Unsupervised Domain Adaptation: A Reality Check

Kevin Musgrave
Cornell Tech

Serge Belongie
University of Copenhagen

Ser-Nam Lim
Meta AI

Abstract

Interest in unsupervised domain adaptation (UDA) has surged in recent years, resulting in a plethora of new algorithms. However, as is often the case in fast-moving fields, baseline algorithms are not tested to the extent that they should be. Furthermore, little attention has been paid to validation methods, i.e., the methods for estimating the accuracy of a model in the absence of target domain labels. This is despite the fact that validation methods are a crucial component of any UDA train/val pipeline. In this paper, we show via large-scale experimentation that 1) in the oracle setting, the difference in accuracy between UDA algorithms is smaller than previously thought, 2) state-of-the-art validation methods are not well-correlated with accuracy, and 3) differences between UDA algorithms are dwarfed by the drop in accuracy caused by validation methods.

1. Domain Adaptation Overview

Imagine the following scenario: you have a model that accurately classifies photos of animals, but you need the model to work on drawings as well. You have a collection of animal drawings, but no corresponding labels, so standard supervised training is not possible. Luckily, you can use unsupervised domain adaptation (UDA) to solve this problem. The goal of UDA is to adapt a model trained on labeled source data $S$ for use on unlabeled target data $T$. More precisely, the $i$th sample of dataset $S$ is

$$s_i = (S.images[i], S.labels[i])$$

and the $i$th sample of dataset $T$ is:

$$t_i = T.images[i]$$

Applications of UDA include semantic segmentation [41], object detection [21], and natural language processing [26]. There are also other types of domain adaptation, including semi-supervised [30], multi-source [24], partial [2, 22, 34], universal [49], and source-free [14]. In this paper, we focus on UDA for image classification, because it is well-studied and often used as a foundation for other domain adaptation subfields.

1.1. Common themes in UDA

In this section we provide a brief overview of ideas commonly used in UDA. See Table 1 for details of specific algorithms.

- **Adversarial** methods use a GAN where the generator outputs feature vectors. The discriminator’s goal is to correctly classify features as coming from the source or target domain, while the generator tries to minimize the discriminator’s accuracy.

- **Feature distance losses** encourage source and target features to have similar distributions.

- **Maximum classifier discrepancy** [33] methods use a generator and multiple classifiers in an adversarial setup. The classifiers’ goal is to maximize the difference between their prediction vectors (i.e., after softmax) for the target domain data, while the generator’s goal is to minimize this discrepancy.

- **Information maximization** methods use the entropy or mutual information of prediction vectors.

- **SVD losses** apply singular value decomposition to the source and/or target features.

- **Image generation** methods use a decoder model to generate source/target-like images from feature vectors, usually as part of an adversarial method.

- **Pseudo labeling** methods generate labels for the unlabeled target domain data, to transform the problem from unsupervised to supervised. This is also known as self-supervised learning.

- **Mixup augmentations** create training data and features that are a blend between source and target domains.
1.2. Validation methods in UDA

The assumption of UDA is that there are no target domain labels available, hence the name *unsupervised* domain adaptation. This raises the question of how to evaluate models for the purpose of selecting algorithms and checkpoints, and tuning hyperparameters. Without labels, we cannot compute the accuracy of our model as we normally would. One potential workaround is to manually label a few of the target samples, and then use just those labeled samples to compute accuracy. However, if any labeled target data is available, we should use that data to train our model, because some labeled data is better than none. But now we are entering the realm of semi-supervised domain adaptation. To be unsupervised, we have to assume there are zero target labels available. Thus, the best we can do is to use methods that serve as a proxy to target domain accuracy. This subject has gotten little attention, so there are only a few methods that have been proposed in the literature:

- **Reverse validation** [7, 52] consists of two steps. First it trains a model via UDA on $S$ and $T$, and uses this model to create pseudo labels for $T$. Next, it trains a reverse model via UDA on $T$ and $S$, where $T$ is the pseudo labeled target data, and $S$ is the “unlabeled” source data. The final score is the accuracy of the reverse model on $S$. One disadvantage of this approach is that it trains two models, doubling the required training time, but still producing only a single usable model.

- **Deep embedded validation (DEV)** [50] computes the classification loss for every source validation sample, and weights each loss by the probability that the sample belongs to the target domain. (The probability comes from a domain classifier trained on source and target data.) The final score is obtained using the control variates method. One practical issue with DEV is that its scores are unbounded. This is because part of the calculation uses $1/var(weights)$, so if the domain classifier creates weights with small or zero variance, the score will be very large or NaN.

- **Ensemble-based model selection (EMS)** [27] uses a linear regressor trained on 5 signals: target entropy, target diversity, Silhoutte & Calinski-Harabasz scores on the target features, source accuracy, and time-consistent pseudo-labels. EMS differs from other methods because it requires a dataset of $\{signal, ground truth accuracy\}$ pairs to train the regressor. These pairs have to be collected by training a model on a domain adaptation task that has labeled target data. After collecting the pairs and training the regressor, we still would not know if the regressor is accurate at predicting ground truth accuracy on our actual UDA task.
To better understand the state of UDA research, we looked at 49 papers accepted at top conferences (CVPR, ECCV, ICCV, ICLR, ICML, NeurIPS, and AAAI) from 2015-2021. Our main goals were 1) to see how papers present the performance gap between state of the art (SOTA) and baseline results and 2) to see what validation methods are used.

2.1. SOTA-baseline performance gaps

For each paper, we checked the results tables (if available) for Office31 [29] and OfficeHome [44], as they are among the most widely used datasets. Then for each transfer task, we compared the best performing algorithm with the two most commonly reported baselines: 1) ResNet50, which represents an ImageNet pretrained model that is fine-tuned on the source dataset (a.k.a source-only model), and 2) DANN, which is one of the seminal deep domain adaptation algorithms. Table 2 summarizes our findings.

2.2. Validation methods

To determine what validation methods are used, we looked at both the papers and their official code repositories (repos) if available. Table 3 shows that most repos use the oracle method, regardless of what validator (if any) is mentioned in the corresponding paper.

2.3. Discussion

From the previous two sections, we can conclude that when using the oracle validator, the latest UDA algorithms can outperform baselines like DANN by over 20 points. However, there are two issues with this conclusion. First, it is uncommon for papers to re-implement baseline methods, so they may have been tested only a few times over the years. A re-implementation and a thorough hyperparameter search might yield surprising results. Second, the reported accuracies are obtained using the oracle validator. A non-oracle method will result in a non-optimal selection of models, hyperparameters, and algorithms, thus leading to a drop in accuracy. If the drop in accuracy is significant, it may render negligible the differences between algorithms in the oracle setting. In other words, the efficacy of the validator may be more important than the relatively subtle differences between algorithms.

With this in mind, we ran a large scale experiment to find out how UDA algorithms really stack up against each other, and how non-oracle validators affect accuracy.

3. Experiment Methodology

In this section, we briefly describe our experiment setup. For more details about our methodology, please see the supplementary material.

---

1 https://github.com/KevinMusgrave/pytorch-adapt
2 https://github.com/KevinMusgrave/powerful-benchmarker

---

| Year | Office31 Source-only | Office31 DANN | OfficeHome Source-only | OfficeHome DANN |
|------|----------------------|---------------|------------------------|-----------------|
| 2016 | -                    | 2.2           | -                      | -               |
| 2017 | 12.5                 | 1.2           | -                      | 4.0             |
| 2018 | 23.4                 | 8.5           | 28.1                   | 11.5            |
| 2019 | 25.3                 | 12.4          | 29.3                   | 15.4            |
| 2020 | 23.9                 | 14.1          | 31.5                   | 17.2            |
| 2021 | 26.5                 | 15.7          | 32.5                   | 20.3            |

Table 2. The largest average SOTA-baseline performance gap per year. For example, the 2021 OfficeHome/DANN value of 20.3 is the gap on the Product→Art task, which is the task with the largest average SOTA-DANN gap for that year. Performance gap is measured as the absolute difference in accuracy.

| Validator       | # Papers | # Matches | # Repos |
|-----------------|----------|-----------|---------|
| full oracle     | 0        | -         | 30      |
| subset oracle   | 3        | 2         | 2       |
| src accuracy    | 0        | -         | 1       |
| src accuracy + loss | 2  | 0         | 0       |
| consistency + oracle | 0  | -         | 1       |
| target entropy  | 0        | -         | 1       |
| reverse validation | 2   | 0         | 0       |
| IWCV [39]       | 2        | 0         | 0       |
| DEV             | 2        | 0         | 0       |

Table 3. Validation methods in papers vs code. Out of 49 papers, 35 come with official repos. Of these 35 papers, 11 mention the validator that is used, and 2 use the same validator in both code and paper. 5 of the 6 papers that claim to use reverse validation, IWCV, or DEV, actually use oracle, and 1 uses target entropy.
Table 4. How the four splits are used. The target train set is used during UDA validation because overfitting is unlikely to happen, due to the difficult unsupervised nature of the task. The source train/val sets may also be used, depending on the validator. The target val set is used for testing, and represents data that is seen for the first time during model deployment.

### 3.1. Datasets

We ran experiments on 19 transfer tasks:

- **MNIST**: 1 task between MNIST and MNISTM [7].
- **Office31** [29]: 6 tasks between 3 domains (Amazon, DSLR, Webcam).
- **OfficeHome** [44]: 12 tasks between 4 domains (Art, Clipart, Product, Real).

MNIST and MNISTM are already divided into train/val splits, but Office31 and OfficeHome are not. So for each domain in these datasets, we created train/val splits with an 80/20 ratio per class (see Table 4).

### 3.2. Models

For the MNIST→MNISTM task, we used a LeNet-like model pretrained on MNIST as the trunk. For Office31 and OfficeHome, we used a ResNet50 [10] pretrained [45] on ImageNet [28], and finetuned this model on every domain. For every task, we started each training run using the model finetuned on the source domain (i.e. the source-only model).

### 3.3. Algorithms

We evaluated algorithms from 20 papers3, 12 of which are from 2018 or later. In addition to the DANN baseline mentioned in Section 2, we also benchmarked minimum entropy (MinEnt) [9], information maximization (IM) [36], and Information Theoretical Learning (ITL) [36]. All algorithms were implemented in PyTorch [23].

### 3.4. Validation methods

We ran experiments using four validation methods: oracle, IM, DEV [50], and SND [31]. The IM validator has the same definition as the IM UDA algorithm, but it uses the whole dataset rather than just a batch:

\[
IM = H\left(\frac{1}{N} \sum_{i=1}^{N} p_i\right) - \frac{1}{N} \sum_{i=1}^{N} H(p_i)
\]

where \(H\) is entropy, \(p_i\) is the \(i\)th prediction vector, and \(N\) is the size of the target dataset.

IM has been used as part of UDA algorithms [14, 36], but we are not aware of any paper that uses IM by itself as a general validation method. Robbiano et al [27] use IM as part of their EMS ensemble, and they also test the components (“diversity” and “entropy”) separately, but not the combination alone.

### 3.5. Hyperparameter search

In the oracle setting, we ran 100 steps of random hyperparameter search for each algorithm/task pair using Optuna [1], and trained four additional models using the best settings. This full search was run using two different feature layers: the output of the trunk model (“FL0”), and the penultimate classifier layer (“FL6”). We also tried DANN with the softmax layer as features (“FL8”).

For the non-oracle validators, we ran a similar hyperparameter search on 11 transfer tasks: MNIST, Office31, and four of the OfficeHome tasks (AP, CR, PA, and RC). We gathered 1.36 million datapoints, where a single datapoint consists of the validation score, source accuracy, and target accuracy collected from a validation step during training.

### 4. Results

#### 4.1. Accuracy in the oracle setting

Tables 6 and 7 present results obtained using the oracle validator. Each table cell is the average of 5 runs using the best settings from all FL0 and FL6 experiments. Bold indicates the best value per column, and better values have a stronger green color. White cells have accuracy equal to or less than the source-only model.

First note that our results are lower than typically reported. There are a few reasons for this:

- Our training sets are 20% smaller due to the creation of train/val splits. This has a big effect on Office31, which is already a small dataset.
- Our results are computed on the target validation set, which is never seen during training (see Table 4). In contrast, papers usually report accuracy on the target training set because no validation set exists.
- Our results use macro-averaged accuracy instead of micro-averaged. This combined with the choice of evaluation split can have a non-trivial effect on accuracy as shown in Table 5.

3At the time of our experiments, the ATDOC paper had a typo. See https://github.com/KevinMusgrave/pytorch-adapt/issues/10.
Next, we summarize the main takeaways of these results:

- The source-only model is a strong baseline for Office31 and OfficeHome. In fact, there are many cases where UDA degrades performance, as indicated by the white table cells.

- MinEnt, IM, ITL, and DANN are strong UDA baselines for Office31 and OfficeHome, often outperforming more complicated methods like MCD, CDAN, CORAL, and ATDOC.

- The SOTA-baseline performance gap is much smaller than typically reported (Figure 1 and Table 8).

- Some methods like DANN perform well on all three datasets. However, other methods perform poorly on MNIST, while scoring very highly on Office31 and OfficeHome, and vice versa. For example, MCC and BNM perform poorly on MNIST, but are the best on Office31 and OfficeHome. Likewise, STAR is among the best on MNIST, but among the lowest on OfficeHome.

### Table 5. The accuracy on train/val splits, using micro and macro averaged accuracy. The values shown are the average of averages across transfer tasks, of all methods that outperform the source-only model. For example, the OfficeHome Val Macro number is the average of all green cells in the Avg column of Table 7.

|          | Office31 | OfficeHome |
|----------|----------|------------|
| Train Micro | 86.7     | 67.9       |
| Train Macro | 87.2     | 66.7       |
| Val Micro  | 85.4     | 67.5       |
| Val Macro  | 85.7     | 66.5       |

![Reported numbers are the average from 2021 papers.](image)

Figure 1. Performance gaps between SOTA and baseline algorithms (source-only and DANN) on OfficeHome tasks. The reported numbers are the average from 2021 papers.

### 4.2. Impact of validation methods on accuracy

We first consider the “global” scenario in which validators are used to select model checkpoints, hyperparameters, and algorithms. Figures 4a-4c show the relationship between validation scores and target accuracy, using data from all transfer tasks. It appears that none of the methods are well-correlated with accuracy. (In fact, SND seems inversely correlated, which prompts us to add the negative SND score, NegSND, to our evaluation.) However, it is possible that the validators are well-correlated within tasks, and are just producing inconsistent scores across tasks (see Figure 3a). In addition, it may be possible to increase correlation by filtering out degenerate models.

Saito et al [31] suggest discarding models with low source accuracy, since they are unlikely to score well on target data. This brings us to Figures 4d-4f, which show that low source accuracy does indeed correspond with low target accuracy, though not vice versa. To determine a suitable threshold, we select the models with the best target accuracy for each transfer task, and take the average of their normalized source accuracies (i.e. normalized by the accuracy of the source-only model). The result is a normalized threshold of 0.98. Table 10 shows how validators perform
Table 7. Accuracy on the MNIST → MNISTM (MM) and OfficeHome transfer tasks. The Avg column is the OfficeHome average.

| Model     | Office31 | OfficeHome |
|-----------|----------|------------|
| Reported  | Source-only | 26.5 | 32.5 |
|           | DANN     | 15.7 | 20.3 |
| Ours      | Source-only | 16.4 | 12.7 |
|           | DANN     | 5.6   | 7.9  |
|           | DANN-FL8 | 8.0   | 5.3  |

Table 8. Average reported performance gap in 2021 papers vs ours. Each number corresponds with the transfer task with the largest performance gap.

with and without a 0.98 threshold. In most cases, the threshold significantly boosts accuracy. But even so, the validators are still a long way from matching the oracle. On most tasks, the drops in accuracy caused by the validators are still much larger than the SOTA-baseline performance gaps (see Figure 2). In other words, the poor performance of the val-
are collected per algorithm like Figure 2 and Table 10, the oracle and non-oracle accuracies and standard deviations are computed across transfer tasks. Un-
idicators, per algorithm, using a 0.98 source threshold. The mean Table 9. Performance gaps between oracle and non-oracle val-
the scope of the hyperparameter search, we used the same
idators is of much greater concern than the relatively small differences between UDA algorithms.
Now we consider the “local” scenario in which the val-
idator selects checkpoints and hyperparameters, but not algo-
rithms. In this case, some algorithm-validator pairs can work quite well, as shown in Table 9. However, many pairs have high variance, so it is difficult to know how reliable they will be when given a new transfer task.
Finally, we consider an unrealistic scenario in which we are able to discard models with low target accuracy. Figure 3b shows that even if we remove models with a target accuracy less than that of the source-only model, the validators’ correlations with accuracy are still below 0.3 on average.

5. Discussion

We have shown that the gap between SOTA and baseline UDA algorithms is smaller than previously thought. Furthermore, existing validators cause large drops in accuracy that make the differences between algorithms seem insignificant. In the scenario where the algorithm is already chosen, some algorithm-validator pairs can be effective, though most suffer from inconsistent performance across tasks. Consistency matters, because if a validator returns a high score, we need to be confident that the accuracy will also be high. Otherwise we will waste time and money that could be better spent on labeling the target data, eliminating the need for UDA altogether. Thus, one direction of research could be to create validators that work consistently well, even if they work with just a single UDA algorithm.

Limitations: To compare algorithms fairly, and to limit the scope of the hyperparameter search, we used the same optimizer, weight decay, learning rate (LR) scheduler, and batch size across all experiments. In addition, for any chosen LR, we applied the same LR to all models, which may not always be optimal. We believe we chose reasonable defaults, and we also allowed for plenty of flexibility in the weighting of loss terms for each algorithm (see the supplemental material). That said, it is possible that some algorithms require a different setting to reach their full potential.

Societal impact: Large scale machine learning experiments consume a great deal of energy. In our case, the end result is a better understanding of UDA, which is an area of central importance in the data efficiency agenda. As unlabeled data becomes available in new domains, UDA will allow for efficient reuse of existing models.

| Algorithm | IM  | DEV | SND | NegSND |
|-----------|-----|-----|-----|--------|
| AFN       | 3.4±3.1 | 9.6±13.8 | 17.0±21.0 | 7.5±12.5 |
| ATDOC     | 10.5±17.1 | 12.4±15.8 | 22.3±19.1 | 4.6±3.3 |
| BNM       | 4.7±3.9 | 7.3±12.1 | 15.4±21.7 | 6.1±2.9 |
| BSP       | 1.2±1.4 | 9.1±11.5 | 18.8±15.3 | 6.8±12.4 |
| CORAL     | 7.8±5.9 | 12.0±8.7 | 13.8±15.0 | 4.6±5.0 |
| DANN      | 9.3±6.9 | 5.5±6.5 | 11.5±14.4 | 7.5±7.6 |
| DC        | 5.0±2.9 | 3.0±2.6 | 12.1±14.0 | 7.1±7.6 |
| GVB       | 9.9±5.4 | 16.0±14.8 | 19.8±17.5 | 8.2±4.6 |
| JMMD      | 9.2±9.9 | 8.7±11.7 | 18.1±18.5 | 7.5±7.5 |
| MCC       | 6.4±3.2 | 3.3±1.9 | 11.6±18.0 | 7.2±3.0 |
| MCD       | 5.5±5.0 | 7.6±10.1 | 21.1±26.0 | 5.1±8.5 |
| MMD       | 8.9±7.1 | 9.6±12.1 | 20.6±15.1 | 8.7±11.0 |
| RTN       | 2.6±2.1 | 11.0±19.5 | 35.5±32.5 | 5.1±4.9 |
| SWD       | 10.0±10.3 | 10.7±15.7 | 17.7±13.7 | 5.9±7.3 |
| SymNets   | 10.1±9.0 | 8.8±8.9 | 57.8±30.8 | 14.7±14.2 |

Table 9. Performance gaps between oracle and non-oracle val-
idicators, per algorithm, using a 0.98 source threshold. The mean and standard deviations are computed across transfer tasks. Un-
like Figure 2 and Table 10, the oracle and non-oracle accuracies are collected per algorithm instead of across algorithms.

Figure 3. The Spearman correlation between validation scores and target accuracy, as a function of accuracy threshold. At a threshold of \( x \), only models with source/target accuracy greater than \( x \) are kept. Accuracies are normalized so that the source-only model has a score of 1. Correlations are computed per transfer task. The lines and bands represent mean and standard deviation. As alluded to in Section 4.2, the correlation within some tasks might be higher than Figure 4 suggests. For example, with no thresholding, DEV’s mean, min, and max correlations are 0.40, 0.14, 0.59.
Figure 4. The relationship between source accuracy, target accuracy, and validation scores. For each validation method and task, the validation score is min-max normalized, the target accuracy is max normalized, and the source accuracy is normalized by the source-only model’s accuracy. Top row colorbars represent normalized source accuracy. Bottom row colorbars represent normalized validation score. As discussed in Section 1.2, DEV can produce extremely large values, and our experiments confirm this. To make plots (b) and (e) legible, we exclude the lowest and highest 5% of DEV validation scores.

Table 10. The best target train accuracy for each validation method under two settings: no source thresholding (“None”), and 0.98 source thresholding. For example, say the source-only model has 50% source accuracy. The 0.98 setting will keep only the models that score higher than 49% on the source data, while the None setting will keep all models. The third setting, Oracle, is the true best target accuracy. Note that these oracle values differ from Tables 6 and 7 because these are computed on the target train set, and are also from entirely different training runs.
References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Jul 2019. 4

[2] Zhangjie Cao, Lijia Ma, Mingsheng Long, and Jianmin Wang. Partial adversarial domain adaptation. In Proceedings of the European Conference on Computer Vision (ECCV), pages 135–150, 2018. 1

[3] Woong-Gi Chang, Tackgeun You, Seonguk Seo, Suha Kwak, and Bohyung Han. Domain-specific batch normalization for unsupervised domain adaptation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 7354–7362, 2019. 2

[4] Xinyang Chen, Sinan Wang, Mingsheng Long, and Jianmin Wang. Transferability vs. discriminability: Batch spectral penalization for adversarial domain adaptation. In International conference on machine learning, pages 1081–1090. PMLR, 2019. 2

[5] Shuhao Cui, Shuhui Wang, Junbao Zhuo, Liang Li, Qingming Huang, and Qi Tian. Towards discriminability and diversity: Batch nuclear-norm maximization under label insufficient situations. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 3941–3950, 2020. 2

[6] Shuhao Cui, Shuhui Wang, Junbao Zhuo, Chi Su, Qingming Huang, and Qi Tian. Gradually vanishing bridge for adversarial domain adaptation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 12455–12464, 2020. 2

[7] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. The journal of machine learning research, 17(1):2096–2030, 2016. 2, 4

[8] Muhammad Ghifary, W Bastiaan Kleijn, Mengjie Zhang, David Balduzzi, and Wen Li. Deep reconstruction-classification networks for unsupervised domain adaptation. In European conference on computer vision, pages 597–613. Springer, 2016. 2

[9] Yves Grandvalet, Yoshua Bengio, et al. Semi-supervised learning by entropy minimization. CAP, 367:281–296, 2005. 4

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016. 4

[11] Ying Jin, Ximei Wang, Mingsheng Long, and Jianmin Wang. Minimum class confusion for versatile domain adaptation. In European Conference on Computer Vision, pages 464–480. Springer, 2020. 2

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. 14

[13] Chen-Yu Lee, Tanmay Batra, Mohammad Haris Baig, and Daniel Ulbricht. Sliced wasserstein discrepancy for unsupervised domain adaptation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 10285–10295, 2019. 2

[14] Jian Liang, Dapeng Hu, and Jiashi Feng. Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation. In International Conference on Machine Learning (ICML), pages 6028–6039, July 13–18 2020. 1, 4

[15] Jian Liang, Dapeng Hu, and Jiashi Feng. Domain adaptation with auxiliary target domain-oriented classifier. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 16632–16642, 2021. 2

[16] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In International conference on machine learning, pages 97–105. PMLR, 2015. 2

[17] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. Conditional adversarial domain adaptation. arXiv preprint arXiv:1705.10667, 2017. 2

[18] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Unsupervised domain adaptation with residual transfer networks. arXiv preprint arXiv:1602.04433, 2016. 2

[19] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Deep transfer learning with joint adaptation networks. In International conference on machine learning, pages 2208–2217. PMLR, 2017. 2

[20] Zhihe Lu, Yongxin Yang, Xiatian Zhu, Cong Liu, Yi-Zhe Song, and Tao Xiang. Stochastic classifiers for unsupervised domain adaptation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 9111–9120, 2020. 2

[21] Poojan Oza, Vishwanath A Sindagi, Vibashan VS, and Vishal M Patel. Unsupervised domain adaption of object detectors: A survey. arXiv preprint arXiv:2105.13502, 2021. 1

[22] Pau Panareda Busto and Juergen Gall. Open set domain adaptation. In Proceedings of the IEEE International Conference on Computer Vision, pages 754–763, 2017. 1

[23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andres Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems 32, pages 8024–8035. Curran Associates, Inc., 2019. 4

[24] Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. Moment matching for multi-source domain adaptation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 1406–1415, 2019. 1

[25] Viraj Prabhu, Shivam Khare, Deeksha Kartik, and Judy Hoffman. Sentry: Selective entropy optimization via committee consistency for unsupervised domain adaptation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 8558–8567, 2021. 2
[26] Alan Ramoni and Barbara Plank. Neural unsupervised domain adaptation in NLP—A survey. In Proceedings of the 28th International Conference on Computational Linguistics, pages 6838–6855, Barcelona, Spain (Online), Dec. 2020. International Committee on Computational Linguistics.

[27] Luca Robbiano, Muhammad Rameez Ur Rahman, Fabio Galasso, Barbara Caputo, and Fabio Maria Carlucci. Adversarial branch architecture search for unsupervised domain adaptation, 2021. 2, 4

[28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. International journal of computer vision, 115(3):211–252, 2015. 4

[29] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In European conference on computer vision, pages 213–226. Springer, 2010. 3, 4

[30] Kuniaki Saito, Donghyun Kim, Stan Sclaroff, Trevor Darrell, and Kate Saenko. Semi-supervised domain adaptation via minimax entropy. ICCV, 2019. 1

[31] Kuniaki Saito, Donghyun Kim, Piotr Teterwak, Stan Sclaroff, Trevor Darrell, and Kate Saenko. Tune it the right way: Unsupervised validation of domain adaptation via soft neighborhood density, 2021. 3, 4, 5

[32] Kuniaki Saito, Yoshitaka Ushiku, and Tatsuya Harada. Asymmetric tri-training for unsupervised domain adaptation. In International Conference on Machine Learning, pages 2988–2997. PMLR, 2017. 2

[33] Kuniaki Saito, Kohei Watanabe, Yoshitaka Ushiku, and Tatsuya Harada. Maximum classifier discrepancy for unsupervised domain adaptation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3723–3732, 2018. 1, 2

[34] Kuniaki Saito, Shohei Yamamoto, Yoshitaka Ushiku, and Tatsuya Harada. Open set domain adaptation by backpropagation. In Proceedings of the European Conference on Computer Vision (ECCV), September 2018. 1

[35] Swami Sankaranarayanan, Yogesh Balaji, Carlos D Castillo, and Rama Chellappa. Generate to adapt: Aligning domains using generative adversarial networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 8503–8512, 2018. 2

[36] Yuan Shi and Fei Sha. Information-theoretical learning of discriminative clusters for unsupervised domain adaptation. In John Langford and Joelle Pineau, editors, Proceedings of the 29th International Conference on Machine Learning (ICML-12), ICML ’12, pages 1079–1086, New York, NY, USA, July 2012. Omnipress. 2, 4

[37] Rui Sha, Hung H Bui, Hirokazu Narui, and Stefano Ermon. A dirt-t approach to unsupervised domain adaptation. arXiv preprint arXiv:1802.08735, 2018. 2

[38] Leslie N. Smith and Nicholay Topin. Super-convergence: very fast training of neural networks using large learning rates. Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications, May 2019. 14

[39] Masashi Sugiyama, Matthias Krauledat, and Klaus-Robert Müller. Covariate shift adaptation by importance weighted cross validation. Journal of Machine Learning Research, 8(5), 2007. 3

[40] Baochen Sun, Jiashi Feng, and Kate Saenko. Return of frustratingly easy domain adaptation. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 30, 2016. 2

[41] Marco Toldo, Andrea Maracani, Umberto Michieli, and Pietro Zanuttigh. Unsupervised domain adaptation in semantic segmentation: a review. Technologies, 8(2):35, 2020. 1

[42] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. In Proceedings of the IEEE international conference on computer vision, pages 4068–4076, 2015. 2

[43] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 7167–7176, 2017. 2

[44] Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. Deep hashing network for unsupervised domain adaptation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 5018–5027, 2017. 3, 4

[45] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019. 4

[46] Yuan Wu, Diana Inkpen, and Ahmed El-Roby. Dual mixup regularized learning for adversarial domain adaptation. In European Conference on Computer Vision, pages 540–555. Springer, 2020. 2

[47] Minghao Xu, Jian Zhang, Bingbing Ni, Teng Li, Chengjie Wang, Qi Tian, and Wenjun Zhang. Adversarial domain adaptation with domain mixup. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 6502–6509, 2020. 2

[48] Ruijia Xu, Guanbin Li, Jihan Yang, and Liang Lin. Larger norm more transferable: An adaptive feature norm approach for unsupervised domain adaptation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 1426–1435, 2019. 2

[49] Kaichao You, Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I Jordan. Universal domain adaptation. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 2720–2729, 2019. 1

[50] Kaichao You, Ximei Wang, Mingsheng Long, and Michael Jordan. Towards accurate model selection in deep unsupervised domain adaptation. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pages 7124–7133. PMLR, 09–15 Jun 2019. 2, 4

[51] Yabin Zhang, Hui Tang, Kui Jia, and Mingkui Tan. Domain-symmetric networks for adversarial domain adaptation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 5031–5040, 2019. 2

[52] Erheng Zhong, Wei Fan, Qiang Yang, Olivier Verscheure, and Jiangtao Ren. Cross validation framework to choose
amongst models and datasets for transfer learning. In José Luis Balcázar, Francesco Bonchi, Aristides Gionis, and Michèle Sebag, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 547–562, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
A. Paper Meta Analysis

Figure 5 shows the reported performance gaps for Office31 and OfficeHome. Table 12 contains definitions of the validation methods we found in papers and repos.

B. Experiment Methodology

Tables 11-17 provide details about dataset splits, models, and other experiment settings.

The $x$-DANN combinations (like MCC-DANN) are missing from the hyperparameter search table (Table 15). For these combinations, we searched only the $x$ hyperparameters, and kept the DANN hyperparameters frozen to the best values found in the DANN experiments.

C. Results

Tables 18 and 19 show the standard deviations of the 5 runs for each algorithm in the oracle setting. Bold indicates the lowest value per column, and lower values have a stronger green color. Dashes indicate that reproductions had not yet run when the tables were constructed, so a standard deviation could not be calculated.

Tables 20-23 show the performance gap between oracle and non-oracle validators, per algorithm, at a 0.98 source threshold. Dashes indicate that either all models were discarded with the 0.98 threshold, or that those algorithm/validator/task combinations had not yet run.

Figures 7-17 are scatter plots of validation scores vs target accuracy, per transfer task and feature layer. All values are unnormalized. For DEV, the lowest and highest 5% are excluded to make the plots legible.

**Note about DEV:** The original DEV risk score is supposed to be minimized. Our code is designed to maximize validation scores, so we maximize the negative DEV risk. For the loss function $\ell$ (described in the DEV paper), we use cross entropy.

| Dataset     | Domain  | Train | Val  |
|-------------|---------|-------|------|
| MNIST       | MNIST   | 60000 | 10000|
|             | MNISTM  | 59001 | 9001 |
| Office31    | Amazon  | 2253  | 564  |
|             | DSLR    | 398   | 100  |
|             | Webcam  | 636   | 159  |
| OfficeHome  | Art     | 1941  | 486  |
|             | Clipart | 3492  | 873  |
|             | Product | 3551  | 888  |
|             | Real    | 3485  | 872  |

Table 11. The size of the train/val split for each domain.

| Method                   | Description                                      |
|--------------------------|--------------------------------------------------|
| full oracle              | accuracy on all target data                      |
| subset oracle            | accuracy on a subset of target data              |
| consistency              | cluster / pseudo-label consistency for           |
|            + oracle       | early stopping, but oracle for hyperparameter    |
| src accuracy             | accuracy on the source data                      |
| src accuracy + loss      | accuracy on the source data plus a loss measuring |
| target entropy           | entropy of predictions in the target domain     |
| reverse validation       | see main paper for explanation                  |
| IWCV                     | importance weighted cross validation             |
| DEV                      | see main paper for explanation                  |

Table 12. A description of the validation methods we found in papers and code repos.

| Layers | Feature name |
|--------|--------------|
| Trunk  | LeNet or ResNet50 FL0 |
|        | Linear(256)  |
|        | ReLU()       |
|        | Dropout(0.5) |
|        | Linear(128)  |
|        | ReLU()       |
|        | Dropout(0.5) |
|        | Linear(num_cls) |
|        | Softmax()    |
| Classifier | FL6 |
| Discriminator | FL8 |
|        | Linear(2048) |
|        | ReLU()       |
|        | Linear(2048) |
|        | ReLU()       |
|        | Linear(1)    |

Table 13. The models used in our experiments. Two classifiers are used for MCD, STAR, SWD, and SymNets; one is pretrained and the other is randomly initialized. The depth of the classifier depends on the choice of feature layer. Using feature layer 6 results in the first 6 layers of the classifier moving to the trunk, i.e. the classifier becomes $\text{Linear(num_cls)} \rightarrow \text{Softmax}()$. Using feature layer 8 eliminates the classifier model, so this setting can be used only by certain algorithms. The discriminator is used only for adversarial methods. It receives the feature layer as input, but keeps the same depth regardless of feature layer.
Figure 5. The average reported SOTA-baseline performance gap per year. For example, in Figure (d), the OfficeHome Product→Art (PA) value for DANN in 2021 is 20.3. This means that, on average, 2021 papers report that the best performing algorithm in the PA task has a 20.3 point advantage over the reported DANN accuracy.

Figure 6. Performance gaps between SOTA and baseline algorithms (source-only and DANN) on Office31 tasks. The reported numbers are the average from 2021 papers.
### Category Settings

#### Optimizer
- Adam \([12]\)
- Weight decay of \(1e^{-4}\)
- \(lr \in \log([1e^{-5}, 0.1])\)

#### LR scheduler
- One Cycle \([38]\)
- 5% warmup period
- \(lr_{\text{init}} = \frac{lr_{\text{max}}}{100}\)
- \(lr_{\text{final}} = 0\)
- Cosine annealing

#### Batch size
- 64 source + 64 target

#### Epochs / patience / val interval
- Digits: 100 / 10 / 1
- Office31: 2000 / 200 / 10
- OfficeHome: 200 / 20 / 2

#### Training image transforms
- Resize(256)
- RandomCrop(224)
- RandomHorizontalFlip()
- Normalize()

#### Val/testing image transforms
- Resize(256)
- CenterCrop(224)
- Normalize()

#### MNIST image transforms
- Resize(32)
- GrayscaleToRGB()
- Normalize()

---

Table 14. Various experiment settings. The learning rate (lr) is one of the hyperparameters, and the same lr is used by trunk, classifier, and discriminator.

| Algorithm | Hyperparameter | Search space |
|-----------|----------------|--------------|
| ADDA      | \(\lambda_D\)  | \([0,1]\)    |
|           | \(\lambda_G\)  | \([0,1]\)    |
|           | \(T_{\text{adda}}\) | \([0,1]\) |
| AFN       | \(\lambda_{\text{fn}}\) | \(\log([1e^{-6}, 1])\) |
|           | \(S_{\text{fn}}\) | \([0,2]\)    |
|           | \(\lambda_L\)  | \([0,1]\)    |
| ATDOC     | \(\lambda_{\text{atdoc}}\) | \([0,1]\) |
|           | \(k_{\text{atdoc}}\) | \(\text{int}([5, 25], \text{step}=5)\) |
|           | \(\lambda_L\)  | \([0,1]\)    |
| BNM       | \(\lambda_{\text{bnm}}\) | \([0,1]\) |
|           | \(\lambda_L\)  | \([0,1]\)    |
| BSP       | \(\lambda_{\text{bsp}}\) | \(\log([1e^{-6}, 1])\) |
|           | \(\lambda_L\)  | \([0,1]\)    |
| CDAN      | \(\lambda_D\)  | \([0,1]\)    |
| DC        | \(\lambda_G\)  | \([0,1]\)    |
|           | \(\lambda_L\)  | \([0,1]\)    |
| CORAL     | \(\lambda_F\)  | \([0,1]\)    |
|           | \(\lambda_L\)  | \([0,1]\)    |
| DANN      | \(\lambda_{\text{grl}}\) | \(\log([0,1,10])\) |
|           | \(\lambda_L\)  | \([0,1]\)    |
| GVB       | \(\lambda_D\)  | \([0,1]\)    |
|           | \(\lambda_{\text{B}}\) | \([0,1]\) |
|           | \(\lambda_{\text{D}}\) | \([0,1]\) |
|           | \(\lambda_{\text{grl}}\) | \(\log([0,1,10])\) |
| IM        | \(\lambda_{\text{max}}\) | \([0,1]\) |
|           | \(\lambda_L\)  | \([0,1]\)    |
| ITL       | \(\lambda_{\text{max}}\) | \([0,1]\) |
|           | \(\lambda_{\text{min}}\) | \([0,1]\) |
|           | \(\lambda_L\)  | \([0,1]\)    |
| JMMM       | \(\lambda_F\)  | \([0,1]\)    |
| MMD       | \(\lambda_L\)  | \([0,1]\)    |
|           | \(\gamma_{\text{keep}}\) | \(\text{int}([1, 8])\) |
| MCC       | \(\lambda_{\text{mcc}}\) | \([0,1]\) |
|           | \(T_{\text{mcc}}\) | \([0.2, 5]\) |
|           | \(\lambda_L\)  | \([0,1]\)    |
| MCD       | \(\lambda_{\text{mod}}\) | \(\text{int}([1, 10])\) |
| STAR      | \(\lambda_L\)  | \([0,1]\)    |
| SWD       | \(\lambda_{\text{disc}}\) | \([0,1]\) |
| MinEnt    | \(\lambda_{\text{ent}}\) | \([0,1]\) |
|           | \(\lambda_L\)  | \([0,1]\)    |
| RTN       | \(\lambda_F\)  | \([0,1]\)    |
|           | \(\lambda_L\)  | \([0,1]\)    |
|           | \(\lambda_{\text{ent}}\) | \([0,1]\) |
| SymNets   | \(\lambda_{\text{SymD}}\) | \([0,1]\) |
|           | \(\lambda_{\text{SymC}}\) | \([0,1]\) |
|           | \(\lambda_{\text{SymConf}}\) | \([0,1]\) |
|           | \(\lambda_{\text{SymEnt}}\) | \([0,1]\) |
| VADA      | \(\lambda_D\)  | \([0,1]\)    |
|           | \(\lambda_G\)  | \([0,1]\)    |
|           | \(\lambda_{\text{V}}\) | \([0,1]\) |
|           | \(\lambda_{\text{V}}\) | \([0,1]\) |

Table 15. Hyperparameter search settings.
| Hyperparameter | Description |
|----------------|-------------|
| $\lambda_{afn}$ | AFN loss weight |
| $\lambda_{atdoc}$ | ATDOC loss weight |
| $\lambda_{bnm}$ | BNM loss weight |
| $\lambda_{bsp}$ | BSP loss weight |
| $\lambda_{disc}$ | Classifier discrepancy loss weight for MCD |
| $\lambda_{ent}$ | Target entropy loss weight |
| $\lambda_{grl}$ | Gradient reversal weight, i.e. gradients are multiplied by $-\lambda_{grl}$ |
| $\lambda_{imax}$ | Information maximization loss weight |
| $\lambda_{imin}$ | Information minimization loss weight |
| $\lambda_{mcc}$ | MCC loss weight |
| $\lambda_{B_G}$ | Generator bridge loss weight for GVB |
| $\lambda_{B_D}$ | Discriminator bridge loss weight for GVB |
| $\lambda_{D}$ | Discriminator loss weight |
| $\lambda_{F}$ | Feature distance loss weight |
| $\lambda_{G}$ | Generator loss weight |
| $\lambda_{L}$ | Source classification loss weight |
| $\lambda_{Sym_D}$ | SymNets classifier domain loss weight |
| $\lambda_{Sym_C}$ | SymNets generator category loss weight |
| $\lambda_{Sym_conf}$ | SymNets generator domain loss weight |
| $\lambda_{Sym_ent}$ | SymNets entropy loss weight |
| $\lambda_{V_s}$ | VAT loss weight for the source domain |
| $\lambda_{V_t}$ | VAT loss weight and entropy weight for the target domain |
| $\gamma_{exp}$ | Exponent of the bandwidth multiplier for MMD. For example, if $\gamma_{exp} = 2$, then the bandwidths used will be $\{2^{-2}x, 2^{-1}x, 2^{0}x, 2^{1}x, 2^{2}x\}$, where $x$ is the base bandwidth. |
| $k_{atdoc}$ | Number of nearest neighbors to retrieve for computing pseudolabels in ATDOC |
| $N_{mcd}$ | Number of times the MCD generator is updated per batch |
| $S_{afn}$ | Step size used by the AFN loss function |
| $T_{adda}$ | Minimum discriminator accuracy required to trigger a generator update in ADDA |
| $T_{mcc}$ | Softmax temperature used by MCC |

Table 16. Description of every hyperparameter in Table 15.

| Task | IM | DEV | SND | Total |
|------|----|-----|-----|-------|
| MM   | 82 | 65  | 78  | 225   |
| AD   | 61 | 48  | 62  | 171   |
| AW   | 64 | 32  | 65  | 161   |
| DA   | 48 | 17  | 36  | 101   |
| DW   | 54 | 25  | 57  | 136   |
| WA   | 49 | 33  | 49  | 131   |
| WD   | 50 | 47  | 51  | 148   |
| AP   | 23 | 20  | 17  | 60    |
| CR   | 20 | 15  | 14  | 49    |
| PA   | 49 | 22  | 56  | 127   |
| RC   | 17 | 19  | 15  | 51    |
| Total| 517| 343 | 500 | 1360  |

Table 17. Number of datapoints (thousands) collected per validator/task pair.

| Method | AD | AW | DA | DW | WA | WD | Avg |
|--------|----|----|----|----|----|----|-----|
| ADDA   | 2.1| 1.8| 0.6| 0.7| 1.1| 3.0| 1.6 |
| AFN    | 2.9| 2.5| 0.7| 0.4| 0.9| 1.3|     |
| AFN-DANN | 2.0| 1.9| 0.7| 0.9| 0.6| 1.1|     |
| ATDOC  | 3.3| 1.3| 1.0| 0.5| 0.5| 1.3|     |
| ATDOC-DANN | 1.0| 2.0| 0.5| 0.4| 1.3| 1.0|     |
| BNM    | 1.2| 2.0| 1.0| 0.5| 0.3| 0.8|     |
| BNM-DANN | 2.2| 2.4| 0.7| 0.8| 0.6| 1.2|     |
| BSP    | 2.9| 1.0| 0.7| 0.4| 0.5| 1.1|     |
| BSP-DANN | 2.0| 1.3| 0.5| 0.3| 0.6| 0.8|     |
| CDAN   | 0.7| 2.2| 0.6| 0.9| 0.3| 0.8| 0.9 |
| CORAL  | 1.2| 1.3| 0.9| 1.7| 0.5| 2.3| 1.3 |
| DANN   | 0.9| 1.0| 1.0| 0.5| 0.6| 0.6| 0.7 |
| DANN-FL8 | 1.6| 1.1| 0.4| 0.3| 0.6| 0.6| 0.8 |
| DC     | 4.2| 1.3| 0.8| 1.6| 0.7| 0.6| 1.5 |
| GVB    | 0.7| 2.0| 1.2| 1.6| 0.7| 0.5| 1.1 |
| IM     | 1.5| 2.0| 0.5| 1.8| 0.8| 0.8| 1.2 |
| IM-DANN | 2.7| 1.9| 0.7| 0.6| 0.8| 1.2|     |
| ITL    | 2.2| 1.4| 0.8| 0.9| 0.5| 0.6| 1.1 |
| JMMD   | 1.9| 1.6| 1.0| 0.9| 0.9| 0.5| 1.1 |
| MCC    | 0.8| 2.3| 0.7| 0.9| 0.3| 0.6| 0.9 |
| MCC-DANN | 0.7| 2.3| 0.9| 0.6| 0.7| 0.6| 1.0 |
| MCD    | 2.9| 2.0| 0.6| 1.2| 0.3| 1.0| 1.3 |
| MMD    | 1.6| 2.6| 0.6| 0.3| 1.1| 0.6| 1.1 |
| MinEnt | 4.2| 2.9| 1.0| 0.5| 0.7| 0.4| 1.6 |
| RTN    | 3.1| 0.6| 0.9| 0.6| 1.0| 1.2| 1.3 |
| STAR   | 2.9| 0.8| 1.3| 2.1| 0.5| 0.8| 1.4 |
| SWD    | 4.9| 0.9| 0.6| 0.9| 0.7| 2.9| 1.8 |
| SymNets| 2.8| 1.3| 2.7| 1.2| 1.5| 0.6| 1.7 |
| VADA   | 3.4| -   | 0.5| 0.5| 0.5| 0.3| 1.1 |

Table 18. Standard deviation on Office31.
### Table 19. Standard deviation on the MNIST $\rightarrow$ MNISTM (MM) and OfficeHome transfer tasks. The Avg column is the OfficeHome average.

| Method     | AC  | AP  | AR  | CA  | CP  | CR  | PA  | PC  | PR  | RA  | RC  | RP  | Avg  |
|------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|
| ADDA       | 0.4 | 1.0 | 0.8 | 0.4 | 1.1 | 0.8 | 0.6 | 1.1 | 0.9 | 0.8 | 1.3 | 1.5 | 0.6  |
| AFN        | 1.0 | 0.5 | 0.6 | 0.5 | 1.1 | 0.6 | 0.8 | 0.2 | 0.7 | 0.7 | 1.0 | 1.0 | 0.9  |
| AFN-DANN   | 0.1 | 0.7 | 0.7 | 0.4 | 0.7 | 0.9 | 1.2 | 1.7 | 1.0 | 0.7 | 0.5 | 0.9 | 1.4  |
| ATDOC      | 9.3 | 0.8 | 0.9 | 0.7 | 1.3 | 1.3 | 0.5 | 1.0 | 0.5 | 0.4 | 0.7 | 0.8 | 0.9  |
| ATDOC-DANN | 6.1 | 0.7 | 0.4 | 0.4 | 0.8 | 1.6 | 0.7 | 0.8 | 0.8 | 0.9 | 1.8 | 1.5 | 0.3  |
| BNM        | 0.4 | 0.9 | 0.6 | 0.6 | 1.4 | 0.9 | 1.0 | 0.8 | 0.8 | 0.8 | 1.0 | 1.2 | 0.6  |
| BNM-DANN   | -   | 1.1 | 0.8 | 0.4 | 1.3 | 0.9 | 1.0 | 1.2 | 1.3 | 0.8 | 1.3 | 1.5 | 1.0  |
| BSP        | 0.1 | 0.8 | 0.2 | 0.6 | 1.3 | 0.6 | 0.9 | 0.8 | 0.3 | 0.3 | 1.1 | 0.9 | 0.7  |
| BSP-DANN   | -   | 0.6 | 0.8 | 0.6 | 1.5 | 1.4 | 0.8 | 0.4 | 1.3 | 0.6 | 1.0 | 2.4 | 0.6  |
| CDAN       | 6.3 | 1.5 | 0.6 | 0.6 | 1.0 | 0.2 | 0.5 | 0.6 | 2.3 | 0.5 | 0.9 | 1.3 | 0.4  |
| CORAL      | 1.5 | 0.4 | 0.6 | 0.4 | 1.2 | 0.5 | 0.6 | 0.7 | 0.6 | 0.5 | 1.1 | 1.3 | 0.6  |
| DANN       | 0.5 | 0.5 | 0.4 | 0.8 | 1.0 | 1.6 | 0.4 | 0.8 | 1.7 | 1.1 | 2.3 | 0.9 | 1.6  |
| DANN-FL8   | 8.7 | 1.0 | 0.9 | 0.6 | 0.6 | 0.6 | 1.0 | 0.7 | 1.6 | 1.1 | 0.8 | 0.7 | 0.6  |
| DC         | 2.6 | 1.4 | 0.8 | 0.4 | 0.4 | 1.2 | 0.6 | 1.7 | 1.3 | 0.7 | 1.1 | 1.2 | 0.7  |
| GVB        | 5.4 | 0.5 | 0.6 | 1.0 | 1.3 | 1.1 | 1.1 | 2.2 | 1.1 | 0.5 | 1.0 | 1.0 | 0.6  |
| IM         | 0.2 | 0.9 | 0.6 | 0.8 | 0.8 | 0.8 | 0.7 | 0.5 | 0.7 | 0.4 | 0.7 | 0.5 | 0.7  |
| IM-DANN    | -   | 0.7 | 1.1 | 0.8 | 1.2 | 1.5 | 0.4 | 2.2 | 0.8 | 1.1 | 0.9 | 0.9 | 1.2  |
| ITL        | 0.2 | 0.4 | 0.7 | 0.9 | 1.1 | 0.4 | 0.4 | 1.3 | 1.2 | 0.6 | 0.9 | 1.2 | 1.1  |
| JMMDO      | 8.1 | 0.3 | 0.5 | 0.5 | 0.9 | 0.2 | 1.3 | 1.4 | 1.0 | 0.4 | 0.3 | 0.8 | 0.9  |
| MCC        | 4.0 | 1.9 | 0.4 | 0.4 | 1.4 | 0.9 | 0.8 | 0.8 | 1.0 | 0.4 | 1.1 | 1.4 | 1.1  |
| MCC-DANN   | -   | 1.1 | 1.2 | 0.9 | 0.9 | 0.3 | 0.8 | 0.9 | 1.3 | 0.7 | 0.4 | 1.1 | 0.6  |
| MCD        | 0.3 | 0.5 | 0.7 | 0.8 | 1.3 | 1.6 | 1.4 | 0.7 | 2.0 | 0.6 | 1.1 | 0.8 | 1.3  |
| MMD        | 0.4 | 0.9 | 0.4 | 0.5 | 0.9 | 1.2 | 0.9 | 0.7 | 1.3 | 0.8 | 0.5 | 0.4 | 1.4  |
| MinEnt     | 0.1 | 0.6 | 1.2 | 0.3 | 0.7 | 0.8 | 0.4 | 1.2 | 0.6 | 0.5 | 1.3 | 0.8 | 1.1  |
| RTN        | 0.7 | 0.4 | 0.9 | 0.6 | 0.9 | 0.6 | 1.0 | 1.4 | 1.2 | 0.7 | 0.6 | 1.3 | 0.9  |
| STAR       | 0.4 | 0.7 | 0.3 | 0.5 | 0.6 | 2.3 | 1.0 | 1.3 | 0.8 | 0.6 | 0.9 | 0.5 | 0.9  |
| SWD        | 1.9 | 1.1 | 0.5 | 0.9 | 1.1 | 0.3 | 0.8 | 0.9 | 1.0 | 0.6 | 1.4 | 1.7 | 0.8  |
| SymNets    | 24.7| 1.4 | 1.1 | 1.1 | 1.2 | 3.7 | 1.2 | 0.7 | 2.1 | 0.9 | 0.5 | 1.1 | 1.5  |
| VADA       | 1.5 | 0.8 | 0.3 | 0.3 | 1.6 | 1.5 | 1.5 | 1.0 | 1.2 | 0.7 | 1.6 | 0.7 | 0.4  |

The Avg column is the OfficeHome average.
| Algorithm | MM | AD | AW | DA | DW | WA | WD | AP | CR | PA | RC |
|----------|----|----|----|----|----|----|----|----|----|----|----|
| ADDA     | 7.2| 2.6| 0.6| 39.8|35.7|33.6|0.7|21.4|0.6|41.7|   |
| AFN      | 10.9|5.3|3.1|6.6|1.6|1.8|0.5|1.9|1.6|1.3|2.9|
| ATDOC    | 55.1|11.4|7.0|1.7|2.6|1.9|7.8|   |1.3|5.3|   |
| BNM      | 1.1|8.7|11.5|3.9|1.1|3.9|1.2|2.3|6.8|9.8|0.9|
| BSP      | 0.0|1.4|0.3|0.2|1.0|0.1|0.0|0.8|3.8|2.1|3.7|
| CDAN     | 22.5|13.5|7.1|9.5|10.2|2.5|5.7|0.9|6.7|9.5|   |
| CORAL    | 12.4|6.5|9.9|19.1|5.7|14.3|0.9|7.5|9.5|6.0|1.3|
| DANN     | 25.0|17.0|8.5|10.5|4.3|6.3|14.1|3.3|2.5|7.0|4.0|
| DC       | 3.7|10.4|6.6|6.9|5.7|4.5|5.7|3.3|0.7|7.0|0.6|
| GVB      | 17.6|3.0|8.3|16.9|9.6|12.9|9.5|   |5.6|13.5|1.8|
| JMMD     | 1.4|27.5|20.0|19.8|1.7|17.8|0.5|1.5|1.6|5.4|4.4|
| MCC      | 2.1|4.6|6.6|9.6|1.0|9.2|5.9|10.0|5.3|10.6|5.0|
| MCD      | 5.6|3.2|7.6|10.5|1.9|15.7|0.5|   |0.6|3.8|   |
| MMD      | 4.9|12.8|14.3|21.9|2.1|16.3|0.5|2.6|5.5|13.2|3.5|
| RTN      | 2.2|0.4|0.9|6.0|5.2|2.9|0.5|2.0|1.5|5.7|1.5|
| SWD      | 15.7|10.8|7.0|29.2|0.4|24.2|1.1|0.0|2.0|9.9|   |
| SymNets  | 0.0|0.9|20.2|10.3|4.5|9.6|3.6|   |16.2|25.8|   |
| VADA     | 9.7|3.7|5.0|0.5|1.6|8.0|0.9|   |7.0|2.8|   |

Table 20. Performance gap between oracle and IM, at 0.98 source thresholding.

| Algorithm | MM | AD | AW | DA | DW | WA | WD | AP | CR | PA | RC |
|----------|----|----|----|----|----|----|----|----|----|----|----|
| ADDA     | 0.3|80.2|   |   |   |2.4|94.9|1.1|   |2.4|1.7|
| AFN      | 44.5|4.5|0.6|7.5|6.4|27.1|0.6|1.4|1.6|7.0|3.9|
| ATDOC    | 52.7|16.8|9.2|6.2|6.8|5.4|10.9|   |0.4|2.9|   |
| BNM      | 43.0|7.3|2.5|0.9|4.4|3.4|0.8|3.1|5.3|1.3|8.5|
| BSP      | 5.8|1.3|3.8|18.6|12.4|35.2|2.3|   |1.4|0.8|   |
| CDAN     | 8.9|7.0|   |   |   |1.7|1.9|1.3|   |1.3|4.5|
| CORAL    | 24.5|8.8|9.2|25.6|7.4|25.1|8.3|3.1|11.1|6.5|2.6|
| DANN     | 20.0|16.5|6.0|0.9|2.9|2.0|3.4|0.6|3.0|3.0|2.2|
| DC       | 2.7|7.0|8.4|0.4|0.8|2.7|2.4|1.3|   |2.7|1.5|
| GVB      | 42.3|8.5|19.9|33.1|9.9|32.4|1.8|   |3.4|4.9|4.2|
| JMMD     | 37.2|5.9|1.7|4.1|1.7|16.7|5.4|   |3.8|1.4|   |
| MCC      | 1.6|3.6|3.8|4.3|2.6|0.0|2.0|5.6|5.2|6.1|1.7|
| MCD      | 22.6|   |   |   |   |   |   |   |   |   |   |
| MMD      | 20.4|8.1|3.7|8.7|5.4|42.2|1.7|2.2|8.6|3.4|1.2|
| RTN      | 22.3|5.2|4.2|1.4|5.2|67.0|2.2|1.4|8.0|2.2|2.4|
| SWD      | 41.0|7.9|2.9|0.6|1.6|29.7|1.3|   |   |   |   |
| SymNets  | 14.6|4.6|9.0|5.3|1.4|1.5|5.9|   |   |28.2|   |
| VADA     | 0.6|6.7|3.1|   |1.1|13.0|0.2|4.3|4.8|3.4|   |

Table 21. Performance gap between oracle and DEV, at 0.98 source thresholding.
| Algorithm | MM  | AD  | AW  | DA  | DW  | WA  | WD  | AP  | CR  | PA  | RC  |
|-----------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| ADDA      | 73.9| 80.4| 75.2| 61.0| 90.3| 63.0| 94.8| 61.7| 61.7| 56.7| 41.5|
| AFN       | 53.4| 5.1 | 9.1 | 50.6| 7.1 | 43.9| 8.2 | 1.7 | 1.5 | 6.6 | 0.0 |
| ATDOC     | 60.8| 23.9| 8.6 | 6.5 | 16.0| 52.4| 13.2| 6.0 | 14.6| 20.6| -   |
| BNM       | 53.9| 2.0 | 5.1 | 37.3| 5.1 | 54.2| 3.5 | 1.4 | 1.8 | 3.4 | 1.9 |
| BSP       | 48.7| 0.6 | 15.5| 30.4| 16.5| 33.1| 7.3 | -   | 23.7| 11.5| 1.1 |
| CDAN      | 47.5| 8.3 | 4.3 | 10.4| 8.8 | 8.0 | 3.1 | 2.7 | 15.4| 7.3 | 0.5 |
| CORAL     | 16.2| 15.0| 10.1| 24.2| 11.9| 54.0| 2.9 | 2.2 | 4.1 | 7.6 | 3.4 |
| DANN      | 43.2| 12.3| 6.0 | 6.0 | 5.7 | 36.5| 2.0 | 0.4 | 2.8 | 6.9 | 4.8 |
| DC        | 47.1| 8.9 | 4.4 | 13.2| 7.8 | 27.7| 0.8 | 1.2 | 4.9 | 14.8| 1.9 |
| GVB       | 60.8| 34.6| 19.6| 9.8 | 6.0 | 34.2| 16.4| 2.0 | 12.2| 19.6| 2.8 |
| JMMD      | 27.0| 12.1| 11.2| 41.5| 17.1| 59.0| 4.6 | 2.3 | 0.4 | 22.3| 2.0 |
| MCC       | 45.2| 0.0 | 1.5 | 4.5 | 11.1| 49.2| 9.0 | 0.9 | 0.9 | 0.8 | 4.5 |
| MCD       | 83.3| 2.7 | 2.6 | 41.7| 15.1| 36.4| 4.9 | -   | 8.3 | 15.2| 1.1 |
| MMD       | 28.6| 15.0| 19.2| 36.3| 19.5| 51.5| 4.8 | -   | 21.3| 8.0 | 2.2 |
| RTN       | 47.8| 9.1 | 20.3| 68.8| 94.8| 69.6| 8.2 | -   | 12.1| 22.8| 1.1 |
| SWD       | 43.3| 10.1| 7.7 | 37.2| 13.8| 28.3| 9.9 | 2.4 | 12.0| 12.2| -   |
| SymNets   | 77.4| 13.1| 80.3| 13.2| 89.2| 54.0| 84.5| -   | -   | 50.8| -   |
| VADA      | 79.0| 2.3 | 5.5 | 1.5 | -   | 7.3 | 0.0 | -   | 2.4 | 9.2 | -   |

Table 22. Performance gap between oracle and SND, at 0.98 source thresholding.

| Algorithm | MM  | AD  | AW  | DA  | DW  | WA  | WD  | AP  | CR  | PA  | RC  |
|-----------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| ADDA      | 10.6| 4.8 | 6.7 | 11.5| 12.3| 16.4| 20.3| 9.9 | 4.6 | 7.0 | 1.9 |
| AFN       | 3.4 | 3.5 | 3.2 | 1.3 | 8.7 | 44.6| 4.8 | 1.9 | 5.2 | 3.1 | 2.5 |
| ATDOC     | 10.9| 4.5 | 4.9 | 0.4 | 5.1 | 2.1 | 4.0 | 5.4 | 0.0 | 8.2 | -   |
| BNM       | 4.0 | 4.7 | 8.9 | 3.5 | 7.9 | 4.3 | 8.0 | 1.5 | 11.5| 6.9 | 5.5 |
| BSP       | 0.2 | 0.8 | 1.7 | 0.6 | 9.6 | 40.8| 8.3 | -   | 4.2 | 0.9 | 1.1 |
| CDAN      | 28.2| 7.7 | 8.8 | 2.2 | 2.1 | 3.0 | 1.9 | 0.3 | 8.5 | 5.4 | 4.4 |
| CORAL     | 16.2| 7.2 | 6.1 | 1.0 | 9.8 | 0.5 | 5.1 | 0.5 | 2.2 | 1.5 | 0.9 |
| DANN      | 29.2| 9.6 | 3.1 | 1.6 | 7.0 | 4.3 | 3.9 | 2.0 | 8.0 | 6.8 | 7.0 |
| DC        | 28.6| 2.8 | 7.3 | 3.2 | 2.0 | 10.5| 7.2 | 0.8 | 5.7 | 4.9 | 4.9 |
| GVB       | 11.8| 5.2 | 13.9| 4.2 | 9.2 | 6.1 | 9.1 | 2.0 | 17.2| 7.5 | 4.4 |
| JMMD      | 25.7| 5.4 | 6.8 | 3.0 | 17.4| 2.0 | 8.8 | 2.7 | 2.3 | 5.3 | 2.6 |
| MCC       | 5.3 | 6.9 | 12.6| 3.3 | 12.0| 5.2 | 9.6 | 5.4 | 7.3 | 7.1 | 4.5 |
| MCD       | 29.1| 3.8 | 3.1 | 0.9 | 3.3 | 0.9 | 1.4 | -   | 4.2 | 3.6 | 1.1 |
| MMD       | 7.9 | 3.5 | 6.3 | 1.8 | 7.8 | 39.5| 6.7 | -   | 2.6 | 4.6 | 6.2 |
| RTN       | 0.0 | 2.9 | 9.2 | 16.6| 5.0 | 2.6 | 4.0 | -   | 5.6 | 5.4 | 0.1 |
| SWD       | 24.7| 2.4 | 9.3 | 0.0 | 3.6 | 9.2 | 1.9 | 2.7 | 1.4 | 3.5 | -   |
| SymNets   | 34.8| 1.9 | 14.4| 9.5 | 4.9 | 8.6 | 4.6 | -   | -   | 38.7| -   |
| VADA      | 32.1| 5.7 | 6.4 | 1.0 | -   | 2.2 | 0.9 | -   | 2.4 | 5.3 | -   |

Table 23. Performance gap between oracle and NegSND, at 0.98 source thresholding.
Figure 7. MNIST→MNISTM task. \textbf{x-axis}: validation score, \textbf{y-axis}: target train accuracy, \textbf{colorbar}: source accuracy.

Figure 8. Office31 AD task. \textbf{x-axis}: validation score, \textbf{y-axis}: target train accuracy, \textbf{colorbar}: source accuracy.

Figure 9. Office31 AW task. \textbf{x-axis}: validation score, \textbf{y-axis}: target train accuracy, \textbf{colorbar}: source accuracy.
Figure 10. Office31 DA task. **x-axis**: validation score, **y-axis**: target train accuracy, **colorbar**: source accuracy.

Figure 11. Office31 DW task. **x-axis**: validation score, **y-axis**: target train accuracy, **colorbar**: source accuracy.

Figure 12. Office31 WA task. **x-axis**: validation score, **y-axis**: target train accuracy, **colorbar**: source accuracy.
Figure 13. Office31 WD task. \textbf{x-axis}: validation score, \textbf{y-axis}: target train accuracy, \textbf{colorbar}: source accuracy.

Figure 14. OfficeHome AP task. \textbf{x-axis}: validation score, \textbf{y-axis}: target train accuracy, \textbf{colorbar}: source accuracy.

Figure 15. OfficeHome CR task. \textbf{x-axis}: validation score, \textbf{y-axis}: target train accuracy, \textbf{colorbar}: source accuracy.
Figure 16. OfficeHome PA task. x-axis: validation score, y-axis: target train accuracy, colorbar: source accuracy.

Figure 17. OfficeHome RC task. x-axis: validation score, y-axis: target train accuracy, colorbar: source accuracy.