping function which aligns the source subspace with the target one. The algothm step is as follows.

1. Transform every source and target data to a D-dimensional z-normalized vector.

2. Using PCA select for each domain the d eigenvectors corresponding to the d largest eigenvalues. These eigenvectors are used as bases of the source and target subspaces.

3. 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 by the operations $\mathbf{y_S}X_S$ and $\mathbf{y_T}X_T$, respectively.

4. Learn a linear transformation that maps the source subspace to the target one. To achieve this task, align basis vectors by using a transformation matrix $\mathbf{M}$ from X_S to $X_T (M \in \mathbb{R}^{d\times d})$. $\mathbf{M}$ is learned by minimizing the following Bregman matrix divergence:

$$F(M) = \left\| X_{SF(M)=\|X_S M - X_T\|_F^2} M - X_T \right\|_F^2 \tag{7}$$

$$M^* = \mathrm{argmin}_M(F(M)) \tag{8}$$

5. Define $Sim(\mathbf{y_S}, \mathbf{y_T})$ as follows:

$$\begin{aligned} \mathrm{sim}\left(\mathbf{y_S}, \mathbf{y_T}\right) &= \left(\mathbf{y_S} X_S \mathbf{M}^*\right)\left(\mathbf{y_T} X_T\right)' \\ &= \mathbf{y_S} X_S \mathbf{M}^* X_T' \mathbf{y_T} \\ &= \mathbf{y_S} \mathbf{A} \mathbf{y}\mathbf{T}' \end{aligned} \tag{9}$$

where $\mathbf{A} = X_S X_S' X_T X_T'$.

6. Use the matrix $\mathbf{A}$ to construct the kernel matrices via $Sim(\mathbf{y_S}, \mathbf{y_T})$ and perform SVM-based classification.

2.4. Importance Weighted (IW) Classifier

Importance weighted classifier assign each sample weight which represent its "importance", then it make the classifier fit the distribution of target domain better. Importance weighted classifier includes a series of methods, we use Kernel Mean Matching (KMM) [7] as an example.

In general, the estimation problem with two different distributions $\Pr(x, y)$ and $\Pr'(x, y)$ are unsolvable, as the two terms could be arbitrarily far apart. In particular, for arbitrary $\Pr(x, y)$ and $\Pr'(x, y)$, there is no way we could infer a good estimator based on the training sample. Hence KMM makes the simplifying assumption that $\Pr(x, y)$ and $\Pr'(x, y)$ only differ via $\Pr(x, y) = \Pr(y|x)\Pr(x)$ and $\Pr(y|x)\Pr'(x)$. In other words, the conditional probabilities of $y|x$ remain unchanged (this particular case of sample selection bias has been termed covariate shift).

KMM begins by stating the problem of regularized risk minimization. In general a learning method minimizes the expected risk

$$R[\Pr, \theta, l(x, y, \theta)] = \mathbf{E}_{(x,y)\sim\Pr}[l(x, y, \theta)] \tag{10}$$

of a loss function $l(x, y, \theta)$ that depends on a parameter θ. However, since typically we only observe examples (x, y) drawn from $Pr(x, y)$ rather than $\Pr(x, y)$, KMM resorts to computing the empirical average

$$R_{\mathrm{emp}}[Z, \theta, l(x, y, \theta)] = \frac{1}{m}\sum_{i=1}^{m} l\left(x_i, y_i, \theta\right) \tag{11}$$

To avoid overfitting, instead of minimizing R_{emp} directly KMM often minimizes a regularized variant $R_{\mathrm{reg}}[Z, \theta, l(x, y, \theta)] := R_{\mathrm{emp}}[Z, \theta, l(x, y, \theta)] + \lambda\Omega[\theta]$ where $\Omega[\theta]$ is a regularizer.

The problem is more involved if $\Pr(x, y)$ and $\Pr'(x, y)$ are different. The training set is drawn from $\Pr$, however what we would really like is to minimize $R\left[\Pr', \theta, l\right]$ as we wish to generalize to test examples drawn from $\Pr$. An observation from the field of importance sampling is that

$$\begin{aligned} R\left[\Pr', \theta, l(x, y, \theta)\right] &= \mathbf{E}_{(x,y)\sim\Pr'}[l(x, y, \theta)] \\ &= \mathbf{E}_{(x,y)\sim\Pr}\left[\underbrace{\frac{\Pr'(x, y)}{:=\beta(x,y)}} l(x, y, \theta)\right] \\ &= R[\Pr, \theta, \beta(x, y) l(x, y, \theta)], \end{aligned} \tag{12}$$

provided that the support of $\Pr$ is contained in the support of $\Pr$. Given $\beta(x, y)$, we can thus compute the risk with respect to $\Pr$ using $\Pr$. Similarly, we can estimate the risk with respect to $\Pr'$ by computing $R_{\mathrm{emp}}[Z, \theta, \beta(x, y) l(x, y, \theta)]$.

2.5. Joint Distribution Adaptation (JDA)

[10] proposed transfer featrue learning with Joint Distribution Adaptation. In JDA, to achieve effective and robust transfer learning, we aim to simultaneously minimize the differences in both the marginal distributions and conditional distributions across domains. The JDA optimization problem is

$$\min_{\mathbf{A}^{\tau}\mathbf{X}\mathbf{H}\mathbf{X}^{T}\mathbf{A}=\mathbf{I}} \sum_{c=0}^{C} \operatorname{tr}\left(\mathbf{A}^{\mathrm{T}}\mathbf{X}\mathbf{M}_{c}\mathbf{X}^{\mathrm{T}}\mathbf{A}\right)+\lambda\|\mathbf{A}\|_{F}^{2} \quad (13)$$

Where λ is the regularization parameter to guarantee the optimization problem to be well-defined. Based on the generalizaed Raleigh quotient, It is one thing of minimizing

$$\operatorname{tr}\left(\mathbf{A}^{\mathrm{T}}\mathbf{X}\mathbf{M}_{0}\mathbf{X}^{\mathrm{T}}\mathbf{A}\right) \quad (14)$$

$$\operatorname{tr}\left(\mathbf{A}^{\mathbf{T}}\mathbf{X}\mathbf{M}_{c}\mathbf{X}^{\mathbf{T}}\mathbf{A}\right) \quad (15)$$

such that whether $\max_{\mathbf{A}^{\tau}\mathbf{A}=\mathbf{I}} \operatorname{tr}\left(\mathbf{A}^{\mathrm{T}}\mathbf{X}\mathbf{H}\mathbf{X}^{\mathrm{T}}\mathbf{A}\right)$ is maximized or is fixed.

The JDA algorithm is as follows,

1. Construct MMD matrix M_0 by

$$(M_0)_{ij}=\begin{cases}\frac{1}{n_s n_s}, & \mathbf{x}_i,\mathbf{x}_j\in\mathcal{D}_s\\ \frac{1_1}{n_s n_t}, & \mathbf{x}_i,\mathbf{x}_j\in\mathcal{D}_t\\ \frac{-1}{n_s n_t}, & \text{otherwise}\end{cases} \quad (16)$$

   set $\{\mathbf{M}_c:=\mathbf{0}\}_{c=1}^{C}$.

2. solve the generalize eigendecomposition problem to

$$\mathbf{X}\mathbf{H}\mathbf{X}^{\mathrm{T}}\mathbf{A}\mathbf{\Phi} \quad (17)$$

   and select the k smallest eigenvectors to construct the adaptation matrix $\mathbf{A}$, and $\mathbf{Z}:=\mathbf{A}^{\mathrm{T}}\mathbf{X}$.

3. Train a standard classifier f on $\left\{\left(\mathbf{A}^{\mathrm{T}}\mathbf{x}_i,y_i\right)\right\}_{i=1}^{n_s}$ to update pseudo target label $\left\{\widehat{y}_j:=f\left(\mathbf{A}^{\mathrm{T}}\mathbf{x}_j\right)\right\}_{j=n_s+1}^{n_s+n_t}$.

4. Construct MMD metrices $\{\mathbf{M}_c\}_{c=1}^{C}$ by

$$(M_c)_{ij}=\begin{cases}\frac{1}{n_s^{(c)}n_s^{(c)}}, & \mathbf{x}_i,\mathbf{x}_j\in\mathcal{D}_s^{(c)}\\ \frac{1_1}{n_s^{(c)}n_t^{(c)}}, & \mathbf{x}_i,\mathbf{x}_j\in\mathcal{D}_t^{(c)}\\ \frac{-1}{n_s^{(c)}n_t^{(c)}}, & \begin{cases}\mathbf{x}_i\in\mathcal{D}_s^{(c)},\mathbf{x}_j\in\mathcal{D}_t^{(c)}\\ \mathbf{x}_j\in\mathcal{D}_s^{(c)},\mathbf{x}_i\in\mathcal{D}_t^{(c)}\end{cases}\\ 0, & \text{otherwise}\end{cases} \quad (18)$$

5. Go back to step 2 until convergence. Then we get an adaptive classifier f trained on $\{\mathbf{A}\mathbf{x}_i,y_i\}_{i=1}^{n_s}$.

It is obvious that TCA can be viewed as a special case of JDA with $C=0$. With JDA, we can simultaneously adapt both the marginal distributions and conditional distributions between domains to facilitate joint distribution adaptation. A fascinating property of JDA is its capability to effectively explore the conditional distributions only using principled unsupervised dimensionality reduction and base classifier. Thus JDA can be easy to implement and deploy practically.

2.6. Transfer Joint Matching (TJM)

[11] proposes Transfer Joint Matching. TJM complete procedure is as follow.

1. Compute MMD matrix $\mathbf{M}$ by eq. (16), and kernel matrix $mathbfK$ by $K_{ij}\leftarrow K\left(\mathbf{x}_i,\mathbf{x}_j\right)$ where $K(\cdot,\cdot)$ is a predefined kernel. Set $\mathbf{M}\leftarrow\mathbf{M}/\|\mathbf{M}\|_F,\mathbf{G}\leftarrow\mathbf{I}$.

2. Solve the generalized eigendecomposition problem $\mathbf{K}\mathbf{H}\mathbf{K}^{\mathrm{T}}\mathbf{A}\mathbf{\Phi}$ and select the k smallest eigenvectors to construct the adaptation matrix A, and $\mathbf{Z}\leftarrow\mathbf{A}^{\mathrm{T}}\mathbf{K}$.

3. Update the sub-gradient matrix G by

$$G_{ii}=\begin{cases}\frac{1}{2\|\mathbf{a}^i\|}, & \mathbf{x}_i\in\mathcal{D}_s,\mathbf{a}^i\neq\mathbf{0}\\ 0, & \mathbf{x}_i\in\mathcal{D}_s,\mathbf{a}^i=\mathbf{0}\\ 1, & \mathbf{x}_i\in\mathcal{D}_t\end{cases} \quad (19)$$

4. Go back to step 2 until convergence. At last we get an adaptive classifier f trained on $\left\{\mathbf{A}^{\mathrm{T}}\mathbf{k}_i,y_i\right\}_{i=1}^{n_s}$.

2.7. Balanced Distribution Adaptation (BDA)

Transfer learning methods often seek to adapt both the marginal and conditional distributions between domains. Specifically, this refers to minimizing the distance

$$\begin{aligned}D\left(\mathcal{D}_s,\mathcal{D}_t\right)\approx & D\left(P\left(\mathbf{x}_s\right),P\left(\mathbf{x}_t\right)\right)\\ & +D\left(P\left(y_s|\mathbf{x}_s\right),P\left(y_t|\mathbf{x}_t\right)\right)\end{aligned} \quad (20)$$

However, simply matching both distributions is not enough. Existing methods usually assume they are equally important, and that implicit assumption does not hold. [18] was proposed to adapively adjust the importance of both the marginal and conditional distributions based on each specific tasks. Concretely speaking, BDA exploits a balance factor μ to leverage the different importance of distributions:

$$\begin{aligned}D\left(\mathcal{D}_s,\mathcal{D}_t\right)\approx & (1-\mu)D\left(P\left(\mathbf{x}_s\right),P\left(\mathbf{x}_t\right)\right)\\ & +\mu D\left(P\left(y_s|\mathbf{x}_s\right),P\left(y_t|\mathbf{x}_t\right)\right)\end{aligned} \quad (21)$$

where $\mu\in[0,1]$. The whole algorithm of BDA is as follow.

1. Train a base classifier on $\mathbf{X}_s$ and apply prediction on $\mathbf{X}_t$ to get its soft labels $\hat{\mathbf{y}}_t$.

2. Construct $\mathbf{X} = [\mathbf{X}_s, \mathbf{X_t}]$, initialize M_0 and M_c by eq. (16) and eq. (18). Specially, another method W-BDA based on BDA has different M_c which is defined as

$$\begin{cases} \frac{P\left(y_s^{(c)}\right)}{n_c^2}, & \mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}_s^{(c)} \\ \frac{P\left(y_t^{(c)}\right)}{m_c^2}, & \mathbf{x}_i, \mathbf{x}_j \in \mathcal{D}_t^{(c)} \\ -\frac{\sqrt{P\left(y_s^{(c)}\right)P\left(y_t^{(c)}\right)}}{m_c n_c}, & \begin{cases} \mathbf{x}_i \in \mathcal{D}_s^{(c)}, \mathbf{x}_j \in \mathcal{D}_t^{(c)} \\ \mathbf{x}_j \in \mathcal{D}_s^{(c)}, \mathbf{x}_i \in \mathcal{D}_t^{(c)} \end{cases} \\ 0, & \text{otherwise} \end{cases} \tag{22}$$

3. Solve the eigendecomposition problem in $\mathbf{XHX}^\top\mathbf{A}\Phi$ or for W-BDA,

$$\begin{aligned} \min \operatorname{tr}\left(\mathbf{A}^\top\mathbf{X}\left((1-\mu)\mathbf{M}_0 + \mu\textstyle\sum_{c=1}^{C}\mathbf{W}_c\right)\mathbf{X}^\top\mathbf{A}\right) \\ +\lambda\|\mathbf{A}\|_F^2 \\ \text{s.t. } \mathbf{A}^\top\mathbf{XHX}^\top\mathbf{A} = \mathbf{I}, \quad 0 \le \mu \le 1 \end{aligned} \tag{23}$$

use d smallest eigenvectors to build $\mathbf{A}$

4. Train a classifier f on $\left\{\mathbf{A}^\top\mathbf{X}_s, \mathbf{y}_s\right\}$

5. Update the soft labels of $\mathcal{D}_t : \hat{\mathbf{y}}_t = f\left(\mathbf{A}^\top\mathbf{X}_t\right)$

6. Update matrix $\mathbf{M}_c$ using eq. (18) or eq. (23)

7. Go back step 2 until convergence. At last we get the classifier f.

2.8. Correlation Alignment (CORAL)

Correlation Alignment was presented by [14], which is a simple yet effect method for unsupervised domain adaptation.CORAL minimizes domain shift by aligning the second-order statistics of source and target distributions, without requiring any target labels. In contrast to subspace manifold methods, it aligns the original feature distributions of the source and target domains, rather than the bases of lower-dimensional subspaces. It is also much simpler than other distribution matching methods. The algorithm steps of CORAL are shown as fig. 1.

2.9. 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 [19]. 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.

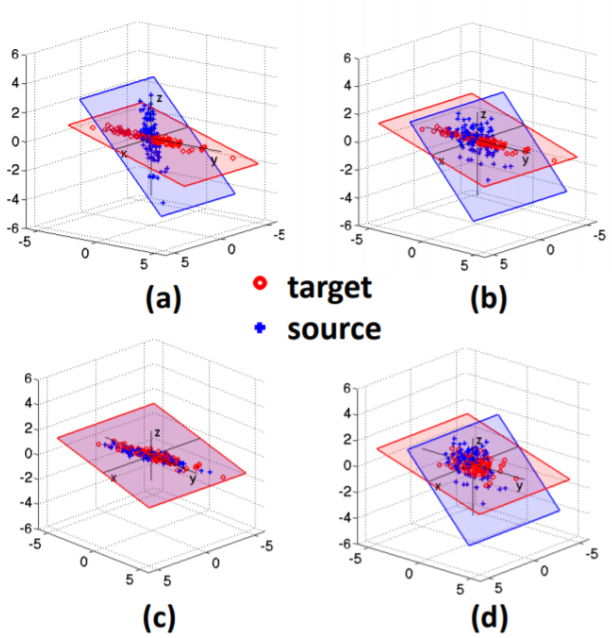


Figure 1. Illustration of CORAL for Domain Adaptation: (a) The original source and target domains have different distribution covariances, despite the features being normalized to zero mean and unit standard deviation. This presents a problem for transferring classifiers trained on source to target. (b) The same two domains after source decorrelation, i.e.removing the feature correlations of the source domain. (c) Target re-correlation, adding the correlation of the target domain to the source features. After this step, the source and target distributions are well aligned and the classifier trained on the adjusted source domain is expected to work well in the target domain. (d) One might instead attempt to align the distributions by whitening both source and target. However, this will fail since the source and target data are likely to lie on different subspaces due to domain shift.

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 domain-invariant classifier f can be learned by summarizing these two steps with the principle of SRM. fig. 2 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

$$\begin{aligned} f = \underset{f \in \sum_{i=1}^{n}\mathcal{H}_K}{\arg\min} \; & \ell\left(f\left(g\left(\mathbf{x}_i\right)\right), y_i\right) + \eta\|f\|_K^2 \\ & + \lambda\overline{D_f}\left(\mathcal{D}_s, \mathcal{D}_t\right) + \rho R_f\left(\mathcal{D}_s, \mathcal{D}_t\right) \end{aligned} \tag{24}$$

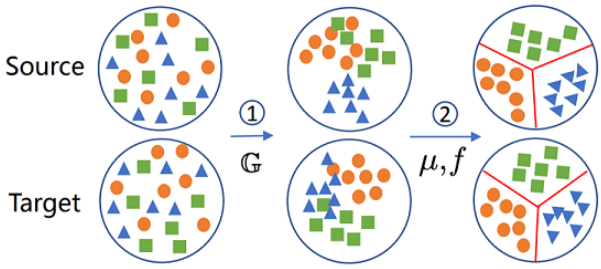


Figure 2. The main idea of MEDA. ①Features in the original space are transformed into manifold space by learning the manifold kernel $\mathbb{G}$. ②Dynamic distribution alignment (by learning μ) with SRM is performed in manifold to learn the final domain-invariant classifier f.

where $\|f\|_K^2$ is the squared norm of f. The term $\overline{D_f}(\cdot,\cdot)$ 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 $\mathbb{G}$. η, λ, and ρ are regularization parameters accordingly.

3. Deep domain adaptation methods

3.1. Deep Domain Confusion (DDC)

The intuition of the DDC [16] model is that if we can learn a representation that minimizes the distance between the source and target distributions, then we can train a classifier on the source labeled data and directly apply it to the target domain with minimal loss in accuracy.

To minimize this distance, we can consider the standard distribution distance metric, Maximum Mean Discrepancy (MMD). This distance is computed with respect to a particular representation, $\phi(\cdot)$. In our case, we define a representation, $\phi(\cdot)$, which operates on source data points, $x_s \in X_S$ and target data points, $x_t \in X_T$. Then an empirical approximation to this distance is computed as followed:

$$\begin{aligned}&\mathrm{MMD}\left(X_S, X_T\right) = \\ &\left\| \frac{1}{|X_S|} \sum_{x_s \in X_S} \phi\left(x_s\right) - \frac{1}{|X_T|} \sum_{x_t \in X_T} \phi\left(x_t\right) \right\|\end{aligned} \tag{25}$$

Not only do we want to minimize the distance between domains (or maximize the domain confusion), but we want a representation which is conducive to training strong classifiers. Such a representation would enable us to learn strong classifiers that readily transfer across domains. One approach to meeting both these criteria is to minimize the loss:

$$\mathcal{L} = \mathcal{L}_C\left(X_L, y\right) + \lambda \,\mathrm{MMD}^2\left(X_S, X_T\right) \tag{26}$$

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

The network architecture, which is shown in fig. 3, consists of a source and target CNN, with shared weights. Only the labeled examples are used to compute the classification loss, while all data is used from both domains to compute the domain confusion loss. The network is jointly trained on all available source and target data.

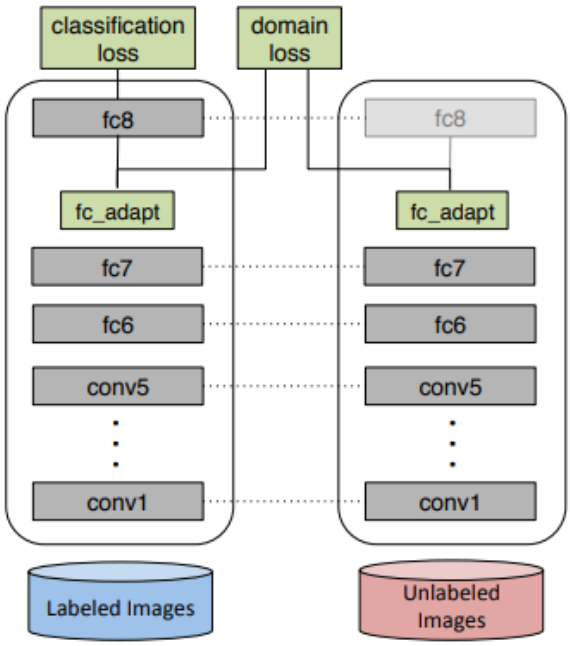


Figure 3. The architecture of DDC optimizes a deep CNN for both classification loss as well as domain invariance. The model can be trained for supervised adaptation, when there is a small amount of target labels available, or unsupervised adaptation, when no target labels are available. We introduce domain invariance through domain confusion guided selection of the depth and width of the adaptation layer, as well as an additional domain loss term during fine-tuning that directly minimizes the distance between source and target representations

3.2. Deep Adaptation Networks (DAN)

Denote by $\mathcal{H}_k$ be the reproducing kernel Hilbert space (RKHS) endowed with a characteristic kernel k. The mean embedding of distribution p in $\mathcal{H}_k$ is a unique element $\mu_k(p)$ such that $\mathbf{E}_{\mathbf{x}\sim p} f(\mathbf{x}) = \langle f(\mathbf{x}), \mu_k(p)\rangle_{\mathcal{H}_k}$, $\mu_k(p)\rangle_{\mathcal{H}_k}$ for all $f \in \mathcal{H}_k$. The MK-MMD dk (p, q) between probability distributions p and q is defined as the RKHS distance between the mean embeddings of p and q. The squared formulation of MK-MMD is defined as

$$d_k^2(p,q) \triangleq \left\| \mathbf{E}_p\left[\phi\left(\mathbf{x}^s\right)\right] - \mathbf{E}_q\left[\phi\left(\mathbf{x}^t\right)\right] \right\|_{\mathcal{H}_k}^2 \quad (27)$$

DAN [9] adopts the idea of MK-MMD-based adaptation for learning transferable features in deep networks.

In standard CNNs, deep features must eventually transition from general to specific by the last layer of the network, and the transfer ability gap grows with the domain discrepancy and becomes particularly large when transferring the higher layers $fc6 - fc8$. In other words, the fc layers are tailored to their original task at the expense of degraded performance on the target task, hence they cannot be directly transferred to the target domain via fine-tuning with limited target supervision. In DAN, the CNNC is fine-tuned on the source labeled examples and require the distributions of the source and target to become similar under the hidden representations of fully connected layers $fc6fc8$. This can be realized by adding an MK-MMD-based multi-layer adaptation regularizer to the CNN risk.

The network structure is shown in fig. 4.

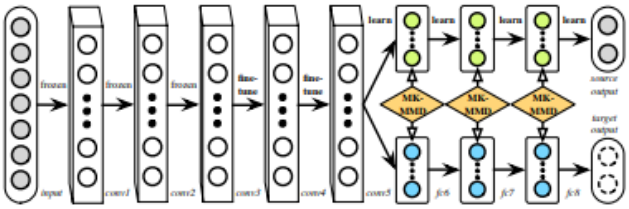


Figure 4. The DAN architecture for learning transferable features. Since deep features eventually transition from general to specific along the network, (1) the features extracted by convolutional layers $conv1conv3$ are general, hence these layers are frozen, (2) the features extracted by layers $conv4conv5$ are slightly less transferable, hence these layers are learned via fine-tuning, and (3) fully connected layers $fc6fc8$ are tailored to fit specific tasks, hence they are not transferable and should be adapted with MK-MMD

$$\min_{\Theta} \frac{1}{n_a} \sum_{i=1}^{n_a} J\left(\theta\left(\mathbf{x}_i^a\right), y_i^a\right) + \lambda \sum_{\ell=l_1}^{l_2} d_k^2\left(\mathcal{D}_s^\ell, \mathcal{D}_t^\ell\right) \quad (28)$$

where $\lambda > 0$ is a penalty parameter, $l1$ and $l2$ are layer indices between which the regularizer is effective.

3.3. Joint Adaptation Networks (JAN)

In unsupervised domain adaptation, we are given a source domain $\mathcal{D}_s = \{(\mathbf{x}_i^s, \mathbf{y}_i^s\}_{i=1}^{n_s}$ of n_s labeled examples and a target domain $\mathcal{D}_t = \{\mathbf{x}_j^t\}_{j=1}^{n_t}$ of n_t unlabeled examples. The source domain and target domain are sampled from joint distribution $P(\mathbf{X}^s, \mathbf{Y}^s)$ and $Q(\mathbf{X}^t, \mathbf{Y}^t)$ respectively, $P \neq Q$. JAN [12] is a deep neural network $\mathbf{y} = f(\mathbf{x})$ which formally reduces the shifts in the joint distributions across domains and enables learning both transferable features and classifiers, such that, the target risk $R_t(f) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim Q}[f(\mathbf{x}) \neq \mathbf{y}]$ can be minimized by jointly minimizing the source risk and domain discrepancy.

Following the virtue of MMD, JAN use the Hilbert space embeddings of joint distributions to measure the discrepancy of two joint distributions to measure the discrepancy of two joint distributions $P(\mathbf{Z}^{s1}, \cdots, \mathbf{Z}^{s|\mathcal{L}|})$ and $Q(\mathbf{Z}^{t1}, \cdots, \mathbf{Z}^{t|\mathcal{L}|})$. The resulting measure is called Joint Maximum Mean Discrepancy (JMMD), which is defined as

$$D_{\mathcal{L}}(P,Q) \triangleq ||\mathcal{C}_{\mathbf{Z}^{s,1:|\mathcal{L}|}}(P) - \mathcal{C}_{\mathbf{Z}^{t,1:|\mathcal{L}|}}(Q)||^2_{\otimes_{\ell=1}^{|\mathcal{L}|}\mathcal{H}^\ell}. \quad (29)$$

Based on the virtue of the kernel two-sample test theory, we will have $P(\mathbf{Z}^{s1}, \cdots, \mathbf{Z}^{s|\mathcal{L}|}) = Q(\mathbf{Z}^{t1}, \cdots, \mathbf{Z}^{t|\mathcal{L}|})$ if and only if $D_{\mathcal{L}}(P,Q) = 0$. Given source domain $\mathcal{D}_s$ of n_s labeled points and target domain $\mathcal{D}_t$ of n_t unlabeled points drawn i.i.d. from P and Q respectively, the deep networks will generate activations in layers $\mathcal{L}$ as $\{(\mathbf{z}_i)^{s1}, \cdots, \mathbf{z}_i^{s|\mathcal{L}|}\}_{i=1}^{n_s}$ and $\{(\mathbf{z}_i)^{t1}, \cdots, \mathbf{t}_i^{s|\mathcal{L}|}\}_{i=1}^{n_t}$. The empirical estimate of $D_{\mathcal{L}}(P,Q)$ is computed as the squared distance between the empirical kernel mean embeddings as

$$\begin{aligned}\hat{D}_{\mathcal{L}}(P,Q) = &\frac{1}{n_s^2}\sum_{i=1}^{n_s}\sum_{j=1}^{n_s}\prod_{\ell\in\mathcal{L}} k^\ell(\mathbf{z}_i^{s\ell}, \mathbf{z}_j^{s\ell}) \\ &+ \frac{1}{n_t^2}\sum_{i=1}^{n_t}\sum_{j=1}^{n_t}\prod_{\ell\in\mathcal{L}} k^\ell(\mathbf{z}_i^{t\ell}, \mathbf{z}_j^{t\ell}) \\ &- \frac{1}{n_s n_t}\sum_{i=1}^{n_s}\sum_{j=1}^{n_t}\prod_{\ell\in\mathcal{L}} k^\ell(\mathbf{z}_i^{s\ell}, \mathbf{z}_j^{t\ell}).\end{aligned} \quad (30)$$

The pipeline of JAN is shown in fig. 5.

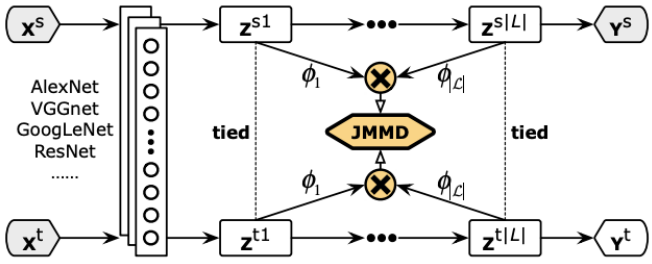


Figure 5. The architectures of Joint Adaptation Network (JAN). Since deep features eventually transition from general to specific along the network, activations in multiple domain-specific layers $\mathcal{L}$ are not safely transferable. And the joint distributions of the activations $P(\mathbf{X}^s, \mathbf{Y}^s)$ and $Q(\mathbf{X}^t, \mathbf{Y}^t)$ in these layers should be adapted by JMMD minimization.

3.4. Correlation Alignmen for Deep Domain Adaption (Deep CORAL)

The network structure of Deep CORAL [15] shown in fig. 6 shares the same structure with the DDC structure.

Similarly, it tries to learn a representation that minimizes the source and target distributions. However, it introduces the traditional CORAL to replace MMD used in DDC. Theoretically, CORAL-based methods could better represent the similarity between the source(denoted as S) and the target(denoted as T) than MMD-based methods, as CORAL not only exploit the "mean distance" information between S and T, but also exploit the "covariance" between them.

In Deep CORAL, the researchers incorporate CORAL with deep neural network, and then introduce the **CORAL loss**, to measure the difference between S and T on a single feature layer. Given the source-domain training examples $D_S = \mathbf{x}_i, \mathbf{x} \in \mathbb{R}^d$ with labels $L_S = \{y_i\}, i \in \{1, \ldots, L\}$, and unlabeled target dataset $D_T = \{\mathbf{u}_i\}, \mathbf{u} \in \mathbb{R}^d$. Suppose the number of source and target dataset are n_s and n_r respectively, and $D_S^{ij}(D_T ij)$ denotes the j-th dimension of the i-th source(target) data example, $C_S(C_T)$ denotes the feature covariance matrix. The CORAL loss is then depicted in eq. (31).

$$\ell_{CORAL} = \frac{1}{4d^2} \|C_S - C_T\|_F^2 \tag{31}$$

$\|\cdot\|_F^2$ denotes the squared matrix Frobenius norm. Here for $C_S(C_T)$, we use batch covariances, and the network parameters are shared on the source net and the target net.

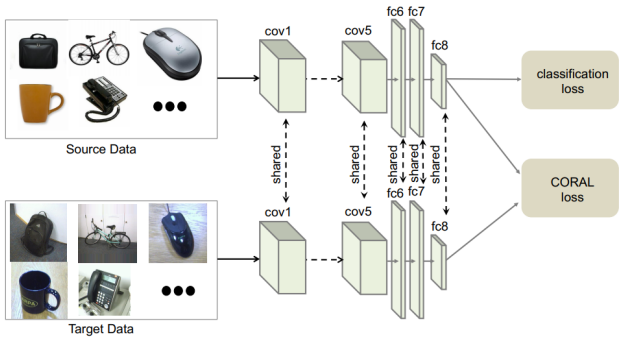


Figure 6. The network architecture for Deep CORAL.

3.5. Domain adversarial neural network (DANN)

[4] proposed an approach, DANN, directly inspired by the theory on domain adaptation suggesting that, for effective domain transfer to be achieved, predictions must be made based on features that cannot discriminate between the training (source) and test (target) domains.

The approach implements this idea in the context of neural network architectures that are trained on labeled data from the source domain and unlabeled data from the target domain (no labeled target-domain data is necessary). As the training progresses, the approach promotes the emergence of features that are (i) discriminative for the main learning task on the source domain and (ii) indiscriminate with respect to the shift between the domains. We show that this adaptation behaviour can be achieved in almost any feed-forward model by augmenting it with few standard layers and a new gradient reversal layer. The resulting augmented architecture can be trained using standard backpropagation and stochastic gradient descent, and can thus be implemented with little effort using any of the deep learning packages. The proposed architecture is shown as fig. 7.

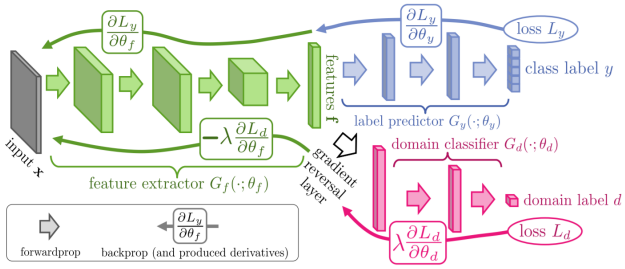


Figure 7. The proposed architecture includes a deep feature extractor (green) and a deep label predictor (blue), which together form a standard feed-forward architecture. Unsupervised domain adaptation is achieved by adding a domain classifier (red) connected to the feature extractor via a gradient reversal layer that multiplies the gradient by a certain negative constant during the back propagation-based training. Otherwise, the training proceeds standardly and minimizes the label prediction loss (for source examples) and the domain classification loss (for all samples). Gradient reversal ensures that the feature distributions over the two domains are made similar (as indistinguishable as possible for the domain classifier), thus resulting in the domain-invariant features.

3.6. Conditional Adversarial Domain Adaptation (CDAN)

[8] formulate Conditional Domain Adversarial Network (CDAN) as a minimax optimization problem with two competitive error terms: (a)$\mathcal{E}(G)$ on the source classifier G, which is minimized to guarantee lower source risk; (b)$\mathcal{E}(D, G)$ on the source classifier G and the domain discriminator D across the source and target domains, which is minimized over D but maximized over $\mathbf{f} = F(x)$ and $\mathbf{g} = G(x)$:

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

where $L(\cdot, \cdot)$ is the cross-entropy loss, and $\mathbf{h}(\mathbf{f}, \mathbf{g})$ is the joint variable of feature representation $\mathbf{f}$ and classifier prediction $\mathbf{g}$. The minimax game of conditional domain adversarial network (CDAN) is

$$\begin{aligned} &\min_G \mathcal{E}(G) - \lambda \mathcal{E}(D, G) \\ &\min_D \mathcal{E}(D, G). \end{aligned} \tag{33}$$

where λ is a hyper-parameter between the two objectives to tradeoff source risk and domain adversary.

We enable conditional adversarial domain adaptation over domain-specific feature representation $\mathbf{f}$ and classifier prediction $\mathbf{g}$. We jointly minimize (1) w.r.t. source classifier G and feature extractor F, minimize (2) w.r.t. domain discriminator D, and maximize (2) w.r.t. feature extractor F and source classifier G. This yields the minimax problem of Conditional Domain Adversarial Network (CDAN):

$$\begin{aligned}&\min_{G}\mathbb{E}_{(\mathbf{x}_i^s,\mathbf{y}_i^s)\sim\mathcal{D}_s} L\left(G\left(\mathbf{x}_i^s\right),\mathbf{y}_i^s\right)\\&\quad+\lambda\left(\mathbb{E}_{\mathbf{x}_i^s\sim\mathcal{D}_s}\log\left[D\left(T\left(\mathbf{h}_i^s\right)\right)\right]+\mathbb{E}_{\mathbf{x}_j^t\sim\mathcal{D}_t}\log\left[1-D\left(T\left(\mathbf{h}_j^t\right)\right)\right]\right)\\&\max_{D}\mathbb{E}_{\mathbf{x}_i^s\sim\mathcal{D}_s}\log\left[D\left(T\left(\mathbf{h}_i^s\right)\right)\right]+\mathbb{E}_{\mathbf{x}_j^t\sim\mathcal{D}_t}\log\left[1-D\left(T\left(\mathbf{h}_j^t\right)\right)\right]\end{aligned}\tag{34}$$

where λ is a hyper-parameter between source classifier and conditional domain discriminator, and note that $\mathbf{h}=(\mathbf{f},\mathbf{g})$ is the joint variable of domain-specific feature representation $\mathbf{f}$ and classifier prediction $\mathbf{g}$ for adversarial adaptation. As a rule of thumb, we can safely set $\mathbf{f}$ as the last feature layer representation and $\mathbf{g}$ as the classifier layer prediction.

The pipeline of CDAN is shown in fig. 8.

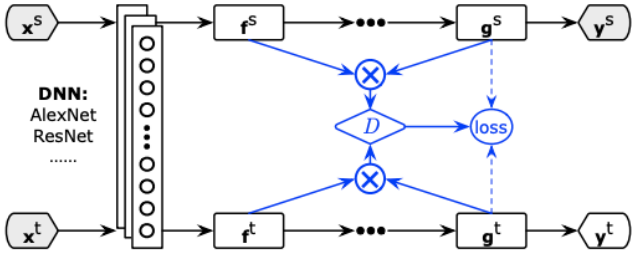


Figure 8. Architectures of Conditional Domain Adversarial Networks (CDAN) for domain adaptation, where domain-specific feature representation $\mathbf{f}$ and classifier prediction $\mathbf{g}$ embody the cross-domain gap to be reduced jointly by the conditional domain discriminator D.

3.7. Domain-Symmetric Networks for Adversarial Domain Adaptation (SymNet)

Recently, impressive progress is made recently by learning invariant features via domain-adversarial training of deep networks. In SymNet [21], a novel adversarial learning method is introduced. It is composed of two-level domain confusion scheme, where the category-level confusion learning and the domain-level confusion learning are updated by each other iteratively.

Given a deep neural network that is composed of convolutional and fully-connected(FC) layers, the domain confusion method uses the lower convolutional layers as the feature extractor G and upper FC layers as the task classifier C. The domain discriminator D, which is in parallel with C, is added on top of G to distinguish features of samples from the two domains. Therefore, source classification objective function is then defined by cross-entropy loss in eq. (35).

$$\min_{C,G}\mathcal{E}_{\text{task}}=\frac{1}{n_s}\sum_{i=1}^{n_s}\mathcal{L}^s\left(C\left(G\left(\mathbf{x}_i^s\right)\right),y_i^s\right)\tag{35}$$

Moreover, given the feature representations of different domains extracted by G, we could also learn domain discriminator D using the following objective in eq. (36)

$$\begin{aligned}\min_{D}\mathcal{E}_{\text{domain}}=&-\frac{1}{n_s}\sum_{i=1}^{n_s}\log\left(1-D\left(G\left(\mathbf{x}_i^s\right)\right)\right)\\&-\frac{1}{n_t}\sum_{j=1}^{n_t}\log\left(D\left(G\left(\mathbf{x}_j^t\right)\right)\right)\end{aligned}\tag{36}$$

And with the adversarial learning theory inspired by GAN, given a D, the domain confusion loss aims to learn G to maximally "confuse" the two domain. The objective function of G is then depicted in eq. (37)

$$\begin{aligned}\min_{G}\mathcal{F}_{\text{domain}}=&\frac{1}{2}\mathcal{E}_{\text{domain}}-\frac{1}{2n_s}\sum_{i=1}^{n_s}\log\left(D\left(G\left(\mathbf{x}_i^s\right)\right)\right)\\&-\frac{1}{2n_l}\sum_{j=1}^{n_t}\log\left(1-D\left(G\left(\mathbf{x}_j^t\right)\right)\right)\end{aligned}\tag{37}$$

Therefore, domain alignment is achieved by learning a domain invariant G based on the following adversarial objective of domain confusion shown in eq. (38)

$$\begin{aligned}&\min_{G,C}\mathcal{E}_{\text{task}}(G,C)+\lambda\mathcal{F}_{domain}(G,D)\\&\min_{D}\mathcal{E}_{domain}(G,D)\end{aligned}\tag{38}$$

where λ is a trade-off parameter.

The overall architecture of SymNet is then shown in fig. 9

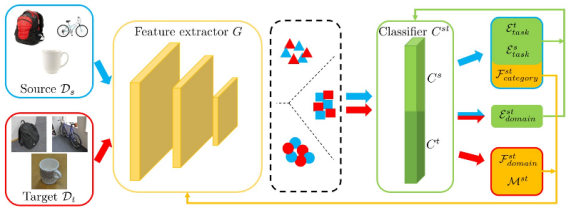


Figure 9. The network architecture for SymNet. It includes a feature extractor G and three classifiers of C^s, C^t and C^{st}. These three classifiers share the same layer neurons.

3.8. Incremental Collaborative and Adversarial Network for Unsupervised domain adaptation(iCAN)

iCAN [20] is based on the motivation that when learning domain-invariant features with the recent method DANN,

some characteristic information from target domain data may be lost. Meanwhile, the representations at lower blocks are often low-level features such as corners and edges, which are expected to be informative for distinguishing images from different domains. iCAN therefore **proposed one domain discriminator at each block** in order to learn domain-informative representations at lower blocks, and domain-uninformative representations at higher blocks.

iCAN introduced "feature extraction blocks" and "domain discriminator blocks" to learn domain-informative representations and domain-uninformative features separately. It then performs a collaborative and adversarial learning scheme to accommodate the two classification tasks. In particular, suppose in total m feature extraction blocks are used, where each block consists of a group of CNN layers, iCAN build a domain discriminator after each block. It also assign a weight $\lambda_k (k = 1, ..., m)$ for each domain discriminator, and automatically optimize the weights when loss is back-propagated.

Denote $\theta_k (k = 1, \ldots, m)$ as the network parameters before the k-th block, and $\mathbf{W}_k$ is the parameters of the domain discriminator at the k-th block. Also, denote the loss term from a domain discriminator as $\mathcal{L}_D(\boldsymbol{\theta}, \mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_D \left(D \left(F \left(\mathbf{x}_i; \boldsymbol{\theta} \right); \mathbf{w} \right), d_i \right)$. The collaborative and adversarial learning scheme is shown in eq. (39):

$$\begin{aligned} \min_{\Theta_F, \boldsymbol{\lambda}} \mathcal{L}_{CAN} = & \sum_{k=1}^{m-1} \lambda_k \min_{\mathbf{w}_k} \mathcal{L}_D \left(\boldsymbol{\theta}_k, \mathbf{w}_k \right) \\ & + \lambda_m \min_{\mathbf{w}_m} \mathcal{L}_D \left(\boldsymbol{\theta}_m, \mathbf{w}_m \right) \\ \text{s.t.} \sum_{k=1}^{m-1} \lambda_k = \lambda_0, & \quad |\lambda_k| \leq \lambda_0 \end{aligned} \tag{39}$$

The overall architecture of the iCAN model is shown in fig. 10.

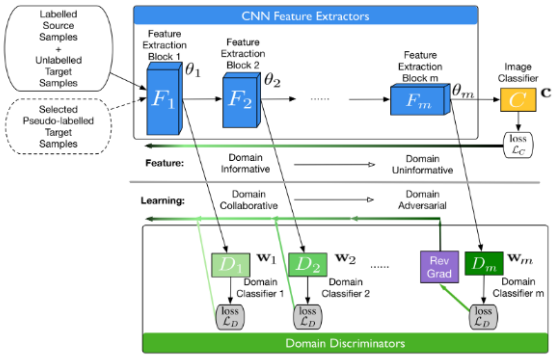


Figure 10. The overall architecture for iCAN model. Each feature extraction block consist of a group of CNN layers. Each Domain Discriminator is composed of several FC layers, which distinguish which domain each sample belongs to. Each block $F_k, k = 1, ..., m$, is followed with a domain classifier D_k in order to learn both domain informative and domain uninformative features.

3.9. GAN approach–PixelDA GAN

PixelDA GAN (Unsupervised pixel-level domain adaptation GAN) [1] has shown competitive results in domain adaptation. Previous works of DA use a single network that performs both domain adaptation and image classification, making the domain adaptation process closely related to the classifier architecture. PixelDA assumes that the differences between the domains are low-level(due to the noise, resolution, etc.), and it decouples the process of domain adaptation from the process of task-specific classification, which means other classification task could be trained without domain adaptation repetition.

PixelDA GAN employ a generative adversarial objective to encourage G to produce images that are similar to the target domain images. The generator $G \left(\mathbf{x}^s, \mathbf{z}; \boldsymbol{\theta}_G \right) \rightarrow \mathbf{x}^f$ maps a source image $\mathbf{x}^s$ to an adapted image $\mathbf{x}^f$. Then this model is augmented by a discriminator function $D \left(\mathbf{x}; \boldsymbol{\theta}_D \right)$ that tries to distinguish between fake images $\mathbf{X}^f$ produced by the generator and the real images from the target domain $\mathbf{X}^t$. In addition to the discriminator, the model is also augmented with a classifier $T \left(\mathrm{x}; \boldsymbol{\theta}_T \right) \rightarrow \hat{\mathbf{y}}$, which assigns task-specific labels $\hat{\mathbf{y}}$ to images $\mathbf{x} \in \left\{ \mathbf{X}^f, \mathbf{X}^t \right\}$.

The goal of PixelDA is to optimize the following objective function shown in eq. (40)

$$\min_{\boldsymbol{\theta}_G, \boldsymbol{\theta}_T} \max_{\boldsymbol{\theta}_D} \alpha \mathcal{L}_d(D, G) + \beta \mathcal{L}_t(G, T) \tag{40}$$

where α and β are tradeoff parameters, and $\mathcal{L}_d$ represents the domain loss

$$\begin{aligned} \mathcal{L}_d(D, G) = & \mathbb{E}_{\mathbf{x}^t} \left[\log D \left(\mathbf{x}^t; \boldsymbol{\theta}_D \right) \right] + \\ & \mathbb{E}_{\mathbf{x}^*, \mathbf{z}} \left[\log \left(1 - D \left(G \left(\mathbf{x}^s, \mathbf{z}; \boldsymbol{\theta}_G \right); \boldsymbol{\theta}_D \right) \right) \right] \end{aligned} \tag{41}$$

while $\mathcal{L}_t$ is the task-specific loss defined by a typical softmax cross-entropy loss.

$$\begin{aligned} \mathcal{L}_t(G, T) = \mathbb{E}_{\mathbf{x}^s, \mathbf{y}^s, \mathbf{z}} [& - \mathbf{y}^{\mathbf{s}\top} \log T \left(G \left(\mathbf{x}^s, \mathbf{z}; \boldsymbol{\theta}_G \right); \boldsymbol{\theta}_T \right) \\ & - \mathbf{y}^{\mathbf{s}\top} \log T \left(\mathbf{x}^s \right); \boldsymbol{\theta}_T] \end{aligned} \tag{42}$$

The overall architecture of PixelDA GAN is then shown in fig. 11

3.10. Extensions-Adversarial Meta-Adaption Networks (AMEAN)

In addition to the basic requirements listed in the assignment, we have also done auxiliary experiments on a more realistic scenery. In practice, our target domain is often composed of multiple sub-targets implicitly blended with each other, and the learners could not identify which subtarget each unlabeled sample belongs to, which is called

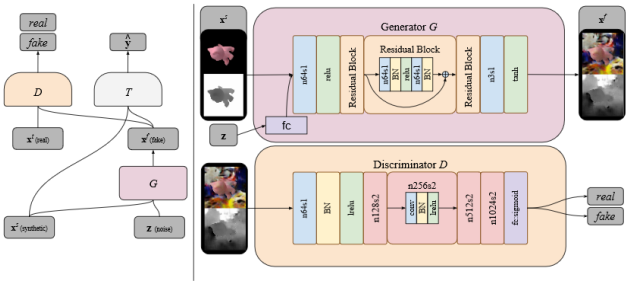


Figure 11. The overall architecture for pixelDA. On the left shows the overall model structure, while on the right shows the detail implementation of generator and discriminator components. The generator G generates an image based on the inpu of the synthetic image x^s and a noise vector z. The discriminator D discriminates between the real and the fake images. The task-specific classifier T assigns labels y to an input image.

the *Blending-target Domain Adaptation*(BDTA) scenario. In this section, we will briefly introduce the AMEAN [2] approach on BTDA. AMEAN is coupled of two adversarial learning processes, which are parallely executed to obtain domain-invariant features by transferring data from source $\mathcal{S}$ to mixed target $\mathcal{T}$, which is composed of k "meta-sub-targets" $\left\{\hat{\mathcal{T}}_j\right\}_{j=1}^{k}$.

Suppose C is the softmax classifier on the transferable feature space, and F denotes the feature extractor we are to optimize in the BTDA scenarios. Inspired by GAN, we also employ a source-target domain discriminator D_{st} to deploy an adversarial learning scheme, where F is trained to match a source set $\mathcal{S}$ and a mixed target set $\mathcal{T}$ at the feature level, while D_{st} is demanded to separate the source and the target features.

The pure source-to-target transfer is blind to the domain gaps and category misalignment among $\{\mathcal{T}_j\}_{j=1}^{k}$, which makes the transfer suffer. AMEAN introduced an unsupervised meta-learner trained to obtain k embedded clusters as the "meta-sub-targets", which automatically and dynamically outputs multi-target adversarial adaptation loss to minmize the category mismatchin inside the mixed target $\mathcal{T}$.

The overall pipeline of AMEAN is then shown in fig. 12.

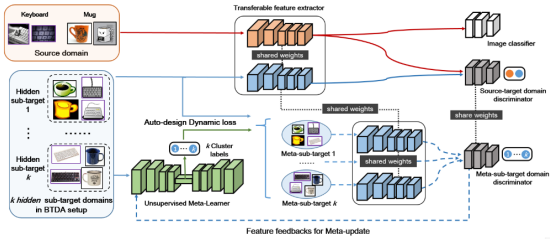


Figure 12. The learning pipeline of Adversarial Meta-Adaption Network (AMEAN). It receives source samples with ground truth and unlabeled samples drawn from an unknown target mixture to synchronously execute two transfer learning processes.

4. Experiments and Results

4.1. Dataset

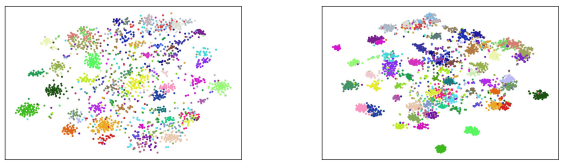
Office-Home This dataset consists of 65 categories of office depot from 4 domains (i.e., A: Art, C: Clipart, P: Product, R: Real-World). The size of different domain data is shown in table 1.

For traditional methods, we use the 2048-dim ResNet50 deep learning features of all images. For deep learning methods, we process the raw images, which can be downloaded from http://hemanthdv.org/OfficeHome-Dataset/.

Table 1. Size of different domain datasets

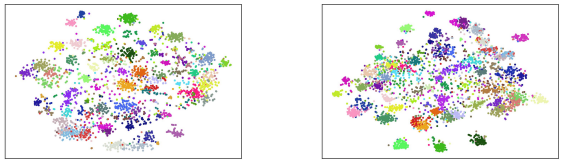
Domain	Description	Size
A	Art	2426
C	Clipart	4346
P	Product	4438
R	Real World	4356

The experiment consists of three domain adaptations:A→R, C→R, and P→R. The visualization of the distribution of these three datasets are shown in fig. 13, fig. 14, fig. 15.



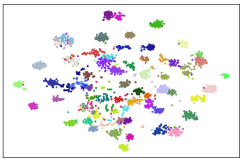
(a) A→R source distribution (b) A→R target distribution

Figure 13. The visualization of dataset distribution in task A→R using **t-SNE**

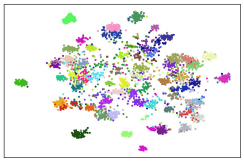


(a) C→R source distribution (b) C→R target distribution

Figure 14. The visualization of dataset distribution in task C→R using **t-SNE**



(a) P→R source distribution



(b) P→R starget distribution

Figure 15. The visualization of dataset distribution in task P→R using **t-SNE**

4.2. Experiment Procedure

1. Using the 2048-dim ResNet50 deep learning features or the raw images, for each of these three domain adaptation tasks (A→R, C→R, P→R), we get the labeled source dataset and the unlabeled target dataset.

2. In X→R settings (X can be A, C or P), we use unsupervised traditional methods or deep learning methods to achieve the domain adaptation task.

3. For traditional methods, we use SVM classifier to get the final classification results on the target dataset. For deep learning methods, we build simple neural networks (usually some fully connected layers) to test our model.

4.3. Settings and Results of traditional methods

4.3.1 SVM (baseline)

For better showing the effect of domain adaptation methods, we choose SVM as our accuracy baseline. In SVM, we just regard source domain as our training set, and target domain as our testing set. In other words, we do not have an adaptation progress, think both domains in the same distribution. The result of SVM is shown as table 2 and fig. 16.

Table 2. The accuracy of baseline method SVM, without any domain adaptation method. the corresponding parameters are given, too.

Task	Optimal Accuracy	Kernel	C
A→R	0.743342	Linear	1.0
C→R	0.647153	Gaussian	1.0
P→R	0.742195	Gaussian	1.0

4.3.2 TCA

In this sub-experiment, we choose SVM with linear kernel, C=1 as our classifier to get the accuracy. The dimension of projected data is 256 and the results are shown as table 3

The visualization results of t-SNE are shown in fig. 17.

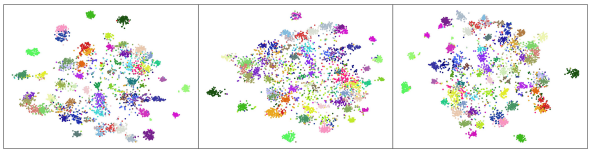


Figure 16. The target domain classification result of SVM, visualized using **t-SNE**. Each color represents each category of samples. From left to right, the subfigure is from task A→R, C→R, and P→R.

Table 3. The accuracy of TCA in three tasks

Task	Optimal Accuracy
A→R	0.733012
C→R	0.616621
P→R	0.676538

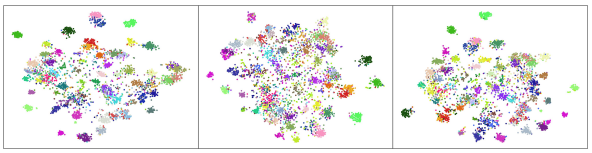


Figure 17. The target domain classification result of TCA, visualized using **t-SNE**. Each color represents each category of samples. From left to right, the sub-figure is from task A→R, C→R, and P→R.

TCA can be one of the earliest (oldest) proposed methods. However, TCA still performs better than the following method like BDA and CORAL in this dataset. TCA gets the accuracy under baseline in all three tasks.

4.3.3 GFK

In this sub-experiment, we choose SVM with linear kernel, C=1 as our classifier to get the accuracy. The dimension of projected data is 512. The results are shown in table 4.

Table 4. The accuracy of GFK in three tasks

Task	Optimal Accuracy
A→R	0.660468
C→R	0.589991
P→R	0.696740

The visualization results of t-SNE are shown in fig. 18.

For dataset Office-Home, GFK's performance is poor, which means this kernel is not suitable for distriution of Office-Home.

4.3.4 CORAL

CORAL needs no hyper-parameters, which is the biggest advantage of CORAL. As [14] shows, The regularization parameters λ has little influence on the final result. Our

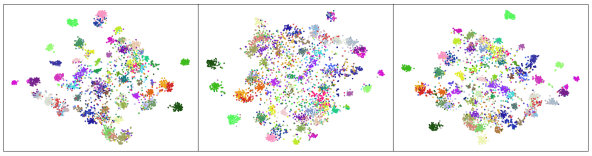


Figure 18. The target domain classification result of GFK, visualized using **t-SNE**. Each color represents each category of samples. From left to right, the sub-figure is from task A→R, C→R, and P→R.

experimental result for three tasks and the visualization of the result is shown as table 5 and fig. 19.

Table 5. The accuracy of CORAL in three tasks

Task	Optimal Accuracy	Parameters
A→R	0.648531	-
C→R	0.581497	-
P→R	0.684803	-

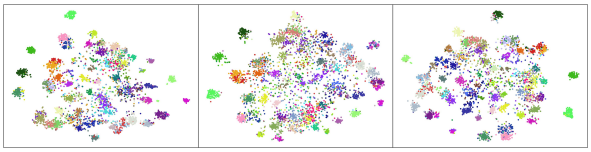


Figure 19. The target domain classification result of CORAL, visualized using **t-SNE**. Each color represents each category of samples. From left to right, the sub-figure is from task A→R, C→R, and P→R.

As the result shows, CORAL performs pretty badly. Actually CORAL is the second worst among our all traditional methods. The accuracy of three tasks is all well under the baseline SVM especially task A→R. We guess that CORAL is inherently inappropriate for no-deep models. However, as [14] mentioned, deep CORAL performs much better than CORAL, which is also included in our experiments.

4.3.5 BDA

For this sub-experiment, we use the 1NN classifier after the projected data is derived. The dimension of projected space is 256 and the kernel is linear. The experimental results of BDA are shown in table 6.

Table 6. The accuracy of BDA in three tasks

Task	Optimal Accuracy
A→R	0.648072
C→R	0.579201
P→R	0.668274

The visualization results using t-SNE are shown in fig. 20.

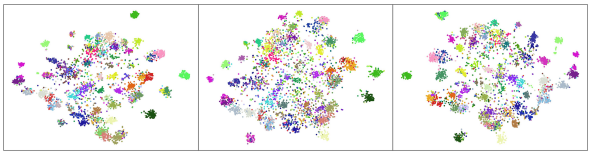


Figure 20. The target domain classification result of BDA, visualized using **t-SNE**. Each color represents each category of samples. From left to right, the sub-figure is from task A→R, C→R, and P→R.

As we can see from the results, BDA performs even worst. The baseline method SVM outperforms BDA on all the three tasks. While BDA does have satisfying performance on other domain adaptation dataset such as Mnist→USPS. We guess Office-Home dataset just might be too hard for BDA to handle. On the other hand, the extensively extension of BDA, MEDA, actually performs quite well on this dataset.

4.3.6 SA

For this sub-experiment, we use the logistic regression after the projected data is derived. The dimension of projected space is 512. The experimental results of SA are shown in table 7.

Table 7. The accuracy of SA in three tasks

Task	Optimal Accuracy
A→R	0.745638
C→R	0.632461
P→R	0.712351

The visualization results using t-SNE are shown in fig. 21.

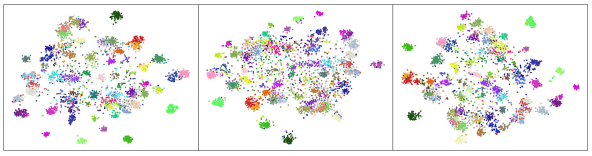


Figure 21. The target domain classification result of SA, visualized using **t-SNE**. Each color represents each category of samples. From left to right, the sub-figure is from task A→R, C→R, and P→R.

As we can see from the results, SA perform a lot better than the previous methods on all three tasks. Compared with other methods, SA could better explore the invariant feature space between the source and the target dataset on "Office-home".

4.3.7 IW

In this sub-experiment, we choose SVM with linear kernel, C=1 as our classifier to get the accuracy. We use KMM as the IW classifier. The results are shown in table 8.

Table 8. The accuracy of IW in three tasks

Task	Optimal Accuracy
A→R	0.730026
C→R	0.602388
P→R	0.687328

The visualization results of t-SNE are shown in fig. 22.

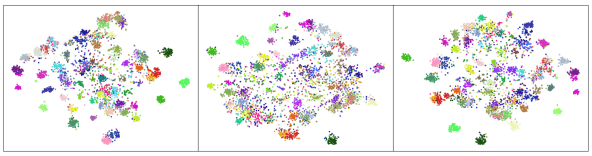


Figure 22. The target domain classification result of IW (KMM), visualized using **t-SNE**. Each color represents each category of samples. From left to right, the sub-figure is from task A→R, C→R, and P→R.

IW (KMM) does not outperform TCA. The limitation of IW is obvious, sometimes the divergence between sample from source domain which is the most simliar to target domain is still large, and Office-Home is an example. In this case, it is not very effective to assign weight to each sample.

4.3.8 JDA

For this sub-experiment, we use the linear regression after the projected data is derived. The dimension of projected space is 256. After projection, we use 1NN to construct a classifier and get the experimental result shown in table 9.

Table 9. The accuracy of JDA in three tasks

Task	Optimal Accuracy
A→R	0.675620
C→R	0.570478
P→R	0.690771

The visualization results using t-SNE are shown in fig. 23.

From the result we can see that JDA still cannot have a positive effect in three tasks compared with baseline SVM though JDA beats BDA in task A→R and P→R. We can say that JDA performs badly in this dataset "Office-Home".

4.3.9 TJM

We still choose the KNN classifier for this sub-experiment. The dimension of projected data is 256, and the kernel we use is linear kernel. The results are shown in table 10.

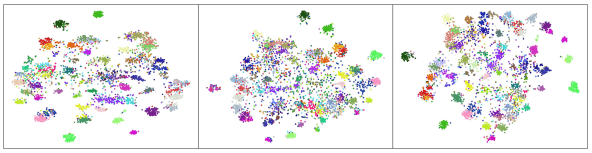


Figure 23. he target domain classification result of JDA, visualized using t-SNE. Each color represents each category of samples. From left to right, the sub-figure is from task A→R, C→R, and P→R

Table 10. The accuracy of TJM in three tasks

Task	Optimal Accuracy
A→R	0.684803
C→R	0.615403
P→R	0.711433

The visualization results of t-SNE are shown in fig. 24.

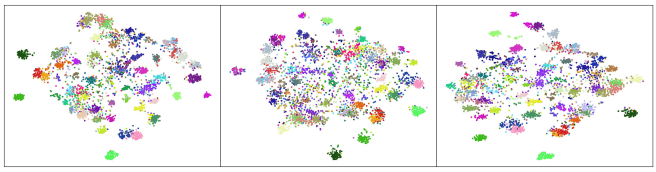


Figure 24. The target domain classification result of TJM, visualized using **t-SNE**. Each color represents each category of samples. From left to right, the sub-figure is from task A→R, C→R, and P→R.

TJM shows a comparable performance compared with Coral and BDA. However, it still underperforms the baseline method SVM. We believe TJM is not much capable to extract a domain-invariant representation on this dataset. From another view, this dataset is quite challenging for a domain adaptation task.

4.3.10 MEDA

The dimension of projected space is 128 and the kernel function is linear kernel, and we set the regularization parameters $\rho = 0.1$,$\eta = 0.1$. After projection, we use 1NN (KNN with one nearest neighbor voting) to classify and get the result. The experimental results of MEDA are shown as table 11 and the visualization results of t-SNE are shown as fig. 25.

Table 11. The accuracy of MEDA in three tasks

Task	Optimal Accuracy
A→R	0.780533
C→R	0.721074
P→R	0.786042

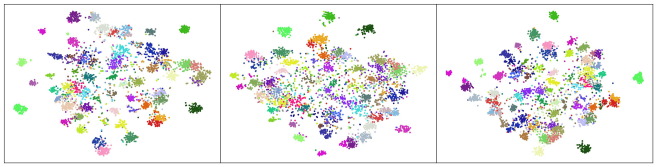


Figure 25. The target domain classification result of MEDA, visualized using **t-SNE**. Each color represents each category of samples. From left to right, the subfigure is from task A→R, C→R, and P→R.

From the result, we can see that MEDA gets the highest accuracy among all the traditional methods. In addition, MEDA even beats all deep domain adaptation methods. Exactly the experiments in [19] shows that MEDA has an overall superiority in this data set, Office-Home, including other tasks between Art, Clipart, and Product.

As for the reason why MEDA performs so well, we guess that MEDA is based on the method BDA, and it is the **first** method to successfully evaluate the parameter μ in eq. (21). In other words, MEDA allows us to know exactly the distribution of which part is more important!

Meanwhile, because MEDA performs very well, we will have a discussion about the impact of the parameters η and ρ in MEDA which is shown in section 5.1 and section 5.2. In other words, we will try to make the accuracy get the highest by adjusting the parameters.

4.4. Settings and Results of deep DA methods

4.4.1 Fine-tuned ResNet (baseline)

ResNet [6] has shown superior performance on many computer vision tasks. For this domain adaptation project, we use the pre-trained ResNet50 model and fine-tune it on Office-Home dataset. Among all the deep learning methods, we consider it as the baseline method.

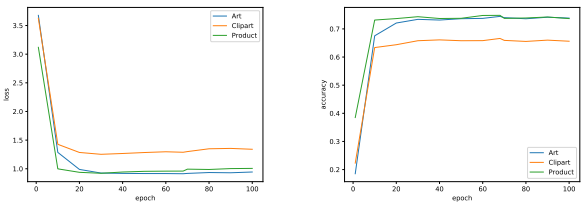
The basic settings for this sub-experiment: *batch size = 32, number of training epochs = 100, one pass through the whole training set per epoch.*

The results are shown in table 12 and fig. 26.

Table 12. The results of fine-tuned ResNet. Apart from the optimal accuracy, the target loss and the corresponding epoch the model achieved the highest accuracy are also listed.

Task	Optimal Accuracy	Loss	Epoch
A→R	0.744549	0.9133	68
C→R	0.666055	1.2880	68
P→R	0.748221	0.9594	66

We could see that the model reaches the optimal accuracy after nearly 70 epochs on these three tasks. Compared to the traditional baseline methods (SVM), the fine-tuned deep models have a higher classification accuracy on this domain adaptation dataset.



(a) Loss (b) Accuracy

Figure 26. Loss curves and accuracy curves of fine-tuned ResNet on three tasks

4.4.2 DDC

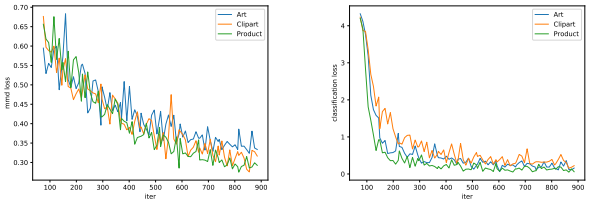
The basic settings for this sub-experiment: *pre-trained ResNet50, batch size = 32, number of training epochs = 10, number of iterations = 74, 135, 138 per epoch for task A→R, C→R and P→R* respectively.

The results are shown in the following table 13.

Table 13. The results of the DDC model. Apart from the optimal accuracy, the target loss and the corresponding epoch the model achieved the highest accuracy are also listed.

Task	Optimal Accuracy	Loss	Epoch
A→R	0.7427	0.9055	7
C→R	0.6583	1.2812	6
P→R	0.7283	1.0052	4

To better show the results, we will exhibit the curves of confusion loss (MMD loss), classification loss of models on target dataset for the three tasks, which are shown in fig. 27.



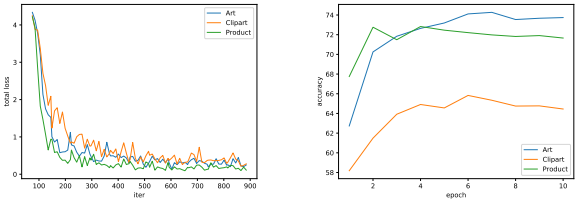
(a) MMD loss (b) Classification loss

Figure 27. Loss curves of the DDC model on three tasks

The total loss curves and the accuracy curves are shown in the following fig. 28.

As we can conclude from the results, both confusion loss (MMD loss) and classification loss on target dataset are declining during training, which means classification and domain invariance are both optimzed to achieve good classification accuracy on anthoer domain dataset (domain R) after trained on one domain dataset (domain A, C or P).

Compared to baseline method fine-tuned ResNet, the DDC model does not show much superior performance.



(a) Total loss (b) Accuracy

Figure 28. Loss curves and accuracy curves of the DDC model on three tasks

4.4.3 DAN

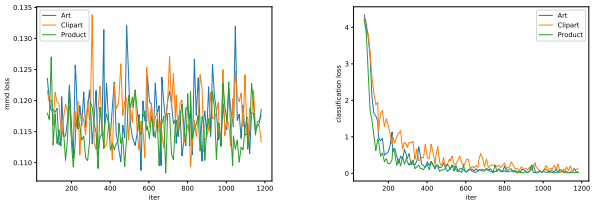
The basic settings for this sub-experiment: *pre-trained ResNet50, batch size = 32, number of training epochs = 15, number of iterations = 74, 135, 138 per epoch for task A→R, C→R and P→R* respectively.

The results are shown in the following table 14.

Table 14. The results of the DAN model. Apart from the optimal accuracy, the target loss and the corresponding epoch the model achieved the highest accuracy are also listed.

Task	Optimal Accuracy	Loss	Epoch
A→R	0.7574	0.9221	10
C→R	0.6764	1.2748	7
P→R	0.7439	0.9848	10

The curves of MMD loss and classification loss for the three tasks are shown in fig. 29.

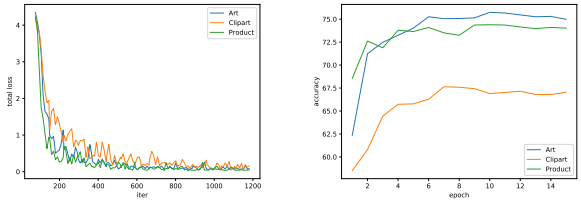


(a) MMD loss (b) Classification loss

Figure 29. Loss curves of the DAN model on three tasks

The total loss curves and the accuracy curves are shown in fig. 30.

According to the results, DAN outperforms the baseline method fine-tuned ResNet on all the three tasks. The performance boost demonstrates that the architecture of the DAN model, which includes multi-layer adaptation via multi-kernel MMD is able to transfer pre-trained deep models across different domains. One interesting finding is that MMD loss tends to fluctuate rather than continuously go down during the training process.



(a) Total loss (b) Accuracy

Figure 30. Loss curves and accuracy curves of the DAN model on three tasks

4.4.4 JAN

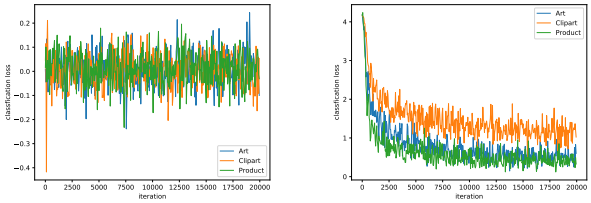
The basic settings for this sub-experiment: *pretrained ResNet50, batch size = 36, number of iterations = 20000.*

The results are shown in table 15.

Table 15. The results of the JAN model. Apart from the optimal accuracy, the target loss and the corresponding iteration the model achieved the highest accuracy are also listed.

Task	Optimal Accuracy	Loss	Iteration
A→R	0.7363	1.5189	16000
C→R	0.6539	1.6702	18500
P→R	0.7395	1.4747	8500

The curves of transfer loss and classification loss for the three tasks are shown in fig. 31.



(a) Transfer loss (b) Classification loss

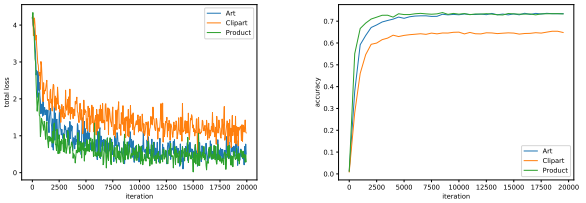
Figure 31. Loss curves of the JAN model on three tasks

The total loss curves and the accuracy curves are shown in fig. 32.

JAN does not outperform the baseline fine-tuned ResNet. However, JAN outperforms ResNet and DAN according to [12]. The first reason may be the settings of hyperparameter need to be adjust. The second reason may be domain adaptation methods sometimes are not robust on different datasets, and JAN cannot work well on dataset Office-Home.

4.4.5 Deep CORAL

The basic settings for this sub-experiment: *pre-trained ResNet50, batch size = 32, number of training epochs =*



(a) Total loss (b) Accuracy

Figure 32. Loss curves and accuracy curves of the JAN model on three tasks

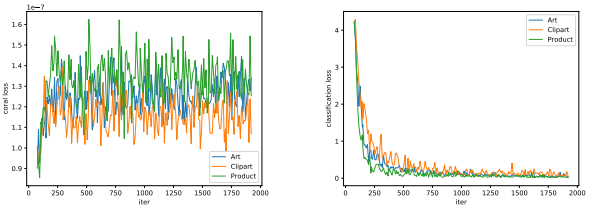
25, number of iterations = 74, 135, 138 per epoch for task A→R, C→R and P→R respectively.

The results are shown in the following table 16.

Table 16. The results of the Deep CORAL model. Apart from the optimal accuracy, the target loss and the corresponding epoch the model achieved the highest accuracy are also listed.

Task	Optimal Accuracy	Loss	Epoch
A→R	0.7652	0.8426	10
C→R	0.6899	1.2023	21
P→R	0.7501	0.9325	9

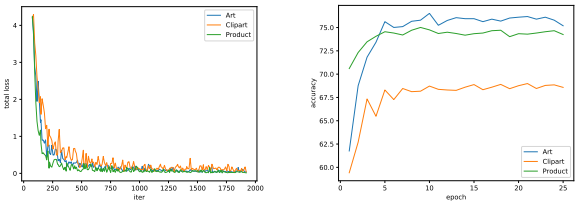
To better show the results, we will exhibit the curves of confusion loss (Coral loss), classification loss of models on target dataset for the three tasks, which are shown in fig. 33.



(a) CORAL loss (b) Classification loss

Figure 33. Loss curves of the Deep CORAL model on three tasks

The total loss curves and the accuracy curves are shown in the following fig. 34.



(a) Total loss (b) Accuracy

Figure 34. Loss curves and accuracy curves of the Deep CORAL model on three tasks

According to the results, Deep CORAL even outperforms DAN on all three tasks. As we have already presented in the previous sections that CORAL loss combines both the mean and the correlation matrix to perform domain adaptation, while MMD loss used in DAN only utilizes the mean information, Deep CORAL is often a better choice in practice.

4.4.6 DANN

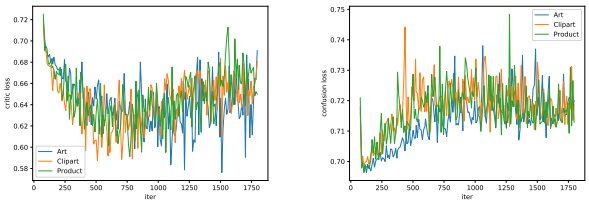
The basic settings for this sub-experiment: *pre-trained ResNet50, batch size = 32, number of training epochs = 18, 18, 28, number of iterations = 74, 135, 138 per epoch for task A→R, C→R and P→R* respectively.

The results are shown in the following table 17.

Table 17. The results of the DANN model. Apart from the optimal accuracy, the target loss and the corresponding epoch the model achieved the highest accuracy are also listed.

Task	Optimal Accuracy	Loss	Epoch
A→R	0.7597	0.8721	6
C→R	0.6771	1.2932	15
P→R	0.7542	0.9826	26

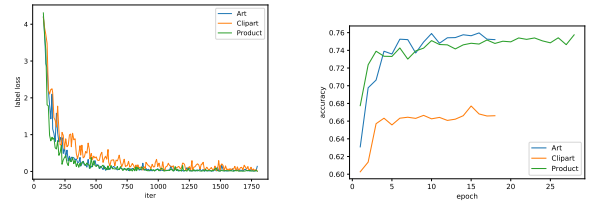
The curves of confusion loss and critic loss are shown below in fig. 35. The label loss curves and the accuracy



(a) Critic loss (b) Confusion loss

Figure 35. Critic loss and confusion loss curves of the DANN model on three tasks

curves are shown in the following fig. 36. According to the



(a) Label loss (b) Accuracy

Figure 36. Label loss and accuracy curves of the DANN model on three tasks.

results, DANN beats the baseline fine-tuned ResNet in all

three tasks. And we noticed that DANN gets the optimal result in a fewer epoch than Deep CORAL and DANN performs well in task C→R.

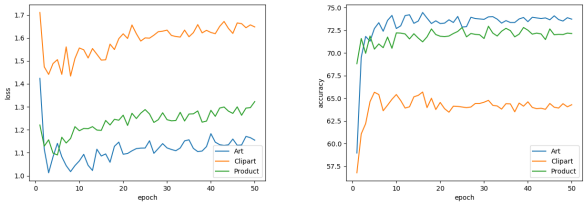
4.4.7 SymNets

The basic settings for this sub-experiment: *pre-trained ResNet50, batch size = 32, number of training epochs = 50 for task A→R, C→R and P→R* respectively.

The results are shown in the following table 18. The curves of total loss and accuracy are shown below in fig. 37.

Table 18. The results of the SymNet model. Apart from the optimal accuracy, the target loss and the corresponding epoch the model achieved the highest accuracy are also listed.

Task	Optimal Accuracy	Loss	Epoch
A→R	0.7445	1.0633	16
C→R	0.6569	1.5047	16
P→R	0.7294	1.2595	31



(a) Total loss (b) Accuracy

Figure 37. Total loss and accuracy curves of the SymNet model on three tasks.

From the results, we could see that SymNet has achieved competitive results on all three tasks.

4.4.8 CDAN

The basic settings for this sub-experiment: *pretrained ResNet50, batch size = 36, number of iterations = 20000.*

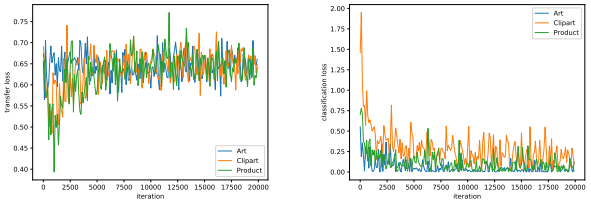
The results are shown in table 19.

Table 19. The results of the CDAN model. Apart from the optimal accuracy, the target loss and the corresponding iteration the model achieved the highest accuracy are also listed.

Task	Optimal Accuracy	Loss	Iteration
A→R	0.7659	0.6798	5000
C→R	0.7133	0.9379	16100
P→R	0.7787	0.6961	19700

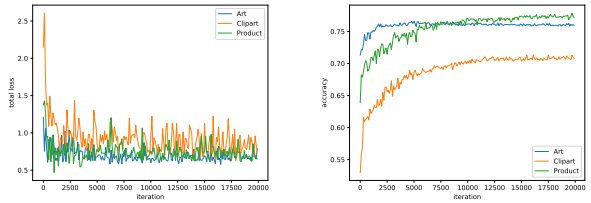
The curves of transfer loss and classification loss for the three tasks are shown in fig. 38.

The total loss curves and the accuracy curves are shown in fig. 39.



(a) Transfer loss (b) Classification loss

Figure 38. Loss curves of the CDAN model on three tasks



(a) Total loss (b) Accuracy

Figure 39. Loss curves and accuracy curves of the CDAN model on three tasks

CDAN reaches the best performance over all of the deep learning based domain adaptation methods, which show the effectiveness of adversarial learning in domain adaptation. By adversarial training, deep neural network can be much more robust on different domains.

4.4.9 iCAN

The basic settings for this sub-experiment: *batch size = 16, number of training epochs = 66.*

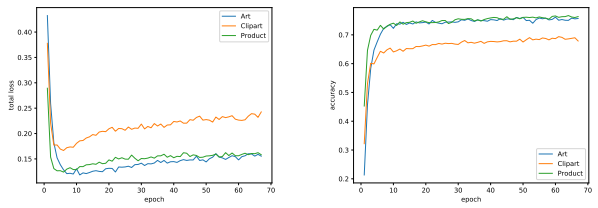
The results are shown in the following table 20.

Table 20. The results of the iCAN model. Apart from the optimal accuracy, the target loss and the corresponding epoch the model achieved the highest accuracy are also listed.

Task	Optimal Accuracy	Loss	Epoch
A→R	0.7615	0.1480	59
C→R	0.6938	0.2259	60
P→R	0.7670	0.1608	63

To better present the results, the total loss curves and the accuracy curves are shown in the following fig. 40.

From the statistics above, we could see that iCAN performs better on all three classification tasks than most of the other networks. It is because iCAN explores both the domain-informative and domain-uninformative features to perform both domain classification and the object classification tasks, while the other approaches only focuses on the domain-uninformative features to project the source and the target domain on a common space.



(a) Total loss (b) Accuracy

Figure 40. Loss curves and accuracy curves of the iCAN model on three tasks.

As iCAN shows the relatively competitive results on all three tasks, to better present the classification ability of iCAN, we have drawn the heatmap of the classification result on iCAN, which is shown in fig. 41.

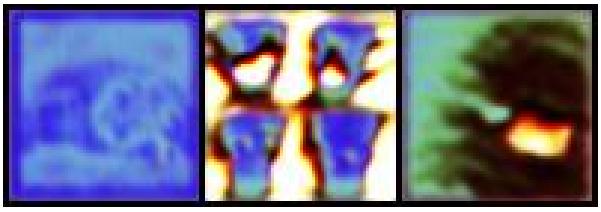
Also, to better visualize the learning ability for iCAN, we also somehow extract the 256-dimension features from the output of the last layer of iCAN(before the fc layers). We then perform tsne on these hidden features and project them to the 2-dimensional space, which is shown in fig. 42

4.4.10 PixelDA GAN

The basic settings for this sub-experiment: *batch size = 8, image size = 64, latent dimension = 10, number of training epochs = 100 for all tasks.*

We have visualized the target images and the images processed by the generator G in GAN in fig. 43.

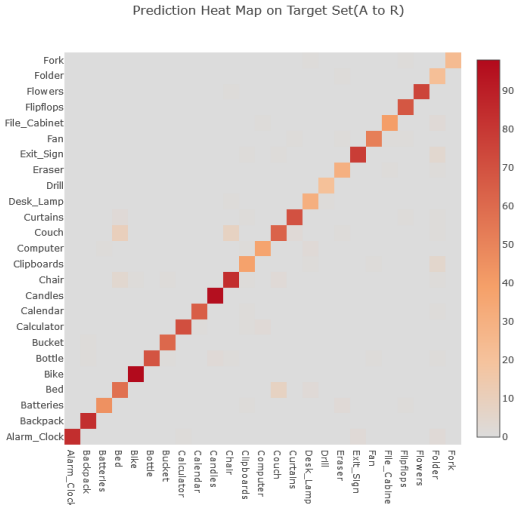
(a) raw target images



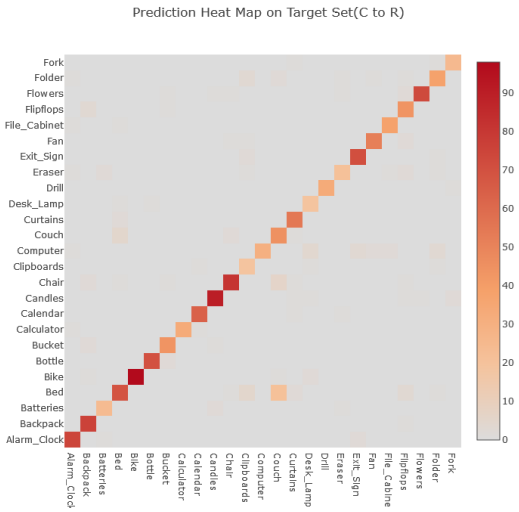
(b) processed by GAN

Figure 43. The raw target images and images processed by PixelDA GAN . We could see that the upper images and bottom images have showed similarities in pair, but in a total different style.

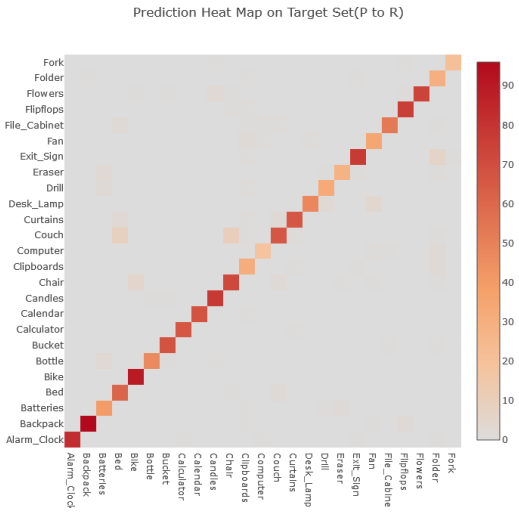
However, although the PixelDA GAN has shown competitive results on the "MNIST" and "SVHR" dataset, its classification performance is relatively poor on the "Office-



(a) $A \rightarrow R$ heatmap

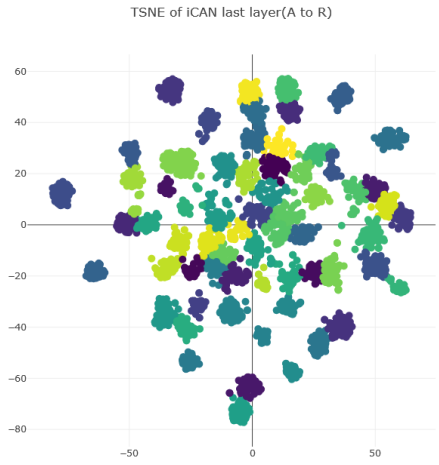


(b) $C \rightarrow R$ heatmap

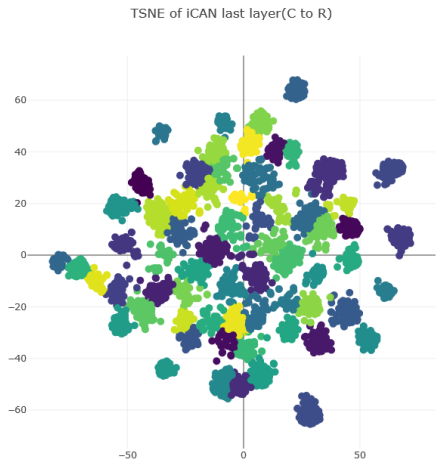


(c) $P \rightarrow R$ heatmap

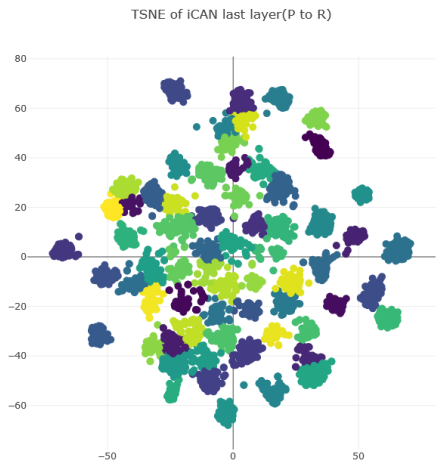
Figure 41. The heat map of the classification results of iCAN on all three tasks. iCAN performs well in all the classification tasks with a high confidence.



(a) $A \rightarrow R$ tsned features



(b) $C \rightarrow R$ tsned features



(c) $P \rightarrow R$ tsned features

Figure 42. The tsned results on the last feature layer extracted by iCAN feature extraction block. Most of the samples are well classified into sub clusters.

home" dataset. We think it's because the "office" dataset is much more harder than the "MNIST" dataset, and it's hard for neural networks to reconstruct a 3-channel 64×64 image within a latent dimension of size 10, thus the classification performance suffers on the target dataset.

4.4.11 AMEAN (Extensions)

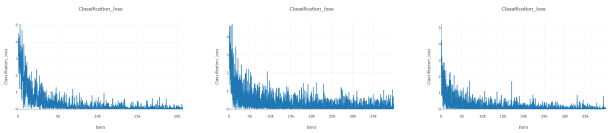
The basic settings for this sub-experiment: *pre-trained ALEXnet, batch size = 4, number of iteration = 40000, update iteration period = 2000, learning rate = 0.001.*

AMEAN is based on the BDTA scenario. BDTA (*Blending-target Domain Adaptation*) is under the circumstances where our target domain is often composed of multiple sub-targets implicitly blended with each other, and the learners could not identify which sub-target each unlabeled sample belongs to. The relative results are shown in 21. Notice that the training process is totally under the BDTA scenario, thus it's accuracy is no doubt slightly lower than the previous methods, but still competitive.

Table 21. The results of the AMEAN model on the BTDA scenario.

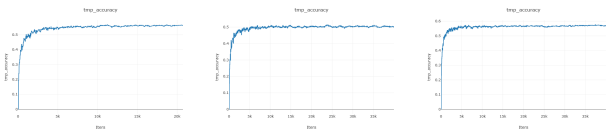
Task	Optimal Accuracy
A→R	0.5715
C→R	0.5144
P→R	0.5745

To better show the results, we will exhibit the curves of classification loss and testing accuracy of all models on target dataset for the three tasks, which are shown in fig. 44 and fig. 45.



(a) $A \rightarrow R$ loss (b) $C \rightarrow R$ loss (c) $P \rightarrow R$ loss

Figure 44. The classification loss of AMEAN on all three tasks. All the curves have shown great convergence property.



(a) $A \rightarrow R$ accuracy (b) $C \rightarrow R$ accuracy (c) $P \rightarrow R$ accuracy

Figure 45. The classification accuracy of AMEAN on all three tasks. All the curves have shown great convergence property.

According to the results, AMEAN has shown competitive performance on the BTDA scenario. The BTDA sce-

nario is more like the natural scenes in life, where domains are mixed and correlations of domains are complicated. AMEAN incorporates meta-learning in dealing with such circumstances, and could produce the state-of-art results in practice.

5. Discussion

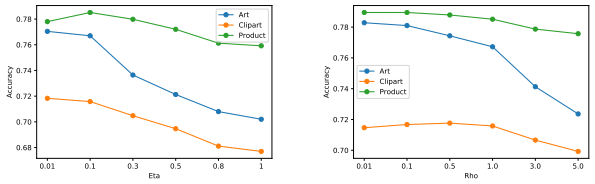
5.1. How the parameters η and ρ influence MEDA, the best one among all listed methods.

Recall that if we denote $g(\cdot)$ the manifold feature learning function, then f can be represented as:

$$f = \underset{f \in \sum_{i=1}^{n} \mathcal{H}_K}{\arg\min} \ell\left(f\left(g\left(\mathbf{x}_i\right)\right), y_i\right) + \eta\|f\|_K^2 + \lambda \overline{D_f}\left(\mathcal{D}_s, \mathcal{D}_t\right) + \rho R_f\left(\mathcal{D}_s, \mathcal{D}_t\right) \tag{43}$$

where $\|f\|_K^2$ is the squared norm of f. The term $\overline{D_f}(\cdot,\cdot)$ 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 $\mathbb{G}$. η, λ, and ρ are regularization parameters accordingly.

In [19], it mentioned that $\eta \in [0.01, 1]$ and $\rho \in [0.01, 5]$ in eq. (43). Therefore, we choose some specific value and test the accuracy. The result can be shown in below as table 22, table 23, and fig. 46.



(a) The impact of η (b) The impact of ρ

Figure 46. The impact of regularization parameters in MEDA, η and rho on the final accuracy result. The dimension is set to 128. From the sub-figures we can see that the accuracy decreases when both parameters increases.

5.2. The impact exploration of projection dimension on the accuracy result in traditional methods

In traditional domain adaptation, there is a very important parameter, the dimension of projected space. In this section we will explore the impact of the dimension on three traditional methods which have this parameter. These three traditional methods are the top 3 in this task, which is TCA, SA and champion MEDA.

The accuracy result is shown in table 24 and fig. 47.

5.3. The result rank and visualization of all listed methods

The rank visualization of all listed methods is shown as fig. 48. **Note that there are comprehensive experimental results and figures in the final page.**

References

[1] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3722–3731, 2017.

[2] Z. Chen, J. Zhuang, X. Liang, and L. Lin. Blending-target domain adaptation by adversarial meta-adaptation networks.

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

[4] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. *J. Mach. Learn. Res.*, 17(1):2096–2030, Jan. 2016.

[5] B. Gong, Y. Shi, F. Sha, and K. Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *CVPR*, pages 2066–2073. IEEE Computer Society, 2012.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[7] J. Huang, A. Gretton, K. Borgwardt, B. Schölkopf, and A. J. Smola. Correcting sample selection bias by unlabeled data. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 601–608. MIT Press, 2007.

[8] M. Long, Z. CAO, J. Wang, and M. I. Jordan. Conditional adversarial domain adaptation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 1640–1650. Curran Associates, Inc., 2018.

[9] M. Long and J. Wang. Learning transferable features with deep adaptation networks. *CoRR*, abs/1502.02791, 2015.

[10] M. Long, J. Wang, G. Ding, J. Sun, and P. S. Yu. Transfer feature learning with joint distribution adaptation. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2013.

[11] M. Long, J. Wang, G. Ding, J. Sun, and P. S. Yu. Transfer joint matching for unsupervised domain adaptation. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 1410–1417, Washington, DC, USA, 2014. IEEE Computer Society.

[12] M. Long, H. Zhu, J. Wang, and M. I. Jordan. Deep transfer learning with joint adaptation networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 2208–2217, 2017.

[13] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22(2):199–210, Feb 2011.

Table 22. The accuracy with different parameter η in MEDA. The other parameter ρ is set as 0.1.

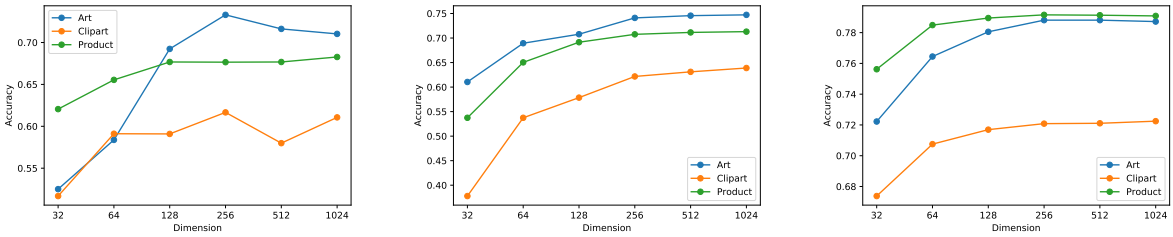
η	0.01	0.1	0.3	0.5	0.8	1.0
A→R	**0.770432**	0.766988	0.736455	0.721304	0.707989	0.702020
C→R	**0.718320**	0.715794	0.704775	0.694674	0.681129	0.676997
P→R	0.778007	**0.785124**	0.779844	0.772039	0.761249	0.759183

Table 23. The accuracy with different parameter ρ in MEDA. The other parameter η is set as 0.1.

ρ	0.01	0.1	0.5	1.0	3.0	5.0
A→R	**0.782828**	0.780992	0.774334	0.767218	0.741276	0.723600
C→R	0.714646	0.716713	**0.717631**	0.715794	0.706612	0.699265
P→R	**0.789486**	**0.789486**	0.787879	0.785124	0.778696	0.775712

Table 24. The accuracy of traditional methods: TCA, SA, and MEDA when dimension changes.

Method	Task	dim=32	dim=64	dim=128	dim=256	dim=512	dim=1024
TCA	A→R	0.525023	0.583792	0.692378	**0.733011**	0.716253	0.710284
	C→R	0.516758	0.591139	0.590909	**0.616621**	0.579890	0.610652
	P→R	0.620523	0.655418	0.676768	0.676538	0.676768	**0.682736**
SA	A→R	0.610422	0.689393	0.707759	0.741047	0.745632	**0.747245**
	C→R	0.377870	0.537420	0.578512	0.621671	0.631084	**0.638889**
	P→R	0.537420	0.650367	0.691460	0.70753	0.711433	**0.713039**
MEDA	A→R	0.722222	0.764463	0.780533	0.788108	0.788108	**0.787190**
	C→R	0.673783	0.707530	0.716942	0.720845	0.721074	**0.722452**
	P→R	0.756198	0.789486	0.789486	**0.791552**	0.791322	0.790863



(a) TCA:the relation of accuracy and dimension (b) SA:the relation of accuracy and dimension (c) MEDA:the relation of accuracy and dimension

Figure 47. The curves of accuracy over the dimension changing. The accuracy of both MEDA and SA increases with dimension increasing among all three tasks while TCA has the highest accuracy in different dimension in different tasks.

[14] B. Sun, J. Feng, and K. Saenko. Correlation alignment for unsupervised domain adaptation. *CoRR*, abs/1612.01939, 2016.

[15] B. Sun and K. Saenko. Deep coral: Correlation alignment for deep domain adaptation. In *European Conference on Computer Vision*, pages 443–450. Springer, 2016.

[16] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell. Deep domain confusion: Maximizing for domain invariance. *CoRR*, abs/1412.3474, 2014.

[17] H. Venkateswara, J. Eusebio, S. Chakraborty, and S. Panchanathan. Deep hashing network for unsupervised domain adaptation. In *(IEEE) Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[18] J. Wang, Y. Chen, S. Hao, W. Feng, and Z. Shen. Balanced distribution adaptation for transfer learning. *CoRR*, abs/1807.00516, 2018.

[19] J. Wang, W. Feng, Y. Chen, H. Yu, M. Huang, and P. S. Yu. Visual domain adaptation with manifold embedded distribution alignment. In *Proceedings of the 26th ACM International Conference on Multimedia*, MM '18, pages 402–410, New York, NY, USA, 2018. ACM.

[20] W. Zhang, W. Ouyang, W. Li, and D. Xu. Collaborative and adversarial network for unsupervised domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3801–3809, 2018.

[21] Y. Zhang, H. Tang, K. Jia, and M. Tan. Domain-symmetric networks for adversarial domain adaptation. *arXiv preprint arXiv:1904.04663*, 2019.

Table 25. The accuracy rank of all listed methods: 10 traditional methods and 10 deep methods (GAN not included).

Rank	Method	A→R	C→R	P→R	Average
1	MEDA	**0.780533**	**0.716942**	**0.789486**	**0.762320**
2	CDAN	0.765894	0.713335	0.778747	0.752659
3	iCAN	0.761500	0.693800	0.767000	0.740767
4	Deep CORAL	0.765200	0.689900	0.750100	0.735067
5	DANN	0.759700	0.677100	0.754200	0.730333
6	DAN	0.757400	0.676400	0.743900	0.725900
7	Fine-tuned ResNet(baseline)	0.744549	0.666055	0.748221	0.719608
8	SVM(baseline)	0.743342	0.647153	0.742195	0.710897
9	SymNet	0.744549	0.656874	0.729401	0.710275
10	JAN	0.736286	0.653890	0.739500	0.709892
11	DDC	0.742700	0.658300	0.728300	0.709767
12	SA	0.745638	0.632461	0.712351	0.696817
13	TCA	0.733012	0.616621	0.676538	0.675390
14	IW	0.730026	0.602388	0.687328	0.673247
15	TJM	0.684803	0.615473	0.711433	0.670570
16	GFK	0.660468	0.589991	0.696740	0.649066
17	JDA	0.675620	0.570478	0.690771	0.645623
18	CORAL	0.648531	0.581497	0.684803	0.638277
19	BDA	0.648072	0.579201	0.668274	0.631849
20	AMEAN	0.571500	0.514400	0.574500	0.553467

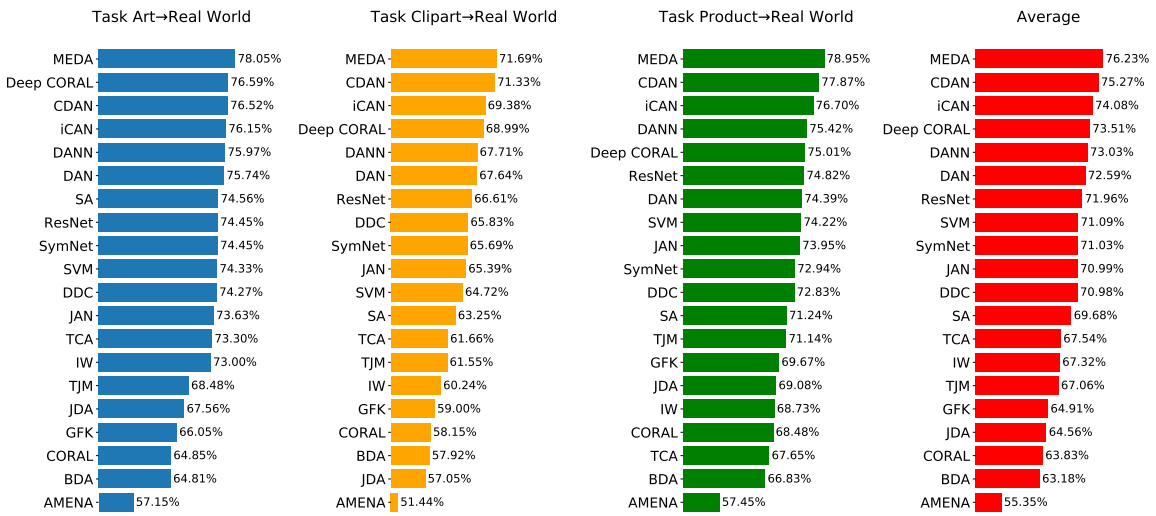


Figure 48. The rank of three tasks and average among all listed methods.