Training With Less Data: Adapting Unsupervised Domain Adaptation Methods to Scaled Down Datasets for Visual Recognition Tasks

HOWARD GOODSALL

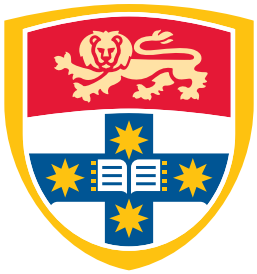
SID: 490477658

Supervisor: Associate Professor Weidong Cai
Associate Supervisor: Dr. Dongnan Liu

This thesis is submitted in partial fulfillment of
the requirements for the degree of
Bachelor of Advanced Computing (Honours)

School of Computer Science
The University of Sydney
Australia

28 May 2023



THE UNIVERSITY OF SYDNEY

Student Plagiarism: Compliance Statement

I certify that:

I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure;

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to the University commencing proceedings against me for potential student misconduct under Chapter 8 of the University of Sydney By-Law 1999 (as amended);

This Work is substantially my own, and to the extent that any part of this Work is not my own I have indicated that it is not my own by Acknowledging the Source of that part or those parts of the Work.

Name: Howard Goodsall

Signature: H. Goodsall **Date**: 28th May 2023

Abstract

One of the most significant challenges for Computer Vision AI is the difficulty of creating large datasets of labelled images for a particular task. Images must be manually labelled to ensure their correctness and even then, the images in the dataset may be inherently different to the images encountered in a real-world situation. This makes it difficult for these technologies to be implemented and used in practical scenarios. Pre-training the AI on a different dataset (the source domain) before training on the intended dataset (the target domain) can help to boost the performance of AI models when the target domain is insufficient for training on its own. Unsupervised Domain Adaptation (UDA) first trains using a labelled source domain, then trains on an unlabelled target domain. This method can be used to create models with high accuracies in situations where a large labelled dataset is not available for the specific task. Furthermore, UDA methods can be adapted to no longer require access to the source domain after using it for training, allowing the privacy of the source data to be maintained during training on the target domain. Recent developments of this method have demonstrated that it can be used to achieve much higher accuracies and can be adapted to different scenarios, such as semi-supervised scenarios where there are a few labelled examples amongst the target domain. However, the problem of collecting large amounts of data still remains as one of the biggest challenges preventing this technology from being widely adopted. This paper aims to adapt methods for performing UDA to data-deficient scenarios. The proposed solution improves UDA methods when the target domain is significantly smaller than the source domain by employing regularisation methods and performance-boosting methods for both the training data and the network architecture. This paper also examines the relationship between the size of the target domain and the resulting accuracy. By improving this relationship, we can provide a practical solution to this problem to relieve the burden of performing a large-scale data collection task on users of pre-trained models.

Acknowledgements

I would like to thank my Supervisor, Associate Professor Weidong Cai, and Associate Supervisor, Dr Dongnan Liu, for their guidance over the course of the project and for their support and encouragement of my research and thesis.

CONTENTS

List of Figures

List of Tables

CHAPTER 1

Introduction

One of the major challenges in the field of Artificial Intelligence (AI) is the availability of large datasets suitable to use for training. In the field of Computer Vision, Deep Learning AI models train on large datasets of labelled images to learn the patterns of objects in the images and determine which patterns correspond to which classes. Convolutional Neural Network (CNN) designs such as 'ResNet' have demonstrated that deep networks can reach higher accuracies than smaller networks, but are more difficult to train (He et al., 2015). In general, the more data we provide to an AI model, the more accurate it will be because it has a wider pool of information from which it can learn. However, the data provided to a model must be manually labelled with a class to ensure its correctness and must be a typical example of whatever task the model is being trained for. For this reason, it is important to gather as much data as possible in order to create the most accurate model. This has restricted the uses of these types of AI to only tasks where large datasets are easily collected or public datasets already exist. However, good performance on large datasets does not necessarily mean that the same design will perform well on small datasets and these models may not perform well in real-life scenarios when they are presented with data that is different from the data seen during training.

Recent research has shown that it is not always necessary to use extremely large datasets and industry trends have also shown a shift away from 'Big Data' towards 'Quality Data'. Despite this, an estimated 80-85% of AI projects in large companies were still in the proof-of-concept stage (Reilly et al., 2019). To bridge this gap, new methods must be proposed to make AI tasks more accessible and easier to implement in real-world scenarios. Recent research has proposed methods for creating accurate object recognition AI using unlabelled data and mixtures of unlabelled and labelled data. Whilst this alleviates the burden of manually labelling each piece of data, the availability of data still remains as a major problem, especially for niche applications. The solution outlined in this paper adapts these existing methods to improve their performance when given smaller amounts of data. The proposed method 'UDA Boost' combines methods for generating more data and uses novel techniques to help the model

adapt its training process to the smaller dataset. This method was able to provide a slower rate of decrease in performance and was able to outperform the benchmark system when the size of the dataset was reduced.

1.1 Motivation

In controlled scenarios, the training and testing data come from the same dataset and are indistinguishable whereas real-world data may be inherently different and include edge cases not found in the training data, this is known as Domain-Shift. When the dataset used to train an AI model is not large enough for sufficient training, a trade-off must be considered of whether to amplify the training process and risk overfitting versus training too little and not achieving a high accuracy. To solve this problem, transfer learning methods can be used, where knowledge learnt from one AI model is used to develop another. Domain adaptation is a type of transfer learning in which a model is first trained on one dataset, the source domain, and then trained further on a different dataset, the target domain, that is different to the source domain but is still closely related and has the same set of classes. This allows the model to start training on the target domain in a state much closer to the optimal approximation compared to a normal initialisation. Unsupervised Domain Adaptation (UDA) is a type of Domain Adaptation where the data in the target domain is unlabelled (Ganin and Lempitsky, 2015). This allows for more accurate models to be developed using small datasets because they can be trained on a large dataset beforehand. Models trained on a particular source domain can be provided as a product by producers, which consumers can then use to train on their specific target domain. Alleviating the effort on the part of the consumers can be of great use to both parties and providing stronger guarantees about the reliability of performance can encourage wider adoption of this idea.

Hypothesis Transfer Learning (HTL) is a method of performing domain adaptation without requiring access to the source domain during training on the target domain, allowing the source domain data to remain private (Kuzborskij and Orabona, 2013). Preserving the privacy of data has become especially important in the Computer Vision AI applications with the introduction of new regulations such as the EU's General Data Protection Regulation and initiatives such as Microsoft's Privacy Preserving Machine Learning Innovation, aimed at increasing the level of public trust in AI systems by providing privacy guarantees (EU, 2016; Ruehle et al., 2021). Preserving privacy also allows for the use of private data in training Recent research such as Liang et al. (2021) has also shown that HTL can be applied in UDA scenarios and still achieve accuracies comparable to non-privacy preserving methods.

1.2 Contributions

The contributions of this paper are as follows:

- A new method for performing UDA that combines existing methods with novel ideas for training a CNN to adapt to smaller dataset sizes.
- A study on the relationship between dataset size and the resulting accuracy for 3 common benchmark datasets.

1.3 Thesis Structure

In this paper we will first examine the existing methods related to this problem in Chapter 2 to examine what strengths each method has that can be applied to our problem. Next, in Chapter 3 we will outline our proposed method and evaluate its suitability to the task as well as outline the design of our experiments and discuss how to interpret them in context. Then, in Chapter 4 we present our results and discuss them. Finally, in Chapter 5 we will summarise our findings and discuss opportunities for future work.

CHAPTER 2

Literature Review

2.1 Background

As Computer Vision AI technologies such as CNNs have developed and shown their potential as powerful tools, their adoption has not followed at quite the same rate. The domain shift encountered when facing real-world data and the negative impact that it has on performance cannot be ignored in real-life scenarios and the unpredictability of this impact is often enough to deter potential adopters (Dwyer, 2021). Whilst many large datasets are publicly available for a variety of different computer vision tasks, AI in practical use requires datasets specific to each use-case to train on that task and reach a similar accuracy to that seen in a controlled scenario. Furthermore, AI models can continue to train after deployment to improve their accuracy using the real-world data and to combat the gradual change in trends and distributions of the data over time, known as domain drift or concept drift (Karimian and Beigy, 2023).

Domain Adaptation methods aim to bridge the gap between theory and reality by efficiently transferring knowledge learnt from large source domains to distinct but related tasks using smaller target domains. Domain adaptation can be performed as 'fully-supervised', where all examples in the target domain have a label, 'unsupervised', where no examples in the target domain have labels or 'semi-supervised', where there are some labelled examples and some unlabelled. The labelling process can take a lot of time and sometimes can only be performed by a person trained in the field related to the task. Manual labelling may also be considered a breach of privacy in some cases, especially if the data is personal data such as medical imaging. For this reason, it is important to consider the semi-supervised and unsupervised cases. Unsupervised learning also allows for learning to continue without human involvement after deployment, however this is rarely used because of the unreliability of un-checked training.

Recent trends in both research and real-life scenarios have also demonstrated the need for Domain Adaptation methods that can use small target domains and adapt to changing data distributions. Systems

such as IBM's Maximo Visual Inspection Tool have shown that this concept can perform well in practice and can be deployed using just a small amount of target data (IBM, 2020). This is a semi-supervised domain adaptation system for visual inspection to check for faults and defects in manufacturing. The user inputs only a few labelled examples for different types of defects and can then deploy the system. The system gathers data through edge devices such as mobile devices or fixed cameras and can continue to train either unsupervised or supervised via user feedback (known as reinforcement learning). The COVID-19 pandemic also demonstrated the need for methods that can perform more reliably on small datasets. Minaee et al. (2020) was a domain adaptation system developed in 2020, just after the start of the pandemic, to diagnose COVID-19 based on chest X-rays at a time when PCR tests and Rapid-Antigen Tests were not widely available. The system used 2000 COVID-positive chest X-rays in the target domain to train to a sensitivity rate (rate of true positives) of 98% ($\pm 3\%$).

2.2 Convolutional Neural Networks

Neural Networks (NNs) are an AI tool used to approximate a solution to a particular task by learning features of the input given to it and adapting to minimise the difference between the predicted value and the actual value (the loss). Convolutional Neural Networks are the most commonly used method of AI for image classification tasks. CNNs analyse inputted images by passing filters over them in a 'convolution' to identify which areas of the image fit the specific pattern of the filter and gradually extract features. This allows them to focus on small local areas and separate them from their position within the whole image, unlike their NN counterparts. CNNs combine filters into convolutional layers, which each operate on the same input and their combined output, called a feature map, is used as input for the next layer. CNN architectures typically use many convolutional layers and some 'fully-connected' layers. The convolutional layers use filters to extract information from the image, with each successive convolutional layer using the output of the previous convolutional layer as input. Fully-connected layers are traditional neural network layers that multiply each input by a scalar weight and sum their result to determine which pieces of information are relevant to the task. CNNs can also apply functions such as activation functions to modify the relationship between layers. The early layers of a CNN typically extract low level features, such as the edges of objects, and the deeper layers learn high level features such as the relations between these features and the last layers learn how to classify these into classes (Lee et al., 2011). The structure of the network or architecture greatly impacts its potential in achieving better results.

To train a CNN for an object recognition task, a dataset of labelled images is fed one-by-one through the network. The network then outputs a classification prediction for the class of the object in the image represented as a set of probabilities that the example belongs to each class (where the class with the highest probability is the predicted class). Then the prediction is compared with the actual label and the difference between the two is measured using a loss function such as the cross-entropy loss function and used to update each of the parameters of the network via the 'backpropagation algorithm' after a certain number of iterations. The backpropagation algorithm iteratively computes the gradient of the loss at each layer from the end to the start, which is then used to determine how to update each of the trainable parameters of the layer to gradually minimise the loss (Rojas, 1996). For unsupervised learning, the actual value is not known, so the 'true' value used for the loss function is the centre of the group of examples for each class. The strength of the weight updates is controlled by the learning rate 'η' which can be updated itself by an optimisation algorithm. The frequency of weight updates is controlled by the 'batch size' which is the number of examples that are inputted before updating the weights. After every example has been inputted the process repeats, the number of repititions is controlled by the number of 'epochs'. Once the training is complete the model is tested using the testing dataset to measure the performance on unseen examples. The testing dataset and the training dataset are two subsets from the same domain and the difference between the performance metrics on the training versus testing datasets can be used to measure the level of overfitting. If overfitting is present, then the training accuracy will be higher and the training loss will be lower than on the testing dataset. Regularisation methods can be used to reduce this difference, however over-regularisation can decrease the overall performance.

Initial research in the field of Deep Learning for visual recognition focused on creating deeper networks with more layers or wider networks with more filters per convolutional layer, however, these traditional designs quickly reached an upper limit of accuracy. Many of the state-of-the-art CNN designs were developed to compete in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), in which networks were trained on ImageNet Visual Recognition dataset, containing 1 million images and 1000 classes (Russakovsky et al., 2015). Traditional designs such as VGGNet in 2014, demonstrated that the upper limit of performance of the network was proportional to its size up until a certain point, when an overall upper limit is reached (Simonyan and Zisserman, 2015). This was due to the degradation problem, in which deeper layers were 'saturated' and unable to efficiently learn features, becoming more saturated the deeper they were (He and Sun, 2014). Unlike previous designs however, the degradation present when adding more layers to 'deep' designs such as VGGNet was not due to overfitting but

rather the network's inability to extract meaningful information after the information had already been extracted from previous layers.

To overcome the problems of traditional designs, new designs looked for ways to solve the degradation problem in deeper networks by changing the way in which information flowed through the network and better methods for controlling the learning process. Skip-connections, introduced by Srivastava et al. (2015), allow for connections between non-sequential layers. 'ResNet', introduced by He et al. (2015), used these connections to make a much deeper network, based on the idea that deeper networks should be able to perform at least as good as their shallower counterparts. Each block of layers and its skip connection forms a residual block, shown in Figure 2.1. This idea proved to work well in practice and outperformed previous designs in the ILSVRC challenge (Russakovsky et al., 2015). ResNet designs such as Resnet50 and ResNet101 (the 50 layer and 101 layer variants) have become the standard for Domain Adaptation tasks using CNNs because of their accuracy and extensibility.

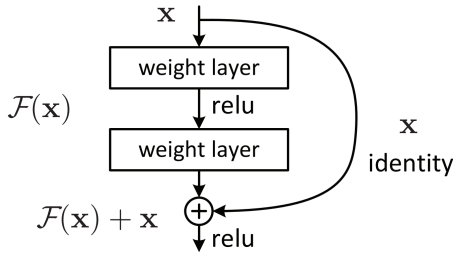


FIGURE 2.1. ResNet residual block (He et al., 2015)

More recently, Transformers, an alternative to CNNs, have emerged as a simpler solution that can achieve comparable and even better results than their counterparts (Dosovitskiy et al., 2021). Transformers do not use a typical feedforward neural network architecture and large models have been hugely successful in the field of Natural Language Processing (such as ChatGPT). They have also been successful in Computer Vision tasks, but suffer from two major drawbacks. The first is that they are unable to process local areas of an image independently because they do not use convolutions. Vision Transformer (ViT) proposed by Dosovitskiy et al. (2021) attempted to solve this problem by splitting the image into 16×16 patches and performing operations on each one. Using this method, ViT was able to achieve results very close to and exceeding that of state-of-the-art CNN architectures (Dosovitskiy et al., 2021). Many of the recent state-of-the-art architectures that have exceeded ResNet's results for image classification on the ImageNet dataset have combined CNN and Transformer architecture (Chen et al., 2023). The current state-of-the-art method for classifying the ImageNet dataset is 'EvoLved Sign Momentum' (Lion), introduced by Chen et al. (2023), which combines a CNN and a ViT-inspired Transformer, and

used a new optimisation technique to boost the performance of the two to achieve the best performance so far. The second drawback of Transformers is that they require more training data in order to outperform CNNs. ViT, for example, is only able to outperform ResNet architectures on the JFT-300M dataset, which contains 300 million training examples (Dosovitskiy et al., 2021). This makes Transformers less favourable for tasks dealing with smaller datasets, so we do not consider their use in this paper. This limitation has so far prevented Transformers from being widely adopted in the field of Computer Vision and has kept them from being more commonly used in research. However, recent trends have shown a wide adoption of Transformers for language processing and generative tasks both in the real-world and in research.

2.3 Domain Adaptation

Transfer Learning in AI is when knowledge learnt from one task is applied to another to improve the performance of the resulting system (Pan and Yang, 2010). Domain adaptation is a type of Transfer Learning in which the two tasks are closely related in some way, such as having the same set of classes but different distributions. Methods of domain adaptation have been theorised since the 1980s and as image classification AI technologies have developed, so have the methods. In general, domain adaptation is performed by training an AI on a source domain followed by a separate target domain, with techniques being applied to either specify the resulting model to only the target domain or to both tasks. For the scope of this paper we consider the case of target classification only, i.e. once the model is trained on the target domain, it will no longer be used to classify examples from the source domain. The difference between the target and source domains, or domain shift, affects how well the model can be adapted. This is commonly measured Maximum-Mean Discrepancy (MMD) or Kullback-Liebler divergence (KL divergence) (Gretton et al., 2008; Kullback and Leibler, 1951). Different forms of domain shift exist, however in the context of image classification tasks, the main type is covariate shift, in which the marginal probability distributions may be different between the two domains, but the conditional probabilities remain the same. For domain adaptation, this means that the source domain could either be very different or very similar to the target domain but the input space (the set of all possible inputs) and output space (all possible outputs) remains the same for both. Given a random input from one of the domains, the model will not immediately know which one it came from.

Early works in Domain Adaptation focused on 'shallow' methods, which are methods that deal with the domains without considering which Machine Learning method is being used to classify them. The

most common type of shallow method is Domain Alignment, which aims to identify the covariate shift in each example and 'match' the distributions to form one joint distribution from which it can learn a joint hypothesis (Farahani et al., 2020). These methods usually memorise the training data or a representation of it and then use the similarity between that and each new example to apply a transformation or mapping to the new example (Fernando et al., 2014). Many of these methods were developed as general solutions that can be used by any Machine Learning method but are now mainly used in deep learning applications. Many methods of Domain Alignment can be used for either classifying both the source and target domains or just the target domain. By aligning the distributions of the domains, they aim to create a unified representation, or a domain invariant representation, that is then used to train the model. As Machine Learning methods improved, Domain Alignment methods shifted away from being solely shallow methods, and eventually started to blend pre-existing shallow methods with deep methods, which take the classification method being used into consideration.

Methods such as those proposed by Gretton et al. (2009) and Sun et al. (2016) align the source and target domains by measuring the discrepancy between them using the statistics of each domain and then using instance based and feature based methods to perform the alignment. Instance based adaptation methods compare target domain data to source domain data to identify the similarity and use methods such as weighting to influence the prediction. Feature based adaptation methods create new representations of the data that minimises the distance between the two domains, transformations are typically used to do this. CORrelation ALignment (CORAL) introduced by Sun et al. (2016) is a method for UDA on deep neural networks using domain alignment. CORAL uses the variance between the two domains to measure the domain discrepancy, which it then uses to apply a weighting to the source domain, called Importance Sampling, so that the two are nearly indistinguishable to the network before finally applying transformations on the feature maps of the source examples at different points between layers in the network (Sun et al., 2016). The purpose of applying transformations mid-way through classification is to ensure that the source data remains regularised and to compensate for any discrepancies in the logical information learnt by the network. This method aims to ensure that the two domains are treated equally by the classification model, however it does not use Machine Learning methods to learn the differences between the domains.

Domain Alignment methods are widely used in practice and still prevalent in research, however, recent works have shown that a significant gap exists between their performance in theory versus in reality. One limitation of Domain Alignment is that the domain invariant representation is harder to classify than each

of the individual domains. Whether one domain is aligned to the other, or both domains are aligned to a domain invariant representation, the best performance that can be achieved is upper-bounded by the best performance on the individual domains (Siry et al., 2021). This is because Domain Alignment methods aim to eliminate the domain discrepancy to minimise the performance gap between domains, however, good classifiers often rely on details that might be removed by the alignment. So in the best case, the only differences between the domains is information not relevant to the classification and when this is removed the two domains are theoretically identical and the best performance possible is the same as the best possible performance on the original domain. Another limitation is that the level of memorisation required can lead to overfitting and potentially cause privacy of the source domain data to be breached. It has been found that it is possible to partially reconstruct training examples from trained CNNs, especially if they have a large network and overfitting is present (Fredrikson et al., 2015). Abadi et al. (2016) also found that reconstruction attacks can reverse the alignment method used for Domain Alignment to enhance the reconstructed data.

Domain Adversarial methods, introduced by Ganin and Lempitsky (2015) and further improved by Ganin et al. (2016), aimed to overcome the performance limitation of Domain Alignment methods in Deep Learning by adding layers to their neural network architecture to learn the difference between the two and use that information to extract the domain invariant information. Domain Adversarial methods are otherwise similar to Domain Alignment methods, and the two names are often used interchangeably, however we consider Domain Adversarial methods to be types of Domain Alignment that explicitly use deep methods to determine the domain of each example. By classifying the domain of each example, the model is able to determine the domain invariant information. This method was based off a method introduced by Goodfellow et al. (2014) that used a generative model to generate 'adversarial' examples that maximise the loss in the base model.

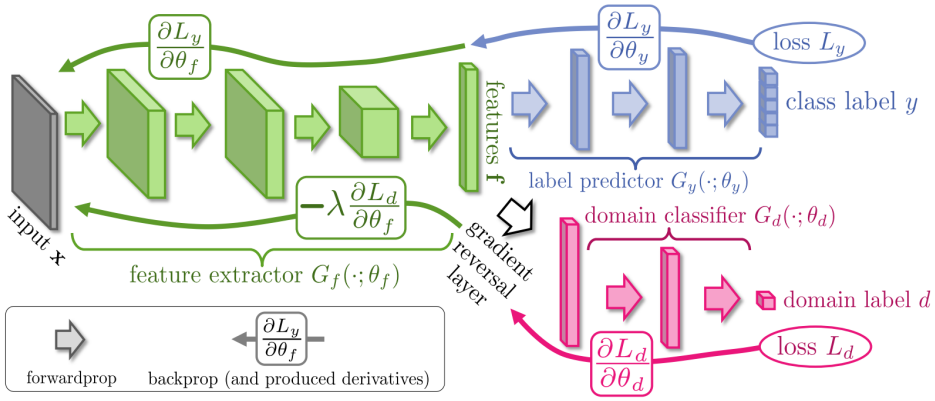


FIGURE 2.2. DANN architecture (Ganin et al., 2016)

Ganin et al. (2016) introduced the 'Domain-Adversarial Neural Network' (DANN) architecture, shown in figure 2.2, which applied the concept of adversarial examples to Domain Adaptation using target domain examples as the 'adversarial' examples. The DANN architecture splits the network into three parts, the 'feature extractor' performs the initial processing and then feeds its output to two separate parts, the 'domain classifier' for predicting the domain and 'label predictor' for predicting the class labels. During backpropagation, the loss gradient at the intersection of the two classifiers is computed by multiplying the domain classifier's loss by -1 and then adding the two (Ganin et al., 2016). This removes the domain specific information so that the feature extractor is trained using only the domain invariant information. This also helps the model to learn to extract information that is more discriminative with respect to the label classification and learn to identify irrelevant information. This method outperformed all of the state-of-the-art Domain Alignment methods, including CORAL (Ganin et al., 2016; Sun et al., 2016).

Domain Adversarial methods are able to achieve better results than their Domain Alignment counterparts due to their learning-based approach for extracting domain invariant information and label discriminative information rather than using a generalised statistics-based approach (Siry et al., 2021). However, recent works such as Bouvier et al. (2020) and Siry et al. (2021) have demonstrated that a gap exists between theory and practice in part due to the same upper-bound on performance that was present for Domain Alignment. Bouvier et al. (2020) found that using domain invariance as the basis for feature extraction makes it harder to classify target examples that are the more dissimilar to the source domain. This paper also found that the size ratio of the source and target domains affected the feature classifier by biasing it towards the larger one in architectures such as DANN and even when this was counteracted by

Importance Sampling, the model had trouble classifying edge cases in the target domain (Bouvier et al., 2020). Siry et al. (2021) found that this gap was present in many of the practical uses of this method.

Domain Adversarial methods also perform training with both the source and target domains at the same time, meaning that they operate on the assumption that both domains are available during training time. This is necessary for networks such as DANN to train the domain classifier, however, it also means that the privacy of the source domain is not preserved. Mehra et al. (2021) also demonstrated that Domain Adversarial networks could be corrupted during training via 'Poisoning Attacks', in which specially crafted data inserted during training could reduce the resulting accuracy to almost 0%.

2.3.1 Hypothesis Transfer Learning

Hypothesis Transfer Learning (HTL) is a method for performing transfer learning using only the knowledge gained on a domain (the hypothesis) and not the source domain itself (Kuzborskij and Orabona, 2013). This is very commonly used in the form of pre-trained models, in which models are made available with their network architecture and pre-trained weights for specific large datasets, such as a ResNet model trained on the ImageNet dataset. These pre-trained models can then be trained further on the target domain for the desired task. This allows network architectures that are known to perform well to be used for different tasks, and by using the corresponding pre-trained weights, the training can start from a point that is most likely going to be much better than random/standard initialisation (Yosinski et al., 2014). HTL as a network initialisation method has existed for a long time, however it has only recently been applied to the case where the target domain is different but related to the source domain. This is because Domain Alignment and Domain Adversarial methods were able to vastly outperform HTL methods until the need for privacy-preserving Domain Adaptation arose and the drawbacks of these methods was uncovered. When HTL is used for Domain Adaptation, to achieve a comparable performance, special care needs to be taken to ensure that the network is able to learn efficiently from the target domain without overfitting.

Source HypOthesis Transfer (SHOT), introduced by Liang et al. (2021), is a method for performing Domain Adaptation using HTL that is also privacy-preserving. The SHOT architecture uses a pre-trained network with two extra layers added to the end as its feature extractor and two more layers after that as the label classifier. SHOT splits the training process into two steps, shown in figure 2.3. The first step trains on the source domain and the second trains further on the target domain, however it only trains the feature extractor in the second step and not the classifier. This design is based on a practical

scenario where the first step is performed by one party and the second step by another, which preserves privacy of the source data.

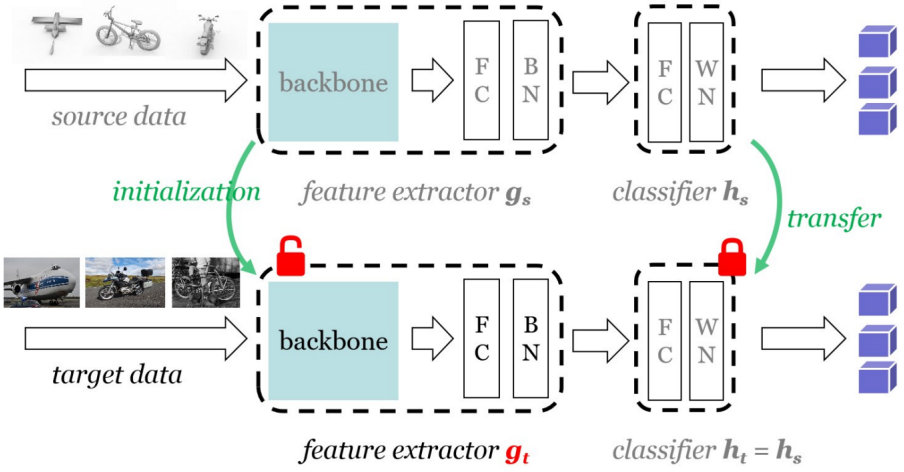


FIGURE 2.3. SHOT training pipeline (Liang et al., 2021)

SHOT on its own is not able to outperform other state-of-the-art methods, however, the algorithm can be improved by adding another training step. The MixMatch algorithm, introduced by Berthelot et al. (2019), uses training examples that were easier to classify as labelled data, using their predicted class as their label and uses the examples that were harder to classify as unlabelled data. Then the training can be performed again using pseudo-semi-supervised learning instead of completely unsupervised. The two groups of examples are created by running each of them through the model at that point (after the second step in SHOT) and measuring the entropy of the prediction, if the entropy of the example is above the average it is considered hard to classify and vice versa (Berthelot et al., 2019). Entropy is a measurement for the degree of randomness in data (different from the Cross-Entropy Loss function) and is calculated using the formula below:

$$H(x) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

The updated version of SHOT that incorporates the MixMatch algorithm, called SHOT++, was able to beat state-of-the-art Domain Alignment methods on most common benchmarks (Liang et al., 2021). The third step in the training process, shown in figure 2.4, trains the whole network using these two groups of examples. The pseudo-labelled examples and unlabelled examples are mixed together in the same batches to ensure that no training bias occurs, which could be caused by the groups containing an imbalanced proportion of classes. SHOT++ improved SHOT's results by 1.1% on average (across the benchmarks measured), however it was more effective the lower SHOT's results were and it did not decrease the accuracy (Liang et al., 2021).

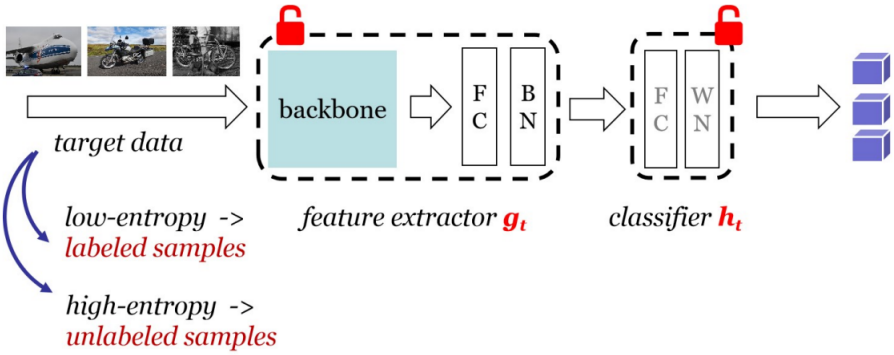


FIGURE 2.4. SHOT++ MixMatch step (Liang et al., 2021)

2.4 Scaled Down Datasets

Although one of the aims of performing Domain Adaptation is to improve the performance on smaller datasets, none of the methods mentioned so far investigate the effect on performance as the size of the target domain decreases. To investigate this relationship, we must artificially decrease the size of the target domain by 'scaling down' the number of examples. We will investigate this in this paper, using SHOT++ as a benchmark. However, other methods for improving performance on small datasets do exist and although they may not be able to improve performance drastically on their own, we can still apply them alongside domain adaptation. The most commonly used methods (aside from Domain Adaptation) are generated synthetic data and regularisation. Generated synthetic data is data that has been generated, usually based off of the existing data and augmented in some way to make it unique from the original. Regularisation techniques are techniques that prevent overfitting, which can help the model to train more heavily on the training data without decreasing the performance on the test data.

Augmenting techniques, which perform transforms such as blurring, random cropping, translation and rotations, can be considered a type of regularisation as well, since they improve the model's performance on unseen data and can prevent the need for excessive training iterations which leads to overfitting. Basic augmentations such as 90 degree rotations are commonly used by many CNNs to prevent them from learning based on the location of features. Information Maximisation is a technique for increasing the information learnt from each example using augmentations that aims to alter non-discriminate features whilst keeping discriminate features the same (Gidaris et al., 2018). Information Maximisation is not always achieved using augmentations alone, as they can also decrease the accuracy, since performing augmentations introduces a degree of randomness which makes examples harder to classify and harder

to learn from. Research in this field has yielded standard practises for performing augmentations to ensure that augmented data aids the training process instead of harming it (Kukačka et al., 2017).

AugMix, introduced by Hendrycks et al. (2020), is a method for performing a set of data augmentations that reduced the difference in accuracy between the train and test datasets of state-of-the-art CNN designs by roughly half. This method combines random variations of augmentations such as cropping, rotating, colour equalisation and sharpness adjustments. The severity and combination of augmentations for each example is randomised, meaning that the similarity to the original could range from no differences to significantly different. After transforming the data, AugMix combines the augmented example with the original example by blending the two so that the resulting image is 20% original image and 80% augmented image (Hendrycks et al., 2020). This technique was proved to increase performance on the testing dataset for many state-of-the-art CNN architectures and had a negligible impact on the performance on the training dataset. However, Hendrycks et al. (2020) only considers using AugMix as a regularisation technique and does not consider the case where the augmented data is being used to increase the size of the training dataset. To increase the size of the dataset, the same augmenting techniques can be used, however the ratio between original examples and augmented examples must also be considered, and the optimal ratio may differ between datasets (Kukačka et al., 2017).

Many different regularisation technique exist and most Deep Learning applications will use some of them by default. The most common is dropout, in which neurons are randomly made inactive during training, preventing them from over-specialising at a cost of a small increase in the loss and therefore preventing overfitting. Batch-normalisation, an alternative to dropout, performs a normalisation function between layers in a network based on the distributions present over each batch, which acts as both a regulariser and a tool for increasing the stability of training (Ioffe and Szegedy, 2015). Batch-normalisation re-scales the data at an intermediary point in the network to prevent a problem called 'internal covariate shift' in which the distributions of the data gradually change and destabilise the training process. Regularisation can also be performed by altering the way the loss is calculated. Label smoothing introduces random noise to labels before they are used by the loss function to prevent over-confidence in predictions as well as to prevent overfitting (Müller et al., 2020). When the loss is being calculated, the label is inputted as a one-hot encoded vector, where for a task with 'n' classes, the vector contains $1\times$'1's and n-$1\times$'0's. Label smoothing uses small noise values instead of 0 and 1 minus a small constant instead of 1 to prevent the harsh training encountered when using one-hot encoded labels for the loss function.

2.5 Summary

Key Idea	Summary
ResNet (He et al., 2015)	Introduced a CNN design based on the idea that deeper networks should be able to perform just as good as shallower ones and achieved state-of-the-art performance for its time.
CORrelation-ALignment (CORAL) (Sun et al., 2016)	One of the first methods for applying Domain Alignment to Deep Learning. It uses the variance between domains to measure their discrepancy and performs the alignment via weighting the source domain (called Importance Sampling). It also performs alignments at intermediate points in the network.
Domain-Adversarial Neural Network (DANN) (Ganin et al., 2016)	Introduced Domain-Adversarial Domain Adaptation that detects which domain an example came from by training a domain classifier to learn the differences between domains. During training, the first section of the network, the 'feature extractor' is trained using only the domain invariant information of each example by removing any information specific to the domain.
Source Hypothesis Transfer (SHOT/SHOT++) (Liang et al., 2021)	A Domain Adaptation method that transfers the knowledge learnt on the source domain (the source hypothesis) to the target domain using an altered training method. This method also does not require the source domain to be available during training on the target domain, allowing for the privacy of the source domain data to be preserved.
MixMatch (Berthelot et al., 2019)	A method used by Liang et al. (2021) in SHOT++ to boost the performance on the target domain by training using mixed batches of labelled and unlabelled examples. The method measures the entropy of the predictions for each example using the partially trained network to determine how difficult it is to classify that example. If the prediction had below average entropy, its prediction is used as the label, if it was above average it stays unlabelled.
AugMix (Hendrycks et al., 2020)	A technique for altering data via augmentations such as random cropping and colour shifting introduced as a regularisation technique. This technique aimed to create a standardised procedure for performing data augmentations to ensure that they improve performance rather than degrade it.

TABLE 2.1. Summary of key findings

CHAPTER 3

Methodologies

3.1 Overview

The Domain Adaptation method proposed in this paper aims to apply an extra training step to a typical example of Hypothesis Transfer Learning for UDA to improve the performance of the model on the target domain when the size of the target domain is reduced. The motivation behind doing so is to demonstrate that the negative correlation between domain size and performance can be slowed by deliberately changing the network to adapt to the reduced size. We propose a new method 'UDA Boost' that uses data augmentations to increase the size of the target domain and the MixMatch method to increase the stability and generalisation as well as introducing two new ideas, a 'target classifier' based on a ResNet residual block (Figure 2.1) to the network architecture to adapt the classification task to the target domain and a method for enhancing the training process called the 'source hypothesis optimiser'. This method was tested on 3 popular benchmark datasets for domain adaptation in object classification tasks, Office-31, OfficeHome and a group of three datasets: MNIST, SVHN and USPS, referred to as 'Digits' (Saenko et al., 2010; Venkateswara et al., 2017; Deng, 2012; Netzer et al., 2011; Hull, 1994)

This new method follows the same general structure as SHOT and SHOT++ proposed in Liang et al. (2021), using the same transfer learning techniques to adapt the network to the target domain. Rather than solely applying the MixMatch algorithm in the third step like SHOT++, we instead apply the UDA Boost methods to adapt the network to the target domain further. We also employ a new optimisation method based on DANN's approach proposed in Ganin et al. (2016) that uses the classifier trained on the source domain alongside the classifier trained on the target domain to enhance the training during the UDA Boost step. This new method compares the loss of each of the two classifiers, if the loss from the target classifier was larger than the source classifier then it trains the feature extractor in that iteration, otherwise it does not.

To test this new method, we train the network on each combination of sub-domains for each of the three datasets using one sub-domain as the source and another as the target domains. For each combination we train using the entire source domain and scale down the size of the target domain. The target domain is first split into a training and testing set using a ratio of 9 training examples to every 1 testing example whilst also keeping the same distribution of classes. Then the training portion is further scaled by factors of 1, 0.75, 0.5 and 0.25, again keeping the class distribution the same. Each combination of $source \rightarrow target$ is trained using each scaled target training dataset and testing using the unscaled testing dataset.

3.2 Proposed Method

The design of the training process used for UDA Boost aims to emulate a typical use case of UDA in the real-world where the model is pre-trained on the source domain, then trained transferred to the target domain and trained further. The 'transfer' step is assumed to take place between two different parties a producer and a consumer, and it is also assumed that the privacy of the source domain must be preserved during this transfer so that at no point can the consumer access the source domain. It is assumed that the producer has access to all of the source domain so we will always train using the entire source domain and only scale down the size of the target domain.

We divide the training process into three steps where each step trains the model from the previous step. The first step trains on the source domain, the second step then trains on the target domain and the final step, the 'UDA Boost' step, performs further training on the target domain. The first two steps are in theory identical to the training pipeline of SHOT and SHOT++ and uses the same general network architecture, with the third step being different in both regards. For this reason we will use these as a benchmark for our system. Since SHOT++ was the better performing of the two, we aim to outperform this specifically.

The main network used for SHOT, SHOT++ and UDA Boost consists of a feature extractor and classifier. The feature extractor's purpose is to analyse the image and extract information that the rest of the network can then use for classification. The classifier's purpose is to determine which pieces of information are relevant and how they can be used to determine the class. The feature extractor consists of a pre-trained 'backbone' network and two neuron layers. The backbone network used is the 50-layer variant of ResNet, pre-trained on the ImageNet dataset, which performed the best on most

benchmarks in SHOT and SHOT++ (Liang et al., 2021). This means that this step is actually already performing a type of transfer learning, as it uses the pre-trained model to initialise part of the network. The backbone network contains all of the convolutional layers in the network. The two layers afterwards are a fully-connected layer (FC layer) and fully-connected layer with batch-normalisation (BN layer). The classifier consists of an FC layer and a fully-connected layer with weight normalisation (WN layer). Weight-normalisation is a method for stabilising the training process introduced by Salimans and Kingma (2016) that performs updates to the direction of the weight (i.e. positive or negative) separate from the magnitude.

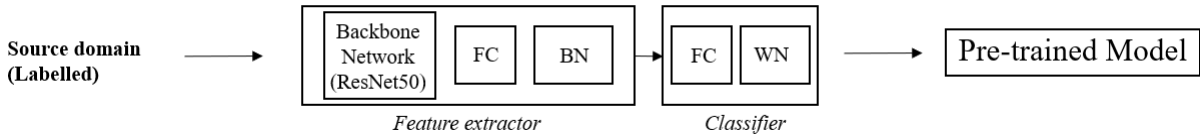


FIGURE 3.1. Step 1: Source Training

The first step, shown in figure 3.1, trains on the labelled source domain and produces a 'pre-trained' model ready for transfer. This step is the same across SHOT, SHOT++ and UDA Boost. Since the source domain is labelled, this step is performed as fully-supervised, which allows it to reach a relatively high accuracy. The aim in this step is to minimise the loss on the source domain to provide the best starting point for the target domain training. We also employ label smoothing as a regularisation technique during training. This is used to stabilise the training process over the large source domain and prevent overfitting. Since we know that the source domain and target domain are at least somewhat related we assume that the best starting point for the target domain to learn from is the optimal model for the source domain. After this step the source domain is not used, however there may be some risk to the privacy of the source domain if overfitting is present via reconstruction attacks. Although, the effectiveness of reconstruction attacks on CNNs is low and using any sort of regularisation technique reduces this risk significantly.

The second step, shown in figure 3.2, performs the transfer to the unlabelled target domain using the pre-trained model from step 1. This step also adds random 90 degree rotations to the data and uses a rotation classifier to classify the angle of rotation to perform Information Maximisation. The addition of the rotation classifier makes it so that the feature extractor does not learn the rotation as a feature, as the loss that is passed to the feature extractor does not contain information regarding the rotation, similar to the way Domain Adversarial networks use a domain classifier. In this step only the feature extractor and rotation classifier are trained and not the classifier. This is because by preserving the source hypothesis

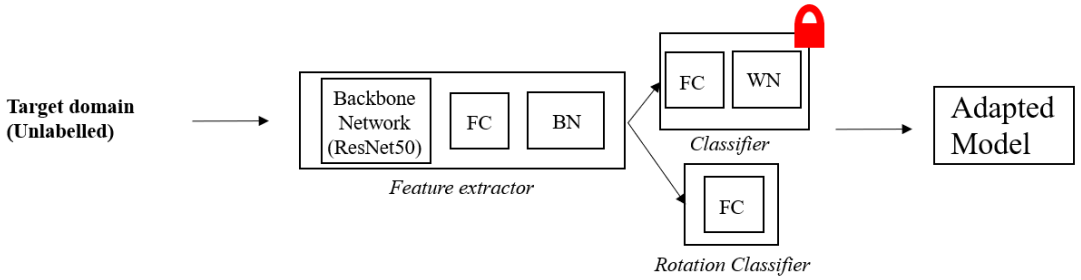


FIGURE 3.2. Step 2: Target Domain Training

of the classifier, we increase the Information Maximisation of the feature extractor. If we were to assume that the model after step 1 is able to perfectly classify examples from the source domain, we can also assume that the increase in loss after transferring to the target domain is entirely due to the differences between the domains. By not training the classifier we are therefore training the feature extractor to determine the most discriminate information that can then be used to minimise the loss.

Without weight updates to the classifier, the responsibility for minimising the loss falls to the feature extractor. This means that the feature extractor is more likely to rely on domain invariant information but it does not restrict it to only that information. As we have seen with Domain Adversarial methods, some of the domain specific information may be useful to the classification, however at this point this information is still processed using the source hypothesis. We also know that the model after step 1 is not guaranteed to be a perfect classifier and therefore neither will the model after step 2.

3.2.1 Benchmark

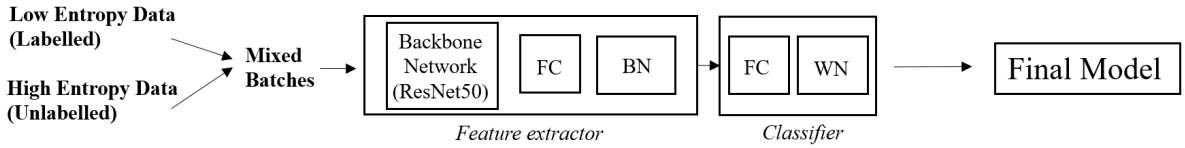


FIGURE 3.3. Step 3: MixMatch

Shown in figure 3.3 is the MixMatch step of SHOT++, which uses the adapted model from step 2 to split the target domain into low entropy and high entropy groups, using the predicted labels as the pseudo-labels for the low entropy group. This step trains the entire network using mixed batches of pesudo-labelled and unlabelled data. Using mixed batches allows for the network to give more confident classifications and to learn more discriminate features. Since at this point the feature extractor has

already been trained to identify the most discriminate features of the target domain using the source hypothesis, we now focus on training the classifier to use this information to minimise the loss. In general, the backpropagation algorithm updates each parameter according to how much it contributed to the loss. Since we have already minimised the loss using the feature extractor alone, we now reduce the learning rate of the feature extractor by $\eta \times 0.1$, whilst keeping it the same for the classifier. This is to ensure that the training of the classifier does not interfere with the feature extractor too much and keeps it stable. This step is used in the place of the proposed solution to provide a benchmark with which we can compare our results.

3.2.2 UDA Boost

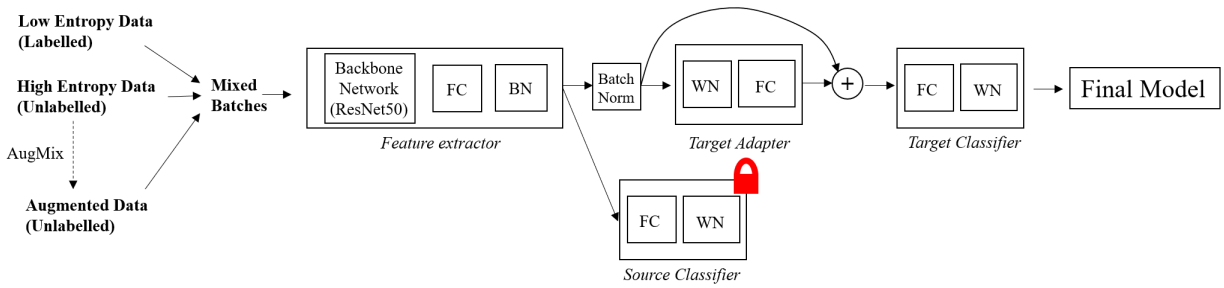


FIGURE 3.4. Step 3: UDA Boost

The proposed solution, UDA Boost, combines four main methods, two data-oriented and two network-oriented. The data-oriented methods are used to enhance the training data by increasing the size through generated synthetic data via data augmentations using the AugMix method, and increasing the confidence of predictions during training through pseudo-labelling utilising the model from the previous step via the Mixmatch method. The network-oriented methods enhance the network architecture by enabling stronger and more specified training to allow the network to adapt to the differences between the source domain and target domain using two novel ideas. The first is the 'target adapter' block of layers added to the network and the second is the 'source hypothesis optimiser', which utilises the frozen source hypothesis to only train the feature extractor when necessary. In this step we train the whole network with the exception of the 'source classifier' and the modified training process of the feature extractor. The source classifier and target classifier are initially identical, with both being trained in Step 1 and 'frozen' (i.e. not trained) in Step 2. In this step, we do train the target classifier and it is located at the end of the network, whereas the source classifier retains its position after the feature extractor and remains frozen.

Before training begins, the model from step 2 is used to split the target domain into high and low entropy groups as per the MixMatch algorithm. Then, new data is generated from the high entropy group using the AugMix algorithm, an example of augmented data is shown in figure 3.5. Then these groups are interleaved into batches consisting of equal amounts pseudo-labelled examples from the low entropy group and unlabelled original examples from the high entropy group and augmented examples from the high entropy group whose amount depends on a predefined ratio (values between 0.5 and 3.0 are used). Both of these methods follow the same general method as outlined by their respective papers, however the combination of the two in this way has not been tested, so we also need to consider the way in which they interact. A feature shared between both of these methods in their original forms is the increased use of regularisation techniques. For the MixMatch algorithm, regularisation is used to ensure that the network does not learn to over specify on the pseudo-labelled examples. For the AugMix algorithm, regularisation is used to ensure that the network learns to extract features from across the augmented examples rather than the augmentations themselves. To accomplish this, we employ a Batch Normalisation step after the feature extractor and use label smoothing to apply regularisation during training. These methods ensure that the network trains efficiently on each of the types of examples within the mixed batches. These operations make up the first two methods used by UDA Boost and are shown on the left hand side of figure 3.4.

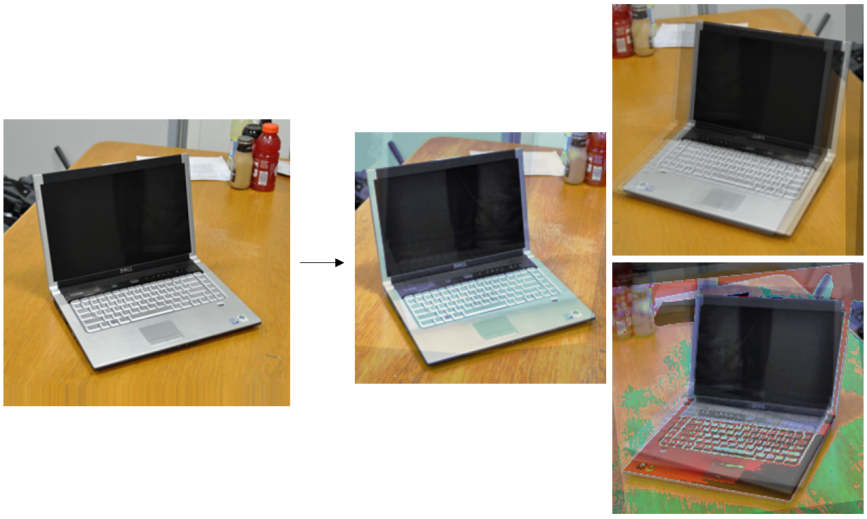


FIGURE 3.5. Data Augmentations via AugMix example from the Office-31 dataset (DSLR)

The UDA Boost method performs two modifications to the network architecture. The first is a new set of layers is added to the network located between the feature extractor and 'target classifier' shown in

figure 3.4, we call these new layers the 'target adapter'. The target adapter block consists of two FC layers, with the first one additionally using weight-normalisation, and a skip-connection that bypasses this block. These new layers are added as entirely untrained and are designed to act as another step that can help the classifier to process the information output from the feature extractor. However, under normal circumstances adding more layers to a network mid-way through the training process would interfere with the structure of the information flow in the network and is guaranteed to degrade the performance. To counteract this, we use a skip-connection to retain the information from the feature extractor in the output to the target classifier. This follows the same idea that was presented in ResNet in that deeper networks should be able to perform just as well as their shallower counterparts. By adding the target adapter block we theorise that the network should be able to perform just as well as without it. However, in practice this may not always be the case as we are not certain that the network will be able to learn to perfectly identify when the output of this block is better or worse than the output from the feature classifier. The outputs from feature extractor (after batch normalisation is applied) and feature extractor are combined by summing the two as done by ResNet (He et al., 2015). Then, the combined output is sent to the two classifiers.

The 'source hypothesis optimiser' uses two classifiers to produce two sets of predictions, with which it can calculate two losses. Both classifiers are copies of the classifier from step 2, which has not been trained since step 1, so initially they are identical but the source classifier is not-trained in this step whilst the target classifier is. If the loss of the target classifier is greater than the loss of the source classifier then we train the feature extractor using a reduced learning rate of $\eta \times 0.1 \times \frac{1}{batchsize}$. At the start of the training, since the target adapter layers are untrained, we can assume that the loss from the target classifier will be greater than the source classifier. The more we train the network, the less likely it is that the target classifier loss will be greater than the source classifier's, so there will be fewer instances where the condition is true and we will train the feature extractor less and less. This ensures that we still train the feature extractor in this step but only when necessary and only small amounts.

3.2.3 Justification of Proposed Method

The underlying motivation for the design choices of this method is to provide the network with more data and more layers so that it can learn to specify itself to the target domain whilst also utilising the knowledge learnt in previous steps. However, each method also has a possibility of degrading performance which we must consider. The methods have been designed to improve the performance as the size

of the target domain decreases, however there is a risk that their introduction decreases the performance by a constant amount which could negate any benefit.

By generating more data via augmentations, we aim to increase the pool of examples from which it can learn the classification hypothesis. It is necessary to use data augmentations rather than just input the same examples more times (i.e. by increasing the number of epochs) because data augmentations alter the contextual information whilst keeping the class information constant. The difference between an example and its augmented version can be viewed as being more like the difference between two examples of the same class, rather than two of the same example. When the size of the target domain is scaled down, we would need to either increase the number of times that each example is inputted, which increases the risk of overfitting, or increase the learning rate, which carries the same risk and is also more unstable. Supplementing the target domain with augmented data mitigates these risks, however it also introduces the risk of degrading performance.

Since augmented data is generated artificially, some of the discriminative information that can be used for classification may be removed or even worse, altered in a way that counteracts the hypothesis and causes it to be more similar to a different class. The removal of discriminative information will make it harder for the network to classify. We expect that this will occur to some extent but if the amount of information removed is not significant it will act as a regularisation tool as well. We can measure this using the mean entropy and mean loss, since we expect that if examples are harder to classify the predictions will become more unstable. If some classes are more similar to others in the domain, there is a risk that using augmentations may alter the discriminative information of examples of a class such that it overlaps with the discriminative information of another class. This would cause the network to attempt to learn to identify this difference, however, this overlap would not be present in the test data, meaning that the performance on the test dataset would decrease. The likelihood of this occurring is low since we are using a standardised procedure of augmenting and we know that there are distinct differences between classes, however in real-world scenarios classes may be more similar to each other and this effect may be more prominent.

The use of the MixMatch algorithm allows for the network to perform more specified training on the target domain by utilising the most confident predictions of the semi-trained network. Since in unsupervised learning the loss is calculated based on the similarity to other predictions of the same class, more uncertain predictions will be more spread out and dissimilar to examples of the same class. By using supervised learning for the already confident predictions, we can remove this noise and instead use an

absolute value for the class prediction, i.e. a one-hot encoded vector with one '1' denoting the class value and the rest are '0's. Since this is not necessarily the true value, we also adopt label smoothing to reduce the harshness of the loss calculation, which reduces the '1' value by a constant 'ϵ' and increases the '0' values by '$\frac{\epsilon}{k-1}$'. We empirically set ϵ to 0.4, which is considerably higher than the default of 0.1. Using a higher ϵ value regularises the loss function more aggressively, which under normal circumstances may degrade performance but given that we have introduced more randomness to the input data with the data augmentations, we want a higher level of regularisation. However, this increased regularisation may have the unintended effect of reducing the effectiveness of the pseudo-labelled examples.

The addition of the target adapter allows the network to learn more complex relationships between features and allows it to specify more to the target domain. Whilst under normal circumstances, adding new layers to a partially trained network would interrupt the information flow and greatly degrade performance, the addition in this scenario is aided by three factors. The first is that the rest of the network is already partially trained, although to different extents, meaning that the information being provided to these layers is easier for them to process, reducing the presence of random variations in the data. The second is that the skip-connection allows for information to bypass these layers, which also prevents the target classifier from being destabilised by the training of the target adapter. The third is that the input to this layer has passed through the Batch-Normalisation layer, which not only stabilises the data but also ensures that fewer weight updates are required to get the target adapter layers from their default initialisation to the optimal point.

The main concern with the addition of the target adapter is that since the three main portions of the network (feature classifier, target adapter and target classifier) are each at different stages of training, the target adapter will be unable to learn efficiently. This may be especially present whilst all three are training at the same time. Another factor that may destabilise the training is that if the output from the target adapter is worse than the output before it, the network will have to gradually learn to ignore it whilst also training the other two portions. This could prevent the other portions from reaching their optimal point and slow down the learning process.

The source hypothesis optimiser is a novel idea proposed in this method that aims to solve the problem that was present in the SHOT++ method of needing to train the feature extractor without over training it. Training the feature extractor in the third step of SHOT++ (the Benchmark step in our pipeline) allowed it to train using the pseudo-labelled data and it also allowed for the network to train as a whole, which makes the training process more stabilised. The method used by SHOT++ was to simply use a reduced

learning rate of $\eta \times 0.1$ (Liang et al., 2021). Our method aims to determine exactly when to train the feature extractor based on the loss comparison between an approximate duplicate of the model from step 2 and the new model in the current step. We can use the comparison condition as a metric of how well the current model is performing versus the old model and as the current one is improving, we train the feature extractor less to preserve the hypothesis that is causing this result.

A limitation of this method is that we only train the feature extractor until the model starts performing better than the old model but the feature extractor could still need more training to reach the optimal point. This is reflective of the limitations that we saw of other methods of Domain Adaptation such as Domain Alignment methods where the performance of the model was upper bounded by the best performance of the individual models. However, in our case the performance of the feature extractor is being upper bounded by the previous step and since the target adapter and target classifier continue to train after the feature extractor has stopped, the overall performance is not upper bounded by anything.

3.3 Datasets

Datasets used for Domain Adaptation in controlled scenarios consist of multiple subsets of data from different sources that all share the same set of classes. Images in each subset for each class will have the same kind of object in the centre, however the background, framing, angle, picture quality, etc could all be different. The domain discrepancy between any two given domains and subsequently the differences in performance between them may depend on any or all of these differences. Domains within the same dataset may be very different or very similar to each other, however we do not measure this difference before the training process, even though this measurement may be useful in aiding the adaptation process. This is because in our system it is not necessary to measure this difference as we aim to only use the resulting model for classification of the target domain. Also, measuring the domain discrepancy requires access to both domains, however in a privacy-preserving system such as this, at no point does the system have access to both domains at the same time.

The Office-31 dataset is the smallest of the three datasets, with the majority of examples belonging to the Amazon domain. The DSLR and Webcam domains contain images of the same objects taken with different cameras, meaning that the domain discrepency between these two is expected to be small.

(A) Amazon (B) DSLR (C) Webcam

FIGURE 3.6. Office-31 Sample for 3 classes: Backpack, Bike and Laptop

	Image Type	**Total Number of Examples**	**Average Number of Examples per class**	**Resolution/Size**
Amazon	Product pages on Amazon	2817	90	Differs
DSLR	Images taken with a DSLR camera	498	15	4288×2848
Webcam	Images taken with a webcam camera	795	25	640×480

TABLE 3.1. Office-31 Dataset Summary

(A) Art (B) Clipart

(C) Product (D) Real World

FIGURE 3.7. Office-Home Sample for 2 classes: Bottle and Flowers

	Image Type	Total Number of Examples	Average Number of Examples per class	Resolution/Size
Art	Objects in art	2427	37	Differs
Clipart	Cartoon objects	4365	67	Differs
Product	Images taken from online retailers	4439	68	Differs
Real World	Images from search engine results	4357	67	Differs

TABLE 3.2. Office-Home Dataset Summary

The Office-Home Dataset consists of 4 domains of images from different sources. The Clipart, Product and Real World domains contain roughly the same amount of examples whilst the Art domain has roughly half the amount of the others. This dataset is considered to be the medium size dataset.

(A) MNIST (B) SVHN (C) USPS

FIGURE 3.8. Digits Sample for 3 classes: '0', '1' and '2'

	Image Type	Total Number of Examples	Average Number of Examples per class	Resolution/Size
MNIST	Handwritten digits	70000	7000	28×28
SVHN	Photos of house numbers	99289	9929	32×32
USPS	Handwritten digits on mail	9298	930	16×16

TABLE 3.3. Digits Dataset Summary

The digits dataset is made from three similar datasets that we will use as domains. Since these datasets are not specifically made to be similar to each other, they are not guaranteed to produce results as

consistent and reliable as the others. This dataset is considered the large dataset for our experiments, however the sizes of the domains varies a large amount. The MNIST and SVHN domains contain a large number of examples, whilst the USPS domain contains roughly $\times 0.1$ the number of examples as the others. This size discrepancy may cause the adaptation between the larger datasets and the USPS domain to not work as intended, however we will investigate this problem. The MNIST and USPS images are very similar to each other and are in greyscale, whereas the SVHN images are in rgb colour, meaning that we need to convert the MNIST and USPS images to rgb colour in the pre-processing step as well. The SVHN images may also contain partial numbers from numbers adjacent to the one in the centre, as shown in figure 3.3 (B), which may affect the performance. We will consider the results for these experiments with these limitations in mind.

3.4 Experiment Design

The aim when designing the experiments is for the experiments to reflect the real-world use case as closely as possible. This is to both ensure that the results are a more reliable representation of the real-world and so that the project can be considered a proof-of-concept for the ideas presented. We consider the real-world use case to be where the network is trained on the source domain by one party and then transferred to the target domain by another party. We assume that each party on has access to their respective domain and that no specific information about the source domain, such as the size or performance metrics, is available when training on the target domain other than the knowledge that the two domains are at least somewhat related and share the same set of classes. This means that hyperparameters such as the ratio of augmented data to the original data and the number of epochs to run for must not depend on knowledge about the source domain. We also assume that for each model trained on the source domain, there could be multiple different target domains that use it. This is reflective of a real-world scenario where a company might have only a few source domains covering a range of different applications and clients can assess the similarity between their own application and the one offered by the company themselves. For this reason, no similarity metric between source and target domain is used to aid the training process.

Each individual experiment is performed by iteratively training the model following the training pipeline steps outlined by UDA Boost. We refer to the output from step 1 as 'source', step 2 as 'target' and the two versions of step 3 as 'Benchmark' and 'UDA Boost'. Since we only perform step 1 using the entire source domain, we test it on the entire target domain as well. For each of the three datasets (Office-31,

Office Home and Digits) we test each combination of source and target domain, using one domain at a time as the source domain and using each of the others in the dataset as the target domain. For the Office-31 dataset there are three domains, Amazon, DSLR and Webcam, so there are 6 combinations. For the Office-Home dataset there are four domains, Art, Clipart, Product and Real World, so there are 12 combinations. The Digits dataset consists of three domains that are actually separate datasets, MNIST, SVHN and USPS, so there are also 6 combinations, however since these datasets were not originally created for the purpose of Domain Adaptation, many papers that use them as a benchmark, such as Liang et al. (2021) for SHOT/SHOT++, only use some combinations of source→target. This is because they have found that some combinations work as intended and are typical examples of Domain Adaptation, but other combinations do not. The combinations used by Liang et al. (2021) are MNIST→USPS, USPS→MNIST and SVHN→MNIST. This paper will consider all combinations, however we will view the results of the other combinations within this context and investigate why this may be the case. The experiments were conducted using a computer with 8GB of RAM and 4GB of GPU memory. We use the same hyperparameters as SHOT and SHOT++ for our tests, with the exception of the batch size, which is set to 32 instead of 64 for steps 1 and 2 of the training. This was because 32 is the maximum batch size possible for this system when using 4GB of GPU memory. This change may cause the results to be different from those presented by SHOT and SHOT++. The 9:1 ratio used to split the target domain into train and test may also cause the results to be different. We will also test the source trained model (i.e. after step 1) on the target domain, however since we only reduce the size of the training set of the target domain, the results on the source trained model will not change as we change the size of the target domain so we only include 1 result for each group of experiments.

3.4.1 Metrics

It is important to measure the performance of the models using a measurement that is reproducible, constant across experiments and representative of how good the model is at the task overall. The main metric we use to measure the performance of our system is the accuracy of the model on the testing dataset. The testing dataset is a subset of the target domain created by separating the target domain into the training and testing datasets using a ratio of 9:1 (train:test). There is no overlap between the testing and training datasets, so every example in the testing dataset has not been trained on by the model. Both datasets are created by randomly sampling the examples of each class, so the number of examples of each class present in both datasets remains roughly constant. The training dataset is also not reduced like the training set is when we scale-down the dataset size. We use a ratio of 9:1 because it ensures

that the testing dataset is sufficiently large enough to encompass a range of different examples whilst ensuring that the training dataset is not reduced by an amount significant enough to influence the results. The testing accuracy can be seen as the most reliable metric of the overall performance as it reflects the aim of the task as a whole, which is to develop a classification system that can reliably classify unseen examples for a given scenario. It is especially important for us to only use unseen examples as testing on examples that were also used for training may be influenced by overfitting. When we train the model on the source domain however, we train on the entire source domain (without splitting it into train and test sets) and test it only using the target domains. This is so that we can measure the performance increase between steps 1 and 2, but this is only considered within the context of the experiments since in a real-world scenario we would not be able to measure this.

We measure the accuracy of the model on the same testing dataset at each step in the training process and for each size of scaled target domain. The difference in performance between steps 1 and 2 for the unscaled target domain can be considered a measure of the domain discrepancy between the source and target domains. The lower the difference in performance is the more similar the two domains are to each other, however this is not a perfect measurement of the domain discrepancy. This measurement may also be influenced by other factors such as the relative sizes of the domains and the difference between training from scratch in step 1 versus training from the source-trained model in step 2. We can also compare the results of the same tests between different domains as an alternative as well as to determine which domains are inherently harder to classify.

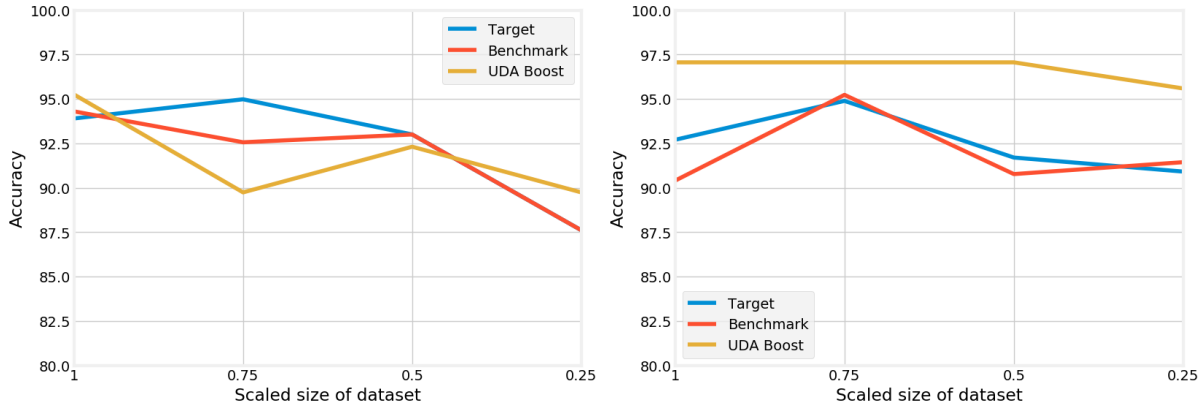
We also measure the mean entropy of the predictions on the testing dataset for the Benchmark and UDA Boost to determine the level of randomness present in the predictions. We use entropy rather than a traditional measurement like loss because for unsupervised learning it is more important to consider the randomness of the predictions rather than their loss since the loss is measured relative to the other examples. The mean entropy measures how confident the predictions are and how similar the predictions for examples of the same class are. Like the mean loss, a high mean entropy does not necessarily mean that the testing accuracy will be low, however, there will most likely be a correlation between the two. We also expect to see an increase in the entropy of each prediction with the introduction of augmented data, as we assume that increasing the randomness of the examples will also lead to an increase in the randomness of the predictions. Although, it is also possible that the mean entropy decreases since the augmented data is meant to enhance the knowledge learnt from each training example, which may allow the model to make more confident predictions.

CHAPTER 4

Evaluation

4.1 Office-31 Results

4.1.1 Amazon



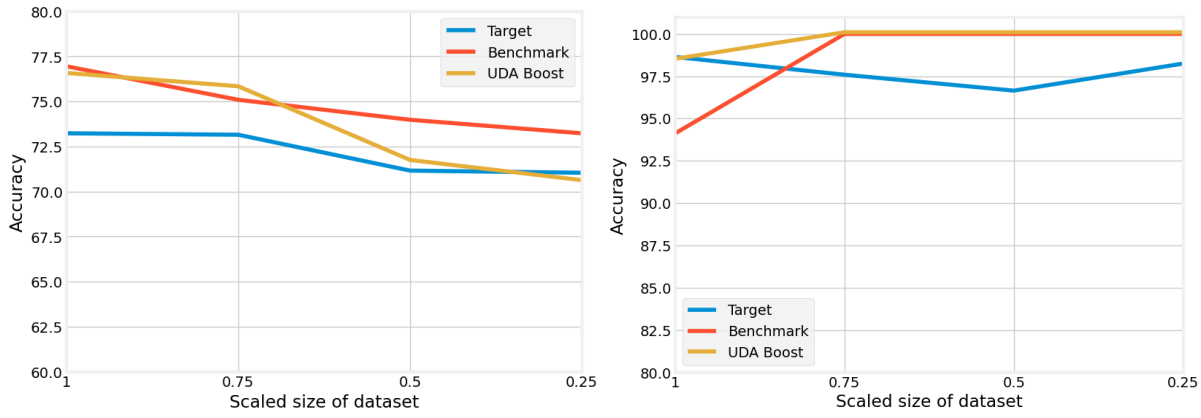
(A) A→D (B) A→W

FIGURE 4.1. Office-31 Amazon: Size of Target Domain vs Testing Accuracy

	A→D				A→W			
	1	**0.75**	**0.5**	**0.25**	**1**	**0.75**	**0.5**	**0.25**
Source	79.92	-	-	-	78.36	-	-	-
Target	93.90	**94.98**	**93.00**	87.61	92.7	94.89	91.70	90.91
Benchmark	94.30	92.56	**93.00**	87.61	90.4	95.23	90.77	91.44
UDA Boost	**95.25**	89.74	92.31	**89.74**	**97.06**	**97.06**	**97.06**	**95.59**

TABLE 4.1. Office-31 Amazon Results

4.1.2 DSLR



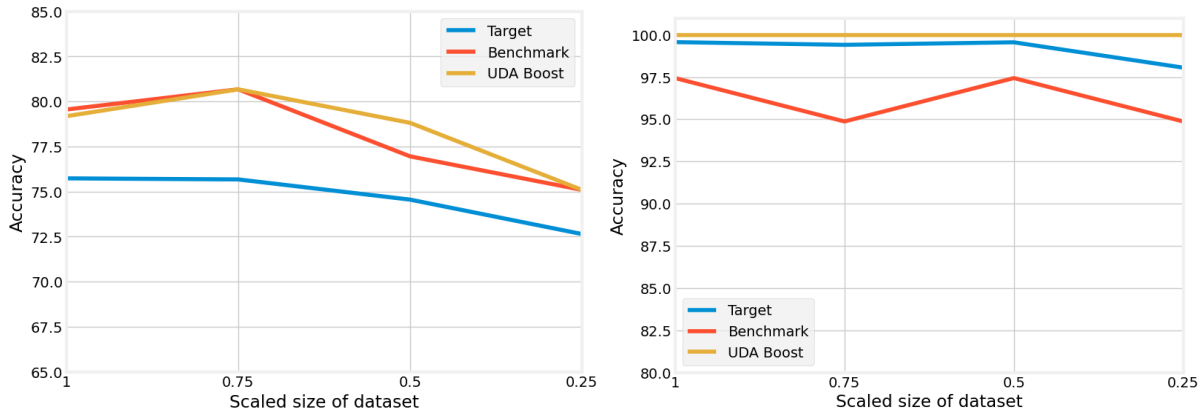
(A) D→A (B) D→W

FIGURE 4.2. Office-31 DSLR: Size of Target Domain vs Testing Accuracy

	D→A				D→W			
	1	**0.75**	**0.5**	**0.25**	**1**	**0.75**	**0.5**	**0.25**
Source	60.1	-	-	-	95.72	-	-	-
Target	73.23	73.15	71.16	71.04	**98.64**	97.59	96.65	98.25
Benchmark	**76.95**	75.09	**73.98**	**73.23**	94.12	**100.00**	**100.00**	**100.00**
UDA Boost	76.58	**75.84**	71.75	70.63	98.53	**100.00**	**100.00**	**100.00**

TABLE 4.2. Office-31 DSLR Results

4.1.3 Webcam



(A) W→A (B) W→D

FIGURE 4.3. Office-31 Webcam: Size of Target Domain vs Testing Accuracy

	W→A				W→D			
	1	**0.75**	**0.5**	**0.25**	**1**	**0.75**	**0.5**	**0.25**
Source	62.8	-	-	-	99.2	-	-	-
Target	75.73	75.67	74.55	72.64	99.57	99.41	99.56	98.06
Benchmark	**79.55**	**80.67**	76.95	**75.09**	97.44	94.87	97.44	94.87
UDA Boost	79.18	**80.67**	**78.81**	**75.09**	**100.00**	**100.00**	**100.00**	**100.00**

TABLE 4.3. Office-31 Webcam Results

4.1.4 Averages and Discussion

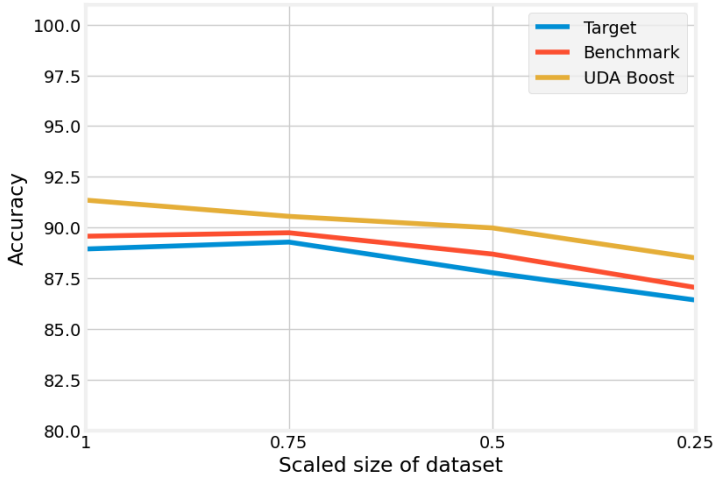


FIGURE 4.4. Office-31: Size of Target Domain vs Testing Accuracy Averages

	Average			
	1	**0.75**	**0.5**	**0.25**
Source	79.35	-	-	-
Target	88.94	89.28	87.77	86.42
Benchmark	89.57	89.74	88.69	87.04
UDA Boost	**91.35**	**90.55**	**89.98**	**88.50**

TABLE 4.4. Office-31 Results Averages

The UDA Boost method outperformed the benchmark on average across all tests on the Office-31 dataset. In most cases, UDA Boost performed roughly the same as both the benchmark and target when the scaled size of the target domain was at $\times 1.0$ (i.e. the full target domain), but ended higher when the scaled size of the target domain was $\times 0.25$. Note that the x-axis of the graphs decreases from left to right. Interestingly, UDA Boost outperformed the benchmark on the full target domain in several cases. This may be because the Office-31 dataset is considered a small dataset so we can see the

merits of UDA Boost method without artificially decreasing the size of the dataset. The UDA Boost result on A→D even outperforms the best result listed in the state-of-the-art benchmarks list from the SHOT/SHOT++ paper, 95.0% versus UDA Boost's 95.25% (Liang et al., 2021). UDA Boost was also able to reach 100% accuracy in some cases, even reach 100% accuracy on all experiments for W→D. However, this only means that the model was able to correctly classify all of the examples in the testing dataset, rather than it being a perfect classifier. We also observed that the benchmark system performed worse than the target in some cases for the $\times 1.0$ size, such as in D→W and W→D. This demonstrates that the benchmark step is not guaranteed to improve performance in the normal case. These particular experiments also had quite high accuracies overall, meaning that these domains are both easy to learn from on their own and highly related to each other, making the adaptation easier. This was expected for the DSLR and Webcam datasets as they consist of images of the same objects in different resolutions (due to having been taken with different cameras). The pre-processing likely also makes these domains more similar as the images are resized to 256×256. Whilst the UDA Boost method outperformed both the Benchmark and Target on average, the rate of decrease in performance as the dataset size decreased was still roughly the same, as shown in figure 4.4. The accuracy of UDA Boost decreased by 2.85% from the $\times 1.0$ dataset to the $\times 0.25$ dataset whereas the Target and Benchmark accuracies decreased by 2.52% and 2.53% respectively.

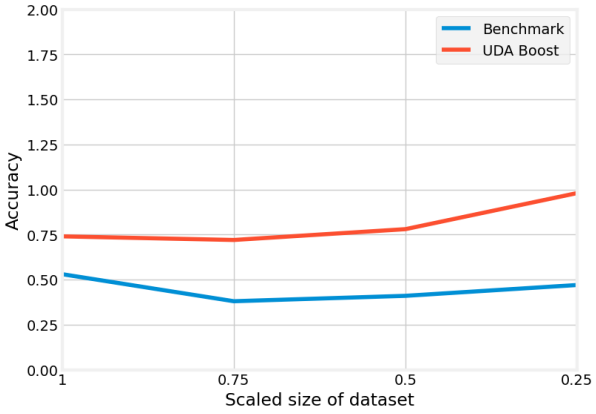


FIGURE 4.5. Office-31: Size of Target Domain vs Average Entropy of Predictions

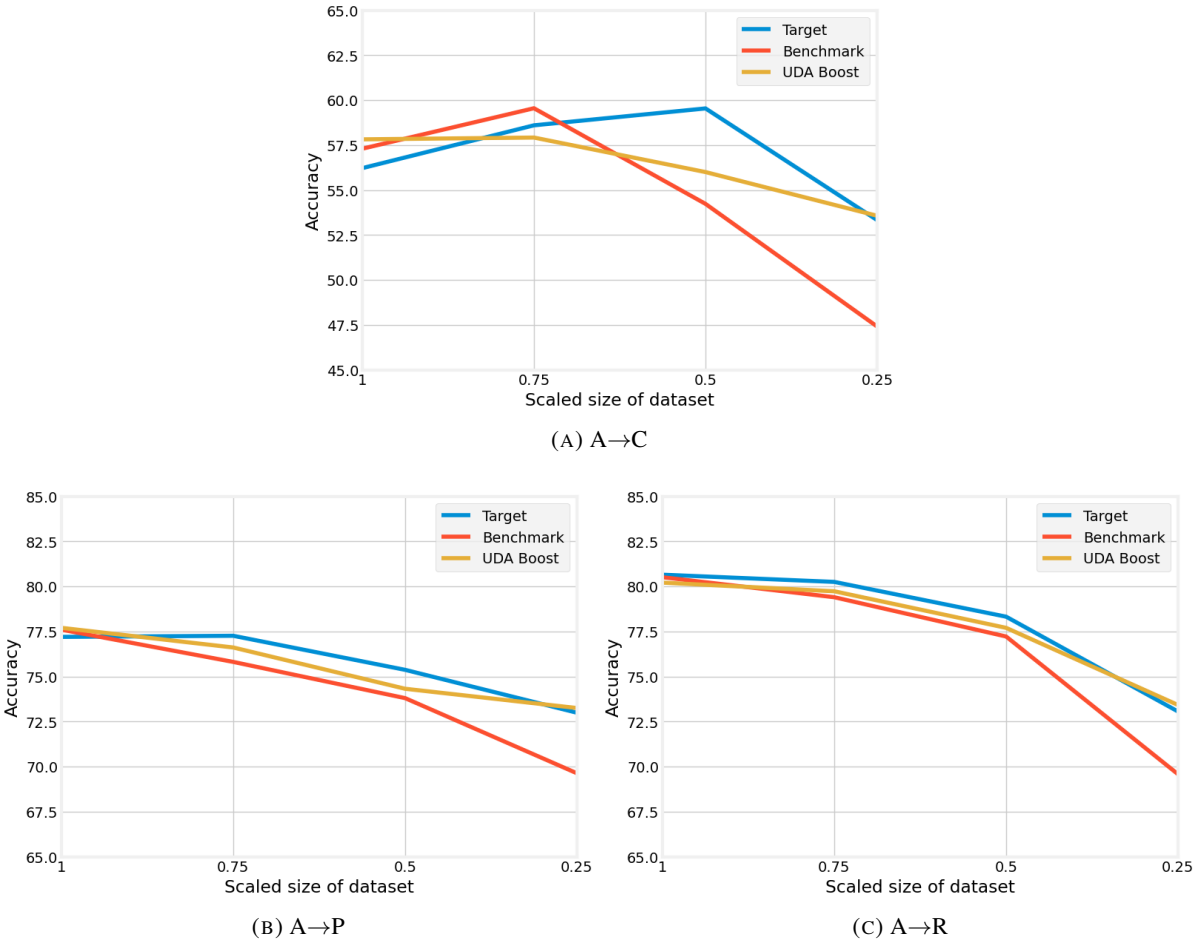
	Average			
	1	**0.75**	**0.5**	**0.25**
Benchmark	**0.53**	**0.38**	**0.41**	**0.47**
UDA Boost	0.74	0.72	0.78	0.98

TABLE 4.5. Office-31 Average Entropy of Predictions

We can see from figure 4.5 that UDA Boost's predictions did have a higher average entropy than the Benchmark's, and also increased at a higher rate as the size of the dataset decreased. The data augmentations likely contributed the most to this increase, however the target adapter and source hypothesis optimiser likely also contributed a small part as well. Despite the higher entropy however, UDA still performed better, which was likely aided by the increased regularisation (via stronger label smoothing and batch normalisation). Figure 4.5 also shows an increase in the slope of UDA Boost's entropy at $\times 0.25$, which suggests that the entropy would start to increase dramatically for lower dataset sizes.

4.2 Office-Home Results

4.2.1 Art



(A) A→C

(B) A→P

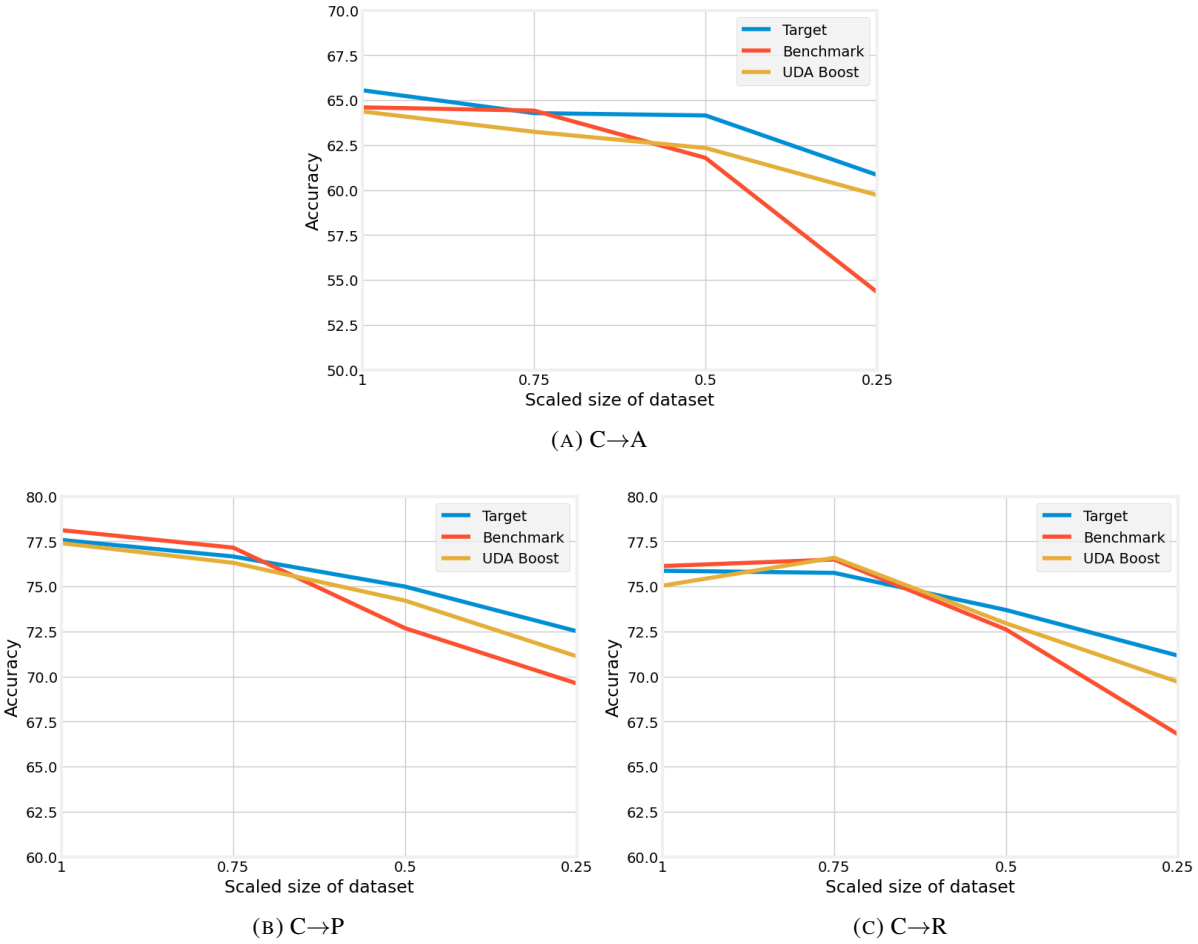
(C) A→R

FIGURE 4.6. Office-Home Art: Size of Target Domain vs Testing Accuracy

	A→C				A→P				A→R			
	1	**0.75**	**0.5**	**0.25**	**1**	**0.75**	**0.5**	**0.25**	**1**	**0.75**	**0.5**	**0.25**
Source	47.56	-	-	-	66.43	-	-	-	74.32	-	-	-
Target	56.22	58.60	**59.54**	53.33	77.20	**77.26**	**75.37**	72.99	**80.65**	**80.25**	**78.32**	73.06
Benchmark	57.30	**59.55**	54.23	47.41	77.60	75.81	73.8	69.63	80.51	79.4	77.22	69.58
UDA Boost	**57.82**	57.92	56.00	**53.57**	**77.70**	76.61	74.32	**73.25**	80.21	79.73	77.7	**73.42**

TABLE 4.6. Office-Home Trained using Art

4.2.2 Clipart



(A) C→A

(B) C→P

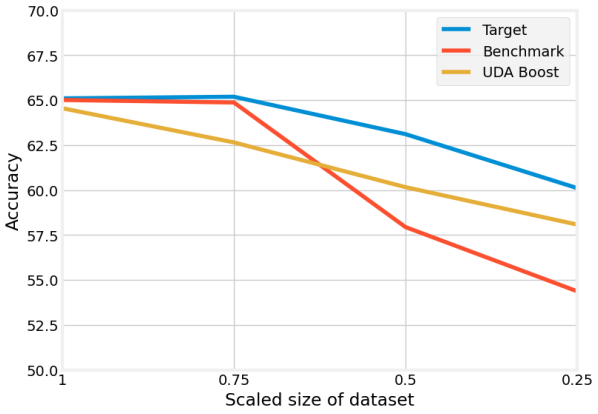
(C) C→R

FIGURE 4.7. Office-Home Clipart: Size of Target Domain vs Testing Accuracy

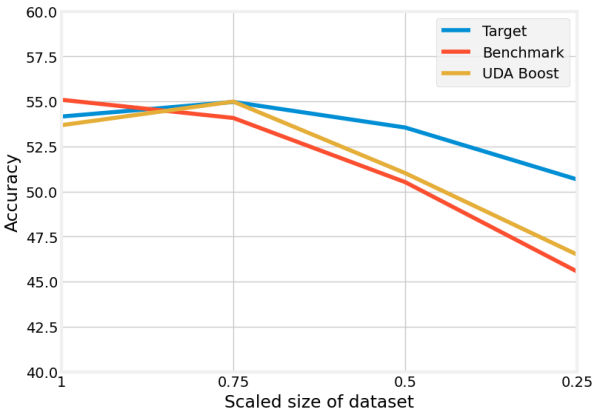
	C→A				C→P				C→R			
	1	**0.75**	**0.5**	**0.25**	**1**	**0.75**	**0.5**	**0.25**	**1**	**0.75**	**0.5**	**0.25**
Source	54.59	-	-	-	63.93	-	-	-	65.60	-	-	-
Target	**65.55**	64.28	**64.15**	**60.84**	77.58	76.66	**74.99**	**72.51**	75.87	75.75	**73.69**	**71.16**
Benchmark	64.60	**64.42**	61.79	54.32	**78.12**	**77.15**	72.68	69.61	**76.13**	76.49	72.61	66.79
UDA Boost	64.37	63.24	62.34	59.72	77.4	76.31	74.21	71.12	75.04	**76.59**	72.95	69.70

TABLE 4.7. Office-Home Trained using Clipart

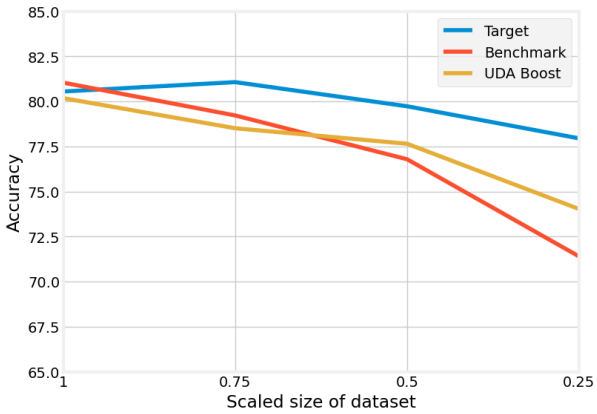
4.2.3 Product



(A) P→A



(B) P→C



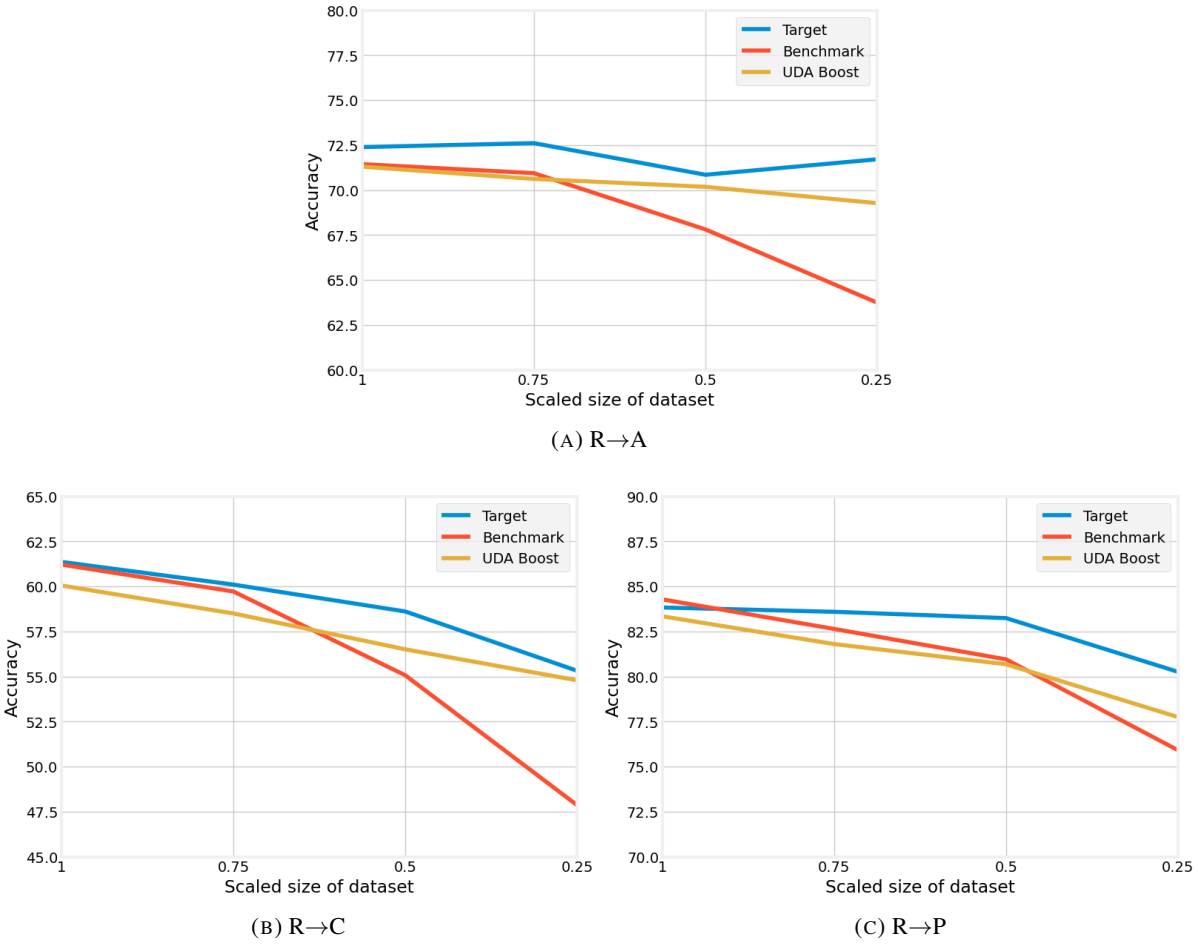
(C) P→R

FIGURE 4.8. Office-Home Clipart: Size of Target Domain vs Testing Accuracy

	P→A				P→C				P→R			
	1	**0.75**	**0.5**	**0.25**	**1**	**0.75**	**0.5**	**0.25**	**1**	**0.75**	**0.5**	**0.25**
Source	53.85	-	-	-	42.47	-	-	-	73.15	-	-	-
Target	**65.10**	**65.19**	**63.11**	**60.11**	54.16	54.97	**53.55**	**50.67**	80.55	**81.07**	**79.73**	**77.95**
Benchmark	65.01	64.87	57.94	54.37	**55.09**	54.08	50.52	45.56	**81.04**	79.22	76.79	71.40
UDA Boost	64.55	62.65	60.16	58.08	53.68	**54.99**	51.02	46.50	80.18	78.51	77.65	74.03

TABLE 4.8. Office-Home Trained using Product

4.2.4 Real World



(A) R→A

(B) R→C

(C) R→P

FIGURE 4.9. Office-Home Real World: Size of Target Domain vs Testing Accuracy

	R→A				R→C				R→P			
	1	**0.75**	**0.5**	**0.25**	**1**	**0.75**	**0.5**	**0.25**	**1**	**0.75**	**0.5**	**0.25**
Source	67.57	-	-	-	47.77	-	-	-	78.64	-	-	-
Target	**72.39**	**72.61**	**70.85**	**71.71**	**61.36**	**60.10**	**58.61**	**55.32**	83.83	**83.59**	**83.24**	**80.26**
Benchmark	71.44	70.94	67.81	63.74	61.21	59.72	55.07	47.86	**84.28**	82.64	80.95	75.91
UDA Boost	71.30	70.62	70.18	69.27	60.05	58.50	56.51	54.79	83.34	81.80	80.68	77.75

TABLE 4.9. Office-Home Trained using Real World

4.2.5 Averages and Discussion

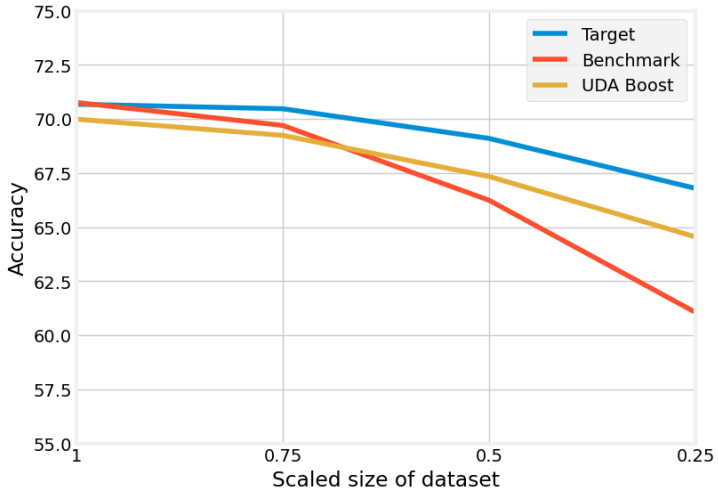


FIGURE 4.10. Office-Home: Size of Target Domain vs Testing Accuracy Averages

	Average			
	1	**0.75**	**0.5**	**0.25**
Source	60.84	-	-	-
Target	70.68	**70.47**	**69.10**	**66.79**
Benchmark	**70.76**	69.70	66.24	61.06
UDA Boost	69.99	69.24	67.34	64.55

TABLE 4.10. Office-Home Results Averages

The majority of the results on the Office-Home dataset show that UDA Boost performed better than the Benchmark, but worse than the Target overall. In all experiments, all three started very close to each other on the $\times 1.0$ dataset, but their performance decreased at different rates as the size of the dataset decreased. In general, the Target method decreased the least, followed by UDA Boost and lastly the Benchmark, which decreased the most. The success of the Target was relatively unexpected,

since it performed worse on the Office-31 dataset, and we had expected that the Benchmark and UDA Boost would be able to improve on these results. On average, UDA Boost's performance decreased by 5.44% from $\times 1.0$ to $\times 0.25$, whilst the Target performance decreased by just 3.89% and the Benchmark performance decreased by 9.7%. The best results for UDA Boost were when Art was used as the Source Domain as it outperformed both the Target and Benchmark for the $\times 0.25$ dataset on all experiments (figure 4.6 and table 4.6. The Art domain was the smallest of the four, at about half the size of each of the others. This suggests that the system is more sensitive to the size of the source domain than previously thought, since we had assumed that the source domain was always going to be sufficiently large enough to create a well-trained starting point for the rest of the system. We can also see that the domain discrepancy between the Art domain and the others is roughly the same as the others between themselves. We can measure this by comparing how each of the source models fare when tested on the same domains, for example when the Art source only model was tested on the Real-World domain, it achieved 74.32% accuracy, whereas when the Clipart domain was tested on the Real-World domain, it only achieved 65.60% accuracy. This shows that the domain discrepancy is not the cause of the different performance on the Art domain, and it is due to the difference in source domain size. We can see that as the target domain size decreases, the performance of UDA Boost decreases at a slightly steeper rate than the Target's, however if we were to test this on even smaller datasets, we may see UDA Boost exhibit some of the same performance as seen with the Office-31 dataset and may improve performance.

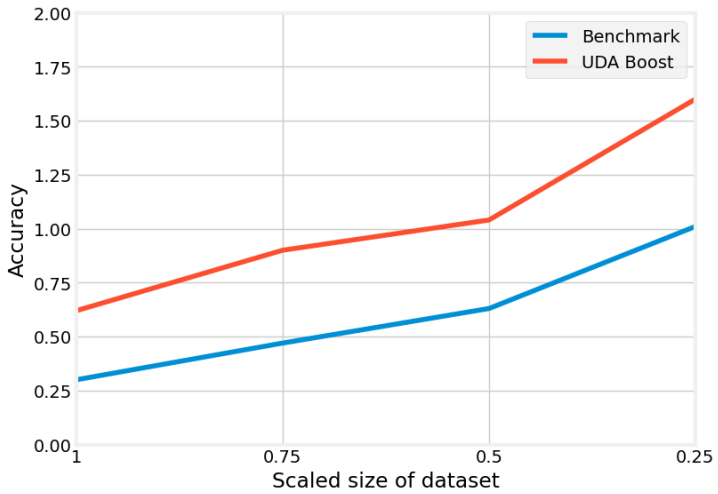


FIGURE 4.11. Office-Home: Size of Target Domain vs Average Entropy of Predictions

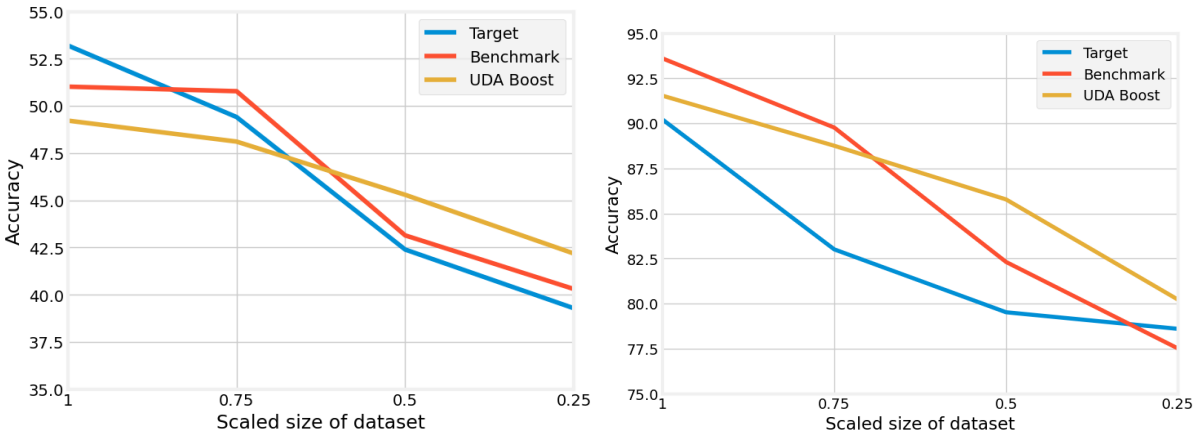
	Average			
	1	**0.75**	**0.5**	**0.25**
Benchmark	**0.30**	**0.47**	**0.63**	**1.01**
UDA Boost	0.62	0.90	1.04	1.60

TABLE 4.11. Office-Home Average Entropy of Predictions

Figure 4.17 demonstrates that the average entropy of the predictions from UDA Boost is higher across all dataset sizes compared to the Benchmark's. However, UDA Boost's entropy is higher than the Benchmark's by a roughly constant amount for all dataset sizes and it follows almost the exact same trend as the Benchmark's. However, since we increase the ratio of data augmentations to original data as the size of the dataset decreases, this suggests that the data augmentations are not the cause of this increase in entropy, which is different to what we have seen in the results on the Office-31 dataset. We can also rule out the MixMatch algorithm as a cause of this discrepancy because the Benchmark system also uses it. The cause is therefore most likely the target adapter or the source hypothesis optimiser or the differences between the Office-31 dataset and Office-Home are causing the model to behave differently.

4.3 Digits Results

4.3.1 MNIST



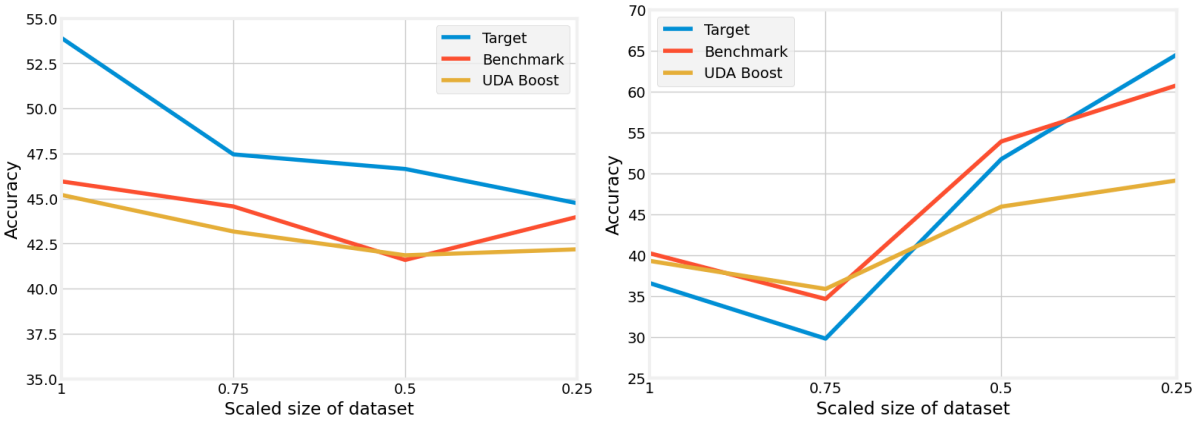
(A) M→S (B) M→U

FIGURE 4.12. Digits MNIST: Size of Target Domain vs Testing Accuracy

	M→S				M→U			
	1	**0.75**	**0.5**	**0.25**	**1**	**0.75**	**0.5**	**0.25**
Source	21.21	-	-	-	93.39	-	-	-
Target	**53.20**	49.41	42.39	39.28	90.22	83.01	79.52	78.60
Benchmark	51.02	**50.78**	43.14	40.30	**93.61**	**89.77**	82.31	77.51
UDA Boost	49.22	48.11	**45.29**	**42.18**	91.54	88.76	**85.78**	**80.22**

TABLE 4.12. Digits Source Trained using MNIST

4.3.2 SVHN



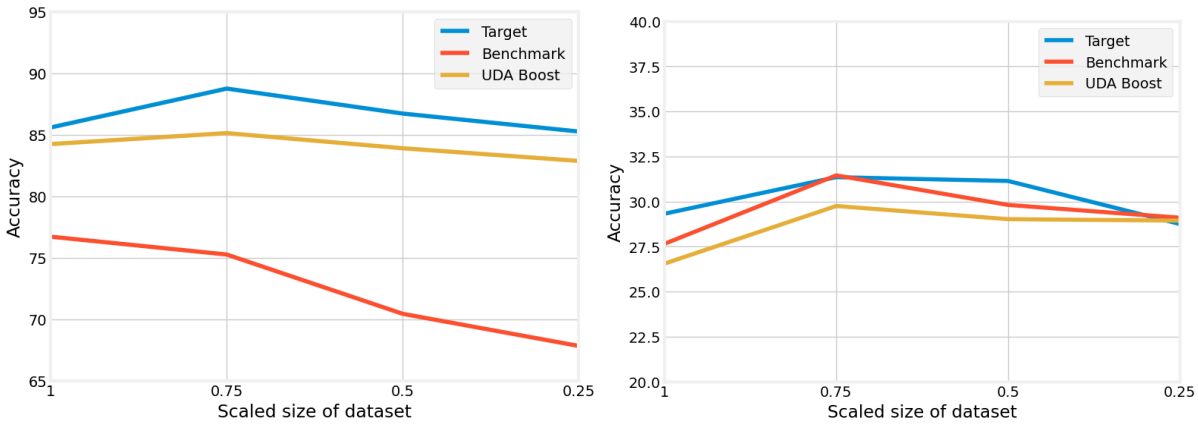
(A) S→M (B) S→U

FIGURE 4.13. Digits SVHN: Size of Target Domain vs Testing Accuracy

	S→M				S→U			
	1	**0.75**	**0.5**	**0.25**	**1**	**0.75**	**0.5**	**0.25**
Source	**78.89**	-	-	-	**54.24**	-	-	-
Target	53.91	**47.45**	**46.64**	**44.74**	36.56	29.78	51.72	**64.52**
Benchmark	45.95	44.56	41.59	43.98	40.21	34.62	**53.88**	60.75
UDA Boost	45.20	43.17	41.85	42.18	39.28	**35.85**	45.91	49.12

TABLE 4.13. Digits Source Trained using SVHN

4.3.3 USPS



(A) U→M (B) U→S

FIGURE 4.14. Digits USPS: Size of Target Domain vs Testing Accuracy

	U→M				U→S			
	1	**0.75**	**0.5**	**0.25**	**1**	**0.75**	**0.5**	**0.25**
Source	66.39	-	-	-	16.6	-	-	-
Target	**85.58**	**88.74**	**86.71**	**85.25**	**29.32**	**31.35**	**31.14**	28.75
Benchmark	76.70	75.25	70.43	67.82	27.65	**31.45**	29.81	**29.10**
UDA Boost	84.23	85.12	83.89	82.86	26.55	29.75	29.02	28.94

TABLE 4.14. Digits Source Trained using USPS

4.3.4 Averages and Discussion

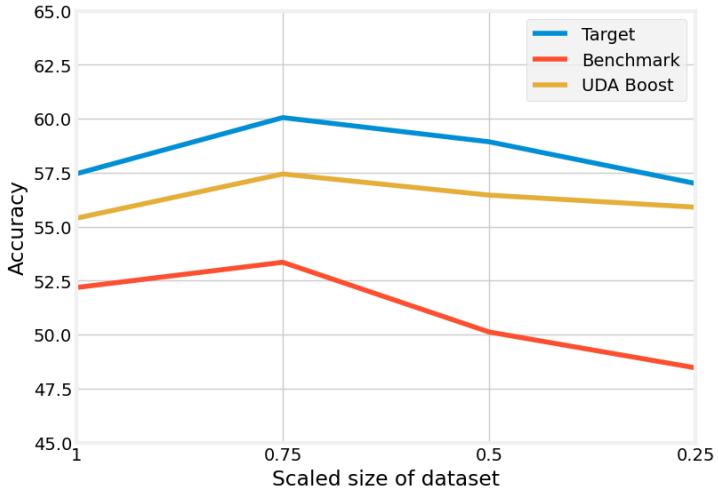


FIGURE 4.15. Digits: Size of Target Domain vs Testing Accuracy Averages

	Average			
	1	**0.75**	**0.5**	**0.25**
Source	41.50	-	-	-
Target	**57.45**	**60.05**	**58.93**	**57.00**
Benchmark	52.18	53.35	50.12	48.46
UDA Boost	55.39	57.44	56.46	55.90

TABLE 4.15. Digits Results Averages

The results of the experiments for the Digits dataset demonstrated similar patterns to the results on the Office-Home in that the Target performed the best overall, followed by UDA Boost and lastly the Benchmark. UDA Boost also demonstrated that it was the least affected by the changing size of the target domain, but still performed worse than the Target on average. We had expected that the system would perform well when using the MNIST and USPS domains since the images from these domains were very similar and the results for M→U and U→M reflect this (tables 4.12 and 4.14). The USPS domain is much smaller than the MNIST domain and when comparing the results of these two experiments we can see that M→U yielded better results than U→M for all steps on the $\times 1.0$ dataset, but U→M ended with better performance on the $\times 0.25$ dataset. This tells us that the performance was better when the sizes of the source and target domains were closer and that the results are more sensitive to changes in the size of the source domain than expected, which is similar to what we have seen in the results for the Office-Home dataset. There was also a very large domain discrepancy between the SVHN domain and the other two, the experiments involving this domain showed much lower performance than other datasets. When the SVHN domain was used as the source domain, the source-only model performed better than any of the others for almost all experiments. Note that even though the results for the source-only model are only listed once in each table, the same results apply for all target domain sizes because the source only model is trained on the full source domain and tested on the testing dataset of the target domain, which does not change sizes.

The average entropy for the Digits results exhibited the same patterns as seen on the other datasets. The entropy was similar to the Benchmark's for the $\times 1.0$ target domain, but increased at a higher rate. The average entropies of both the Benchmark and UDA Boost were also higher overall compared to the previous datasets, likely due to the inconsistent results from the SVHN domain.

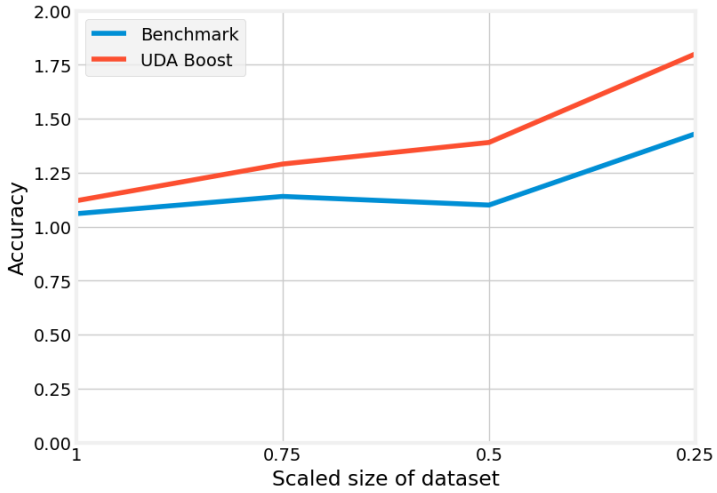


FIGURE 4.16. Digits: Size of Target Domain vs Average Entropy of Predictions

	Average			
	1	**0.75**	**0.5**	**0.25**
Benchmark	**1.06**	**1.14**	**1.10**	**1.43**
UDA Boost	1.12	1.29	1.39	1.8

TABLE 4.16. Digits Average Entropy of Predictions

4.4 Data Augmentations Study

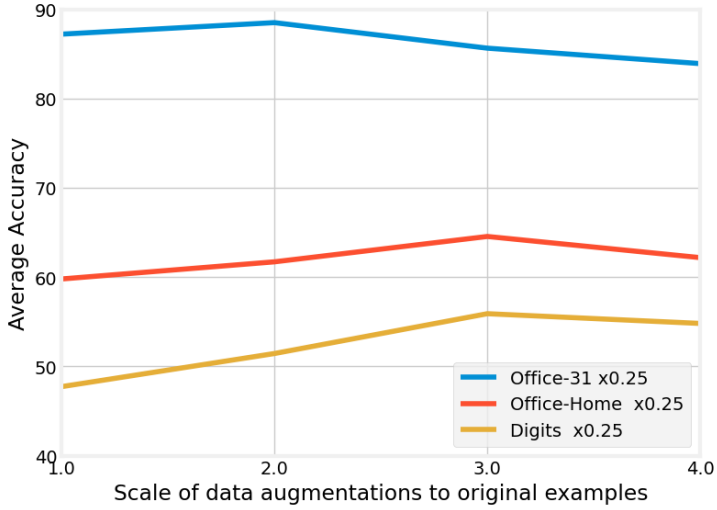


FIGURE 4.17. Ratio of augmented examples to original examples vs Average accuracy for $\times 0.25$ target domains

	Ratio			
	1.0	**2.0**	**3.0**	**4.0**
Office-31×0.25	87.21	**88.50**	85.64	83.91
Office-Home×0.25	59.79	61.71	**64.55**	62.18
Digits×0.25	47.73	51.44	**55.90**	54.81

TABLE 4.17. Average Accuracy on the ×0.25 target domains for varying ratios of generated synthetic data

To decide on the ratio of generated data to original data, we performed an auxiliary study on the relationship between this ratio and the accuracy when the target domain is scaled down by ×0.25. We found that on the Office-31 dataset, the best ratio was '2.0', whist on the two other datasets it was '3.0'. Based on these results we used ratios of 0.5, 1.0, 1.5 and 2.0 for the ×1.0, ×0.75, ×0.5 and ×0.25 target domains respectively on the Office-31 dataset and 0.5, 1.0, 2.0 and 3.0 for the others.

4.5 Evaluation

The UDA Boost method was able to provide a more moderate decline in performance as the size of the target domain decreased and provided substantial performance improvements for the smallest dataset tested (Office-31), but failed to perform better than the Target model on the larger two. The success of UDA Boost on the larger datasets was undermined by the initial overheads of the added methods which harmed the performance and even though UDA Boost was able to slow down the rate of performance degradation as the target domain size decreased, the smallest dataset size tested was not small enough to see an improvement. For many of the more difficult tasks, the Target was able to outperform both UDA Boost and the Benchmark for the scaled-down target domains. We also know that the entropy of the predictions of UDA Boost increased somewhat exponentially as the size was reduced, which indicates that the performance of UDA Boost may have started to get worse at smaller dataset sizes. This is most likely caused by the ratio of original data to augmented data, which increased as the target domain size got smaller, so it is likely that the benefit provided by increasing the ratio slowly diminishes. We also found that the UDA Boost method was able to boost the performance of the ×1.0 target domain in some scenarios.

The performance of all methods was highest on the Office-31 dataset on average, which was the smallest dataset and lowest on the Digits dataset, which was the largest. This demonstrated that the performance of the system depended more on the task itself than the number of examples present. We also observed

that the system was sensitive to the size of the source domain relative to the target domain, even though we ensured that the source domain was as large as it could be. Domains with large domain discrepancies between them, such as the SVHN and USPS domains (from Digits), were observed to have much lower accuracies when adapting between them. All methods experienced low accuracies when adapting between these domains and in some cases the source-only model performed better, meaning that the adaptation step was causing it to perform worse when testing on the target domain.

The main benefit of the UDA Boost method is that as the size of the target domain decreases, its performance decreases at a slower rate compared to the other methods. This was the initial aim of the method but as we have seen, UDA Boost also adds some overhead that decreases the performance by a roughly constant amount, which limits the effectiveness of the method so that it only outperforms the Target and Benchmark methods when the target domain is significantly reduced. This was expected to some degree as we knew that adding the target adapter and data augmentations was going to degrade performance slightly but we did not know if the combination of each of the different techniques used would cause them to benefit each other or limit their capabilities. Overall, the results demonstrated that UDA Boost was able to provide a more reliable and slower decrease in performance as the size of the dataset decreased. UDA Boost was also less susceptible to decreased performance due to higher domain discrepancy in some cases, which has positive implications for the practical use of this method since in real-life scenarios the domain discrepancy between source and target domain is likely to be large.

To improve this study we would perform tests using smaller reduction sizes for the larger two datasets to find the point at which either UDA Boost overtakes the Target and Benchmark or to find the limit of the ratio of data augmentations to find when the addition of more augmented data starts to dramatically degrade performance. To improve the UDA Boost method itself we would test different training parameters to determine the optimal way to train the model, such as by altering the learning rates for the three main sections of the network or using stronger training for the feature extractor. A wider range of experiments could also be conducted, such as using different datasets or even different learning scenarios such as multi-label classification and semi-supervised learning.

CHAPTER 5

Conclusion

As the issues of privacy and the difficult of constructing large labelled datasets grow whilst advances in the capabilities of Deep Learning applications rise, it is increasingly necessary for designs to offer privacy guarantees and more reliable standards for performance. The UDA Boost method provides both of these and was even able to perform exceptionally well on the smallest dataset tested (Office-31), outperforming the benchmark method for the full-sized target domain. By combining pre-existing methods with new ideas, the UDA Boost method was able to provide a slower rate of decline in the performance as the size of the target domain decreased compared to the Benchmark method and performed better on many of the unscaled target domains in the Office-31 dataset than the other systems. The largest limitation of this method is the overhead that is added by the techniques used, which was expected, however for most of the experiments on the two larger datasets, the target domain was not reduced enough to see the point where UDA Boost may have performed better. We also found that for the adaptation cases that were more difficult due to a higher domain discrepancy or differences in domain sizes, the Target method (step 2) outperformed both the Benchmark and UDA Boost. This indicated that it may be better to preserve the source hypothesis in the classifier and solely focus on the feature extractor in cases where there is higher domain discrepancy or an unbalanced proportion of dataset sizes between the source and target domains. Improvements to the UDA Boost method could use this knowledge to focus on adapting the feature extractor such as utilising the source classifier to determine domain invariant information to train the feature extractor. The reliability demonstrated by UDA Boost makes it more useful in situations where there is little knowledge shared between the producer of the source-trained model and the consumer, and the guarantee of privacy of the source domain provided by this method also allows it to be used in a wide range of scenarios. The domain discrepancy and size difference between source and target will still greatly affect the results, however UDA Boost is able to perform more reliably in these conditions when compared to the Benchmark so it is still advantageous to chose this method over alternatives.

5.1 Future Work

The recent rise in popularity of Transformer based architectures such as ViT (Dosovitskiy et al., 2021) has demonstrated that these methods have the potential to overtake traditional CNN designs. The 'Lion' method, introduced by Chen et al. (2023), which achieved the current state-of-the-art performance on the ImageNet dataset, has demonstrated the large potential for Transformers in the field of Computer Vision. Transformer based models have already begun to be used in large scale practical uses in the field of natural language processing, which has been demonstrated by the recent success of ChatGPT using the GPT-3 and more recently, GPT-4 (Brown et al., 2020). However, the need for an extremely large training dataset remains as the most significant barrier in the advancement of Transformers for Computer Vision Tasks. The findings of this paper could be used to try to overcome this problem by adapting these methods to Transformer based architectures.

The novel ideas proposed in the UDA Boost method, mainly the target adapter and source hypothesis optimisation, could also be used to improve the performance of HTL Domain Adaptation methods for normal uses of Domain Adaptation where the target domain is not scaled down. The results on the Office-31 dataset demonstrated that in some cases these methods were able to outperform the benchmark system on the un-scaled down target domain. The focus of this project was not to improve the performance on the whole target domain, however these results showed that the UDA Boost method could be suitable for this task. Future works could build upon the UDA Boost method to improve the performance on larger datasets and increase the reliability of these performance improvements.

Bibliography

Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM.

David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. 2019. Mixmatch: A holistic approach to semi-supervised learning.

Victor Bouvier, Philippe Very, Clément Chastagnol, Myriam Tami, and Céline Hudelot. 2020. Robust domain adaptation: Representations, weights and inductive bias.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V. Le. 2023. Symbolic discovery of optimization algorithms.

Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale.

Brad Dwyer. 2021. Andrew ng: "deploying to production means you're halfway there.". *RoboFlow*.

EU. 2016. General data protection regulation.

Abolfazl Farahani, Sahar Voghoei, Khaled Rasheed, and Hamid R. Arabnia. 2020. A brief review of domain adaptation.

Basura Fernando, Amaury Habrard, Marc Sebban, and Tinne Tuytelaars. 2014. Subspace alignment for domain adaptation.

Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. CCS '15, page 1322–1333. Association for Computing Machinery, New York, NY, USA.

Yaroslav Ganin and Victor Lempitsky. 2015. Unsupervised domain adaptation by backpropagation.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-adversarial training of neural networks.

Spyros Gidaris, Praveer Singh, and Nikos Komodakis. 2018. Unsupervised representation learning by predicting image rotations.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial networks.

Arthur Gretton, Karsten Borgwardt, Malte J. Rasch, Bernhard Scholkopf, and Alexander J. Smola. 2008. A kernel method for the two-sample problem.

Arthur Gretton, Alex Smola, Jiayuan Huang, Marcel Schmittfull, Karsten Borgwardt, and Bernhard Schölkopf. 2009. Covariate shift by kernel mean matching.

Kaiming He and Jian Sun. 2014. Convolutional neural networks at constrained time cost.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition.

Dan Hendrycks, Norman Mu, Ekin D. Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. 2020. Augmix: A simple data processing method to improve robustness and uncertainty.

J.J. Hull. 1994. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554.

IBM. 2020. Ibm maximo visual inspection.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Mahmood Karimian and Hamid Beigy. 2023. Concept drift handling: A domain adaptation perspective. *Expert Systems with Applications*, 224:119946.

Jan Kukačka, Vladimir Golkov, and Daniel Cremers. 2017. Regularization for deep learning: A taxonomy.

S. Kullback and R. A. Leibler. 1951. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86.

Ilja Kuzborskij and Francesco Orabona. 2013. Stability and hypothesis transfer learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 942–950. PMLR, Atlanta, Georgia, USA.

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Ng. 2011. Unsupervised learning of hierarchical representations with convolutional deep belief networks. *Commun. ACM*, 54:95–103.

Jian Liang, Dapeng Hu, Yunbo Wang, Ran He, and Jiashi Feng. 2021. Source data-absent unsupervised domain adaptation through hypothesis transfer and labeling transfer.

Akshay Mehra, Bhavya Kailkhura, Pin-Yu Chen, and Jihun Hamm. 2021. Understanding the limits of unsupervised domain adaptation via data poisoning. *CoRR*, abs/2107.03919.

Shervin Minaee, Rahele Kafieh, Milan Sonka, Shakib Yazdani, and Ghazaleh Jamalipour Soufi. 2020. Deep-covid: Predicting covid-19 from chest x-ray images using deep transfer learning.

Rafael Müller, Simon Kornblith, and Geoffrey Hinton. 2020. When does label smoothing help?

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*.

Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

Reilly, Depa, and Douglass. 2019. Scaling ai: From experimental to exponential.

Raúl Rojas. 1996. *The Backpropagation Algorithm*, pages 149–182. Springer Berlin Heidelberg, Berlin, Heidelberg.

Victor Ruehle, Robert Sim, Sergey Yekhanin, Nishanth Chandran, Melissa Chase, Daniel Jones, Kim Laine, Boris Köpf, Jaime Teevan, Jim Kleewein, and et al. 2021. Privacy preserving machine learning: Maintaining confidentiality and preserving trust.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. 2010. Adapting visual category models to new domains. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 213–226. Springer Berlin Heidelberg, Berlin, Heidelberg.

Tim Salimans and Diederik P. Kingma. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks.

Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*.

Rodrigue Siry, Louis Hémadou, Loïc Simon, and Frédéric Jurie. 2021. On the inductive biases of deep domain adaptation.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway networks.

Baochen Sun, Jiashi Feng, and Kate Saenko. 2016. Correlation alignment for unsupervised domain adaptation.

Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. 2017. Deep hashing network for unsupervised domain adaptation.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?