Towards a Deeper Understanding of Unsupervised Domain Adaptation

Brandon Leung

Section 1: Introduction

In supervised machine learning, a common paradigm is to use labeled data (the "training set") to learn the weights of some model, so that the model can not only produce accurate output labels for the training data points, but also generalize to new data not in the training set. In the past decade, with the introduction of fast GPUs and large datasets, tremendous progress in this area has been made with deep neural networks. However, one of the base assumptions is that the data used to train a model and the data seen at test time (when it is time to deploy the model) are drawn from the same underlying distribution.

Unfortunately, this is not always realistic in practice, because labeled data can be hard to obtain. Thus, we often find ourselves in a situation where we don't have labeled data in the distribution we're actually interested in (called the test distribution). In the test distribution, we only have unlabeled data. However, we do have labeled data in a similar distribution (called the target distribution). One way to address this issue is called Unsupervised Domain Adaptation (UDA). In this setting, we assume that during training, we have access to labeled data from our source distribution as well as unlabeled data from our target distribution. The goal is to somehow effectively leverage our labeled source data and unlabeled target data during training, so that at test time, our model can predict outputs for the target data well. UDA is supervised with regards to the source distribution, but unsupervised with regard to target distribution. For example, a common scenario is that the source distribution is composed of rendered 3D models, (which are easy to obtain since you can render as many lighting conditions, backgrounds, viewpoints as desired). We are interested in using these rendered images and applying them to the real-world image distribution, e.g. for a classification task. However, UDA is much more general than this and in principle can apply to any reasonable shift in distribution.

There has been much research in the UDA literature recently, with many proposed algorithms and methods. In Section 2 of this report, we precisely describe the UDA setting and give a literature review of the field, with a focus on methods tailored to image data. However, most can be generalized to other data types. In Section 2d, a new, unified taxonomy is proposed which serves to generalize most of the current UDA methods. This allows us to better understand the core principles used. Finally, in Section 3, we take a specific look into one of the current state of the art methods for UDA image classification in computer [1]. It uses a few techniques, such as the MMD distance metric, spherical k-means clustering, and psuedolabeling. We summarize the method and perform some experiments beyond what was presented in the original paper, which yield interesting insights into how it operates.

Section 2: Literature Review of UDA

The mathematical formulation of UDA is as follows. First, $D^S = \{(x_1^s, y_1^s), \ldots (x_{N_s}^s, y_{N_s}^s)\}$ are our source domain samples, where the x_i^s realizations of the random variable X^s and y_i^s is the ground truth label for x_i^s. For example, x_i^s could be an RGB image containing a dog while y_i^s is an integer mapped to the label "dog". Next, $D^T = \{(x_1^t), \ldots (x_{N_t}^t)\}$ are our target domain samples, where the x_i^t realizations of the random variable X^t. We don't have access to the ground truth value of y_i^t; our goal is to predict it. We assume that X^t and X^s are in the same feature space, and likewise, that y_i^t and y_i^s are in the same label space. However, X^t and X^s have a different distribution. Then, most current methods achieve UDA by one of two ways: 1) learning to map samples to a useful domain invariant feature space, or 2) learning a mapping function between domains. These two approaches to UDA, along with an overview of other significant works and considerations for the UDA literature, are discussed below.

Section 2a: UDA by Learning Domain Invariant Features

The most common and generally successful approach to UDA is to first learn some feature extractor(s) whose outputs are domain invariant (i.e. one cannot tell if the outputs are from the source domain or target domain), and then learn some task-specific model from the labeled source training data which takes the extracted features and outputs the desired label. Some methods, like use the same feature extractor (ie, sharing all the weights) for both source and target inputs. Meanwhile, other methods have different feature extractors for the source and target, but regularize/constrain them in some way. The other main design choice is the loss which enforces the output features to be domain invariant. The high-level idea is that the distribution of feature outputs from the model should be the same, regardless if the inputs are from the source distribution or the target distribution. Thus, no labels are necessary – we simply want to minimize some notion of distance in feature distributions. Some examples used in UDA are as follows:

- One simple way to perform UDA is to align the first-order and second-order statistics of the source and target distributions. CORAL [2] does this with linear algebra, by first whitening the source data and making it unit covariance. Then, the source is "recolored" with the target covariance. However, this requires SVD, which is not scalable – Deep CORAL [3] is an alternative by performing alignment in feature space by enforcing that the second order statistics of the extracted features be the same. Additionally, [4,5] formulate a geodesic version of Deep CORAL.

- One of the most prominent notions of distribution distance used in practice is the Maximum Mean Discrepancy (MMD). The general idea and results are as follows. Let $X \sim P$ be a random variable with domain Ω and distribution P. Suppose we have a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ with some kernel $k: \Omega \times \Omega \to \mathbb{R}$ defined as $k(x, x') = < \phi(x), \phi(x') >_{\mathcal{H}}$ where $\phi(x) = [\ldots \phi_i(x) \ldots]$ is some feature map $\phi: \Omega \to \mathcal{H}$, whose output can be infinite dimensional. The output of the kernel can be viewed as a measure of similarity between x and x'. Next, define $\mu_p = E_P[\phi(X)] = [\ldots E_P(\phi_i(X)) \ldots]$. That is, μ_p is the mean feature embedding for X; it is a point within the RKHS. Then, given two distributions P and Q, their MMD distance is defined as the distance between these feature means:

$$MMD^2(P,Q) = \left\|\mu_Q - \mu_Q\right\|_{\mathcal{H}}^2$$
$$= E_P k(X,X') + E_Q k(Y,Y') - 2E_{P,Q} k(X,Y).$$

Empirically, given datasets $x_1, \dots, x_n, where\ x_i \sim P$ and $y_1, \dots, y_m\ where\ y_i \sim Q$, this quantity can be estimated as:

$$\widehat{MMD}(P,Q) = \left\| \frac{1}{n}\sum_{i=1}^{n} \phi(x_i) - \frac{1}{m}\sum_{i=1}^{m} \phi(y_i) \right\|_{\mathcal{H}}^2$$
$$= \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} k(x_i,x_j) + \sum_{i=1}^{m}\sum_{j=1}^{m} k(y_i,y_j) - \frac{2}{mn}\sum_{i=1}^{n}\sum_{j=1}^{m} k(x_i,y_j).$$

This is the “feature view” of MMD. Furthermore, it can be shown that:

$$MMD(P,Q) = \|\mu_p - \mu_Q\| = \sup_{f\in\mathcal{F}} (E_P[f(X)] - E_Q[f(Y)])$$

where F is a unit ball in our RKHS. In other words, there is also a “functional” view of MMD which is equivalent to the feature view. A key result that if the kernel chosen for the RKHS is a characteristic kernel, then $MMD(P,Q) = 0 \Leftrightarrow P = Q$, because each distribution can be uniquely defined in the RKHS and all statistical moments are captured. For example, the gaussian kernel (also known as the RBF kernel) is a characteristic kernel. One advantage of the MMD is that it does not require density estimation (unlike the KL divergence).

The proofs of these results require a familiar understanding of topics in functional analysis, probability theory, statistic, and measure theory, so they have been omitted here but can be found in [6]. While there are many MMD-based UDA papers, we next go over some of the most popular ones. The MMD was first introduced as a two-sample test statistic to determine if samples were drawn from different distributions [6]. Then, this was first used in a UDA context by DDC [7]. Subsequently, the DAN [8] architecture extended DDC by introducing the MMD loss at multiple layers, and a selection strategy which can consider multiple kernel functions. On the other hand, JAN [9] aligns the joint distributions of multiple layers in the neural network through their joint MMD criterion. RTN [10] also uses the MMD loss, but relaxes the shared-classifier assumption an allow the source and target feature extractors differ by a residual function. CAN uses psuedolabels obtained by clustering to compute the on the class-conditional distribution MMD loss between the source and target domain. Since it represents the current state-of-the-art, a more detailed description of CAN is given in Section 4.

- Besides MMD, another notion of "distribution distance" derived from distribution classifiers has also seen success. Here, the main insight is two given distributions are similar if given some sample, it is difficult to classify and label the distribution that the sample belongs to. Conversely, given two distributions and samples drawn from those distributions, if it's easy to classify the samples then the two distributions are quite different (the distance is large). This was studied in the DANN paper [11], where a neural network classifies the domain given the output of the feature extractors. A minimax scenario is created where the domain classifier optimizes accuracy on the domain label while the feature extractor tries to reduce the accuracy of the domain classifier (by outputting domain-invariant features). In practice, this was achieved by a gradient reversal layer (GRL) during backpropagation. Other papers also successfully utilize this idea. For example, Domain Confusion [12] uses a confusion loss based on optimizing a domain classification loss and a domain confusion loss (in an iterative, alternating way) instead of the GRL. Meanwhile, ADDA [13] pretrains an encoder on the source domain and then fixes it when learning a domain-classifier minimizing feature extractor on the target domain, instead of the GRL.

- Some methods [14] use a measure of discrepancy based on optimal transport, which at a high level, measures the amount of "effort" required to change one distribution into another.

<u>Section 2b: UDA by Learning a Domain Mapping</u>

Another class of UDA methods aims to learn a transformation of the input data. For example, if we were able to learn a reasonable pixel-level transformation from the source distribution to the target distribution (ie, preserving the inherent content and label of images but changing the style)., then we could just transform all our source images to match images drawn from the target distribution Nearly all of these methods use a GAN to learn the transformation [15, 16]. These methods have found success on simpler datasets like MNIST, and provides an easy to understand, intuitive view into how it works (we are able to visually access the quality of the transformation). However, they generally struggle when it comes to more difficult datasets, because learning a realistic transformation in pixel-space is difficult. Indeed, it could be argued that the task of learning this transformation, to subsequently be applied to UDA, is more difficult than the original UDA task to begin with.

A related direction focuses on interpolating on the manifold between the source and target domain in feature space [17, 18, 19, 20]. Most of these are somewhat dated, except for [20] which is similar in sprit to the older methods but do the interpolation in pixel space using a CycleGan. So, this is essentially UDA by learning a domain mapping.

Section 2c: Other Significant UDA Works

Besides algorithms to achieving UDA, are also some other considerations for UDA:

- First, an important work by Ben-David et al. [21] provides a mathematical, theoretical basis on why UDA works. In particular, a bound on a classifier's error on the target domain based on the source domain error and the divergence of the two domains is presented.

- Another concern is the evaluation of the models – many works tune hyperparameters with a target validation set, and/or evaluate performance on a target test set. This is a somewhat controversial practice, since "pure" UDA assumes no labeled target data at all. Though research in this direction is still quite sparse, one work which addresses this well is [22] which estimates the target risk using the source risk in an unbiased way using importance weights, computed with a domain classifier network.

- Next, note that while there are many UDA models which can be easily generalized to many tasks, other papers present UDA models tailored to specific tasks. For example, [23] uses self-training and UDA for the task of object detection, while [24] uses curriculum learning and UDA for the task of urban scene semantic segmentation.

- There are also other settings similar to UDA. Domain generalization (DG) [25] can be viewed as a harder version of UDA which does not use any target data at all; the goal is to learn a robust, generalizable classifier that will perform well on unseen target data. Meanwhile, settings like open set DA [27] and Universal DA [26] allow that the source and target domains have different label spaces, by the introduction of an "unknown" class. However, UDA still remains the most widely studied setting while DG, Open Set DA, and Universal DA have received less attention.

In conclusion, there are many different methods to achieve UDA. While the more recent methods generally have better performance than older methods on the usual benchmark datasets, it is difficult to say which UDA method is objectively the "best". This is because the methods generally have different trade-offs in computational complexity, implementation complexity, and levels of current theoretical understanding. In practice, the best approach to picking a method to deploy is to try a few different modes which are close to state-of-the-art for the standard benchmark datasets and seeing which work best for your particular dataset/task/use case. In this regard, currently, choosing a UDA method for deployment it is partially an "art", partially a science.

Section 2d -- A Unified Taxonomy for Domain-Invariant Feature UDA Methods

In this section, a categorization scheme is proposed to organize the many different UDA methods. We focus on the methods which aim to find domain-invariant features (i.e. the methods as described in Section 2a), since those are the ones which have found the most success. For example, standard CORAL (not Deep CORAL) can be best understood as a preprocessing algorithm, and has been omitted. Similarly, methods which learn a domain mapping (as described in Section 2b) have also been omitted.

The categorization is summarized in Table 1. It was created based on the following observations:

- In general, there are three key choices that methods in this class need to make:
  - **Weight Sharing:** Some methods, share the same feature extractor (ie, sharing all the weights) for both source and target inputs. Meanwhile, other methods have different feature extractors for the source and target, but regularize/constrain them in some way.
  - **Loss Function:** Loss functions can be divided into two classes: MMD-based and Domain Classifier Based.
    - **MMD-Based:** Here, some variant of the MMD is used. They can change by the type of kernel used or the type of distribution (i.e. marginal, conditional, or joint) used to compute the empirical MMD.
    - **Domain-**Classifier: Here, we access the distribution distance by the ability of a classifier to assign a binary label to features: if the feature is from the source or target distribution. Methods differ by the way this is trained: through a minimax formulation, iteratively, or with a GAN-inspired loss.
  - **Class-Awareness:** This is a high-level characterization of models which addresses if the method aligns class-conditional distributions or not.
- While not explicitly said in the paper, it's easy to see that the loss used in DDC [7] is a linear kernel; that is, $K(x, y) = xy$ where $\varphi(x) = [x]$.
- CORAL [3] is actually equivalent to minimizing MMD with the polynomial kernel $K(x, y) = (1 + xy)^2 = 1 + 2xy + x^2y^2$, where $\varphi(x) = [1, x, x^2]$. Then the MMD loss is $MMD^2(P, Q) = \left\|\mu_p - \mu_q\right\|_{\mathcal{H}}^2$ where $\mu_p = E_{x\sim P}[\varphi(x)] = [1, E(X), E(x^2)]$.

Method Name	**Weight Sharing**	**Loss**	**Aligning Class?**
DDC [7]	Shared	MMD (Linear Kernel)	Class-Agnostic
DAN [8]	Unshared	MMD (multiple RBF kernels)	Class-Agnostic
JAN [9]	Shared	MMD (Joint, RBF kernel)	Class-Agnostic
RTN [10]	Shared	MMD	Class-Agnostic
Deep CORAL [3]	Shared	MMD w/ Poly. kernel	Class-Agnostic
DANN/ GRL [11]	Shared	Domain Classifier + Minimax	Class-Agnostic
Domain Conf. [12]	Shared	Domain Classifier + Iterative	Class-Agnostic
ADDA [13]	Unshared	Domain Classifier + GAN	Class-Agnostic
CAN [1]	Shared	MMD (on cond. Dist)	Class-Aware

Table 1. *A categorization of the different domain-invariant feature UDA methods.*

Section 4a – Contrastive Adaptation Network for UDA Image Classification

One current state of the art methods for UDA applied to image classification tasks is the Contrastive Adaptation Network (CAN) [1]. The main insight is that since we want to do classification, we should pay attention to aligning the class-conditional distributions, not just aligning the marginal distributions in a class-agnostic way as most other works have done [3, 7, 8, 9, 10, 11, 12, 13]. Suppose we have our source dataset $D^S = \{(x_1^s, y_1^s), \dots (x_{n_s}^s, y_{n_s}^s)\}$ and our target dataset $D^T = \{(x_1^t, \widehat{y_1^t}), \dots (x_{n_t}^t, \widehat{y_{n_t}^t})\}$. Usually, we don't have access to target domain labels $\widehat{y_i^t}$, but let us assume for now that we have some way to estimate them (the details of this are described later). Also, let $\phi(x)$ denote the feature output of some layer in a neural network. Then, we can compute an empirical estimate of the MMD between the distributions conditioned on classes c_1 and c_2 as

$\widehat{D}^{c_1c_2} = MMD(P(\phi(X^s)|Y^s = c_1), P(\phi(X^T)|Y^T = c_2)) = e_1 + e_2 - 2e_3$ where

$$e_1 = \sum_{i=1}^{n_s} \sum_{j=1}^{n_s} \frac{\mu_{c_1c_1}(y_i^s, y_j^s) k(\phi(\boldsymbol{x}_i^s), \phi(\boldsymbol{x}_j^s))}{\sum_{i=1}^{n_s} \sum_{j=1}^{n_s} \mu_{c_1c_1}(y_i^s, y_j^s)}$$

$$e_2 = \sum_{i=1}^{n_t} \sum_{j=1}^{n_t} \frac{\mu_{c_2c_2}(\hat{y}_i^t, \hat{y}_j^t) k(\phi(\boldsymbol{x}_i^t), \phi(\boldsymbol{x}_j^t))}{\sum_{i=1}^{n_t} \sum_{j=1}^{n_t} \mu_{c_2c_2}(\hat{y}_i^t, \hat{y}_j^t)}$$

$$e_3 = \sum_{i=1}^{n_s} \sum_{j=1}^{n_t} \frac{\mu_{c_1c_2}(y_i^s, \hat{y}_j^t) k(\phi(\boldsymbol{x}_i^s), \phi(\boldsymbol{x}_j^t))}{\sum_{i=1}^{n_s} \sum_{j=1}^{n_t} \mu_{c_1c_2}(y_i^s, \hat{y}_j^t)}.$$

Note that when $c_1 = c_2 = c$, this measures intra-class (within-a-class) domain discrepancy. We want to minimize this for all classes c to encourage domain-invariant features. Meanwhile, when $c_1 \neq c_2$, this becomes the inter-class (between class) domain discrepancy. We want to maximize this for all pairs of different classes c_1, c_2 in order to make the distributions farther away. This practice effectively increases the margin of the decision boundary in the space, making predictions more robust. These notions, along with a standard cross-entropy classification loss on the source data, are described in the following loss function:

$\mathcal{L} = \ell^{ce} + \beta\widehat{D}^{cdd}$

$$\hat{\mathcal{D}}^{cdd} = \underbrace{\frac{1}{M}\sum_{c=1}^{M} \hat{\mathcal{D}}^{cc}(\hat{y}_{1:n_t}^t, \phi)}_{intra} \underbrace{- \frac{1}{M(M-1)} \sum_{c=1}^{M} \sum_{\substack{c'=1 \\ c' \neq c}}^{M} \hat{\mathcal{D}}^{cc'}(\hat{y}_{1:n_t}^t, \phi)}_{inter},$$

$$\ell^{ce} = -\frac{1}{n_s} \sum_{i=1}^{n_s} \log P_\theta(y_i^s | x_i^s)$$

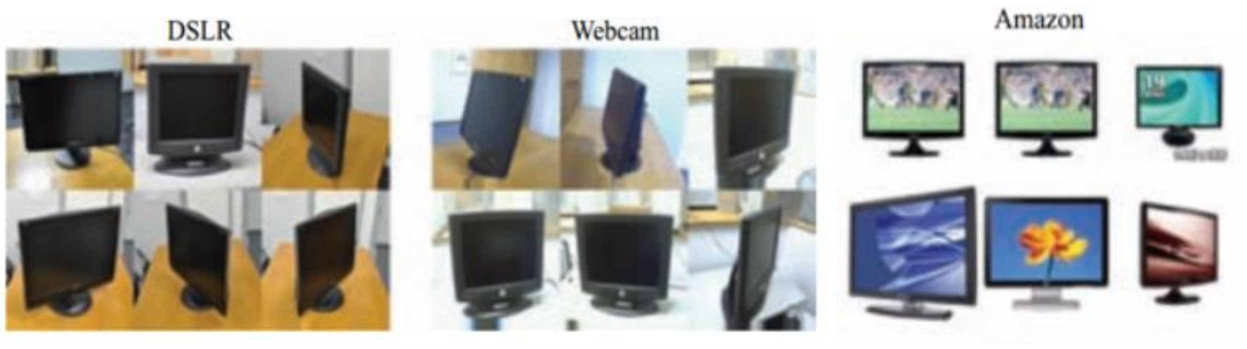


Figure 1. *Examples of images in the monitor class from the Office-31 dataset.*

where $P_\theta(y|x)$ is the output probability distribution for the predicted labels of an input x, from a CNN parameterized by parameters θ.

The missing part as mentioned earlier is actually estimating the target domain "psuedolabels" $\widehat{y_i^t}$. While this could be done naively by just using the output of the CNN, this would most likely lead to poor results due to noise. Instead, the CAN paper proposes a method based on spherical k-means. Here, "spherical" refers to use of the cosine dissimilarity $dist(a,b) = \frac{1}{2}\left(1 - \frac{a \cdot b}{\|a\| \, \|b\|}\right)$ as the metric to measure between points a, b in feature space for Lloyd's k-means algorithm. The algorithm is as follows:

- First, we forward $\{x_i^s\}$ through our feature extractor, and find the mean feature for each class.
- We initialize the target centers as the source class centers, and perform k-means on $\{x_i^t\}$ after forwarding those through our feature extractor
- We assign each sample x_i^t to the label corresponding the class of the nearest centroid
- Ambiguous data (far from their cluster centers) are filtered out based on a threshold
- Ambiguous classes (containing few target samples around their cluster center) are filtered out based on a threshold

This k-means algorithm to obtain psuedolabels is alternated with optimizing the network by backpropagation. In general, as training progresses the amount of ambiguous data and ambiguous classes decreases.

Section 4b – Contrastive Adaptation Network Experimental Results

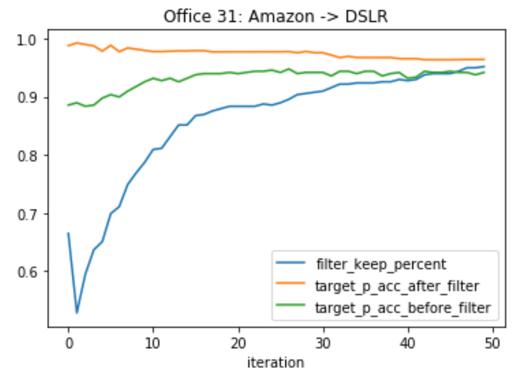
In this section we present some experiments beyond what was presented in the original CAN paper [1], which yield interesting insights into how it operates. In particular, we take a look at how the pseudolabel accuracy progresses during training. One could argue this is the most interesting aspect of the algorithm, since the MMD formulation has already been established in previous works, and their proposed loss is a small modification of it. Moreover, the method which is used to select psuedolabels is (spherical) k-means – this is a known NP-hard problem, and the standard algorithm (Lloyd's algorithm) is sensitive to initialization. Meanwhile, it is unclear if CAN's method of initialization (initialize the target centers as the source class centers) necessarily works well. Since the CAN UDA method relies so heavily on the performance of the k-means algorithm to do well, some experiments were done to look more closely into its performance.

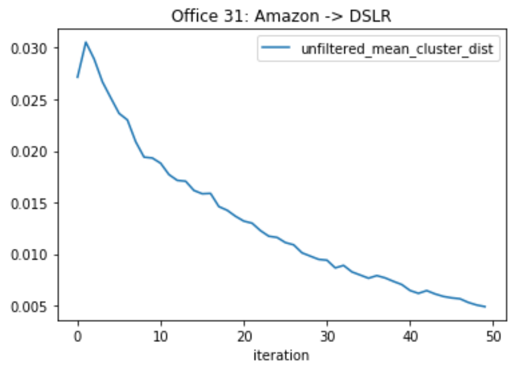
To do this, the Office-31 dataset was utilized (see Figure 1). This dataset has 3 domains over 31 different object classes. The classes are objects one would normally find in an office environment like monitors, bookcases, and printers. The three domains are Amazon (clean professionally shot images from Amazon.com), DSLR (objects taken with a high quality DSLR camera), and webcam (low quality images from a webcam camera). Then, several quantities were tracked during the training of CAN. These results can be found in Table 2. We train four different Source Domain -> Target Domain models: Amazon -> DSLR, DSLR -> Amazon, Amazon -> Webcam, and Webcam -> Amazon. For each, in the left column we track the percentage of target psuedolabels kept ("filter_keep_percent", in blue), the overall accuracy of the psuedolabels before filtering them ("target_p_acc_before_filter", in green), and the overall accuracy of the psuedolabels after filtering them ("target_p_acc_before_filter", in orange). In the right column, we track the mean distance from each point to their closest centroid (this is similar to the objective function for k-means). All of these measurements are taken each iteration of the algorithm (an iteration is a psuedolabeling step in addition to a backpropagation step), for 50 iterations of training. We also report the final accuracy on the target domain test set as a heading above each row of plots. Note that tracking these quantities is normally impossible to do, because in UDA we don't have the ground truth target labels. However, in our case because we are using a dataset meant for research, we can take a closer look into how CAN algorithm behaves with regards to the psuedolabeling.

From taking a look at the results, we confirm the intuition that training with a clean, simple source domain like Amazon and transferring to a more complicated domain (like Webcam or DSLR) leads to better performance than the other away around. When Amazon is the source domain, both the psuedolabel accuracy before filtering and the amount of psuedolabels kept steadily increases. The psuedolabel accuracy after filtering stays relatively constant at a high percentage throughout, which means that the filter is effectively doing its job. This also supports the method of centroid initialization used.

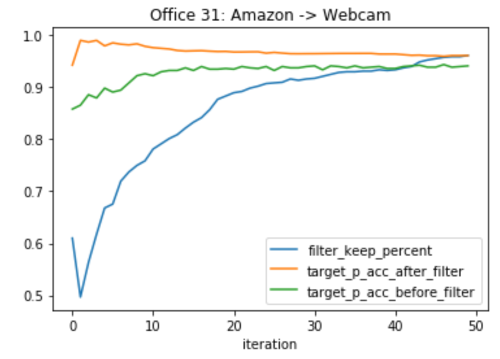
However, when using DSLR or Webcam as the source domain and using Amazon as the target domain, results are much different. We see that the filter is not working properly – it is constant at 100%, meaning that all psuedolabels are being used. As a result, the psuedolabel accuracy does not improve much after the 10th iteration. Finally, note that in all cases the k-means cost decreases to near zero. This indicates that the MMD-based loss is effective, in reducing the diameter o the clusters, even though the psuedolabels. In conclusion, these results suggest that more attention needs to be paid to the clustering algorithm used for psuedolabeling. It seems like algorithm is quite sensitive to the filter thresholds. Additionally, further work could be done on the k-means clustering algorithm, since it seems like it is currently prone to produce undesirable results.

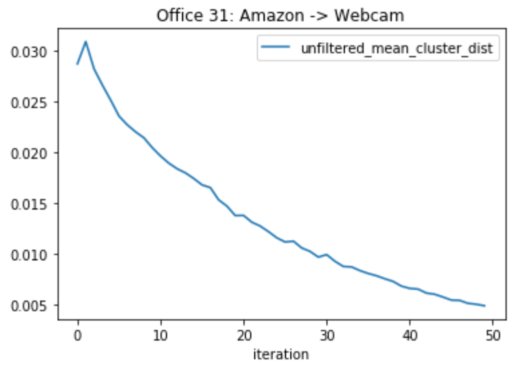
Soure Domain Amazon -> Target Domain DSLR (Final Target Acc 95)



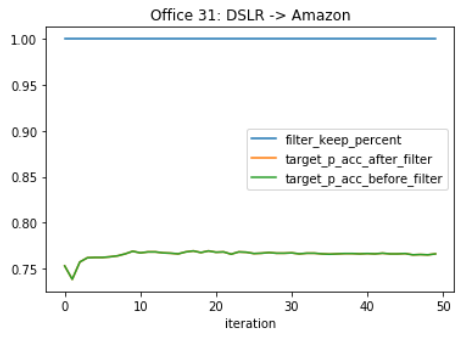


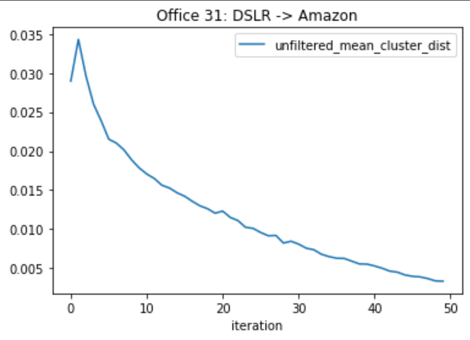
Soure Domain Amazon -> Target Domain Webcam (Final Target Acc 94. 5)



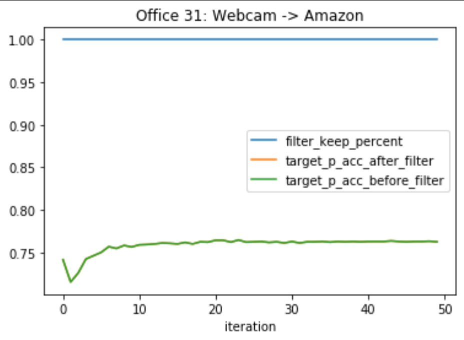


Soure Domain DSLR -> Target Domain Amazon (Final Target Acc 78)





Soure Domain Webcam -> Amazon Domain DSLR (Final Target Acc 77)



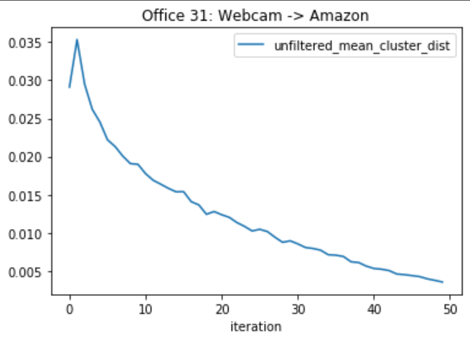


Table 2. *Results on the Office-31 dataset, which plot psuedolabel-related measurements as a function of iteration, during the training of CAN.*

References

[**1**] Kang, Guoliang, et al. "Contrastive adaptation network for unsupervised domain adaptation." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019.

[**2**] Sun, Baochen, Jiashi Feng, and Kate Saenko. "Return of frustratingly easy domain adaptation." *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[**3**] Sun, Baochen, and Kate Saenko. "Deep coral: Correlation alignment for deep domain adaptation." *European conference on computer vision*. Springer, Cham, 2016.

[**4**] Morerio, Pietro, Jacopo Cavazza, and Vittorio Murino. "Minimal-entropy correlation alignment for unsupervised deep domain adaptation." *arXiv preprint arXiv:1711.10288* (2017).

[**5**] Wang, Yifei, et al. "Deep domain adaptation by geodesic distance minimization." *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017.

[**6**] Gretton, Arthur, et al. "A kernel two-sample test." *Journal of Machine Learning Research* 13.Mar (2012): 723-773.

[**7**] Tzeng, Eric, et al. "Deep domain confusion: Maximizing for domain invariance." *arXiv preprint arXiv:1412.3474* (2014).

[**8**] Long, Mingsheng, et al. "Learning transferable features with deep adaptation networks." *arXiv preprint arXiv:1502.02791* (2015).

[**9**] Long, Mingsheng, et al. "Deep transfer learning with joint adaptation networks." *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017.

[**10**] Long, Mingsheng, et al. "Unsupervised domain adaptation with residual transfer networks." *Advances in neural information processing systems*. 2016.

[**11**] Ganin, Yaroslav, et al. "Domain-adversarial training of neural networks." *The Journal of Machine Learning Research* 17.1 (2016): 2096-2030.

[**12**] Tzeng, Eric, et al. "Simultaneous deep transfer across domains and tasks." *Proceedings of the IEEE International Conference on Computer Vision*. 2015.

[**13**] Tzeng, Eric, et al. "Adversarial discriminative domain adaptation." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017.

[**14**] Bhushan Damodaran, Bharath, et al. "Deepjdot: Deep joint distribution optimal transport for unsupervised domain adaptation." *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018.

[**15**] Sankaranarayanan, Swami, et al. "Generate to adapt: Aligning domains using generative adversarial networks." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

[**16**] Volpi, Riccardo, et al. "Adversarial feature augmentation for unsupervised domain adaptation." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

[**17**] Gong, Boqing, et al. "Geodesic flow kernel for unsupervised domain adaptation." *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012.

[**18**] Chopra, Sumit, Suhrid Balakrishnan, and Raghuraman Gopalan. "Dlid: Deep learning for domain adaptation by interpolating between domains." *ICML workshop on challenges in representation learning*. Vol. 2. No. 6. 2013.

[**19**] Gopalan, Raghuraman, Ruonan Li, and Rama Chellappa. "Domain adaptation for object recognition: An unsupervised approach." *2011 international conference on computer vision*. IEEE, 2011.

[**20**] Gong, Rui, et al. "DLOW: Domain flow for adaptation and generalization." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019.

[**21**] Ben-David, Shai, et al. "A theory of learning from different domains." *Machine learning* 79.1-2 (2010): 151-175.

[**22**] You, Kaichao, et al. "Towards accurate model selection in deep unsupervised domain adaptation." *International Conference on Machine Learning*. 2019.

[**23**] Kim, Seunghyeon, et al. "Self-training and adversarial background regularization for unsupervised domain adaptive one-stage object detection." *Proceedings of the IEEE International Conference on Computer Vision*. 2019.

[**24**] Zhang, Yang, Philip David, and Boqing Gong. "Curriculum domain adaptation for semantic segmentation of urban scenes." *Proceedings of the IEEE International Conference on Computer Vision*. 2017.

[**25**] Motiian, Saeid, et al. "Unified deep supervised domain adaptation and generalization." *Proceedings of the IEEE International Conference on Computer Vision*. 2017.

[**26**] You, Kaichao, et al. "Universal domain adaptation." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019.

[**27**] Panareda Busto, Pau, and Juergen Gall. "Open set domain adaptation." *Proceedings of the IEEE International Conference on Computer Vision*. 2017.