

Feature Encoding for Image Classification

Haowei Huang
516021910491
Shanghai Jiao Tong University
Shanghai, China
1270927224@qq.com

Zhixin Lin
516021910495
Shanghai Jiao Tong University
Shanghai, China
1069066484@qq.com

Yaojie Ding
516021910430
Shanghai Jiao Tong University
Shanghai, China
416914846@qq.com

Abstract—Abstract of Project4.

I. INTRODUCTION

Introduction of Project4.

II. APPROACHES

In this section, we present our methods applied for classification. And we will emphasize differences of our approaches compared with standard ones.

A. Traditional era

1) *TCA*: Pan, Sinno Jialin, et al. proposed to find a representation through a learning method, transfer component analysis(TCA), for domain adaptation in [4]. TCA tries to learn a set of common transfer components underlying both domains such that the difference in data distributions of the different domains, when projected onto this subspace, can be dramatically reduced and data properties can be preserved.

2) *CORAL*: Sun, Baochen, Jiashi Feng, and Kate Saenko proposed a simple, effective, and efficient method for unsupervised domain adaptation called CORrelation ALignment(CORAL) in [5]. CORAL minimizes domain shift by aligning the second-order statistics of source and target distributions, without requiring any target labels.

3) *KMM*: Huang, Jiayuan, et al. presented a nonparametric method which directly produces resampling weights without distribution estimation to deal with data adaptation in [6]. Their method is called kernel mean matching(KMM), which works by matching distributions between training and testing sets in feature space.

4) *JDA*: Mingsheng Long, et al. put forward a novel transfer learning approach, referred to as Joint Distribution Adaptation(JDA) in [7]. Specifically, JDA aims to jointly adapt both the marginal distribution and conditional distribution in a principled dimensionality reduction procedure, and construct new feature representation that is effective and robust for substantial distribution difference.

B. DaNN

Muhammad Ghifary, et al. propose a simple neural network model to deal with the domain adaptation problem in [8]. This model incorporates the Maximum Mean Discrepancy(MMD) measure as a regularization in the supervised learning to reduce the distribution mismatch between the source and target

domains in the latent space. This model is called Domain Adaptive Neural Network(DaNN), which in fact is a variant of the standard feed forward neural network.

The way this model runs can be described as follows:

- 1) Calculate the loss function. Given the labeled source data $\{\mathbf{X}_s^{(i)}, \mathbf{y}_s^{(i)}\}_{i=1, \dots, n_s}$ and the unlabeled target data $\{\mathbf{x}_t^{(j)}\}_{j=1, \dots, n_t}$, the loss function of a single layer DaNN is given by

$$J_{\text{DaNN}} = J_{\text{NNS}} + \gamma \mathcal{MMD}_e^2(\mathbf{q}_s, \bar{\mathbf{q}}_t) \quad (1)$$

where $J_{\text{NNS}} = -\frac{1}{n_s} \sum_{i=1}^{n_s} \sum_{k=1}^l \left(\left[\mathbf{y}_s^{(i)} \right]_k \log \left(\left[f \left(\mathbf{x}_s^{(i)} \right) \right]_k \right) \right)$, $\mathbf{q}_s = \mathbf{W}_1^\top \mathbf{x}_s + \mathbf{b}$, $\bar{\mathbf{q}}_t = \mathbf{W}_1^\top \mathbf{x}_t + \mathbf{b}$ are the linear combination outputs before the activation, and γ is the regularization constant controlling the importance of MMD contribution to the loss function.

- 2) Minimize the loss function. We choose the Gaussian kernel as the kernel function of the form $k_G(\mathbf{x}, \mathbf{y}) = \exp \left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2s^2} \right)$, where s is the standard deviation and rewrite $\mathcal{MMD}_e^2(\cdot, \cdot)$ as:

$$\begin{aligned} \mathcal{MMD}_e^2(\mathbf{U}_1^\top \mathbf{x}_s, \mathbf{U}_1^\top \mathbf{x}_t) = & \frac{1}{n_s^2} \sum_{i,j=1}^{n_s} \exp \left(-\frac{(\mathbf{x}_s^{(i)} - \mathbf{x}_s^{(j)})^\top \mathbf{U}_1 \mathbf{U}_1^\top (\mathbf{x}_s^{(i)} - \mathbf{x}_s^{(j)})}{2s^2} \right) \\ & + \frac{1}{n_t^2} \sum_{i,j=1}^{n_t} \exp \left(-\frac{(\mathbf{x}_t^{(i)} - \mathbf{x}_t^{(j)})^\top \mathbf{U}_1 \mathbf{U}_1^\top (\mathbf{x}_t^{(i)} - \mathbf{x}_t^{(j)})}{2s^2} \right) \\ & - \frac{2}{n_s n_t} \sum_{i,j=1}^{n_s, n_t} \exp \left(-\frac{(\mathbf{x}_s^{(i)} - \mathbf{x}_t^{(j)})^\top \mathbf{U}_1 \mathbf{U}_1^\top (\mathbf{x}_s^{(i)} - \mathbf{x}_t^{(j)})}{2s^2} \right) \end{aligned}$$

In the implementation, we separate the minimization of J_{NNS} and $\mathcal{MMD}_e^2(\cdot, \cdot)$ into two step. Firstly, J_{NNS} is minimized using a *mini-batched* stochastic gradient descent with respect to \mathbf{U}_1 update. Then, $\mathcal{MMD}_e^2(\cdot, \cdot)$ is minimized by re-updating \mathbf{U}_1 . The latter step is accomplished by a full-batched gradient descent.

DaNN is rather a simple neural network(with only 1 hidden layer) for domain adaptation. But its idea is important that brings MMD(maximum mean discrepancy) for adaptation in neural network.

C. ADDA

Eric Tzeng, et al. summarized a generalized architecture for adversarial domain adaptation and introduced a method of

domain adaptation, Adversarial Discriminative Domain Adaptation, in his work [1]. Using their summarization, ADDA is a combination of generative and discriminative neural network model that uses untied weight sharing between source mapping and target mapping and a GAN loss.

The general ADDA approach is presented in figure 1. There are overall four relatively separated subnetworks within the ADDA model:

- 1) Source encoder network, M_s . Source encoder take source data set as input and output the encoded source features.
- 2) Target encoder network, M_t . Target encoder take source data set as input and output the encoded target features.
- 3) Discriminator network, D . Discriminator take encoded source features and encoded target features and tries to identify which come from source dataset and target dataset.
- 4) Classifier network, C . Classifier network take encoded features, from either source or target domain, as input and output the class prediction.

According to work of Eric Tzeng, et al., we can divide the way the model runs into three stages:

- 1) Pre-training. In this stage, we feed source training data, X_s for source encoder network and use the output features, $M_s(X_s)$, to feed classifier network and use cross entropy as classification loss, L_c :

$$L_c(X_s, Y_s) = -\mathbb{E}_{(x_s, y_s) \sim (X_s, Y_s)} \sum_{k=1}^K \mathbb{I}_{[k=y_s]} \log C(M_s(X_s)) \quad (5)$$

to train the source network and classifier network. After that, both source network and classifier network are fixed. K is the total number of classes.

- 2) Adversarial adaptation. In this stage, we use the idea of GAN to train M_t to generate features, $M_t(X_t)$, to be similarly distributed as $M_s(X_s)$. We feed X_s and X_t for M_s and M_t respectively and combination of $M_s(X_s)$ and $M_t(X_t)$ for D . We in turn optimize D to minimize L_D :

$$L_D(X_s, X_t, M_s, M_t) = -\mathbb{E}_{(x_s) \sim (X_s)} \log D(M_s(X_s)) - \mathbb{E}_{(x_t) \sim (X_t)} \log(1 - D(M_t(X_t))) \quad (6)$$

and optimize M_t to minimize L_t :

$$L_t(X_s, X_t, D) = -\mathbb{E}_{(x_t) \sim (X_t)} \log D(M_t(X_t)) \quad (7)$$

D tries to distinguish $M_s(X_s)$ and $M_t(X_t)$ while M_t want to deceive D .

Our approach is not always quite standard as Eric Tzeng's work. Except that we use fully connected layers instead of CNN for our X_s and X_t , we also initialized parameters of M_t using pre-trained M_s 's. This is not mentioned in the paper and it is not likely to be feasible in most cases. We can do so because our structure of

M_s is designed to be the same as M_t . And the measure really help a lot.

- 3) Testing. In this stage, we feed $M_t(X_t)$ for C and evaluate the classification accuracy.

With our practice, we combined the last two stages into one, just say adversarial adaptation. ADDA method is just like trying to encode the target domain data, $M_t(X_t)$, to match the distribution of encoded source domain data, $M_s(X_s)$ so that the classifier working on source domain data can also work on target domain data.

D. DANN

Yaroslav Ganin, et al. proposed a representation learning approach for domain adaptation in their work [2]. DANN can also be viewed as an instance of Eric Tzeng, et al.'s summarization of domain adaptation method. Compared with ADDA, in general, the only difference of DANN is that DANN is an architecture of tied weight sharing. The general architecture is presented in figure 2; it's simpler than ADDA. Domain classifier D of DANN performs similar function as discriminator D of ADDA does. DANN is similar to ADDA but target encoder and source encoder share the same weight weights. In other words, only one feature encoder, M , is used within DANN. Thus, we do not go to details of components of DANN and just simply present the way the DANN model work. We will use similar representations as ADDA to make things easy to understand (although they may not be Yaroslav Ganin et al.'s symbols).

- 1) Pre-training. In this stage, we optimize M and C to minimize L_C .
- 2) Adversarial adaptation. In this stage, we feed both X_s and X_t for M . By fixing C , we optimize M and D to minimize L_{da} :

$$L_{da}(X_s, X_t, M) = -(\mathbb{E}_{(x_s) \sim (X_s)} \log D(M(X_s)) + \mathbb{E}_{(x_t) \sim (X_t)} \log(1 - D(M(X_t)))) \times c_{da} - \mathbb{E}_{(x_s, y_s) \sim (X_s, Y_s)} \sum_{k=1}^K \mathbb{I}_{[k=y_s]} \log C(M(X_s)) \quad (8)$$

Unlike ADDA, parameters of M and D are updated at the same time. And the loss, L_{da} of DANN is combination of both L_C and L_D of ADDA. There is subtle difference between ADDA and DANN in this stage. c_{da} is a balance factor for classification loss and discriminator loss. c_{da} is not mentioned in the paper, but we use it and find it helps improve our model. We use $c_{da} = 2.0$.

- 3) Testing. In this stage, we feed $M(X_t)$ for C and evaluate the classification accuracy.

Also, we combined stage of adversarial adaptation and testing together as adversarial adaptation. Compared to ADDA method that tries to bring $M_t(X_t)$ to $M_s(X_s)$ as closer as possible, DANN method is just like trying to adjust a well-trained source domain encoder M_s to work on target domain data, X_t .

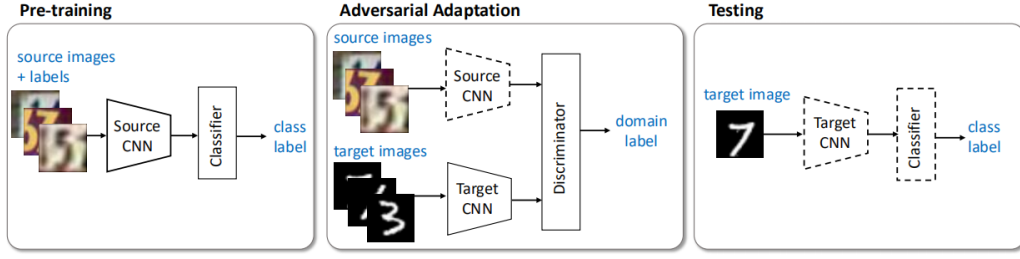


Fig. 1. ADDA Overview: An overview of standard ADDA architecture. Dashed lines indicate fixed network parameters in the indicated stage.

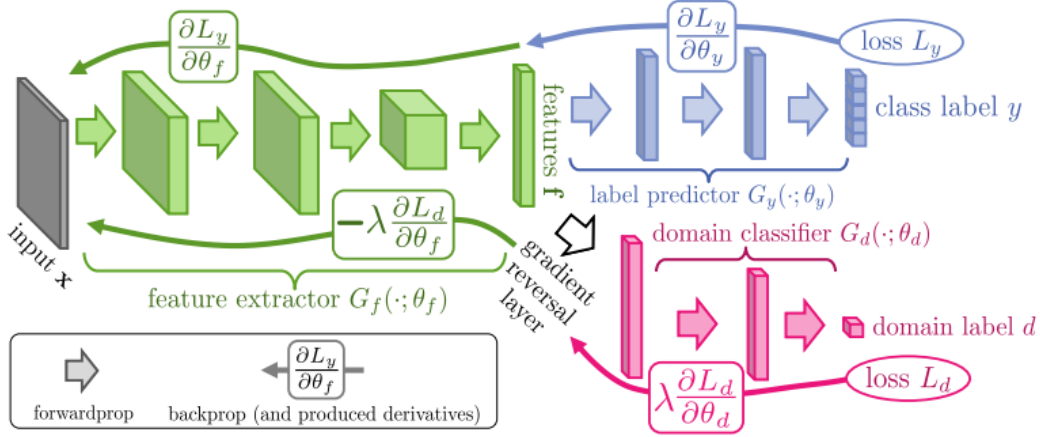


Fig. 2. DANN Overview: An overview of standard DANN architecture.

III. EXPERIMENTS AND RESULTS

A. Traditional era

In this section we will show the experiments that we have done by using traditional domain adaptation methods. First of all, in Table I, we will show the test accuracy of some classification methods on target domain if we do not any apply domain adaptation method. In this part, we train the classifiers on source domain and test them on target domain. 1NN means k-NN method with $k = 1$. In this section, we use X_s , X_t , X_{t_new} , X_{s_new} respectively to denote the features of source domain, the features of target domain, the features of source domain which has been reconstructed by domain adaptation method and the features of target domain which has been reconstructed by domain adaptation method. We use *A2R* to show the source domain is Art and the target domain is RealWorld. *C2R* and *P2R* are similar to *A2R*. And *SVM_lin* means SVM with linear kernel while *SVM_RBF* means SVM with RBF kernel.

TABLE I
TESTING ACCURACY WITHOUT DOMAIN ADAPTATION METHOD

Src	Tar	1NN	SVM(linear)	SVM(RBFC=1)	SVM(RBFC=5)
<i>Art</i>	<i>Real</i>	0.6580	0.7310	0.7436	0.7473
<i>Clipart</i>	<i>Real</i>	0.5864	0.6013	0.6488	0.6511
<i>Product</i>	<i>Real</i>	0.6957	0.6879	0.7266	0.7296

1) *TCA Experiments*: In this section we have done three parts of experiments. The experiments of the first two parts have been shown in Table II, Table III and Table IV

TABLE II
A2R: PART1 AND PART2

testing accuracy \ dim \ classifier	32	64	128	256	512	1024	2048
<i>SVM_lin</i>	0.7067	0.7434	0.7540	0.7563	0.7572	0.7567	0.7567
<i>SVM_RBF</i> ($C = 1$)	0.7122	0.7418	0.7519	0.7466	0.7464	0.7448	0.7434
<i>SVM_RBF</i> ($C = 5$)	0.7207	0.7519	0.7590	0.7599	0.7604	0.7602	0.7602

TABLE III
C2R: PART1 AND PART2

testing accuracy \ dim \ classifier	32	64	128	256	512	1024	2048
<i>SVM_lin</i>	0.6282	0.6523	0.6612	0.6677	0.6688	0.6695	0.6693
<i>SVM_RBF</i> ($C = 1$)	0.6440	0.6553	0.6550	0.6543	0.6530	0.6527	0.6530
<i>SVM_RBF</i> ($C = 5$)	0.6369	0.6534	0.6587	0.6562	0.6557	0.6562	0.6553

TABLE IV
P2R: PART1 AND PART2

testing accuracy \ dim \ classifier	32	64	128	256	512	1024	2048
<i>SVM_lin</i>	0.7101	0.7287	0.7354	0.7377	0.7404	0.7406	0.7402
<i>SVM_RBF</i> ($C = 1$)	0.7248	0.7429	0.7372	0.7393	0.7395	0.7395	0.7393
<i>SVM_RBF</i> ($C = 5$)	0.7131	0.7310	0.7296	0.7305	0.7305	0.7310	0.7308

In these two parts we have set the kernel method TCA uses to be 'primal' which means TCA does not apply kernel method.

The first part is about different dimensions of features of source domain and target domain after TCA process. From the three tables above, we can find a interesting phenomenon that with the number of dimensions growing, all the testing accuracy of different classifier and different combination of source domain and target domain initially goes higher but stays stable when dimensions have reach 512. We guess the reason is that with the reconstruction of TCA, when X_{s_new} dimensions reach 512, X_{s_new} and X_{t_new} contains enough information. The Maximum Mean Discrepancy becomes small between source domain and target domain and different domains are close to each other and testing accuracy reach a high point.

The second part is about different classifiers. We can find for all $A2R$, $C2R$ and $P2R$, when the number of dimensions is small, SVM_RBF performs better than SVM_lin while the case is the contrary when dimensions is bigger. This is because SVM_lin is suitable for the situation that number of features and number of samples are close and SVM_RBF is suitable for the situation that the number of features is small. And for the hyper-parameter C in SVM_RBF , we find a bigger C can performs better on $A2R$ and $C2R$ but a small C is better on $P2R$.

Then we show the result of the third part in Table V, Table VI and Table VII.

TABLE V
A2R: PART3

testing accuracy classifier	TCA kernel type		
	primal	linear	RBF
SVM_lin	0.7567	0.4983	0.5554
SVM_RBF	0.7434	0.1958	0.02454

TABLE VI
C2R: PART3

testing accuracy classifier	TCA kernel type		
	primal	linear	RBF
SVM_lin	0.6693	0.5625	0.5965
SVM_RBF	0.6530	0.3002	0.3514

TABLE VII
P2R: PART3

testing accuracy classifier	TCA kernel type		
	primal	linear	RBF
SVM_lin	0.7402	0.6635	0.6945
SVM_RBF	0.7393	0.5280	0.5843

For this part we have set dimensions to be 2048 and try to find the difference between different kernel method in TCA. From Table V we can find that we get rather bad results with TCA(RBF) and TCA(linear). Besides, in the experimental

process, the run time of TCA(RBF) and TCA(linear) is very long.

2) *CORAL Experiments*: As Baochen Sun, et al. has pointed in [5], CORAL is a "frustratingly easy" domain adaptation method which does not have many parameters. So in this section, we have compared the performances of CORAL on different source-target combinations and different classifier. From Table VIII we can find SVM_RBF performs

TABLE VIII
CORAL EXPERIMENTS

testing accuracy Src&Tar	classifier			
	1NN	SVM_lin	$SVM_RBF(C = 1)$	$SVM_RBF(C = 5)$
$A2R$	0.6488	0.7090	0.7390	0.7450
$C2R$	0.5830	0.5997	0.6514	0.6504
$P2R$	0.6851	0.6906	0.7310	0.7276

much better than 1NN and SVM_lin . The change of hyper-parameter C does not have a big influence on the performance. And we can compare the performance of CORAL with results that have been shown in Table I. The performance of CORAL is even worse than the situation without domain adaptation method. This may be the price of "frustrating easy".

3) *KMM Experiments*: In this section, we have change the kernel type that KMM uses to see the different performance of KMM. Of course we have tried different classifier and apply them on different source domain and target domain.

TABLE IX
KMM EXPERIMENTS ON A2R

testing accuracy classifier	kernel type	
	linear	RBF
SVM_lin	0.7296	0.7312
$SVM_RBF(C = 1)$	0.6729	0.7356
$SVM_RBF(C = 5)$	0.6723	0.7471

TABLE X
KMM EXPERIMENTS ON C2R

testing accuracy classifier	kernel type	
	linear	RBF
SVM_lin	0.6004	0.5143
$SVM_RBF(C = 1)$	0.5974	0.6518
$SVM_RBF(C = 5)$	0.5116	0.6518

TABLE XI
KMM EXPERIMENTS ON P2R

testing accuracy classifier	kernel type	
	linear	RBF
SVM_lin	0.6876	0.6927
$SVM_RBF(C = 1)$	0.6617	0.7299
$SVM_RBF(C = 5)$	0.6633	0.7273

From Table IX, Table X and Table XI we can find when KMM uses linear kernel, SVM_lin performs better than SVM_RBF . When KMM uses RBF kernel, the case is the

contrary. We conclude that the kernel method used in KMM and the kernel method used in SVM should be consistent.

4) *JDA Experiments*: In this part we will show performances of JDA with different dimensions after transfer. In this experiment, we use SVM_{RBF} as classifier and the hyper-parameter C of SVM is 1. Similar to the results of TCA,

TABLE XII
JDA EXPERIMENTS

testing accuracy	dim	32	64	128	256	512	1024	2048
Src&Tar								
$A2R$		0.7092	0.7462	0.7530	0.7471	0.7464	0.7464	0.7457
$C2R$		0.6543	0.6638	0.6656	0.6628	0.6601	0.6605	0.6608
$P2R$		0.7351	0.7459	0.7436	0.7471	0.7482	0.7478	0.7478

with the increase of dimensions, the testing accuracy initially increase and then reach the highest point. Finally the testing accuracy stay stable. This similarity can be explained by the fact that TCA and JDA both use Maximum Mean Discrepancy (MMD) to measure the difference between the distribution of source domain and the distribution of target domain.

B. DaNN Experiments

We will present five parts of experiments in this section to evaluate the influence of different parameter in this model. Since DaNN is rather a simple neural network with only 1 hidden layer for domain adaptation, there are not many details of building this network. In the all five parts of experiments, we set $Learning_rate = 0.05$, $Epochs = 30$. For each combination of source domain and target domain, we use the data in source domain to be the training set and the data in target domain to be the testing set. Besides, in this section we use λ to denote the coefficient of mmd_loss in the calculation of total loss, DR to denote the drop rate and BS to denote the batch size.

For the first part, we show the change of training accuracy, testing accuracy, training loss and testing loss during the running process of this network. All values of the loss curves are scaled to interval $[0.0, 1.0]$ for a better observation. For this part, we have set $\lambda = 1$, $DR = 0.5$, $BS = 64$. The combination of target domain and source domain we choose is $P2R$. Figure 3 shows the result.

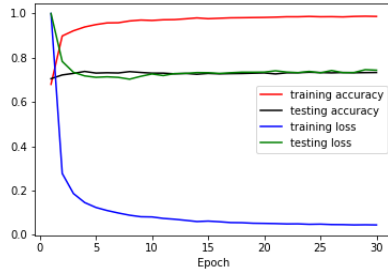


Fig. 3. DaNN: part 1

From figure 3 we can find both training accuracy and testing accuracy have a quick convergence. However, there exists a

problem that though training accuracy and training loss have a nice trend, the testing accuracy stay stable at about 70% and the testing loss stay at a high level. In fact, the result is only a little better than the result of scenario without domain adaptation method.

For the second part, we focus on different combination of source domain and target domain. We set $\lambda = 1$, $DR = 0.5$, $BS = 64$ in this part and figure 4 shows the result.

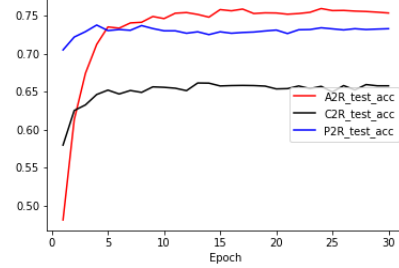


Fig. 4. DaNN: part 2

From figure 4 we notice that $A2R$ converge more slowly than $C2R$ and $P2R$ but it can reach the highest testing accuracy. The accuracy of $P2R$ is slightly lower than $A2R$ while the testing accuracy of $C2R$ is much lower. The result is consistent with the results of most traditional methods and the situation without domain adaptation method.

Then we begin to evaluate the influence of some parameters of this model.

The third part is about parameter λ . λ in this model means the coefficient of mmd_loss in the calculation of total loss which theoretically can change the optimized direction. In this part we set $DR = 0.5$, $BS = 64$ and choose $A2R$ be the combination of source domain and target domain. Figure 5 shows the result.

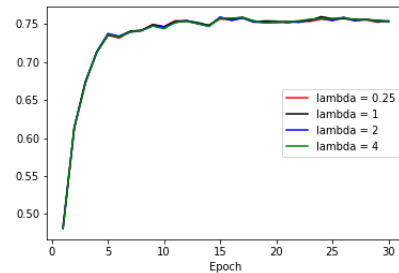


Fig. 5. DaNN: part 3

Something strange happens in figure 5. Almost all the curves are coincident with others, which means that parameter λ does not play a key role in DaNN model. But there must be some other reason. We think it is because DaNN is used to deal with images but we change it to deal with deep learning feature. This change may make λ useless.

The fourth part is about parameter BS . BS means the batch size of every training batch. In this part we set $DR = 0.5$,

$\lambda = 1$ and choose $A2R$ be the combination of source domain and target domain. Figure 6 shows the result.

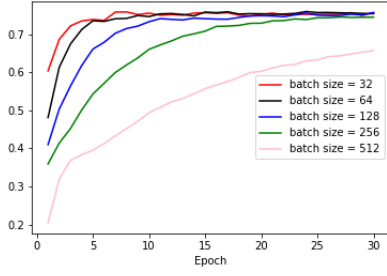


Fig. 6. DaNN: part 4

From figure 6 we can easily find that a small batch size is better than a big batch size in this experiment. A smaller batch size can converge more quickly and when batch size is too big, its testing accuracy can not get as high as a small batch size.

The fifth part is about parameter DR . DR means drop rate, which is used to deactivate neuron and suppress overfitting. In this part we set $BS = 64$, $\lambda = 1$ and choose $A2R$ be the combination of source domain and target domain. Figure 7 shows the result.

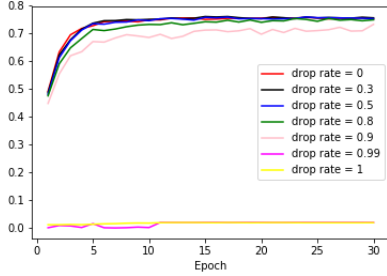


Fig. 7. DaNN: part 5

From figure 7 we find when $DS \leq 0.8$, there is almost no difference. When $DS = 0.9$, the accuracy begin to flop. And when $DS \geq 0.99$, the accuracy is close to 0. It is because when drop rate is too high, most neurons are deactivated and there are not enough information to be output.

C. ADDA Experiments

We will present six experiments in this part. We use fully connected networks (FCNs) to construct the four subnetworks. First we show some common parameters of all of these experiments in table XIII. We split both X_s and X_t into training set, X_s^{tr} and X_t^{tr} , and test set, X_s^{te} and X_t^{te} . X_s^{tr} is used to train ADDA the first stage and X_s^{te} is used to select the best parameters of M_s and C correspondingly, preventing overfitting. X_t^{te} is used to train ADDA the second stage and X_t^{te} is used to select the best parameters of M_t and D correspondingly, also preventing overfitting. It should be mentioned that both X_t^{te} and X_t^{tr} are unlabeled.

TABLE XIII
HYPER-PARAMETERS OF ADDA

Symbol	Value	Description
K	65	Number of classes
s	$5e-5$	Optimization step
b	256	Batch size
l_e	2048, 2048	Layers of M_s and M_t
l_c	1024, K	Layers of C , connected from oM_c or M_t
l_d	512, 512, 2	Layers of D , connected from M_c or M_t
p_k	0.05	Dropout probability of a neuron.
l_2	$1e-5$	L2 regularization for weight parameters.
r_s^{tr}	0.9	Ratio of X_s used as training set.
r_t^{tr}	0.6	Ratio of X_t used as training set.
I	8000	Total iterations of first two training stages.

Then we present a overview of different configurations of the six experiments and the corresponding results in table XIV. Target domain of all of the experiments presented is real world. Src is short for source data domain. PT stands for whether to transfer parameters of M_s to M_t for M_t 's initialization. AN means whether to apply learning rate annealing. BN means whether to apply batch normalization for each layer. Besides, we uses RELU function for hidden layer's activation and adds a softmax layer before calculating cross entropy. With learning rate annealing applied, we have learning rate of stage pre-training and adversarial adaptation to be $l_{r1} = \frac{1.5 \times s}{1.0 + 0.01 \times i}$ and $l_{r2} = \frac{0.1 \times s}{1.0 + 0.01 \times i}$, where i it the iteration number. Otherwise, $l_{r1} = s$ and $l_{r2} = 0.05 \times s$. We acquired these formulas by preliminary experiments.

$Ac(x)^*$ indicates the best accuracy towards indicator x . For example, in the formula $Ac(M_t(X_t^{te}))^*$, X_t^{te} , M_t and Ac^* stands for test set of unlabeled target domain, target encoder and the best accuracy respectively. Thus, $Ac(M_t(X_t^{te}))^*$ means the best test classification accuracy towards target test set using target encoder. $Ac(M_t(X_t^{te}))^*$ is actually the final classification evaluation of our domain adaptation method. Similarly, $Ac(M_s(X_s^{te}))^*$ represents the best accuracy of the classifier towards source test dataset using source encoder, which can be viewed as an upper bound of $Ac(M_t(X_t^{tr}))^*$. $Ac(M_t(X_t^{tr}))^*$ represents the best accuracy of the classifier towards target training dataset using target encoder, which also can be viewed as an upper bound of $Ac(M_t(X_t^{te}))^*$. So, we theoretically have $Ac(M_s(X_s^{te}))^* \geq Ac(M_t(X_t^{tr}))^* \geq Ac(M_t(X_t^{te}))^*$. As for $Ac(M_s(X_t^{te}))^*$, it represents the best accuracy of the classifier towards target test dataset using source encoder. We can use $g = Ac(M_t(X_t^{te}))^* - Ac(M_s(X_t^{te}))^*$ as the performance gain with our domain adaptation method.

Now, we go to details of the six experiments. The figures 8, 9, 10, 11, 12 and 13 show visualization of the results. Four each figure, there are four subfigures showing target dataset distribution before and after domain adaptation, classifier training history in stage pre-training, discriminator training history in stage adversarial adaptation and accuracies of X_t changing in stage adversarial adaptation. Remember we combine test stage into adversarial adaptation stage. In the fourth subfigure, acc(t_ec, training set), acc(t_ec, test set), acc(s_ec,

TABLE XIV
DIFFERENT CONFIGURATIONS AND RESULTED PERFORMANCES OF ADDA EXPERIMENTS

Sym	Src	PT	AN	BN	$Ac(M_s(X_s^{te}))^*$	$Ac(M_s(X_t^{te}))^*$	$Ac(M_t(X_t^{tr}))^*$	$Ac(M_t(X_t^{te}))^*$	g
Ex_1	Art	×	×	×	0.6074	0.5080	0.0222	0.0189	-0.4891
Ex_2	Art	✓	×	×	0.6818	0.5712	0.5897	0.5821	0.0109
Ex_3	Art	✓	✓	×	0.6488	0.6223	0.6551	0.6372	0.0149
Ex_A	Art	✓	✓	✓	0.7893	0.7313	0.7621	0.7354	0.0041
Ex_C	Clipart	✓	✓	✓	0.8624	0.6561	0.6646	0.6636	0.0075
Ex_P	Product	✓	✓	✓	0.9391	0.7313	0.7556	0.7353	0.0040

training set), $acc(s_ec, test\ set)$ stands for $Ac(M_t(X_t^{tr}))$, $Ac(M_t(X_t^{te}))$, $Ac(M_s(X_t^{tr}))$ and $Ac(M_s(X_t^{te}))$ respectively. And all values of the loss curves are scaled to interval $[0.0, 1.0]$ for a better observation.

As illustrated in figure 8, the domain adaptation just shows no effect ($Ac(M_t(X_t^{te}))^* < 0.03$). We would rather use the source encoder M_s for our target dataset and achieves a test accuracy above 0.50. We tried Ex_1 exactly the same as Eric Tzeng et al.'s work. Just we are using features of images instead of images themselves (Maybe we missed something in the paper). By observing transfer visualization (the first subfigure), the adapted target data just moved to another distribution and still obviously separated from the source data. By observing discriminator's training history (the third subfigure), we can see the discriminator loss falls fast and within 800 iterations, the discriminator can successfully distinguish $M_t(X_t)$ and $M_s(X_s)$ no matter how M_t is trained. Thus, we believe it must be that a randomly initialized M_t parameters are too chaotic for M_t to optimize and deceive the discriminator.

To fix the problem, we decided to initialize M_t parameters with trained M_s 's since we think the object features from source and target domain must have a lot in common and M_t can be adjusted from M_s . As illustrated in figure 9, the result is much better. From discriminator's training history, we see GAN training curves. There exists a negative correlation between L_D and L_t , represented by the blue and orange curve in the third subfigure respectively. In this experiment, the $Ac(M_t(X_t^{te}))^*$ overcomes $Ac(M_s(X_t^{te}))^*$ by about 0.01. The domain adaptation shows a positive effect. But, we found a problem that, in the stage of adversarial adaptation, $Ac(M_t(X_t^{te}))$, represented by the orange curve in the fourth subfigure, drops quickly at first (though increases then). And we found this is not a corner case, Eric Tzeng et al. mentioned the problem on the internet and suggested we try decreasing the learning rate.

So we have Ex_3 , shown in 10, we applied method of learning rate annealing. The process of C 's training in the pre-training stage (the second subfigure) improves a lot. It also help ease dropping of $Ac(M_t(X_t^{te}))$. However, still we found the training processes converge too slow in both stages. The C doesn't seem to converge even after 8000 iterations.

To solve the problem, we tried method of batch normalization, as illustrated in 11. The training of classifier C converges within 800 iterations, far more quickly than before. Discriminator's training becomes more gently although D grows too

powerful for M_t to deceive. And $Ac(M_t(X_t^{te}))^*$ grows to be 0.7354, far better than the previous three experiments. We are using the same dataset in the four experiments so far.

We show figures 12 and 13 for completeness of our study. It seems domain adaptation does not work so well when it comes to transfer art dataset into real world dataset compared to clipart and product dataset though the classifier works the best on the source dataset. We think it is because there exists a larger difference between the two domains, real world and clipart.

D. DANN Experiments

We present three experiments using DANN method. Because of time issue, we did not design DANN experiments that sophisticatedly, but we learned a lot from ADDA experiments. Common hyper-parameters are listed in table XV. Tricks of L2 normalization and neuron dropout are not used in the experiments because these tricks did not perform that well as expected in the previous experiments of DANN. And we use sigmoid function for activation of hidden layers. Also, a softmax layer is added after the output layer before calculating cross entropy. Batch normalization is not applied in DANN experiments. DANN networks we designed are much simpler and more light-weighted. Learning rate, l_r , of both two stages anneals according to the formula: $l_r = \frac{0.00004}{(1.0 + \frac{1.0 \times i}{100})^{0.75}}$, where i is the current iteration. Primitive of the formula was proposed in Yaroslav Ganin, et al.'s work. We just adjust some factors according to our preliminary experiments.

TABLE XV
HYPER-PARAMETERS OF DANN

Symbol	Value	Description
K	65	Number of classes
b	256	Batch size
l_e	2048, 1024	Layers of M
l_c	256, K	Layers of C , connected from oM
l_d	512, 512, 2	Layers of D , connected from M
p_k	0.0	Dropout probability of a neuron.
r_s^{tr}	0.75	Ratio of X_s used as training set.
r_t^{tr}	0.75	Ratio of X_t used as training set.
I	8000	Total iterations of first two training stages.

We first present an overview of all of the three DANN experiments in table XVI. Different from ADDA model, we have only one encoder M in DANN. M is trained with (X_s, Y_x) in the first stage and then trained to adapt target domain in the second stage. Thus, to maintain consistency of our symbols, we use M_s to represent the encoder after the

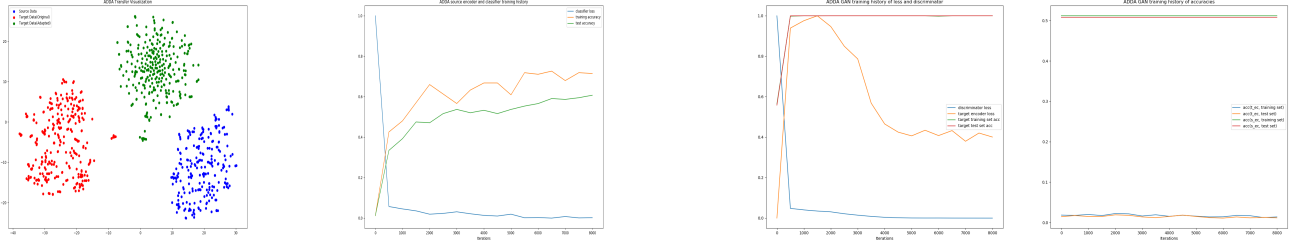


Fig. 8. Visualization of ADDA Ex1 Result

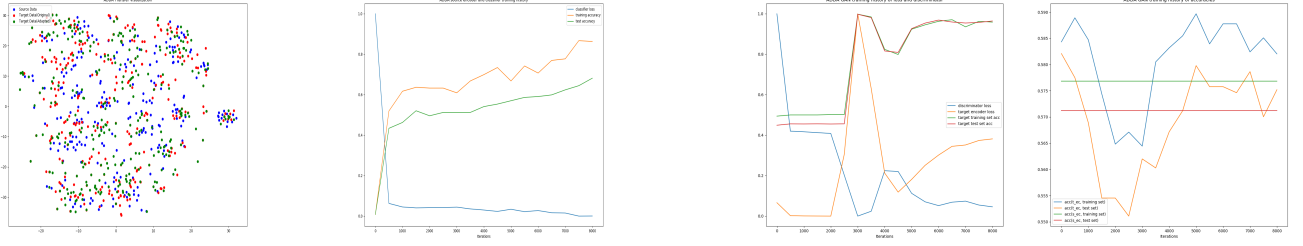


Fig. 9. Visualization of ADDA Ex2 Result

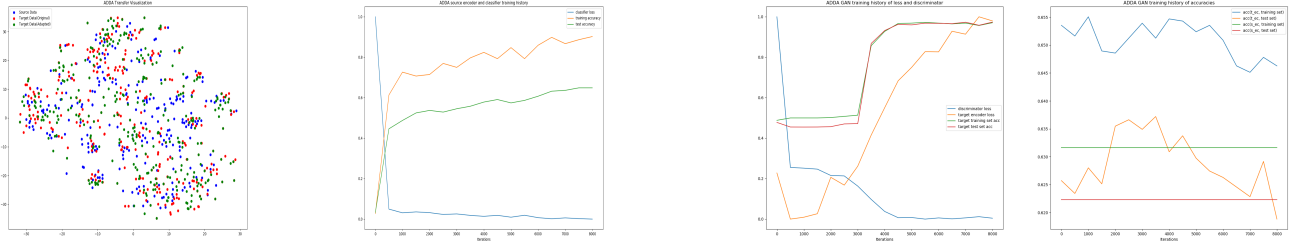


Fig. 10. Visualization of ADDA Ex3 Result

TABLE XVI
DIFFERENT CONFIGURATIONS AND RESULTED PERFORMANCES OF DANN EXPERIMENTS

Sym	Src	$Ac(M_s(X_s^{te}))^*$	$Ac(M_s(X_t^{te}))^*$	$Ac(M_t(X_t^{tr}))^*$	$Ac(M_t(X_t^{te}))^*$	\bar{g}
Ex_A	Art	0.8020	0.7172	0.7488	0.7301	0.0129
Ex_C	Clipart	0.8671	0.6501	0.6683	0.6547	0.0046
Ex_P	Product	0.9596	0.7328	0.7411	0.7392	0.0064

first-stage training and M_t to represent the encoder after the second-stage training.

Compared with the results of ADDA shown in the last three rows of table XIV, except the first-stage trained classification accuracies towards source test set, $Ac(M_s(X_s^{te}))^*$, show better performances, the other three indicators fall. The classifier works better with source domain dataset after pre-training but doesn't show such improvement when it comes to classify adapted target domain data.

Figures 14, 15, 16 shows visualization of the three DANN experiments. The are four subfigures in each figure. The first two subfigures illustrate visualization of encoded source and target data distribution before($M_s(X_s^{te})$, $M_s(X_t^{te})$) and

after($M_t(X_s^{te})$, $M_t(X_t^{te})$) domain adaptation. The third subfigure shows training history of the classifier C in the pre-train stage. We name feature encoder in this stage M_s . Legends of "source accuracy(test)" and "source accuracy(train)" represent $Ac(M_s(X_s^{te}))$ and $Ac(M_s(X_s^{tr}))$ respectively. The fourth figure shows training history of the adversarial adaptation stage. Recall that discriminator D and feature encoder M are trained in this stage, and we name feature encoder in this stage M_t . Legends of "source accuracy(test)", "target accuracy(test)", "source accuracy(train)" and "target accuracy(train)" represent $Ac(M_t(X_s^{te}))$, $Ac(M_t(X_t^{te}))$, $Ac(M_t(X_s^{tr}))$ and $Ac(M_t(X_t^{tr}))$ respectively.

Unlike ADDA, the power of D doesn't improve so

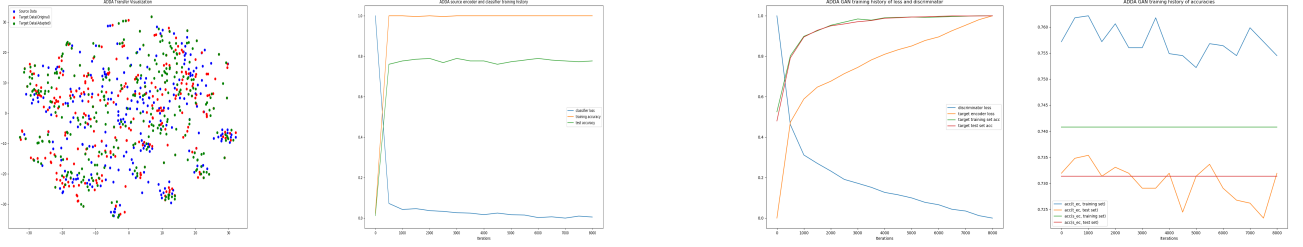


Fig. 11. Visualization of ADDA ExA Result

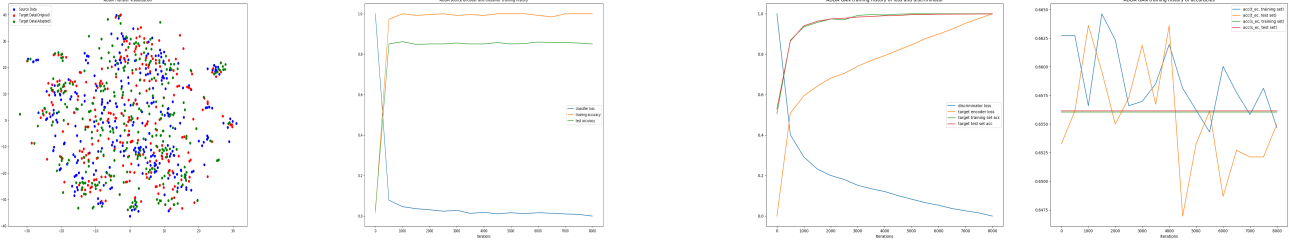


Fig. 12. Visualization of ADDA ExC Result

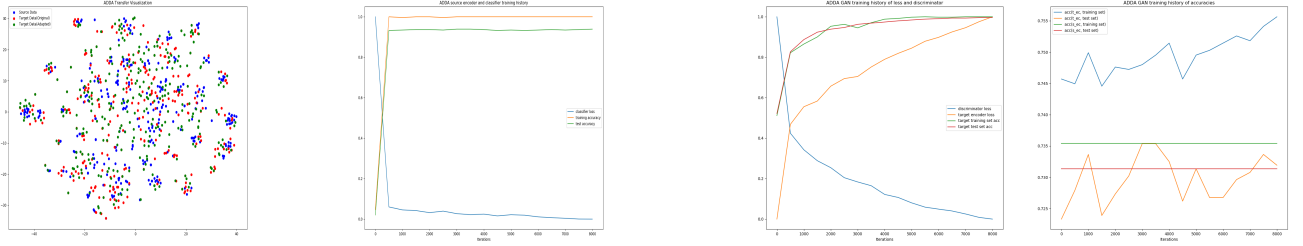


Fig. 13. Visualization of ADDA ExP Result

smoothly. We think it is because we are not minimizing L_C and L_D respectively but minimizing $L_{da} = L_C + L_D$. We want to keep a good performance of classifier C the same time when trying to adjust M to adapt the target domain data.

IV. DISCUSSIONS

A. Problems of Network Design

No matter which deep model we applied, severe overfitting occurs when training the classifier. Take ADDA for example, $Ac(M_s(X_s^{tr}))$ are always greater than $Ac(M_s(X_s^{te}))$ by about 0.15. We tried dropping out some neurons and increases L2 regularization constraint, but these measures do not ease overfitting but brings negative effects on $Ac(M_s(X_s^{te}))$.

Besides, specifically for ADDA, we found the discriminator can really grow too powerful to distinguish X_s and X_t . We can design a more complicated and powerful M to deceive the discriminator, of course. But a more complicated M can also bring difficulties for the classification work.

And, it should be noticed that the visualization doesn't show an obvious effect after domain adaptation and the performance

gain using domain adaptation is also poor, $0.004 \leq g \leq 0.013$. We think there can still be room for improvement.

B. Theory of Batch Normalization

From ADDA experiments, we can see batch normalization really make a learning process converge easier and faster(refer to figure 10 and 11). Method of batch normalization is proposed in Sergey Ioffe et al.'s work [3].

Batch normalization is used before activation function applied to the output of a layer. Assuming that every batch fed for the network satisfy a distribution $d_0(x, \theta_0)$, with the network forwarding, the distribution changes into $d_1(x, \theta_1), d_2(x, \theta_2), \dots, d_i(x, \theta_i)$ and goes far away from the Y-axis, which is named internal covariate shift in Sergey Ioffe et al.'s work [3], as illustrated in the left subfigure of figure 17. Supposing sigmoid function is used for activation, the gradients will become flat, making it difficult for update of parameters using SGD method. As a result, we normalize the output x to normal distribution. But, that is not done because distribution of x is constrained to Gaussian. So a linear

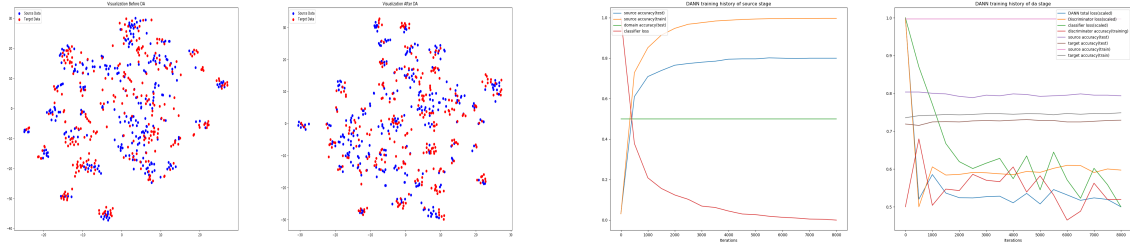


Fig. 14. Visualization of DANN ExA Result

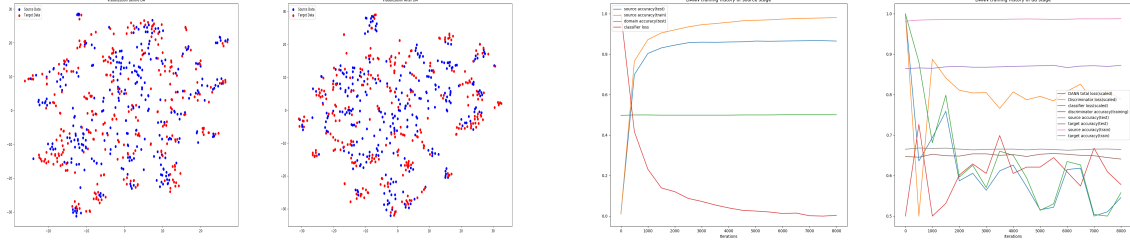


Fig. 15. Visualization of DANN ExC Result

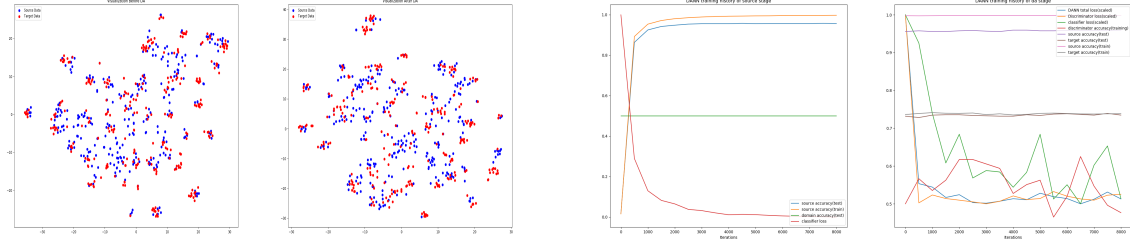


Fig. 16. Visualization of DANN ExP Result

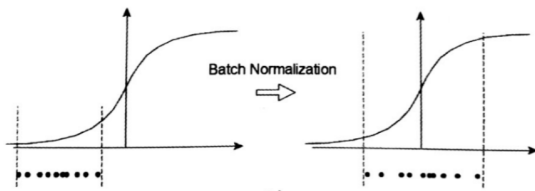


Fig. 17. Problem of Internal Covariate Shift

transformation, scale and shift, is applied to keep the feature of \mathbf{x} 's original distribution. Intuitively, batch normalization aims to pull \mathbf{x} back to where gradients of the activation function work to accelerate the convergence process.

V. CONCLUSION

Conclusions.

REFERENCES

REFERENCES

- [1] Eric Tzeng, Judy Hoffman, Kate Saenko and Trevor Darrel, "Adversarial Discriminative Domain Adaptation".
- [2] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, Francois Laviolette, Mario Marchand, Victor Lempitsky, "Domain-Adversarial Training of Neural Networks".
- [3] Sergey Ioffe, Christian Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift".
- [4] Pan, Sinno Jialin, et al. "Domain adaptation via transfer component analysis." IEEE Transactions on Neural Networks 22.2 (2010): 199-210.
- [5] Sun, Baochen, Jiashi Feng, and Kate Saenko. "Return of frustratingly easy domain adaptation." Thirtieth AAAI Conference on Artificial Intelligence. 2016.
- [6] Huang, Jiayuan, et al. "Correcting sample selection bias by unlabeled data." Advances in neural information processing systems. 2007.
- [7] Long, Mingsheng, et al. "Transfer feature learning with joint distribution adaptation." Proceedings of the IEEE international conference on computer vision. 2013.
- [8] Ghifary, Muhammad, W. Bastiaan Kleijn, and Mengjie Zhang. "Domain adaptive neural networks for object recognition." Pacific Rim international conference on artificial intelligence. Springer, Cham, 2014.