Maximum Classifier Discrepancy for Unsupervised Domain Adaptation

E4040.2018Fall.BOBA.report
Mingwei Sun ms5497, Jingyuan Bian jb4076, Bowen Li bl2698
Columbia University

Abstract

Maximum Classifier Discrepancy Domain Adaptation is a new adversarial method that maximizes the discrepancy between two classifiers' outputs and minimize the discrepancy while training the generator. Since the paper is very up-to-date and its official code used Pytorch, we have few reference materials and we built the whole adversarial training process of the generator and two classifiers on our own. We researched a lot and refined our model by parameter tuning. Finally, we built a model that could perform better on USPS and MNIST than other domain adaptation methods like Domain Adaptation by Backpropagation[12] and Domain-Adversarial Training of Neural Networks[13].

1. Introduction

We present a method for unsupervised domain adaptation. Two problems of many adversarial learning methods that train a feature generator network to mimic the discriminator are: (1) the domain classifier does not consider task-specific decision boundaries between classes, leading to generation ambiguous features near class boundaries; (2) it is difficult to match each domain's characteristics. To solve these problems, a new approach attempts to align distributions of source and target by utilizing the task-specific decision boundaries. We propose to maximize the discrepancy between two classifiers' outputs to detect target samples that are far from the support of the source. A feature generator learns to generate target features near the support to minimize the discrepancy.

The paper was published recently and only provided their oficial code in Pytorch, so we needed to searched a lot about adversarial process and domain adaptation to learn about how to construct the whole model using Tensorflow. Finally, we got better results on two domain adaptation processes, including from USPS to MNIST and from SVHN to MNIST, than other domain adaptation methods. Especially, our model outperforms the original paper on USPS and MNIST dataset via parameter tuning and changing optimizer.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

We train a feature generator network G and two classifiers in the adversarial manner. The adversarial manner is as below:

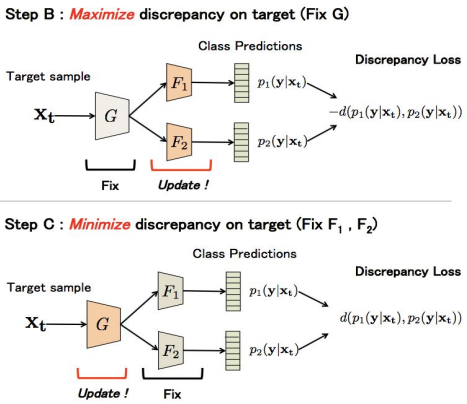

Figure1. Flow charts of adversarial manner

2.2 Key Results of the Original Paper

The original paper assessed four types of adaptation scenarios by using the digits datasets, namely MNIST [9], Street View House Numbers (SVHN) [5], and USPS [7] and further evaluated our method on the traffic sign datasets, Synthetic Traffic Signs (SYN SIGNS) [10] and the German Traffic Signs Recognition Benchmark [35] (GTSRB).

METHOD	SVHN to MNIST	SYNSIG to GTSRB	MNIST to USPS	MNIST* to USPS*	USPS to MNIST
Source Only	67.1	85.1	76.7	79.4	63.4
Distribution Matching based Methods					
MMD † [21]	71.1	91.1	-	81.1	-
DANN † [7]	71.1	88.7	77.1±1.8	85.1	73.0±0.2
DSN † [4]	82.7	93.1	91.3	-	-
ADDA [39]	76.0±1.8	-	89.4±0.2	-	90.1±0.8
CoGAN [19]	-	-	91.2±0.8	-	89.1±0.8
PixelDA [3]	-	-	-	95.9	-
Ours ($n = 2$)	94.2±2.6	93.5±0.4	92.1±0.8	93.1±1.9	90.0±1.4
Ours ($n = 3$)	95.9±0.5	94.0±0.4	93.8±0.8	95.6±0.9	91.8±0.9
Ours ($n = 4$)	**96.2**±0.4	**94.4**±0.3	**94.2**±0.7	**96.5**±0.3	**94.1**±0.3
Other Methods					
ATDA † [32]	86.2	96.2	-	-	-
ASSC [11]	95.7±1.5	82.8±1.3	-	-	-
DRCN [9]	82.0±0.1	-	91.8±0.09	-	73.7±0.04

Table 1. The results of the visual DA experiment on the digits and traffic signs datasets

3. Methodology

In this section, we'll discuss the challenges we faced during this project. In section 3.2, we briefly talk about this MCD(Maximum Classifier Discrepancy)algorithm.

3.1. Objectives and Technical Challenges

We aim to realize the original paper's method on the digits datasets experiment. In addition, we want to try out different hyper-parameters and neural networks. Finally, we would make a comparison between the original paper's results and ours.

Technical challenges:

First, to rewrite the original Pytorch codes with Tensorflow. Due to the difference between these two deep learning frames, we had to construct brand new structures for our training functions from the beginning. For example, PyTorch uses object oriented approach to define basic building blocks and give some 'rails' to move on while providing ability to extend functionality via subclassing. Meanwhile, in Tensorflow we need to write lots boilerplate code and define the biases and weights time by time [4]. Since the original paper is rather recent, we can find few online Tensorflow codes realizing the method in this paper for reference.

Second, to pre-process the image datasets. The MNIST and USPS images are in 28x28 size in a black-and-white format, while SVHN has 32x32 images in RGB format. The original paper does not introduce how they process the datasets and only includes the transposing steps in data processing in its Pytorch codes. So in the SVHN-to-MNIST experiment, we have to convert MNIST images into a 32x32 format, while converting SVHN images into grayscale.

Third, time-consuming process of tuning parameters and adjusting neural networks. These tasks include but are not limited to using different optimizers, changing the number of times to update the feature generator (which is a hyper-parameter peculiar to this method) and tuning other hyper-parameters like batch size, learning rate and number of epoch. It takes long time to try out the many combinations.

3.2. Problem Formulation and Design

3.2.1 Training Formulation and Design

Discrepancy: In this study, we utilize the absolute values of the difference between the two classifiers' probabilistic outputs as discrepancy loss:

$$d(p_1, p_2) = \frac{1}{K}\sum_{k=1}^{K} |p_{1k} - p_{2k}|$$

To sum up the previous discussion in Section 3, we need to train two classifiers, which take inputs from the generator and maximize $d(p_1, p_2)$, and the generator which tries to mimic the classifiers. Here, we need both the classifiers and generator must classify source samples correctly(in a high accuracy).

The main training algorithm can be described as 3 steps: A, B and C.

StepA First, we train both classifiers and generator to classify the source samples correctly. In order to make classifiers and generator obtain task-specific discriminative features, this step is crucial. We train the networks to minimize softmax cross entropy($L(X_s, Y_s)$). The objective is as follows:

$$L(X_s, Y_s) = -E_{(x_s, y_s)\sim(X_s, Y_s)} \sum_{k=1}^{K} 1_{[k=y_s]} log p(y|X_s)$$

StepB In this step, we train the classifiers (F_1, F_2) as a discriminator for a fixed generator (G). By training the classifiers to increase the discrepancy, they can detect the target samples excluded by the support of the source. This step corresponds to **Step B** in Fig. 3. We add a classification loss on the source samples. Without this loss, we experimentally found that our algorithm's performance drops significantly. We use the same number of source and target samples to update the model. In this step, we want to minimize the term $L(X_s, Y_s) - L_{adv}(X_t)$. The objective is as follows:

$$L_{adv}(X_t) = E_{x_t \sim X_t}[d(p_1(y|x_t), p_2(y|x_t))]$$

StepC We train the generator to minimize the discrepancy for fixed classifiers. This step corresponds to Step C in Fig. 3. The number n indicates the number of times we repeat this for the same mini-batch. This number is a hyper- parameter of our method. This term denotes the trade-off between the generator and the classifiers. In this Step, we need minimize $L_{adv}(X_t)$.

These three steps are repeated in our method. The order of the three steps is not important. But in our training process, we implement StepC more than once in each iteration. Instead, our major concern is to train the

classifiers and generator in an adversarial manner under the condition that they can classify source samples correctly.

3.2.2 Flowcharts

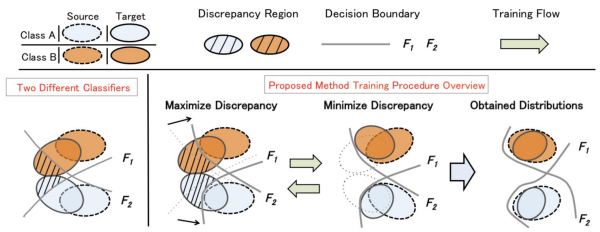

Figure 2. Training Procedure Overview(Example of two classifiers.)

First, we can see that the target samples outside the support of the source can be measured by two different classifiers (Leftmost, *Two different classifiers*). Second, regarding the training procedure, we solve a minimax problem in which we find two classifiers that *maximize* the discrepancy on the target sample, and then generate features that *minimize* this discrepancy.

3.2.4 Flow charts and pseudo code descriptions

The training lower level flow charts we've mentioned above in FIgure 1.

pseudo code:

```
n_iter = length(x_targt)/batchsize
for epc in range(epoch):
   for i in range(n_iter):
        img_t = data['T']
        img_s = data['S']
        label_s = data['S_label']
# Step A
        # Generate source data features
        feat_s = G(img_s)
        # Assign source data use two classifiers
        output_s1 = C1(feat_s)
        output_s2 = C2(feat_s)
        # Compute loss for Step A
        loss_s1 = criterion(output_s1, label_s)
        loss_s2 = criterion(output_s2, label_s)
        loss_s = loss_s1 + loss_s2
        loss_s.backward()
# Step B(Fix Generator)
        # Generate source data features
        feat_s =.G(img_s)
        # Assign target data use two classifiers
        output_s1 = C1(feat_s)
        output_s2 = C2(feat_s)
        # Generate source data features
        feat_t = G(img_t)
        # Assign target data use two classifiers
        output_t1 = C1(feat_t)
        output_t2 = C2(feat_t)
        # Compute loss for Step B
        loss_s1 = criterion(output_s1, label_s)
        loss_s2 = criterion(output_s2, label_s)
        loss_s = loss_s1 + loss_s2
        loss_dis = discrepancy(output_t1, output_t2)
        loss = loss_s - loss_dis
        # Update Classifier
        loss.backward()
# Step C(Fix Classifier), repeat Step C in num_k times
        for i in range(num_k):
            # Generate source data features
            feat_t = G(img_t)
            # Assign target data use two classifiers
            output_t1 =C1(feat_t)
            output_t2 = C2(feat_t)
            loss_dis = discrepancy(output_t1, output_t2)
            # Update Generator
            loss_dis.backward()
```

4. Implementation

In this section, we present the detail of our method. First, we give the overall idea of our Neural Networks in 4.1. Second, we explain about the training process details we used in experiments in Section 4.2. Finally, we talk about our flowcharts and used data in Section 4.3 and 4.4.

4.1 Architecture of Network

For our unsupervised DA(UDA), we have two feature Generator(use the same CNN as original paper) and a task-specific classifier(use the same MLP as original paper).

For the Generator(CNN) and Task-Specific Classifiers we mentioned above, we use the architecture used in [12] and [3] below.

SVHN → MNIST	USPS → MNIST
conv5-64	conv5-32
BatchNorm	BatchNorm
Relu	Relu
maxpool	maxpool
conv5-64	conv5-48
BatchNorm	BatchNorm
Relu	Relu
maxpool	maxpool
conv5-128	Flatten
BatchNorm	
Relu	
FC	
BatchNorm	
Relu	
Dropout	

SVHN → MNIST	USPS → MNIST
FC	Dropout
BatchNorm	FC
Relu	BatchNorm
FC	Relu
	Dropout
	FC
	BatchNorm
	Relu
	Dropout
	FC
	BatchNorm

Table 2. Generator and Classifier Architecture

4.2 Training algorithm details

As we mentioned above, we implemented our training steps follow Step A, B and C in Section 3. In each iteration, we first implement Step A and Step B, and then followed with Step C for n times. n is a hyper-parameter peculiar to our method, which means in each iteration, we update Generator more times. we varied the value of n from 2 to 4 in our experiment and observed the results.

Some other parameters like learning rate, kernel size, filter numbers, batch size, dropout keep-prob etc., we refine them with reference of original paper and have some changes.

4.3 Flowcharts

This part we've mentioned in Section 3.

4.4 Data used

We assessed two types of adaptation scenarios by using the digits datasets, namely MNIST [5], Street View House Numbers (SVHN) [9], and USPS [7].

SVHN → MNIST
In this experiment, we evaluate the adaptation of the model on two scenarios. SVHN [9] and MNIST [5] have distinct properties because SVHN datasets contain images with a colored background, multiple digits, and extremely blurred digits, meaning that the domain divergence is very large between these datasets.

USPS → MNIST
We also evaluate our method on MNIST and USPS datasets [6] to compare our method with other methods. We followed the different protocols provided by the paper, ADDA [8]..

In this setting, we followed the different protocols provided by the paper, ADDA [8]. The protocol provides the setting where a part of training samples are utilized during training. 2,000 training samples are picked up for MNIST and 1,800 samples are used for USPS. We added Batch Normalization layer to the architecture.

5. Results

5.1. Project Results

We received our best results for each n in two experiments with different optimizers. Please see the figures below for details. The learning rate is the same as the original paper as 0.0002, but other parameters varies.

	SVHN to MNIST	USPS to MNIST
source only	67.1%	63.4%
ADDA	76.0%	90.1%
Ours(n=2)	88.6%	96.8%
Ours(n=3)	86.7%	95.3%
Ours(n=4)	83.6%	94.6%

Table 3. Results. Reported accuracy is obtained as the better one between the 2 classifiers' accuracy rates.

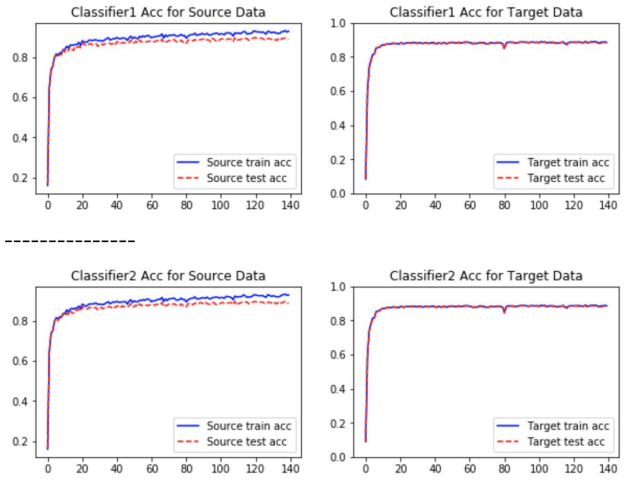

Figure 3. SVHN to MNIST, n=2, Adam optimizer, batch_size = 64, epochs = 20

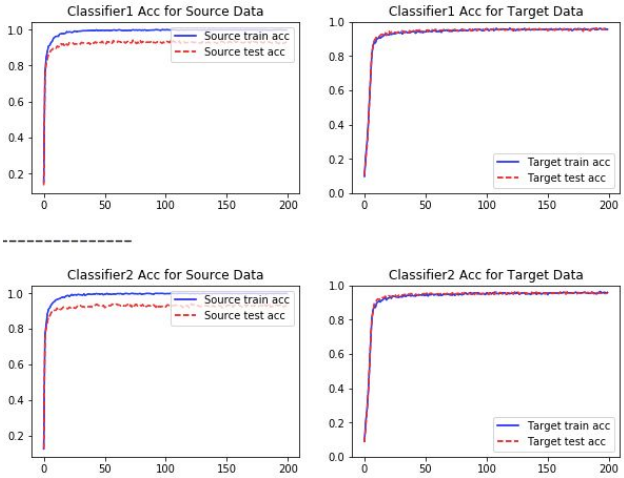

Figure 4. USPS to MNIST, n=2, Adam optimizer, batch_size = 64, epochs = 200

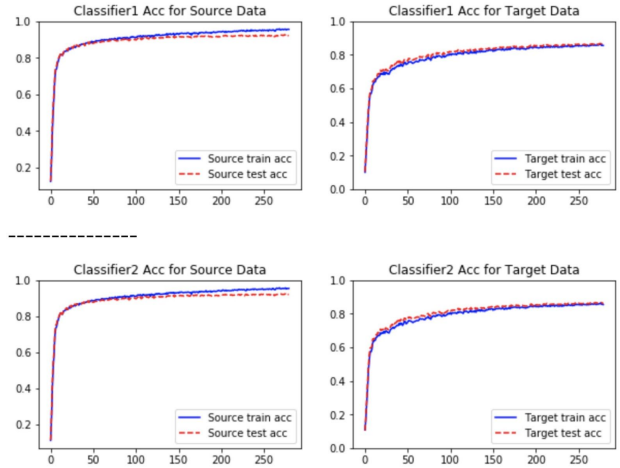

Figure 5. SVHN to MNIST, n=3, SGD with momentum optimizer, batch_size = 64, epochs = 40

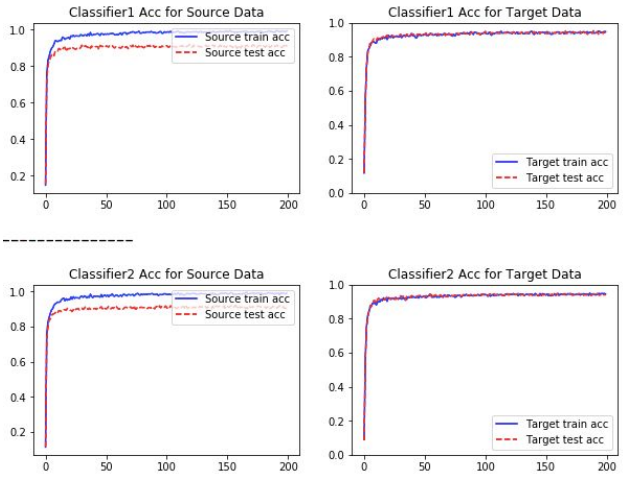

Figure 6. USPS to MNIST, n=3, SGD with momentum optimizer, batch_size = 32, epochs = 200

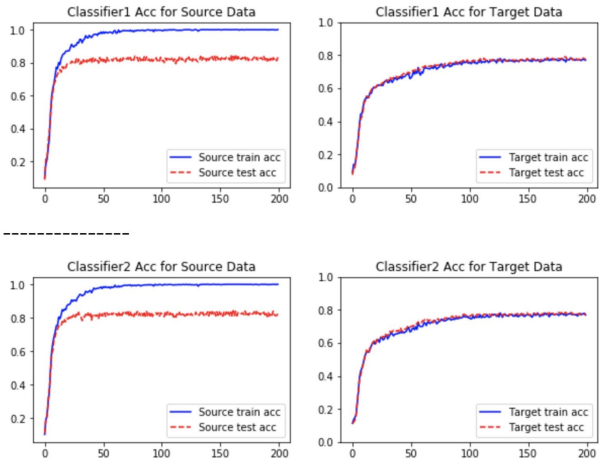

Figure 7. SVHN to MNIST, n=4, SGD with momentum optimizer, batch_size = 32, epochs = 20

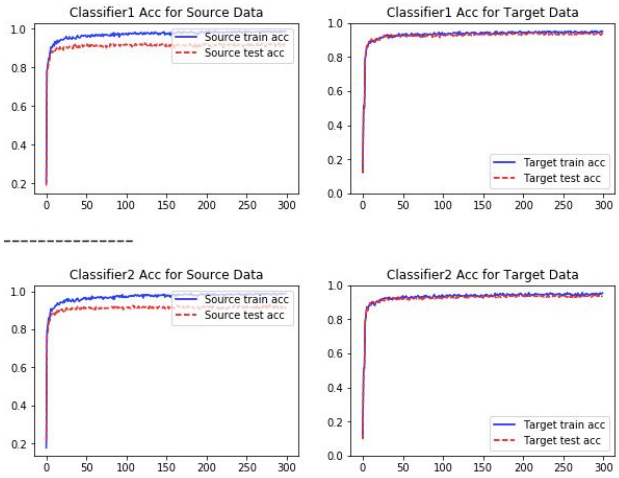

Figure 8. USPS to MNIST, n=4, Adam optimizer, batch_size = 32, epochs = 200

5.2. Comparison of Results

	Original Paper (n=2)	Our Results(n=2)
SVHN to MNIST	94.2%	88.6%
USPS to MNIST	90.0%	96.8%
	Original Paper (n=3)	Our Results(n=3)
SVHN to MNIST	95.9%	86.7%

USPS to MNIST	91.8%	95.3%
	Original Paper (n=4)	Our Results(n=4)
SVHN to MNIST	96.2%	83.6%
USPS to MNIST	94.1%	94.6%

Table 4. Results comparison. Reported accuracy is obtained as the better one between the 2 classifiers' accuracy rates.

5.3. Discussion of Insights Gained

Our models differ from the original models in dataset size, numbers of epoch and batch size. From our results (not all the tried models have their results showed in this report), we may conclude to some insights.

The number of epoch has a diminishing but long-lasting positive impact on accuracy in our results. In the SVHN to MNIST experiment, we only run no more than 40 epoches while the original paper runs 20000 iterations, which we think as the main reason why our result is inferior to the the original paper's result. We believe the results would be even improved if a larger number of epoch is used, as the learning curves indicate.
We use a batch size of 64, which wins over the batch size of 128 used in the original paper in most cases where n = 3 or 4.

We use 60000 training samples and 20000 testing samples for MNIST, 20000 training samples and 5000 testing samples for SVHN instead of the entire standard training and testing sets. However, this did not lead to significant difference in accuracy.

6. Conclusion

The original paper proposed a new approach for UDA, which utilizes task-specific classifiers to align distributions. Utilizing task-specific classifiers as discriminators that try to detect target samples that are far from the support of the source. A feature generator learns to generate target features near the support to fool the classifiers. Since the generator uses feedback from task-specific classifiers, it will avoid generating target features near class boundaries.

We performed our method on two digits dataset and evaluated our method on image classification. In both two experiments, our method outperformed state-of-the-art methods. For the USPS to MNIST dataset, our method even outperformed the provided the results of the original paper after fine tuning.

6. References

Include all references - papers, code, links, books.
[1]https://ElinaBian@bitbucket.org/ElinaBian/uda_with_maximum_classifier_discrepancy.git
[2] H. Li, "Author Guidelines for CMPE 146/242 Project Report", *Lecture Notes of CMPE 146/242*, Computer Engineering Department, College of Engineering, San Jose State University, March 6, 2006, pp. 1.
[3]K.Bousmalis, N.Silberman, D.Dohan, D.Erhan, and D.Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *CVPR*, 2017. 6, 10
[4] PyTorch vs TensorFlow - spotting the difference – Towards Data Science. Retrieved from https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b
[5] Y.Netzer, T.Wang, A.Coates, A.Bissacco, B.Wu, and A.Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, 2011.6
[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.6
[7] J. J. Hull. A database for handwritten text recognition research. *PAMI*, 16(5):550–554, 1994.6
[8] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell. Adversarial discriminative domain adaptation. In *CVPR*, 2017. 6, 10
[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition.
[10] B. Moiseev, A. Konev, A. Chigorin, and A. Konushin. Evaluation of traffic sign recognition methods trained on synthetically generated data. In ACIVS, 2013.6
[11] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In IJCNN, 2011.6
[12] Y.Ganin and V.Lempitsky .Unsupervised domain adaptation by backpropagation. In *ICML*, 2014. 6, 7, 8, 10, 11, 12
[13] Y.Ganin,E.Ustinova,H.Ajakan,P.Germain,H.Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. JMLR, 17(59):1–35, 2016. 1, 2, 5

8. Appendix

8.1 Individual student contributions in fractions - table

	CUID1	CUID2	CUID3
Last Name	Bowen Li	Mingwei Sun	Jingyuan Bian
Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1	data collecting, data processing	data collecting, data processing	data collecting, data processing
What I did 2	Coding: Generator, Classifier Networks	Coding:training process	Coding:training process
What I did 3	Report	Report	Report