Robust Unsupervised Domain Adaptation

8 ECTS Semester Project - Computational Science and Engineering

Eelis Mielonen (Sciper 323681)

28th January 2023

Approved by the Examining Committee:

Grigorios Chrysos, Project Supervisor

Volkan Cevher, Project Advisor

École Polytechnique Fédérale de Lausanne (EPFL)
Laboratory for Information and Inference Systems (LIONS)

Abstract

Unsupervised domain adaptation (UDA) is a set of techniques used in machine learning and artificial intelligence to adapt a model trained on a source domain to a target domain where only unlabeled data is available. The goal in domain adaptation is to improve the performance of a model on the target domain by transferring knowledge learnt in the source domain. In this project we focus on image classification methods in UDA, and identify the worst-class performance as a potential area of improvement. We study how optimising a re-weighted class-conditioned risk in the source domain impacts accuracy in the target domain. Through experiments on the Visda2017 and DomainNet benchmarks we show that label conditional value at risk (LCVaR) can improve worst-class performance in the target domain, and that uniform class-conditioned sampling with class focused online learning (CFOL) can lead to the largest improvements. However, CFOL leads to less consistent results when the sampling distribution is not uniform.

Acknowledgements

I would like to thank my supervisor Grigoris for guiding the project, and the Machine Learning Group at Tsinghua University for creating a great publicly available library for transfer learning, on which a lot of this work was based.

Contents

Figure 1: Example of a domain shift in the MNIST (top) vs. MNIST-M (bottom) dataset. Both sets of images contain digits, but MNIST-M has background features which can confuse a model.

1 Introduction

One problem that machine learning practitioners may encounter is a decrease in the performance of a model when the input data distribution changes. For instance, a model trained on street-view images taken on a clear summer day may not work as effectively in a production setting where the images are taken in a fog. Even a basic digit classification model trained on MNIST performs worse on the similar but stylistically different MNIST-M dataset (Fig. 1). As a response, the fields of transfer learning and domain adaptation have emerged to mitigate this effect.

Unsupervised domain adaptation is a technique used in machine learning and artificial intelligence to adapt a model trained on a source domain to a target domain where only unlabeled data is available. The goal of unsupervised domain adaptation is to improve the performance of the model on the target domain by using knowledge learned from the source domain. One common approach to unsupervised domain adaptation is to use a shared feature space, where the model is trained to learn common features shared between the source and target domains.

Unsupervised domain adaptation (UDA) is useful in situations where there is a lack of labeled data in the target domain, but there is a related source domain with abundant labeled data. For example, in medical imaging, we may have an image classification model that classifies diseases but which does not work with different imaging modalities. Obtaining accurate diagnosis labels would require evaluation by a doctor, making the labeling effort time consuming. Sometimes patient diagnoses are private, so labels are unavailable irregardless. With UDA we can fine-tune the existing model to work on a new imaging modality as well, despite the absence of labeled data in the target domain.

In the scope of this semester project, we restrict our attention to UDA in image classification. The purpose of this project was to pinpoint some basic limitations of existing techniques in this area, and to propose solutions for them. We found that poorly performing classes in the source domain translate to poorly performing classes in the target domain, and recommend a modification to the source domain loss which could fix this issue. We test our method empirically on common UDA benchmarks and find that we can get marginal improvements.

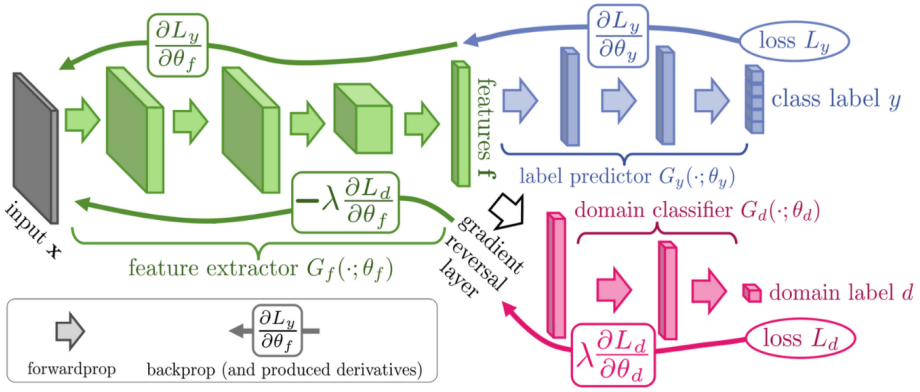

Figure 2: Illustration of the domain adversarial network (DANN). Figure from [4].

2 Background

2.1 Domain Adaptation

There are several unsupervised domain adaptation techniques that have been proposed for image classification, and the best technique may depend on the specific application and the characteristics of the data. The most popular unsupervised domain adaptation techniques include Adversarial Domain Adaptation, which use adversarial training to learn domain-invariant feature representations. Correlation Alignment (CORAL) aligns the second-order statistics of the feature representations of the source and target domains, by minimizing the distance between their covariance matrices. Finally, Transfer Component Analysis (TCA) learns a linear transformation that maps the source domain data to a new feature space where it is more similar to the target domain data. Needless to say, literature on domain adaptation is vast and it is hard to pinpoint a solution that works in a broad variety of contexts. However, for applications in computer vision, one can argue that methods based on adversarial training have been the most successful.

2.2 Unsupervised Domain Adaptation with Adversarial Learning

One of the seminal works in UDA called Domain Aversarial Neural Networks (DANN) aimed to learn domain invariant features by adding a branch to a classifier backbone that predicts which domain the image comes from (Fig. 2). The model is trained to correctly classify images in the source domain, and to poorly classify the domain label, which leads to solving a min-max optimisation problem over the classifier backbone parameters. Intuitively, this forces the network to learn features that are similar between domains that simultaneously separate classes.

This work, and a lot of the ones that followed, worked under the principle of using the unlabeled target domain images in order to "align" the two domains. This approach has it's roots in generalization bounds [2] for the error in the target domain that involve the empirical error rate in the source domain and a domain separation term. Mathematically speaking, a domain is the joint probability distribution of model inputs $x \in \mathcal{X}$ and the correct labels

$y \in \mathcal{Y}$, and a domain shift can be described as a difference between two such distributions. On an abstract level, the derived bounds relate the error rate of a classifier $h : \mathcal{X} \to \mathcal{Y}$ in domain $\mathcal{D}_T$, $\epsilon(h)_{\mathcal{D}_T}$, with the empirical error rate of the same classifier on domain $\mathcal{D}_S$, which we denote by $\hat{\epsilon}(h)_{\mathcal{D}_S}$:

$$\epsilon(h)_{\mathcal{D}_T} \leq \hat{\epsilon}(h)_{\mathcal{D}_S} + \mathcal{A}(\mathcal{D}_\mathcal{S}, \mathcal{D}_\mathcal{T}) + \lambda$$

Where $\mathcal{A}$ is some notion of dissimilarity between the domains $\mathcal{D}_S$ and $\mathcal{D}_T$, and λ is a constant. To evaluate $\mathcal{A}$ we don't need labels in the target domain. Therefore if we minimize the combination of the empirical risk and domain discrepancy $\mathcal{A}$ will bound the true risk in the target domain. Some methods in UDA aim to minimize a computable version of $\mathcal{A}$, while some others simply borrow the same intuition. For instance, in PixelDA [3], a generative adversarial neural network is trained to translate the source training images to target images. With labeled source images transformed to match the style of target images, we can train a classifier with better performance in the target domain. This mechanism "aligns" the two domains on the level of image pixels, without explicitly minimizing some $\mathcal{A}$.

Some computable definitions of $\mathcal{A}$ involve a supremum, so minimizing the right hand size of the generalization bound above is equivalent to solving a min-max problem. Additionally, adversarial networks are used throughout UDA in classification and semantic segmentation. The challenges related to min-max optimisation, such as instability, are often encountered in UDA.

2.3 Class Focused Online Learning (CFOL)

Typically in classification we wish to minimize the average expected loss:

$$\min_\theta \mathbb{E}_{(x,y)\sim P(x,y)}[L(\theta, x, y)]$$

Where θ refers to the model parameters, L is some loss function, and P is the data distribution. In the adversarial training set-up, where L is replaced by adversarial loss, we may end up with models that have an acceptable average accuracy but which have a few under-performing classes. For example, one class with a 30% validation accuracy whilst others have a 90% accuracy. This kind of imbalance can arise from an imbalanced training set, but also from deep models learning "shortcuts" that neglect some classes in favor of a good average performance. Having a poor worst-class performance may not be acceptable in key application areas such as medical diagnoses and self-driving cars.

We can instead minimize the loss associated with the worst performing class [1], or the maximum class conditioned risk:

$$\min_\theta \max_{y\in[k]} \mathbb{E}_{x\sim P(x|\cdot)}[L(\theta, x, y)] = \min_\theta \max_{y\in[k]} L_y(\theta)$$

Intuitively speaking, we optimize the model parameters with respect to the maximum loss that choosing a particular class can incur. Minimizing this expression instead of ER bears

no obvious risks, because we know that an average of a set of values is below the maximum value on that set: the maximum class conditioned risk thus upper bounds expected risk. Unfortunately, in the form above, we have a mixed optimisation problem with integers for each class y, and continuous variables for model parameters θ, making it difficult to solve.

In order to arrive at a concrete learning algorithm, we begin by making a convex relaxation:

$$\min_{\theta} \max_{y \in [k]} L_y(\theta) \leq \min_{\theta} \max_{p \in \Delta_k} \sum_{i=1}^{k} p_i L_i(\theta) \tag{1}$$

Where Δ_k denotes the probability simplex, $\sum_{i=1}^{k} p_i = 1$. Here we can view the objective as a min-max game between the model parameters θ, and the weights p_i of the per-class losses. This linear form can be solved with **Hedge** and the multiplicative weights update algorithm.

Hedge is an algorithm for the expert advice problem. We imagine K slot machines or "experts", and proceeding through a set of T rounds where we pick one of the experts on each round. Picking expert i on round t gives us a random reward c_i^t, and our goal is to maximize the average profits over time. We pick expert i with probability p_i^t, and the average expected reward over T iterations is given by:

$$\frac{1}{T} \sum_{i=1}^{T} \vec{c_t} \cdot \vec{p_t} = \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{K} p_i^t c_i^t$$

The hedge algorithm provides a way to maximise the above expression by modifying the sampling distribution $\vec{p}$ over experts. The theoretical guarantees of Hedge apply even when c is chosen by an adversary, given that the rewards are bounded. The linear expression above is identical to equation 1, if we replace c_i^t by the class conditioned risk $L_i(\theta)$. Minimizing with respect to the model parameters θ corresponds to choosing c_i^t adversarially from the point of view of p, which we choose to maximize the loss. The multiplicative weights update algorithm works by repeating the following steps:

1. Observe: $c_t^i \; \forall \; i \in [K]$
2. Receive reward: $\vec{c_t} \cdot \vec{p_{t-1}}$
3. Compute weights: $w_t^i = w_{t-1}^i e^{\epsilon \cdot c_t^i} \; \forall \; i \in [K]$
4. Compute new distribution: $p_i^t = w_t^i / \sum_{j=1}^{K} w_t^j \; \forall \; i \in [K]$

For some constant ϵ. For an appropriately chosen ϵ, Hedge guarantees that the regret (the difference between the payoff of the strategy vs. always choosing the best expert), is bounded as $\mathcal{O}(\sqrt{T \log(k)})$. We can therefore optimise the objective in equation 1 in a principled way by interleaving model update steps with steps where we update the distribution $\vec{p}$. Over a large

number of iterations, the class with the highest classification loss will contribute even more to the total loss, forcing the model to make updates that are beneficial to the worst performing class.

In practice, we must resort to approximations of the class conditioned risk, since computing it would require a pass over the whole dataset each iteration. The authors of CFOL propose the following algorithm that works with batched samples of data, but which is equivalent in expectation:

Data: A rule F for updating model parameters, step-size η, mixing parameter γ.
while $t \leq T$ **do**
  $y^t \sim p^t$ `// sample class`
  $i^t \sim U(|N_{y^t}|)$ `// sample index uniformly from class`
  $\theta^{t+1} = F(\theta^t, L_{y^t,i^t})\forall y$ `// Update Model`
  $\hat{L}_y^t = 1[y = y^t]L_{y,i^t}(\theta_t)/p_y^t$ `// Compute Estimator of Class Loss`
  $w_y^{t+1} = w_y^t - \eta\hat{L}_y^t$ `// Compute hedge MW update in log space`
  $q_y^{t+1} = \exp(w_y^{t+1})/\sum_{y=1}^k \exp(w_y^{t+1})$ `// Normalize distribution`
  $p_y^{t+1} = (1-\gamma)1/k + \gamma q_y^{t+1}\ \forall y$ `// Mix with uniform distribution`
end

Algorithm 1: CFOL

The actual sampling distribution is a convex combination between a uniform distribution and the modified distribution. This trade-off is controlled by the parameter γ. The loss used in the fourth step of the algorithm is the 0-1 loss, and in practice it can be different from the loss function used in the model update step.

2.4 Label Conditional Value at Risk (LCVaR)

Another robust loss function that we may wish to optimise instead of average risk is conditional value at risk (CVaR). CVaR is a risk management metric that quantifies the level of loss that may occur with a given probability. It is also known as expected shortfall. It is commonly used in finance and insurance to measure the potential loss from a portfolio or investment. CVaR is typically calculated as the expected value of the tail of the loss distribution, beyond a certain threshold (or confidence level). For example, if the CVaR for a portfolio is \$1 million at a 99% confidence level, it means that there is a 1% chance that the expected loss equals \$1 million. In the context of machine learning, we can attempt to minimize CVaR with parameter α to focus on minimizing the $(1-\alpha)\%$ largest loss.

For a discrete random variable c over a probability distribution $\vec{p}$, CVaR may be written as [9]:

$$\sup_{q \in Q_\alpha} \sum_{i=1}^{k} q_i p_i c_i$$

Where $\alpha \in [0, 1]$ is the confidence level, and Q_α is a set of weightings associated with α:

$$Q_\alpha = \{q_i : i \in [K], \sum_{i=1}^{k} p_i q_i = 1, q_i \in [0, \alpha^{-1}]\}$$

We can replace c_i with the class conditioned loss $L_i(\theta)$, and interpret p_i as the probability of the i'th class. We then obtain label CVaR (LCVaR):

$$\sup_{q \in Q_\alpha} \{\sum_{i=1}^{k} q_i p_i L_i(\theta)\}$$

The above objective has the dual formulation:

$$\inf_{\lambda \in \mathbb{R}} \{\frac{1}{\alpha} \sum_{i=1}^{K} p_i \text{ReLU}(L_i(\theta) - \lambda) + \lambda\} \tag{2}$$

Equation 2 can be optimised with back-propagation. Like CFOL, LCVaR moves additional weight to the worst losses. The main difference with CFOL is that we don't modify the sampling of training data with LCVAR. Additionally, we can use the parameter α to control the maximum weighting that is allocated to any single class, and this has a connection with the α quantile of the loss distribution.

2.5 Brief Description of Selected UDA methods

There is a huge amount of literature on unsupervised domain adaptation techniques for classification, and there is no clear "winning" method. Therefore, we studied UDA from a more general lens. For this study we selected a subset of methods that are competitive with the current state of the art, and take different approaches to the UDA task. The first is Minimum Class Confusion (MCC) [5], a simple self-supervised method. The authors propose adding a regularizer that penalizes models that don't make confident predictions of class probabilities in the target domain. In simple terms, a "confused" model that predicts equal probabilities for a set of classes has a higher class confusion than a model that places a high probability on a single class.

In addition to this non-adversarial method, we experiment with Conditional Domain Adversarial Networks (CDAN) [6], and Margin Disparity Discrepancy [10] (MDD). As the name of the method suggests, CDAN jointly minimizes classification loss and maximizes the loss of a domain classifier. MDD, on the other hand, aims to minimize a computable version of the distribution discrepancy, minimizing the right hand side of a generalization bound like in [2]. Computation of the discrepancy term involves taking a supremum over model parameters, giving us a min-max game. The reader is referred to the original works for further details.

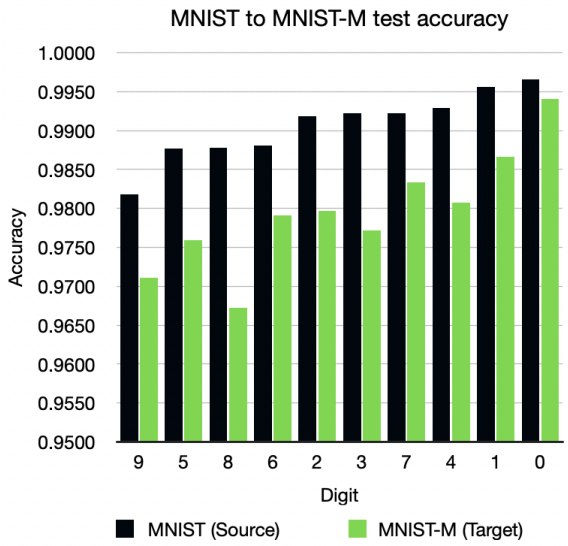

Figure 3: Test accuracies for unsupervised domain adaptation with PixelDA on the mnist dataset. The easily confused classes perform even worse in the target domain.

3 Method

3.1 CFOL with Unsupervised Domain Adaptation Techniques

The authors of CFOL [1] demonstrated that we can get marginal improvements in worst class accuracy in adversarial training. The question is, can we derive similar benefits in unsupervised domain adaptation?

In the MNIST dataset there are a few pairs of classes like 8 and 9, as well as 5 and 6, which can easily be confused with each other because the only difference between them is a few darker pixels in the lower left corner of the digit. They are also typically the worst performing classes for classification models. Likewise, if we compare the validation accuracy over classes for pixelDA (a method using generative adversarial networks), for the transition from MNIST to MNIST-M, we see the same result in the target domain. As expected, the performance of the worst classes are even worse in the target domain.

Therefore, it may be worth focusing on the pain-points of the model in the source domain. Since class re-weighting techniques or CFOL only either modify the way the dataset is sampled or the way that classification loss is computed, it is straightforward to integrate with multiple UDA techniques. We always have access to source domain labels in UDA, so we can apply CFOL to the sampling of source domain images directly (Fig. 4).

3.2 Uniform Class Sampling with PseudoLabels

Since the target domain training dataset doesn't have labels, we can't use classification errors to guide how to reweight the sampling of target domain images. We could use regular sampling for target domain images while using CFOL sampling for source images. However,

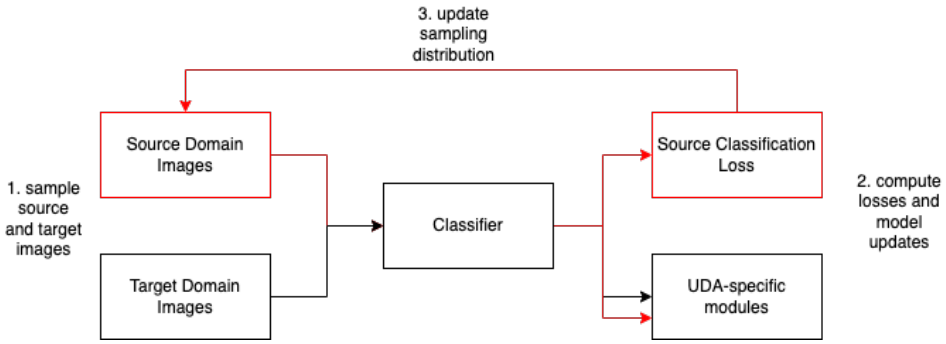

Figure 4: Integrating CFOL and LCVaR into existing UDA methods is straightforward: after each iteration we simply update the source sampling distribution or the classification loss. Flow of source domain data marked in red.

our experiments showed that in the case of uniform sampling over classes, it is beneficial to sample the target domain images in a class-conditioned fashion.

The baseline approach is to compute pseudo-labels for the target domain images, and then sample the dataset based on them. We first train each technique as-per-usual for a set of warm-up epochs. Then we use the trained classifier to predict the labels in the target domain. After this step, we construct batches of target domain images based on these pseudo-labels, and sample the source domain images with CFOL. A summary of the steps is given below:

1. Sample class i with probability p_i from source images.
2. Sample class j with probability $1/k$ from target images.
3. Form batch of source images B_s from class i
4. Form batch of target images B_t from class j, using pseudo-labels.
5. Perform forward pass, compute losses.
6. Perform a model update: $\theta_t = \text{Update}(\theta_{t-1}, B_t, B_s)$
7. Update sampling weights for source domain images.

3.3 LCVaR for UDA methods

To study the effects of LCVaR we simply replaced the source domain classification loss with LCVaR, and sampled the target domain images as per-usual. In figure 4 this corresponds to changing the upper right component.

4 Experiments

We chose to fine-tune a Resnet101 architecture for each UDA method and dataset. The classifier backbone was trained on ImageNet, and then fine-tuned on the labeled source domain images. The domain adaptation task is completed simultaneously, using the particular components from MCC, CDAN, and MDD. The classification head is replaced with a randomly initialized layer, and trained with a different learning rate from the feature extractor. The learning rate of the backbone is set to 0.001 the learning rate of the classifier head.

4.1 Evaluation Protocol

Validation and test accuracy is defined in a special way in unsupervised domain adaptation, since we can have training and validation splits for both source and target domain images. We use the labels of the training split of the target domain data to measure validation accuracy, because those labels are discarded during the model update step. Test accuracy is measured on the validation split of the target domain images. Below is a table outlining where each of the 4 splits of data are used [7].

Split	Training	Validation	Testing
Target Train	✓	✓	-
Target Val.	-	-	✓
Source Train	✓	-	-
Source Val.	-	✓	-

4.2 Hyperparameters

We use the following training hyperparameters:

1. SGD with Nesterov momentum, with a value 0.9
2. Weight decay, at a value of 0.001
3. A decaying learning rate schedule of the form $l(t) = l_0(1 + \gamma t)^{-\beta}$, with $\gamma = 0.001$, $\beta = 0.75$.
4. For CFOL, $\gamma = 0.5$. We set $\eta = 0.001$ to obtain a non-uniform distribution and $\eta = 0.0000001$ for a non-adaptive uniform distribution.
5. For LCVAR we set $\alpha \in \{0.5, 0.8\}$

4.3 Datasets

One of the most popular benchmarks in unsupervised domain adaptation is the VisDA2017 dataset (Fig. 5) [8]. The VisDA2017 dataset is a large-scale dataset for visual domain adaptation, which includes synthetic images and real images. The synthetic images were created using 3D models and rendered with various lighting conditions, backgrounds, and object poses. The real images were collected from the internet and cover a total of 12 object classes.

Figure 5: Examples images from the Visda dataset, with source images (top) and target images (bottom).

Figure 6: Pair of images from the DomainNet dataset. We use sketches (right) as the source domain images.

A documented issue with the **Visda2017** dataset is that the meaning of some class labels are confused between source and target images. For example, the "car" class in the source domain includes some images that are classified "truck" in some target domain images, whilst some target images of cars contain trucks in them. In this sense it may not allow for a perfect evaluation of a UDA classifier, because the features associated with some classes aren't uniform across domains. Nevertheless, the dataset is large and most images are labeled correctly, allowing us to get meaningful estimates of per-class accuracies in both domains. This may not be possible with the popular Office31 dataset, which has only have about 30 images for some classes.

We also made use of the less popular **DomainNet** benchmark, with 345 unique classes. DomainNet is another large-scale multi-domain image dataset that is designed to evaluate the performance of computer vision models in handling domain shift problems. The dataset contains images from six different domains, including clip art, infographics, painting, quick draw, real-world, and sketch, and has hundreds of thousands of images. We chose to focus on the "sketch" $\rightarrow$ "real" transition.

4.4 DomainNet Sketch to Real Transition

On DomainNet we ran training for a total of 10 epochs for each of the 3 methods. To test the effect of class re-weighting in the source domain, we measured the validation and test accuracies across all classes with both CFOL and LCVAR switched on and off. Setting $\alpha = 0.5$ had no effect with LCVAR, whereas setting $\alpha = 0.8$ (focus on top 20% of the loss) resulted in a marginal improvement of the worst performing target domain classes, at the expense of the better performing classes (7). The average accuracy was unaffected.

The same tradeoff occurs with CFOL (Fig. 8). Contrary to intuition, the most consistent improvements occur when we sample each class uniformly. Forgetting class re-weighting, and enforcing class conditioned sampling so that all batches contain the same class can speed up training for DomainNet (Fig. 9). Using a large $\eta = 0.001$ lead to a less uniform sampling distribution over classes, but this tends to hurt performance.

4.5 Pseudolabeling Approach on Visda2017

For MCC, MDD, and CDAN, we measured the validation and test accuracy for 3 runs each with CFOL turned on and off. Training was performed over 30 epochs. To isolate the effect of CFOL with uniform sampling, the method was switched on after 10 epochs, ie. the pseudolabels were computed before the 10'th epoch starts after which we always sample batches of images from the same class.

Even though the effect is small, it is possible to get consistently better results for the Visda2017 dataset using this method. The average accuracy across each tested UDA technique improved by an average of 1%. We can attribute this improvement to better performance in the worst classes, because once we turn on CFOL their average accuracy improves (See Fig. 11). Interestingly this didn't happen with LCVaR for either value of α.

To see this effect from a more general lens, we can also plot the minimum average accuracy recorded across the different methods. What we observe is that the minimum performance of the truck class is improved by 8% (10). The best class, "aeroplane" has a worse accuracy, but this is compensated for by an accuracy increase over the other classes. The average accuracy typically stays the same in the source domain, meaning that on the Visda2017 dataset there doesn't seem to be a tradeoff when using CFOL.

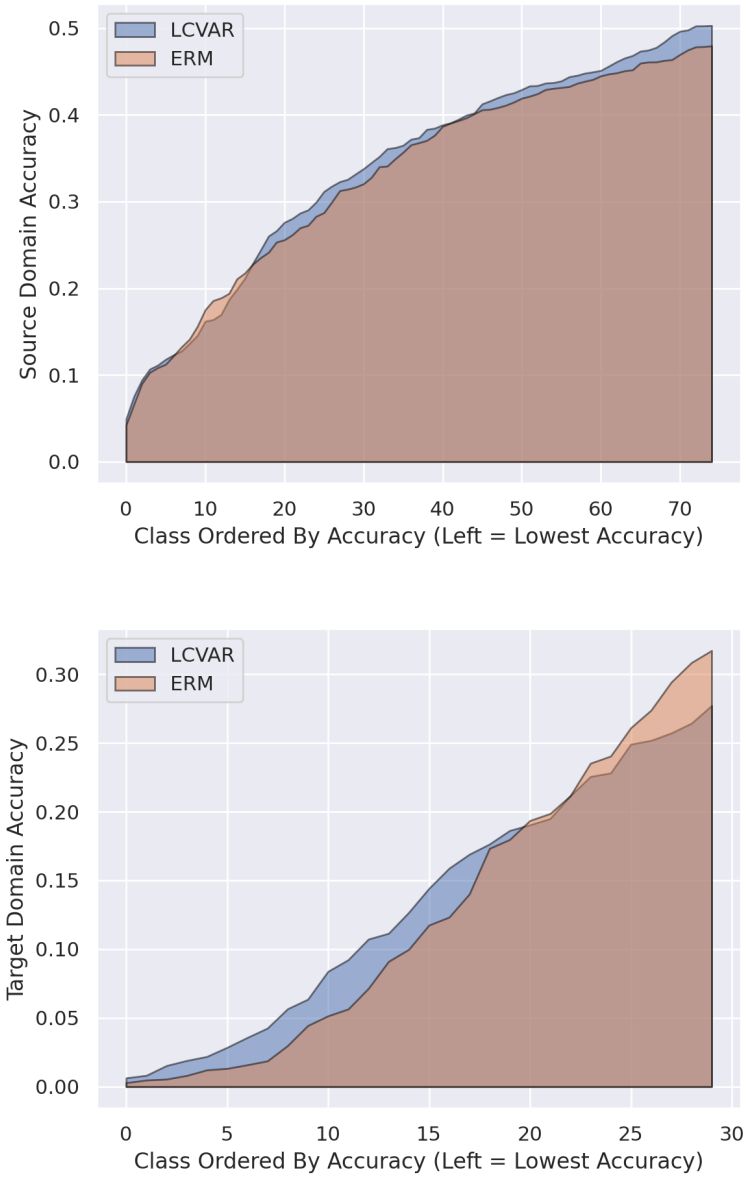

Figure 7: Average validation accuracies of the worst 20% of classes in the source and target domains for DomainNet. Average is computed over the 3 methods. In the target domain only classes with non-zero accuracies are shown.

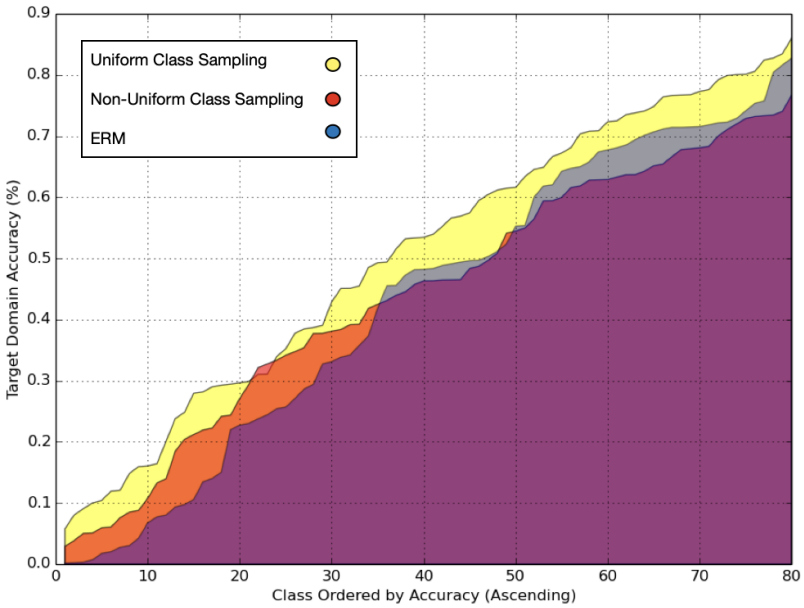

Figure 8: CFOL applied to CDAN on DomainNet. Target domain accuracies are shown for each class, ordered by accuracy, with the worst performing classes on left hand side. Using a high $\eta = 0.01$ results in a more non-uniform sampling distribution, which seems to result in an increase in worst performing classes at the expense of an accuracy drop in the best performing classes. Sampling each class uniformly seems to circumvent this issue.

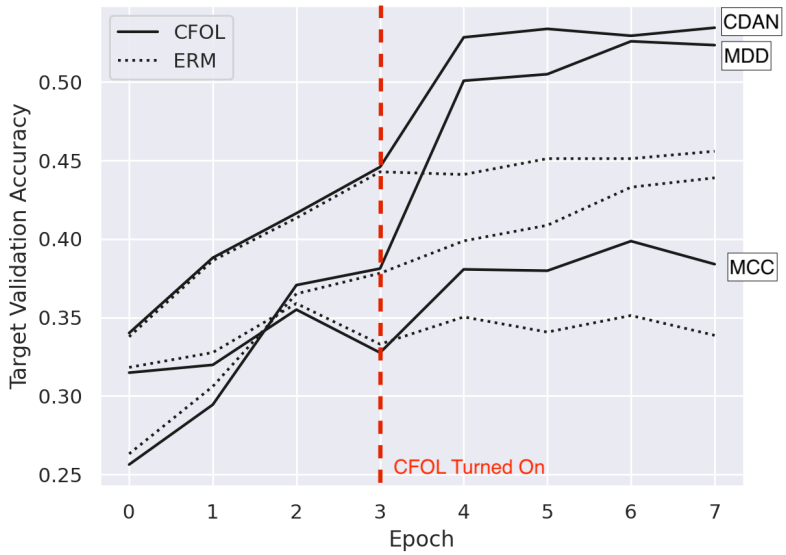

Figure 9: Sampling classes uniformly can speed up training.

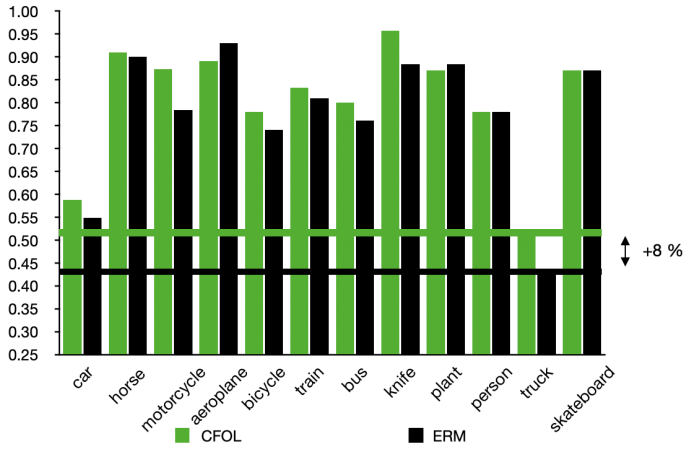

Figure 10: Plotted above is a summary of the per-class accuracy across different methods using uniform sampling over classes and CFOL. Each bar is the minimum of the accuracy recorded for MCC, MDD, or CDAN. The worst-case accuracy of the truck class is improved by 8%

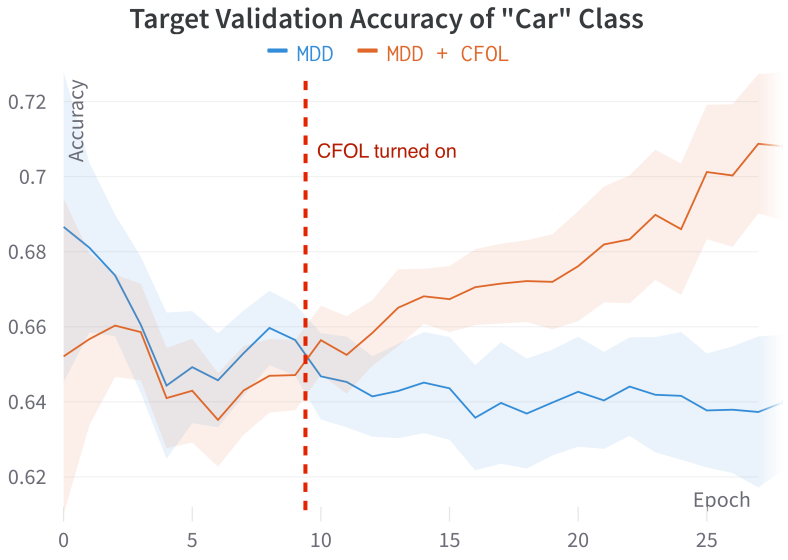

Figure 11: Illustration of turning on CFOL during training for the MDD method.

Table 1: Target Domain Test Accuracy on DomainNet

Method	MDD	CDAN	MCC	Worst Classes Improved
ERM	0.452	0.483	0.323	-
Uniform-CFOL	**0.541**	**0.551**	**0.368**	✓
LCVaR $\alpha = 0.8$	0.459	0.483	0.315	✓

Table 2: Source Domain Validation Accuracy on DomainNet

Method	MDD	CDAN	MCC	Worst Classes Improved
ERM	0.633	0.649	0.655	-
Uniform-CFOL	0.620	0.640	0.646	✓
LCVaR $\alpha = 0.8$	**0.651**	**0.658**	**0.663**	✓

5 Discussion

The results on DomainNet showed that optimising a loss function that is skewed towards the worst performing classes doesn't have much of an impact on average accuracy. However, peering into the distribution of the accuracy of all 300 class shows us that LCVAR works in a predictable manner. The worst performing classes had the largest relative improvements, and this pattern can also manifest in the target domain class accuracies. However, this comes at the cost of an accuracy drop in the best performing classes. There doesn't appear to be a free lunch.

With CFOL, changing the sampling distribution aggressively with a high η parameter hurts target domain accuracy with little to gain in the source domain accuracies. It is then surprising that if we enforce uniform sampling over classes, in both the target and source domain, we can get the largest improvements. Sampling both target and source domain batches with a single class may help with domain alignment, given that a model is constantly comparing sets of images where the biggest difference between them is the domain, not the class.

The target validation and test accuracies improved by a few percentage points across 3 methods on 2 different benchmark datasets using uniform class sampling. Our experiments show that with plug-in methods such as CFOL and LCVAR it's possible to minimize the risk of poorly performing classes, but only by a marginal amount. Out of the two approaches tried, LCVAR was more robust. While it may not have lead to the largest overall increases in target accuracy it also never deteriorated performance in either domain.

This project illustrates that it is possible to alter generalization performance of models by re-weighting a class focused loss in the source domain. Nevertheless, the current study could be improved in multiple ways. For instance, in order to solidify the findings statistics should be gathered for a broader variety of domain transitions and classifier backbones. The fact that the classifier backbone is trained on ImageNet can bias the results, and this is not commonly discussed in UDA literature. In addition to investigating why uniform class sampling worked the best, it would be worth making a fine-grained ablation study of the η and γ parameters.

Bibliography

[1] Anonymous. Revisiting adversarial training for the worst-performing class. *Submitted to Transactions of Machine Learning Research*, 2022. Under review. 3, 7

[2] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006. 2, 6

[3] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks, 2016. 3

[4] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario March, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(59):1–35, 2016. 2

[5] Ying Jin, Ximei Wang, Mingsheng Long, and Jianmin Wang. Minimum class confusion for versatile domain adaptation, 2019. 6

[6] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I. Jordan. Conditional adversarial domain adaptation, 2017. 6

[7] Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. Unsupervised domain adaptation: A reality check, 2021. 9

[8] Xingchao Peng, Ben Usman, Neela Kaushik, Judy Hoffman, Dequan Wang, and Kate Saenko. Visda: The visual domain adaptation challenge, 2017. 9

[9] Ziyu Xu, Chen Dan, Justin Khim, and Pradeep Ravikumar. Class-weighted classification: Trade-offs and robust approaches, 2020. 5

[10] Yuchen Zhang, Tianle Liu, Mingsheng Long, and Michael I. Jordan. Bridging theory and algorithm for domain adaptation, 2019. 6