Beyond Sharing Weights for Deep Domain Adaptation

Artem Rozantsev, Mathieu Salzmann, *Member, IEEE*, and Pascal Fua, *Fellow, IEEE*

Abstract—The performance of a classifier trained on data coming from a specific domain typically degrades when applied to a related but different one. While annotating many samples from the new domain would address this issue, it is often too expensive or impractical. Domain Adaptation has therefore emerged as a solution to this problem; It leverages annotated data from a source domain, in which it is abundant, to train a classifier to operate in a target domain, in which it is either sparse or even lacking altogether. In this context, the recent trend consists of learning deep architectures whose weights are shared for both domains, which essentially amounts to learning domain invariant features. Here, we show that it is more effective to explicitly model the shift from one domain to the other. To this end, we introduce a two-stream architecture, where one operates in the source domain and the other in the target domain. In contrast to other approaches, the weights in corresponding layers are related but *not shared*. We demonstrate that this both yields higher accuracy than state-of-the-art methods on several object recognition and detection tasks and consistently outperforms networks with shared weights in both supervised and unsupervised settings.

Index Terms—Domain adaptation, deep learning

1 Introduction

Deep Neural Networks [1], [2] have emerged as powerful tools that outperform traditional Computer Vision algorithms in a wide variety of tasks, but only when sufficiently large amounts of training data are available. This is a severe limitation in fields in which obtaining such data is either difficult or expensive. For example, this work was initially motivated by our need to detect drones against complicated backgrounds with a view to developing anti-collision systems. Because the set of possible backgrounds is nearly infinite, creating an extensive enough training database of representative real images has proven very challenging.

Domain Adaptation [3] and Transfer Learning [4] have long been used to overcome this difficulty by making it possible to exploit what has been learned in one specific domain, for which enough training data is available, to effectively train classifiers in a related but different domain, where only very small amounts of additional annotations, or even none, can be acquired. Following the terminology of Domain Adaptation, we will refer to the domain in which enough annotated data is available as the *source domain* and the one with only limited amounts of such data, or none at all, as the *target domain*. In the drone case discussed above, the source domain can comprise synthetic images of drones, which can be created in arbitrary quantities, and the target domain a relatively small number of annotated real images. Again as in the domain adaptation literature, we will refer to this scenario as the *supervised* one. An even more difficult situation arises when there is absolutely no annotated data in the target domain. We will refer to this as the *unsupervised* scenario. In this paper, we will investigate both of these scenarios.

Recently, Domain Adaptation has been investigated in the context of Deep Learning with promising results. The simplest approach involves fine-tuning a Convolutional Neural Network (CNN) pre-trained on the source data using labeled target samples [5], [6]. This, however, results in overfitting when too little target data is available. Furthermore, it is not applicable in the unsupervised case. To overcome these limitations, other works have focused on using both source and target samples to learn a network where the features for both domains, or their distributions, are related via an additional loss term [7], [8], [9], [10]. To the best of our knowledge, all such methods use the same deep architecture with the same weights for both source and target domains. As such, they can be understood as attempts to make the features invariant to the domain shift, that is, the changes from one domain to the other.

In this paper, we show that imposing feature invariance is detrimental to discriminative power. To this end, we introduce the two-stream architecture depicted by Fig. 1. One stream operates on the source domain and the other on the target one. This makes it possible *not* to share the weights in some of the layers. To nonetheless encode the fact that both streams tackle the same recognition problem, albeit in different domains, we introduce a loss function that relates the corresponding weights in both layers. This loss function is lowest when the weights are linear transformations of each other. Furthermore, we introduce a criterion to automatically determine which layers should share their weights and which ones should not. In short, our approach explicitly models the domain shift by learning features adapted to each domain, but not fully independent, to account for the fact that both domains depict the same underlying problem.

We demonstrate that our approach is more effective than state-of-the-art weight-sharing schemes on standard

• *The authors are with the Computer Vision Laboratory, École Polytechnique Fédérale de Lausanne, Lausanne 1015, Switzerland.*
E-mail: {artem.rozantsev, mathieu.salzmann, pascal.fua}@epfl.ch.

Manuscript received 7 Mar. 2017; revised 27 Feb. 2018; accepted 3 Mar. 2018. Date of publication 7 Mar. 2018; date of current version 13 Mar. 2019.
(Corresponding author: Artem Rozantsev.)
Recommended for acceptance by D. Xu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPAMI.2018.2814042

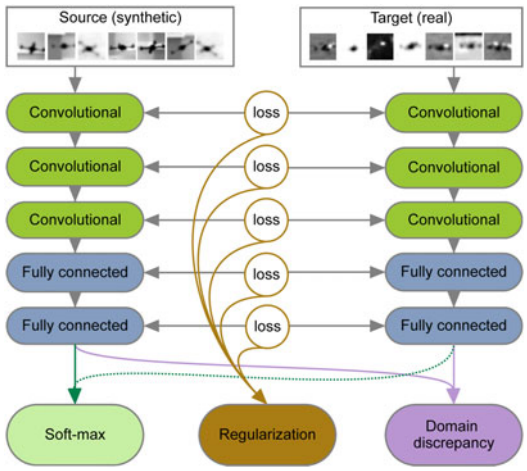


Fig. 1. *Our two-stream architecture.* One stream operates on the source data and the other on the target one. Their weights are *not* shared. Instead, we introduce loss functions that prevent corresponding weights from being too different from each other.

Domain Adaptation benchmarks for image recognition. We also show that it is well suited to leveraging synthetic data to increase the performance of a classifier on real images. Given that this is one of the easiest ways to provide the large amounts of training data that Deep Networks require, this scenario has become popular. Here, we treat the synthetic images as forming the source domain and the real images the target one. We then make use of our two-stream architecture to learn an effective model for the real data even though we have only few annotations for it. We demonstrate the effectiveness of our approach at leveraging synthetic data for both detection of Unmanned Aerial Vehicles (UAVs) and facial pose estimation. The first application involves classification and the second regression. Nevertheless, they both benefit from using synthetic data. We outperform the state-of-the-art methods in all these cases, and our experiments support our contention that specializing the network weights outperforms sharing them.

2 Related Work

In many practical applications, classifiers and regressors may have to operate on various kinds of related but visually different image data. The differences are often large enough for an algorithm that has been trained on one kind of images to perform poorly on another. Therefore, new training data has to be acquired and annotated to re-train it. Since this is typically expensive and time-consuming, there has long been a push to develop Domain Adaptation (DA) techniques that allow re-training with minimal amount of new data or even none. Here, we briefly review some recent trends, with a focus on Deep Learning based methods, which are the most related to our work.

2.1 Classical Domain Adaptation

A natural approach to Domain Adaptation is to modify a classifier trained on the source data using the available labeled target data. This was done, for example, using SVM [11], [12], Boosted Decision Trees [13] and other classifiers [14]. In the context of Deep Learning, fine-tuning [5], [6] essentially follows this pattern. In practice, however, when only a small amount of labeled target data is available, this often results in overfitting.

Another approach is to learn a metric between the source and target data, which can also be interpreted as a linear cross-domain transformation [15] or a non-linear one [16]. Instead of working on the samples directly, several methods involve representing each domain as one separate subspace [17], [18], [19], [20], [21]. A transformation can then be learned to align them [19]. Alternatively, one can interpolate between the source and target subspaces [17], [18], [20] or between their weighted versions [21]. In [22], this interpolation idea was extended to Deep Learning by training multiple unsupervised networks with increasing amounts of target data. The final representation of a sample was obtained by concatenating all intermediate ones. It is unclear, however, why this concatenation should be meaningful to classify a target sample.

Another way to handle the domain shift is to explicitly try making the source and target data distributions similar. While many metrics have been proposed to quantify the similarity between two distributions, the most widely used in the Domain Adaptation context is the Maximum Mean Discrepancy (MMD) [23]. The MMD has been used to reweight [24], [25] or select [26] source samples such that the resulting distribution becomes as similar as possible to the target one. An alternative is to learn a transformation of the data, typically both source and target, such that the resulting distributions are as similar as possible in MMD terms [27], [28], [29]. In [30], the MMD was used within a shallow neural network architecture. It was also used in conjunction with multiple kernel learning for visual event recognition [31] so that a pre-learned classifier can be adapted from source to target data by minimizing both the structural risk functional and the MMD between the domains. This was further extended to joint learning of both the kernel function and the classifier [32]. A different SVM-based technique was proposed in [33]. The algorithm finds a set of subdomains in the source data for training a collection of simple SVMs, the weighted sum of which is then used to classify the images in the target domain.

In [34], [35], the authors propose a different approach to task and knowledge transfer respectively. It starts from the observation that even different tasks and domains can leverage information encoded by the same features. Therefore the goal should be selecting such features to be shared across tasks, which is done by solving an optimization problem. While effective, the implementation, similarly to that of [30], [31], [32], [33], relies on hand-crafted features, such as SIFT [36] or SURF [37], for its initial image representation, which limits its effectiveness.

2.2 Deep Domain Adaptation

Recently, using Deep Networks to learn features has proven effective at increasing the accuracy of Domain Adaptation methods. In [38], it was shown that using DeCAF features instead of hand-crafted ones mitigates the domain shift effects even without performing any kind of adaptation. Y. H. Tsai et al. [39] further suggests learning Cross-Domain Landmarks that allow for heterogeneous domain adaptation, which, in essence, means that source and target data are represented by different features, such as SURF for the source images and DeCAF for the target ones. Y. H. Tsai et al. [39], however, requires some of the target labels available at training time, which limits its applicability to supervised Domain Adaptation. Furthermore, it is unclear why two domains of the same modality should be represented by different features.

S. Chopra et al. [40], X. Glorot et al. [41] propose performing domain adaptation within a Deep Learning framework instead of relying on a set of pretrained features, which is shown to boost performance. For example, in [40], a Siamese architecture was introduced to minimize the distance between pairs of source and target samples, which requires training labels available in the *target* domain thus making the method unsuitable for unsupervised Domain Adaptation. X. Glorot et al. [41] suggests using auto-encoders to learn the transformation between the source and target domains. This approach was further extended by [42] with bi-shifting auto-encoders, which first map the input samples to a common latent feature space, and then use two separate decoders to map from this latent feature space to either the source or the target domain.

In [7], [8], [43], the authors proposed instead to relate the source and target data representations learned by Deep Networks by minimizing the Maximum Mean Discrepancy between the feature representations of the source and target data. To this end, [7] proposed an additional term in the loss function that minimizes the MMD between the outputs of the last fully-connected layers. This was extended by [8] to having multiple independent MMD terms relating the outputs of several fully-connected layers. This, in turn, was further improved by [43], who proposed inserting a fusion layer between the intermediate feature representations regularized with the MMD loss. This fusion layer allows for full interactions across multiple feature layers, which in turn facilitates model selection and the learning process. A similar idea was followed in [44] to minimize the difference in second order feature statistics using the CORAL loss [45] and to relate higher order feature statistics in [46]. In [47], authors propose replacing the MMD term with a Joint Distribution Discrepancy loss, which aims to jointly minimize the distance between both feature distributions and the distributions of the classifier layer outputs across the domains.

In [9], [10], [48] a different loss term was introduced to encode an additional classifier predicting from which domain each sample comes. This was motivated by the fact that, if the learned features are domain-invariant, such domain classifier should exhibit very poor performance. M. Ghifary et al. [49] further suggested replacing the domain classifier by a reconstruction framework that operates on the target samples. Such an architecture favors learning features that are representative of the structure of the target domain.

Finally, another class of methods use pseudo labels to facilitate domain adaptation [50], [51]. In [50], the authors proposed to combine the classifier with a set of detectors, each one of which is trained in one-versus-all manner. This essentially means that the detector is using source images from one object category as positives and the rest of the classes from the source domain as negatives. During training, the images that are positively classified with high confidence by such a detector are added to the detector training set. This, however, was done based on the pretrained set of CNN features and therefore achieved only limited accuracy. By contrast, [51] proposed to enforce *cyclic consistency* in the model. This is based on the intuition that, if we can predict target data labels from the available source data and then predict back the source data labels, these predicted labels should be close to the ground truth ones. This, however, requires the initial classifier to be good enough to prevent relying on a majority of unreliable predicted target data labels.

All these Deep Learning approaches rely on the same architecture with the same weights for both the source and target domains. In essence, they attempt to reduce the impact of the domain shift by learning domain-invariant features. In practice, however, domain invariance might very well be detrimental to discriminative power. As discussed in the introduction, this is the hypothesis we set out to test in this work by introducing an approach that explicitly models the domain shift instead of attempting to enforce invariance to it. The work closest in spirit to ours is the method of [52], who proposed to use three networks to separately model the difference and the similarity between domains. This, however, results in a significant overhead and increases of the number of parameters that need to be learned, which makes the approach less attractive for larger architectures, such as the AlexNet [53]. Here, by contrast, we introduce a more compact, yet effective way of modeling both the similarity and the difference between the source and target data. We achieve this by introducing a two-stream architecture, where each stream serves to regularize the other one, thus avoiding overfitting to either source or target data. Note that [54] also relies on two separate CNN streams and minimizes the domain discrepancy with a special 'alignment cost layer'. However, the weights of these streams are completely independent, which increases the risk of overfitting if either of the domains feature a small number of examples. Our experiments show that both explicitly modeling the difference between the domains and connecting the weights of the corresponding layers in the two-stream architecture with a specific loss function contribute to a significant boost in accuracy over existing methods.

3 Our Approach

The core idea of our method is that, for a Deep Network to adapt to different domains, the weights should be related, yet different for each of the two domains. As shown empirically, this constitutes a major advantage of our method over the competing ones discussed in Section 2. To implement this idea, we therefore introduce a two-stream architecture, such as the one depicted by Fig. 1. The first stream operates on the source data, the second on the target one, and they are trained jointly. While we allow the weights of the corresponding layers to differ between the two streams, we prevent them from being too far from each other. Additionally we use the MMD between the learned source and target representations. This combination lets us encode the fact that, while different, the two domains are related.

More formally, let $\mathbf{X}^s = \{\mathbf{x}_i^s\}_{i=1}^{N^s}$ and $\mathbf{X}^t = \{\mathbf{x}_i^t\}_{i=1}^{N^t}$ be the sets of training images from the source and target domains, respectively, with $Y^s = \{y_i^s\}$ and $Y^t = \{y_i^t\}$ being the corresponding labels. To handle unsupervised target data as well, we assume, without loss of generality, that the target samples are ordered, such that only the first N_l^t ones have valid labels, where $N_l^t = 0$ in the unsupervised scenario. Furthermore, let $\Theta^s = \{\theta_j^s\}$ and $\Theta^t = \{\theta_j^t\}$ denote the parameters, that is, the weights and biases, of all the layers in the source and target streams, respectively. We train the network by minimizing a loss function of the form

$$L(\Theta^s, \Theta^t | \mathbf{X}^s, Y^s, \mathbf{X}^t, Y^t) = L_s + L_t + L_w + L_{DD}, \quad (1)$$

$$L_s = \frac{1}{N^s} \sum_{i=1}^{N^s} c(\Theta^s | \mathbf{x}_i^s, y_i^s), \quad (2)$$

$$L_t = \frac{1}{N_l^t} \sum_{i=1}^{N_l^t} c(\Theta^t | \mathbf{x}_i^t, y_i^t), \quad (3)$$

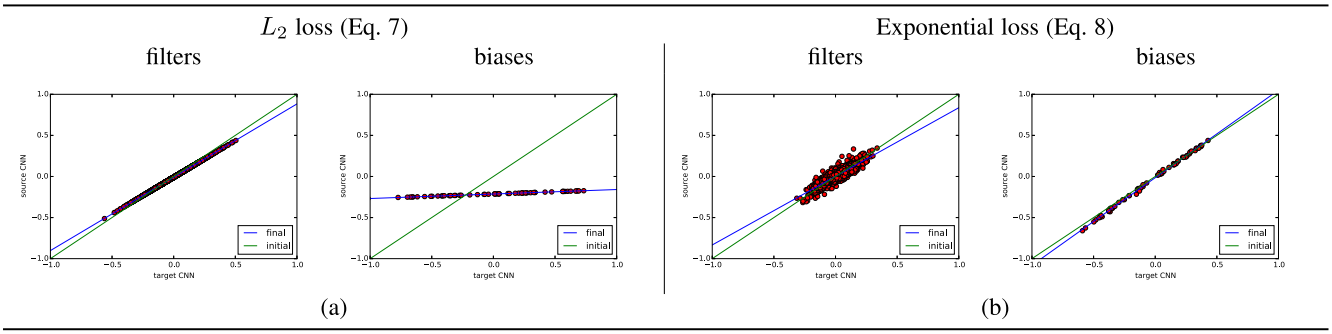


Fig. 2. Correlation between the parameters θ_1^s and θ_1^t of the first convolutional layer of our two-stream architecture, regularized by either the L_2 (**a**) or exponential (**b**) norm defined in Eq. (7) and Eq. (8), respectively. The red dots denote the correlation between the corresponding layer parameters. The green line shows the initial estimate for the linear transformation, and the blue line illustrates its final estimate after the training process. (Best viewed in color.).

$$L_w = \lambda_w \sum_{j \in \Omega} r_w(\theta_j^s, \theta_j^t), \tag{4}$$

$$L_{DD} = \lambda_u r_u(\Theta^s, \Theta^t | \mathbf{X}^s, \mathbf{X}^t), \tag{5}$$

where $c(\theta^{\cdot}|\mathbf{x}_i^{\cdot}, y_i^{\cdot})$ is a standard classification loss, such as the logistic loss or the hinge loss. Note that in the case of unsupervised domain adaptation L_t in Eq. (3) is equal to '0', as no information regarding the labels of the target data is available. Furthermore, $r_w(\cdot)$ and $r_u(\cdot)$ are the weight and unsupervised regularizers discussed below. The first one, L_w, represents the loss between corresponding layers of the two streams and thus only acts on the set Ω of indices of the layers whose parameters are not shared. This set is problem-dependent and, in practice, can be obtained *automatically* by comparing the MMD values for different configurations, as will be discussed in Section 4.1.2. The second regularizer, L_{DD}, encodes the domain discrepancy and favors similar distributions of the source and target data representations. These regularizers are weighted by coefficients λ_w and λ_u, respectively. In practice, we found our approach to be robust to the specific values of these coefficients and we set them to the same values in all our experiments.

3.1 Weight Regularizer

While our goal is to go beyond sharing the layer weights, we still believe that corresponding weights in the two streams should be related. This models the fact that the source and target domains are related, and prevents overfitting in the target stream, when only very few labeled samples are available. Our weight regularizer $r_w(\cdot)$ therefore represents the distance between the source and target weights in a particular layer. In principle, we could take it to directly act on the difference of those weights, and thus write

$$r_w(\theta_j^s, \theta_j^t) = \left\| \theta_j^s - \theta_j^t \right\|_2^2 , \; j \in \Omega \; . \tag{6}$$

This, however, would not truly attempt to model the domain shift, for instance to account for different means and ranges of values in the two types of data. To better model the shift and introduce more flexibility in our model, we therefore propose not to penalize linear transformations between the source and target weights. We then write our regularizer either by relying on the L_2 norm as

$$r_w(\theta_j^s, \theta_j^t) = \left\| a_j \theta_j^s + b_j - \theta_j^t \right\|_2^2 , \tag{7}$$

or in an exponential form as

$$r_w(\theta_j^s, \theta_j^t) = \exp\left(\left\| a_j \theta_j^s + b_j - \theta_j^t \right\|^2 \right) - 1 \; . \tag{8}$$

In both cases, a_j and b_j are scalar parameters that are different for each layer $j \in \Omega$ and learned at training time along with all other network parameters. While simple, this parameterization can account, for example, for global illumination changes in the first layer of the network. As shown in the results section, we found empirically that the exponential version gives better results. We have tried replacing the simple linear transformation of Eqs. (7) and (8) by more sophisticated ones, such as quadratic or piecewise linear ones. This, however, did not yield any performance improvement.

Fig. 2 depicts the correlation between the θ_1^s and θ_1^t weights in the first convolutional layer of our two-stream architecture, regularized by the term of either Eq. (7) or Eq. (8). The plot of Fig. 2a shows that, when we apply the L_2 norm, the parameters of the corresponding layers are very close to being a linear transformation of each other. The plot of Fig. 2b shows that the exponential loss of Eq. (8) gives more freedom to θ_1^s and θ_1^t, while still keeping their values close to being a linear transformation of each other.

3.2 Unsupervised Regularizer

In addition to regularizing the weights of corresponding layers in the two streams, we also aim at learning a final representation, that is, the features before the classifier layer, which is domain invariant. Here, we study two different versions of the function $r_u(\cdot)$ used to compute the domain discrepancy regularizer L_{DD} of Eq. (5), which embody two different ways to encode invariance. The first version aims to minimize the distance between the distributions of the source and target representations by minimizing the Maximum Mean Discrepancy [7], [8], [23], [55] while the second relies on a domain classifier [9]. In the results section, we will use the same one as the baselines we compare ourselves against to ensure fairness.

3.2.1 Maximum Mean Discrepancy

As the name suggests, given two sets of data, the MMD measures the distance between the mean of the two sets after mapping each sample to a Reproducing Kernel Hilbert Space (RKHS). In our context, let $\mathbf{f}_i^s = \mathbf{f}_i^s(\Theta^s, \mathbf{x}_i^s)$ be the feature representation at the last layer of the source stream, and $\mathbf{f}_j^t = \mathbf{f}_j^t(\Theta^t, \mathbf{x}_j^t)$ of the target stream. The squared MMD between the source and target domains can be expressed as

$$\mathrm{MMD}^2(\{\mathbf{f}_i^s\},\{\mathbf{f}_j^t\}) = \left\| \sum_{i=1}^{N^s} \frac{\phi(\mathbf{f}_i^s)}{N^s} - \sum_{j=1}^{N^t} \frac{\phi(\mathbf{f}_j^t)}{N^t} \right\|^2 , \quad (9)$$

where $\phi(\cdot)$ denotes the mapping to RKHS. In practice, this mapping is typically unknown. Expanding Eq. (9) and using the kernel trick to replace inner products by kernel values lets us rewrite the squared MMD, and thus our regularizer as

$$\begin{aligned} r_u(\Theta^s,\Theta^t|\mathbf{X}^s,\mathbf{X}^t) = &\sum_{i,i'} \frac{k(\mathbf{f}_i^s,\mathbf{f}_{i'}^s)}{(N^s)^2} \\ &- 2\sum_{i,j} \frac{k(\mathbf{f}_i^s,\mathbf{f}_j^t)}{N^s N^t} + \sum_{j,j'} \frac{k(\mathbf{f}_j^t,\mathbf{f}_{j'}^t)}{(N^t)^2} , \end{aligned} \quad (10)$$

where the dependency on the network parameters comes via the $\mathbf{f}_i$s, and where $k(\cdot,\cdot)$ is a kernel function. In practice, we make use of the standard RBF kernel $k(u,v) = \exp\left(-\|u-v\|^2/\sigma\right)$, with bandwidth σ. In our experiments, we found our approach to be insensitive to the exact value of σ. We therefore systematically set it to 1. This is not surprising because the learned features can automatically adapt to a given σ value. We investigate this more thoroughly in Section 4.1.3.

3.2.2 Adversarial Domain Confusion

As discussed in Section 2, another way of decreasing the domain discrepancy is to create a classifier that will try to predict from which domain the sample is coming. The intuition behind this is that, if the final feature representation is truly domain invariant, such a classifier will have very low accuracy.

One way to tackle this problem was proposed by [9], where the authors suggested introducing a small neural network $\hat{y} : \hat{y}(\mathbf{f}) = \phi(\theta^{DC},\mathbf{f})$, where θ^{DC} and $\mathbf{f}$ are the network parameters and the final feature representation of an image, respectively. During training θ^{DC} is learned by minimizing the cross entropy Loss:

$$L_{DC}(y_n,\mathbf{f}) = -\frac{1}{N}\sum_{n=1}^{N}[y_n \log(\hat{y}_n) + (1-y_n)\log(1-\hat{y}_n)], \quad (11)$$

where N is the number of samples in the batch and $y_n : y_n \in \{0,1\}$ is the domain label, with '0' corresponding to *source* data and '1' – to the *target*.

To model the fact that domain invariance corresponds to a poor performance of the domain classifier, we write our domain regularizer as

$$r_u(\Theta^s,\Theta^t|\mathbf{X}^s,\mathbf{X}^t) = L_{DC}(1-y_n,\mathbf{f}). \quad (12)$$

As in [9], every training iteration comprises the following two steps. First, the parameters of the domain classification network θ^{DC} are updated by minimizing $L_{DC}(y_n,\mathbf{f})$, as

$$\theta_*^{DC} \leftarrow \arg\min_{\theta^{DC}} L_{DC}(y_n,\mathbf{f}) , \quad (13)$$

where $\mathbf{f}$ is the final feature representation. It is computed from the source and target images using the fixed parameters Θ^s and Θ^t of the source and target streams, respectively. The resulting parameters of the domain classification network θ_*^{DC} then remain fixed for the second step

$$\{\Theta_*^s,\Theta_*^t\} \leftarrow \arg\min_{\Theta^s,\Theta^t} L_s + L_t + L_w + L_{DC}(1-y_n,\mathbf{f}) , \quad (14)$$

which aims at learning source and target stream parameters Θ^s and Θ^t by minimizing the overall cost of Eq. (1). As can be seen from its last term, Eq. (14) maximizes the confusion between the domains, which is important in our case, as we use exactly the same classification layer for both source and target images. The optimization procedure alternates between these two steps until convergence.

3.3 Training

To learn the model parameters, we first pre-train the source stream using the source data only. We then simultaneously optimize the weights of both streams according to the loss of Eqs. (2), (3), (4), and (5) using both source and target data, with the target stream weights initialized from the pre-trained source weights. Note that this also requires initializing the linear transformation parameters of each layer, a_j and b_j for all $j \in \Omega$. We initialize these values to $a_j = 1$ and $b_j = 0$, thus encoding the identity transformation. All parameters are then learned jointly via backpropagation using Adam [56] with a learning rate of 0.0005. To make the comparison to the baseline approaches [7], [9] on the *Office* dataset fair, in this experiment, we have used Stochastic Gradient Descent [57], [58] as an optimization technique with 0.9 momentum and a learning rate (lr) annealing defined as

$$\mathrm{lr}_p = \frac{\mathrm{lr}_0}{(1+\alpha p)^\beta} , \quad (15)$$

with p linearly increasing from 0 to 1 at every iteration, $\mathrm{lr}_0 = 0.01, \alpha = 10, \beta = 0.75$. Note that we rely on mini-batches, and thus in practice compute all the terms of our loss over these mini-batches rather than over the entire source and target datasets.

3.4 Architectures

Depending on the task, we use different network architectures, to provide a fair comparison with the baselines. For example, for the UAV detection task we used the architecture depicted in Fig. 1; for the *Office* benchmark, we adopted the AlexNet [53] architecture, as was done in [7], [9]; and for digit classification, we relied on the standard network structure of [59] for each stream.

We found our approach to be insensitive to the specific values of λ_u and λ_w and we will demonstrate this thoroughly for the UAV detection task in Section 4.1.3. For a fair comparison with the competing method of [9] on the *Office* dataset, we used the same parameters as they did, that is, $\lambda_u = 0.1$. We take the weight λ_w to be the 1 for all the stream-specific layers. A better accuracy could be potentially be achieved by tuning λ_w for each stream-specific layer independently but we did not do it to preserve the framework genericity.

4 Experimental Results

In this section, we demonstrate the potential of our approach in both the supervised and unsupervised scenarios using different network architectures. We first thoroughly evaluate our method for the drone detection task. We then demonstrate that it generalizes well to other classification problems by testing it on the *Office*, *MNIST+USPS*

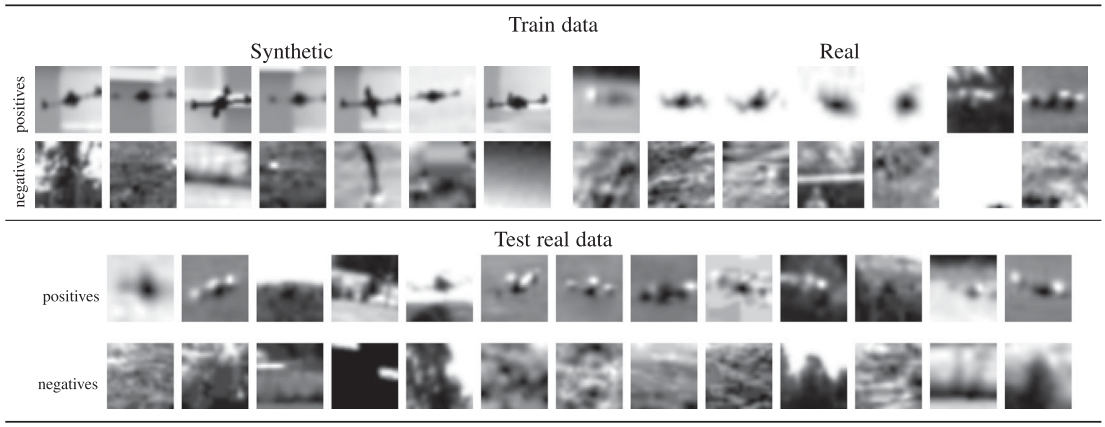


Fig. 3. Our UAV dataset. [Top] Synthetic and real training examples. [Bottom] Real samples from the test dataset.

TABLE 1
Statistics of Our Two UAV Datasets

Dataset	Training			Validation		Testing	
	Pos		Neg	Pos	Neg	Pos	Neg
Data type:	(Real)	(Synthetic)	(Real)	(Real)	(Real)	(Real)	(Real)
UAV-200 (full)	200	32800	190000	500	1500	3100	135000
UAV-200 (small)	200	1640	9500	500	1500	3100	6750

Note that UAV-200 (small) *is more balanced than* UAV-200 (full).

and *MNIST+SVHN* datasets. Finally, to show that our approach can be naturally applied to regression problems, such as estimating the position of facial landmarks.

4.1 Leveraging Synthetic Data for Drone Detection

Due to the lack of large publicly available datasets, UAV detection is a perfect example of a problem where training videos are scarce and do not cover a wide enough range of possible shapes, poses, lighting conditions, and backgrounds against which drones can be seen. However, it is relatively easy to generate large amounts of synthetic examples, which can be used to supplement a small number of real images and increase detection accuracy [60]. We show here that our approach allows us to exploit these synthetic images more effectively than other state-of-the-art Domain Adaptation techniques.

4.1.1 Dataset and Evaluation Setup

We used the approach of [60] to create a large set of synthetic examples. Fig. 3 depicts sample images from the real and synthetic dataset that we used for training and testing. In our experiments, we treat the synthetic images as source samples and the real images as target ones.

We report results using two versions of this dataset, which we refer to as *UAV-200 (small)* and *UAV-200 (full)*. Their sizes are given in Table 1. They only differ in the number of synthetic and negative samples used for training and testing. The ratio of positive to negative samples in the first dataset is better balanced than in the second one. For *UAV-200 (small)*, we therefore express our results in terms of *accuracy*, which is a standard metric that is commonly used in Domain Adaptation.

In real detection tasks, however, training datasets are typically quite unbalanced, since one usually encounters many negative windows for each positive example. *UAV-200 (full)* reflects this more realistic scenario, in which the accuracy metric is poorly-suited. For this dataset, we therefore compare various approaches in terms of *precision-recall*. Precision corresponds to the number of true positives detected by the algorithm divided by the total number of detections. Recall is the number of true positives divided by the number of test examples labeled as positive. Additionally, we report the *Average Precision* (AP), which is computed as $\int_0^1 p(r)dr$, where p and r denote precision and recall, respectively.

For this experiment, we follow the supervised Domain Adaptation scenario. In other words, training data is available with labels for both source and target domains.

4.1.2 Network Design

Our network consists of two streams, one for the source data and one for the target data, as illustrated by Fig. 1. Each stream is a CNN that comprises three convolutional and max-pooling layers, followed by two fully-connected ones. The classification layer encodes a hinge loss, which was shown to outperform the logistic loss in practice for some tasks [61], [62].

As discussed above, some pairs of layers in our two-stream architecture may share their weights while others do not, and we must decide upon an optimal arrangement. To this end, we trained one model for every possible combination. In all cases, we implemented our regularizer using one of the following:

- the simple L_2 loss of of Eq. (6);
- its relaxed version of Eq. (7), which allows for a linear transformation between the parameters;
- the exponential loss of Eq. 8, which also allows for a linear transformation between the parameters.

After training, we computed the MMD^2 value between the output of both streams for each configuration. Since we

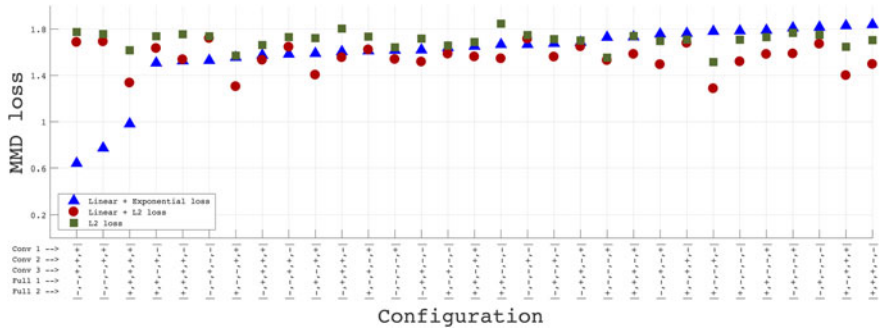


Fig. 4. Evaluation of the best network architecture. The y-axis corresponds to the MMD^2 loss between the outputs of the corresponding streams that operate on real and synthetic data, respectively. The x-axis denotes the network configuration, where a '$+$' sign indicates that the corresponding network layers are regularized with a loss function and a '$-$' sign that the weights are shared for the corresponding layers. (Best seen in color).

use a common classification layer, the MMD^2 value ought to be small when our architecture accounts well for the domain shift [7]. We therefore choose the configuration that yields the smallest MMD^2 value. Note that increasing the number of layers that are stream-specific does not necessarily yield the lowest MMD^2 score. This would only happen if the streams were completely independent of each other and the weight of the stream that operates on the target data were influenced only by the costs of Eqs. (3), (5). In practice, by adding the weight regularizer of Eq. (4), we constrain the parameter learning process. As a result, finding the architecture with the lowest MMD^2 score does not necessarily result in all layers not sharing their weights.

In fact, it usually does not. To show this in a specific case, we plot the MMD^2 values for different architectures and weight regularizers in Fig. 4. The x-axis in the plot illustrates different architectures, sorted from lowest to highest MMD^2 value, with the $+$ and $-$ signs indicating whether the weights are stream-specific or shared. The architecture with the first three convolutional layers being stream specific, with parameters connected via the exponential loss yields the lowest MMD^2 score between the domains. Our intuition is that, even though the synthetic and real images feature the same objects, they differ in appearance, which is mostly encoded by the first network layers. Thus, allowing the weights to differ in these layers yields good adaptive behavior, as will be demonstrated in Section 4.1.4.

To further illustrate that increasing the set of layers that do not share their weights does not necessarily yield the lowest MMD^2 score, we performed the following experiment. As can be see in Fig. 4, the model that minimizes the MMD^2 score has three stream-specific layers followed by two that are shared. We therefore restarted the learning process with the learned weights but with the fourth layer not being shared anymore, like the first three. We then performed 6000 additional Adam [56] iterations to minimize the total cost function of Eq. (1). Recall from Section 3 that it is the weighted sum of the classification losses L_s and L_t of Eqs. (2) and (3), the weight regularizer of Eq. (4), and the domain discrepancy loss L_{DD} of Eq. (5). In Fig. 5a, we plot the the evolution of the different loss terms. As expected, the overall cost function, shown in blue, decreases slightly, as do the regularization and classification losses drawn in orange and green, respectively. However, the domain discrepancy loss *increases*. In other words, the fact that the sum of the losses decreases does not guarantee that all its individual components do so. In this specific example, by untying the weights of the fourth layer and thus making the system more flexible, we enabled the optimizer to find stream-specific parameters that are closer to being scaled and shifted versions of each other, hence the substantial decrease in L_w at the cost of a smaller increase in L_{DD}. This can be seen in Fig. 5b where we plot the contribution of individual layers to L_w. As the weights of the fourth layer become increasingly different, it starts contributing to the regularization loss. However this is more than offset by a larger drop in the contribution of the first layer, resulting in an overall decrease in L_w.

4.1.3 Hyperparameter Selection

In addition to the choice of which layers should share their weights or be stream-specific, our framework depends on a set of hyperparameters:

- λ_w is defined in Eq. (4) and regulates the impact of the weight regularizer;
- λ_u regulates the influence of the Domain Discrepancy term in the loss function and is defined in Eq. (5);
- σ is the bandwidth of the RBF kernel in the MMD formulation of Eq. (10).

We first evaluate the robustness of our approach to the choice of λ_w on the UAV-200 (full) dataset. To this end, as discussed in Section 3.3, we started by pretraining the model on the synthetic data and then fine-tuned it by optimizing the loss function of Eq. (1) with $\lambda_w \in [0.001 : 5]$. We then computed the Average Precision on the validation set and provide the results in Fig. 6[Top]. While taking λ_w to be either very small or very large results in a slight performance decrease, the accuracy remains stable within the relatively large interval $[0.01 : 2]$. Therefore, in all further experiments we set λ_w to 1.

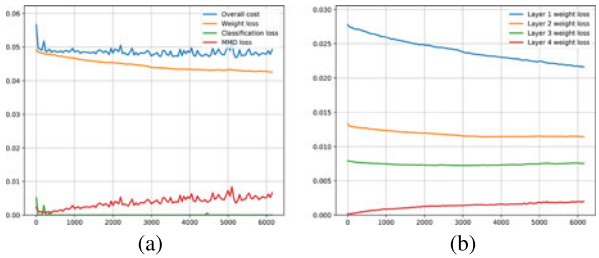


Fig. 5. Several iterations of Adam [56] to minimize the objective function, defined in Eq. 1. This experiment was carried out on the *UAV-200* benchmark with a network architecture whose first 4 layers are *not* shared and initialized from the pre-trained optimal configuration, selected by the MMD criterion, whose 3 first layers only are *not* shared. [Left] Behavior of the different terms of the cost function, defined in Eq. 1. [Right] Contribution of different layers to the L_w term of the cost function. (Best seen in color).

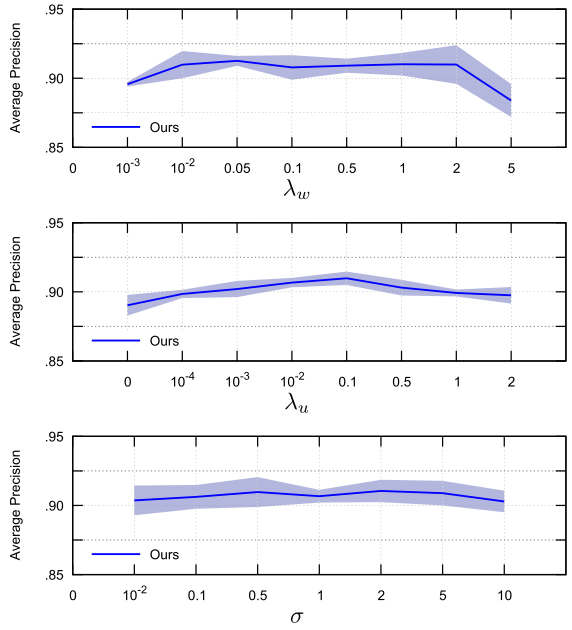


Fig. 6. Average precision of our approach, tested on the validation dataset for different values of λ_w [Top], λ_u [Middle] and σ [Bottom]. We report the mean and standard deviation of the AP score across 5 runs of our method. In all three cases, the performance remains stable across a wide interval.

TABLE 2
Comparison to Other Domain Adaptation Techniques on the *UAV-200 (small)* Dataset

	Accuracy
ITML [15]	0.60
ARC-t assymetric [16]	0.55
ARC-t symmetric [16]	0.60
HFA [63]	0.75
DDC [7]	0.89
Ours	**0.92**

We then analyze our method's sensitivity to changes in the value of λ_u. As before, we start from a pretrained model and fine-tune it by minimizing Eq. (1) with $\lambda_u \in [0:2]$. Fig. 6[Middle] depicts the results. As expected, when λ_u is too small, the influence of the domain discrepancy term is insufficient and the precision decreases. Values in the range $[0.01, 0.5]$ yield the best results, which agrees with the findings of [7], suggesting using $\lambda_u = 0.25$. This is therefore the value we will use from now on.

Finally, we quantify the influence of the bandwidth parameter of the RBF kernel σ, which is used to compute the MMD in Eq. (4). We again fine-tune the pre-trained model, this time with $\sigma \in [0.01:10]$. We then compute the AP score on the validation dataset and plot the results in Fig. 6[Bottom]. As before, even though the accuracy decreases slightly when σ becomes too small, the overall performance remains stable in the whole interval $[0.1:5]$, and we set σ to 1 in the rest of our experiments.

4.1.4 Evaluation

We first compare our approach to other Domain Adaptation methods on *UAV-200 (small)*. As can be seen in Table 2, it significantly outperforms many state-of-the-art baselines in terms of accuracy. In particular, we believe that outperforming DDC [7] goes a long way towards validating our hypothesis that modeling the domain shift is more effective than trying to be invariant to it. This is because, as discussed in Section 2, DDC relies on minimizing the MMD loss between the learned source and target representations much as we do, but uses a single stream for both source and target data. In other words, except for the non-shared weights, it is the method closest to ours. Note that the original DDC paper used a slightly different network architecture than ours, which was required due to the larger images it used as input. To avoid any bias, we therefore modified this architecture so that it matches ours.

TABLE 3
Comparison of Our Method Against Several Baselines on the *UAV-200 (full)* Dataset

	AP (Average Precision)
*CNN (trained on Synthetic only (**S**))*	.314
*CNN (trained on Real only (**R**))*	.575
*CNN (pre-trained on **S**and fine-tuned on **R**):*	
Loss: L_t	.612
Loss: $L_t + L_w$ (with fixed source CNN)	.655
*CNN (pre-trained on **S**and fine-tuned on **R**and **S**:)*	
Loss: $L_s + L_t$ [60]	.569
DDC [7] *(pre-trained on **S**and fine-tuned on **R**and **S**)*	.664±.002
*Our approach (pre-trained on **S**and fine-tuned on **R**and **S**)*	
Loss: $L_s + L_t + L_w$	.673±.001
Loss: $L_s + L_t + L_{MMD}$	.700±.002
Loss: $L_s + L_t + L_w + L_{MMD}$	.722±.003
Loss: $L_s + L_t + L_w + L_{DC}$	**.732±.003**

As discussed in Section 3, the terms L_s, L_t, L_w, L_{MMD} and L_{DC} correspond to the elements of the loss function, defined in Section 3 and Eqs. (2), (3), (4), (5).

We then turn to the complete dataset *UAV-200 (full)*. In this case, the baselines whose implementations only output accuracy values become less relevant because it is not a good metric for unbalanced data. We therefore compare our approach against DDC [7], which we found to be our strongest competitor in the previous experiment, and against the Deep Learning approach of [60], which also tackles the drone detection problem. We also turn on and off some of our loss terms to quantify their influence on the final performance. We give the results in Table 3. In short, all loss terms contribute to improving the AP of our approach, which itself outperforms all the baselines by large margins. More specifically, we get a 10 percent boost over DDC and a 20 percent boost over using real data only. By contrast, simply using real and synthetic examples together, as was done in [60], does not yield significant improvements. Note that dropping the terms linking the weights in corresponding layers while still minimizing the MMD loss (Loss: $L_s + L_t + L_{MMD}$) performs worse than using our full loss function. The former essentially corresponds to the method proposed in [54], where the authors suggested using a two-stream architecture for domain adaptation, allowing the weights of the corresponding layers in both streams to vary independently of each other. We attribute this decrease in accuracy to overfitting of the target stream to the small amount of available examples.

In Table 4, we report the behavior of our approach when using the different regularizers introduced in Section 3.1 on the test set of the *UAV-200 (full)* in "extreme" cases, that is, when either all corresponding layers share their weights or none do. The configurations that do *not* share weights systematically outperform those that do, which illustrates the

TABLE 4
Comparison of Different Extreme Cases of Sharing and Not-Sharing the Weights on the Test Images of the UAV-200 Dataset

	AP (Average Precision)
All weights shared ≡ DDC [7]	0.664
Simple L_2 loss (Eq. (6))	
ALL-SS	0.670
SELECTED	0.682
Linear + L_2 loss (Eq. (7))	
ALL-SS	0.689
SELECTED	0.691
Linear + Exponential loss (Eq. (8))	
ALL-SS	0.696
SELECTED	**0.722**

Here ALL-SS corresponds to not *sharing parameters in all layers. Furthermore, SELECTED illustrates the performance of the network with the configuration chosen by the* MMD^2 *criterion.*

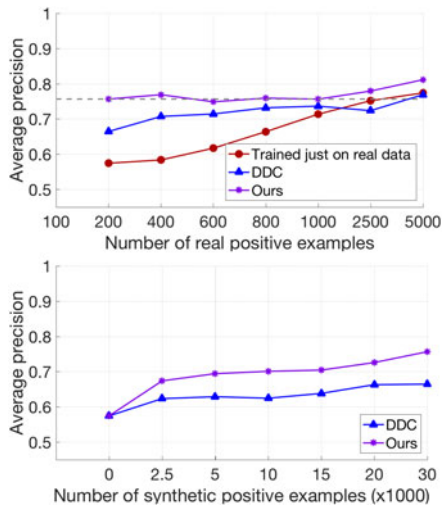


Fig. 7. **Influence of the ratio of synthetic to real data.** [TOP] AP of our approach (violet stars), DDC (blue triangles), and training using real data only (red circles) as a function of the number of real samples used given a constant number of synthetic ones. [BOTTOM] AP of our approach (violet stars) and DDC (blue triangles) as a function of the number of synthetic examples used given a small and constant number of real ones. (Best seen in color).

power of our approach even without a sophisticated method to choosing when to share or not to share weights. Furthermore, the regularizer of Eq. (8) outperforms the others by a large margin.

4.1.5 Influence of the Number of Samples

Using synthetic data in the UAV detection scenario is motivated by the fact that it is hard and time consuming to collect large amounts of real data. We therefore evaluate the influence of the ratio of synthetic to real data. To this end, we first fix the number of synthetic samples to 32800, as in *UAV-200 (full)*, and vary the amount of real positive samples from 200 to 5000. The results of this experiment are reported in Fig. 7 [TOP], where we again compare our approach to DDC [7] and to the same CNN model trained on the real samples only. Our model always outperforms the one trained on real data only. This suggests that it remains capable of leveraging the synthetic data, even though more real data is available, which is not the case for DDC. More importantly, looking at the leftmost point on our curve shows that, with only 200 real samples, our approach performs similarly to, and even slightly better than, a single-stream model trained using 2500 real samples. In other words, one only needs to collect 5-10 percent of labeled training data to obtain good results with our approach, which, we believe, can have a significant impact in practical applications.

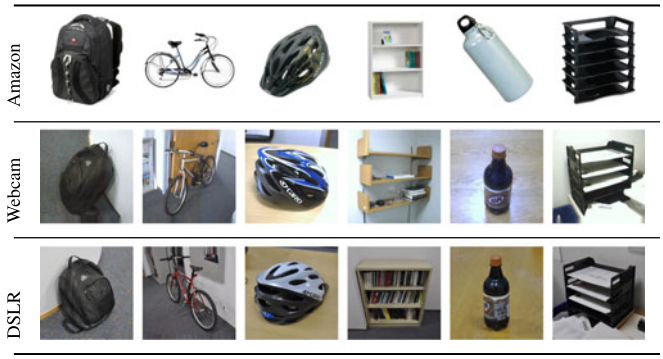


Fig. 8. Some examples from three domains in the Office dataset.

Fig. 7[BOTTOM] depicts the results of an experiment where we fixed the number of real samples to 200 and increased the number of synthetic ones from 0 to 32800. Note that the AP of our approach steadily increases as more synthetic data is used. DDC also improves, but we systematically outperform it except when we use no synthetic samples, in which case both approaches reduce to a single-stream CNN trained on real data only.

4.2 Domain Adaptation on Office

To demonstrate that our approach extends to the unsupervised case, we further evaluate it on the *Office* dataset, which is a standard domain adaptation benchmark for image classification. Following standard practice, we express our results in terms of accuracy. The *Office* dataset [15] comprises three different sets of images (Amazon, DSLR, Webcam) featuring 31 classes of objects. Fig. 8 depicts some images from the three different domains.

4.2.1 Unsupervised Domain Adaptation

For this experiment, we used the "fully-transductive" evaluation protocol proposed in [15], which means using all the available information on the source domain and having no labels at all for the target domain. For the comparison with [9] to be fair, we also report results obtained using the domain classifier, as described in Section 3.2.2. We used the same architecture as in [9] for this classifier.

Fig. 9[TOP] illustrates the network architecture we used for this experiment. Each stream corresponds to the standard AlexNet CNN [53]. As in [7], [9], we start with the model pretrained on ImageNet and fine tune it. However, instead of forcing the weights of both streams to be shared, we allow them to deviate as discussed in Section 3. To identify which layers should share their weights and which ones should *not*, we used the MMD-based criterion introduced in Section 4.1.2. In Fig. 9[BOTTOM], we plot the MMD^2 value as a function of the configuration on the *Amazon* → *Webcam* scenario, as we did for the drones in Fig. 4. In this case, not sharing the last two fully-connected layers achieves the lowest MMD^2 value, and this is the configuration we use for our experiments on this dataset. As the number of layers that have stream-specific

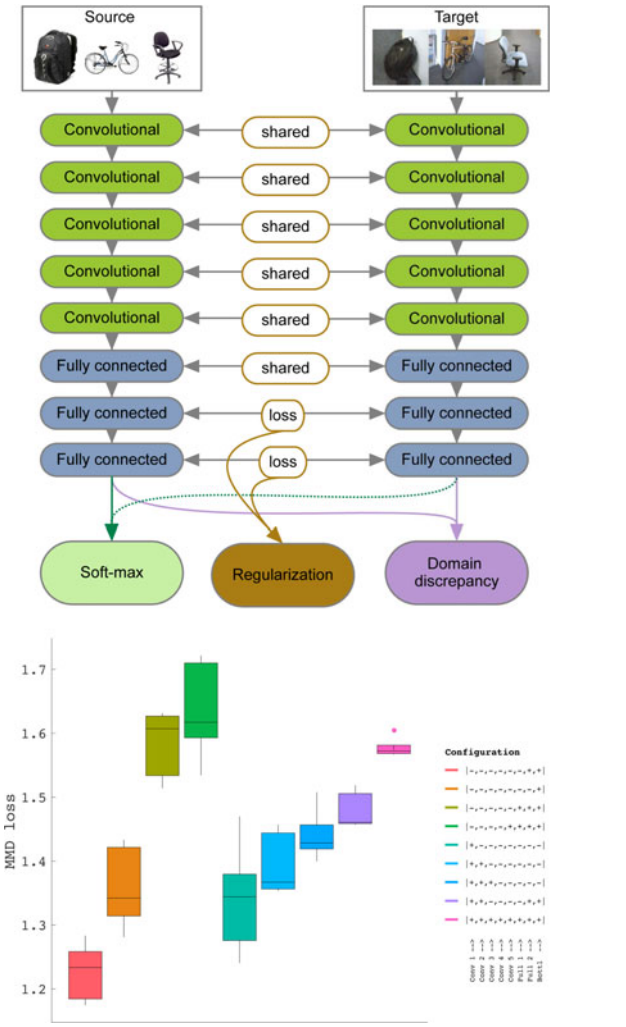


Fig. 9. Office dataset. [Top] The network architecture that proved to be the best according to our MMD-based criterion. [Bottom] The y-axis corresponds to the MMD^2 loss between the outputs of the corresponding streams that operate on *Amazon* and *Webcam*, respectively. Different configurations are depicted by different colors.

parameters, the MMD^2 value drops. This suggests that the streams become too flexible and begin overfitting to source and/or target data. This issue could be addressed by increasing the strength λ_w of the weight regularizer. However, this is not necessary because sharing the weights of all layers, except the two last ones, sufficiently constrains the problem. On the other hand, in the case when just the last fully-connected layer has stream-specific parameters, the system is too rigid and fails to efficiently model the domain shift. This leads to a slight increase in the MMD^2 value.

In Table 5, we compare our approach against other Domain Adaptation techniques. It outperforms them on all the pairs, except for *DSLR→Amazon*, where it achieves second best performance. More importantly, our method outperforms GRL [9], which confirms that allowing the weights not to be shared increases accuracy. The results of the baseline approaches are taken from the respective papers.

4.2.2 Supervised Domain Adaptation

For this experiment, we used the evaluation protocol proposed in [15], which corresponds to the supervised scenario. This involves using a fraction of the available labeled samples in the target domain for training purposes along with all the labeled data from the source domain. As in [15], we used the labels of 20 randomly sampled images for each class for the Amazon domain and 8 labeled images per class for the DLSR and Webcam domains, when used as source datasets. For the target domain, we only used 3 randomly selected labeled images per class. The rest of the dataset was then used as unlabeled data for the calculation of the MMD loss of Eq. (5).

In Table 6, we compare our approach against other Domain Adaptation techniques on the three commonly-reported source/target pairs. It outperforms them on all three. More importantly, the comparison against DDC confirms that allowing the weights not to be shared increases accuracy.

4.3 Unsupervised Domain Adaptation on MNIST-USPS

The *MNIST* [59] and *USPS* [67] datasets for digit classification both feature 10 different classes of images corresponding to the 10 digits. They have recently been employed for the task of Domain Adaptation [68].

For this experiment, we used the evaluation protocol of [68], which involves randomly selecting of 2000 images from *MNIST* and 1800 images from *USPS* and using them interchangeably as source and target domains. As in [68],

TABLE 5
Comparison Against Other Domain Adaptation Techniques on the Office Benchmark

	Accuracy						
	A → W	W → A	A → D	D → A	D → W	W → D	Average
TCA [64] (`AlexNet-fc7 features`)	.610	.509	.608	.516	.932	.952	.688
GFK [18] (`AlexNet-fc7 features`)	.604	.481	.606	.524	.956	.950	.687
DLID [22]	.519	–	–	–	.782	.899	–
LapCNN [65]	.604 ± .003	.482 ± .005	.631 ± .006	.516 ± .004	.947 ± .005	.991 ± .002	.695
CNN [53]	.616 ± .005	.498 ± .004	.638 ± .005	.511 ± .006	.954 ± .003	.990 ± .002	.701
DDC [7]	.618 ± .004	.522 ± .004	.644 ± .003	.521 ± .008	.950 ± .005	.985 ± .004	.706
D-CORAL [44]	.664 ± .004	.515 ± .003	.668 ± .006	.528 ± .002	.957 ± .003	.992 ± .001	.721
DAN [8]	.685 ± .005	.531 ± .005	.670 ± .004	.540 ± .005	.960 ± .003	.990 ± .003	.729
DRCN [49]	.687 ± .003	.549 ± .005	.668 ± .005	.560 ± .005	.964 ± .003	.990 ± .002	.736
RTN [43]	.733 ± .003	.510 ± .001	.710 ± .002	.505 ± .003	.968 ± .002	.996 ± .001	.737
GRL [9]	.730 ± .005	.542 ± .002	.753 ± .004	.527 ± .002	.964 ± .003	.992 ± .003	.751
JAN-xxy [47]	.749 ± .003	.550 ± .004	.718 ± .003	**.583 ± .003**	.966 ± .002	.995 ± .002	.760
Ours	**.758 ± .004**	**.576 ± .003**	**.755 ± .002**	.557 ± .004	**.967 ± .002**	**.996 ± .002**	**.768**

We evaluate on all 31 categories and report the mean and standard deviation over 5 runs, following the "fully-transductive" evaluation protocol of [15].

TABLE 6
Comparison to Other Domain Adaptation Techniques on the Office Standard Benchmark

	Accuracy			
	A → W	D → W	W → D	Average
GFK [18]	.464 ± .005	.613 ± .004	.663 ± .004	.530
SA [19]	.450	.648	.699	.599
DA-NBNN [66]	.528 ± .037	.766 ± .017	.762 ± .025	.685
DLID [22]	.519	.782	.899	.733
$DeCAF_6$ [38]	.807 ± .023	.948 ± .012	–	–
DaNN [30]	.536 ± .002	.712 ± .000	.835 ± .000	.694
DDC [7]	.841 ± .006	.954 ± .004	.963 ± .003	.919
DC + Soft Labs. [10]	.827 ± .008	.957 ± .005	.976 ± .002	.920
So-HoT [46]	.845 ± .017	.955 ± .006	.975 ± .007	.925
Ours	**.867 ± .015**	**.960 ± .008**	**.988 ± .006**	**.938**

We evaluate on all 31 categories according to the supervised evaluation protocol described in [15] and report the mean and standard deviation over 5 runs.

TABLE 7
Comparison Against Other Domain Adaptation Techniques on the MNIST+USPS Standard Benchmark

	Accuracy		
	M→U	U→M	Average
PCA	0.451	0.334	0.392
SA [19]	0.486	0.222	0.354
GFK [18]	0.346	0.226	0.286
TCA [27]	0.408	0.274	0.341
SSTCA [27]	0.406	0.222	0.314
TSL [69]	0.435	0.341	0.388
JCSL [68]	0.467	0.355	0.411
DDC [7]	0.478	0.631	0.554
Ours	**0.607**	**0.673**	**0.640**

we work in the unsupervised setting, and thus ignore the target domain labels at training time. Following [55], as the image patches in the *USPS* dataset are only 16×16 pixels, we rescaled the images from *MNIST* to the same size and applied L_2 normalization of the pixel intensities. For this experiment, we relied on the standard CNN architecture of [59] and employed our MMD-based criterion to determine which layers should not share their weights. We found that allowing all layers of the network not to share their weights yielded the best performance.

In Table 7, we compare our approach with DDC [7] and with methods that do not rely on deep networks [18], [27], [68]. Our method yields superior performance in all cases, which we believe to be due to its ability to adapt the feature representation to each domain, while keeping these representations close to each other.

4.4 Unsupervised Domain Adaptation on SVHN-MNIST

We now compare our method to the ADDA algorithm [48], which is close in spirit to ours. As discussed in Section 2.2, it aims at modeling the domain shift instead of learning features invariant to it. In practice, this involves pretraining the network on the source data and then fine-tuning it by minimizing the discrepancy between the target features and the fixed source feature representation in an adversarial fashion using the loss function introduced in [10].

TABLE 8
Evaluation on the SVHN → MNIST Domain Adaptation Task

method	SVHN → MNIST
CNN (trained on Source only)	.601 ± .011
ADDA [48]	.760 ± .002
GRL [9]	.807 ± .016
Ours	**.828 ± .002**

To perform this comparison we applied our method to the SVHN [70] to MNIST domain adaptation task. We rely on the same LeNet [59] architecture and domain classifier as in [48]. Also as in [48], we use all the SVHN images as source data and the full training set from MNIST as unlabeled training data in the target domain. We rescaled the SVHN images to 28×28 pixels to match the size of the MNIST ones. In Table 8, we report the average accuracy and its variance across 5 runs. Our method significantly outperforms ADDA, presumably because allowing the network weights to vary freely makes the problem too unconstrained and leads to overfitting.

As also depicted by Table 8, we outperform GRL [9], which our approach with *all* layers shared closely approximates. This confirms again that allowing some layers not to be shared increases performance.

4.5 Facial Pose Estimation

To demonstrate that our method can be used not only for classification or detection tasks but also for regression ones, we further evaluate it for facial pose estimation purposes. More specifically, the task we address consists of predicting the location of 5 facial landmarks given 50×50 image patches, such as those of Fig. 10. To this end, we first apply gamma correction [71] to these patches as a preprocessing step and then train a regressor to predict a 10D vector with two floating point coordinates for each landmark. As we did for drones in Section 4.1, we use *synthetic* images, such as the ones illustrated in the top portion of Fig. 10, as our source domain and *real* ones, such as those shown at the bottom, as our target domain. Both datasets contain $\sim 10k$ annotated images.

To evaluate our approach, we considered the unsupervised domain adaptation scenario. In this case, we use only annotations for the images from the source domain and do not use any in the target one.

Our architecture is depicted by Fig. 11. The same figure further shows the MMD^2 values corresponding to different configurations of layers with shared/non-shared weights. We report the mean and standard deviation of our method with different configurations across 5 runs. As in the UAV case, having the first layers of the network be stream specific helps handle appearance changes and the resulting features are close to each other in terms of the MMD^2 score. By contrast, allowing later layers not to be shared introduces too much flexibility and results in overfitting.

In Table 9, we compare our Domain Adaptation results to those of DDC [7] and GRL [9] in terms of root mean square error (RMSE). The latter is computed as

$$\mathrm{RMSE} = \frac{1}{N}\sqrt{\sum_{i=1}^{N}\sum_{j=1}^{10}(\hat{y}_{ij} - y_{ij})^2}, \tag{16}$$

where y_{ij} and $\hat{y}_{ij}$ are the predicted and the ground-truth values of the jth landmark vector's coordinate of test sample

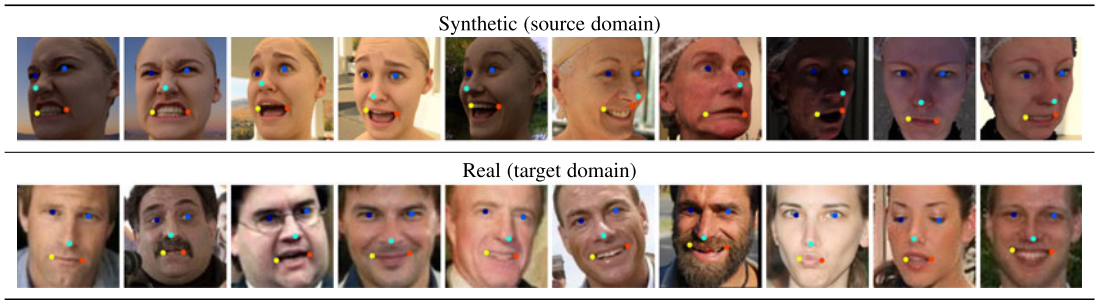


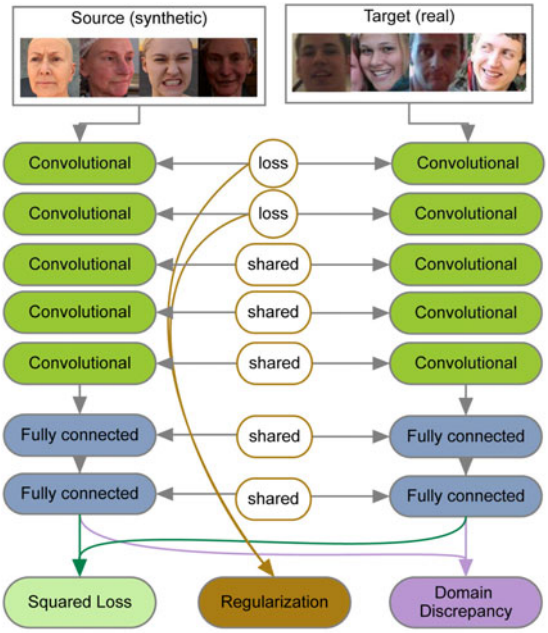
Fig. 10. Samples from *Source* and *Target* datasets with synthetic and real images respectively.

$i \in [1..N]$, with N being the number of test samples. Note that, again, by not sharing the weights, our approach outperforms the competitors.

4.6 Discussion

In all the experiments reported above, allowing the weights *not* to be shared in some fraction of the layers of our two-stream architecture boosts performance. This validates our initial hypothesis that explicitly modeling the domain shift is generally beneficial.

However, the optimal choice of which layers should or should not share their weights is application dependent.



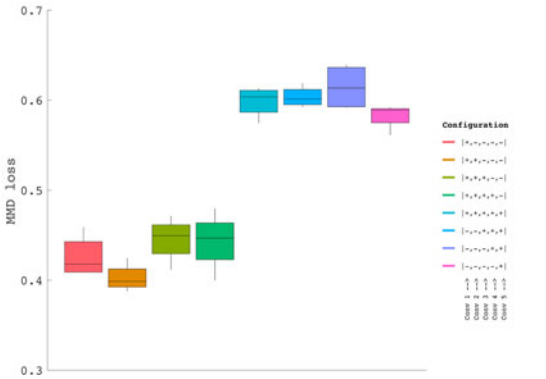


Fig. 11. [Top] Network architecture for facial pose estimation. [Bottom] Network architecture selection. '-' and '+' denote that the weights of the corresponding layers are shared and stream-specific, respectively.

In the UAV detection and facial pose estimation cases allowing the weights in the first two layers to be different yields top performance, which we understand to mean that the domain shift is caused by low-level changes that are best handled in the early layers. By contrast, for the *Office* dataset, it is best to only allow the weights in the last two layers to differ. This network configuration was determined using *Amazon* and *Webcam* images, such as those shown in Fig. 8. Close examination of these images reveals that the differences between them are not simply due to low-level phenomena, such as illumination changes, but to more complex variations. It therefore seems reasonable that the higher layers of the network, which encode higher-level information, should be domain-specific.

Fortunately, we have shown that MMD provides us with an effective criterion to choose the right configuration. This makes our two-steam approach practical, even when no validation data is available. Furthermore, the works presented in [72], [73] suggest alternatives to the MMD that also could be used for architecture selection. For example, we could use cross-stitch units [72], which would add 4 additional parameters to each pair of layers and give the network the needed flexibility to adapt its architecture to a specific task. Another approach would be to progressively increase the network complexity, as in [73]. Instead of adding a complete stream to an already existing network as in the original paper, one could change the parameters in some of the layers to be stream-specific. This may, however, lead to uncontrollable network growth as we allow the network to increase its capacity by adding stream-specific parameters. This issue might be addressed by employing approaches that reduce the number of neurons in the network, such as the ones described in [74], [75], which would constitute an interesting direction for future research.

5 Conclusion

In this paper, we have postulated that Deep Learning approaches to Domain Adaptation should not focus on learning features that are invariant to the domain shift,

TABLE 9
Regression Results for the Facial Pose Estimation Task

	Synthetic → Real
	RMSE [10^{-2} px]
*CNN (trained on Synthetic only (**S**))*	6.79
DDC [7]	6.56± .01
GRL [9]	6.46± .04
Ours	**6.34± .03**

which makes them less discriminative. Instead, we should explicitly model the domain shift. To prove this, we have introduced a two-stream CNN architecture, where the weights of the streams may or may not be shared. To nonetheless encode the fact that both streams should be related, we encourage the non-shared weights to remain close to being linear transformations of each other by introducing an additional loss term.

Our experiments on very diverse datasets have clearly validated our hypothesis. Our approach consistently yields higher accuracy than networks that share all weights for the source and target data, both for classification and regression. In the future, we intend to study if more complex weight transformations could help us further improve our results, with a particular focus on designing effective constraints for the parameters of these transformations.

Acknowledgements

This work was supported in part by the Swiss Commission for Technology and Innovation.

References

[1] G. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, pp. 1391–1415, 2006.
[2] Y. LeCun, L. Bottou, G. Orr, and K. Müller, *Neural Networks: Tricks of the Trade*. Berlin, Germany: Springer, 1998, ch. Efficient Backprop.
[3] J. Jiang, "A literature survey on domain adaptation of statistical classifiers," vol. 3, 2008, http://sifaka. cs. uiuc. edu/jiang4/domainadaptation/survey
[4] S. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.
[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 580–587.
[6] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 1717–1724.
[7] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell, "Deep domain confusion: Maximizing for domain invariance," *CoRR*, vol. abs/1412.3474, 2014, http://arxiv.org/abs/1412.3474
[8] M. Long, Y. Cao, J. Wang, and M. I. Jordan, "Learning transferable features with deep adaptation networks," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 97–105.
[9] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. S. Lempitsky, "Domain-adversarial training of neural networks," *J. Mach. Learn. Res.*, vol. 17, pp. 59:1–59:35, 2016.
[10] E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko, "Simultaneous deep transfer across domains and tasks," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 4068–4076.
[11] L. Duan, I. Tsang, D. Xu, and S. Maybank, "Domain transfer SVM for video concept detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 1375–1381.
[12] A. Bergamo and L. Torresani, "Exploiting weakly-labeled web images to improve object classification: A domain adaptation approach," in *Proc. Adv. Neural Inf. Process. Syst.*, 2010, pp. 181–189.
[13] C. Becker, M. Christoudias, and P. Fua, "Non-linear domain adaptation with boosting," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 485–493.
[14] H. Daumé and D. Marcu, "Domain adaptation for statistical classifiers," *J. Artif. Intell. Res.*, vol. 26, no. 1, pp. 101–126, 2006.
[15] K. Saenko, B. Kulis, M. Fritz, and T. Darrell, "Adapting visual category models to new domains," in *Proc. Eur. Conf. Comput. Vis.*, 2010, pp. 213–226.
[16] B. Kulis, K. Saenko, and T. Darrell, "What you saw is not what you get: Domain adaptation using asymmetric Kernel transforms," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2011, pp. 1785–1792.
[17] R. Gopalan, R. Li, and R. Chellappa, "Domain adaptation for object recognition: An unsupervised approach," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2011, pp. 999–1006.
[18] B. Gong, Y. Shi, F. Sha, and K. Grauman, "Geodesic flow Kernel for unsupervised domain adaptation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 2066–2073.
[19] B. Fernando, A. Habrard, M. Sebban, and T. Tuytelaars, "Unsupervised visual domain adaptation using subspace alignment," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 2960–2967.
[20] R. Caseiro, J. Henriques, P. Martins, and J. Batista, "Beyond the shortest path : Unsupervised domain adaptation by sampling subspaces along the spline flow," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 3846–3854.
[21] S. Chen, F. Zhou, and Q. Liao, "Visual domain adaptation using weighted subspace alignment," in *Proc. SPIE Int. Conf. Vis. Commun. Image Process.*, 2016, pp. 1–4.
[22] S. Chopra, S. Balakrishnan, and R. Gopalan, "DLID: Deep learning for domain adaptation by interpolating between domains," in *Proc. Int. Conf. Mach. Learn. Workshop Challenges in Representation Learn*, 2013.
[23] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. J. Smola, "A kernel method for the two-sample problem," *CoRR*, vol. abs/0805.2368, http://arxiv.org/abs/0805.2368, 2008,
[24] J. Huang, A. Smola, A. Gretton, K. Borgwardt, and B. Scholkopf, "Correcting sample selection bias by unlabeled data," in *Proc. Adv. Neural Inf. Process. Syst.*, 2006, pp. 601–608.
[25] A. Gretton, A. Smola, J. Huang, M. Schmittfull, K. Borgwardt, and B. Schölkopf, "Covariate shift by kernel mean matching," *J. Roy. Statist. Soc.*, vol. 3, no. 4, pp. 5–13, 2009.
[26] B. Gong, K. Grauman, and F. Sha, "Connecting the dots with landmarks: Discriminatively learning domain-invariant features for unsupervised domain adaptation," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 222–230.
[27] S. Pan, I. Tsang, J. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," in *Proc. Int. Joint Conf. Artif. Intell.*, 2009, pp. 1187–1192.
[28] K. Muandet, D. Balduzzi, and B. Schölkopf, "Domain generalization via invariant feature representation," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 10–18.
[29] M. Baktashmotlagh, M. Harandi, B. Lovell, and M. Salzmann, "Unsupervised domain adaptation by domain invariant projection," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 769–776.
[30] M. Ghifary, W. B. Kleijn, and M. Zhang, "Domain adaptive neural networks for object recognition," in *Proc. Pacific Rim Int. Conf. Artif. Intell.*, 2014, pp. 898–904.
[31] L. Duan, D. Xu, I. Tsang, and J. Luo, "Visual event recognition in videos by learning from web data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 9, pp. 1667–1680, Sep. 2012.
[32] L. Duan, I. Tsang, and D. Xu, "Domain transfer multiple Kernel learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 3, pp. 465–479, Mar. 2012.
[33] W. Li, Z. Xu, D. Xu, D. Dai, and L. Van Gool, "Domain generalization and adaptation using low rank exemplar SVMs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PP, no. 99, p. 1, 2018, doi: 10.11069/TPAMI.2017.2704624.
[34] Y. Yang, Z. Ma, A. Hauptmann, and N. Sebe, "Feature selection for multimedia analysis by sharing information among multiple tasks," *IEEE Trans. Multimedia*, vol. 15, no. 3, pp. 661–669, Apr. 2013.
[35] Z. Ma, Y. Yang, N. Sebe, and A. Hauptmann, "Knowledge adaptation with partially shared features for event detection using few exemplars," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 9, pp. 1789–1802, Sep. 2014.
[36] D. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 20, no. 2, pp. 91–110, 2004.
[37] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," *Comput. Vis. Image Understanding*, vol. 10, no. 3, pp. 346–359, 2008.
[38] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "DeCAF: A deep convolutional activation feature for generic visual recognition," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 647–655.
[39] Y. H. Tsai, Y. Yeh, and Y. F. Wang, "Learning cross-domain landmarks for heterogeneous domain adaptation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 5081–5090.
[40] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2005, pp. 539–546.

[41] X. Glorot, A. Bordes, and Y. Bengio, "Domain adaptation for large-scale sentiment classification: A deep learning approach," in *Proc. Int. Conf. Mach. Learn.*, 2011, pp. 513–520.
[42] M. Kan, S. Shan, and X. Chen, "Bi-shifting auto-encoder for unsupervised domain adaptation," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 3846–3854.
[43] M. Long, H. Zhu, J. Wang, and M. I. Jordan, "Unsupervised domain adaptation with residual transfer networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 136–144.
[44] B. Sun, J. Feng, and K. Saenko, "Correlation alignment for unsupervised domain adaptation," *Domain Adaptation in Computer Vision Applications*, pp. 153–171, 2017, doi: 10.1007/978-3-319-58347-1_8.
[45] B. Sun, J. Feng, and K. Saenko, "Return of frustratingly easy domain adaptation," in *Proc. Amer. Assoc. Artif. Intell. Conf.*, 2016, pp. 2058–2065.
[46] P. Koniusz, Y. Tas, and F. Porikli, "Domain adaptation by mixture of alignments of second- or higher-order scatter tensors," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 7139–7148.
[47] M. Long, J. Wang, and M. I. Jordan, "Deep transfer learning with joint adaptation networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2208–2217.
[48] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 7167–7176.
[49] M. Ghifary, W. B. Kleijn, M. Zhang, D. Balduzzi, and W. Li, "Deep reconstruction-classification networks for unsupervised domain adaptation," in *Proc. Eur. Conf. Comput. Vis.*, pp. 597–613, 2016.
[50] P. Liu, C. Wang, P. Yang, K. Huang, and T. Tan, "Cross-domain object recognition using object alignment," in *Proc. Brit. Mach. Vis. Conf.*, 2015, pp. 66.1–66.12.
[51] O. Sener, H. O. Song, A. Saxena, and S. Savarese, "Learning transferrable representations for unsupervised domain adaptation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2110–2118.
[52] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan, "Domain separation networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 343–351.
[53] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Advances Neural Inf. Process. Syst.*, 2012, pp. 1106–1114.
[54] Q. Chen, J. Huang, R. Feris, L. M. Brown, J. Dong, and S. Yan, "Deep domain adaptation for describing people based on fine-grained clothing attributes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 5315–5324.
[55] M. Long, J. Wang, G. Ding, J. Sun, and P. Yu, "Transfer feature learning with joint distribution adaptation," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 2200–2207.
[56] D. Kingma and J. Ba, "Adam: A method for stochastic optimisation," in *Proc. Int. Conf. Learn. Representations*, 2015, Art. no. 13.
[57] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.
[58] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of momentum and initialization in deep learning," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1139–1147.
[59] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[60] A. Rozantsev, V. Lepetit, and P. Fua, "On rendering synthetic images for training an object detector," *Comput. Vis. Image Understanding*, vol. 137, pp. 24–37, 2015.
[61] J. Jin, K. Fu, and C. Zhang, "Traffic sign recognition with hinge loss trained convolutional neural networks," *Trans. Intell. Transp. Syst.*, vol. 15, pp. 1991–2000, 2014.
[62] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2017–2025.
[63] W. Li, L. Duan, D. Xu, and I. W. Tsang, "Learning with augmented features for supervised and semi-supervised heterogeneous domain adaptation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 6, pp. 1134–1148, Jun. 2014.
[64] S. Pan, I. Tsang, J. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *IEEE Trans. Neural Netw.*, vol. 22, no. 2, pp. 199–210, Feb. 2011.
[65] J. Weston and F. Ratle, "Deep learning via semi-supervised embedding," in *Proc. Int. Conf. Mach. Learn.*, 2008, pp. 1168–1175.
[66] T. Tommasi and B. Caputo, "Frustratingly easy NBNN domain adaptation," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2013, pp. 897–904.
[67] J. Hull, "A database for handwritten text recognition research," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 5, pp. 550–554, May 1994.
[68] B. Fernando, T. Tommasi, and T. Tuytelaars, "Joint cross-domain classification and subspace learning for unsupervised adaptation," *Pattern Recognit. Lett.*, vol. 65, pp. 60–66, 2015.
[69] S. Si, D. Tao, and B. Geng, "Bregman divergence-based regularization for transfer subspace learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 7, pp. 929–942, Jul. 2010.
[70] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014.
[71] C. A. Poynton, *Digital Video and HDTV: Algorithms and Interfaces*. San Mateo, CA, USA: Morgan Kaufmann, 2003.
[72] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 3994–4003.
[73] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *CoRR*, vol. abs/1606.04671, 2016.
[74] M. D. Collins and P. Kohli, "Memory bounded deep convolutional networks," *CoRR*, vol. abs/1412.1442, http://arxiv.org/abs/1606.04671, 2014.
[75] J. M. Alvarez and M. Salzmann, "Learning the number of neurons in deep networks," *Adv. Neural Inf. Process. Syst.*, pp. 2262–2270, Dec. 2016.

Artem Rozantsev received the Specialist degree in mathematics and computer science from Lomonosov Moscow State University, in 2012. He is working toward the PhD degree in Computer Vision Laboratory, École Polytechnique Fédérale de Lausanne, under the supervision of Prof. Pascal Fua and Prof. Vincent Lepetit, in 2012. His research interests include object detection, synthetic data generation, and machine learning.

Mathieu Salzmann received the MSc and PhD degrees from École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, in 2004 and 2009, respectively. He then worked with the International Computer Science Institute and the Department of Electrical and Electronic Communication Systems, University of California at Berkeley, Berkeley, California, as a postdoctoral fellow under the supervision of Prof. Trevor Darrell. He was later a research assistant professor with the Toyota Technical Institute at Chicago, and a senior researcher and Rresearch leader with NICTA, in Canberra. He is a member of the IEEE and currently a senior researcher with the Computer Vision Laboratory, École Polytechnique Fédérale de Lausanne.

Pascal Fua received the engineering degree from Ecole Polytechnique, Paris, in 1984 and the PhD degree in computer science from the University of Orsay, in 1989. He then worked with SRI International and INRIA Sophia-Antipolis as a computer scientist. He started work with EPFL, in 1996 where he is now a professor with the School of Computer and Communication Science and heads the Computer Vision Laboratory. His research interests include shape modeling and motion recovery from images, analysis of microscopy images, and Augmented Reality. He has (co)authored more than 200 publications in refereed journals and conferences. He has been an associate editor of the *IEEE Transactions for Pattern Analysis and Machine Intelligence*. He often serves as program committee member, area chair, and program chair of major vision conferences. He is a fellow of the IEEE.