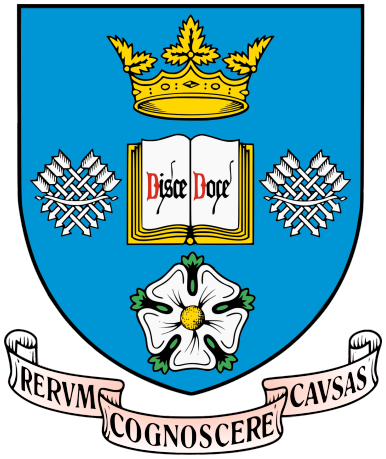
University of Sheffield

Source-Free Unsupervised Domain Adaptation for Image and Graph-Structured Data



Igor Kolasa

Supervisor: Shuo Zhou

A report submitted in fulfilment of the requirements
for the degree of BSc in Computer Science

in the

Department of Computer Science

May 10, 2023

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: Igor Kolasa

Signature: Igor Kolasa

Date: 10/05/2023

Abstract

Unsupervised Domain Adaptation (UDA) aims to adjust a machine learning model trained on data following one probability distribution to unlabeled dataset drawn from different distribution. Source-Free Unsupervised Domain Adaptation (SFUDA) is a related problem setting that additionally assumes that access to the original training set is restricted during the adaptation process.

In recent years, various SFUDA strategies have been developed for computer vision or natural language processing tasks. However, only a few techniques have been proposed for graph-structured data.

To address this issue, the project first analyses the state-of-the-art image classification SFUDA technique (Attracting and Dispersing), and next applies it to graph classification SFUDA task. Several executed experiments show improvement of the graph classification performance when the method is used.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, machine learning has become a central point of today's technology [57], contributing to the rapid development in disciplines like natural language processing [30], computer vision [64] or bioinformatics [77]. Nowadays, most of the state-of-the-art solutions to the discipline specific tasks are based on the Deep Neural Networks (DNNs) [35]. However, their good performance depends on the large amount of labeled data for training. This requirement can often be not satisfied due to the difficulties in data collection or laborious and expensive labeling process [14]. Moreover, data on which DNN model is trained should accurately reflect characteristics of data the model will face after deployment. In response, significant research effort has been devoted to the *Transfer Learning* [47] strategies that allow DNNs to transfer knowledge learnt from one data set to different related collections of data. Often, these methods are designed for and tested on one specific type of data, e.g. text or images, while only a few techniques have been proposed for other types, e.g. graph data. Therefore, an interesting direction for further research is to test transferability of some of the methods between different disciplines. We will take a closer look at tasks of *Image Classification* [52], *Graph Classification* [29], and one specific subcategory of transfer learning called *Domain Adaptation* [12].

1.1 Image Classification

The objective of the image classification is to develop software able to identify objects in images (e.g. people, animals, diseases). The term *classification* refers to the fact that we want to assign an observed image to one of the pre-defined *classes* (e.g. cat or dog, healthy or unhealthy, car or bike or train). In the classical setting of the task, we are given a collection of labeled images called a *training* data set. It is used to train a machine learning model such that after the training the model is able to associate patterns observed in the images with the corresponding classes. The performance of the model is then evaluated on a separate, unseen collection of images called *test* data set. Each test image is input into the model, which classifies it based on the previously learnt pattern-class relation. In general, the model is assumed to perform well, if the majority of the predicted class labels is the

same as the ground true classes of the test samples. However, due to some possible data set biases, such as class imbalance [28], there exist different evaluation metrics [23], which can be more appropriate for specific tasks. In the last decade, the Convolutional Neural Networks (CNNs) [1] proved their superiority over other image classification methods and achieve the state-of-the-art results on the most challenging computer vision benchmarks [55]. Interestingly, CNNs operating on image data share some architectural design principles with Graph Neural Networks (GNNs) [70], which are designed specifically for problems where data are represented as graphs.

1.2 Graph Classification

Instead of images, the input data in the graph classification task are given as graphs, which are a suitable representation for the structures like molecules [71], social networks [7], or source code [2]. Following the same training principles as introduced before, a machine learning model is optimised to associate a given graph with one of the pre-defined classes. For instance, we may want to predict if a given molecule will inhibit HIV virus replication or not, which can be a vital question in drug research. Another example can be to assign protein structure of species to a taxonomic group of its origin (e.g. mammals, bacterial families, archaeans), which could give us insights into processes of biological evolution. GNN architectures like Graph Convolutional Networks (GCNs) [32] or Graph Isomorphism Networks (GINs) [73] have been established as baseline solutions to the graph classification benchmark challenges [24]. However, similarly to CNNs and all the other models that derive from DNN design principles, they require abundant labeled data and are sensitive to discrepancy between training set and test set. As will be discussed later in the report, the transfer learning papers distinguish various causes for the difference between data sets. Our particular focus will be put on the problems caused by the occurrence of the *Domain Distribution Shift*.

1.3 Domain Distribution Shift

When discussing the related problems, a training data set is usually referred to as a *source domain*, and a term *target domain* is used instead of a test data set. Domain distribution shift can be then described as a setting where source and target domains have different data probability distributions [51]. Without diving into mathematical details for now, the distribution shift can be easily observed in image data. From Fig.1.1, it can be clearly seen that although both domains contain the same objects, their representations are visually different. This kind of discrepancy is not an impediment to human's eye, but it can significantly deteriorate a machine learning model's classification performance [49, 59] as patterns observed during training may no longer appear in the target domain. Distribution shifts occur in all types of data and are caused by various factors. For example, graph representations of proteins extracted from one animal species will often have different characteristics than proteins from other species, while both animals can belong to the same

taxonomy group. In many real-world scenarios it happens that we have access to abundant training data from one domain, but we want to secure good performance in a related domain which lacks labels. Imagine a company building a road signs classifier for self-driving cars. They train their deep learning model with carefully labelled pictures taken on the streets of New York. As a company grows, it must adjust their model to work reliably in different cities or even countries. While the collection of new images could be theoretically done through online sources such as Google Maps, the labelling of data for every new location would be an extremely laborious and infeasible process.

1.4 Domain Adaptation Proposition

Domain Adaptation (DA) is a specific setting of the transfer learning task to address the type of issues described above. In general, DA aims to train a machine learning model robust to the discrepancies between related source and target domains, under assumption of insufficient amount or complete lack of labels in the latter. This project will explore the *Unsupervised Domain Adaptation* (UDA) [38], which is a setting where all the target labels are missing. It should not be confused with the more challenging task of *Domain Generalisation* (DG) [81] , where we are usually given more than one source domain and we seek to train a model that achieves a minimum prediction error on an unseen target domain (i.e., target data can not be accessed in training). Classical UDA methods require access to the both source and target data in order to align disparity between domains. However, in many real-world scenarios access to the source data can not be guaranteed. For example, if a data set contains confidential information which serves as an identifier for an individual, accessing that data could violate privacy policies [68, 44]. In order to tackle this problem setting, *Source-Free Unsupervised Domain Adaptation* (SFUDA) [11] has been established as a new direction of research. SFUDA seeks to adapt a pre-trained source model to an unlabeled target data set without access to the source data.

1.5 Project Overview

While numerous UDA and SFUDA methods have been designed for computer vision tasks, only a few works have attempted to tackle the same problem setting in the context of graph-structured data [8, 41]. Considering the architectural similarities between the DNN models used to classify images and graph-structured data, it could be investigated if techniques proposed for computer vision problems can be generalised and effectively applied to Graph Neural Networks. The main objective of this project is to analyse an existing state-of-the-art image classification SFUDA method, and test its applicability to the graph classification tasks. The work will follow the *Attracting and Dispersing* (AaD) [76] SFUDA approach from the latest Neural Information Processing Systems (NeurIPS) 2022 conference paper, and it will start by reproducing the paper's experiments on the University of Sheffield High Performance Computing (HPC) machines. Next, the AaD technique will

be integrated with some of the existing graph classification models to run experiments on selected graph benchmark datasets in a specially designed setting imitating the SFUDA task. The remaining chapters of the report contain:

- **Chapter 2**: Summary of the standard image and graph classification methods. Both tasks will be discussed in greater detail, together with explanation of the key concepts behind their dedicated DNN architectures, and introduction of common classification performance evaluation metrics.
- **Chapter 3**: Formal definition of the domain distribution shift problem and introduction of different variants of the domain adaptation task, followed by the review of the existing UDA and SFUDA methods.
- **Chapter 4**: Detailed description of the project objectives, planning, and introduction of the visual and graph datasets on which the selected SFUDA method will be evaluated.
- **Chapter 5**: Description of the experimental environment, including the most important code implementation details and preprocessing of the datasets. It will also contain details of the executed experiments, such as a choice of model architectures or hyperparameters setting.
- **Chapter 6**: Presentation of all the results generated during experiments, together with discussion of the main findings and goals achieved by the project. It will also indicate potential directions for further research.
- **Chapter 7**: Key conclusions drawn from the project.

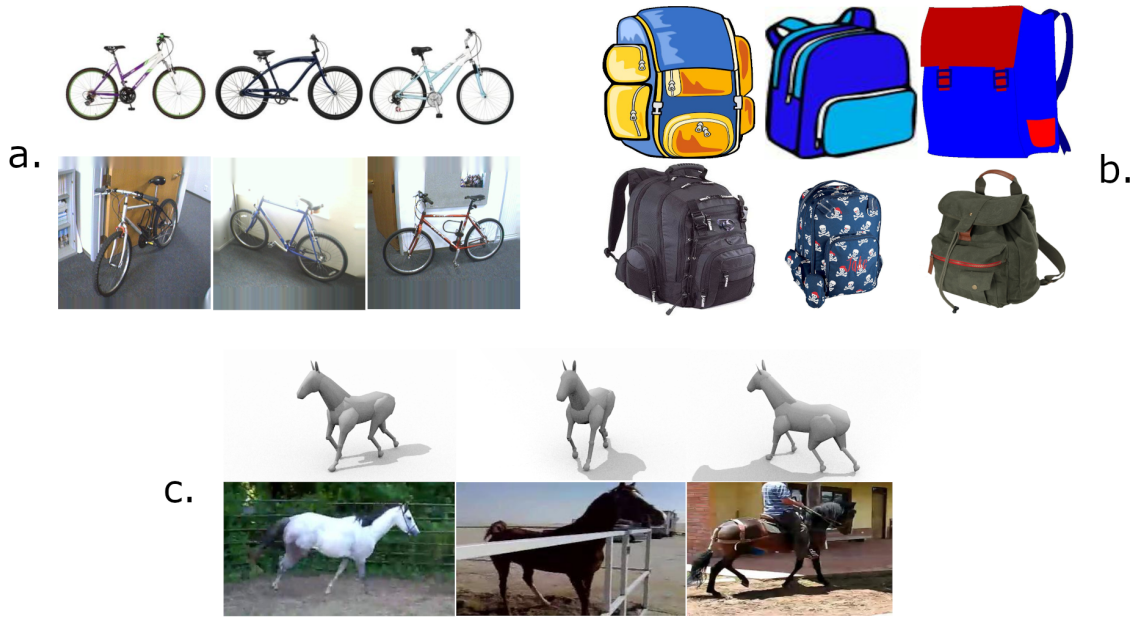


Figure 1.1: Examples of visual domain shifts. **a**: Bike product pictures extracted from Amazon vs low-quality webcam bike images (Office31 [56]); **b**: Clipart images of a backpack vs real backpacks (Office-Home [63]); **c**: Synthesized horse images vs real horses (VisDA-C [48])

Chapter 2

Image and Graph Classification Overview

Before discussing challenges and achievements in the domain adaptation field, we should first develop a deeper understanding of the classical machine learning classification methods. The next pages will describe key characteristics of the image and graph data types, and the DNN architectures that are designed to operate on them. Next, we will see evaluation metrics used to assess the predictive performance of the models.

2.1 Image Classification

2.1.1 Image Data

The simplest representation of an image is in the form of a 2D matrix, where each entry represents a pixel's light intensity value ranging from 0 to 255. It is called a grayscale image as it consists of shades of grey only, with 0 corresponding to black and 255 being white. In order to carry colour information, the RGB model is used, where an image is stored as a 3D matrix and each pixel is represented by a list of red, green, and blue light intensity values (see Fig.2.1). These values are usually termed as colour channels of an image. There also exist representations using different colour models such as HSV [27], or models including additional information, for example RGB-D [21] which contains an additional image depth channel. Before loading images into a DNN model, usually some standard preprocessing is required for faster and better learning. Common practice is to scale down the pixel values range from [0, 255] to [0.0, 1.0] and apply normalisation to each colour channel (i.e., transform the pixel values such that their mean becomes 0.0 and standard deviation equals 1.0) [58].

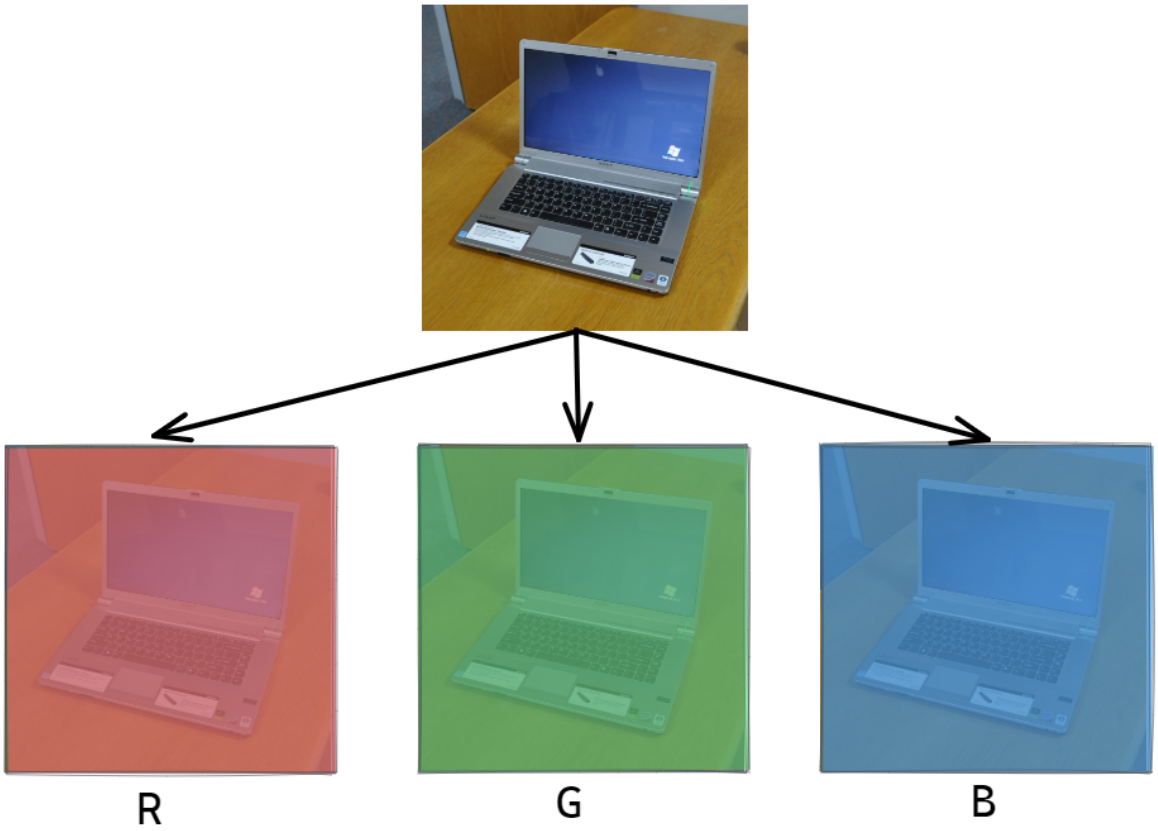


Figure 2.1: A sample image of a laptop from Office-31 dataset decomposed into Red, Green, Blue (RGB) colour channels. Pixel values in each channel represent the corresponding light colour intensity.

2.1.2 Convolutional Neural Networks

Currently, the best image classification solutions are based on the Convolutional Neural Network design principles. The conventional CNN architecture can be decomposed into four types of building blocks (see Fig.2.2). An *input layer* is simply where a preprocessed image matrix is fed into the model. A *convolutional layer* contains a number of filters, also called kernels, which are matrices of learnable parameters - the model's weights [60]. Kernels are applied to the input matrix in the operation called convolution [10], which outputs a new, higher-level matrix representation of the input. For example, features such as an object's edges can be more clearly defined in the resulting matrix. The matrix is next passed through the *pooling layer*, which reduces its size and extracts the most important image features. Output of the pooling layer is then given as an input to the subsequent convolutional layer and the same operations are repeated. Finally, output of the last pooling layer is flattened into a column *feature vector*, the final representation of the input image, and fed into the *multilayer perceptron* (MLP) [46]. MLP is usually a simple neural network with

its own learnable parameters, which task is to predict the class membership of the feature vector. The *softmax function* [18] is applied to the last layer of the MLP, which outputs the probability distribution of all possible classes for the given feature vector. The image is then classified as belonging to the class for which the highest probability was returned. The part of the CNN containing the convolution and pooling layers is usually termed as a *feature extractor*, because its task is to learn the most informative, condensed feature representation of the input image. Adequately, the MLP is called a *classifier*.

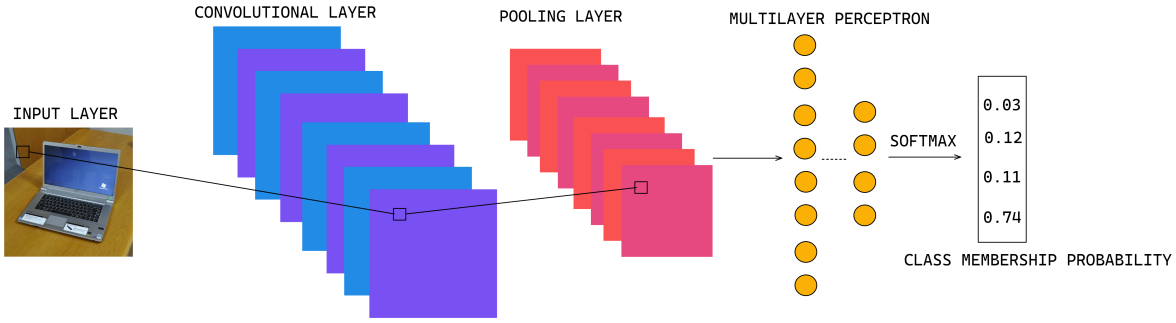


Figure 2.2: Convolutional Neural Network architecture consisting of the input layer, single convolutional and pooling layer, and multilayer perceptron. The final layer of the perceptron is processed by the softmax function that outputs the image class membership probability distribution.

In order to train a CNN, we first need to choose a *loss function* [66], which measures the difference (error) between model's predictions and ground truth labels of training data samples. *Cross Entropy Loss* [9] is an example of such a function commonly used in classification tasks. It measures the dissimilarity between the softmax output from a model and a training sample's label. The label is represented as the probability distribution with value 1.0 for the sample's true class and 0.0 for all the other classes, allowing comparison with the softmax distribution. The CNN training objective is to minimise the prediction error by tuning the weights of the classifier and the feature extractor. This is achieved in the three-step process. First, in the stage called *forward propagation*, a training image is passed through all the model layers and the corresponding loss value is calculated. Next, the *backpropagation* [72] algorithm is applied to calculate gradients of the loss with respect to the model's weights. Finally, one of the optimization algorithms such as Stochastic Gradient Descent [3] or Adam [31] is applied to update the weights using the calculated gradients, and causing the loss to decrease. We define a parameter called *learning rate* that is used to control magnitude of the update. Usually, for the performance purposes, multiple images are input simultaneously into the model as a subset of data samples called *batch*. The training time is measured in the number of *epochs*, where one epoch is a single pass of all the data samples through a model.

2.2 Graph Classification

2.2.1 Graph Data

Graphs are powerful data structures that can be used to represent interrelated data such as social networks, rail networks or chemical molecules. Typical graph consists of a set of *nodes* connected by lines called *edges*. Each node and edge is associated with a list of *attributes*, which describe their properties. For example, in the rail network graph, a train station name could be one of the node attributes, and distance between two connected stations could be an edge's attribute. Before training, each graph must be preprocessed to a format that will be suitable as an input to the selected Graph Neural Network. Usually, it is converted to a data object consisting of:

- *Node feature matrix* with shape [num nodes, num node attributes], where each row stores attribute values of a single node,
- *Edge index* with shape [2, num edges], where each column stores a pair of indexes of connected nodes, and
- *Edge feature matrix* with shape [num edges, num edge features], where each row stores attribute values of a single edge.

2.2.2 Graph Neural Networks

Common GNN architectures are built of the multiple blocks called *message passing layers*, which consist of the *aggregate* and *combine* functions. Purpose of the message passing operations is to return a new representation of each node that somehow integrates information about other nodes in the graph and connections between them (see Fig.2.3). First, feature vectors of a node and its direct neighbours are multiplied by a learnable weights matrix, which results in new representations of nodes, also called their *embeddings*. The *embedding dimensionality* of a node can be different from the original feature vector's size, and is one of the adjustable attributes of a GNN model. The embeddings of neighbours are then aggregated in a specified way, and finally they are combined together with the parent node. In the subsequent message passing layers, the node is gradually updated with the information coming from the more distant nodes. This general procedure stays the same for all the variants of GNN, and usually the only way they are different is how they perform the aggregate and combine functions. Some architectures can also replace a single weight matrix with MLP to learn the node embeddings. There also exist different approaches to include the edge features information. In some cases they are updated separately, while other solutions include them in the update of the node features [78, 17]. After going through all the message passing layers, we obtain a new node feature matrix with updated representations. In order to perform a graph level classification, we need to combine representations of all nodes into a single feature vector representing the whole graph. This can be achieved with a *graph pooling* operation, such as summing or taking an average of node representations.

Therefore, the message passing layers construct the feature extractor module of a GNN, and analogously to the CNN architecture, the vectorised graph embedding is input to the MLP classifier to estimate its class membership probability distribution.

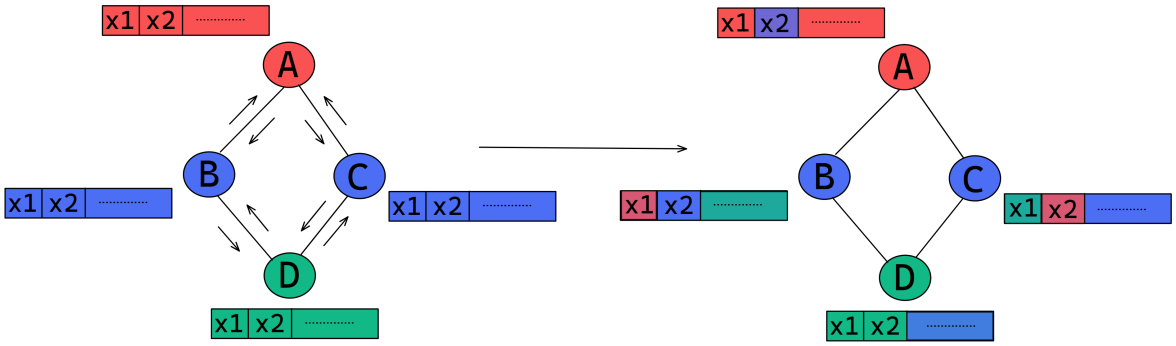


Figure 2.3: Simplified representation of a single message passing layer. Information from each node is sent to its direct neighbours, which results in the updated feature vectors combining information from the original node and the neighbours.

The training process involves identical steps as the ones described for CNN. The same loss functions can be used with both types of models, and their parameters can be updated using the same optimization algorithms. The key difference is that in CNN the kernel weights of the convolutional layer are updated, while in GNN learning is applied to the weight matrices in the message passing layers.

2.3 Evaluation Metrics

Choice of the appropriate evaluation metric is crucial to get unbiased insights into the classifier's performance. We will discuss the most commonly used metrics from the perspective of the binary classification task, i.e., setting where a data sample can be classified as *positive* (being a member of the predefined class) or *negative* (not a member of the class). However, the same concepts can be generalised to the multi-class problems. The metrics we are about to discuss are based on the four base values:

- *True Positive* (TP) predictions - a number of positive samples correctly classified as positives.
- *False Positive* (FP) predictions - a number of negative samples incorrectly classified as positives.
- *True Negative* (TN) predictions - a number of negative samples correctly classified as negatives.
- *False Negative* (FN) predictions - a number of positive samples incorrectly classified as negatives.

The *accuracy* score is often the first metric of choice, and it describes a number of correct predictions over all predictions, i.e., (TP + TN) / (TP + FP + TN + FN). It generally gives a good performance estimation if we operate on a balanced data, i.e., similar amount of positive and negative samples, and if correct classification of both classes is equally important for us. The *precision* measures what proportion of samples classified as positives really belongs to the positive class, i.e., (TP) / (TP + FP). High precision means that a model rarely misclassifies the negative samples, which may be important if we expect a high confidence of positive predictions. On the other hand, the *recall* score is used to measure how many of all the positive cases were correctly identified by the model, i.e., (TP) / (TP + FN). High recall may be crucial in tasks such as the brain cancer identification, where we want to detect all the positive cases, even by the cost of misclassifying some of the healthy brains. *F1-score* combines both precision and recall into one metric defined as their harmonic mean, i.e., 2 * (precision * recall) / (precision + recall). It is good if we want to have a balance between the two metrics, as it is sensitive to one of the inputs having significantly lower value. *Area Under the ROC Curve* (AUC-ROC) is a metric that explicitly uses the concept of the classification *threshold*. The threshold is an arbitrary selected probability value, which is sufficient to assign a given sample to one of the classes. For example, if we set the threshold to 0.5, then all the instances for which a model predicts the probability of positive class membership greater than 0.5 will be classified as positive. ROC curve is a plot of the *True Positive Rate* (TPR) over the *False Positive Rate* (FPR) at all possible classification thresholds (from 0.0 to 1.0). TPR is a synonym for recall while FPR is described as the ratio between the number of incorrectly classified negative samples and the total number of negative instances, i.e., (FP) / (FP + TN). In general, AUC can be thought of as a measure of separability or a summary of how good at distinguishing between classes a model is for different thresholds. AUC value ranges from 0.0, meaning a model classifies all true negatives as positives and vice versa, to 1.0 for models that can distinguish perfectly between both classes. A value in between, for instance AUC = 0.8, means that the model is able to distinguish between positive and negative classes with 80% probability [23].

2.4 Summary

In this chapter we became familiar with the data types that will be used in the project. The RGB model was introduced, which is used to represent images as matrices of numbers, and we saw how to describe a graph with a corresponding node feature matrix, edge index, and edge feature matrix. It should be also clear how the classification of these data objects can be achieved with the common Deep Neural Network architectures. In particular, it should be understood that both CNN and GNN models consist of two core modules: a feature extractor responsible for transforming the input image or graph into the feature vector representation, and a classifier dedicated to predict a correct label for a given data sample. In addition, we are now equipped with tools to evaluate and compare models' performance. The remaining part of the report will use the introduced terminology, and will build on top of the concepts

introduced in this chapter.

Chapter 3

Domain Adaptation Overview

Domain Adaptation is a broad term and there often exists some level of an ambiguity in how different research publications define its tasks and variants. The purpose of this chapter is to provide a formal definition of the problem setting tackled in this project, and separate it from different, but related challenges. It will also explain causes of the data distribution shifts. Further, it will review concrete examples of the existing Unsupervised Domain Adaptation and Source-Free Unsupervised Domain Adaptation methods, followed by the detailed description of the Attracting and Dispersing technique.

3.1 Problem of Different Domains

In the introduction we used the term *domain* as a synonym of a dataset. However, throughout the rest of this report we will refer to the more precise definition from [12], which describes domain D as consisting of the three main elements: *input feature space* $\mathcal{X}$, *output label space* $\mathcal{Y}$, and associated joint probability distribution $P(X, Y)$ over the feature-label space $\mathcal{X} \times \mathcal{Y}$, i.e., $D = \{\mathcal{X}, \mathcal{Y}, P(X, Y)\}$. The input feature space $\mathcal{X}$ defines the dimensionality and the set of values from which the set of data samples $X = \{x_1, ..., x_n\}$ is drawn, i.e., $X \in \mathcal{X}$. For example, in the context of computer vision, a data set of equally sized (100 pixels each) grayscale images would be said to be drawn from the 100-dimensional feature space of values ranging from 0 to 255. Similarly, the set of labels $Y = \{y_1, ..., y_n\}$ is drawn from the output label space $\mathcal{Y}$, which is either a space of $\{0, 1\}$, when considering binary classification, or $\{0...C\}$ if tackling a multi-class classification scenario with C classes, i.e., $Y \in \mathcal{Y}$. The joint probability distribution can be described as $P(X, Y) = P(X)P(X|Y)$ or $P(X, Y) = P(Y)P(X|Y)$, where $P(.)$ is a marginal probability distribution of samples or labels and $P(.|.)$ is a conditional probability distribution. Traditional machine learning assumes that the source domain $D_s = \{\mathcal{X}_s, \mathcal{Y}_s, P(X_s, Y_s)\}$ and target domain $D_t = \{\mathcal{X}_t, \mathcal{Y}_t, P(X_t, Y_t)\}$ are equal, i.e., $D_s = D_t$. Based on the introduced definitions, there are three general situations when this assumption is violated:

1. Different input feature spaces, i.e., $\mathcal{X}_s \neq \mathcal{X}_t$,

2. Different output label spaces, i.e., $\mathcal{Y}_s \neq \mathcal{Y}_t$,
3. Different joint probability distributions, i.e., $P(X_s, Y_s) \neq P(X_t, Y_t)$.

3.2 Domain Adaptation Setting

At this point, we have enough information to specify the Domain Adaptation problem setting that is of interest of this project. First of all, we will assume that source and target datasets are drawn from the same input feature spaces, which is termed as the *homogeneous* DA setting, and the opposite is called *heterogeneous* DA. Similarly, it is assumed that both domains share the same output label spaces, which is a *closed-set* DA scenario. *Open-set* DA is other setting where the domains share some classes and also they may have private classes, i.e., $(\mathcal{Y}_s \not\subset \mathcal{Y}_t) \wedge (\mathcal{Y}_t \not\subset \mathcal{Y}_s) \wedge (\mathcal{Y}_s \cap \mathcal{Y}_t \neq \emptyset)$, and in *partial-set* DA the target label space is considered to be a subset of the source label space, i.e., $(\mathcal{Y}_t \subset \mathcal{Y}_s) \wedge (\mathcal{Y}_s \neq \mathcal{Y}_t)$. Finally, DA literature distinguishes three types of the *probability distribution shift* [51, 40]:

- **Prior shift** is defined as the case where $P(Y_s) \neq P(Y_t)$, i.e., marginal probability distributions of the label sets Y_s and Y_t are different.
- **Covariate shift** is defined as the case where $P(X_s) \neq P(X_t)$, i.e., marginal probability distributions of the input feature sets X_s and X_t are different. (Fig.3.1)
- **Concept shift** is defined as the case where $P(Y_s|X_s) \neq P(Y_t|X_t)$ and $P(X_s) = P(X_t)$. It is the situation when the functional relationship between the model inputs and outputs changes. The cause of the relationship change is some kind of external event or process, not related to the process of data collection.

The focus of the conventional DA techniques reviewed in this project is put on the covariate shift problem. This is a commonly faced issue in computer vision tasks, where the discrepancy in the input features distributions can be caused by factors such as a change of the object background colour or more subtle changes like different lighting conditions or contrast levels. In the case of the graph property prediction tasks, the origins of the covariate shift are specific to the nature of data that graphs represent. For example, they can be a result of structural differences between molecules or different densities of transportation networks.

Finally, DA can be divided into further categories based on the availability of labels in the target domain [65]:

- **Supervised Domain Adaptation**: We have fully labelled data in both source and target domains. However, the amount of samples in the target domain is usually small and not sufficient for training.
- **Semi-supervised Domain Adaptation**: We have fully labelled data in the source domain and only a small subset of samples in the target domain has labels.

- **Unsupervised Domain Adaptation**: We have fully labelled data in the source domain, but labels are not available in the target domain. **Source Free Unsupervised Domain Adaptation** is the special case of UDA task, where only a pre-trained source model and unlabelled target domain samples are available, i.e., access to the source data samples is restricted.

In addition to the criteria discussed above, we need to highlight one more essential requirement. It is assumed that the source and target domains are directly related, which allows for the successful transfer of knowledge between them. [65] describes this setting as a one-step DA. For example, it should be possible to directly adapt a model trained on images of horses to a dataset containing pictures of zebras. However, it may be extremely hard or even impossible to adapt the same model to an astrophotography dataset, as pictures of planets and stars do not share any features with animals. This requirement is described more formally in [53], which states that one of the common assumptions in domain adaptation is the *similarity of the marginal distributions*. In other words, the assumption is made that the covariate shift is not extreme.

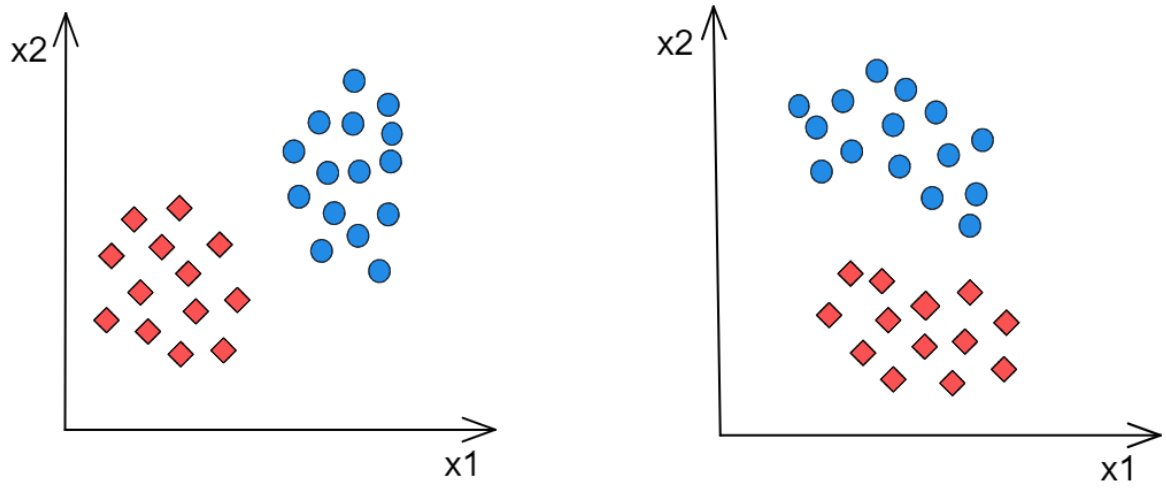


Figure 3.1: Covariate shift visualised in a 2-dimensional feature space.

3.3 Visual Unsupervised Domain Adaptation Methods

Before discussing the SFUDA methods, it is worth providing some background information about the classical UDA techniques. UDA has been extensively researched in recent years, and reviewing all the existing strategies is not in the scope of this project. Instead, following the [12], we will introduce three general categories of UDA techniques that can be applied to deep neural networks.

3.3.1 Discrepancy-based methods

Methods falling into this category aim to achieve the domain adaptation by measuring the distance between features coming from two domains and trying to minimise it. Deep Adaptation Network (DAN) [39] is one of the first works applying this idea to convolutional neural networks. DAN modifies the standard CNN architecture by adding a second Multilayer Perceptron classifier. One MLP takes the source feature vectors as input and learns the model's weights in the standard way using available labels. The new MLP is fed with target features, but it can not update its weights in a supervised manner as the target labels are not available. Instead, a distance metric called Multi-Kernel Maximum Mean Discrepancy (MK-MMD) [20] is applied to estimate the distance between source features in the first MLP and target features in the second MLP. The MK-MMD distance is then added to the model's loss function and minimised in the training, resulting in the decreasing discrepancy between the domains.

3.3.2 Reconstruction-based methods

Next category of deep domain adaptation is based on the concept of autoencoders [6], which are a class of neural networks used in unsupervised learning to learn an efficient representation of data. Deep Reconstruction-Classification Network (DRCN) [16] is an example architecture utilising autoencoding to achieve the UDA task. It consists of the encoding module, which is a standard CNN feature extractor, and the decoding module utilising the deconvolutional network [45]. DRCN training involves two pipelines that are run simultaneously. The first pipeline is the standard supervised learning on the labelled source data, that passes the features extracted by the encoder to the MLP classifier, and optimises the model for the object classification task. The second pipeline takes target features representations output by the CNN decoder and feeds them to the encoder network, which tries to achieve the best possible reconstruction of the original input image. The shared CNN feature extractor is therefore optimised by the both pipelines, i.e., it must learn high-level feature representations that will be good for both classification of source images and also allow for a successful reconstruction of target images.

3.3.3 Adversarial-based methods

The last category of techniques derives from the idea of the Generative Adversarial Network (GAN) [19]. GANs are an approach to generative modelling, which is an unsupervised machine learning task that aims to learn a model able to generate new data samples indistinguishable from the original dataset. Domain-Adversarial Neural Network (DANN) [15] is the classical example of this type of domain adaptation method. In its most basic form, it consists of three modules: CNN feature extractor, MLP label classifier, and a domain classifier. The domain classifier can be trained to distinguish if a data sample is coming from the source or the target domain. The DANN loss function is defined in such a way that when training proceeds, the model simultaneously learns to correctly predict the source

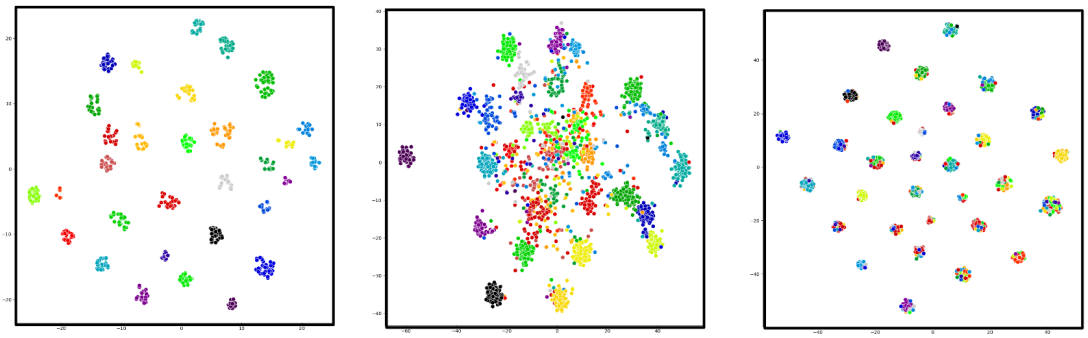
samples labels, but also make the source and target features indistinguishable for the domain classifier. In other words, the CNN feature extractor is forced to learn domain-invariant feature representations while maintaining a good classification performance.

3.4 Visual Source-Free Unsupervised Domain Adaptation Methods

Compared to the Unsupervised Domain Adaptation its Source-Free counterpart is a relatively new field of research. However, there already exists a number of widely accepted SFUDA techniques, and some of them offer even better performance than standard UDA methods.

In order to define the general categories of the SFUDA strategies we should have a closer look into outputs of the deep neural network feature extractor. As discussed in the previous chapter, a DNN architecture such as CNN has a feature extractor module, which takes a raw data sample as an input and returns its high-level feature vector representation. This representation can be described as a point in a d-dimensional feature space, where d is the size of the feature vector. As can be seen on the t-SNE [62] plot (Fig.3.2), the source feature vectors extracted by the pre-trained source model tend to form clusters of features belonging to the same class. In the ideal scenario, the clusters should not contain features from different classes, and they should be well separated from each other. If that is the case, the classifier module of DNN is able to easily learn a decision boundary that correctly classifies all the samples. However, such clear intra-cluster consistency and inter-class separability can not be achieved for the target data under the condition of the covariate distribution shift. The existing SFUDA methods aim to, either implicitly or explicitly, adapt the source model to achieve clear and consistent clustering of the target features.

As suggested in [61], we can identify two leading subcategories of strategies to tackle the SFUDA problem: global adaptation methods and local consistency methods. According to this categorisation, the first group of methods is better in achieving well-separated target feature class clusters, which comes with the price of false predictions inside each cluster. On the other hand, though the local strategies are good in maintaining consistent predictions within the clusters, they usually fail to achieve satisfactory separation between classes.



A. OFFICE-31 SOURCE-WEBCAM FEATURES

B. OFFICE-31 TARGET-AMAZON FEATURES (BEFORE ADAPTATION)

C. OFFICE-31 TARGET-AMAZON FEATURES (AFTER ADAPTATION)

Figure 3.2: t-SNE visualisation of feature vectors output by the source model feature extractor. Colours correspond to different objects. Plot **A** shows clear clustering of source features representing the same objects (same classes). On plot **B** it can be seen that the feature outputs are more noisy due to the introduced domain shift. Plot **C** shows outputs of the feature extractor adapted with the Attracting and Dispersing method. The clear separation of clusters is restored, however some noise can still be observed inside the clusters.

3.4.1 Global Adaptation Methods

One of the example solutions falling into this category is based on the Collaborative Class Conditional Generative Adversarial Networks (3C-GAN) [36]. It proposes the conditional variant of GAN, independent of the source data, that is able to produce labeled, target-style images. The new synthesized data provide supervision that helps to adapt a model. Several methods are based on the concept of pseudo labeling [4]. Pseudo labeling is a semi-supervised learning technique, which uses predictions of a model trained on the human-labeled data, to label unseen unlabeled test data, and then adds them to the original training set to train a new model. Contrastive Prototype Generation and Adaptation (CPGA) [50] approach exploits the pre-trained source model to generate representative features (prototypes) for each source class and also produce pseudo-labels for target data. It then proposes a special algorithm to align the pseudo-labeled target samples with the synthesized source class prototypes. Both prototype generation and adaptation are achieved via the contrastive learning [34]. Another example, and probably one of the most well-established SFUDA method is Source HypOthesis Transfer (SHOT) [37]. SHOT learns a target-specific feature extractor based on self-supervised pseudo-labeling of target data and concept of Information Maximization (IM) loss function [33]. In principal, optimising the IM loss trains the source model to output target predictions that are certain and globally diverse. Certainty means that the estimated class membership probability is high for one class and low for other classes (i.e., softmax function output for target sample should be close to one-hot vector [1.0, 0.0, 0.0, ...]). Diversity ensures that we do not end up with a trivial solution where for all the target samples model outputs the same one-hot encoding. In addition, leveraging the global supervision from pseudo-labels,

further reduces error in the target domain.

3.4.2 Local Consistency Methods

In Fig.3.2 we presented how the source features extracted from the source model tend to form well-separated clusters of class features. However, as can be seen in the same figure, feature representations obtained for the target data are not completely chaotic. We observe more noise within the clusters and there is not clear separation between them, but in general, the cluster-like structure is preserved. Because of the similarity of both domains, some of the patterns learned on the source domain are still reflected in the target domain. What distinguishes the local methods from the global adaptation approach is the fact that they explicitly leverage this intrinsic neighbourhood nature. Each of these techniques proposes optimization loss function, which aims to encourage consistent predictions between nearest neighbours in the target feature space. One of the first works pursuing this objective is General Source Free Domain Adaptation (G-SFDA) [75] which introduced a simple algorithm to ensure the same predictions for k nearest neighbour target features. Neighborhood Reciprocity Clustering (NRC) [74] extends this idea, and assigns different importance to reciprocal and non-reciprocal nearest neighbours, i.e. two neighbours that are among each other's k nearest neighbours are more likely to share the same label. Finally, Attracting and Dispersing (AaD) [76], one of the most recent works, introduces a new optimization objective, which directly forces samples that lie outside the closest neighbourhood to have dissimilar predictions. The main weakness of these approaches lies in their sensitivity to noisy neighbours. If the majority of the nearest neighbours of the given sample are classified incorrectly, forcing a consistency of predictions will degrade the model's performance instead of improving it. However, local consistency techniques are usually simpler and require less computational power, as they do not use any additional neural network modules or processes such as pseudo-label generation.

3.4.3 Attracting and Dispersing Algorithm Details

The AaD method operates in the previously specified SFUDA setting where we are given a source-pretrained DNN model and an unlabeled target dataset $X_t = \{x_1, ..., x_n\}$ containing samples drawn from the target feature space $\mathcal{X}_t$. As we already know, the DNN model consists of two parts: the feature extractor module f, and the classifier module g. For each target sample x_i, the feature extractor outputs a corresponding d-dimensional feature representation vector z_i, i.e., $f(x_i) = z_i \in \mathbb{R}^d$. The classifier takes the feature vector z_i as an input and outputs a membership probability distribution p_i of C possible classes, i.e., $p_i = \delta(g(z_i)) \in \mathbb{R}^C$, where δ is a softmax function.

AaD is based on the assumption that the feature vectors representing target samples belonging to the same class are located relatively close to each other in the d-dimensional space compared to the feature vectors from different classes. In order to capture this relation,

for each feature vector z_i, the AaD paper defines two feature sets: close neighbourhood set N_i containing K-nearest neighbours of z_i, and background set B_i containing all the features that are not in N_i. However, as we will see in a short moment, the actual implementation of the AaD algorithm uses a simplified version of B_i with only a subset of target samples. **Algorithm 1** presents a high-level overview of the AaD adaptation process. We will go through each individual step and discuss it in more detail. The flow of data is also shown in Fig.3.3.

Before the actual adaptation starts, we first create two memory banks (i.e., data stores that will be accessed throughout the whole execution of the program). The features memory bank F is a matrix dedicated to store the feature vectors of all samples in the target dataset, hence its shape is [X_t size, z_i size]. The softmax scores memory bank S keeps softmax predictions output by the model's classifier for every feature vector, therefore its shape is [X_t size, p_i size (number of classes C)]. As the starting point, the memory banks are initialised with values output by the unadapted model (i.e., target data are passed through the source model to obtain initial feature representations and predictions). In each iteration of the actual adaptation we sample a batch of data (i.e., a subset of target dataset X_t) and pass it to the feature extractor. The rows of the feature bank, which store old feature representations of the samples in the current batch, are now replaced by the new feature vectors. The feature vectors are then sent to the classifier, and the obtained softmax predictions are used to update the softmax scores memory bank. Next, we compute cosine similarity [] between each feature vector in the batch and each feature vector in the feature bank (i.e., for each batch sample we calculate its distance from **every** sample in the target dataset). The K feature vectors with the largest cosine similarity (i.e., the lowest distance) constitute the neighbourhood set N_i of a feature z_i, and all the features in the batch except z_i are assigned to a background set B_i. B_i constructed in this way guarantees a good estimate of the distribution of the whole target dataset, and it avoids computationally expensive extraction of all the features that do not belong to N_i. We also extract corresponding softmax predictions for each of the K nearest neighbours. Having obtained all the required values, we are ready to define the loss function L the AaD seeks to optimise:

$$L = \mathbb{E}[L_i(N_i, B_i)], \text{ with } L_i(N_i, B_i) = -\sum_{j \in N_i} p_i^T p_j + \lambda \sum_{m \in B_i} p_i^T p_m \tag{3.1}$$

The first term of L_i is calculated as a dot product between softmax score vector p_i of a feature vector z_i and softmax score vector of each of its K-nearest neighbours. The second term is defined as a dot product between z_i and softmax score vector of each the remaining feature vectors in the batch. The final batch loss L is an arithmetic average $\mathbb{E}$ of losses L_i for each feature z_i in the batch. To understand the role of each term, it is important to notice, that the dot product between the softmaxed predictions is maximal if both vectors have the greatest value for the same class and are close to one-hot vector. For example, consider two cases: (1) $p_i = [0.2, 0.4, 0.35, 0.25]$ and $p_j = [0.3, 0.1, 0.25, 0.35]$, and (2) $p_i = [0.8, 0.05, 0.04, 0.11]$ and

$p_j = [0.9, 0.02, 0.05, 0.03]$. The dot product $p_i^T p_j$ for the values in (2) will be greater than for predictions in (1). With this knowledge, we can give the following intuitive interpretation of terms in the loss function:

- The first term is preceded by a minus sign, hence to minimise the loss function L, we must maximise the dot product between predictions for feature vector z_i and predictions for its K-nearest neighbours. This should enforce the prediction consistency between the local neighbours.
- In the second term we want to minimise the dot product between predictions for feature z_i and predictions for the features that do not belong to its neighbour set, which should enforce different predictions for the dissimilar features.

The implementation of the second term can be confusing at first look, as it is possible for sets N_i and B_i to overlap (i.e., there is a chance, that a neighbour of z_i was also sampled in the same batch). This would result in the situation where we try to force the consistent softmax prediction with some feature vector's prediction, and simultaneously make the both predictions maximally different. However, as argued in the AaD paper and proved experimentally, overall similarity with features in N_i is higher than that with B_i, and the occasional intersections do not disturb the training. The λ hyperparameter, which has not been yet discussed, is applied to decay the second term of the loss function as the training continues. The reason behind it is that as features are gradually clustering, we want to weaken the influence of the dispersing term.

Algorithm 1 AaD Algorithm [76]

Require: Source-pretrained model and target dataset X_t

Initialise features memory bank F and softmax scores memory bank S
while Adaptation **do**
  Sample batch of data from X_t and input it into the model
  Update F with output of the feature extractor
  Update S with outputs from the classifier
  for each feature vector z_i in current batch **do**
    Retrieve its K-nearest neighbours from F
    Retrieve softmax score from S for each of its K-nearest neighbours
  end for
  Update model parameters to minimise the loss function 3.1
end while

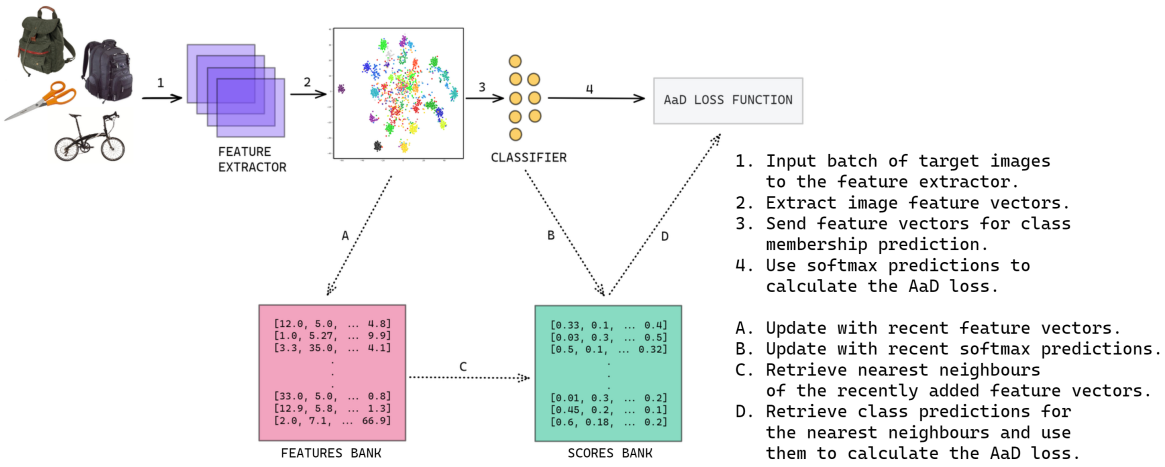


Figure 3.3: Data flow in a single iteration of the Attracting and Dispersing (AaD) adaptation procedure. Steps 1-4 correspond to the standard flow in the classical model training. Steps A-D are introduced by the AaD method. The model parameters are updated after each iteration to minimise the AaD loss function.

3.4.4 Methods Comparison

Similarly to the AaD paper, we can compare all the introduced methods based on the two optimisation objectives, *discriminability* and *diversity*. Discriminability relates to the previously mentioned goal of the consistent clustering of target features belonging to the same class. Diversity term corresponds to the goal of the clear inter-cluster separability. We can decompose each method's loss function into terms that express the above two objectives. This is presented in Table 3.1:

Method	**discriminability term**	**diversity term**
3C-GAN [36]	l_{cluReg} first term	l_{cluReg} second term
CPGA [50]	$\mathcal{L}_{NC}$	$\mathcal{L}^{w}_{con}$
SHOT [37]	$H(Y\|X)$	$-H(Y)$
G-SFDA [75]	$\mathcal{L}_{LSC}$ first term	$\mathcal{L}_{LSC}$ second term
NRC [74]	$\mathcal{L}_{N}$	$\mathcal{L}_{div}$
AaD [76]	$-\sum_{j\in N_i} p_i^T p_j$	$\lambda \sum_{m\in B_i} p_i^T p_m$

Table 3.1: Decomposition of methods into two terms: discriminability and diversity. l_{cluReg} is clustering regularisation applied to the final 3C-GAN loss, $\mathcal{L}_{nc}$ is neighbourhood clustering loss and $\mathcal{L}^{w}_{con}$ is contrastive loss of CPGA, $H(Y|X)$ is a conditional entropy term and $-H(Y)$ is a marginal entropy term of SHOT IM loss function, $\mathcal{L}_{LSC}$ is a local structure clustering loss minimised by G-SFDA, $\mathcal{L}_{N}$ is term measuring similarity of predictions and $\mathcal{L}_{div}$ calculates predictions diversity in the NRC final loss.

3.5 Domain Adaptation for Graph-Structured Data

While most of the methods introduced so far have been designed specifically for visual data tasks, the field of graph-structured data domain adaptation remains to be an insufficiently researched area. Only a few methods have been proposed for the graph UDA task. The examples include Unsupervised Domain Adaptive Graph Convolutional Networks (UDA-GCN) [69] which uses an attention mechanism to achieve global and local consistency of features, or Domain Adaptive Network Embedding (DANE) [80] also dependent on the graph convolutional network architecture and adversarial learning regularisation. Disentanglement-based Graph Domain Adaptation Model (DGDA) [8] is another proposed technique, based on the graph generation process. The graph SFUDA task setting is an even less popular research direction, with the Source-Free Domain Graph Adaptation Algorithm (SOGA) [41] being the only approach identified at the time of writing this review. However, SOGA is dedicated to the node property prediction task, which focuses on classifying nodes, not a whole graph. Considering the architectural similarities between deep learning models dedicated for images and those proposed for graphs, some of the classical domain adaptation strategies are applicable to both of them. For instance, the Domain-Adversarial Neural Network approach discussed before can be easily integrated with Graph Neural Networks, and is in fact selected as one of the baseline domain adaptation techniques in both UDA-GCN and DANE paper experiment settings. Margin Disparity Discrepancy (MDD) [79] is also adapted to graph-structured domain adaptation in some of the works, but it was originally established as a method for the computer vision UDA tasks. However, it is hard to find examples of the visual SFUDA strategies applied to graphs, which creates promising opportunities for research.

3.6 Summary

After this chapter it should be understood that domain can be defined as consisting of three elements: input feature space, output label space, and the joint probability distribution of features and labels drawn from the both spaces. The conventional domain adaptation aims to solve the covariate shift problem, which is a case where marginal probability distribution of the input features is different between domains. The literature distinguishes different domain adaptation settings based on the availability of labels in the target domain datasets. The Unsupervised and Source-Free Unsupervised Domain Adaptation are particularly interesting realistic problem settings that assume complete lack of target labels, and access to the source data is also restricted in the latter. We saw examples of UDA strategies that can be assigned to one of the three general categories: discrepancy-based, reconstruction-based, and adversarial-based methods. The existing SFUDA techniques can be described as global adaptation or local consistency methods, and the Attracting and Dispersing method belongs to the second category. We overviewed the AaD algorithm and compared it with the other methods based on the discriminability and diversity optimisation objectives. Finally,

the brief review of the existing graph domain adaptation approaches indicated lack of the well-established techniques for this type of data.

Chapter 4

Project Requirements and Analysis

The project's ultimate objective is to test if a SFUDA method designed originally for the computer vision tasks could be also successfully applied to the models operating on the graph-structured data. More specifically, it aims to imitate the SFUDA problem setting in the graph classification task, and approach it with the introduced Attracting and Dispersing method. The relative simplicity of the AaD's strategy, and fact that it relies only on the feature extractor and classifier outputs, give foundations to reason that it can be effectively used with both the CNN and GNN types of models. The project will be split in two main stages, that should incrementally build towards showing if this expectations are valid:

1. Reproduction of the AaD paper experiment results to ensure validity of the proposed method and understand its implementation details.
2. Integration of the AaD technique with the existing graph property prediction solutions, and examining its performance in the specially designed graph classification SFUDA setting.

4.1 Stage I: Reproducing the State-of-the-Art Visual Source-Free Unsupervised Domain Adaptation

Authors of the AaD paper provide public access to the GitHub repository with PyTorch implementation of their method: `https://github.com/Albert0147/AaD_SFDA`. Using the provided code, the effort required to reproduce the experiments can be summarised in the following steps:

1. cloning and debugging the repository,
2. modifying the code to report results needed for this project,
3. setting up the experimental environment on the University of Sheffield High Performance Computing (HPC) machines, and running the experiments with the code we tested locally.

It should be emphasised that the experiments reproduction is not a trivial task of just cloning and running other people's code. First of all, any potential bugs in the files provided by researchers must be identified, which unfortunately are not a rare issue. The debugging can be done on the local machine by running the code on the reduced-size version of the experiment datasets. Things that should be potentially investigated before any experiments can be run include: a correct format of data fed into a model, correct implementation of the evaluation metrics, or the general reasonability of results (i.e., if outputs from a model roughly match our expectations). After ensuring the code is working as expected, it can be further adjusted to our needs, such as generation of plots or reporting of additional statistics that have not been originally implemented. Moreover, Deep Neural Network models and datasets used in the experiments require access to powerful GPUs and abundance of RAM to allow for the efficient computations. For these reasons, all the models required for the experiments reproduction will be trained using resources provided by the HPC service. Enough time must be reserved to set up the experimental environment and install all the required software on the remote machines.

4.1.1 Datasets and Evaluation

Both UDA and SFUDA settings are extensively researched problems in the field of computer vision. Therefore, a set of well-established visual benchmarks has been defined to allow consistent and fair comparison of developed methods. The three datasets, on which AaD was originally tested, and which will be also used by this project, include:

- **Office31** [56] containing images split into 3 domains of origin: pictures of items from *Amazon*, pictures taken with a *webcam*, and pictures taken with a digital camera (*DSLR*). Each domain contains 31 classes of items commonly encountered in office settings. There are 4,652 images in total.
- **Office-Home** [63] split into 4 domains of origin: artistic images in the form of sketches or paintings (*Art*), *Clipart* images, images from *product* offering websites, and images captured with a camera in a *real-world* setting. Each domain consists of 65 classes of items encountered in an office and/or home. There are 15,500 images in total.
- **VisDA-C** [48] contains 2 domains: 152,000 *synthetic* images and 55,000 *real-world* images presenting 12 object classes including people, vehicles or nature.

To show validity of results reported by authors of the AaD paper, it is important to reproduce the same experimental domain adaptation scenarios. The standard set of the SFUDA settings defined for each of the datasets is consistent across related research publications and described as follows:

- **Office-31**: train a source model for each domain, which results in 3 pre-trained models. Next, adapt each model to the 2 remaining target domains, which gives 6 different adaptation scenarios (Source Domain $\rightarrow$ Target Domain):

Amazon → DSLR, Amazon → Webcam, DSLR → Webcam, Webcam → DSLR, DSLR → Amazon, Webcam → Amazon.

- **Office-Home**: following the same procedure results in 12 possible adaptation tasks (Source Domain → Target Domain):
  Art → Clipart, Art → Product, Art → Real-World, Clipart → Art, Clipart → Product, Clipart → Real-World, Product → Art, Product → Clipart, Product → Real-World, Real-World → Art, Real-World → Clipart, Real-World → Product.

- **VisDA-C**: only one domain adaptation setting is defined (Synthetic → Real-World). However it is common to report results obtained for each of the 12 classes: Plane, Bicycle, Bus, Car, Horse, Knife, Motorcycle, Person, Plant, Skateboard, Train, Truck.

The reproduction of exactly the same results as declared in the paper may be difficult or even impossible due to the factors such as discrepancy between software library versions and intrinsic randomness of Deep Neural Network algorithms. In addition, factors like different types of GPUs used for training a model can also affect the performance. Therefore, the aim is to obtain the scores that are sufficiently close to the original paper's results. In general, differences of 1-3% point between obtained and declared scores are acceptable to consider the method as valid. Performance on all the three datasets is conventionally assessed with the *accuracy* score of target predictions.

4.2 Stage II: Applying Attracting and Dispersing to Graph Source-Free Unsupervised Domain Adaptation

Application of the selected method to the completely new type of problem is a more complex challenge. First of all, as has been already discussed in the previous chapter of this report, the UDA and SFUDA settings for graph-structured data have been addressed directly in only a few previous works. This stays in contrast to the extensive research done in the computer vision field, and results in lack of equivalently well-established domain adaptation benchmarks. Next, from the purely technical point of view, the provided AaD code implementation is tailored to the specifications of the visual benchmarks experiments, hence a significant effort must be dedicated to integrate it with existing graph classification systems. In general, the plan of work to be completed in this stage of the project can be summarised in the following steps:

1. use the public classical graph property prediction benchmark datasets and preprocess them to create the SFUDA experimental scenarios similar to the ones defined for visual experiments,

2. clone and modify one of the existing graph classification repositories to work with the new experimental setting, in particular, make it compatible with the AaD framework,

3. run a set of experiments investigating performance of the AaD method in the graph classification SFUDA setting of our design.

4.2.1 Datasets and Evaluation

The data selected for this project will come from the Open-Graph Benchmark (OGB) [24] service, which provides a collection of realistic benchmark datasets for machine learning on graphs. More specifically, the three molecular-biology datasets will be utilised:

- **ogbg-molbbbp**: containing 2,039 molecules adopted from MoleculeNet [71] dataset and pre-processed to graph-structured representations, where nodes are atoms, and edges are chemical bonds. The graphs are labelled with either 0 or 1, which indicates a molecule's ability to penetrate blood-brain barrier (binary classification task),
- **ogbg-molhiv**: obtained in the same way, and consisting of 41,127 molecular graphs with 0/1 labels indicating a molecule's ability to inhibit the HIV virus replication (binary classification task),
- **ogbg-ppa**: consisting of 158,100 graphs representing protein association neighbourhoods of 1,581 different species. Nodes are proteins, and edges are assigned attributes that describe biological associations between proteins. Each graph is labelled based on its membership in one of the 37 broad taxonomic groups (e.g., mammals, bacterial families, archaeans) (multi-class classification task).

To create the suitable domain adaptation setting it is required to split the above datasets into source and target domain sets with different probability distributions. This can be achieved by leveraging the natural properties of the original data samples, such as molecular structure. Details of these splits are the subject of the *Implementation and Experiments Setting* chapter. Assuming that the datasets have been already divided in the appropriate way, it should be possible to apply the same experimental methodology as followed in the visual SFUDA papers.

Regarding the experimental code, it will be also convenient to rely on the resources provided by the OGB's team, and use their repository of graph classification examples `https://github.com/snap-stanford/ogb/tree/master/examples/graphproppred`.
They implement a number of conventional Graph Neural Network architectures, and utilise specially designed libraries for efficient and standardised pre-processing and loading of graphs. In addition, the OGB's package comes with standardised evaluators to assess the classification performance of applied techniques. The evaluation metrics recommended for the datasets used in this project are: *roc-auc* score for ogbg-molbbbp and ogbg-molhiv, and *accuracy* score for ogbg-ppa.

4.3 Summary

The project is divided into the two main parts. In the first stage, the work focuses on the execution of experiments described in the Attracting and Dispersing paper, which can be done by preparing the code repository published by the authors to run on the University High Performance Computing machines. The method should be evaluated on the Office-31, Office-Home and VisDA-C datasets, and obtained results should be similar to those declared in the paper. The aim of the second part of the project is to test if the same method can be successfully applied to the neural networks operating on graph-structured data. The experimental environment can be based on the datasets and code provided by the Open Graph Benchmark team. However, appropriate datasets pre-processing and modifications to the code are required to enable reliable experiments in the Source-Free Unsupervised Domain Adaptation setting.

Chapter 5

Implementation and Experiments Setting

A significant part of time spent on this project was dedicated to developing the suitable experimental environment. This chapter is meant to provide an overview of the most important code implementation details, give specifics of the datasets preparation, and precisely describe the applied experimental procedure.

5.1 Code Modifications

The final version of the code used for the AaD experiments reproduction is not much different from the original paper's implementation. Instead of keeping separate code files for each benchmark, which was the state of the original version, the code was modified to make all the experiments runnable from the same files. This helped to ensure consistency of the implementation across experiments, and eased debugging. The code was modified to enable the t-SNE embedding and plotting of feature vectors output from a feature extractor module of a model. It was achieved by altering the model's trainer code to save the state of the AaD memory banks before and after the adaptation. The saved features and softmax scores are then sent to the plotter module.

The OGB's graph property prediction repository relies on their own data loading and evaluation package, which required a few minor changes to make it compatible with the new experimental setting. Specifically, it was necessary to alter functionality of the OGB's PygGraphPropPredDataset class, to make it recognise new types of dataset splits that will be introduced later in this chapter. In addition, a small change was also needed in the PyTorch Geometric (PyG) [13] InMemoryDataset class, to make it return indexes of samples passed to a model in each training iteration. This information is required for the AaD algorithm to update appropriate rows of the memory banks. To avoid changing the original third party packages source code, the *Monkey Patching* [26] approach was used to modify the mentioned functionalities at runtime [Appendix A]. The AaD algorithm was

implemented following the conference paper implementation. However, unlike in the original AaD code, it was structured as a separate Python class, which can be imported to the training file [Appendix A]. This implementation makes the adaptation procedure easier to follow, and it also facilitates reusability in any potential future experiments, as it is not bound to any particular benchmark or network type. Similarly to the experiments on the visual benchmarks, the new version of the experimental code also enables the t-SNE features plotting to allow for deeper analysis and understanding of obtained results.

5.2 Datasets Preparation

The three visual benchmark datasets (Office31, Office-Home, VisDA-C) did not require any additional preprocessing as the images were already split into folders by their domain of origin and by their corresponding class. It was only needed to make the datasets stored in the HPC's storage accessible by the experimental code.

In contrast, creating the suitable graph classification SFUDA setting (i.e., setting with two related datasets with different probability distributions) was a challenge of its own. This could be achieved by applying the type of dataset split that ensures some level of dissimilarity between samples in the train (source) and test (target) set. The OGB package originally applies the following splits [24]:

- **ogbg-molbbbp** and **ogbg-molhiv** are divided into training, validation, and test sets using the *scaffold* splitting technique. Scaffold splitting partitions molecules into different sets based on their 2D dimensional structures. This results in the structurally different molecules being assigned to different sets.
- **ogbg-ppa** is also divided into training, validation, and test sets, but it uses the *species* splitting strategy. The protein association neighbourhood graphs in the validation and test sets are extracted from different species than those in the training set.

Although scaffold splitting seems to fit our purpose perfectly, there exist some issues related to this strategy. For example, [5] argues that in certain situations, molecules sharing the same topology can be treated as dissimilar and end up in different sets. Therefore, the *cluster-based* split is proposed, which partitions data to clusters and guarantees that distances between clusters are always greater than a predefined threshold. Specifically, in this project we use the single-linkage hierarchical clustering algorithm proposed in [42]. The algorithm takes the original SMILES [67] string representations of molecules from the MoleculeNet datasets, and converts them to ECFP4 [54] representation. Next, it uses the minimum Jaccard distance threshold of 0.5 to split the molecules into clusters. Finally, 60% of randomly selected clusters is distributed to the source set while the remaining 40% is treated as a target set, which creates the cross-domain scenario analogous to the setting of the visual benchmarks. The species split applied to the proteins dataset is already a good imitation of the cross-domain scenario,

and should guarantee the sufficient distribution shift. The only modification made to it, was merging the given validation and test sets into one target domain dataset, where the original training set serves the purpose of the source dataset.

5.3 Experiments Details

The general experimental methodology was be the same across all datasets and for all possible source-domain scenarios:

1. Train a model on the full source dataset in the classical supervised manner (i.e., using labels)
2. In each training epoch, track the model's score on the source dataset, and save the version with the best performance (i.e., the highest accuracy or roc-auc score)
3. Retrain the best source model on the unlabeled target dataset using the AaD method.
4. In each adaptation epoch, track the model's score on the target dataset, and report the best obtained score.

5.3.1 Visual Benchmark Experiments

The experiments on the three visual domain adaptation benchmarks followed exactly the same choice of the neural network model architecture, optimisation algorithm, and hyperparameters as in the original AaD paper:

Office-31: ResNet-50 [22] CNN architecture, Stochastic Gradient Descent (SGD) optimiser with momentum 0.9, and batch size of 64. The learning rate applied to the classifier module of the model was equal to 1e-2 and for the feature extractor module it was set to 1e-3. In the source training stage the Cross Entropy with label smoothing [43] was selected as a loss function. The number of nearest neighbours K in the AaD algorithm was set to 3. The decaying parameter in the second term of the AaD loss function was calculated as $\lambda = (1 + 10 * \frac{current\_adaptation\_iteration}{total\_num\_of\_adaptation\_iterations})^{-\beta}$, where the exponent β controls the decaying speed and it was set to 2. Both source training and target adaptation ran for 40 epochs.

Office-Home: Setting equivalent to the one for Office-31 except the decaying exponent value β, which was set to 0 in this case.

VisDA-C: The bigger ResNet-101 [22] CNN architecture was selected, and the learning rates were set to 1e-4 and 1e-3 for the feature extractor and classifier respectively. The number of nearest neighbours K was set to 5, and the decaying exponent β was also equal to 5. Source training and target adaptation were set to run for 15 epochs.

5.3.2 Graph Data Experiments

The experiments for graph-structured data were executed on all the GNN architectures available in the used OGB's repository: Graph Convolutional Network (GCN), Graph Convolutional Network with Virtual Node (GCN+V), Graph Isomorphism Network (GIN), and Graph Isomorphism Network with Virtual Node (GIN+V) [78, 32, 73]. This allowed for effective performance comparison between different types of conventional models. All the models were defined with the following same attributes: *number of message passing layers* = 5, *embedding dimensionality* = 300, and *mean global pooling* operation applied to combine graph nodes representations into a final feature vector. Similarly to the original code version, the Adam optimization algorithm was used with learning rate value of 1e-2 for source training, and the source training loss was calculated with the standard Cross Entropy function. The learning rates were reduced in the target adaptation training and set to 1e-5 and 1e-4 for the feature extractor and classifier respectively. In addition:

ogbg-molbbbp used a batch size of 16 samples. The number of nearest neighbours K in the AaD algorithm was equal to 3, and the decaying exponent β of 5 was applied. The source training ran for 100 epochs, while target adaptation took 30 epochs.

ogbg-molhiv all the parameters and training times were equal to those applied in mol-bbbp experiments.

ogbg-ppa used a larger batch size of 64, and the number of nearest neighbours K equal to 7, but the decaying exponent β was set to 5, similarly to other datasets. The source model training was also set to 100 epochs, while the target adaptation was executed for 15 epochs.

Chapter 6

Experiment Results and Discussion

6.1 Image Data Results

The reported results are the average of three random runs. Specifically, for Office-31 and Office-Home experiments, the results for all possible source-domain combinations (tasks) are presented (e.g., Amazon to DSLR, Amazon to Webcam, etc.), and the average score in all the tasks is also reported. In VisDA-C, the average result for each class is reported (e.g. plane, car, horse, etc.), and the final score is the average over all the classes.

Model	**A→D**	**A→W**	**D→A**	**D→W**	**W→A**	**W→D**	**Avg**
ResNet-50	79.10	74.54	60.23	94.01	62.73	98.30	78.15
ResNet-50$_{AaD}$	95.25	92.49	76.20	98.36	76.68	99.93	89.81

Table 6.1: Target domain accuracies (%) on the Office31 dataset achieved by the source model before adaptation and after adaptation with the Attracting and Dispersing method. In each column, the source domain is the one at the left side of an arrow. (Amazon = A, DSLR = D, Webcam = W)

Model	**A→C**	**A→P**	**A→R**	**C→A**	**C→P**	**C→R**	**P→A**	**P→C**	**P→R**	**R→A**	**R→C**	**R→P**	**Avg**
ResNet-50	35.15	50.21	57.42	37.11	40.99	46.06	39.90	32.10	60.77	54.04	41.82	60.55	46.34
ResNet-50$_{AaD}$	57.75	77.97	79.76	66.34	77.60	78.50	65.43	56.62	80.84	70.87	58.38	84.20	71.18

Table 6.2: Target domain accuracies (%) on the Office-Home dataset achieved by the source model before adaptation and after adaptation with the Attracting and Dispersing method. In each column, the source domain is the one at the left side of an arrow. (Art = A, Clipart = C, Product = P, Real = R)

Model	plane	bicycle	bus	car	horse	knife	motorcycle	person	plant	skateboard	train	truck	Avg
ResNet-101	55.22	53.12	62.54	59.89	80.03	18.52	79.67	31.00	80.73	27.12	73.14	8.75	52.47
ResNet-101$_{AaD}$	97.11	90.02	80.84	75.91	97.24	96.03	89.81	82.33	95.19	93.50	91.87	62.60	87.70

Table 6.3: Target domain accuracies (%) on the VisDA-C dataset achieved by the source model before adaptation and after adaptation with the Attracting and Dispersing method. In each column, the source domain is the one at the left side of an arrow.

The original AaD paper reports average accuracies of 89.90%, 72.70% and 88.00% after adaptation on Office-31, Office-Home, and VisDA-C benchmarks respectively. The accuracy scores obtained during the reproduced experiments match these results, and therefore prove validity of the proposed method. The differences of ∼1.00% from the original results were expected and, as argued previously, are likely to be caused by different version of the PyTorch library, different hardware used, and the natural randomness involved in the model initialisation and training. AaD improved the average classification accuracy by ∼10.00% on Office-31, ∼25.00% on Office-Home, and ∼35.00% on VisDA-C, which shows the same level of performance as achieved by the current state-of-the-art SFUDA methods. Moreover, it can be observed, that the accuracy gain is comparable across or individual tasks, independently of how serious the initial domain shift was. For example, the first column of the Office-Home table shows a relatively hard scenario, where a model trained on the Art domain images had the initial accuracy of 35.15% in the Clipart domain, which rose to 57.75% after the adaptation. The similar improvement of ∼23.00% was also seen in the easier Real to Product task (last column), where the source model achieved over 60.00% accuracy even before the adaptation. In addition, VisDA-C table shows how significant can the performance improvement be for individual classes. For instance, the knives classification accuracy grew by almost 80.00% after the adaptation.

6.2 Graph Data Results

For ogbg-molbbbp and ogbg-molhiv three separate random cluster splits were applied, which result in three different domain adaptation tasks. For each task, the average scores of three random runs are reported, and additionally, the average score over all three splits is also calculated. For ogbg-ppa, only one source-target setting is evaluated (i.e., the original species split with training set used as source domain, and merged validation and test sets treated as target domain). The results are also the average of three random runs.

ogbg-molbbbp				
Model	**cluster split I**	**cluster split II**	**cluster split III**	**Avg**
GCN	0.783 ± 0.020	0.845 ± 0.017	0.835 ± 0.026	0.821 ± 0.027
GCN+V	0.802 ± 0.008	0.816 ± 0.004	0.825 ± 0.010	0.814 ± 0.009
GIN	0.777 ± 0.016	0.860 ± 0.009	0.797 ± 0.021	0.811 ± 0.035
GIN+V	0.785 ± 0.004	0.815 ± 0.009	0.848 ± 0.005	0.816 ± 0.025
GCN_{AaD}	0.790 ± 0.028	0.853 ± 0.020	0.885 ± 0.011	**0.842 ± 0.039**
$GCN+V_{AaD}$	0.817 ± 0.010	0.815 ± 0.025	0.847 ± 0.006	0.826 ± 0.014
GIN_{AaD}	0.794 ± 0.020	0.862 ± 0.011	0.860 ± 0.005	0.838 ± 0.031
$GIN+V_{AaD}$	0.822 ± 0.001	0.805 ± 0.008	0.863 ± 0.019	0.830 ± 0.024

Table 6.4: Target domain roc-auc scores (%) on the ogbg-molbbbp dataset. The results are presented as mean±standard deviation. The best score for each split is underlined and the best average score on all splits is marked in bold.

ogbg-molhiv				
Model	**cluster split I**	**cluster split II**	**cluster split III**	**Avg**
GCN	0.617 ± 0.016	0.622 ± 0.013	0.597 ± 0.009	0.612 ± 0.010
GCN+V	0.629 ± 0.018	0.659 ± 0.014	0.630 ± 0.013	**0.639 ± 0.013**
GIN	0.605 ± 0.005	0.608 ± 0.002	0.625 ± 0.004	0.612 ± 0.008
GIN+V	0.660 ± 0.016	0.637 ± 0.001	0.622 ± 0.023	0.639 ± 0.015
GCN_{AaD}	0.618 ± 0.020	0.584 ± 0.011	0.602 ± 0.004	0.601 ± 0.013
$GCN+V_{AaD}$	0.646 ± 0.008	0.636 ± 0.025	0.600 ± 0.017	0.627 ± 0.019
GIN_{AaD}	0.603 ± 0.011	0.620 ± 0.026	0.637 ± 0.009	0.620 ± 0.013
$GIN+V_{AaD}$	0.673 ± 0.018	0.578 ± 0.027	0.640 ± 0.015	0.630 ± 0.039

Table 6.5: Target domain roc-auc scores (%) on the ogbg-molhiv dataset. The results are presented as mean±standard deviation. The best score for each split is underlined and the best average score on all splits is marked in bold.

ogbg-ppa	
Model	**species split**
GCN	0.654 ± 0.005
GCN+V	0.663 ± 0.010
GIN	0.669 ± 0.002
GIN+V	0.674 ± 0.011
GCN_{AaD}	0.673 ± 0.0004
$GCN+V_{AaD}$	0.691 ± 0.005
GIN_{AaD}	0.681 ± 0.002
$GIN+V_{AaD}$	**0.716 ± 0.013**

Table 6.6: Target domain accuracies (%) on the ogbg-ppa dataset. The results are presented as mean±standard deviation. The best score is marked in bold.

The results obtained on the ogbg-molbbbp dataset show that application of the AaD adaptation method improved the average target roc-auc score of all the tested GNN

architectures. The best score of 84.20% was achieved with the adapted GCN model, which is better by ∼2.00% compared to the performance before the adaptation. However, when looking at results for each individual split, it can be seen that domain adaptation was more effective for GIN with virtual node model in the first split. In the second split, the adapted GIN model achieved the best score, but performance of models with virtual nodes was worse after the adaptation. The plots in Fig.6.1 also show an example of how applying the AaD technique affected the distribution of samples in the feature space. Target features output from the adapted model seem to reflect the original source data distribution and better separation of two classes can be observed. However, it does not resemble the well-separated cluster structure that appeared in the visual data example (Fig.3.2). The performance of AaD was less successful in the case of ogbg-molhiv dataset. Although the adapted models offered the best roc-auc score in two out of three splits, the best average target score was achieved by the vanilla source GCN model with virtual node. Applying the AaD adaptation to that model decreased its average roc-auc by ∼1.00%. From the t-SNE plots (Fig.6.2) it can be deduced that the original source model was not able to achieve any reasonable separation of two classes for target data, therefore there was no initial structure that the domain adaptation could improve on. The AaD strategy assumes that some level of separation is preserved for the target data even before the adaptation, which can not be observed in this case. The experiments executed on the ogbg-ppa dataset showed that the GIN model with virtual node achieved the best accuracy score under the introduced species split scenario, and after adaptation with the AaD method its score improved by ∼4.00%. The generated t-SNE plots (Fig.6.3) show the clustering of features that is similar to the structures observed with visual data. Outputs for the target features are noisy, but not completely chaotic and it is possible to indicate areas occupied by each class. However, it is hard to observe the significant structural difference after the adaptation. The class separation and intra-cluster consistency is not much better than before the adaptation, which is reflected by only small prediction accuracy improvement.

The experiments proved that AaD method can be integrated with several conventional GNN architectures and improve their classification performance in many of the prepared SFUDA scenario tasks. However, the observed prediction score gain was not as significant as in the visual data experiments. Moreover, in some cases, applying the adaptation technique resulted in slightly lower scores than before the adaptation. There are multiple reasons that could lead to the discrepancy between the image data and graph-structured data experiment outcomes. First of all, the utilised graph datasets and applied splitting strategies could create more challenging task setting. Especially, the selected cluster split threshold value could be too high resulting in stronger distribution shifts between source and target domains. Furthermore, the used GNN models were considerably smaller than the architectures used in the visual experiments. The ResNet-50 used with Office-31 and Office-Home data is a 50 layers deep model with around 25 million learnable parameters, and ResNet-101 is 101 layers deep containing around 45 million parameters. In contrast, the standard GCN model has only

500,000 parameters, and GIN with virtual node was the biggest GNN model used with around 3 million parameters in total. It is possible that such simple models do not have the same adaptation ability as the larger models. In addition, some works indicate that GNN models have limited ability to learn generalisable features, and the effective transfer of knowledge between domains may be not possible without special pre-training techniques [25]. Finally, the AaD strategy is sensitive to the choice of hyperparameters, such as a number of nearest neighbours, value of the decaying term, or even size of the data batch. It is possible that different configurations of hyperparameters would result in better performance on graph data.

Taking all of the above issues into consideration, the completed work can be treated as a starting point for the further, deeper investigation. One can experiment with different graph classification datasets or less rigorous splitting strategies and analyse how they affect the results. It would be also interesting to run the same experiments with more sophisticated, larger GNN model architectures, but it is also possible to try different settings of the already used models. For example, one could train models with different number of message passing layers, embedding sizes, or global pooling strategies. The AaD hyperparameters optimization is another direction to take, but it would be also worthwhile to evaluate and compare different existing domain adaptation methods in the same problem setting.

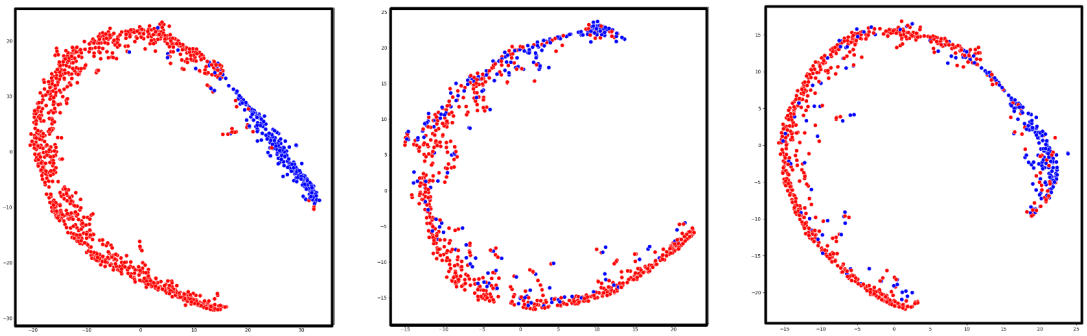


Figure 6.1: t-SNE plots visualising differences in distribution of ogbg-molbbbp feature vectors in one of the experiments. Blue - molecules that can not penetrate the blood-brain barrier, Red - molecules that can penetrate the blood-brain barrier.

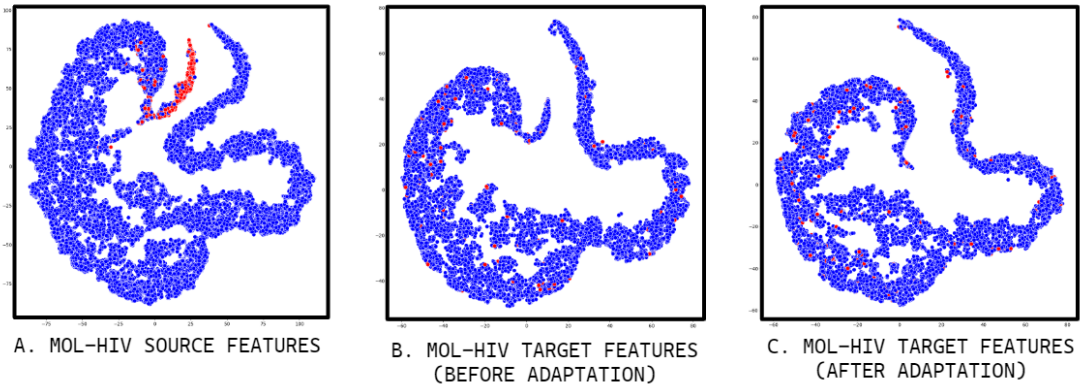


Figure 6.2: t-SNE plots visualising differences in distribution of ogbg-molhiv feature vectors in one of the experiments. Blue - molecules that can not inhibit HIV replication, Red - molecules that can inhibit HIV replication.

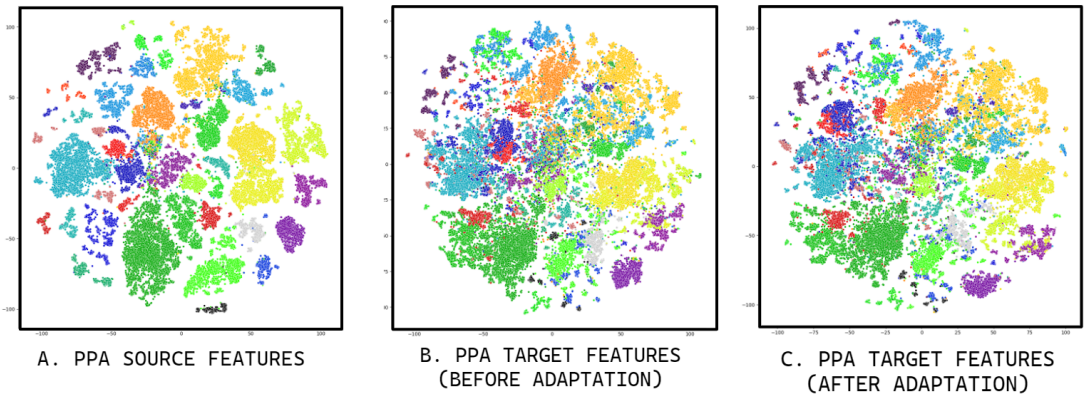


Figure 6.3: t-SNE plots visualising differences in distribution of ogbg-ppa feature vectors in one of the experiments. Colours represent different taxonomy groups.

Chapter 7

Conclusions

Domain Adaptation is an exceptionally broad area of research, and many proposed methods draw from the most recent machine learning concepts. Diversity of strategies is remarkable, and new unique approaches are published every month. It is also interesting to discover that some of them rely on techniques that were initially designed for different tasks. For instance, we could see examples of methods that have been based on autoencoders or adversarial networks. It is not an overstatement to say that domain adaptation explores frontiers of the current machine learning knowledge. Domain distribution shift is a universal problem that can be observed in all types of data. The probability distribution shift between two datasets can origin from different collection strategies, changing environment conditions or natural characteristics of samples. Even the most powerful machine learning algorithms are still sensitive to the above issues, and domain adaptation should remain an open research challenge in years to come.

The project analysed the Convolutional Neural Network model architecture commonly used in image classification tasks, and Graph Neural Network models that are dedicated to the operations such as graph classification. Although different in their implementation details, both architectures follow the same high level design principles, with the feature extractor module learning the vector representation of input samples, and classifiers trained to assign these vectors to correct classes. Considering these similarities, and motivated by the fact that Graph Neural Networks lack well-established domain adaptation methods, the project investigated the possibility of applying existing visual source-free unsupervised domain adaptation strategy to graph-structured data classification. To the best of the author's knowledge, this is one of the first works approaching this problem setting.

Attracting and Dispersing, the selected state-of-the-art visual SFUDA technique, was shown to be compatible with the conventional Graph Neural Network models. Most of the experiments executed in the specially designed graph data distribution shift scenarios showed improvement of the classification performance after application of the method. Although the obtained results should be the subject of deeper verification and analysis, they indicate

that the used strategy has potential to become a successful approach to graph SFUDA. The completed work established the solid foundations for further research. Among many directions that could be followed, experimenting with more advanced Graph Neural Network architectures or analysing effects of different domain adaptation methods in the prepared experimental setting seem to be particularly interesting. They could be chosen as topics of new separate projects.

Bibliography

[1] Albawi, S., Mohammed, T. A., and Al-Zawi, S. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)* (2017), Ieee, pp. 1–6.

[2] Allamanis, M., Barr, E. T., Devanbu, P., and Sutton, C. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR) 51*, 4 (2018), 1–37.

[3] Amari, S.-i. Backpropagation and stochastic gradient descent method. *Neurocomputing 5*, 4-5 (1993), 185–196.

[4] Arazo, E., Ortego, D., Albert, P., O'Connor, N. E., and McGuinness, K. Pseudo-labeling and confirmation bias in deep semi-supervised learning. In *2020 International Joint Conference on Neural Networks (IJCNN)* (2020), IEEE, pp. 1–8.

[5] Bai, P., Miljković, F., Ge, Y., Greene, N., John, B., and Lu, H. Hierarchical clustering split for low-bias evaluation of drug-target interaction prediction. In *2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)* (2021), IEEE, pp. 641–644.

[6] Bank, D., Koenigstein, N., and Giryes, R. Autoencoders. *arXiv preprint arXiv:2003.05991* (2020).

[7] Borgatti, S. P., Everett, M. G., and Johnson, J. C. *Analyzing social networks*. Sage, 2018.

[8] Cai, R., Wu, F., Li, Z., Wei, P., Yi, L., and Zhang, K. Graph domain adaptation: A generative view. *arXiv preprint arXiv:2106.07482* (2021).

[9] De Boer, P.-T., Kroese, D. P., Mannor, S., and Rubinstein, R. Y. A tutorial on the cross-entropy method. *Annals of operations research 134* (2005), 19–67.

[10] Dumoulin, V., and Visin, F. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285* (2016).

[11] Fang, Y., Yap, P.-T., Lin, W., Zhu, H., and Liu, M. Source-free unsupervised domain adaptation: A survey. *arXiv preprint arXiv:2301.00265* (2022).

[12] Farahani, A., Voghoei, S., Rasheed, K., and Arabnia, H. R. A brief review of domain adaptation. *Advances in Data Science and Information Engineering: Proceedings from ICDATA 2020 and IKE 2020* (2021), 877–894.

[13] Fey, M., and Lenssen, J. E. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428* (2019).

[14] Fredriksson, T., Mattos, D. I., Bosch, J., and Olsson, H. H. Data labeling: An empirical investigation into industrial challenges and mitigation strategies. In *International Conference on Product-Focused Software Process Improvement* (2020), Springer, pp. 202–216.

[15] Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. Domain-adversarial training of neural networks. *The journal of machine learning research 17*, 1 (2016), 2096–2030.

[16] Ghifary, M., Kleijn, W. B., Zhang, M., Balduzzi, D., and Li, W. Deep reconstruction-classification networks for unsupervised domain adaptation. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14* (2016), Springer, pp. 597–613.

[17] Gong, L., and Cheng, Q. Exploiting edge features for graph neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (2019), pp. 9211–9219.

[18] Goodfellow, I., Bengio, Y., and Courville, A. Softmax units for multinoulli output distributions. deep learning, 2018.

[19] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks. *Communications of the ACM 63*, 11 (2020), 139–144.

[20] Gretton, A., Smola, A., Huang, J., Schmittfull, M., Borgwardt, K., and Schölkopf, B. Covariate shift by kernel mean matching. *Dataset shift in machine learning 3*, 4 (2009), 5.

[21] Gupta, S., Girshick, R., Arbeláez, P., and Malik, J. Learning rich features from rgb-d images for object detection and segmentation. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VII 13* (2014), Springer, pp. 345–360.

[22] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[23] Hossin, M., and Sulaiman, M. N. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process 5*, 2 (2015), 1.

[24] Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems 33* (2020), 22118–22133.

[25] Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., and Leskovec, J. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265* (2019).

[26] Hunt, J., and Hunt, J. Monkey patching and attribute lookup. *A Beginners Guide to Python 3 Programming* (2019), 325–336.

[27] Ibraheem, N. A., Hasan, M. M., Khan, R. Z., and Mishra, P. K. Understanding color models: a review. *ARPN Journal of science and technology 2*, 3 (2012), 265–275.

[28] Japkowicz, N., and Stephen, S. The class imbalance problem: A systematic study. *Intelligent data analysis 6*, 5 (2002), 429–449.

[29] Kashima, H., and Inokuchi, A. Kernels for graph classification. In *ICDM workshop on active mining* (2002), vol. 2002.

[30] Khurana, D., Koli, A., Khatter, K., and Singh, S. Natural language processing: State of the art, current trends and challenges. *Multimedia tools and applications 82*, 3 (2023), 3713–3744.

[31] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[32] Kipf, T. N., and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[33] Krause, A., Perona, P., and Gomes, R. Discriminative clustering by regularized information maximization. *Advances in neural information processing systems 23* (2010).

[34] Le-Khac, P. H., Healy, G., and Smeaton, A. F. Contrastive representation learning: A framework and review. *Ieee Access 8* (2020), 193907–193934.

[35] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature 521*, 7553 (2015), 436–444.

[36] Li, R., Jiao, Q., Cao, W., Wong, H.-S., and Wu, S. Model adaptation: Unsupervised domain adaptation without source data. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (2020), pp. 9641–9650.

[37] Liang, J., Hu, D., and Feng, J. Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation. In *International Conference on Machine Learning* (2020), PMLR, pp. 6028–6039.

[38] Liu, X., Yoo, C., Xing, F., Oh, H., El Fakhri, G., Kang, J.-W., Woo, J., et al. Deep unsupervised domain adaptation: A review of recent advances and perspectives. *APSIPA Transactions on Signal and Information Processing 11*, 1 (2022).

[39] Long, M., Cao, Y., Wang, J., and Jordan, M. Learning transferable features with deep adaptation networks. In *International conference on machine learning* (2015), PMLR, pp. 97–105.

[40] Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., and Zhang, G. Learning under concept drift: A review. *IEEE transactions on knowledge and data engineering 31*, 12 (2018), 2346–2363.

[41] Mao, H., Du, L., Zheng, Y., Fu, Q., Li, Z., Chen, X., Han, S., and Zhang, D. Source free unsupervised graph domain adaptation. *arXiv preprint arXiv:2112.00955* (2021).

[42] Mayr, A., Klambauer, G., Unterthiner, T., Steijaert, M., Wegner, J. K., Ceulemans, H., Clevert, D.-A., and Hochreiter, S. Large-scale comparison of machine learning methods for drug target prediction on chembl. *Chemical science 9*, 24 (2018), 5441–5451.

[43] Müller, R., Kornblith, S., and Hinton, G. E. When does label smoothing help? *Advances in neural information processing systems 32* (2019).

[44] Naderi, H., Soleimani, B. H., Matwin, S., Araabi, B. N., and Soltanian-Zadeh, H. Fusing iris, palmprint and fingerprint in a multi-biometric recognition system. In *2016 13th Conference on Computer and Robot Vision (CRV)* (2016), IEEE, pp. 327–334.

[45] Noh, H., Hong, S., and Han, B. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision* (2015), pp. 1520–1528.

[46] Noriega, L. Multilayer perceptron tutorial. *School of Computing. Staffordshire University 4* (2005), 5.

[47] Pan, S. J., and Yang, Q. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering 22*, 10 (2010), 1345–1359.

[48] Peng, X., Usman, B., Kaushik, N., Hoffman, J., Wang, D., and Saenko, K. Visda: The visual domain adaptation challenge. *arXiv preprint arXiv:1710.06924* (2017).

[49] Pooch, E. H., Ballester, P. L., and Barros, R. C. Can we trust deep learning models diagnosis? the impact of domain shift in chest radiograph classification. *arXiv preprint arXiv:1909.01940* (2019).

[50] Qiu, Z., Zhang, Y., Lin, H., Niu, S., Liu, Y., Du, Q., and Tan, M. Source-free domain adaptation via avatar prototype generation and adaptation. *arXiv preprint arXiv:2106.15326* (2021).

[51] Quinonero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. D. *Dataset shift in machine learning*. Mit Press, 2008.

[52] Rawat, W., and Wang, Z. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation 29*, 9 (2017), 2352–2449.

[53] Redko, I., Morvant, E., Habrard, A., Sebban, M., and Bennani, Y. A survey on domain adaptation theory: learning bounds and theoretical guarantees. *arXiv preprint arXiv:2004.11829* (2020).

[54] Rogers, D., and Hahn, M. Extended-connectivity fingerprints. *Journal of chemical information and modeling 50*, 5 (2010), 742–754.

[55] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International journal of computer vision 115* (2015), 211–252.

[56] Saenko, K., Kulis, B., Fritz, M., and Darrell, T. Adapting visual category models to new domains. In *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11* (2010), Springer, pp. 213–226.

[57] Sarker, I. H. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science 2*, 3 (2021), 1–21.

[58] Shao, J., Hu, K., Wang, C., Xue, X., and Raj, B. Is normalization indispensable for training deep neural network? *Advances in Neural Information Processing Systems 33* (2020), 13434–13444.

[59] Stacke, K., Eilertsen, G., Unger, J., and Lundström, C. A closer look at domain shift for deep learning in histopathology. *arXiv preprint arXiv:1909.11575* (2019).

[60] Sun, Z., Ozay, M., and Okatani, T. Design of kernels in convolutional neural networks for image classification. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VII 14* (2016), Springer, pp. 51–66.

[61] Tian, Y., Henaff, O. J., and van den Oord, A. Divide and contrast: Self-supervised learning from uncurated data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2021), pp. 10063–10074.

[62] Van der Maaten, L., and Hinton, G. Visualizing data using t-sne. *Journal of machine learning research 9*, 11 (2008).

[63] Venkateswara, H., Eusebio, J., Chakraborty, S., and Panchanathan, S. Deep hashing network for unsupervised domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 5018–5027.

[64] Voulodimos, A., Doulamis, N., Doulamis, A., Protopapadakis, E., et al. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience 2018* (2018).

[65] Wang, M., and Deng, W. Deep visual domain adaptation: A survey. *Neurocomputing 312* (2018), 135–153.

[66] Wang, Q., Ma, Y., Zhao, K., and Tian, Y. A comprehensive survey of loss functions in machine learning. *Annals of Data Science* (2020), 1–26.

[67] Weininger, D. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences 28*, 1 (1988), 31–36.

[68] Woodard, D. L., Pundlik, S. J., Lyle, J. R., and Miller, P. E. Periocular region appearance cues for biometric identification. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Workshops* (2010), IEEE, pp. 162–169.

[69] Wu, M., Pan, S., Zhou, C., Chang, X., and Zhu, X. Unsupervised domain adaptive graph convolutional networks. In *Proceedings of The Web Conference 2020* (2020), pp. 1457–1467.

[70] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems 32*, 1 (2020), 4–24.

[71] Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., Leswing, K., and Pande, V. Moleculenet: a benchmark for molecular machine learning. *Chemical science 9*, 2 (2018), 513–530.

[72] Wythoff, B. J. Backpropagation neural networks: a tutorial. *Chemometrics and Intelligent Laboratory Systems 18*, 2 (1993), 115–155.

[73] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[74] Yang, S., van de Weijer, J., Herranz, L., Jui, S., et al. Exploiting the intrinsic neighborhood structure for source-free domain adaptation. *Advances in neural information processing systems 34* (2021), 29393–29405.

[75] Yang, S., Wang, Y., Van De Weijer, J., Herranz, L., and Jui, S. Generalized source-free domain adaptation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2021), pp. 8978–8987.

[76] Yang, S., Wang, Y., Wang, K., Jui, S., et al. Attracting and dispersing: A simple approach for source-free domain adaptation. In *Advances in Neural Information Processing Systems* (2022).

[77] Yi, H.-C., You, Z.-H., Huang, D.-S., and Kwoh, C. K. Graph representation learning in bioinformatics: trends, methods and applications. *Briefings in Bioinformatics 23*, 1 (2022), bbab340.

[78] Zhang, S., Tong, H., Xu, J., and Maciejewski, R. Graph convolutional networks: a comprehensive review. *Computational Social Networks 6*, 1 (2019), 1–23.

[79] Zhang, Y., Liu, T., Long, M., and Jordan, M. Bridging theory and algorithm for domain adaptation. In *International conference on machine learning* (2019), PMLR, pp. 7404–7413.

[80] Zhang, Y., Song, G., Du, L., Yang, S., and Jin, Y. Dane: Domain adaptive network embedding. *arXiv preprint arXiv:1906.00684* (2019).

[81] Zhou, K., Liu, Z., Qiao, Y., Xiang, T., and Loy, C. C. Domain generalization: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022).

Appendices

Appendix A

Experimental Code Examples

```
# Alters PygGraphPropPredDataset.get_idx_split() method to work with
# new defined splitting types
def get_idx_split(self, split_type=None):
    if split_type is None:
        split_type = self.meta_info['split']

    path = osp.join(self.root, 'split', split_type)

    # short-cut if split_dict.pt exists
    if os.path.isfile(os.path.join(path, 'split_dict.pt')):
        return torch.load(os.path.join(path, 'split_dict.pt'))

    if split_type == 'cluster':
        train_src_idx = pd.read_csv(osp.join(path, 'train_src.csv.gz'), compression='gzip', header=None).values.T[0]
        train_tar_idx = pd.read_csv(osp.join(path, 'train_tar.csv.gz'), compression='gzip', header=None).values.T[0]
        test_tar_idx = pd.read_csv(osp.join(path, 'test_tar.csv.gz'), compression='gzip', header=None).values.T[0]

        return {'train_src': torch.tensor(train_src_idx, dtype=torch.long),
                'train_tar': torch.tensor(train_tar_idx, dtype=torch.long),
                'test_tar': torch.tensor(test_tar_idx, dtype=torch.long)}
    elif split_type == 'species_adaptation':
        train_src_idx = pd.read_csv(osp.join(path, 'train.csv.gz'), compression='gzip', header=None).values.T[0]
        train_tar_idx = pd.read_csv(osp.join(path, 'test.csv.gz'), compression='gzip', header=None).values.T[0]

        return {'train_src': torch.tensor(train_src_idx, dtype=torch.long),
                'train_tar': torch.tensor(train_tar_idx, dtype=torch.long)}
    else:
        train_idx = pd.read_csv(osp.join(path, 'train.csv.gz'), compression='gzip', header=None).values.T[0]
        valid_idx = pd.read_csv(osp.join(path, 'valid.csv.gz'), compression='gzip', header=None).values.T[0]
        test_idx = pd.read_csv(osp.join(path, 'test.csv.gz'), compression='gzip', header=None).values.T[0]

        return {'train': torch.tensor(train_idx, dtype=torch.long),
                'valid': torch.tensor(valid_idx, dtype=torch.long),
                'test': torch.tensor(test_idx, dtype=torch.long)}

```

Figure A.1: Modified PygGraphPropPredDataset get_idx_split() method.

```python
# Alters InMemoryDataset.get() method to return indexes of data
def get(self, idx: int) -> Data:
    if self.len() == 1:
        return copy.copy(self.data), idx

    if not hasattr(self, '_data_list') or self._data_list is None:
        self._data_list = self.len() * [None]
    elif self._data_list[idx] is not None:
        return copy.copy(self._data_list[idx]), idx

    data = separate(
        cls=self.data.__class__,
        batch=self.data,
        idx=idx,
        slice_dict=self.slices,
        decrement=False,
    )

    self._data_list[idx] = copy.copy(data)

    return data, idx
```

Figure A.2: Modified InMemoryDataset get() method.

```
55      def _loss(self, softmax_output, near_softmax_output):
56          softmax_output_un = softmax_output.unsqueeze(1).expand(-1, self.k, -1)
57
58          # equivalent to mean of a dot product between softmax predictions for samples in the batch
59          # and softmax predictions of their nearest neighbours
60          # objective is to maximise mean value of this dot product which corresponds to achieving more similar
61          # predictions between the nearest neighbours (prediction consistency)
62          loss = -torch.mean(torch.einsum('ijk,ijk->ij', softmax_output_un, near_softmax_output).sum(-1))
63          first_term = loss.item()
64
65          mask = torch.ones(softmax_output.shape[0], softmax_output.shape[0])  # batch x batch
66          mask.fill_diagonal_(0)  # matrix of ones with zeros on a diagonal
67          copy = softmax_output.T
68          dot_neg = softmax_output @ copy  # dot product of softmax output with itself transposed
69                                           # i.e. the second term of the loss equation done on all
70                                           # features from the batch at once and before summation applied.
71                                           # Cells in the dot_neg matrix are all possible
72                                           # dot products between predictions for each feature in a batch
73                                           # and all the remaining features in a batch. Diagonal cells are
74                                           # dot products of predictions for the same feature.
75          dot_neg = (dot_neg * mask.cuda()).sum(-1)  # summation of dot products, and we ignore
76                                                     # elements on the diagonal, hence multiplication by the mask
77          neg_pred = torch.mean(dot_neg)
78          second_term = neg_pred.item() * self.alpha
79
80          loss = loss + neg_pred * self.alpha
81
82          return loss, first_term, second_term
```

Figure A.3: Method from the Attracting and Dispersing Python class implementing the AaD loss function.