Revisiting Batch Normalization For Practical Domain Adaptation

Yanghao Li†, Naiyan Wang‡, Jianping Shi⋄, Jiaying Liu†, and Xiaodi Hou‡

† Peking University ‡ TuSimple ⋄ SenseTime
lyttonghao@pku.edu.cn winsty@gmail.com shijianping5000@gmail.com liujiaying@pku.edu.cn xiaodi.hou@gmail.com

Abstract. Deep neural networks (DNN) have shown unprecedented success in various computer vision applications such as image classification and object detection. However, it is still a common (yet inconvenient) practice to prepare at least tens of thousands of labeled image to fine-tune a network on every task before the model is ready to use. Recent study [1] shows that a DNN has strong dependency towards the training dataset, and the learned features cannot be easily transferred to a different but relevant task without fine-tuning.

In this paper, we propose a simple yet powerful remedy, called Adaptive Batch Normalization (AdaBN), to increase the generalization ability of a DNN. Our approach is based on the well-known Batch Normalization technique [2] which has become a standard component in modern deep learning. In contrary to other deep learning domain adaptation methods, our method does not require additional components, and is parameter-free. It archives state-of-the-art performance despite its surprising simplicity. Furthermore, we demonstrate that our method is complementary with other existing methods. Combining AdaBN with existing domain adaptation treatments may further improve model performance.

Keywords: Domain adaptation; Batch normalization

1 Introduction

Training a DNN for a new image recognition task is expensive. It requires a large amount of labeled training images that are not easy to obtain. One common practice is to use a training set from a different source. For instance, one can borrow training data from an existing dataset, or query images from search engines and then label them using Amazon Turk. These approaches usually suffer from inferior performance due to dataset discrepancies, or “dataset bias”, because 1) the distributions of the source domains (third party datasets or Internet images) are often different from the target domain (testing images); and 2) DNN is particularly good at capturing dataset bias in its internal representation [3], which eventually leads to overfitting.

Known as domain adaptation, the effort to bridge the gap between training and testing data distribution has been discussed several times under the context
Yanghao Li†, Naiyan Wang‡, Jianping Shi⋄, Jiaying Liu†, and Xiaodi Hou‡

Fig. 1. Illustration of our proposed method. In training and testing, we use separate statistics of the output of each convolution or fully connected layer in source domain and target domain for batch normalization layer. The domain specific normalization mitigates the domain shift issue.

of deep learning [4,5,6,7]. Under common settings, an algorithm is provided with labeled data from source domain and unlabeled data from target domain. In order to make the connection between data from two domains, most of these methods require additional optimization steps and extra parameters. These extra computational burden could greatly complicate the training of a DNN which is already intimidating enough for most people.

In this paper, we propose a simple yet effective approach called AdaBN for batch normalized DNN domain adaptation. We hypothesize that the label related knowledge is stored in the weight matrix of each layer, whereas domain related knowledge is represented by the statistics of the Batch Normalization (BN) layer. Therefore, we can easily transfer the trained model to a new domain by modulating the statistics in the BN layer. This approach is straightforward to implement, has zero parameter to tune, and requires minimal computational resources. Moreover, our AdaBN is ready to extend to more sophisticated scenarios such as multi domain adaptation and semi-supervised settings. Fig. 1 illustrates the flow-chart of AdaBN. To summarize, our contributions are as follows:

1. We propose a novel domain adaptation technique called Adaptive Batch Normalization (AdaBN). We show that AdaBN can naturally dissociate bias and variance terms of a dataset, which is ideal for domain adaptation tasks.
2. We validate the effectiveness of our approaches on standard benchmarks for both single source and multiple sources domain adaptation. Our method outperforms the state-of-the-art methods.

2 Related Work

Domain transfer in visual recognition tasks has gained increasing attention in recent literature [8,9]. Often referred as covariance shift [10] or dataset bias [3], this problem poses great challenge to the generalization ability of a learned model. One key component of domain transfer is to model the difference between source and target distributions. In [11], the authors assign each dataset with an explicit bias vector, and train one discriminative model to handle multiple classification problems with different bias terms. A more explicit way to compute dataset difference is based on Maximum Mean Discrepancy [12]. It uses a non-linear mapping to project each data sample into a Reproducing Kernel Hilbert Space, and then compute the difference of sample means. To reduce dataset discrepancies, many methods are proposed, including sample selections [13,14], explicit projection learning [15,16,17] and principal axes alignment [18,19].

All of these methods face the same challenge of devising an effective domain transfer function in high-dimensional non-linear space. Due to computational constraints, most of the proposed transfer functions are in the category of simple linear projections.

In the field of deep learning, feature transferability across different domains is a tantalizing yet generally unsolved topic [20,1]. To transfer the learned representations to a new dataset, pre-training and fine-tuning [21] have become de facto procedures. However, adaptation by fine-tuning is far from perfect. It requires a considerable amount of labeled data from the target domain, and non-negligible computational resources to re-train the whole network.

A series of progress has been made in DNN to facilitate domain transfer. Early works of domain adaptation either focus on reordering fine-tuning samples [22], or regularizing MMD [12] in a shallow network [23]. It is only until recently that the problem is directly attacked under the assumption of unlabeled target domain and modern convolutional neural network architecture. Tzeng et al. [4] used the classical MMD loss to regularize the representation in the last layer of CNN. Long et al. [5] further extend the method to multiple kernel MMD and multiple layer adaptation. Ganin et al. [7] devised a gradient reverse layer to reverse the gradient that helps to distinguish the domains of each data sample. This layer efficiently anonymizes the domain information hidden in the CNN features. Recently, Tzeng et al. [6] proposed to simultaneously transfer task correlations and maximize domain confusion for (semi)-supervised domain adaptation.

Another related work is CORAL [24]. Different from our approach, this model focuses on the last layer of CNN. CORAL whitens both the data in source domain and target domain, and then re-correlate the source domain features to target domain. This operation aligns the second order statistics of source domain and
target domain distributions. Surprisingly, such simple approach yields state-of-the-arts results in various text classification and visual recognition tasks.

2.1 Batch Normalization Revisited

In this section, we briefly review Batch Normalization (BN) [2] which is closely related to our AdaBN. The BN layer is originally designed to alleviate the issue of internal covariance shifting — a common problem while training a very deep neural network. It first standardizes each feature in a mini-batch, and then learn a common slope and bias for each mini-batch. Formally, given the input to a BN layer \( \mathbf{X} \in \mathbb{R}^{n \times p} \), where \( n \) denotes the batch size, and \( p \) is the feature dimension, BN layer transform a feature \( j \in \{1 \ldots p\} \) into:

\[
\hat{x}_j = \frac{x_j - \mathbb{E}[\mathbf{X}_j]}{\sqrt{\text{Var}[\mathbf{X}_j]}}
\]

\[
y_j = \gamma_j \hat{x}_j + \beta_j,
\]

where \( x_j \) and \( y_j \) are the input/output scalars of one neuron response in one data sample; \( \mathbf{X}_j \) denotes the \( j^{th} \) column of the input data; and \( \gamma_j \) and \( \beta_j \) are parameters to be learned. This transformation guarantees that the input distribution of each layer remains unchanged across different mini-batches. For Stochastic Gradient Descent (SGD) optimization, a stable input distribution could greatly facilitate model convergency, leading to much faster training speed for CNN. Moreover, if training data are shuffled at each epoch, one training sample is transformed, or augmented differently in each epoch. This property acts as an additional regularization to combat against overfitting. In the testing phase, we use the global statistics instead of the statistics from one mini-batch to stabilize the testing results.

Since BN layer can both reduce internal covariant shift and overfitting, it significantly reduces the number of iteration to converge, and improve the final performance at the same time. BN layer has become a standard component in recent top-performing CNN architectures, such as deep residual network [25], and Inception V3 [26].

3 The Model

In this section, we first analyze the domain shift in deep neural network, and reveal two key observations in Sec. 3.1. Then in Sec. 3.2, we introduce our Adaptive Batch Normalization (AdaBN) method based on these observations. At last, we analyze our method in depth in Sec. 3.3.

3.1 A Pilot Experiment

Although the Batch Normalization (BN) technique is originally proposed to help SGD optimization, its core idea is about aligning training data from different
distributions. From this perspective, it is interesting to examine the BN parameters (batch-wise mean and variance) over different dataset at different layers of the network.

In this pilot experiment, we use MXNet implementation [27] of the Inception-BN model [2] pre-trained on ImageNet classification task [28] as our baseline DNN model. Our image data are drawn from [29], which contains the same classes of images from both Caltech-256 dataset [30] and Bing image search results. For each mini-batch sampled from one dataset, we first choose one layer, and then concatenate the mean and variance of each neuron to form a feature vector. Using linear SVM, we can perfectly classify whether the mini-batch feature vector is from Caltech-256 or Bing image search. Fig. 2 visualizes the distributions of mini-batch feature vectors in 2D, which further corroborate the observation that the BN statistics from different domains are clearly separated into clusters.

This pilot experiment suggests:

1. Both shallow layers and deep layers exhibit domain shift in DNN, thus only adapting the last layer is not enough. Deep adaptation is a must.
2. The statistics of BN layer is a good indication of domains. This confirms the hypothesis that the domain specific knowledge are stored in the statistics of BN layer.

![Shallow layer distributions](a) ![Deep layer distributions](b)

Fig. 2. t-SNE [31] visualization of the mini-batch BN feature vector distributions in both shallow and deep layers, across different datasets. Each point represents the BN statistics in one mini-batch. Red dots come from Bing domain, while the blue ones are from Caltech-256 domain. The batch size of each mini-batch is 64.

Both two observations motivate us to adapt the representation across different domains by BN layer.
3.2 Adaptive Batch Normalization

Given the pre-trained DNN model and a target domain, our Adaptive Batch Normalization (AdaBN) algorithm is as follows:

**Algorithm 1 Adaptive Batch Normalization (AdaBN)**

for neuron $j$ in DNN, image $m$ in target domain do

- Concatenate neuron responses on all images of target domain: $x_j = [x_j(m), ...]$.
- Compute the mean and variance of the target domain: $\mu_j = \mathbb{E}(x_j)$, $\sigma_j^2 = \sqrt{\text{Var}(x_j)}$.

end for

for neuron $j$ in DNN, testing image $m$ in target domain do

- Compute BN output $y_j(m) := \gamma_j \frac{(x_j(m) - \mu_j)}{\sigma_j^2} + \beta_j$.

end for

The intuition behind our method is very straightforward: The standardization of each layer by domain ensures that each layer received data from a similar distribution no matter in source domain or target domain, thus the domain information is anonymized.

For multi-source domain adaptation, the only change is we must standardize each sample by the statistics in its own domain. Since in training the statistics are calculated in one mini-batch, we only need to make sure that the samples in one mini-batch are from the same domain. While for (semi-)supervised domain adaptation, we may use the labeled data to fine-tune the weights as well. As a result, our method could fit in all different settings of domain adaptation with minor effort.

3.3 Further thoughts about AdaBN

One natural question to ask is whether such simple translation and scaling of features could deal with complicated domain shift? The answer is yes. As we all know, neural networks is a highly non-linear composite functions. Consider a simple neural network with input input $x \in \mathbb{R}^{p_1 \times 1}$. It has one BN layer with mean and variance of each feature being $\mu_i$ and $\sigma_i^2$, one fully connected layer with weight matrix $W \in \mathbb{R}^{p_1 \times p_2}$ and bias $b \in \mathbb{R}^{p_2 \times 1}$, and a non-linear transformation layer $f(\cdot)$. The output of this network is $f(W^T_a x + b_a)$, where

$$W_a = W^T \Sigma^{-1}, \quad b_a = -W^T \Sigma^{-1} \mu + b, \quad \Sigma = \text{diag}(\sigma_1^2, ..., \sigma_{p_1}^2), \quad \mu = (\mu_1, ..., \mu_{p_1}).$$

(2)

In practice we adopt an online algorithm [32] to efficiently estimate the mean and variance.
The output without BN is simply $f(W^T x + b)$. We can see that the transformation is highly non-linear even for a simple network with one computation layer. As CNN architecture goes deeper, it will gain increasing power to represent more complicated transformations.

Another question is why we transform the neuron responses independently, not decorrelate and then re-correlate the responses as suggested in [24]. We admit that decorrelation may improve the performance, however in the optimization of CNN the mini-batch size is usually smaller than the feature dimension. As a result, the covariance matrix is always singular. In addition, decorrelation requires to compute the inverse of the covariance matrix which is computationally intensive, especially if we plan to apply AdaBN for every layer of the network.

4 Experiments

In this section, we demonstrate the effectiveness of AdaBN on standard domain adaptation datasets, and empirically analyze the adapted features.

4.1 Experimental Settings

We first introduce our experimental settings on two standard datasets: Office [33] and Caltech-Bing [29], some baselines and the configurations of our experiments.

Office [33] is a standard benchmark for domain adaptation, which is a collection of 4652 images in 31 classes from three different domains: Amazon(A), DSRL(D) and Webcam(W). We evaluate all the six adaptation tasks in our experiments, which are commonly adopted by other domain adaptation methods [4,24,5]. For multi-source domain adaptation, we evaluate our method on the three transfer tasks (A, W) → D, (A, D) → W, (D, W) → W.

Caltech-Bing [29] is a much larger domain adaptation dataset, which contains 30,607 and 121,730 images in 256 categories from two domains Caltech-256(C) and Bing(B), respectively. The images in the Bing set are collected from Internet by Bing image search with the corresponding keywords, which has significantly different data distribution from that from Caltech-256.

We compare our method with a variety of methods, including three shallow methods: SA [18], GFK [19], CORAL [24], and three deep methods: DDC [4], DAN [5], RevGrad [7]. Specifically, GFK models domain shift by integrating an infinite number of subspaces that characterize changes in statistical properties from the source to the target domain. SA and CORAL align the source and target subspaces by explicit feature space transformations that would map source distribution into the target ones. DDC and DAN are the deep learning based methods which maximizes domain invariance by adding to AlexNet one or several adaptation layers using MMD. RevGrad incorporates a gradient reversal layer in the deep model to encourage learning domain-invariant features. It should be noted that these deep learning methods only add adaptation layers on several top layers of the DNN, in contrast that our method delves into early convolution layers as well with the help of the BN layers.
We follow the full protocol [21] for the single source setting; while for multiple sources setting, we use all the samples in the source domains as training data, and use all the samples in the target domain as testing data. We fine-tune the Inception-BN [2] model on source domain in each task for 100 epochs. The learning rate starts from 0.01, and then decreases by factor 0.1 every 40 epoch. Since the office dataset is quite small, following the best practice in [5], we freeze the first three groups of Inception modules, and set the learning rate of fourth and fifth group one tenth of the base learning rate to avoid overfitting. For Caltech-Bing dataset, we fine-tune the whole model with the same base learning rate.

4.2 Results

**Office Dataset** We report our results on Office in the single and multiple sources settings in Table 1 and Table 2, respectively. Note that the models in the first part of the Table 1 are pre-trained on AlexNet [34] instead of the Inception-BN [2] model, because there is no public pre-trained Inception BN model in Caffe [35]. Thus, the absolute number cannot be directly compared, the relative improvements over baseline is a more reasonable metric.

From Table 1, we first notice that the Inception BN [2] model indeed improves over the AlexNet [34] model on average, which means that the CNN pre-trained on ImageNet indeed learns general features, the improvements on ImageNet can be transferred on new tasks. Among the methods based on Inception BN features, our method improves the most over the baseline, which validates the need of deep adaptation. Moreover, since our method is complementary to other ones, we simply apply CORAL on the adapted features by our method. Not surprisingly, this simple combination indeed exhibits 0.5% increase in performance. This preliminary test reveals the further potential of our method if combining with other advanced domain adaptation methods. Finally, we could improve 1.7% over the baseline, and advance the state-of-the-art results for this dataset. Compared to other methods based on AlexNet, our method is better than DDC and RevGrad, and worse than DAN in terms of relative improvements over corresponding baselines.

Since all the compared methods do not report the performance on multi-source domain adaptation setting, here we only compare AdaBN with the best algorithm CORAL in the single source setting. Analyzing the results of the baseline in Table 2, we find that simply combining two domains will not lead to better performance. The result is generally worse than the higher one of using each source domain separately. This phenomenon suggests that if we cannot properly cope with domain bias, the increase of training samples may be reversely harmful to the testing performance. This result confirms the necessity of domain adaptation. In this more challenging setting, our method still outperforms the baseline and CORAL on average. Again, when combining with CORAL, our method demonstrates further improvements. At last, our method archives 2.3% gain over the baseline.
Revisiting Batch Normalization For Practical Domain Adaptation

| Method       | A → W D → W W → D A → D D → A W → A Avg |
|--------------|------------------------------------------|
| AlexNet [34] | 61.6 95.4 99.0 63.8 51.1 49.8 70.1       |
| DDC [4]      | 61.8 95.0 98.5 64.4 52.1 52.2 70.6       |
| DAN [5]      | 68.5 96.0 99.0 67.0 54.0 53.1 72.9       |
| RevGrad [7]  | 67.3 94.0 93.7 - - - -                    |
| Inception BN [2] | 70.3 94.3 100 70.5 60.1 57.9 75.5 |
| SA [18]      | 69.8 95.5 99.0 71.3 59.4 56.9 75.3       |
| GFK [19]     | 66.7 97.0 99.4 70.1 58.0 56.9 74.7       |
| CORAL [24]   | 70.9 95.7 99.8 71.9 59.0 60.2 76.3       |
| AdaBN        | 74.2 95.7 99.8 73.1 59.8 57.4 76.7       |
| AdaBN + CORAL| **75.4** 96.2 99.6 72.7 59.0 **60.5** 77.2 |

Table 1. Single source domain adaptation results on Office-31 [33] dataset with standard unsupervised adaptation protocol.

| Method       | A, D → W A, W → D D, W → A Avg |
|--------------|---------------------------------|
| Inception BN [2] | 90.8 95.4 60.2 82.1           |
| CORAL [24]   | 92.1 96.4 61.4 83.3           |
| AdaBN        | 94.2 97.2 59.3 83.6           |
| AdaBN + CORAL| **95.0** 97.8 60.5 **84.4**   |

Table 2. Multi-source domain adaptation results on Office-31 [33] dataset with standard unsupervised adaptation protocol.

Caltech-Bing Dataset To further evaluate our method on the large-scale dataset, we show our results on Caltech-Bing Dataset in Table 3. Compared with CORAL, AdaBN achieves better performance, which improves 1.8% over the baseline. Note that all the domain adaptation methods show minor improvements over the baseline in the task C → B. We hypothesize the reason is that the images in Bing dataset are collected from Internet, which are more diverse and noisy. Thus, it is not easy to adapt on the Bing dataset from the relatively clean dataset Caltech-256.

| Method       | C → B | B → C | Avg |
|--------------|-------|-------|-----|
| Inception BN [2] | 35.1  | 64.6  | 49.9 |
| CORAL [24]   | **35.3** | 67.2  | 51.3 |
| AdaBN        | 35.2  | **68.1** | **51.7** |
| AdaBN + CORAL| 35.0  | 67.5  | 51.2 |

Table 3. Single source domain adaptation results on Caltech-Bing [29] dataset.
4.3 Empirical Analysis

In this section, we conduct two experiments to empirically analyze the features adapted by our method and another experiment to illustrate how the number of samples in target domain affects the performance.

**Visualization of Features.** We first visualize the features of the last layer before and after adaptation using t-SNE [31] in Fig. 3. We choose two adaption settings for illustration: Amazon to webcam and Amazon to DSLR. Each red circle represents one training sample, while each blue circle represents one testing sample. We can see that the features of testing data after adaption are blended more evenly with the training data compared to those without adaption. In other words, the distribution of testing samples is more consistent with the training one. This intuitive illustration again confirms that our method is effective against domain shift.

Fig. 3. Visualization of last layer features. Red circles are training samples, while blues ones are testing samples. Best viewed in color.
Analysis of Feature Divergence. In the second experiment, we analyze the statistics of the output of one shallow layer (the output of second convolution layer) and one deep layer (the output of last Inception module before ReLU) in the network. In particular, we compute the distance of source domain distribution and target domain distribution before and after adaptation. We denote each feature $i$ as $F_i$, and assume that the output of each feature generally follows a Gaussian distribution with mean $\mu_i$ and variance $\sigma_i^2$. Then we use the symmetric KL divergence as our metric:

$$D(F_i \| F_j) = \text{KL}(F_i \| F_j) + \text{KL}(F_j \| F_i),$$

$$\text{KL}(F_i \| F_j) = \log \frac{\sigma_j}{\sigma_i} \sigma_i^2 + \frac{\sigma_j^2 + (\mu_i - \mu_j)^2}{2\sigma_j^2} - \frac{1}{2}. \quad (3)$$

We plot the distribution of the distances in Fig. 4. Obviously, our method both reduce the domain discrepancy in shallow layer and deep layer. We also report the quantitative results in Table. 4. This experiment once more verifies the effectiveness of our proposed method.

![Distibution of the symmetric KL divergence](image)

**Fig. 4.** Distribution of the symmetric KL divergence of the outputs in shallow layer and deep layer, respectively. Best viewed in color.
Table 4. The average symmetric KL divergence of the outputs in shallow layer and deep layer, respectively.

|                  | A → W, shallow | A → W, deep | A → D, shallow | A → D, deep |
|------------------|----------------|-------------|----------------|-------------|
| Before Adapt     | 0.0716         | 0.0614      | 0.2307         | 0.0502      |
| After Adapt      | 0.0227         | 0.0134      | 0.0266         | 0.0140      |

Sensitivity to Target Domain Size. Since the key of our method is to calculate the mean and variance of the target domain on different BN layers, it is very natural to ask how many target images is necessary to obtain the stable statistics. Therefore, we randomly select a part of images in target domain to calculate the statistics and then evaluate the performance on the whole target images set in the task B → C and A → W, respectively. Fig. 5 illustrates the effect of using different number of batches. The results demonstrate that our method can obtain good results when using only a small part of the target examples. It should also be noted that even using one batch of target examples, our method still achieves better results than the baseline method. This is very valuable in practice use since a large number of targets are often not available.

Fig. 5. Accuracy when varying the number of mini-batches used for calculating the statistics of BN layers in A → W and B → C, respectively. For B → C, we only show the results of using less than 100 batches, since the results are very stable when adding more examples. The batch size is 64 in this experiment.

5 Conclusion and Future Works

In this paper, we have described an amazingly simple yet effective approach for domain adaptation for batch normalized neural networks. We have exploited another functionality of BN layer besides its original uses: domain adaptation. The main idea is to replace the statistics of each BN layer in source domain
Revisiting Batch Normalization For Practical Domain Adaptation 13

with those on target domain. The proposed method is easy to implement and parameter-free, and it takes almost no effort to extend to multiple source domains and semi-supervised settings. Moreover, our method is not sensitive to the target domain size. Thus it is more favorable for practitioners compared with other deep learning based domain adaptation methods. At last, we have tested our method on standard benchmarks. Our method established new state-of-the-art results on both single source and multiple source domain adaptation settings. We believe our method opens up a new direction for domain adaptation.

In contrary to other methods that use MMD or domain confusion loss to update the weights in CNN for domain adaptation, our method only modifies the statistics of BN layer. Therefore, our method is fully complementary to other existing deep learning based methods. It is interesting to see how they collaborate in one unified model.

References

1. Tommasi, T., Patricia, N., Caputo, B., Tuytelaars, T.: A deeper look at dataset bias. arXiv preprint arXiv:1505.01257 (2015) 1, 3
2. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: ICML. (2015) 448–456 1, 4, 5, 8, 9
3. Torralba, A., Efros, A.A.: Unbiased look at dataset bias. In: CVPR. (2011) 1521–1528 1, 3
4. Tzeng, E., Hoffman, J., Zhang, N., Saenko, K., Darrell, T.: Deep domain confusion: Maximizing for domain invariance. arXiv preprint arXiv:1412.3474 (2014) 2, 3, 7, 9
5. Long, M., Cao, Y., Wang, J., Jordan, M.: Learning transferable features with deep adaptation networks. In: ICML. (2015) 97–105 2, 3, 7, 8, 9
6. Tzeng, E., Hoffman, J., Darrell, T., Saenko, K.: Simultaneous deep transfer across domains and tasks. In: ICCV. (2015) 4068–4076 2, 3
7. Ganin, Y., Lempitsky, V.: Unsupervised domain adaptation by backpropagation. In: ICML. (2015) 1180–1189 2, 3, 7, 9
8. Beijbom, O.: Domain adaptations for computer vision applications. arXiv preprint arXiv:1211.4860 (2012) 3
9. Patel, V.M., Gopalan, R., Li, R., Chellappa, R.: Visual domain adaptation: A survey of recent advances. IEEE Signal Processing Magazine 32(3) (2015) 53–69 3
10. Shimodaira, H.: Improving predictive inference under covariate shift by weighting the log-likelihood function. Journal of statistical planning and inference 90(2) (2000) 227–244 3
11. Khosla, A., Zhou, T., Malisiewicz, T., Efros, A.A., Torralba, A.: Undoing the damage of dataset bias. In: ECCV. Springer (2012) 158–171 3
12. Gretton, A., Borgwardt, K.M., Rasch, M.J., Schölkopf, B., Smola, A.: A kernel two-sample test. The Journal of Machine Learning Research 13(1) (2012) 723–773 3
13. Huang, J., Gretton, A., Borgwardt, K.M., Schölkopf, B., Smola, A.J.: Correcting sample selection bias by unlabeled data. In: NIPS. (2006) 601–608 3
14. Gong, B., Grauman, K., Sha, F.: Connecting the dots with landmarks: Discriminatively learning domain-invariant features for unsupervised domain adaptation. In: ICML. (2013) 222–230 3
15. Pan, S.J., Tsang, I.W., Kwok, J.T., Yang, Q.: Domain adaptation via transfer component analysis. IEEE Transactions on Neural Networks 22(2) (2011) 199–210
16. Gopalan, R., Li, R., Chellappa, R.: Domain adaptation for object recognition: An unsupervised approach. In: ICCV. (2011) 999–1006
17. Baktashmotlagh, M., Harandi, M., Lovell, B., Salzmann, M.: Unsupervised domain adaptation by domain invariant projection. In: ICCV. (2013) 769–776
18. Fernando, B., Habrard, A., Sebban, M., Tuytelaars, T.: Unsupervised visual domain adaptation using subspace alignment. In: ICCV. (2013) 2960–2967
19. Gong, B., Shi, Y., Sha, F., Grauman, K.: Geodesic flow kernel for unsupervised domain adaptation. In: CVPR. (2012) 2066–2073
20. Yosinski, J., Clune, J., Bengio, Y., Lipson, H.: How transferable are features in deep neural networks? In: NIPS. (2014) 3320–3328
21. Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., Darrell, T.: DeCAF: A deep convolutional activation feature for generic visual recognition. In: ICML. (2014) 647–655
22. Chopra, S., Balakrishnan, S., Gopalan, R.: DLID: Deep learning for domain adaptation by interpolating between domains. In: ICML Workshop on Challenges in Representation Learning, Volume 2. (2013)
23. Ghifary, M., Kleijn, W.B., Zhang, M.: Domain adaptive neural networks for object recognition. In: PRICAI 2014: Trends in Artificial Intelligence. (2014) 898–904
24. Sun, B., Feng, J., Saenko, K.: Return of frustratingly easy domain adaptation. arXiv preprint arXiv:1511.05547 (2015)
25. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385 (2015)
26. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. arXiv preprint arXiv:1512.00567 (2015)
27. Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., Zhang, Z.: MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274 (2015)
28. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: ImageNet large scale visual recognition challenge. International Journal of Computer Vision 115(3) (2015) 211–252
29. Bergamo, A., Torresani, L.: Exploiting weakly-labeled web images to improve object classification: a domain adaptation approach. In: NIPS. (2010) 181–189
30. Griffin, G., Holub, A., Perona, P.: Caltech-256 object category dataset. (2007)
31. Van der Maaten, L., Hinton, G.: Visualizing data using t-sne. Journal of Machine Learning Research 9(2579-2605) (2008) 85
32. Donald, E.K.: The art of computer programming. Sorting and searching (1999) 426–458
33. Saenko, K., Kulis, B., Fritz, M., Darrell, T.: Adapting visual category models to new domains. In: ECCV. (2010) 213–226
34. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS. (2012) 1097–1105
35. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. In: ACM MM. (2014) 675–678