DLID: Deep Learning for Domain Adaptation by Interpolating between Domains

Sumit Chopra SCHOPRA@RESEARCH.ATT.COM
Suhrid Balakrishnan SUHRID@RESEARCH.ATT.COM
Raghuraman Gopalan RAGHURAM@RESEARCH.ATT.COM
AT&T Labs Research, Building 103, Florham Park, NJ 07302

Abstract

In many real world applications of machine learning, the distribution of the training data (on which the machine learning model is trained) is different from the distribution of the test data (where the learnt model is actually deployed). This is known as the problem of Domain Adaptation. We propose a novel deep learning model for domain adaptation which attempts to learn a predictively useful representation of the data by taking into account information from the distribution shift between the training and test data. Our key proposal is to successively learn multiple intermediate representations along an "interpolating path" between the train and test domains. Our experiments on a standard object recognition dataset show a significant performance improvement over the state-of-the-art.

1. Problem Motivation and Context

Oftentimes in machine learning applications, we have to learn a model to accomplish a specific task using training data drawn from one distribution (the **source domain**), and deploy the learnt model on test data drawn from a different distribution (the **target domain**). For instance, consider the task of creating a mobile phone application for "image search for products"; where the goal is to look up product specifications and comparative shopping options from the internet, given a picture of the product taken with a user's mobile phone. In this case, the underlying object recognizer will typically be trained on a labeled corpus of images (perhaps scraped from the internet), and tested on the images taken using the user's phone camera. The challenge here is that the distribution of training and test images is not the same. A naively trained object recognizer, that is just trained on the training images and applied directly to the test images, cannot be expected to have good performance. Such issues of a mismatched train and test sets occur not only in the field of Computer Vision (Duan et al., 2009; Jain & Learned-Miller, 2011; Wang & Wang, 2011), but also in Natural Language Processing (Blitzer et al., 2006; 2007; Glorot et al., 2011), and Automatic Speech Recognition (Leggetter & Woodland, 1995).

The problem of differing train and test data distributions is referred to as Domain Adaptation (Daume & Marcu, 2006; Daume, 2007). Two variations of this problem are commonly discussed in the literature. In the first variation, known as *Unsupervised Domain Adaptation*, no target domain labels are provided during training. One only has access to source domain labels. In the second version of the problem, called *Semi-Supervised Domain Adaptation*, besides access to source domain labels, we additionally assume access to a few target domain labels during training[1].

Previous approaches to domain adaptation can broadly be classified into a few main groups. One line of research starts out assuming the input representations are fixed (the features given are not learnable) and seeks to address domain shift by modeling the source/target distributional difference via transformations of the given representation. These transformations lead to a different distance metric which can be used in the domain adaptation classification/regression task. This is the approach taken, for instance, in (Saenko et al., 2010) and the recent linear manifold papers of (Gopalan et al., 2011; Gong et al., 2012). Another set of approaches in this fixed representation view of the problem treats domain adaptation as a conventional semi-supervised learning (Bergamo & Torresani, 2010; Dai et al., 2007; Yang et al., 2007; Duan et al., 2012). These works essentially construct a classifier using the labeled source data, and

Appeared in the proceedings of the ICML 2013, Workshop on Representation Learning, Atlanta, Georgia, USA, 2013.

[1] Often, the number of such labelled target samples are not sufficient to train a robust model using target data alone.

impose structural constraints on the classifier using unlabeled target data.

A second line of research focusses on directly learning the representation of the inputs that is somewhat invariant across domains. Various models have been proposed (Daume, 2007; Daume et al., 2010; Blitzer et al., 2006; 2007; Pan et al., 2009), including deep learning models (Glorot et al., 2011).

There are issues with both kinds of the previous proposals. In the fixed representation camp, the type of projection or structural constraint imposed often severely limits the capacity/strength of representations (linear projections for example, are common). In the representation learning camp, existing deep models do not attempt to explicitly encode the distributional shift between the source and target domains.

In this paper we propose a novel deep learning model for the problem of domain adaptation which combines ideas from both of the previous approaches. We call our model ($DLID$): **D**eep **L**earning for domain adaptation by **I**nterpolating between **D**omains. By operating in the deep learning paradigm, we also learn hierarchical non-linear representation of the source and target inputs. However, we explicitly define and use an "interpolating path" between the source and target domains while learning the representation. This interpolating path captures information about structures intermediate to the source and target domains. The resulting representation we obtain is highly rich (containing source to target path information) and allows us to handle the domain adaptation task extremely well.

There are multiple benefits to our approach compared to those proposed in the literature. First, we are able to train intricate non-linear representations of the input, while explicitly modeling the transformation between the source and target domains. Second, instead of learning a representation which is independent of the final task, our model can learn representations with information from the final classification/regression task. This is achieved by fine-tuning the pre-trained intermediate feature extractors using feedback from the final task. Finally, our approach can gracefully handle additional training data being made available in the future. We would simply fine-tune our model with the new data, as opposed to having to retrain the entire model again from scratch.

We evaluate our model on the domain adaptation problem of object recognition on a standard dataset (Saenko et al., 2010). Empirical results show that our model out-performs the state of the art by a significant margin. In some cases there is an improvement of over 40% from the best previously reported results. An analysis of the learnt representations sheds some light onto the properties that result in such excellent performance (Ben-David et al., 2007).

2. An Overview of DLID

At a high level, the $DLID$ model is a deep neural network model designed specifically for the problem of domain adaptation. Deep networks have had tremendous success recently, achieving state-of-the-art performance on a number of machine learning tasks (Bengio, 2009). In large part, their success can be attributed to their ability to learn extremely powerful hierarchical non-linear representations of the inputs. In particular, breakthroughs in unsupervised pre-training (Bengio et al., 2006; Hinton et al., 2006; Hinton & Salakhutdinov, 2006; Ranzato et al., 2006), have been critical in enabling deep networks to be trained robustly.

As with other deep neural network models, $DLID$ also learns its representation using unsupervised pre-training. The key difference is that in $DLID$ model, we explicitly capture information from an "interpolating path" between the source domain and the target domain. As mentioned in the introduction, our interpolating path is motivated by the ideas discussed in Gopalan et al. (2011); Gong et al. (2012).

In these works, the original high dimensional features are linearly projected (typically via PCA/PLS) to a lower dimensional space. Because these are linear projections, the source and target lower dimensional subspaces lie on the Grassman manifold. Geometric properties of the manifold, like shortest paths (geodesics), present an interesting and principled way to transition/interpolate smoothly between the source and target subspaces. It is this path information on the manifold that is used by Gopalan et al. (2011); Gong et al. (2012) to construct more robust and accurate classifiers for the domain adaptation task. In $DLID$, we define a somewhat different notion of an interpolating path between source and target domains, but appeal to a similar intuition.

Figure 1 shows an illustration of our model. Let the set of data samples for the source domain $\mathcal{S}$ be denoted by $D_{\mathcal{S}}$, and that of the target domain $\mathcal{T}$ be denoted by $D_{\mathcal{T}}$. Starting with all the source data samples $D_{\mathcal{S}}$, we generate intermediate sampled datasets, where for each successive dataset we gradually increase the proportion of samples randomly drawn from $D_{\mathcal{T}}$, and decrease the proportion of samples drawn from $D_{\mathcal{S}}$. In particular, let $p \in [1, \ldots, P]$ be an index over the P datasets we generate. Then we have $D_p = D_{\mathcal{S}}$ for $p = 1$, $D_p = D_{\mathcal{T}}$ for $p = P$. For $p \in [2, \ldots, P-1]$, datasets D_p and D_{p+1} are created in a way so that the proportion of samples from $D_{\mathcal{T}}$ in D_p is less than in D_{p+1}. Each of these data sets can be thought of as a single point on a particular kind of interpolating path between $\mathcal{S}$ and $\mathcal{T}$.

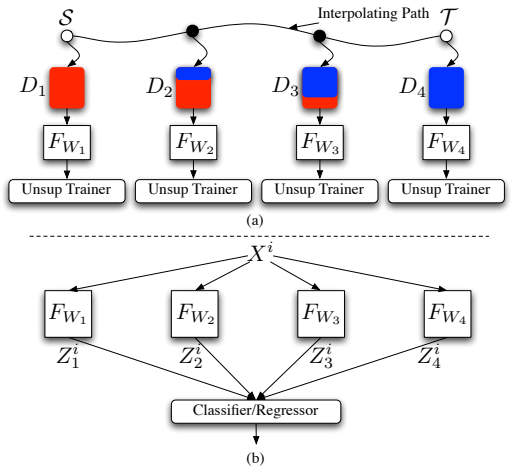

Figure 1. Overview of DLID model. (a) Schematic showing the interpolating path between the source domain $\mathcal{S}$ and target domain $\mathcal{T}$, consisting of the intermediate domains that we create. The figure also shows the unsupervised pre-trainer. (b) Once the feature extractors are trained, any input X^i is represented by concatenating the the outputs of all each the feature extractors. This representation is passed through classifier/regressor.

For each intermediate dataset D_p, we train a deep non-linear feature extractor F_{W_p}, with parameters W_p, in an unsupervised manner. Abstractly, the feature extractor F_{W_p} can be thought of as a parametric function that learns how to represent the inputs so that their most salient characteristics can be reconstructed from its outputs (imagine a non-linear denoising process). This provides a very powerful and flexible mechanism for representation learning as we can impose structural constraints on the form of the feature extractors F_{W_p} and then learn their optimal form under these constraints. We use the Predictive Sparse Decomposition (PSD) algorithm (Kavukcuoglu et al., 2008) to train F_{W_p}. See Figure 1(a) for schematic of this part of $DLID$. The details of the architecture of the F_{W_p} and PSD are discussed in the next section. Note that for any input sample X^i, each of the trained feature extractors generate hierarchical non-linear representations $Z_p^i = F_{W_p}(X^i)$, attuned to capturing salient information particular to the intermediate dataset D_p it is trained on. Once feature extractors corresponding to all points on the path are trained, any input sample is then represented by concatenating all of the outputs from the feature extractors together to create "path" features for the input. More precisely, the $DLID$ path representation Z^i for an input sample X^i is given by

$$\begin{aligned} Z^i &= [F_{W_1}(X^i)\ F_{W_2}(X^i) \ldots F_{W_P}(X^i)] \\ &= [Z_1^i\ Z_2^i \ldots Z_P^i]. \end{aligned}$$

Our hope is that this path representation will be more effective at domain adaptation because it is constructed to capture information about incremental changes between the source and target domains.

After we create the path representation of the inputs, the last stage of $DLID$ is the final classifier/regressor, which depends on the task at hand. This classifier/regressor is trained on the training pairs (Z^i, Y^i) generated from source domain data $D_{\mathcal{S}}$ by minimizing an appropriate loss function (Figure 1(b)).

Note that the entire $DLID$ model, from the intermediate feature extractors to the final classifier/regressor, is a deep neural network. This implies we can train *all* the parameters of the model together using standard gradient based techniques with the final loss (commonly referred to as fine-tuning). This ability to fine-tune in the light of a final classification/regression task confers huge advantages of $DLID$ over the previous approaches: there is usually a significant performance advantage that fine-tuned features provide as the the final features are optimized to the task at hand, new training data can be incorporated via fine-tuning etc.

Once the final classifier/regressor is fully trained, testing proceeds by forward propagating the inputX^o through the $DLID$ model to obtain final class probabilities/regression estimates (the representation Z^o is implicit in this step).

The architecture of the feature extractors F_{W_p}, and the associated unsupervised algorithm for their (pre-) training are crucial elements in the $DLID$ model. The best choices of these elements will be application domain and task dependent. For instance, when dealing with unstructured text, one might want to use a fully connected neural network, trained using the de-noising auto-encoder regime used in Glorot et al. (2011). For computer vision problems, convolutional neural networks seem a natural choice to capture the spatial correlation of images. Pre-training of the convolution networks could be done by a sparse coding algorithm (Ranzato et al., 2006; Kavukcuoglu et al., 2008). For speech applications, Time Delay Neural Networks (TDNN) might work best, since they can capture the temporal structure in an audio signal, and so on. In the next section we detail the architecture of the feature extractors we use for our object recognition case study.

3. Deep Representation Learning

We use convolutional networks for the feature extractors in our object recognition task (LeCun et al., 1998). Convolutional networks were one of the first deep networks that were reliably trainable and have had widespread success in a large number of computer vision applications. The architecture for our feature extractor is similar to the one proposed by Jarrett

et al. (2009). Each stage of the hierarchical feature extractor is composed of four components cascaded together, namely, Filter Bank Units (FB), Rectification Units (R), Contrast Normalization Unit (N), and Boxcar Pooling Unit (P). Figure 2 illustrates this single stage cascade. Each component in the cascade has a different purpose. For their detailed description and purpose see Jarrett et al. (2009).

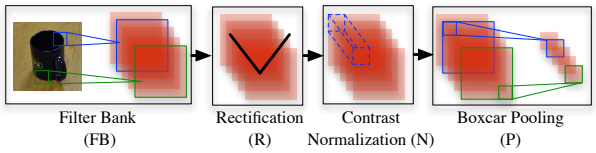

Figure 2. A single stage feature extractor $F^{S_1}_{W_p} : FB - R - N - P$ generated by cascading four basic units, namely Filter Bank, Rectification, Contrast Normalization, and Boxcar Pooling. A two stage feature extractor $F^{S_2}_{W_p}$ is built by cascading the outputs of one single stage feature extractor to another single stage feature extractor.

We denote a single stage of the feature extractor, by $F^{S_1}_{W_p}$, and it is composed of these four units cascaded together ($F^{S_1}_{W_p} : FB - R - N - P$). The learnable parameters W_p of the feature extractor are the filter weights k_{ij} and associated gain coefficients g_j of the FB unit. We also use two-stage feature extractors $F^{S_2}_{W_p}$, which we create by cascading two single stage feature extractors together. Thus, $F^{S_2}_{W_p} : FB_1 - R - N - P - FB_2 - R - N - P$, with the filters and gain coefficients of both FB_1 and FB_2 being the learnable parameters. The hyper-parameters associated with the feature extractors, such as, the size of the filters, the number of output feature maps generated by each stage, the connections from input to output feature maps etc, are dependent on the dataset and should be chosen using cross validation. The values we use in our experiments are reported in Section 5.

Having described the architecture of our convolutional network, we now describe the algorithm we use to train its parameters (the filter weights k_{ij} and the gain coefficient g_j of the FB units).

3.1. Unsupervised Pre-Training using PSD

As has become standard in training deep models, we train each layer of our network separately and in an unsupervised manner (only using unlabeled data). The unsupervised taining algorithm we use per layer is called Predictive Sparse Decomposition (PSD) (Kavukcuoglu et al., 2008; Jarrett et al., 2009).

When training the single stage feature extractor $F^{S_1}_{W_p}$, the input X to the PSD is an image patch equal to the size of the filters. When training the second stage of the two stage feature extractor $F^{S_2}_{W_p}$, we first train the the first stage filter bank FB_1 and then use its outputs as the input to the second stage (this is the layer wise unsupervised training we referred to earlier).

Also note that even though we train the filter bank on a collection of patches (of size is equal to the size of the filters), the trained filters can be applied directly to the whole image to generate its feature representation. This is achieved by convolving the filters of the filter bank with the entire image.

4. DLID Case Study

Now that our $DLID$ model is fully specified, in this section we describe the experiments we perform to gauge $DLID's$ performance on an object recognition task across multiple domains.

We use the dataset provided in Saenko et al. (2010)[2], which consists of images of 31 object categories from three different domains. The categories of objects consists of the commonly used objects, in a typical house/office environment, such as, mugs, notebooks, (computer) monitors etc. The three domains are, Amazon (product images of the objects collected from the www.amazon.com website), Dslr (images of the objects taken with a Dslr camera), and Webcam (images of objects taken with a webcam). There are a total of 4110 images with an average of 90 images per class for Amazon, 16 images per class for Dslr, and 26 images per class for Webcam. See Figure 3 for sample images from the three domains. The images highlight the difficulty of the object recognition task and the challenges of domain adaptation. Amazon images in particular are quite distinct from the other two domains: for the most part the Amazon objects are better lit, well framed, have little to no background clutter and are far more visually appealing than images from other two domains. This should come as no surprise since the Amazon images are typically professionally taken photographs of various products for sales purposes. The images from Dslr and Webcam have inconsistent lighting, low resolution and significantly more background clutter.

We minimally pre-process the images to create the input for $DLID$: 1) We convert the images from color to gray scale. 2) We re-scaled all images to size 90×90 pixels. 3) Each pixel of the image was subtracted by the mean pixel value of that image, and the result was divided by the standard deviation of the pixel values (a form of image normalization). 4) Contrast Normalization (N) was applied to the resulting image, with a neighborhood window size of 9×9. This process highlights the edges, and texture in an image. After taking into account the border effects, these steps resulted in

[2]Available from the first author's website

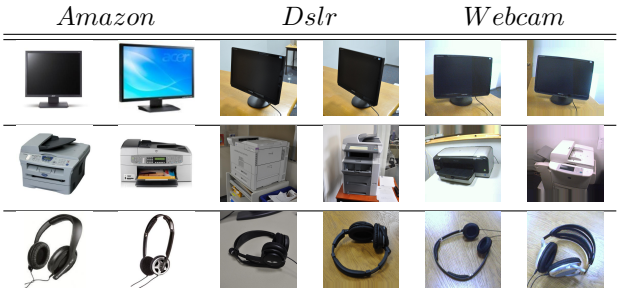

Figure 3. Example images for the objects in the Monitor category, Printer, and Headphones categories.

an image of size 82×82.

4.1. Unsupervised Pre-training

We now walk through the unsupervised pre-training procedure for $DLID$. For clarity, we consider a running example with Amazon as the source domain ($\mathcal{S}$) and Webcam as the target domain ($\mathcal{T}$) for the remainder of this section. Starting with the 82×82 pre-processed images from Amazon and Webcam, we created three data sets D_1, D_2, and D_3. Dataset D_1 constituted of only the Amazon images, D_2 consisted of Amazon and Webcam images, and D_3 consisted of only Webcam images. In other words we considered just one intermediate dataset on the interpolating path between $\mathcal{S}$ and $\mathcal{T}$. For each of these datasets $D_i : i = \{1, 2, 3\}$, we trained one single stage feature extractor $F^{S_1}_{W_i}$ and one two stage $F^{S_2}_{W_i}$ feature extractor.

Single Stage $F^{S_1}_{W_i}$: There are a total of 64 filters of size 7×7 in the FB Unit. Each filter is connected to the single input feature map (patch of a grayscale pre-processed image), generating 64 output feature maps. The size of the neighborhood window of N (Contrast Normalization Unit) is set to 7×7. The window size of P (Boxcar Pooling Unit) is 8×8, with a step size of 4. The filters of each $F^{S_1}_{W_i}$ are trained using the PSD algorithm with the data D_i. The training inputs X^i to the algorithm are a collection of 7×7 patches sampled at random from various locations in the preprocessed images. The PSD sparsity coefficient $\lambda = 1$.

Two Stage $F^{S_2}_{W_i}$: The two stage feature extractor is composed of two layers cascaded together, with the first layer being identical to $F^{S_1}_{W_i}$. The output of the first layer is passed as an input to the second layer. The FB units of the second layer use filters of size 5×5. This unit takes as input a set of 64 feature maps and generates 256 output feature maps. Each output feature map is connected to 8 randomly chosen input feature maps, resulting in a total of 2048 $(= 256 \times 8)$ filters in the FB Unit. The neighborhood window of N is 5×5, and of P is 4×4 with a step size of 2. As mentioned earlier, the training of two stage feature extractors is performed layer-wise. For the first stage, $F^{S_1}_{W_i}$, we use the single stage training procedure described in the previous paragraph. For the second stage, we forward propagate the entire training dataset D_i through the pre-trained first layer feature extractor to generate input features for second layer. With these inputs, training for the second stage follows the same procedure as that for the first layer. The only difference lies in the inputs used: it is now a patch with 64 feature maps obtained from the outputs generated by the first layer. The PSD sparsity coefficient $\lambda = 1$.

We use a multinomial logistic regression classifier for the final object recognition task . The inputs to the classifier are the path representations of the images (the Z^is, which concatenate the three feature extractor outputs when applied to the input images). The output is the class probability vector of size 31.

4.2. Fine-Tuning of Feature Extractor

As we mentioned before, since $DLID$ is a deep neural network, we have the ability to fine-tune all the parameters (feature extractors as well as classifier parameters) with respect to the task (multi-class classification in our present case). We perform such fine-tuning by back propagating gradients from the classifier all the way to the feature extractors. This is a significant advantage of $DLID$ over the previously proposed solutions, since we are now able to learn intermediate representations of the input which by design work well with the task at hand.

5. Empirical Results

We depart slightly from the standard methodology of evaluating models on the Saenko et al. (2010) dataset. Instead of randomly sampling from the full dataset to generate 20 smaller train/test folds, we instead use the complete labeled source dataset as training data in the Unsupervised problem (the complete target dataset is the test data). We made this decision primarily to have more training data per domain adaptation task, and also for the convenience of only having to train on only one dataset. For the Semi-supervised dataset, we follow the protocol in Saenko et al. (2010) to select a small number of labeled training target domain images in addition to the labeled source images for training. The rest of the target samples serve as test data.

To place our results in the context of literature, we train a baseline model and a state-of-the-art model on our dataset. The state-of-the-art model trained is the Geodesic Flow Kernel method (GFK) (Gong et al., 2012), using the code provided by the authors.

For each of the three pairs of domains, Tables 1 and 2

Table 1. Recognition accuracies on target domains in unsupervised domain adaptation setting. A: Amazon, W: Webcam, D: Dslr

Models	A → W	D → W	W → D
Baseline Saenko	21.76	55.40	60.84
GFK (PCA,PCA)	10.94	42.52	45.18
GFK (PLS, PCA)	12.20	39.50	43.78
Deep Feature Extraction			
$DL - S^1$	16.98	58.74	71.08
$DL - S^2$	22.40	64.65	78.31
$DL - S^1 - FT$	18.11	62.14	76.71
$DL - S^2 - FT$	23.65	66.42	82.53
Deep Intermediate Feature Extraction			
$DLID - S^1$	17.10	60.00	73.89
$DLID - S^2$	23.27	67.54	80.21
$DLID - S^1 - FT$	18.36	63.52	77.71
$DLID - S^2 - FT$	**26.13**	**68.93**	**84.94**

Table 2. Recognition accuracies on target domains in semi-supervised domain adaptation setting.

Models	A → W	D → W	W → D
Baseline Saenko	34.93	67.92	70.62
GFK (PCA,PCA)	34.90	59.12	54.32
GFK (PLS, PCA)	34.33	57.69	55.06
Deep Feature Extraction			
$DL - S^1$	38.90	65.38	76.54
$DL - S^2$	42.31	69.94	80.49
$DL - S^1 - FT$	43.87	70.10	83.21
$DL - S^2 - FT$	44.87	75.21	84.94
Deep Intermediate Feature Extraction			
$DLID - S^1$	39.74	65.95	78.52
$DLID - S^2$	42.74	74.93	87.16
$DLID - S^1 - FT$	46.16	71.23	82.71
$DLID - S^2 - FT$	**51.85**	**78.21**	**89.88**

compares the object recognition accuracy on the target domain for the Unsupervised and Semi-Supervised domain adaptation setting respectively. The methods we report results on are:

Baseline: We have a very simple, yet strong baseline model. Starting with features generated by Saenko et al. (2010) (SIFT-based features), we train a multinomial logistic regression classifier on the source domain data and test it on the target domain. This model is denoted by "Baseline Saenko" in the result tables.

GFK: The Geodesic Flow Kernel method (GFK) (Gong et al., 2012), using the code provided by the authors. We train both GFK (PCA, PCA), and GFK (PLS, PCA) (see their paper for details) on our dataset and report the results. GFK is understood to be the state-of-the-art.

DL: In order to quantify the benefit of intermediate representation learning, we also train a single stage and a two stage feature extractor (Section 3) using the inputs from all the three domains, ignoring the notion of the interpolating path between the domains. The underlying philosophy behind this model is exactly the same as in Glorot et al. (2011). The only difference is in the form of the parametric feature extractor and the pre-training algorithm. The single stage and two stage versions of this model are denoted by $DL - S^1$ and $DL - S^2$. Their fine-tuned counterparts are denoted by $DL - S^1 - FT$ and $DL - S^2 - FT$.

DLID: $DLID - S^1$ refers to the $DLID$ model with single stage feature extractors $F^{S^1}_{W_p}$, pre-trained using the PSD algorithm (see Section 4). The same model with fine-tuned filters is denoted by $DLID - S^1 - FT$. Similarly, we denote the two stage feature extractor version of this model by $DLID - S^2$, and its fine-tuned counterpart by $DLID - S^2 - FT$.

We make a number of observations based on our results. First, our $DLID$ models perform significantly better than the baseline and the current state-of-the-art in both unsupervised and semi-supervised setting. Second, the baseline using Saenko's features is quite strong. In fact it gives better results than both the versions of GFK algorithms trained on the same dataset. Note that GFK was trained using the code provided by the authors. Third, the single layer models perform worse than their two layer counter parts. This is in agreement with literature and provides further justification for learning deep hierarchical representations instead of "shallow" representations. Finally, also in accordance with existing deep learning literature, fine-tuning helps improve the performance across the board. Thus the best performing model is $DLID - S^2 - FT$ by far, which shows improvements over the baseline by up to 40% in some cases.

6. Analysis of Representations

So why does DLID work? We try to explore reasons by performing an analysis on our learnt representations in the theoretical framework of Ben-David et al. (2007). In their paper, the authors bound the generalization error of classifying the target samples (in the Unsupervised setting) with five terms (Theorem 2 in Ben-David et al. (2007)). Of the five terms, two asymptote to zero as the number of source examples grow, and one another term simply encodes the difficulty of domain shift. Thus the only two terms in the bound which depend on the representation of the examples are: 1. The source generalization error (SGE), and 2. The $\mathcal{A}$-distance between the source and target distributions. Intuitively, the error bound implies that a representation will have a better predictive performance on the target domain if it satisfies two criteria simultaneously. First, the representation should enable high prediction

accuracy on the source domain data (SGE is small). Second, the representation should make it difficult to distinguish between examples from the source and target distributions ($\mathcal{A}$-distance between the two domains is small).

We conjecture that the excellent performance of *DLID* model is attributed to two key properties. First, by learning/fine-tuning the feature extractors we obtain predictively good representations, and thus affect the SGE term in the error bound. Second, by explicitly including the intermediate domains, the learnt representations are able to capture the structures/concepts that contain information from the source, target and intermediate domains. This results in a rich representation, which in some manner bridges the gap between the source and target distributions, hence decreasing the $\mathcal{A}$-distance between the two domains.

We examined both SGE and $\mathcal{A}$-distance for the unsupervised domain adaptation non-fine tuned results in section 5. However, since the $\mathcal{A}$-distance is NP-hard to approximate for arbitrary distributions, along the lines of Ben-David et al. (2007), we evaluate a proxy for it, called the proxy $\mathcal{A}$-distance (PAD). PAD is estimated by training a classifier to distinguish between samples from the source and the target domain (Ben-David et al., 2007; Glorot et al., 2011). Concretely, PAD is defined as PAD$= 2(1-2\epsilon(\mathcal{S},\mathcal{T}))$, where $\epsilon(\mathcal{S},\mathcal{T})$ is the generalization error of the binary classifier separating the source and target examples. Glorot et al. (2011) have shown that the standard application of deep learning for feature extraction, results in decreased SGE and increased PAD. Our experiments thus focus on how learnt intermediate representations fare in terms of these quantities.

Table 3. How representation affects the generalization error bound terms. A: Amazon, W: Webcam, D: Dslr. *DL*: Deep Learning model, *DLID*: Intermediate Deep Learning Model, S^1: single stage model, S^2: two stage model.

Task	Repr.	*DL*		*DLID*	
		SGE	PAD	SGE	PAD
A → W	S^1	0.403	1.88	0.403	1.87
A → W	S^2	0.355	2.00	0.336	1.79
D → W	S^1	0.419	0.74	0.366	0.78
D → W	S^2	0.380	0.71	0.303	0.78
W → D	S^1	0.373	0.78	0.308	0.86
W → D	S^2	0.289	0.70	0.265	0.85

The results are shown in Table 3. They lend support to our conjecture. According to the error bound, dominating points of pairs in the table (where both SGE and PAD for one model are lower than the corresponding SGE and PAD for another model) are expected to have better generalization performance. In every case where there is a dominant condition (for instance, incase of $DLID - S^2$ v/s $DL - S^2$ for A → W), our results in the previous section agree. It is harder to argue for non-dominating points, but nonetheless, several interesting observations can be made. First, in all the cases *DLID* improves SGE over *DL*. This is important because, as shown in the experimental results in (Ben-David et al., 2007), when there is no dominating condition, SGE appears to be the more critical term in determining target generalization error. We attribute credit for this to the intermediate representations, in that including them helps define predictively useful structures in the representation. Second, in A → W, it appears *DLID* also improves PAD relative to *DL*. As stated previously, we believe this is due to creating a representation that contains information from both the source and target domains. We also note that this table corroborates the need to go deep. Results from S^2 are expected to be far superior to S^1, which is borne out in our experiments.

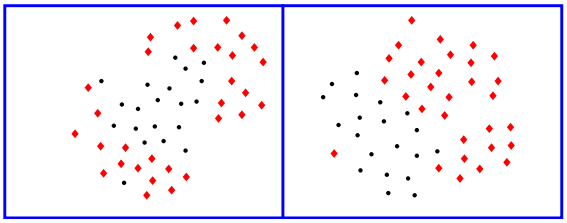

Figure 4. Schematic illustrating the quality of the learnt representations for the D → W domain pair. Shown are 2D projections of the learned representations from DL (left box) and DLID (right box) models, of all the letter tray (black circles) and mobile phone (red diamonds) images from the target domain (Webcam).

Finally, we also visually inspect the learnt representations by *DL* and *DLID* model, using the *tSNE* (van der Maaten & Hinton, 2008) algorithm, by projecting them onto a 2D space. Figure 4 shows the result of projecting all the test/target domain images for just two (out of the 31) classes on the D → W adaptation task. In this case, the *DLID* representations appear to be far better for classification.

7. Conclusion and Future Work

We present a model for domain adaptation where we learn the representation of the examples while trying to take into account domain shift. Our proposal is a contribution to deep learning methodology for domain adaptation which takes into account data distributions which are interpolated between the source and target data distributions. Our experiments on object recognition show excellent results and we shine light on the inner workings of our proposal, via an analysis in Ben-David et al. (2007)'s theoretical framework.

We envision many extensions to our work. One avenue would be to try to design a model that more directly trains using not just unsupervised pre-training, but also perhaps proxy $\mathcal{A}$-distance or its surrogate. Other application domains like NLP or speech seem ripe for exploration.

References

Ben-David, S., Blitzer, J., Crammer, K., and Pereira, F. Analysis of representations for domain adaptation. *Advances in Neural Information Processing Systems (NIPS)*, 2007.

Bengio, Y. Learning deep architecture for ai. *Foundations and Trends in Machine Learning*, 2(1):1 – 127, 2009.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. Greedy layer-wise training of deep networks. *Advances in Neural Information Processing Systems (NIPS)*, 2006.

Bergamo, A. and Torresani, L. Exploiting weakly-labeled web images to improve object classification: A domain adaptation approach. *Neural Information Processing Systems (NIPS)*, pp. 181 – 189, 2010.

Blitzer, J., McDonald, R., and Pereira, F. Domain adaptation with structural correspondence learning. *Emperical Methods on Natural Language Processing (EMNLP)*, 2006.

Blitzer, J., Dredze, M., and Pereira, F. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. *Association of Computational Linguistics (ACL)*, 2007.

Dai, W., Xue, G.R., Yang, Q., and Yu, Y. Transferring naive bayes classifiers for text classification. *Association for the Advancement of Artificial Intelligence (AAAI)*, 2007.

Daume, H. Frustratingly easy domain adaptation. *Association of Computational Linguistics (ACL)*, 2007.

Daume, H. and Marcu, D. Domain adaptation for statistical classifiers. *JAIR*, 26(1):101 – 126, 2006.

Daume, H., Kumar, A., and Saha, A. Co-regularization based semi-supervised domain adaptation. *Advances in Neural Information Processing Systems (NIPS)*, 2010.

Duan, L., Tsang, I., Xu, D., and Maybank, S. Domain transfer svm for video concept detection. *Computer Vision and Pattern Recognition (CVPR)*, pp. 1375 – 1381, 2009.

Duan, L., Tsang, I., and Xu, D. Domain transfer multiple kernel learning. *T-PAMI*, 34(3):465 – 479, 2012.

Glorot, X., Bordes, A., and Bengio, Y. Domain adaptation for large-scale sentiment classification: A deep learning approach. *International Conference on Machine Learning (ICML)*, pp. 97 – 110, 2011.

Gong, B., Shi, Y., Sha, F., and Grauman, K. Geodesic flow kernel for unsupervised domain adaptation. *Computer Vision and Pattern Recognition (CVPR)*, 2012.

Gopalan, R., Li, R., and Chellappa, R. Domain adaptation for object recognition: An unsupervised approach. *Internations Conference on Computer Vision (ICCV)*, 2011.

Hinton, G.E. and Salakhutdinov, R. Reducing the dimensionality of data with neural networks. *Science*, 313 (5768):504 – 507, 2006.

Hinton, G.E., Osindero, S., and Teh, Y. A fast learning algorithm for deep belief nets. *Neural Computations*, 18:1527 – 1554, 2006.

Jain, V. and Learned-Miller, E. Online domain adaptation of a pre-trained cascade of classifiers. *Computer Vision and Pattern Recognition (CVPR)*, pp. 577 – 584, 2011.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. What is the best multi-stage architecture for object recognition? *Internations Conference on Computer Vision (ICCV)*, 2009.

Kavukcuoglu, K., Ranzato, M., and LeCun, Y. Fast inference in sparse coding algorithms with applications to object recognition. Technical report, CBLL, Courant Institute, NYU, 2008.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278 – 2324, 1998.

Leggetter, C. and Woodland, P. Maximum likelihood linear regression for speaker adaptation of continuous density hidden markov models. *Computer Speech and Language*, 9(2):17 – 185, 1995.

Olshausen, B.A. and Field, D.J. Sparse coding with an over-complete basis set: A strategy employed by v1? *Vision Research*, 37:3311 – 3325, 1997.

Pan, S. J., Tsang, I.W., Kwok, J.T., and Yang, Q. Domain adaptation via transfer component analysis. *IEEE Trans. NN*, 99:1 – 12, 2009.

Ranzato, M., Poultney, C., Chopra, S., and LeCun, Y. Efficient learning of sparse overcomplete representations with energy-based model. *Advances in Neural Information Processing Systems (NIPS)*, 2006.

Saenko, K., Kulis, B., Fritz, M., and Darrell, T. Adapting visual category models to new domains. *Eurpoean Conference on Computer Vision (ECCV)*, 2010.

van der Maaten, L.J.P. and Hinton, G.E. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9: 2579 – 2605, 2008.

Wang, M. and Wang, X. Automatic adaptation of a generic pedestrain detector to a specific traffic scene. *Computer Vision and Pattern Recognition (CVPR)*, pp. 3401 – 3408, 2011.

Yang, J., Yan, R., and Hauptmann, A. Cross-domain video concept detection using adaptive svms. *ACM MM*, 2007.