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. 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:
$$\begin{aligned} &L_c(X_s, Y_s) = \\ &\quad -\mathbb{E}_{(x_s,y_s)\sim(X_s,Y_s)} \sum_{k=1}^{K} \mathbb{I}_{[k=y_s]} \log C(M_s(X_s)) \end{aligned} \tag{1}$$
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:
$$\begin{aligned} L_D(X_s, X_t, M_s, M_t) = \\ &- \mathbb{E}_{(x_s)\sim(X_s)} \log D(M_s(X_s)) \\ &- \mathbb{E}_{(x_t)\sim(X_t)} \log(1 - D(M_t(X_t)) \end{aligned} \tag{2}$$
and optimize M_t to minimize L_t:
$$L_t(X_s, X_t, D) = -\mathbb{E}_{(x_t)\sim(X_t)} \log D(M_t(X_t)) \tag{3}$$
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.

During our practice, we combined the last two stages into one, just say adversarial adaptation.

B. 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 G 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

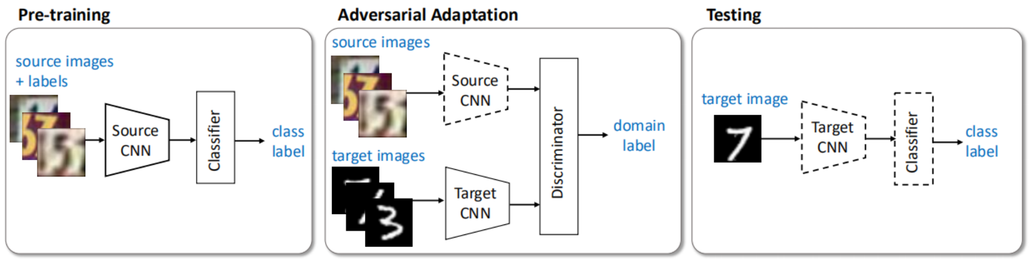

Fig. 1. ADDA Overview: An overview of standard ADDA architecture. Dashed lines indicate fixed network parameters in the indicated stage.

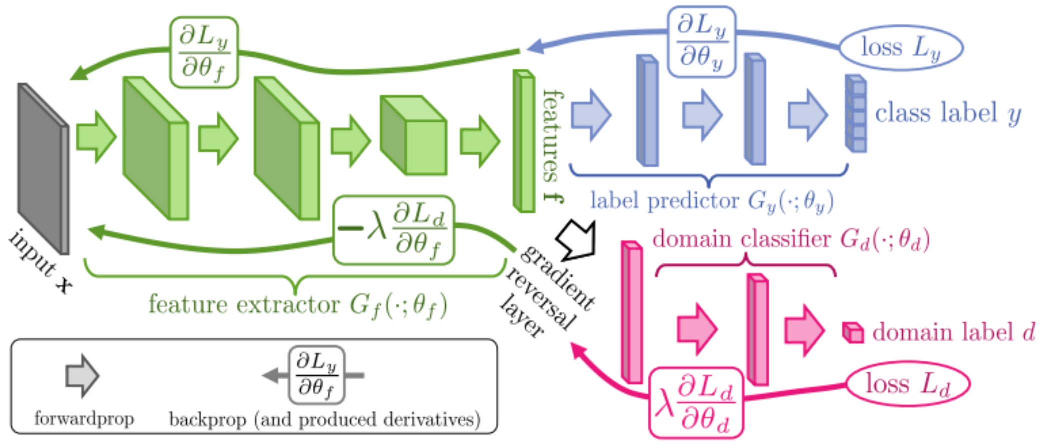

Fig. 2. DANN Overview: An overview of standard DANN architecture.

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 G to minimize L_{da}:

$$\begin{aligned} L_{da}(X_s, X_t, M) = \\ & -(\mathbb{E}_{(x_s)\sim(X_s)} \log D(M(X_s)) \\ & + \mathbb{E}_{(x_t)\sim(X_t)} \log(1 - D(M(X_t))) \times c_{da} \\ & - \mathbb{E}_{(x_s,y_s)\sim(X_s,Y_s)} \sum_{k=1}^{K} \mathbb{I}_{[k=y_s]} \log C(M(X_s)) \end{aligned} \tag{4}$$

Unlike ADDA, parameters of M and G 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.
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.

III. Experiments And Results

A. ADDA Experiment

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 I. 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^{te} are unlabeled.

Then we present a overview of different configurations of the six experiments and the corresponding results in table II. 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

TABLE I
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.

$l_{r2} = \frac{0.1\times s}{1.0+0.01\times i}$, 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 III-A, III-A, III-A, III-A, III-A and III-A 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, 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 III-A, 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 Ex1 exactly the same as Eric Tzeng st 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 III-A, 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 Ex3, shown in III-A, 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 III-A. 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 III-A and III-A 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 thought the classifier works the best on the source dataset. We think it is because there exists a larger different between the two domains, real world and clipart.

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 $Ac(M_s(X_s^{te}))$.

TABLE II
DIFFERENT CONFIGURATIONS AND RESULTED PERFORMANCES OF ADDA EXPERIMENTS

Sym	Src	PT	AN	BN	$Ac(M_s(X_s^{te}))^*$	$Ac(M_s(X_t^{te}))^*$	$Ac(M_t(X_t^{tr}))^*$	$Ac(M_t(X_t^{te}))^*$
$Ex1$	Art	×	×	×	0.6074	0.5080	0.0222	0.0189
$Ex2$	Art	✓	×	×	0.6818	0.5712	0.5897	0.5821
$Ex3$	Art	✓	✓	×	0.6488	0.6223	0.6551	0.6372
Ex_A	Art	✓	✓	✓	0.7893	0.7313	0.7621	0.7354
Ex_C	Clipart	✓	✓	✓	0.8624	0.6561	0.6646	0.6636
Ex_P	Product	✓	✓	✓	0.9391	0.7313	0.7556	0.7353

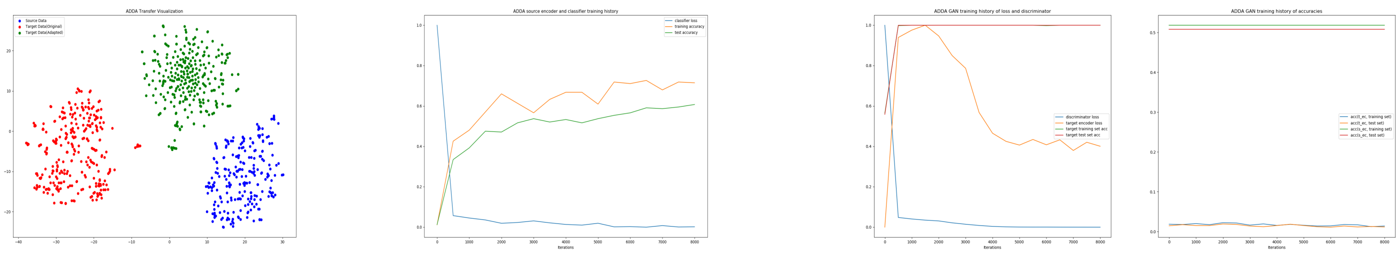

Fig. 3. Visualization of Ex1 Result

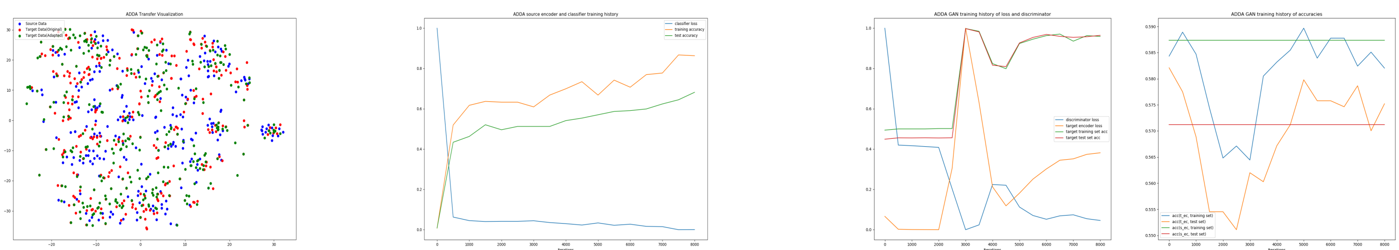

Fig. 4. Visualization of Ex2 Result

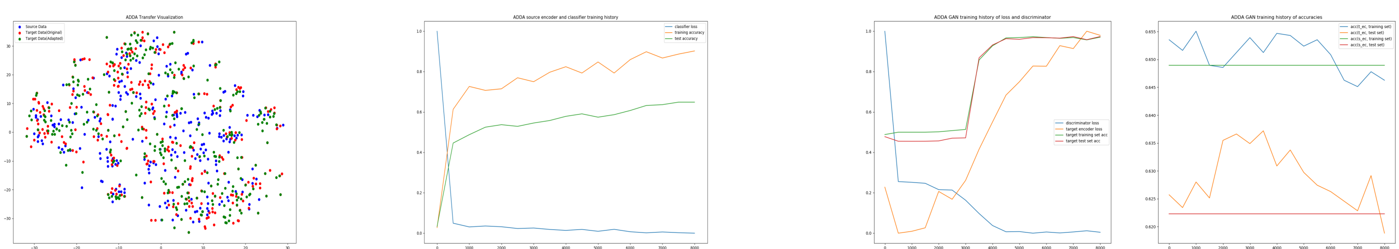

Fig. 5. Visualization of Ex3 Result

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.

V. CONCLUSION

Conclusions.

REFERENCES

Please number citations consecutively within brackets [**?**]. The sentence punctuation follows the bracket [**?**].

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”.

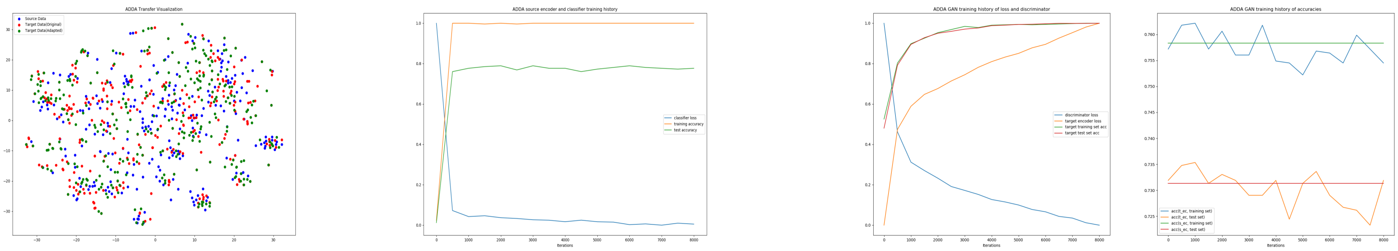

Fig. 6. Visualization of ExA Result

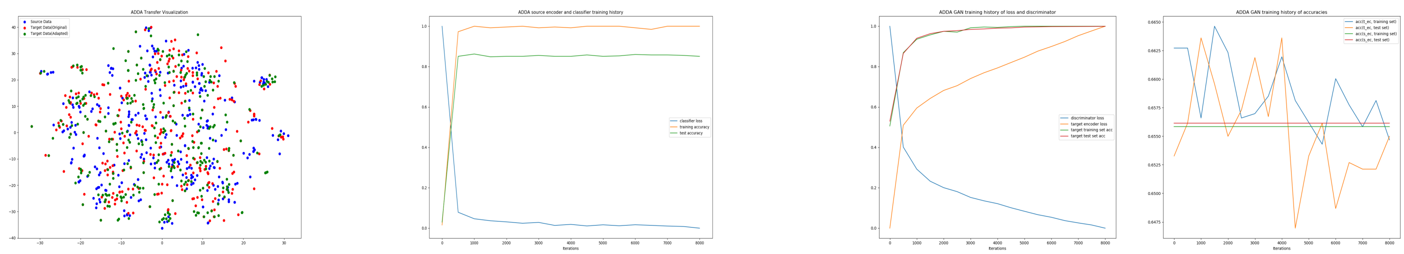

Fig. 7. Visualization of ExC Result

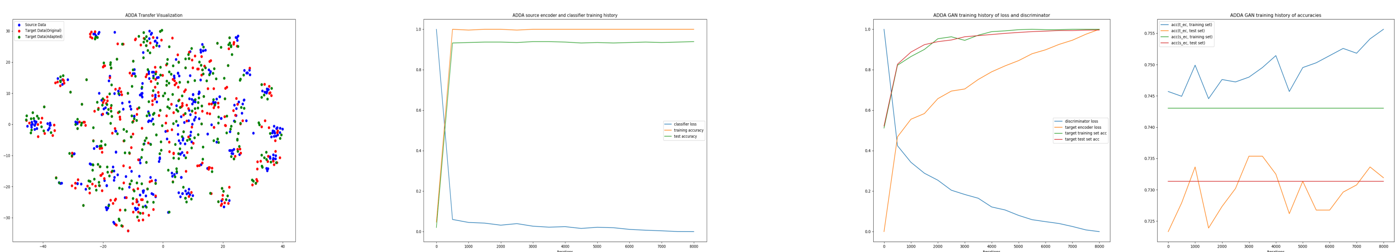

Fig. 8. Visualization of ExP Result

IEEE conference templates contain guidance text for composing and formatting