An Implementation of "Adversarial Discriminative Domain Adaptation"

Ali Jafari, Behrouz Ghamkhar

March 30, 2023

Contents

1 Introduction

While methods based on deep learning have achieved state-of-the-art results on numerous tasks, most of these methods are based on the assumption that the data used to train the model is drawn from the same distribution that the data used to test the model is drawn from. This assumption can be violated in the real world, where AI models should be able to adapt themselves to new environments with little human supervision. In order to address this problem, methods based on *single-source unsupervised domain adaptation* utilize labeled data from a *source domain* to achieve satisfactory performance on unlabeled data from a different, but related *target domain* [1].

In this document, we discuss our implementation of "Adversarial Discriminative Domain Adaptation" [2], which is based on PyTorch. In addition to documenting our implementation of the paper, we provide the results of our replications of the performed experiments. Furthermore, we comment on the result of each replicated experiment, and offer possible reasons for the success or failure of every experiment.

1.1 Overview of the Method

In [2], the authors propose a framework for single-source unsupervised domain adaptation on a classification task that is trained in three stages. The assumptions of the method are those of single-source unsupervised domain adaptation presented earlier in the introduction, as well as the assumption that the source domain and the target domain share the same set of classes.

Firstly, a *source encoder* and a *classifier* are jointly trained on labeled data from the source domain. Secondly, a *target encoder* is trained adversarially against a *discriminator*. This is performed in a way that given unlabeled data from the target domain, the target encoder can generate representations that are similar to the representations generated by the source encoder, given data from the source domain. In this sense, the second stage of training in the framework proposed by [2] can be understood like the training of a Generative Adversarial Network [3]. In this analogy, the target encoder is the generator, and the 'real data' that the target encoder is trying to forge consists of representations generated by the source encoder from data belonging to the source domain. In the beginning of the second stage of training, the weights of the target encoder are initialized from the weights of the source encoder trained on the source domain in the first stage. During the second stage, the weights of the source encoder are fixed, and only the

target encoder and the discriminator are trained. Finally, the weights of the target encoder trained in the second stage and the classifier trained in the first stage are fixed, and given unlabeled data from the target domain, the target encoder and the classifier attempt in conjunction to predict class labels for the unlabeled target data [2]. The intuitive justification behind this framework is that if the source encoder and the classifier are trained well on the source domain in the first stage, then given representations generated by the source encoder from data belonging to the source domain, the classifier can accurately predict class labels for data from the source domain. Consequently, if the target encoder can receive data from the target domain and generate similar representations to those that the source encoder can create from source data, then we can give the representations generated by the target encoder to the classifier and expect a reasonably accurate classification performance on the unlabeled target data. Figure 1 illustrates the three stages of the framework.

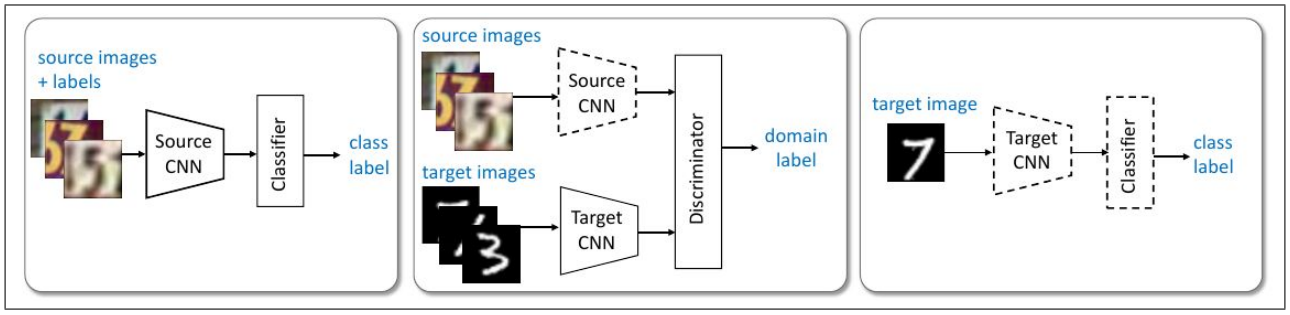

Figure 1: The framework proposed in [2], in which dashed lines around a network indicate that the weights of that network are fixed. This figure belongs to [2].

1.2 Outline of the Document

The rest of this document is organized as follows: Firstly, we give an explanation of how our implementation is structured and what the role of each module is. We attempt to follow a 'top-down' approach in explaining the implementation, by discussing the more important parts of the code first and then focusing our attention on the other modules. Next, we report the results on our replications of the experiments performed in [2]. As mentioned earlier, we aim to offer insights on possible reasons behind the success or failure of each replicated experiment.

2 The Implementation

On the highest level, our implementation consists of seven 'modules':

1. `Training`: This module contains three classes, each of which implements a single stage of the framework proposed in [2]. The three classes are called `SourceTrainer`, `Adapter`, and `NetworkTester`. The `SourceTrainer` class implements the first stage of the learning framework, while the `Adapter` and `NetworkTester` classes implement the second and third stages of the learning framework, respectively.

2. `Experiments`: This module contains a single class, called `Experiment`, which uses the three classes provided in the `Training` module to run the experiments described in [2].

3. `Datasets`: This module contains the different datasets that

References

[1] G. Wilson and D. J. Cook, "A survey of unsupervised deep domain adaptation," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 11, no. 5, pp. 1–46, 2020.

[2] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7167–7176, 2017.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.