Machine Learning Fall 2019 Final Project
Final Submission Report

Zhuchkov Alexey BS17 -DS-01

Abstract

Domain adaptation is a field associated with transfer learning which is used when we aim at learning from a source data distribution a well performing model on a different (but related) target data distribution. To resolve this problem, a lot of methods have been proposed, including techniques such as generative adversarial networks (GAN), however the complexity of such methods mitigates their usability in a number of tasks, as it is quite time-consuming to fine-tune such networks. This report analyzes and summarizes the implementation of Triplet Loss Network for Unsupervised Domain Adaptation [1]. This assignment concentrated on solving Domain Adaptation task and the report gives a description of a final model that is based on implementation of TripLet Loss.

1.Introduction

In the real world, it is quite a common situation when the domain of the input data changed while the task domain (the labels) remained the same. The input and task distributions could differ at the same time. In these cases, domain adaptation comes to the rescue. Domain adaptation is a sub-discipline of machine learning which deals with scenarios in which a model trained on a source distribution is used in the context of a different (but related) target distribution. In general, domain adaptation uses labeled data in one or more source domains to solve new tasks in a target domain. The level of relatedness between the source and target domains hereby usually determines how successful the adaptation will be.

The datasets for training and testing were SVHN and MNIST accordingly. The model explanation, train and test accuracies and experiments with different techniques will be given in the following sections.

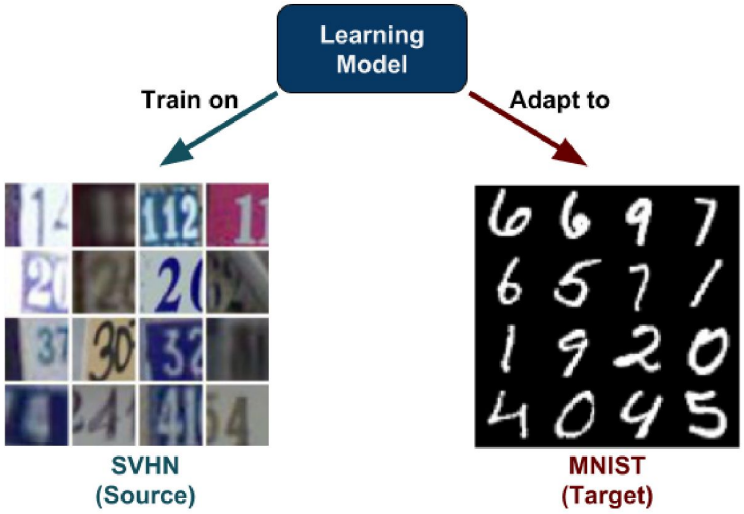


Figure 1. Both datasets (SVHN and MNIST) are images of digits; however, there are many differences between them: in style, brightness, and contrast. SVHN images are real-world RGB images of digits (in different orientations) taken from street house numbers, whereas MNIST is a handwritten-digit dataset, where all images are in greyscale and have digits in a fixed orientation.

2. Related works

DupGAN [3] has good result on UDA from SVHN to MNIST (92.46%). It uses encoder and classifier for determining the label of image, decoder for forcing encoder to learn important features to be able to reproduce input image. Discriminator forces encoder to make latent space domain invariant. However, this architecture tends to have slow convergence, consumes a lot of resources.

3.Architecture

3.1.Overview

The following section describes the proposed model for Unsupervised Domain Adaptation. Model contains three main parts: an encoder, a classifier, and a discriminator, as shown in Figure 2. The final classification model consists of the encoder and the classifier. The discriminator is used to train the encoder to generate domain-invariant features

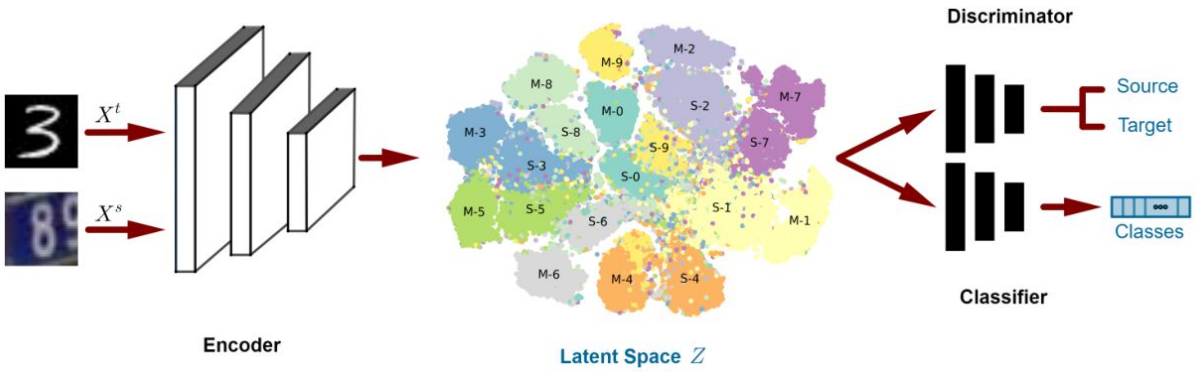


Figure 2. Architecture of model that can be divided into three parts: an encoder, a discriminator, and a classifier. The encoder translates the images (X space) to embeddings in the latent space (Z space). In the latent space, each group of embeddings is either from MNIST (M) or SVHN (S) with corresponding labels. The latent representation is passed to both the discriminator and the classifier. The discriminator distinguishes if the latent representation is from source or target domain, whereas the classifier determines a label for it.

1. **Encoder:** The encoder E is a Convolutional Neural Network-like network with weights W^E that maps the input image to latent representation for both source and target domain. Vectors from latent space are the extracted features are domain-invariant and category-informative. The output of the encoder is passed to both the discriminator and the classifier.

2. **Discriminator:** As was mentioned in Related Works part, discrimination between the source and the target in the latent space is a frequently used in many of the recent paper in Domain Adaptation. Discriminator D is a Deep Neural Network with weights W^D. The discriminator classifies the latent representation of the images to one of the domains: MNIST or SVHN

3. **Classifier:** The classifier C is a feed-forward neural network with weights W^C for multi-class classification with a softmax output activation function. It takes the latent representation and produces the probabilities for each class (0 - 9)

The classifier C and the encoder E are pre-trained on the source data first. Then pseudo-label technique is applied to the target domain, using the output of the classifier on the target images.

There are three important losses used in training pipeline.

1. **Classification Loss:** This is the usual cross-entropy loss for the output of source images and their labels and the output of target images and their corresponding pseudo-labels.

2. **Discrimination Loss:** The discrimination loss is used to train the discriminator to distinguish between the features for both domains, using binary cross-entropy in order to get domain-independent features.

3. **Separability Loss:** The loss function to maximize the between-class variability and minimize the within-class variability. For more details refer to the original paper [1]

3.2.Difference from base model

First let's start from similarities. Base model implements CNN with feed-forward neural network (2 FC) with softmax output to determine class of input image. Final model uses CNN-like encoder and classifier that is feed-forward neural network (4 FC) with softmax output to determine class of input image. However, in base model there is no special modules to force CNN to learn important features in domain invariant form. Discriminator in final model used to ensure that the features extracted from the encoder network could be used to classify images of both domains (i.e. features are domain-invariant). Also, final model implements pseudo-labeling that is used to decrease the gap between the target and source domains, by providing pseudo-labels for the unlabeled samples from the target domain. This gives an opportunity to train model on target domain similarly to source.

3.3.Source of inspiration

I've started using DupGAN [3] that train generator and classifier with decoder and discriminator that forces the generator to learn important features from image. However, it converges very slow and takes too much resources to train. Then I discovered TripNet paper [1] that has faster convergence than DupGAN and have taken code from its GitHub as my final model [2]. I have made some changes to source code: changed hyperparameters (see Hyperparametrs part), removed augmentations, replaces Adam optimizer to Adamax in pretrain part.

3.4.Domain adaptation

As it was mentioned above, in order to make the model to generalize to new domain, the following techniques are used:

- Discriminator - ensures that the features extracted from the encoder network are domain-invariant
- Pseudo-labeling - decreases the training gap between the target and source domains
- Separability Loss - maximizes the between-class variability and minimizes the within-class variability.

3.5.Hyperparameters

Model	β_{sep}	β_C	β_P	λ_S	λ_T	PL_{Th}	lr	lr_D	train iter	pre train iter	batch size
Source code	1.5	1	4	0.5	0.8	0.999	0.0003	0.0003	1000	20000	64
Final model	1.5	1	4	0.5	0.8	0.999	0.001	0.001	1000	5000	1024

*red color - parameters of source and final model are different

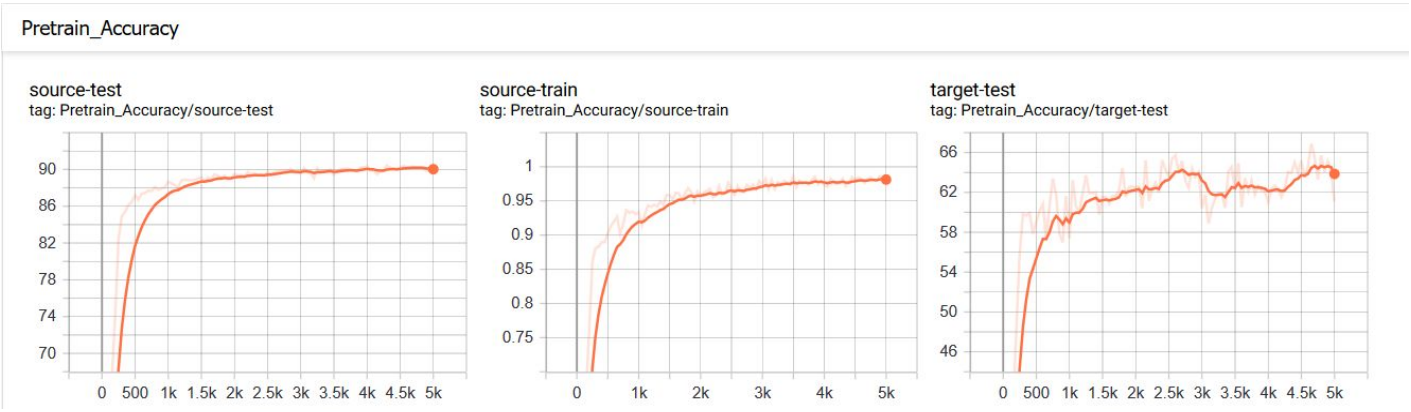
Mainly hyperparameters have been changed in trial-and-error way. Pretrain number of iterations have been decreased. There is a tendency from accuracy plot that model becomes confident on pretrain stage in much less number of steps. Increasing a batch size gives growth of pretrain accuracy by 5-10%. Increasing of learning rate speeds up the train time.

4. Experiments and Results

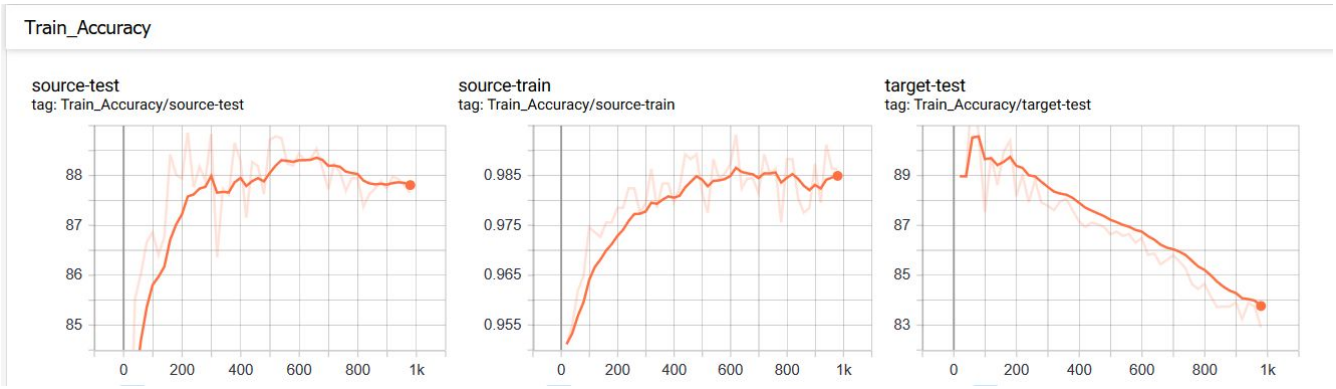
I used tensorboard to visualize data [4, 5, 6]. Here are the results.

4.1.Accuracy

1. *on pretrain stage (max 66.9%)*



2. *on train stage (max 92.76%)*



Test Accuracy of 0: 99% (973/980)
Test Accuracy of 1: 99% (1126/1135)
Test Accuracy of 2: 98% (1012/1032)
Test Accuracy of 3: 95% (967/1010)
Test Accuracy of 4: 99% (974/982)
Test Accuracy of 5: 97% (868/892)
Test Accuracy of 6: 3% (37/958)
Test Accuracy of 7: 94% (974/1028)
Test Accuracy of 8: 73% (719/974)
Test Accuracy of 9: 60% (608/1009)

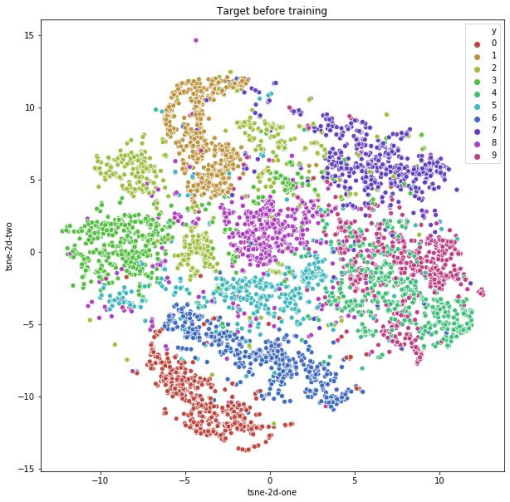
Test Target Accuracy (Overall) [8258 / 10000]: 82%

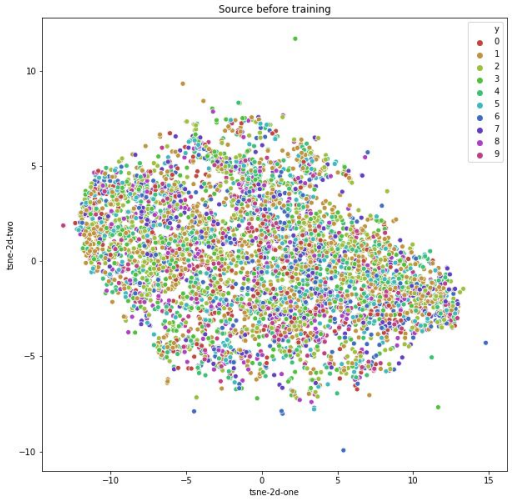
Test Source Accuracy (Overall) [22773 / 26032]: 87%

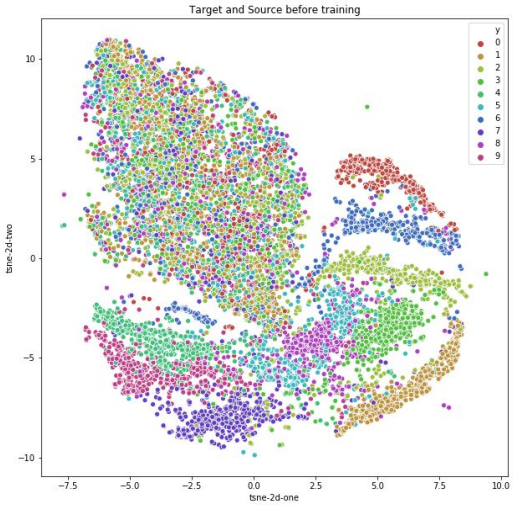
4.3.Latent space visualization

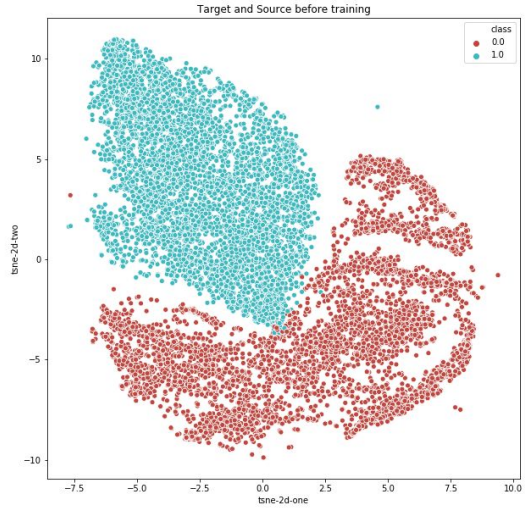
1. *Before training*

   Target before training, Source before training, Target and source before training





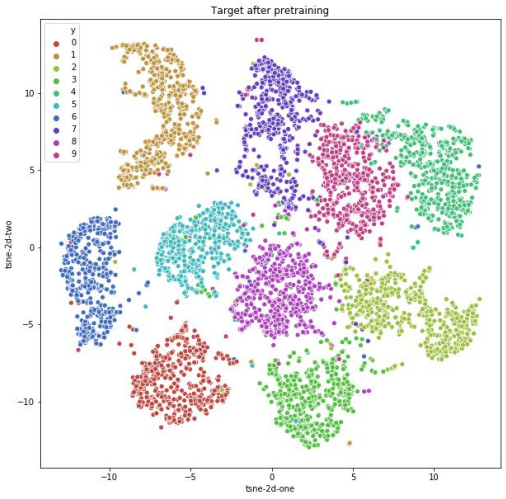


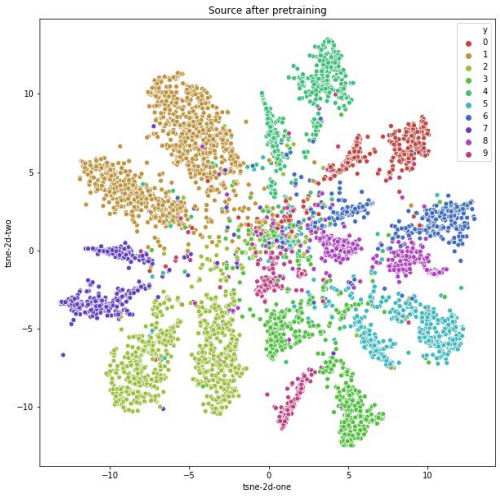


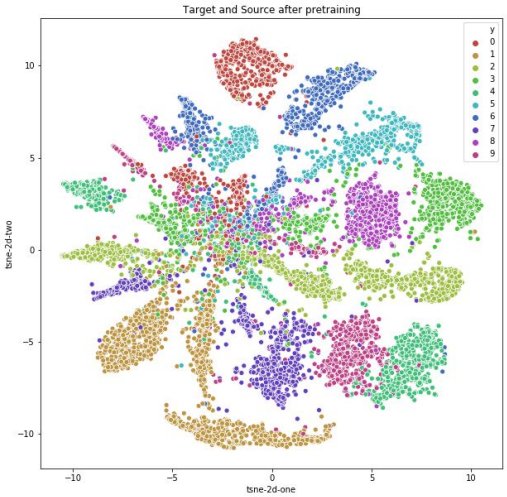
Before training, model cannot retrieve domain invariant features for both datasets

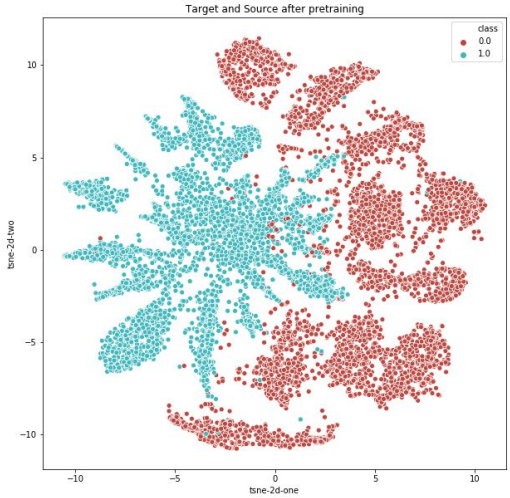
2. *After pretraining*

Target after pretraining, Source after pretraining, Target and source after pretraining



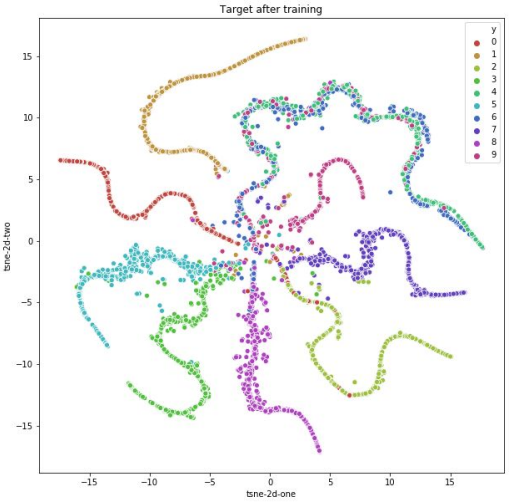


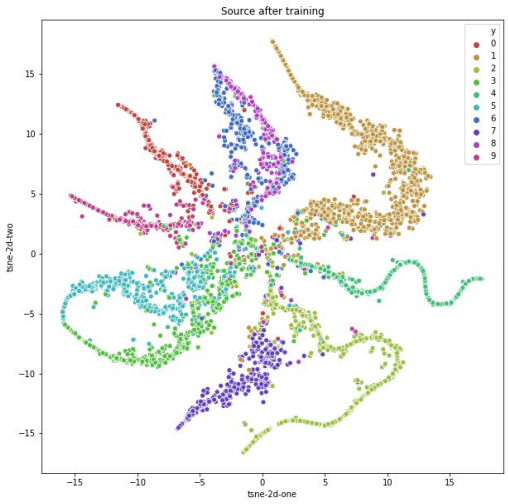


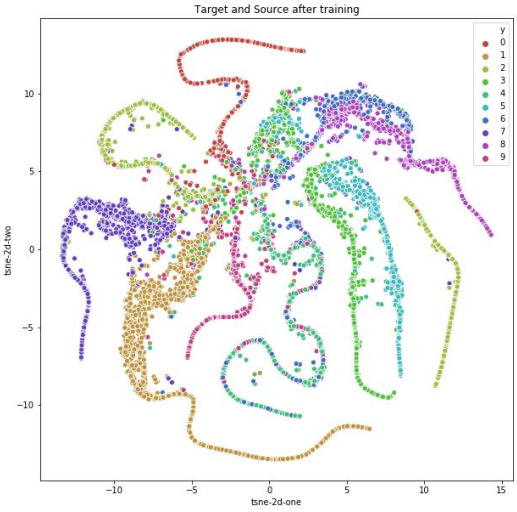


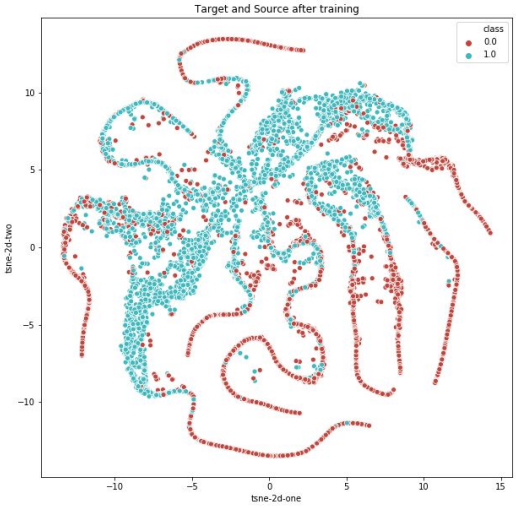
3. *After training*

   Target after pretraining, Source after pretraining, Target and source after pretraining









As we can see after training stage both datasets become similar shape and centered to one point.

5.Conclusion

The described architecture gives good results to perform domain adaptation: when new data previously not seen by the model is given as input, the accuracy of a model can decline slightly but remains quite high. It happens because the model has not adapted for a specific dataset but successfully generalized to both of them. Also, I've observed that quality of training (accuracy) highly depend on quality of pretraining. That's happened because of pseudo-labeling technique.

6.Link to code
https://colab.research.google.com/drive/1w-1INTV7J8SAxpWf5jdZG_t88Km9LR4m

References

[1] Triplet Loss Network for Unsupervised Domain Adaptation
[2] Pytorch Implementation of TripLet Loss for Unsupervised Domain Adaptation
[3] Duplex Generative Adversarial Network for Unsupervised Domain Adaptation
[4] Visualising high-dimensional datasets using PCA and t-SNE in Python
[5] Using TensorBoard in Notebooks
[6] Documentation torch.utils.tensorboard
[7] Deep Domain Adaptation In Computer Vision
[8] A Survey of Unsupervised Deep Domain Adaptation