Chapter 10
Domain-Adversarial Training of Neural Networks

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand and Victor Lempitsky

Abstract We introduce a representation learning approach for domain adaptation, in which data at training and test time come from similar but different distributions. Our approach is directly inspired by the theory on domain adaptation suggesting that, for effective domain transfer to be achieved, predictions must be made based on features that cannot discriminate between the training (source) and test (target) domains. The approach implements this idea in the context of neural network architectures that are trained on labeled data from the source domain and unlabeled data from the target domain (no labeled target-domain data is necessary). As the training progresses, the approach promotes the emergence of features that are (i) discriminative for the main learning task on the source domain and (ii) indiscriminate with respect to the shift

Y. Ganin (✉) · E. Ustinova · V. Lempitsky
Skolkovo Institute of Science and Technology, Skolkovo, Moscow Region, Russia
e-mail: ganin@skoltech.ru; yaroslav.ganin@gmail.com

E. Ustinova
e-mail: evgeniya.ustinova@skoltech.ru

V. Lempitsky
e-mail: lempitsky@skoltech.ru

H. Ajakan · F. Laviolette · M. Marchand
Département d'informatique et de génie logiciel, Université Laval, Québec, QC, Canada
e-mail: hana.ajakan.1@ulaval.ca

F. Laviolette
e-mail: francois.laviolette@ift.ulaval.ca

M. Marchand
e-mail: mario.marchand@ift.ulaval.ca

P. Germain
INRIA Paris - École Normale Supérieure, Paris, France
e-mail: pascal.germain@inria.fr

H. Larochelle
Twitter, Cambridge, MA, USA
e-mail: hugo.larochelle@usherbrooke.ca

H. Larochelle
Université de Sherbrooke, Québec, QC, Canada

G. Csurka (ed.), *Domain Adaptation in Computer Vision Applications*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-3-319-58347-1_10

between the domains. We show that this adaptation behavior can be achieved in almost any feed-forward model by augmenting it with few standard layers and a new *Gradient Reversal* Layer. The resulting augmented architecture can be trained using standard backpropagation, and can thus be implemented with little effort using any of the deep learning packages. We demonstrate the success of our approach for image classification, where state-of-the-art domain adaptation performance on standard benchmarks is achieved. We also validate the approach for descriptor learning task in the context of person re-identification application.

10.1 Introduction

The cost of generating labeled data for a new prediction task is often an obstacle for applying machine learning methods. In particular, this is a limiting factor for the further progress of deep neural network architectures. Another common problem is that, even when the training sets are big enough for training large-scale deep models, they may that suffer from the *shift* in data distribution from the actual data encountered at "test time." One important example is training an image classifier on synthetic or semi-synthetic images, which may come in abundance and be fully labeled, but which inevitably have a distribution that is different from real images. Learning a predictor in the presence of a *shift* between training and test distributions is known as *domain adaptation* (DA). The appeal of the DA approaches is the ability to learn a mapping between domains in the situation when the target domain data are either fully unlabeled (*unsupervised domain annotation*) or have few labeled samples (*semi-supervised DA*). Below, we focus on the harder unsupervised case, although the proposed approach (*domain-adversarial learning*) can be generalized to the semi-supervised case rather straightforwardly.

Unlike many DA methods that work with fixed feature representations, we focus on combining domain adaptation and deep feature learning within one training process. Our goal is to embed domain adaptation into the process of learning representation, so that the final classification decisions are made based on features that are both discriminative and invariant to the change of domains, i.e., have the same or very similar distributions in the source and the target domains. In this way, the obtained feed-forward network can be applicable to the target domain without being hindered by the shift between the two domains. Our approach is motivated by the theory on domain adaptation [30, 31], that suggests that a good representation for cross-domain transfer is one for which an algorithm cannot learn to identify the domain of origin of the input observation.

We thus focus on learning features that combine (i) discriminativeness and (ii) domain-invariance. This is achieved by jointly optimizing the underlying features as well as two discriminative classifiers operating on these features: (i) the *label predictor* that predicts class labels and is used both during training and at test time and (ii) the *domain classifier* that discriminates between the source and the target domains during training. While the parameters of the classifiers are optimized in

order to minimize their error on the training set, the parameters of the underlying deep feature mapping are optimized in order to *minimize* the loss of the label classifier and to *maximize* the loss of the domain classifier. The latter update thus works *adversarially* to the domain classifier, and it encourages domain-invariant features to emerge in the course of the optimization.

Crucially, we show that all three training processes can be embedded into an appropriately composed deep feed-forward network, called *domain-adversarial neural network* (DANN) (illustrated by Fig. 10.1) that uses standard layers and loss functions, and can be trained using standard backpropagation algorithms based on stochastic gradient descent or its modifications (e.g., SGD with momentum). The approach is generic as a DANN version can be created for almost any existing feed-forward architecture that is trainable by backpropagation. In practice, the only nonstandard component of the proposed architecture is a rather trivial *gradient reversal* layer that leaves the input unchanged during forward propagation and reverses the gradient by multiplying it by a negative scalar during the backpropagation.

We provide an experimental evaluation of the proposed domain-adversarial learning idea over a range of deep architectures and applications. We first consider the simplest DANN architecture where the three parts (label predictor, domain classifier and feature extractor) are linear, and study its behavior on a toy dataset. We further evaluate the approach extensively for an image classification task, and present results on traditional deep learning image data sets. Finally, we evaluate domain-adversarial *descriptor* learning in the context of person re-identification application [201], where the task is to obtain good pedestrian image descriptors that are suitable for retrieval and verification.

Related Work This book chapter is strongly based on a recent journal paper [183]. We focus here on the theoretical foundation of our method and the applications in computer vision. Please refer to [183] for an in-depth discussion about previous related literature and application of our method to a text sentiment analysis benchmark. For a DA survey see Chap. 1.

10.2 Domain Adaptation

We consider classification tasks where X is the input space and $Y = \{0, 1, \ldots, L-1\}$ is the set of L possible labels. Moreover, we have two different distributions over $X \times Y$, called the *source domain* $\mathcal{D}_S$ and the *target domain* $\mathcal{D}_T$. An *unsupervised domain adaptation* learning algorithm is then provided with a *labeled source sample* S drawn *i.i.d.* from $\mathcal{D}_S$, and an *unlabeled target sample* T drawn *i.i.d.* from $\mathcal{D}_T^X$, where $\mathcal{D}_T^X$ is the marginal distribution of $\mathcal{D}_T$ over X.

$$S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n} \sim (\mathcal{D}_S)^n\,; \quad T = \{\mathbf{x}_i\}_{i=n+1}^{N} \sim (\mathcal{D}_T^X)^{n'},$$

with $N = n + n'$ being the total number of samples. The goal of the learning algorithm is to build a classifier $\eta : X \to Y$ with a low *target risk*

$$R_{\mathcal{D}_\mathrm{T}}(\eta) = \Pr_{(\mathbf{x},y)\sim\mathcal{D}_\mathrm{T}} \Big(\eta(\mathbf{x}) \neq y\Big),$$

while having no information about the labels of $\mathcal{D}_\mathrm{T}$.

Domain Divergence To tackle the challenging domain adaptation task, many approaches bound the target error by the sum of the source error and a notion of distance between the source and the target distributions. These methods are intuitively justified by a simple assumption: the source risk is expected to be a good indicator of the target risk when both distributions are similar. Several notions of distance have been proposed for domain adaptation [30, 31, 188, 322, 324]. In this paper, we focus on the $\mathcal{H}$-divergence used in [30, 31], and based on the earlier work of Kifer et al. [269]. Note that we assume in Definition 10.1 below that the hypothesis class $\mathcal{H}$ is a (discrete or continuous) set of binary classifiers $\eta : X \to \{0, 1\}$.[1]

Definition 10.1 (c.f. [30, 31, 269]) Given two domain distributions $\mathcal{D}_\mathrm{S}^X$ and $\mathcal{D}_\mathrm{T}^X$ over X, and a hypothesis class $\mathcal{H}$, the *$\mathcal{H}$-divergence* between $\mathcal{D}_\mathrm{S}^X$ and $\mathcal{D}_\mathrm{T}^X$ is

$$d_{\mathcal{H}}(\mathcal{D}_\mathrm{S}^X, \mathcal{D}_\mathrm{T}^X) = 2 \sup_{\eta\in\mathcal{H}} \left| \Pr_{\mathbf{x}\sim\mathcal{D}_\mathrm{S}^X} \big[\eta(\mathbf{x}) = 1\big] - \Pr_{\mathbf{x}\sim\mathcal{D}_\mathrm{T}^X} \big[\eta(\mathbf{x}) = 1\big] \right|.$$

That is, the $\mathcal{H}$-divergence relies on the capacity of the hypothesis class $\mathcal{H}$ to distinguish between examples generated by $\mathcal{D}_\mathrm{S}^X$ from examples generated by $\mathcal{D}_\mathrm{T}^X$. BenDavid et al.in [30, 31] proved that, for a symmetric hypothesis class $\mathcal{H}$, one can compute the *empirical $\mathcal{H}$-divergence* between two samples $S \sim (\mathcal{D}_\mathrm{S}^X)^n$ and $T \sim (\mathcal{D}_\mathrm{T}^X)^{n'}$ by computing

$$\hat{d}_{\mathcal{H}}(S, T) = 2\left(1 - \min_{\eta\in\mathcal{H}} \left[\frac{1}{n}\sum_{i=1}^{n} I[\eta(\mathbf{x}_i)=0] + \frac{1}{n'}\sum_{i=n+1}^{N} I[\eta(\mathbf{x}_i)=1]\right]\right), \quad (10.1)$$

where $I[a]$ is the indicator function which is 1 if predicate a is true, and 0 otherwise.

Proxy Distance (c.f. [31]) suggested that, even if it is generally hard to compute $\hat{d}_{\mathcal{H}}(S, T)$ exactly (e.g., when $\mathcal{H}$ is the space of linear classifiers on X), we can easily approximate it by running a learning algorithm on the problem of discriminating between source and target examples. To do so, we construct a new data set $U = \{(\mathbf{x}_i, 0)\}_{i=1}^n \cup \{(\mathbf{x}_i, 1)\}_{i=n+1}^N$, where the examples of the source sample are labeled 0 and the examples of the target sample are labeled 1. Then, the risk of the classifier trained on the new data set U approximates the "min" part of Eq. (10.1). Given a

[1] As mentioned in [31], the same analysis holds for multi-class setting. However, to obtain the same results when $|Y| > 2$, one should assume that $\mathcal{H}$ is a symmetrical hypothesis class. That is, for all $h \in \mathcal{H}$ and any permutation of labels $c : Y \to Y$, we have $c(h) \in \mathcal{H}$. Note that this is the case for most commonly used neural network architectures.

generalization error ϵ on the problem of discriminating between source and target examples, the $\mathcal{H}$-divergence is then approximated by $\hat{d}_{\mathcal{A}} = 2(1-2\epsilon)$. In [31], the value $\hat{d}_{\mathcal{A}}$ is called the *Proxy $\mathcal{A}$-distance* (PAD). The *$\mathcal{A}$-distance* being defined as $d_{\mathcal{A}}(\mathcal{D}_{\mathrm{S}}^{x}, \mathcal{D}_{\mathrm{T}}^{x}) = 2\sup_{A\in\mathcal{A}} |\Pr_{\mathcal{D}_{\mathrm{S}}^{x}}(A) - \Pr_{\mathcal{D}_{\mathrm{T}}^{x}}(A)|$, where $\mathcal{A}$ is a subset of X. Note that, by choosing $\mathcal{A} = \{A_\eta | \eta \in \mathcal{H}\}$, with A_η the set represented by the characteristic function η, the $\mathcal{A}$-distance and the $\mathcal{H}$-divergence of Definition 10.1 are identical.

Generalization Bound on the Target Risk BenDavid et al. in [30, 31] also showed that the $\mathcal{H}$-divergence $d_{\mathcal{H}}(\mathcal{D}_{\mathrm{S}}^{x}, \mathcal{D}_{\mathrm{T}}^{x})$ is upper bounded by its empirical estimate $\hat{d}_{\mathcal{H}}(S, T)$ plus a constant complexity term that depends on the *VC dimension* of $\mathcal{H}$ and the size of samples S and T. By combining this result with a similar bound on the source risk, the following theorem is obtained.

Theorem 10.1 ([31]) *Let $\mathcal{H}$ be a hypothesis class of VC dimension d. With probability $1-\delta$ over the choice of samples $S \sim (\mathcal{D}_{\mathrm{S}})^n$ and $T \sim (\mathcal{D}_{\mathrm{T}}^{x})^n$, for every $\eta \in \mathcal{H}$:*

$$R_{\mathcal{D}_{\mathrm{T}}}(\eta) \leq \widehat{R}_S(\eta) + \sqrt{\frac{4}{n}\left(d\log\frac{2en}{d} + \log\frac{4}{\delta}\right)} + \hat{d}_{\mathcal{H}}(S,T) + 4\sqrt{\frac{1}{n}\left(d\log\frac{2n}{d} + \log\frac{4}{\delta}\right)} + \beta,$$

with $\beta \geq \inf_{\eta^* \in \mathcal{H}} \left[R_{\mathcal{D}_{\mathrm{S}}}(\eta^*) + R_{\mathcal{D}_{\mathrm{T}}}(\eta^*)\right]$, and $\widehat{R}_S(\eta) = \frac{1}{n}\sum_{(\boldsymbol{x},y)\in S} I[\eta(\boldsymbol{x})\neq y]$ is the empirical source risk.

The previous result tells us that $R_{\mathcal{D}_{\mathrm{T}}}(\eta)$ can be low only when the β term is low, i.e., only when there exists a classifier that can achieve a low risk on both distributions. It also tells us that, to find a classifier with a small $R_{\mathcal{D}_{\mathrm{T}}}(\eta)$ in a given class of fixed VC dimension, the learning algorithm should minimize (in that class) a trade-off between the source risk $\widehat{R}_S(\eta)$ and the empirical $\mathcal{H}$-divergence $\hat{d}_{\mathcal{H}}(S, T)$. As pointed out in [31], a strategy to control the $\mathcal{H}$-divergence is to find a representation of the examples where both the source and the target domain are as indistinguishable as possible. Under such a representation, a hypothesis with a low source risk will, according to Theorem 10.1, perform well on the target data.

10.3 Domain-Adversarial Neural Networks (DANN)

An original aspect of our approach is to explicitly implement the idea exhibited by Theorem 10.1 into a neural network classifier. That is, to learn a model that can generalize well from one domain to another, we ensure that the internal representation of the neural network contains no discriminative information about the origin of the input (source or target), while preserving a low risk on the source (labeled) examples. In this section, we detail the proposed approach for incorporating a "domain adaptation component" to neural networks. We start by developing the idea for the simplest possible case, i.e., a single hidden layer, fully connected neural network. We then describe how to generalize the approach to arbitrary (deep) architectures.

Shallow Neural Network Let us first consider a standard neural network architecture with a single hidden layer. For simplicity, we suppose that the input space is formed by m-dimensional real vectors. Thus, $X = \mathbb{R}^m$. The hidden layer G_f learns a function $G_f : X \to \mathbb{R}^D$ that maps an example into a new D-dimensional representation,[2] and is parametrized by a matrix-vector pair $(\mathbf{W}, \mathbf{b}) \in \mathbb{R}^{D \times m} \times \mathbb{R}^D$:

$$G_f(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \text{sigm}\big(\mathbf{W}\mathbf{x} + \mathbf{b}\big), \quad \text{with } \text{sigm}(\mathbf{a}) = \left[\frac{1}{1+\exp(-a_i)}\right]_{i=1}^{|\mathbf{a}|}. \tag{10.2}$$

Similarly, the prediction layer G_y learns a function $G_y : \mathbb{R}^D \to [0, 1]^L$ parametrized by a pair $(\mathbf{V}, \mathbf{c}) \in \mathbb{R}^{L \times D} \times \mathbb{R}^L$:

$$G_y(G_f(\mathbf{x}); \mathbf{V}, \mathbf{c}) = \text{softmax}\big(\mathbf{V}G_f(\mathbf{x}) + \mathbf{c}\big), \quad \text{with } \text{softmax}(\mathbf{a}) = \left[\frac{\exp(a_i)}{\sum_{j=1}^{|\mathbf{a}|}\exp(a_j)}\right]_{i=1}^{|\mathbf{a}|}.$$

Here we have $L = |Y|$. By using the softmax function, each component of vector $G_y(G_f(\mathbf{x}))$ denotes the conditional probability that the neural network assigns $\mathbf{x}$ to the class in Y represented by that component. Given a source example $(\mathbf{x}_i, y_i)$, the natural classification loss to use is the negative log-probability of the correct label: $\mathcal{L}_y\big(G_y(G_f(\mathbf{x}_i)), y_i\big) = -\log\big[G_y(G_f(\mathbf{x}))_{y_i}\big]$. Training the neural network then leads to the following optimization problem on the source domain:

$$\min_{\mathbf{W},\mathbf{b},\mathbf{V},\mathbf{c}} \left[\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_y^i(\mathbf{W}, \mathbf{b}, \mathbf{V}, \mathbf{c}) + \lambda \cdot R(\mathbf{W}, \mathbf{b})\right], \tag{10.3}$$

where $\mathcal{L}_y^i(\mathbf{W}, \mathbf{b}, \mathbf{V}, \mathbf{c}) = \mathcal{L}_y\big(G_y(G_f(\mathbf{x}_i; \mathbf{W}, \mathbf{b}); \mathbf{V}, \mathbf{c}), y_i\big)$ is a shorthand notation for the prediction loss on the i-th example, and $R(\mathbf{W}, \mathbf{b})$ is an optional regularizer that is weighted by hyperparameter λ.

The heart of our approach is to design a *domain regularizer* directly derived from the $\mathcal{H}$-divergence of Definition 10.1. To this end, we view the output of the hidden layer G_f (Eq. (10.2)) as the internal representation of the neural network. Thus, we denote the source sample representations as $S(G_f) = \big\{G_f(\mathbf{x}) \,\big|\, \mathbf{x} \in S\big\}$. Similarly, given an unlabeled sample from the target domain we denote the corresponding representations $T(G_f) = \big\{G_f(\mathbf{x}) \,\big|\, \mathbf{x} \in T\big\}$. Based on Eq. (10.1), $\hat{d}_{\mathcal{H}}\big(S(G_f), T(G_f)\big)$, i.e., the empirical $\mathcal{H}$-divergence of a symmetric hypothesis class $\mathcal{H}$ between samples $S(G_f)$ and $T(G_f)$, is given by:

$$2\left(1 - \min_{\eta \in \mathcal{H}}\left[\frac{1}{n}\sum_{i=1}^{n} I\big[\eta(G_f(\mathbf{x}_i))=0\big] + \frac{1}{n'}\sum_{i=n+1}^{N} I\big[\eta(G_f(\mathbf{x}_i))=1\big]\right]\right). \tag{10.4}$$

[2] For brevity of notation, we will sometimes drop the dependence of G_f on its parameters $(\mathbf{W}, \mathbf{b})$ and shorten $G_f(\mathbf{x}; \mathbf{W}, \mathbf{b})$ to $G_f(\mathbf{x})$.

Let us consider $\mathcal{H}$ as the class of hyperplanes in the representation space. Inspired by the Proxy $\mathcal{A}$-distance (see Sect. 10.2), we suggest estimating the "min" part of Eq. (10.4) by a *domain classification layer* G_d that learns a logistic regressor $G_d : \mathbb{R}^D \to [0, 1]$, parametrized by a vector-scalar pair $(\mathbf{u}, z) \in \mathbb{R}^D \times \mathbb{R}$, that models the probability that a given input is from the source domain $\mathcal{D}_{\mathrm{S}}^{X}$ or the target domain $\mathcal{D}_{\mathrm{T}}^{X}$. Thus,

$$G_d(G_f(\mathbf{x}); \mathbf{u}, z) \;=\; \mathrm{sigm}\big(\mathbf{u}^\top G_f(\mathbf{x}) + z\big)\,. \tag{10.5}$$

Hence, the function $G_d(\cdot)$ is a *domain regressor*. We define its loss by:

$$\mathcal{L}_d\big(G_d(G_f(\mathbf{x}_i)), d_i\big) \;=\; -d_i \log\big[\mathbf{d}(\mathbf{f}(\mathbf{x}_i))\big] - (1-d_i)\log\big[1-\mathbf{d}(\mathbf{f}(\mathbf{x}_i))\big]\,.$$

where d_i denotes the binary variable (*domain label*) for the i-th example, which indicates whether $\mathbf{x}_i$ come from the source distribution ($\mathbf{x}_i \sim \mathcal{D}_{\mathrm{S}}^{X}$ if $d_i=0$) or from the target distribution ($\mathbf{x}_i \sim \mathcal{D}_{\mathrm{T}}^{X}$ if $d_i=1$).

Recall that for the examples from the source distribution ($d_i=0$), the corresponding labels $y_i \in Y$ are known at training time. For the examples from the target domains, we do not know the labels at training time, and we want to predict such labels at test time. This enables us to add a domain adaptation term to the objective of Eq. (10.3), giving the following regularizer:

$$R(\mathbf{W}, \mathbf{b}) \;=\; \max_{\mathbf{u},z}\left[-\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_d^i(\mathbf{W}, \mathbf{b}, \mathbf{u}, z) - \frac{1}{n'}\sum_{i=n+1}^{N}\mathcal{L}_d^i(\mathbf{W}, \mathbf{b}, \mathbf{u}, z)\right], \tag{10.6}$$

where $\mathcal{L}_d^i(\mathbf{W}, \mathbf{b}, \mathbf{u}, z) = \mathcal{L}_d\big(G_d(G_f(\mathbf{x}_i; \mathbf{W}, \mathbf{b}); \mathbf{u}, z), d_i\big)$. This regularizer seeks to approximate the $\mathcal{H}$-divergence of Eq. (10.4), as $2(1 - R(\mathbf{W}, \mathbf{b}))$ is a surrogate for $\hat{d}_{\mathcal{H}}\big(S(G_f), T(G_f)\big)$. In line with Theorem 10.1, the optimization problem given by Eqs. (10.3) and (10.6) implements a trade-off between the minimization of the source risk $\widehat{R}_S(\cdot)$ and the divergence $\hat{d}_{\mathcal{H}}(\cdot, \cdot)$. The hyperparameter λ is then used to tune the trade-off between these two quantities during the learning process.

For learning, we first note that we can rewrite the complete optimization objective of Eq. (10.3) as follows:

$$\begin{aligned} &E(\mathbf{W}, \mathbf{V}, \mathbf{b}, \mathbf{c}, \mathbf{u}, z) \\ &\quad= \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_y^i(\mathbf{W}, \mathbf{b}, \mathbf{V}, \mathbf{c}) - \lambda\left(\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_d^i(\mathbf{W}, \mathbf{b}, \mathbf{u}, z) + \frac{1}{n'}\sum_{i=n+1}^{N}\mathcal{L}_d^i(\mathbf{W}, \mathbf{b}, \mathbf{u}, z)\right). \end{aligned} \tag{10.7}$$

We are seeking the parameters $\hat{\mathbf{W}}, \hat{\mathbf{V}}, \hat{\mathbf{b}}, \hat{\mathbf{c}}, \hat{\mathbf{u}}, \hat{z}$ that deliver a saddle point given by

$$(\hat{\mathbf{W}}, \hat{\mathbf{V}}, \hat{\mathbf{b}}, \hat{\mathbf{c}}) = \operatorname*{arg\,min}_{\mathbf{W},\mathbf{V},\mathbf{b},\mathbf{c}} E(\mathbf{W}, \mathbf{V}, \mathbf{b}, \mathbf{c}, \hat{\mathbf{u}}, \hat{z})\,, \text{ with } (\hat{\mathbf{u}}, \hat{z}) = \operatorname*{arg\,max}_{\mathbf{u},z} E(\hat{\mathbf{W}}, \hat{\mathbf{V}}, \hat{\mathbf{b}}, \hat{\mathbf{c}}, \mathbf{u}, z)\,.$$

Thus, the optimization problem involves a minimization with respect to some parameters, as well as a maximization with respect to the others.

We propose to tackle this problem with a simple stochastic gradient procedure, in which updates are made in the opposite direction of the gradient of Eq. (10.7) for the minimizing parameters, and in the direction of the gradient for the maximizing parameters. Stochastic estimates of the gradient are made, using a subset of the training samples to compute the averages.

During training, the neural network (parametrized by $\mathbf{W}$, $\mathbf{b}$, $\mathbf{V}$, $\mathbf{c}$) and the domain regressor (parametrized by $\mathbf{u}$, z) are competing against each other, in an adversarial way, over the objective of Eq. (10.7). For this reason, we refer to networks trained according to this objective as Domain-Adversarial Neural Networks (DANN). It will effectively attempt to learn a hidden layer $G_f(\cdot)$ that maps an example (either source or target) into a representation allowing the output layer $G_y(\cdot)$ to accurately classify source samples, but crippling the ability of the domain regressor $G_d(\cdot)$ to detect whether each example belongs to the source or target domains.

Generalization to Arbitrary Architectures It is straightforward to generalize the single hidden layer DANN, presented above, to other sophisticated architectures, which might be more appropriate for the data at hand. To do so, let us now use a more general notation for the different components of DANN. Namely, let $G_f(\cdot;\theta_f)$ be the D-dimensional neural network feature extractor, with parameters θ_f. Also, let $G_y(\cdot;\theta_y)$ be the part of DANN that computes the network's label prediction output layer, with parameters θ_y, while $G_d(\cdot;\theta_d)$ now corresponds to the computation of the domain prediction output of the network, with parameters θ_d. Note that for preserving the theoretical guarantees of Theorem 10.1, the hypothesis class $\mathcal{H}_d$ generated by the domain prediction component G_d should include the hypothesis class $\mathcal{H}_y$ generated by the label prediction component G_y. Thus, $\mathcal{H}_y \subseteq \mathcal{H}_d$. We will note the prediction loss and the domain loss respectively by

$$\mathcal{L}_y^i(\theta_f,\theta_y) = \mathcal{L}_y\big(G_y(G_f(\mathbf{x}_i;\theta_f);\theta_y), y_i\big)\,;\;\; \mathcal{L}_d^i(\theta_f,\theta_d) = \mathcal{L}_d\big(G_d(G_f(\mathbf{x}_i;\theta_f);\theta_d), d_i\big).$$

Training DANN then parallels the single layer case and consists in optimizing

$$E(\theta_f,\theta_y,\theta_d) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_y^i(\theta_f,\theta_y) - \lambda\left(\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_d^i(\theta_f,\theta_d) + \frac{1}{n'}\sum_{i=n+1}^{N}\mathcal{L}_d^i(\theta_f,\theta_d)\right) \quad (10.8)$$

by finding the saddle point $\hat{\theta}_f, \hat{\theta}_y, \hat{\theta}_d$ such that

$$(\hat{\theta}_f,\hat{\theta}_y) = \arg\min_{\theta_f,\theta_y} E(\theta_f,\theta_y,\hat{\theta}_d)\,, \quad \text{with } \hat{\theta}_d = \arg\max_{\theta_d} E(\hat{\theta}_f,\hat{\theta}_y,\theta_d)\,. \quad (10.9)$$

As suggested previously, a saddle point defined by Eq. (10.9) can be found as a stationary point of the following gradient updates:

$$\theta_f \longleftarrow \theta_f - \mu\left(\frac{\partial \mathcal{L}_y^i}{\partial \theta_f} - \lambda \frac{\partial \mathcal{L}_d^i}{\partial \theta_f}\right), \tag{10.10}$$

$$\theta_y \longleftarrow \theta_y - \mu \frac{\partial \mathcal{L}_y^i}{\partial \theta_y}, \quad \text{and} \quad \theta_d \longleftarrow \theta_d - \mu\lambda \frac{\partial \mathcal{L}_d^i}{\partial \theta_d}, \tag{10.11}$$

where μ is the learning rate. We use stochastic estimates of these gradients, by sampling examples from the data set.

The updates of Eqs. (10.10) and (10.11) are very similar to stochastic gradient descent (SGD) updates for a feed-forward deep model that comprises feature extractor fed into the label predictor and into the domain classifier (with loss weighted by λ). The only difference is that in Eq. (10.10), the gradients from the class and domain predictors are subtracted, instead of being summed (the difference is important, as otherwise SGD would try to make features dissimilar across domains in order to minimize the domain classification loss). Since SGD—and its many variants, such as ADAGRAD [141] or ADADELTA [571]—is the main learning algorithm implemented in most libraries for deep learning, it would be convenient to frame an implementation of our stochastic saddle point procedure as SGD.

Fortunately, such a reduction can be accomplished by introducing a special *Gradient Reversal Layer* (GRL), defined as follows. The Gradient Reversal Layer has no parameters associated with it. During the forward propagation, the GRL acts as an identity transformation. During the backpropagation however, the GRL takes the gradient from the subsequent level and changes its sign, i.e. multiplies it by -1, before passing it to the preceding layer. Implementing such a layer using existing object-oriented packages for deep learning is simple, requiring only to define procedures for the forward propagation (identity transformation), and backpropagation (multiplying by -1). The layer requires no parameter update.

The GRL as defined above is inserted between the feature extractor G_f and the domain classifier G_d, resulting in the architecture depicted in Fig. 10.1. As the backpropagation process passes through the GRL, the partial derivatives of the loss that is downstream the GRL (i.e., $\mathcal{L}_d$) w.r.t. the layer parameters that are upstream the GRL (i.e., θ_f) get multiplied by -1, i.e., $\frac{\partial \mathcal{L}_d}{\partial \theta_f}$ is effectively replaced with $-\frac{\partial \mathcal{L}_d}{\partial \theta_f}$. Therefore, running SGD in the resulting model implements the updates of Eqs. (10.10) and (10.11) and converges to a saddle point of Eq. (10.8).

Mathematically, we can formally treat the Gradient Reversal Layer as a "pseudofunction" $\mathcal{R}(\mathbf{x})$ defined by two (incompatible) equations describing its forward and backpropagation behavior:

$$\mathcal{R}(\mathbf{x}) = \mathbf{x}, \quad \text{and} \quad \frac{d\mathcal{R}}{d\mathbf{x}} = -\mathbf{I},$$

where $\mathbf{I}$ is an identity matrix. Then, we define the objective "pseudofunction" of $(\theta_f, \theta_y, \theta_d)$ that is being optimized by the gradient descent within our method:

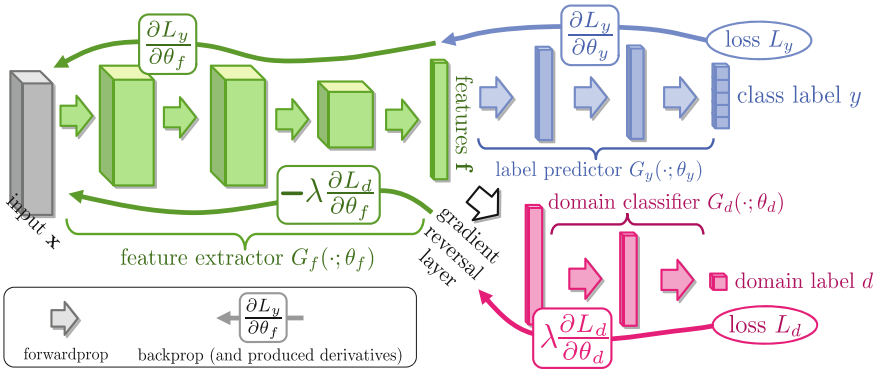


Fig. 10.1 The **proposed architecture** includes a *feature extractor* (*green*) and a *label predictor* (*blue*), which together form a standard feed-forward architecture. Unsupervised DA is achieved by adding a *domain classifier* (*red*) connected to the feature extractor via a *Gradient Reversal Layer* (GRL) that multiplies the gradient by a certain negative constant during the backpropagation-based training. The GRL ensures that the feature distributions over the two domains are made similar (as indistinguishable as possible for the domain classifier)

$$\tilde{E}(\theta_f,\theta_y,\theta_d)=\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_y\left(G_y(G_f(\mathbf{x}_i;\theta_f);\theta_y),y_i\right)$$
$$-\lambda\Big(\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_d\left(G_d(\mathcal{R}(G_f(\mathbf{x}_i;\theta_f));\theta_d),d_i\right)+\frac{1}{n'}\sum_{i=n+1}^{N}\mathcal{L}_d\left(G_d(\mathcal{R}(G_f(\mathbf{x}_i;\theta_f));\theta_d),d_i\right)\Big). \quad (10.12)$$

Running updates Eqs. (10.10)–(10.11) can then be implemented as doing SGD for Eq. (10.12) and leads to the emergence of features that are domain-invariant and discriminative at the same time. After the learning, the label predictor $G_y(G_f(\mathbf{x};\theta_f);\theta_y)$ can be used to predict labels for both target and source samples.

10.4 Experiments

10.4.1 Toy Experiment with Shallow Neural Networks

In this first experiment section, we evaluate the adaptation capability of the simple version of DANN described by Eq. (10.7). To do so, we compare its decision boundary and internal representation to the ones of a standard neural network (NN), on a variant of the *intertwinning moons* 2D problem.[3] In these toy experiments, both algorithms share the same network architecture, with a hidden layer size of 15 neurons. The

[3]To create the source sample S, we generate a lower moon and an upper moon labeled 0 and 1 respectively, each of which containing 150 examples. The target sample T is obtained by (1) generating a sample S' the same way S has been generated; (2) rotating each example by 35°; and (3) removing all the labels. Thus, T contains 300 unlabeled examples.

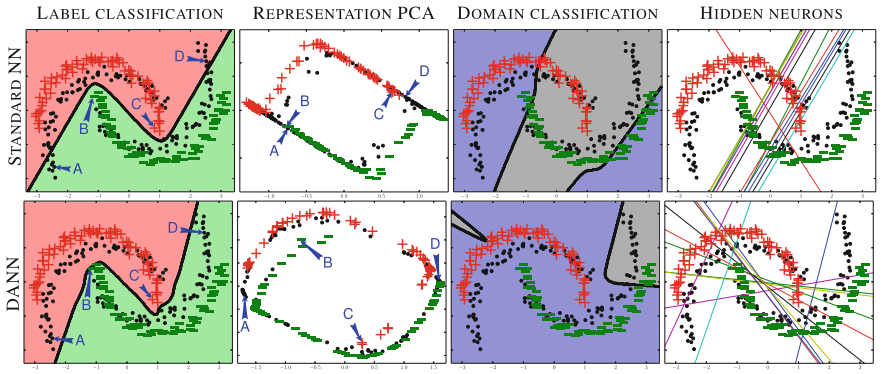


Fig. 10.2 The *intertwinning moons* toy problem. Examples from the source sample are represented as a "+" (label 1) and a "−" (label 0), while examples from the unlabeled target sample are represented as *black dots*. See text for the figure discussion

stochastic gradient descent approach here consists of sampling a pair of source and target examples and performing a gradient step update of all network parameters. Crucially, while the update of the regular parameters follows as usual the opposite direction of the gradient, for the adversarial parameters the of the DANN step must the gradient's direction. We train the NN using a very similar procedure. That is, we even keep updating the domain regressor component using target sample T (with a hyperparameter $\lambda = 6$; the same value is used for DANN), but we disable the *adversarial* backpropagation into the hidden layer. This allows recovering the NN learning algorithm—based on the source risk minimization of Eq. (10.3) without any regularizer—and simultaneously train the domain regressor of Eq. (10.5) to discriminate between source and target domains.

The analysis of the experiment appears in Fig. 10.2, where we compare standard NN (upper graphs) and DANN (lower graphs) from four different perspectives, described in details below.

Label Classification As expected, NN accurately classifies the two classes of the source sample S, but is *not fully adapted* to the target sample T. On the contrary, the decision boundary of DANN perfectly classifies examples from both source and target samples. In the studied task, DANN clearly adapts to the target distribution.

Representation PCA The graphs are obtained by applying a Principal component analysis (PCA) on the set of all representation of source and target data points, i.e., $S(G_f) \cup T(G_f)$. Thus, given the trained network (NN or DANN), every point from S and T is mapped into a 15-dimensional feature space through the hidden layer, and projected back into a two-dimensional plane by the PCA transformation. In the DANN-PCA representation, we observe that target points are homogeneously spread out among source points; In the NN-PCA representation, many target points belong to clusters containing no source points. Hence, labeling the target points seems an easier task given the DANN-PCA representation. To highlight that the

representation promoted by DANN is better suited to the adaptation problem, the "Label classification" and "PCA representation" graphs tag four data points (denoted A, B, C, D) that correspond to the moon extremities in the original space. Points A and B are very close to each other in the NN-PCA representation, while they clearly belong to different classes. The same happens to points C and D. Conversely, these four points are at the opposite four corners in the DANN-PCA representation. Also, the target point A (resp. D)—that is difficult to classify in the original space—is located in the "+" cluster (resp. "−" cluster) in the DANN-PCA representation.

Domain Classification We show the decision boundary on the domain classification problem, which is given by the domain regressor G_d of Eq. (10.5). More precisely, an example $\mathbf{x}$ is classified as a source example when $G_d(G_f(\mathbf{x})) \geq 0.5$, and is classified as a target example otherwise. As explained above, we trained a domain regressor during the learning process of NN, but without allowing it to influence the learned representation $G_f(\cdot)$. On one hand, the DANN domain regressor clearly fails to generalize source and target distribution topologies. On the other hand, the NN domain regressor shows a better (although imperfect) generalization capability, as it roughly capture the rotation angle of the target distribution.

Hidden Neurons We show the configuration of hidden layer neurons (by Eq. (10.2), we have that each neuron is indeed a linear regressor). In other words, each of the 15 plot line corresponds to the coordinates $\mathbf{x} \in \mathbb{R}^2$ for which the ith component of $G_f(\mathbf{x})$ equals $\frac{1}{2}$, for $i \in \{1, \ldots, 15\}$. We observe that the standard NN neurons are grouped in three clusters, each one allowing to generate a straight line of the *zigzag* decision boundary for the label classification problem. However, most of these neurons are also able to (roughly) capture the rotation angle of the domain classification problem. Hence, we observe that the adaptation regularizer of DANN prevents these kinds of neurons to be produced. It is indeed striking to see that the two predominant patterns in the NN neurons (i.e., the two parallel lines crossing the plane from lower left to upper right) are vanishing in the DANN neurons.

10.4.2 Deep Networks on Image Classification

We now perform extensive evaluation of a deep version of DANN (see Eq. (10.12)) on a number of popular image data sets and their modifications. These include large-scale data sets of small images popular with deep learning methods (see examples in Fig. 10.3), and the OFF31 dataset [407], which are a *de facto* standard for DA in computer vision, but have much fewer images.

Baselines The following baselines are evaluated in the experiments of this subsection. The *source-only* model is trained without consideration for target-domain data (no domain classifier branch included into the network). The *train-on-target* model is trained on the target domain with class labels revealed. This model serves as an upper bound on DA methods, assuming that target data are abundant and the shift

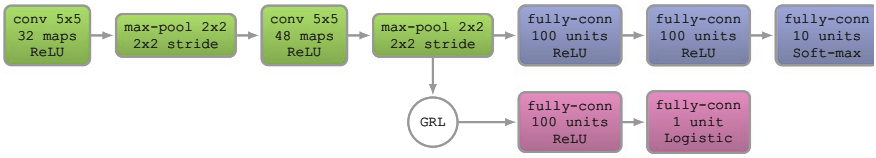
Fig. 10.3 Examples of domain pairs used in the experiments of Sect. 10.4.2

between the domains is considerable. In addition, we compare our approach against the recently proposed unsupervised DA method based on *Subspace Alignment (SA)* [164], which is simple to setup and test on new data sets, but has also been shown to perform very well in experimental comparisons with other "shallow" DA methods. To boost the performance of this baseline, we pick its most important free parameter (the number of principal components) from the range $\{2, \ldots, 60\}$, so that the test performance on the target domain is maximized. To apply SA in our setting, we train a source-only model and then consider the activations of the last hidden layer in the label predictor (before the final linear classifier) as descriptors/features, and learn the mapping between the source and the target domains [164]. Since the SA baseline requires training a new classifier after adapting the features, and in order to put all the compared settings on an equal footing, we retrain the last layer of the label predictor using a standard linear SVM [150] for all four considered methods (including ours; the performance on the target domain remains approximately the same after the retraining). For the Office (OFF31) dataset [407], we directly compare the performance of our full network (feature extractor and label predictor) against recent DA approaches using previously published results.

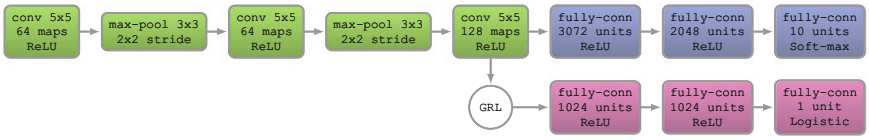
CNN Architectures and Training Procedure In general, we compose feature extractor from two or three convolutional layers, picking their exact configurations from previous works. More precisely, four different architectures were used in our experiments. The first three are shown in Fig. 10.4. For the OFF31 domains, we use pretrained `AlexNet` from the `Caffe`-package [255]. The adaptation architecture is identical to [499].[4]

For the domain adaption component, we use three $(x \to 1024 \to 1024 \to 2)$ fully connected layers, except for MNIST where we used a simpler $(x \to 100 \to 2)$ architecture to speed up the experiments. Admittedly these choices for domain classifier are arbitrary, and better adaptation performance might be attained if this part of the architecture is tuned. For the loss functions, we set $\mathcal{L}_y$ and $\mathcal{L}_d$ to be the logistic regression loss and the binomial cross-entropy respectively. Following [455] we also use dropout and ℓ_2-norm restriction when we train the SVHN architecture. The learning rate is adjusted during the stochastic gradient descent using the formula $\mu_p = \frac{\mu_0}{(1+\alpha \cdot p)^\beta}$, where p is the training progress linearly changing from 0 to 1, $\mu_0 = 0.01$, $\alpha = 10$ and $\beta = 0.75$ (the schedule was optimized to promote convergence and low error

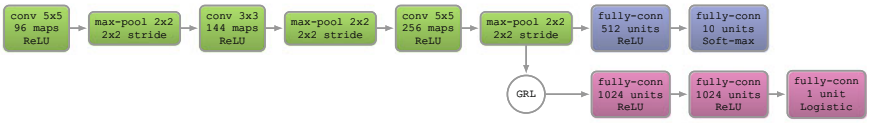
[4] A 2-layer domain classifier $(x \to 1024 \to 1024 \to 2)$ is attached to the 256-dimensional bottleneck of `fc7`.



(a) MNIST architecture; inspired by the classical LeNet-5 [291].



(b) SVHN architecture; adopted from [455].



(c) GTSRB architecture; we used the single-CNN baseline from [93] as our starting point.

Fig. 10.4 CNN architectures used in the experiments. *Boxes* correspond to transformations applied to the data. Color-coding is the same as in Fig. 10.1

on the *source* domain). A momentum term of 0.9 is also used. The domain adaptation parameter λ is initiated at 0 and is gradually changed to 1 using the schedule $\lambda_p = \frac{2}{1+\exp(-\gamma \cdot p)} - 1$, where γ was set to 10 in all experiments (the schedule was not optimized/tweaked). This strategy allows the domain classifier to be less sensitive to noisy signal at the early stages of the training procedure. Note however that these λ_p were used only for updating the *feature extractor* component G_f. For updating the *domain classification* component, we used a fixed $\lambda = 1$, to ensure that the latter trains as fast as the *label predictor* G_y.[5]

Finally, note that the model is trained on 128-sized batches (images are pre-processed by the mean subtraction). A half of each batch is populated by the samples from the source domain (with known labels), the rest constitutes the target domain (with labels not revealed to the algorithms except for the train-on-target baseline).

Visualizations We use t-SNE [501] projection to visualize feature distributions at different points of the network, while color-coding the domains (Fig. 10.5). As we already observed with the shallow version of DANN (see Fig. 10.2), there is a strong correspondence between the success of the adaptation in terms of the classification accuracy for the target domain, and the overlap between the domain distributions in such visualizations.

[5] Equivalently, one can use the same λ_p for both feature extractor and domain classification components, but use a learning rate of μ/λ_p for the latter.

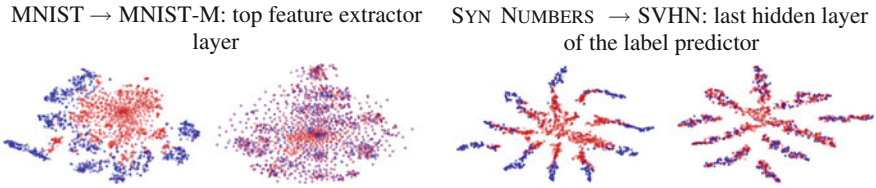


Fig. 10.5 The effect of adaptation on the distribution of the extracted features (*best viewed in color*). The figure shows t-SNE [501] visualizations of the CNN's activations (**a**) when no adaptation was performed and (**b**) when our adaptation procedure was incorporated into training. *Blue* points correspond to the source domain examples, while *red* ones correspond to the target domain

Table 10.1 Digit image classifications accuracies. The first row corresponds to the lower performance bound (no adaptation). The last row corresponds to training on the target data with known labels (upper bound on the DA performance). For SA and DANN we show how much of the gap between the lower and the upper bounds was covered (in brackets)

METHOD	SOURCE	MNIST	SYN NUMBERS	SVHN	SYN SIGNS
	TARGET	MNIST-M	SVHN	MNIST	GTSRB
SOURCE ONLY		0.5225	0.8674	0.5490	0.7900
SA [164]		0.5690 (4.1%)	0.8644 (−5.5%)	0.5932 (9.9%)	0.8165 (12.7%)
DANN		**0.7666** (52.9%)	**0.9109** (79.7%)	**0.7385** (42.6%)	**0.8865** (46.4%)
TRAIN ON TARGET		0.9596	0.9220	0.9942	0.9980

Results On Image Data Sets We now discuss the experimental settings and the results. In each case, we train on the source data set and test on a different target domain data set, with considerable shifts between domains (see Fig. 10.3). The results are summarized in Tables 10.1 and 10.2.

MNIST → MNIST-M Our first experiment deals with the MNIST data set [291] (source). In order to obtain the target domain (MNIST- M) we blend digits from the original set over patches randomly extracted from color photos from BSDS500 [16]. This operation is formally defined for two images I^1, I^2 as $I^{out}_{ijk} = |I^1_{ijk} - I^2_{ijk}|$, where i, j are the coordinates of a pixel and k is a channel index. In other words, an output sample is produced by taking a patch from a photo and inverting its pixels at positions corresponding to the pixels of a digit. For a human the classification task becomes only slightly harder compared to the original data set (the digits are still clearly distinguishable) whereas for a CNN trained on MNIST this domain is quite distinct, as the background and the strokes are no longer constant. Consequently, the source-only model performs poorly. Our approach succeeded at aligning feature distributions (Fig. 10.5), which led to successful adaptation results (considering that the adaptation is unsupervised). At the same time, the improvement over source-only model achieved by SA [164] is quite modest, thus highlighting the difficulty of the adaptation task.

Table 10.2 Accuracy evaluation of different DA approaches on the standard OFF31 [407] data set. All methods (except SA) are evaluated in the "fully transductive" protocol (some results are reproduced from [309])

METHOD S → T	GFK [200]	SA* [164]	DLID [88]	DDC [499]	DAN [309]	SOURCE ONLY	DANN
A → W	0.197	0.450	0.519	0.618	0.685	0.642	**0.730**
D → W	0.497	0.648	0.782	0.950	0.960	0.961	**0.964**
W → D	0.663	0.699	0.899	0.985	0.990	0.978	**0.992**

Synthetic numbers → SVHN To address a common scenario of training on synthetic data and testing on real data, we use Street-View House Number data set SVHN [342] as the target domain and synthetic digits as the source. The latter (SYN NUMBERS) consists of ≈500,000 images generated by ourselves from Windows™ fonts by varying the text (that includes different one-, two-, and three-digit numbers), positioning, orientation, background and stroke colors, and the amount of blur. The degrees of variation were chosen manually to simulate SVHN, however the two data sets are still rather distinct, the biggest difference being the structured clutter in the background of SVHN images.

The proposed backpropagation-based technique works well covering almost 80% of the gap between training with source data only and training on target domain data with known target labels. In contrast, SA [164] results in a slight classification accuracy drop (probably due to the information loss during the dimensionality reduction), indicating that the adaptation task is even more challenging than in the case of the MNIST experiment.

MNIST ↔ SVHN In this experiment, we further increase the gap between distributions, and test on MNIST and SVHN, which are significantly different in appearance. Training on SVHN even without adaptation is challenging—classification error stays high during the first 150 epochs. In order to avoid ending up in a poor local minimum we, therefore, do not use learning rate annealing here. Obviously, the two directions (MNIST → SVHN and SVHN → MNIST) are not equally difficult. As SVHN is more diverse, a model trained on SVHN is expected to be more generic and to perform reasonably on the MNIST data set. This, indeed, turns out to be the case and is supported by the appearance of the feature distributions. We observe a quite strong separation between the domains when we feed them into the CNN trained solely on MNIST, whereas for the SVHN-trained network the features are much more intermixed. This difference probably explains why our method succeeded in improving the performance by adaptation in the SVHN → MNIST scenario (see Table 10.1) but not in the opposite direction (SA is not able to perform adaptation in this case either). Unsupervised adaptation from MNIST to SVHN gives a failure example for our approach: it doesn't manage to improve upon the performance of the non-adapted model which achieves ≈0.25 accuracy (we are unaware of any unsupervised DA methods capable of performing such adaptation).

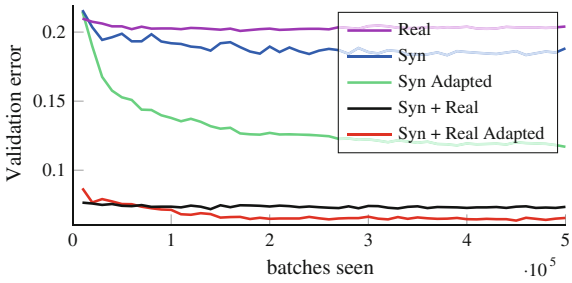


Fig. 10.6 Results for the traffic signs classification in the semi-supervised setting. *Syn* and *Real* denote available labeled images (100,000 synthetic and 430 real respectively); *Adapted* means that ≈31,000 unlabeled target domain images were used for adaptation. The best performance is achieved by employing both the labeled samples and the unlabeled corpus in the target domain

Synthetic Signs → *GTSRB* Overall, this setting is similar to the SYN NUMBERS → SVHN experiment, except the distribution of the features is more complex due to the significantly larger number of classes (43 instead of 10). For the source domain we obtained 100,000 synthetic images (which we call SYN SIGNS) simulating various imaging conditions. In the target domain, we use 31,367 random training samples for unsupervised adaptation and the rest for evaluation. Once again, our method achieves a sensible increase in performance proving its suitability for the synthetic-to-real data adaptation.

As an additional experiment, we also evaluate the proposed algorithm for semi-supervised DA, i.e., when one is additionally provided with a small amount of labeled target data. Here, we reveal 430 labeled examples (10 samples per class) and add them to the training set for the label predictor. Figure 10.6 shows the change of the validation error throughout the training. While the graph clearly suggests that our method can be beneficial in the semi-supervised setting, thorough verification of semi-supervised setting is left for future work.

OFF31 Dataset We finally evaluate our method on OFF31 data set, which is a collection of three distinct domains: (A)MAZON, (D)SLR, and (W)EBCAM. Unlike previously discussed data sets, OFF31 is rather small-scale with only 2817 labeled images spread across 31 different categories in the largest domain. The amount of available data is crucial for a successful training of a deep model, hence we opted for the fine-tuning of the CNN pretrained on the ImageNet (`AlexNet` from the `Caffe` package [255]) as it is done in some recent DA works [127, 240, 309, 499]. We make our approach comparable with [499] by using exactly the same network architecture replacing domain mean-based regularization with the domain classifier.

Following previous works, we assess the performance of our method across three transfer tasks most commonly used for evaluation. Our training protocol is adopted from [88, 196, 309] as during adaptation we use all available labeled source examples and unlabeled target examples (the premise of our method is the abundance of unlabeled data in the target domain). Also, all source domain data are used for

training. Under this "fully transductive" setting, our method is able to improve previously reported state-of-the-art accuracy for unsupervised adaptation very considerably (Table 10.2), especially in the most challenging AMAZON → WEBCAM scenario (the two domains with the largest domain shift). Interestingly, in all three experiments we observe a slight overfitting (performance on the target domain degrades while accuracy on the source continues to improve) as training progresses, however, it does not ruin the validation accuracy. Moreover, switching off the domain classifier branch makes this effect far more apparent, from which we conclude that our technique serves as a regularizer.

10.4.3 Deep Image Descriptors for Re-identification

In this section we discuss the application of the described adaptation method to person re-identification *(re-id)* problem. The task of person re-identification is to match pedestrian images coming from multiple cameras. Unlike classification problems that are discussed above, re-identification problem implies that each image is mapped to a vector descriptor. The distance between descriptors is then used to match images. To evaluate results of re-id methods the *Cumulative Match Characteristic* (CMC) curve is commonly used. It is a plot of the probability of correct identification for different sizes of candidate list returned.

Most existing works train descriptor mappings and evaluate them within the same data set containing images from a certain camera network with similar imaging conditions. Several papers, however, observed that the performance of the resulting re-identification systems drops considerably when descriptors trained on one data set and tested on another. It is therefore natural to handle such cross-domain evaluation as a domain adaptation problem, where each camera network (data set) constitutes a domain. Recently, several papers with significantly improved re-identification performance [352, 583, 584] have been presented, with [317] reporting good results in cross-data-set evaluation scenario. At the moment, deep learning methods [566] do not achieve state-of-the-art results probably because of the limited size of the training sets. Domain adaptation thus represents a viable direction for improving deep re-identification descriptors.

Data Sets and Protocols Following [317], we use PRID [233], VIPeR [211], CUHK [298] as target data sets for our experiments (see examples in Fig. 10.7). The *PRID* data set exists in two versions, and as in [317] we use a single-shot variant. It contains images of 385 persons viewed from camera A and images of 749 persons viewed from camera B, 200 persons appear in both cameras. The *VIPeR* data set also contains images taken with two cameras, and in total 632 persons are captured, for every person there is one image for each of the two camera views. The *CUHK* data set consists of images from five pairs of cameras, two images for each person for each camera inthe pair. We refer to the subset of this data set that includes the first pair of cameras only as *CUHK/p1* (as most papers use this subset).

Fig. 10.7 Matching and non-matching pairs of probe-gallery images from different person re-identification data sets. The three data sets are treated as different domains in our experiments

We perform extensive experiments for various pairs of data sets, where one data set serves as a source domain, i.e., it is used to train a descriptor mapping in a supervised way. The second data set is used as a target domain (the labeling is not used). In more detail, CUHK/p1 is used for experiments when CUHK serves as a target domain and two settings ("whole CUHK" and CUHK/p1) are used for experiments when CUHK serves as a source domain. Given PRID as a target data set, we randomly choose 100 persons appearing in both camera views as training set. The images of the other 100 persons from camera A are used as probe, all images from camera B excluding those used in training (649 in total) are used as gallery at test time. For VIPeR, we use random 316 persons for training and all others for testing. For CUHK, 971 persons are split into 485 for training and 486 for testing.

Following [566], we augmented our data with mirror images, and during test time we calculate similarity score between two images as the mean of the four scores corresponding to different flips of the two compared images. In case of CUHK, where there are 4 images (including mirror images) for each of the two camera views for each person, all 16 combinations' scores are averaged.

CNN architectures and Training Procedure In our experiments, we use siamese architecture proposed in [566] for learning deep image descriptors on the source data set. This architecture incorporates two convolution layers (with 7×7 and 5×5 filter banks), followed by ReLU and max pooling, and one fully connected layer, which gives 500-dimensional descriptors as an output. There are three parallel flows within the CNN for processing three part of an image: the upper, the middle, and the lower one. The first convolution layer shares parameters between three parts, and the outputs of the second convolution layers are concatenated. During training, we follow [566] and calculate pairwise cosine similarities between 500-dimensional features within each batch and backpropagate the loss for all pairs within batch.

To perform domain-adversarial training, we construct a DANN architecture. The feature extractor includes the two convolutional layers followed by max pooling and ReLU. The label predictor in this case is replaced with *descriptor predictor* that includes one fully connected layer. The domain classifier includes two fully connected layers with 500 units in the intermediate representation ($x \rightarrow 500 \rightarrow 1$). For the verification loss function in the descriptor predictor we used Binomial Deviance loss, defined in [566] with similar parameters: $\alpha = 2$, $\beta = 0.5$, $c = 2$. The domain

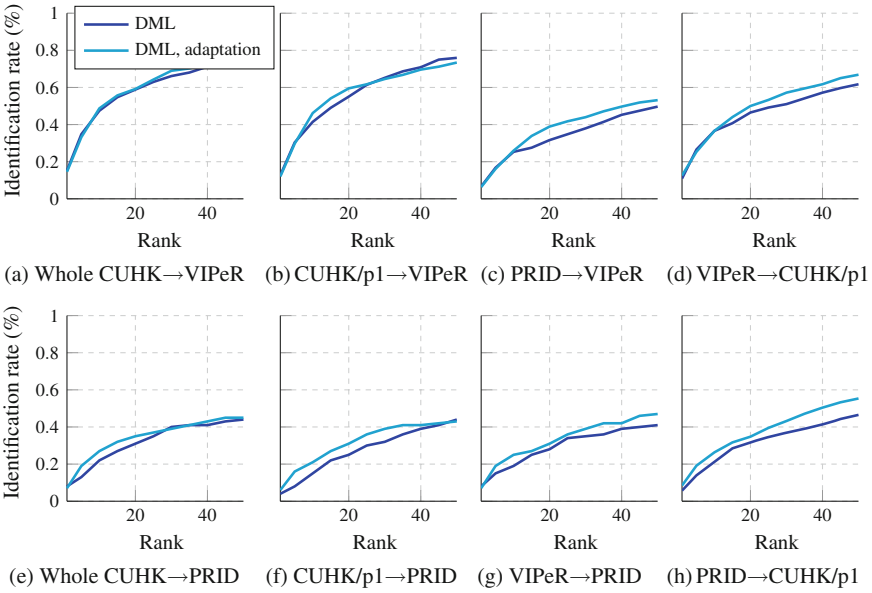


Fig. 10.8 Results on VIPeR, PRID and CUHK/p1 with and without domain-adversarial learning. Across the eight domain pairs DANN improves re-identification accuracy. For some domain pairs the improvement is considerable

classifier is trained with logistic loss as in Sect. 10.4.2. We used learning rate fixed to 0.001 and momentum of 0.9. The schedule of adaptation similar to the one described in Sect. 10.4.2 was used. We also inserted dropout layer with rate 0.5 after the concatenation of outputs of the second max pooling layer. 128-sized batches were used for source data and 128-sized batches for target data.

Results on Re-identification data sets Figure 10.8 shows results in the form of CMC curves for eight pairs of data sets. Depending on the hardness of the annotation problem we trained either for 50,000 iterations (CUHK/p1 → VIPeR, VIPeR → CUHK/p1, PRID → VIPeR) or for 20,000 iterations (the other five pairs). After the sufficient number of iterations, domain-adversarial training consistently improves the performance of re-identification. For the pairs that involve PRID data set, which is more dissimilar to the other two data sets, the improvement is considerable. Overall, this demonstrates the applicability of the domain-adversarial learning beyond classification problems.

10.5 Conclusion

The paper proposes a new approach to DA of feed-forward neural networks, which allows large-scale training based on large amount of annotated data in the source domain and large amount of unannotated data in the target domain. Similarly to many previous shallow and deep DA techniques, the adaptation is achieved through aligning the distributions of features across the two domains. However, unlike previous approaches, the alignment is accomplished through standard backpropagation training. Moreover, our approach is motivated and supported by the domain adaptation theory of [30, 31]. The main idea behind DANN is to enjoin the network hidden layer to learn a representation which is predictive of the source example labels, but uninformative about the domain of the input (source or target). We have shown that our approach is flexible and achieves state-of-the-art results on a variety of benchmark in DA.

A convenient aspect of our approach is that the DA component can be added to almost any neural network architecture that is trainable with backpropagation, allowing simple implementation within virtually any deep learning package through the introduction of a simple Gradient Reversal Layer. Toward this end, we have demonstrated experimentally that the approach is not confined to classification tasks but can be used in other feed-forward architectures, e.g., for descriptor learning for person re-identification.

Acknowledgements This work has been supported by National Science and Engineering Research Council (NSERC) Discovery grants 262067 and 0122405 as well, as the Russian Ministry of Science and Education grant RFMEFI57914X0071. We also thank the Graphics & Media Lab, Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University for providing the synthetic road signs data set. Most of the work of this paper was carried out while P. Germain was affiliated with Département d'informatique et de génie logiciel, Université Laval, Québec, Canada.