Joint Hierarchical Domain Adaptation and Feature Learning

Hien V. Nguyen*, Member, IEEE, Huy Tho Ho†, Student Member, IEEE, Vishal M. Patel Member, IEEE, and Rama Chellappa, Fellow, IEEE

Abstract—Complex visual data contain discriminative structures that are difficult to be fully captured by any single feature description. While recent work in domain adaptation focuses on adapting a single hand-crafted feature, it is important to perform adaptation on a hierarchy of features to exploit the richness of visual data. We propose a novel framework for domain adaptation using a sparse and hierarchical network (DASH-N). Our method jointly learns a hierarchy of features together with transformations that address the mismatch between different domains. The building block of DASH-N is designed using the theory of latent sparse representation. It employs a dimensionality reduction step that can prevent the data dimension from increasing too fast as we increase the depth of the hierarchy. Experimental results show that our method compares favorably with competing state-of-the-art methods. Moreover, it is shown that a multi-layer DASH-N performs better than a single-layer DASH-N.

Index Terms—Domain adaptation, hierarchical sparse representation, dictionary learning, object recognition.

I. INTRODUCTION

In many practical computer vision applications, we are often confronted with the situation where the data that we use to train our algorithm have different distribution or representation than that presented during the testing. The ubiquity of this problem is well-known to machine learning and computer vision researchers. For instance, indoor images are quite different from outdoor images, just as videos captured with a high definition camera are from that collected with a webcam. Differences in distributions between training and test data are often a dominant factor accounting for the poor performances of many computer vision algorithms. As an example of the effect of distribution mismatch, Ben-David et al. [1] show that, under certain assumption, the bound on the test error linearly increases with the $\ell_1$ divergence between training and testing distributions. Even worse, data from the test domain are often scarce and expensive to obtain. This makes it impractical to retrain an algorithm from the scratch since a learning algorithm would generalize poorly when an insufficient amount of data is presented [2]. Regardless of the cause, any distributional change that may occur after training can degrade the performance of the system when it comes to testing. Domain adaptation also known as domain-transfer learning attempts to minimize this degradation.

Understanding the problem caused by domain changes has received substantial attention in recent years. The problem can be informally stated as follows. Given a source domain whose representation or distribution can be different from that of the target domain, how to effectively utilize the model trained on the source to achieve a good performance on the target data. It is also often assumed that the source domain has sufficient labeled training samples while there are only a few (both labeled and unlabeled) samples available from the target domain. It has been shown in [3]–[7] that domain adaptation techniques can significantly improve the performance of computer vision tasks such as visual object detection and recognition.

Many existing algorithms for adapting a recognition system to a new visual domain share a common architecture containing two main stages. First, features are extracted separately for source and target using hand-crafted feature descriptors, followed by the second stage where transformations are learned in order to rectify the discrepancy between the two domains. This approach has several drawbacks. Without any knowledge about the target domain, the feature extraction process performed on the source data can ignore information important to the target data. For instance, let us consider a picture of a small bird in a forest background. The forest features apparently dominate the picture description. While these features might be sufficient for classifying the picture into land animals or sea animals, they do not benefit the target domain which has the same task on images of animals without background.

Another issue is that discriminative information can be embedded in multiple levels of the features hierarchy. High-level features are sometimes more useful than low-level ones. In fact, this is one of the main motivations behind the development of hierarchical networks (e.g. [8], [9]) so that more complex abstraction from a visual object can be captured. The traditional framework of domain adaptation employs a shallow architecture containing a single layer. This ignores the possibility of transferring at multiple levels of the feature hierarchy.

Finally, the process of designing features, such as SIFT [10] or SURF [11], is tedious and time-consuming. It requires a deep understanding and a careful examination of the underlying physics that governs the generation of data. Such requirements might be impractical given that the data from the target domain are often very scarce.

*: equal contribution.
Hien V. Nguyen is with Siemens Corporate Research, Princeton, NJ 08540 (e-mail: hien@umiacs.umd.edu).
Huy Tho Ho, Vishal M. Patel and Rama Chellappa are with the Department of Electrical Engineering and the Center for Automation Research, UMIACS, University of Maryland, College Park, MD 20742 USA (e-mail: {huytho, pvishalm, rama}@umiacs.umd.edu).
Contributions: In order to address the limitations of existing methods, we propose a novel approach for domain adaptation that possesses the following advantages:

- Adaptation is performed on multiple levels of the feature hierarchy in order to maximize the knowledge transfer. The hierarchical structure allows the transfer of useful information that might not be well captured by existing domain adaptation techniques.
- Adaptation is done jointly with feature learning. Our method learns a hierarchy of sparse codes and uses them to describe a visual object instead of relying on any low-level feature.
- Unlike existing hierarchical networks, our approach is more computationally efficient with a mechanism to prevent the data dimension from increasing too fast as the number of layer increases.

We provide extensive experiments to show that our approach performs better than many state-of-the-art domain adaptation methods. This is interesting since in our method, training is entirely generative followed by a linear support vector machine while other methods such as [12] employ discriminative training together with non-linear kernels. Furthermore, we introduce a new set of data for benchmarking the performance of our algorithm. The new dataset has two domains containing half-toned and edge images, respectively. In order to facilitate future research in the area, a Matlab implementation of our method will be made available.

A. Organization of the paper

This paper is organized as follows. Related works on domain adaptation are discussed in Section II. The main formulation of DASH-N is given in Section III, followed by the optimization procedure in Section V. Experimental results on domain adaptation for object recognition are presented in Section VI. Finally, Section VII concludes the paper with a brief summary and discussion.

II. RELATED WORKS

In this section, we review some related works on domain adaptation and hierarchical feature learning.

A. Domain Adaptation

While domain adaptation was first investigated in speech and natural language processing [13]–[15], it has been studied extensively in other areas such as machine learning [1], [16] and computer vision, especially in the context of visual object recognition [3]–[7]. Domain adaptation for visual recognition was introduced by Saenko et al. [3] in a semi-supervised setting. They employed metric learning to learn the domain shift using partially labeled data from the target domain. This work was extended by Kulis et al. [4] to handle asymmetric domain transformations. Gopalan et al. [5] addressed the problem of unsupervised domain adaptation, where samples from the target domain are unlabeled, by using an incremental approach based on Grassmann manifolds. By formulating a geodesic flow kernel, Gong et al. [6] and Zheng et al. [17] independently extended the idea of interpolation to integrate an infinite number of subspaces on the geodesic flow from the source domain to the target domain. Chen et al. [18] presented a co-training based method that slowly adapted a training set from the source to the target domain. An information-theoretic method for unsupervised domain adaptation was proposed by Shi and Sha [19] that attempted to find a common feature space, where the source and target data distributions are similar and the misclassification error is minimized.

Sparse representation and dictionary-based methods for domain adaptation [12], [20], [21] are also gaining a lot of traction. In particular, [20] modeled dictionaries across different domains with a parametric mapping function, while [12] enforced different domains to have a common sparse representation on some latent domain. Another class of techniques performed domain adaptation by directly learning a target classifier from classifiers trained on the source domain(s) [22], [23].

A major drawback of some of the existing approaches is that the domain shifting transformation is considered only at a single layer and may not capture adequately the shift between the source and target domain. It is worth noting that although [24] also named their method hierarchical domain adaptation, the paper is not quite related to ours. They made use of hierarchical Bayesian prior, while we employ a multi-layer network of sparse representation.

There are also some closely related machine learning problems that have been studied extensively, including transfer learning or multi-task learning [25], self-taught learning [26], semi-supervised learning [27] and multiview analysis [28]. Reviews of domain adaptation methods in machine learning and computer vision can be found in [29], [30]. A survey on the related field of transfer learning can be found in [16].

B. Hierarchical Feature Learning

Designing features for visual objects is a time-consuming and challenging task that requires a deep understanding of the domain knowledge. It is also non-trivial to adapt these manually designed features to new types of data such as hyperspectral or range-scan images. For this reason, learning features from the raw data has become increasingly popular and demonstrated competitive performances on practical computer vision tasks [8], [9], [31]. In order to capture the richness of data, a multi-layer or hierarchical network is employed to learn a spectrum of features, layer by layer.

The design of multi-layer networks has been an active research topic in computer vision. One of the early works includes [32], which used a multistage system to extract salient features in the image at different spatial scales. By learning higher-level feature representations from unlabeled data, deep belief networks (DBN) [8] and its variants, such as convolutional DBNs [9] and deep autoencoders [33], have been shown to be effective when applied to classification problems. Motivated by the recent works on deep learning, multi-layer sparse coding networks [31], [34], [35] have been proposed to build feature hierarchies layer by layer using sparse codes and spatial pooling. Each layer in these networks...
contains a coding step and a pooling step. A dictionary is learned at each coding step which then serves as a codebook for obtaining sparse codes from image patches or pooled features. Spatial pooling schemes, most notably max-pooling, group the sparse codes from adjacent blocks into common entities. This operation makes the resulting features more invariant to certain changes caused by translation and rotation. The pooled sparse codes from one layer serve as inputs to the next layer. Although the high dimension of the feature vectors obtained from a hierarchical network may provide some improvements in classification tasks [8], [9], [31], it may also lead to high redundancy and thus, reduce the efficiency of these algorithms. As a result, it is desirable to have a built-in mechanism in the hierarchy to reduce the dimension of the feature vectors while keeping their discriminative power. Hierarchical feature learning has also been used in domain adaptation such as in [36], [37].

III. BACKGROUND

Since our formulation is based on sparse coding and dictionary learning, in this section, we briefly give a background on these topics.

A. Dictionary Learning

Given a set of training samples \( Y = [y_1, \ldots, y_n] \in \mathbb{R}^{d \times n} \), the problem of learning a dictionary together with the sparse codes is typically posed as the minimization of the following cost function over \( (D, X) \):

\[
\|Y - DX\|_F^2 + \beta \Psi(X) \quad \text{s.t.} \quad \|d_i\|_2 = 1, \forall i \in [1, K]
\]

where \( \|Y\|_F \) denotes the Frobenius norm defined as \( \|Y\|_F = \sqrt{\sum_{i,j} |Y_{i,j}|^2} \), \( D = [d_1, \ldots, d_K] \in \mathbb{R}^{d \times K} \) is the sought dictionary, \( X = [x_1, \ldots, x_n] \in \mathbb{R}^{K \times n} \) is the horizontal concatenation of the sparse codes, \( \beta \) is a non-negative constant and \( \Psi \) promotes sparsity. Various methods have been proposed in the literature for solving such optimization problem. In the case when \( \ell_1 \) norm is enforced, the K-SVD [38] algorithm can be used to train a dictionary. One can also promote sparsity by enforcing the \( \ell_1 \) norm on \( X \). In this case, one can use the algorithm proposed in [39] to solve the above problem. See [38] and [39] for more details.

B. Latent Sparse Representation

From the observation that signals often lie on a low-dimensional manifold, several authors have proposed to perform dictionary learning and sparse coding on a latent space [40]–[42]. We call it latent sparse representation to distinguish from the formulation in (1). This is done by minimizing the following cost function over \( (P, D, X) \):

\[
\mathcal{L}(Y, P, D, X, \alpha, \beta) = \|PY - DX\|_F^2 + \alpha \|Y - P^TYPY\|_F^2 + \beta \|X\|_1 \\
\text{s.t. } PP^T = I \quad \text{and} \quad \|d_i\|_2 = 1, \forall i \in [1, K],
\]

where \( P \in \mathbb{R}^{p \times d} \) is a linear transformation that brings the data to a low-dimensional feature space \( (p < d) \). Note that the dictionary is now in the low-dimensional space \( D \in \mathbb{R}^{p \times K} \). Besides the computational advantage, [41] shows that this optimization can recover the underlying sparse representation better than traditional dictionary learning methods. This formulation is attractive since it allows the transformation of the data into another domain to better handle different sources of variation such as illumination and geometric articulation.

IV. HIERARCHICAL DOMAIN ADAPTATION

Motivated by the work of [31], we propose a method to perform hierarchical domain adaptation jointly with feature learning. Figure 1 shows an overview of the proposed method. The network contains multiple layers, each of which contains 3 sub-layers as illustrated in Figure 1. The first sub-layer performs contrast-normalization and dimensionality reduction on the input data. Sparse coding is carried out in the second sub-layer. In the final sub-layer, adjacent features are max-pooled together to produce a new features. Output from one layer becomes the input to the next layer. For the simplicity of notation, we consider a single source domain. The extension of DASH-N to multiple source domains is straightforward and is included in the Appendix.

Let \( Y_S \in \mathbb{R}^{d_S \times n_S} \) and \( Y_T \in \mathbb{R}^{d_T \times n_T} \) be the input data at each layer from source and target domains, respectively. Note that there are \( n_S, d_S \)-dimensional samples in the source domain and \( n_T, d_T \)-dimensional samples in the target domain.

Given \( Y_S \) and \( Y_T \), in each layer of DASH-N, we learn a joint latent sparse representation by minimizing the following cost function with respect to \( (P_S, P_T, D, X_S, X_T) \):

\[
\mathcal{L}(Y_S, P_S, D, X_S, \alpha, \beta) + \lambda \mathcal{L}(Y_T, P_T, D, X_T, \alpha, \beta)
\]

\[
\text{s.t. } PP_S^T = P_T P_T^T = I, \quad \|d_i\|_2 = 1, \forall i \in [1, K],
\]

where \( \alpha, \beta, \lambda \) are the non-negative constants, \( D \in \mathbb{R}^{n \times K} \) is the common dictionary, \( P_S \in \mathbb{R}^{p \times d_S} \) and \( P_T \in \mathbb{R}^{p \times d_T} \) are the transformations to the latent domain, \( X_S \in \mathbb{R}^{K \times n_S} \) and \( X_T \in \mathbb{R}^{K \times n_T} \) are the sparse codes of the source and the target domains, respectively. As can be seen from the above formulation, the two domains are forced to share a common dictionary in the latent domain. Together with sparsity constraint, the common \( D \) provides a coupling effect that promotes the discovery of common structure between the two domains.

For simplicity, in what follows, we provide a detailed discussion on a two-layer DASH-N network. Extension of DASH-N to multiple layers is straight forward.

A. Layer 1

We perform dense sampling on each training image to get a set of overlapping patches. These patches are then contrast-normalized. If \( f \) is a vector corresponding to a patch, then the contrast-normalization can be performed as in [35]

\[
\hat{f} = \frac{f}{\sqrt{\|f\|^2 + \epsilon}},
\]

where \( \epsilon \) is some parameter. We set the value of \( \epsilon \) equal to 0.1 as it is found to work well in our experiments. In order to make the computation more efficient, only a random subset
of patches from each image is used for learning the latent sparse representation. We found that setting this number to 150 for images of maximum size of $150 \times 150$ provides a good trade-off between accuracy and computational efficiency. After learning the dictionary $D_1$ and the transformations $(P^s_S, P^t_T)$, the sparse codes $(X^s_1, X^t_1)$ are computed for all sampled patches by solving the following optimization problem

$$
\min_{X^1} \| P^1 S Y^1 - D_1 X^1 \|_2^2 + \beta_1 \| X^1 \|_1, \tag{6}
$$

where $\ast$ indicates that the above problem can either correspond to source data or target data. Each column of $Y^1_\ast$ is the vectorized pixel values inside a patch. A fast implementation of the LARS algorithm is used for solving this optimization problem [39].

Spatial max pooling is used to aggregate the sparse codes over each $4 \times 4$ neighborhood as this pooling method is particularly well-suited for the separation of sparse features [43].

B. Layer 2

In this layer, we perform similar computations except that the input is the sparse codes from layer 1 instead of image pixels. The features obtained from the previous layer are aggregated by concatenation over each $4 \times 4$ neighborhood and contrast-normalized. This results in a new representation that is more robust to occlusion and illumination. Similar to layer 1, we also randomly sample 150 normalized feature vectors $f$ from each image for training. $\ell_1$ optimization is again employed to compute the sparse codes of the normalized features $f$.

At the end of layer 2, the sparse codes are then aggregated using max pooling in a multi-level patch decomposition (i.e. spatial pyramid max pooling). At level 0 of the spatial pyramid, a single feature vector is obtained by performing max pooling over the whole image. At level 1, the image is divided into four quadrants and max pooling is applied to each quadrant, yielding 4 feature vectors. Similarly, for level 2, we obtain 9 feature vectors, and so on. In this paper, max pooling using a three level spatial pyramid is used. As a result, the final feature vector returned by the second layer for each image is the result of concatenating 14 feature vectors from the spatial pyramid.

V. OPTIMIZATION PROCEDURE

In this section, we elaborate on the details of how the cost function in (3) is minimized. First, let us define

$$
K_S = Y_S^T Y_S, \; K_T = Y_T^T Y_T, \; K = \begin{pmatrix} K_S & 0 \\ 0 & \sqrt{\lambda} K_T \end{pmatrix} \tag{7}
$$

to be the Gram matrix of source, target, and their block diagonal concatenation, respectively. It can be shown that (see the Appendix) the optimal solution of (3) takes the following form

$$
D = [A_S^T K_S, \sqrt{\lambda} A_T^T K_T] B, \tag{8}
$$

$$
P_S = (Y_S A_S)^T, \; P_T = (Y_T A_T)^T, \tag{9}
$$

for some $A_S \in \mathbb{R}^{n_S \times p}$, $A_T \in \mathbb{R}^{n_T \times p}$ and $B \in \mathbb{R}^{(n_S+n_T) \times K}$. Notice that rows of each transformation live in the column subspace of the data from its own domain. In contrast, columns of the dictionary are jointly created by the data of both source and target.

A. Solving for $(A_S, A_T)$

The orthogonal constraint in (4) can be re-written using (9) as

$$
A_S^T K_S A_S = I, \; A_T^T K_T A_T = I. \tag{10}
$$
By substituting (9), (8) into (3) and making use of the orthogonal constraint in (10), the formulation can be simplified as follows (see derivation in the Appendix)

$$\min_G \text{tr}(G^T H G) \quad \text{s.t.} \quad G^T S G = G^T G = I, \quad (11)$$

where $H$ is defined as

$$H = \Lambda^\frac{1}{2} V^T K (I - BX)(I - BX)^T - \alpha I )K V\Lambda^\frac{1}{2}, \quad (12)$$

$$V = \begin{pmatrix} V_S & 0 \\ 0 & V_T \end{pmatrix}, \quad \Lambda = \begin{pmatrix} \Lambda_S & 0 \\ 0 & \sqrt{\Lambda_T} \end{pmatrix}, \quad (13)$$

$$K_S = V_S \Lambda_S V_S^T, \quad K_T = V_T \Lambda_T V_T^T. \quad (14)$$

Here (14) is given by the eigen-decompositions of the Gram matrices. Finally, $G$ is defined as

$$G = [G_S, \sqrt{\Lambda} G_T],$$

$$G_S = \Lambda_S^\frac{1}{2} V^T_S A_S, \quad G_T = \Lambda_T^\frac{1}{2} V^T_T A_T. \quad (16)$$

The optimization in (11) is non-convex due to the orthogonality constraints. However, $G$ can be learned efficiently using the feasible method for optimization based on Cayley transform proposed by [44]. Given $G$, the solution of $(A_S, A_T)$ is simply given by

$$A_S = V_S \Lambda_S^{-\frac{1}{2}} G_S, \quad A_T = V_T \Lambda_T^{-\frac{1}{2}} G_T. \quad (17)$$

We note that the optimization step involves the eigen-decompositions of large Gram matrices whose dimensions equal to the number of training samples ($\approx 10^5$ in our experiments). As this process is very computationally expensive, we instead compute the eigen-decomposition of the following matrix

$$C_S = Y_S Y_S^T = U_S \Lambda'_S U_S^T = \mathbb{R}^{d_S \times d_S}. \quad (18)$$

Then, the $d_S$ dominant eigenvectors of $K_S$ can be recovered as

$$V_S = Y_S^T U_S \Lambda_S^{-\frac{1}{2}}. \quad (19)$$

The relationship in (19) between $V_S$ and $U_S$ can be easily verified using an SVD-decomposition of $Y_S$. The computations for the target domain can be performed in a similar fashion.

The signal dimension $d_S$ is much smaller than the number of training samples $n_S$ in our experiments (e.g. $10^3$ versus $10^5$). The eigen-decomposition of $C_S$ is therefore much more efficient than that of $K_S$. Finally, non-zero eigenvalues in $A_S$ are given by the diagonal coefficients of $\Lambda'_S$.

### B. Solving for $(B, X)$

If we fix $(A_S, A_T)$, then learning $(B, X_S, X_T)$ can be done using any dictionary learning algorithm. In order to see this, let us define

$$Z = [A_S K_S, \sqrt{\Lambda} A_T K_T], \quad (20)$$

$$X = [X_S, \sqrt{\Lambda} X_T]. \quad (21)$$

The cost function can be re-written in a familiar form as follows

$$\|Z - DX\|^2_F + \beta(\|X_S\|_1 + \lambda \|X_T\|_1). \quad (22)$$

We use an efficient implementation of LARS [39] to solve for the sparse codes $X$ and the efficient online dictionary learning algorithm [39] to solve for $D$. $B$ can be recovered, using the relationship in (8), simply by

$$B = Z' D,$$

where $\dagger$ denotes the Moore-Penrose pseudo-inverse.

It is straightforward to extend the above formulation to handle the case of multiple source domains. Details of derivation for this case are included in the Appendix.

### VI. Experiments

The proposed algorithm is evaluated in the context of object recognition using a recent domain adaptation database [3], containing 31 classes, with the addition of images from the Caltech-256 dataset [45]. There are 10 common classes between the two datasets (BACKPACK, TOURING-BIKE, CALCULATOR, HEADPHONES, COMPUTER-KEYBOARD, LAPTOP-101, COMPUTER-MONITOR, COMPUTER-MOUSE, COFFEE-MUG, and VIDEO-PROJECTOR) which contain a total of 2533 images. Domain shifts are caused by variations in factors such as pose, lighting, resolution, etc., between images in different domains. Figure 2 shows example images from the LAPTOP-101 class with respect to different domains. We compare our method with state-of-the-art adaptation algorithms reported in [3], [5], [6], [12]. Baseline results obtained using the hierarchical feature learning in [31] by learning the dictionaries separately for the source and target domains without performing domain adaptation are also included. Furthermore, in order to better assess the ability to adapt to a wide range of domains, experimental results are also reported on new images obtained by performing halftoning [46] and edge detection [47] algorithms on images from the datasets in [3], [45].

#### A. Experiment Setup

We follow the experimental set-ups of [6]. The results for 10 as well as 31 common classes are reported. In both cases,
TABLE I: Recognition rates of different approaches on four domains (C: Caltech, A: Amazon, D: DSLR, W: Webcam). 10 common classes are used. Red color denotes the best recognition rates. Blue color denotes the second best recognition rates.

<table>
<thead>
<tr>
<th>Method</th>
<th>C → A</th>
<th>C → D</th>
<th>A → C</th>
<th>A → W</th>
<th>W → C</th>
<th>W → A</th>
<th>D → A</th>
<th>D → W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric [3]</td>
<td>33.7 ± 0.8</td>
<td>35.0 ± 1.1</td>
<td>27.3 ± 0.7</td>
<td>36.0 ± 1.0</td>
<td>21.7 ± 0.5</td>
<td>32.3 ± 0.8</td>
<td>30.3 ± 0.8</td>
<td>55.6 ± 0.7</td>
</tr>
<tr>
<td>SCG [3]</td>
<td>40.2 ± 0.7</td>
<td>36.6 ± 0.8</td>
<td>37.7 ± 0.5</td>
<td>37.9 ± 0.7</td>
<td>29.2 ± 0.7</td>
<td>38.2 ± 0.6</td>
<td>39.2 ± 0.7</td>
<td>69.5 ± 0.9</td>
</tr>
<tr>
<td>GFK (PLS-PCA) [6]</td>
<td>46.1 ± 0.6</td>
<td>55.0 ± 0.9</td>
<td>39.0 ± 0.4</td>
<td>66.9 ± 1.0</td>
<td>32.8 ± 0.1</td>
<td>46.2 ± 0.6</td>
<td>46.2 ± 0.6</td>
<td>80.2 ± 0.4</td>
</tr>
<tr>
<td>SDDL [12]</td>
<td>49.5 ± 2.6</td>
<td>76.7 ± 3.9</td>
<td>27.4 ± 2.3</td>
<td>72.0 ± 4.8</td>
<td>29.7 ± 1.9</td>
<td>49.4 ± 2.1</td>
<td>48.9 ± 3.8</td>
<td>72.6 ± 2.1</td>
</tr>
<tr>
<td>HMP [31]</td>
<td>67.7 ± 2.3</td>
<td>70.2 ± 5.1</td>
<td>51.7 ± 4.3</td>
<td>70.0 ± 4.2</td>
<td>46.8 ± 2.1</td>
<td>61.5 ± 3.8</td>
<td>64.7 ± 2.0</td>
<td>76.0 ± 4.0</td>
</tr>
<tr>
<td>DASH-N (1st layer)</td>
<td>60.3 ± 2.7</td>
<td>79.6 ± 3.1</td>
<td>52.2 ± 2.1</td>
<td>74.1 ± 4.6</td>
<td>45.31 ± 3.7</td>
<td>68.5 ± 2.9</td>
<td>65.9 ± 2.1</td>
<td>76.3 ± 2.3</td>
</tr>
<tr>
<td>DASH-N (1st+2nd layers)</td>
<td><strong>71.6 ± 2.2</strong></td>
<td><strong>81.4 ± 3.5</strong></td>
<td><strong>54.9 ± 1.8</strong></td>
<td><strong>75.5 ± 4.2</strong></td>
<td><strong>50.2 ± 3.3</strong></td>
<td><strong>70.4 ± 3.2</strong></td>
<td><strong>68.9 ± 2.9</strong></td>
<td><strong>77.1 ± 2.8</strong></td>
</tr>
</tbody>
</table>

TABLE II: Single-source recognition rates on all 31 classes.

<table>
<thead>
<tr>
<th>Method</th>
<th>A → W</th>
<th>D → W</th>
<th>W → D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric [3]</td>
<td>44</td>
<td>31</td>
<td>27</td>
</tr>
<tr>
<td>RDALR [1]</td>
<td>50.7 ± 0.8</td>
<td>36.9 ± 19.9</td>
<td>32.9 ± 1.2</td>
</tr>
<tr>
<td>SCG [5]</td>
<td>57 ± 3.5</td>
<td>36 ± 1.1</td>
<td>37 ± 2.3</td>
</tr>
<tr>
<td>GFK (PLS-PCA) [6]</td>
<td>46.4 ± 0.5</td>
<td>61.3 ± 0.4</td>
<td>66.3 ± 0.4</td>
</tr>
<tr>
<td>SDDL [12]</td>
<td>50.1 ± 2.5</td>
<td>51.2 ± 2.1</td>
<td>50.6 ± 2.6</td>
</tr>
<tr>
<td>HMP [31]</td>
<td>55.7 ± 2.5</td>
<td>50.5 ± 2.7</td>
<td>56.8 ± 2.6</td>
</tr>
<tr>
<td>DASH-N (1st layer)</td>
<td>59.9 ± 2.7</td>
<td>65.8 ± 1.3</td>
<td>69.6 ± 2.1</td>
</tr>
<tr>
<td>DASH-N (1st+2nd layers)</td>
<td><strong>60.6 ± 3.5</strong></td>
<td><strong>67.9 ± 1.1</strong></td>
<td><strong>71.1 ± 1.7</strong></td>
</tr>
</tbody>
</table>

TABLE III: Multiple-source recognition rates on all 31 classes.

<table>
<thead>
<tr>
<th>Method</th>
<th>{D, A} → W</th>
<th>{A, W} → D</th>
<th>{W, D} → A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-SVM [50]</td>
<td>30.4 ± 0.6</td>
<td>25.3 ± 1.1</td>
<td>17.3 ± 0.9</td>
</tr>
<tr>
<td>RDALR [7]</td>
<td>36.9 ± 1.1</td>
<td>31.2 ± 1.3</td>
<td>20.9 ± 0.9</td>
</tr>
<tr>
<td>SCG [5]</td>
<td>52 ± 2.3</td>
<td>39 ± 1.1</td>
<td>28 ± 0.8</td>
</tr>
<tr>
<td>GFK [31]</td>
<td>61.0 ± 2.4</td>
<td>38.4 ± 3.4</td>
<td>19.30 ± 1.2</td>
</tr>
<tr>
<td>SDDL [12]</td>
<td>57.8 ± 2.4</td>
<td>56.7 ± 2.3</td>
<td>24.1 ± 1.6</td>
</tr>
<tr>
<td>HMP [31]</td>
<td>47.2 ± 1.9</td>
<td>51.3 ± 2.4</td>
<td>37.3 ± 1.4</td>
</tr>
<tr>
<td>DASH-N (1st layer)</td>
<td><strong>61.7 ± 2.5</strong></td>
<td><strong>64.1 ± 3.5</strong></td>
<td><strong>39.6 ± 1.3</strong></td>
</tr>
<tr>
<td>DASH-N (1st+2nd layers)</td>
<td><strong>64.5 ± 2.3</strong></td>
<td><strong>68.6 ± 3.7</strong></td>
<td><strong>41.8 ± 1.1</strong></td>
</tr>
</tbody>
</table>

experiments are performed in 20 random trials for each pair of source and target domains. If the source domain is Amazon or Caltech, 20 samples are used in the training. Otherwise, only 8 training samples are used for DSLR and Webcam. The number of target training samples is always set equal to 3. The remaining images from the target domain in each split are used for testing.

B. Parameter Settings

In our experiments, all images are resized to be no larger than 150 × 150 with preserved ratio and converted to grayscale. The patch size is set equal to 5 × 5. The parameter λ is set equal to 4 in order to account for fewer training samples from the target than that from the source, and α is set equal to 1.5 for all experiments. We also found that using β\textsubscript{train} = 0.3 for training and β\textsubscript{test} = 0.15 for testing yields the best performance. Smaller sparsity constant often makes the decoding more stable, thus, leads to more consistent sparse codes. This is similar to the finding in [31]. The number of dictionary atoms is set equal to 200 and 1500 in the first and second layer, respectively. The dimension of the latent domain is set equal to 20 and 750 in the first and second layer, respectively. It is worth noting that the input feature to layer 2 has the dimension of 3200. This results from aggregating sparse codes obtained from the first layer over a 4 × 4 spatial cell (4 × 4 × 200). By projecting them onto a latent domain of dimension 750, the computations become more tractable. A three level spatial pyramid, partitioned into 1 × 1, 2 × 2, and 3 × 3, is used to perform the max pooling in the final layer. A linear SVM [48] with the regularization parameter of 10 is employed for classification. It is worth noting that we do not use any part of the testing data in tuning the algorithm parameters. The sparsity constants such as α, β\textsubscript{train} and β\textsubscript{test} are set to the standard values used by many popular sparse learning softwares such as SPAMS [39] and ScSPM [49]. For parameters such as patch size, dictionary size, latent space dimensions and the linear SVM regularization parameter, the findings in [31], [38], [41] are employed to create a small subset of values and cross-validation on the training data is performed to obtain the optimal settings.

C. Computation Time

It takes an average of 35 minutes to perform the dictionary learning and feature extraction of all training samples using our Matlab implementation on a computer with a 3.8 GHz Intel i7 processor. It takes less than 2 seconds to compute the feature for a test image of size 150 × 150 using both layers of the hierarchy.

D. Object Recognition

1) 10 Common Classes: The recognition results of different algorithms on 8 pairs of source-target domains are shown in Table I. It can be seen that DASH-N outperforms all compared methods in 7 out of 8 pairs of source-target domains. For pairs such as Caltech-Amazon, Webcam-Amazon, or DSLR-Amazon, we achieve more than 20% improvements over the next best algorithm without feature learning used in the comparison (from 49.5% to 71.6%, 49.4% to 70.4%, and 48.9% to 68.9%, respectively). It is worth noting that while we employ a generative approach for learning the feature, our method consistently achieves better performance than [12], even that this method uses discriminative training together with non-linear kernels. It is also clear from the table that the multi-layer DASH-N outperforms the single-layer DASH-N. In the case of adapting from Caltech to Amazon, the performance gain by using a combination of features obtained from both layers rather than just features from the first layer is more than 10% (from 60.3% to 71.6%).
The results obtained by using Hierarchical Matching Pursuit (HMP) [31], without performing domain adaptation, are also included in the comparison in order to better evaluate the improvements provided by the proposed approach. In order to extract features using HMP, a dictionary is learned separately for source and target domains in each layer. Sparse codes for data from the source and target domains are then computed using the corresponding dictionary. Similar to our approach, the classification is performed using the concatenated features obtained by a two-layer network. HMP parameters are selected using cross-validation on the training data. It can be seen from Table I that, although HMP does not perform as well as the proposed method, it achieves reasonably good performance on the dataset. In many cases, it even outperforms other domain adaptation methods used in the comparison. This demonstrates the effectiveness of learning feature representation. However, it is also clear from the table that by learning a common representation at each layer of the hierarchy, our algorithm is able to capture the domain shift better. As a result, it consistently achieves better classification rates compared to HMP in all scenarios.

In order to illustrate the encoding of features using the learned dictionary in the first layer, Figure 3 shows the responses of the training and testing data for the BACKPACK class with respect to each atom of the dictionary in the first layer for the pair DSLR-Webcam domains. The sparse codes for all the patches of the training and testing images belong to the class are computed. The absolutes of these sparse vectors are summed together and normalized to unit length. Small components of the normalized sparse codes are thresholded to better show the correspondences between the training and testing data. It can be seen from the figure that the sparse codes for the training and testing data for the BACKPACK class both have high responses in four different dictionary atoms (43, 103, 136 and 160).

2) 31 Classes and Multiple Sources: We also compare the recognition results for all 31 classes between our approach and other methods in both cases of single (Table II) and multiple source domains (Table III). It can be seen from Tables II and III that our results, even using only features extracted from the first layer, are consistently better than that of other algorithms in all the domain settings. This proves the effectiveness of the feature learning process. The performance of our algorithm further improves when combining features learned from both layers of the hierarchy. Especially in the case of adapting from Webcam and DSLR to Amazon, we achieve an improvement of more than 15% compared to the result of SDDL [12] (from 24.1% to 41.8%).

3) Dimensions of Latent Domains: Dimensions of latent domains are some of the important parameters affecting the performance of DASH-N. Figure 4a shows the recognition rates with respect to different dimensions of the latent domain in the first layer for three pairs of source-target domains (Amazon-DSLR, Caltech-Amazon and Webcam-DSLR), while keeping the dimension of latent domain in the second layer to 750. As the patch size is set at 5 × 5, we vary the dimension of the first layer dictionary from 5 to 25. It can be seen from the figure that if the latent domain dimension is too low, the accuracy decreases. The optimal dimension is achieved at 20.

Similarly, the recognition rates with respect to different dimensions of the second layer latent domain are shown in Figure 4b while the first layer latent dimension is kept at 20. It can be seen from Figure 4b that the curves for all three pairs of source-target domains peak at the dimension 750. Once again, we observe that the performance decreases if the dimension of the latent domain is too low. More interestingly, as we can observe for the pair Caltech-Webcam and Webcam-DSLR, setting the dimension of the latent domain too high is as detrimental as setting it too low. In all of our experiments, we set the dimension of the latent domain using the cross validation technique.

E. Half-toning and Edge

In order to evaluate the ability of DASH-N in adapting to a wide range of domains, we also perform experiments on object recognition from the original image domain to two new
domains generated by applying half-toning and edge extraction algorithms to the original images. Half-toning images, which imitate the effect of jet-printing technology in the past, are generated using the dithering algorithm in [46]. Edge images are obtained by applying the Canny edge detector [47] with the threshold set to 0.07.

Figure 5 is the visualization of the reconstructed dictionaries atoms at layer 1 when adapting the original images (source) to edge images (target). Reconstructed dictionaries are obtained by $D_1 = (P_i^*)D_1$, where $\dagger$ denotes the Moore-Penrose pseudo-inverse. We observe that the dictionary atoms of original images contain rather fat and smooth regions. In contrast, dictionary atoms of edge images have many thin and highly varying patterns that are more suitable for capturing edges.

Table IV shows the performance of different algorithms when adapting to these new domains. It can be seen that DASH-N outperforms other methods used in the comparison in both cases of half-toning and edge images. This proves the ability of our approach to adapt well to new domains. Both the source code and two new datasets will be released for research purposes.

VII. CONCLUSION

We have presented a hierarchical method for performing domain adaptation using multi-layer representations of images. In the proposed approach, the features and domain shifts are learned jointly in each layer of the hierarchy in order to obtain a better representation of data from different domains. Unlike the other hierarchical approaches, our method prevents the dimension of feature vectors from increasing too fast as the number of layers increase. Experimental results show that the proposed approach significantly outperforms the other domain adaptation algorithms used in the comparison.

Several future directions of inquiry are possible considering our new approach to domain adaptation and feature learning. It would also be of interest to incorporate non-linear learning frameworks to DASH-N.

APPENDIX

In this section, we provide the derivations for the forms of $D$ in (8) and $P_i$ in (9) as well as extend the optimization to the case of multiple source domains.

A. Form of $D$

We consider a general case where there are $m$ different domains. Let $\{Y_i, P_i, X_i, n_i\}_{i=1}^m$ be the training data, the transformation, the sparse coefficients, and the number of samples for the $i$-th domain, respectively. Let $D$ denote the common dictionary in the latent domain. The objective function that we want to minimize is

$$\sum_{i=1}^m \lambda_i \mathcal{L}(Y_i, P_i, D, X_i, \alpha, \beta)$$

$$= \sum_{i} \lambda_i \left( \|P_i Y_i - DX_i\|_F^2 + \alpha \|Y_i - P_i^T P_i Y_i\|_F^2 \right) + \beta \|X_i\|_1. \quad (23)$$

Let us first define

$$Z = [\sqrt{\lambda_1 P_1 Y_1}, \ldots, \sqrt{\lambda_m P_m Y_m}]. \quad (24)$$

One can write $D$ in the following form

$$D = D_∥ + D_\perp, \text{ where } D_∥ = ZB \text{ and } D_\perp^TZ = 0, \quad (25)$$

for some $B \in \mathbb{R}^{(\sum_i n_i) \times K}$. In other words, columns of $D_∥$ and $D_\perp$ are in and orthogonal to the column subspace of $Z$, respectively. Let $S \in \mathbb{R}^{K \times K}$ be a diagonal matrix with positive coefficients such that columns of $D_∥ = D_∥S$ have unit-norm. Since the columns of $D$ have unit-norm and $D = D_∥ + D_\perp$, the columns of $D_∥$ must have norms of no larger than 1. Therefore, in order for the columns of $D_∥$ to have norm 1, the diagonal coefficients in $S$ must be no less than 1. This gives us the following corollary

$$\|X_i\|_1 = \|SS^{-1}X_i\|_1 = \|S\| \|X_i\|_1 \geq \|X_i\|_1. \quad (26)$$
MODEL APPLICA

where \( \hat{X}_i = S^{-1}X_i \). In addition, we also have

\[
\|P_i Y_i - DX_i\|_F^2 = \|P_i Y_i - D_i X_i\|_F^2 + \|D - D_i\|_F^2.
\]

Using the two inequalities in (26) and (27), we can show that

\[
\sum_{i=1}^m \lambda_i \mathcal{L}(Y_i, P_i, \hat{D}, \hat{X}_i, \alpha, \beta)
\geq \sum_{i=1}^m \lambda_i \left( \|P_i Y_i - \hat{D}_i\|_F^2 + \alpha \|Y_i - P_i^T P_i Y_i\|_F^2 \\
+ \beta \|\hat{X}_i\|_1 \right) + \sum_{i=1}^m \lambda_i \mathcal{L}(Y_i, P_i, \hat{D}, \hat{X}_i, \alpha, \beta).
\]

This means that given any feasible solution \((D, X_i)\), we can find another feasible solution \((\hat{D}_i, \hat{X}_i)\), whose dictionary atoms normalized to unit-norm, that does not increase the cost function. Therefore, an optimal solution for \(D\) must be in form of \(\hat{D}_i\), which can be generally written as

\[
D = ZB = [\sqrt{\lambda_1} P_1 Y_1, \ldots, \sqrt{\lambda_m} P_m Y_m] B. \tag{29}
\]

B. Form of \(P_i\)

We perform the orthogonal decomposition of \(P_i\) as follows

\[
P_i = P_i \perp + P_i \parallel, \text{ where } P_i \perp Y_i = 0 \text{ and } P_i \parallel = (Y_i A_i)^T. \tag{30}
\]

In other words, rows of \(P_i \parallel\) and \(P_i \perp\) are in and orthogonal to the column subspace of \(Y_i\), respectively. For convenience of notation, let us define

\[
P = \begin{pmatrix}
P_1 & \cdots & 0 \\
\vdots & \ddots & \vdots \\
0 & \cdots & P_m
\end{pmatrix} \tag{31}
\]

\[
Y = \begin{pmatrix}
\sqrt{\lambda_1} Y_1 \\
\vdots \\
\sqrt{\lambda_m} Y_m
\end{pmatrix} \tag{32}
\]

\[
X = (\sqrt{\lambda_1} X_1, \ldots, \sqrt{\lambda_m} X_m). \tag{33}
\]
After simple algebraic manipulations, the cost function can be re-written as
\[
\sum_{i=1}^{m} \lambda_i \mathcal{L}(Y_i, P_i, D, X_i, \alpha, \beta) = \sum_{i=1}^{m} \lambda_i \left( \|P_i Y_i - |DX_i|_F^2 + \alpha \|Y_i - P_i^T P_i Y_i\|_F^2 \right) + \beta \|X_i\|_1.
\]

Removing all the terms independent of \(P\), we have
\[
\text{tr}\left( P_i^T Y_i (I - BX)(I - BX)^T - \alpha I \right) Y_i^T P_i^T.
\]
The objective function is independent of \(P_\perp\). Moreover, an optimal solution of \(P_\perp\) is given by the eigenvectors of
\[
\min_{P_\perp} \|P_\perp\|^2 F = \|Z(I - BX)(I - BX)^T - \alpha I\|_F^2.
\]

C. Optimization for Multiple Source Domains

The first term of (23) is
\[
\sum_{i=1}^{m} \lambda_i \|P_i Y_i - DX_i\|_F^2 \geq \|Z(I - BX)\|_F^2,
\]
where \(X = [\sqrt{\lambda_1} X_1, \ldots, \sqrt{\lambda_m} X_m]\). The second term of (23) can be written as
\[
\alpha \sum_{i=1}^{m} \lambda_i \|Y_i - P_i^T P_i Y_i\|_F^2 = \alpha \sum_{i=1}^{m} \text{tr}\left( K_i - Y_i^T P_i^T P_i Y_i \right)
= \alpha \text{tr}\left( \sum_{i=1}^{m} (K_i) - Z^T Z \right).
\]

After discarding all constant terms, the objective function in (23) is equivalent to
\[
\|Z(I - BX)\|_F^2 - \alpha \text{tr}(Z^T Z) + \beta \sum_{i=1}^{m} \lambda_i \|X_i\|_1.
\]

Solving for \(A_i\): First, we perform the eigen-decomposition \(K_i = V_i \Lambda_i V_i^T\). Then the eigen-decomposition of \(K\) is given by
\[
K = \Lambda V \Lambda V^T,
\]
where,
\[
V = \left( \begin{array}{ccc}
V_1 & \cdots & 0 \\
\vdots & \ddots & \vdots \\
0 & \cdots & V_m
\end{array} \right), \qquad \Lambda = \left( \begin{array}{ccc}
\sqrt{\lambda_1} \Lambda_1 & \cdots & 0 \\
\vdots & \ddots & \vdots \\
0 & \cdots & \sqrt{\lambda_m} \Lambda_m
\end{array} \right).
\]

Moreover, let us define \(G_i = \Lambda_i^{1/2} V_i^T A_i\). Then, the constraints become
\[
P_i P_i^T = A_i^T K_i A_i = G_i^T G_i = I.
\]

In order to solve for \(A_i\), we assume that \((B, X_i)\) are fixed. After removing all the terms independent of \(A_i\), and using (24) together with (39), the objective in (38) is equivalent to
\[
\|Z(I - BX)\|_F^2 - \alpha \text{tr}(Z^T Z) = \text{tr}\left( Z((I - BX)(I - BX)^T - \alpha I) Z^T \right) = \text{tr}\left( K((I - BX)(I - BX)^T - \alpha I) K \right)
= \text{tr}\left( (A^T V \Lambda^1/2 V^T (I - BX)(I - BX)^T - \alpha I) V \Lambda^1/2 V^T A_i \right)
= \text{tr}(G^T H G),
\]
where,
\[
G = \Lambda^{1/2} V^T A = \left( \begin{array}{ccc}
V_1 & \cdots & 0 \\
\vdots & \ddots & \vdots \\
0 & \cdots & V_m
\end{array} \right) \left( \begin{array}{ccc}
\sqrt{\lambda_1} \Lambda_1^T & \cdots & 0 \\
\vdots & \ddots & \vdots \\
0 & \cdots & \sqrt{\lambda_m} \Lambda_m^T
\end{array} \right)
\]
\[
= \left( \begin{array}{ccc}
A_1 \\
\vdots \\
A_m
\end{array} \right) = [\sqrt{\lambda_1} G_1, \ldots, \sqrt{\lambda_m} G_m].
\]

Then the solution of \(A_i\) can be obtained by first minimizing
\[
\min_{G_i} \text{tr}(G^T H G) \text{ s.t. } G_i^T G_i = I.
\]

This can be solved efficiently in the same way as for the case of two domains using the algorithm proposed by [44]. The solution for \(A_i\) of each domain is recovered simply by
\[
A_i = V_i \Lambda_i^{-1/2} G_i.
\]

Solving for \((B, X_i)\): We now assume that \(A_i\) are fixed. After discarding the terms independent of \((B, X_i)\) in (38), the objective function is re-written as
\[
\|Z - DX\|_F^2 + \beta \sum_{i=1}^{m} \lambda_i \|X_i\|_1.
\]

This is in the familiar form of dictionary learning problem. We use the online dictionary learning algorithm proposed by [39] to learn \((D, X_i)\). The sparse coding is done using the LASSO algorithm. After obtaining \(D\), the solution of \(B\) is obtained by
\[
B = Z^T D.
\]