Domain Adaptation for Remote Sensing Using Deep Learning

by

Ahmed Elshamli

A Thesis
presented to
the University of Guelph

In partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in
Engineering

Guelph, Ontario, Canada

© Ahmed Elshamli, January, 2020
ABSTRACT

DOMAIN ADAPTATION FOR REMOTE SENSING USING DEEP LEARNING

Ahmed Elshamli
University of Guelph, 2020

Advisor:
Professor Shawki Areibi,
Professor Graham Taylor

Traditional machine learning (ML) techniques are often employed to perform complex pattern recognition tasks for remote sensing images, such as land-use classification. In order to obtain acceptable classification results, these techniques require sufficient training data to be available for every particular image. Obtaining training samples is challenging, particularly for near real-time applications. Therefore, past knowledge must be utilized to overcome the lack of training data in the current regime. This challenge is known as domain adaptation (DA), in which the training data (source) and the test data (target) are sampled from different domains.

DA is a challenging problem and the performance of the proposed techniques can be significantly affected by the type of problem, the nature of the data, and the type of data shift associated with the domains. Even though more data can be obtained from different sources, adapting these sources to obtain acceptable results is also a challenging task. This is especially true when the different domains contain a severely imbalanced class distribution.

In this study, different techniques based on deep neural networks (DNNs) were developed and evaluated to solve the DA problem for remote sensing image classification in different settings. First, the single-source DA problem was addressed by finding invariant representations for both the source and the target. Denoising autoencoders (DAE) and domain-adversarial neural networks (DANN) were adopted to find these invariant representations. Results showed that both techniques were able to outperform traditional approaches, such as principal component analysis (PCA) and kernel PCA. Second, the multi-source domain adaptation for large-scale applications was addressed. A novel, efficient, scalable, yet simple, adaptive multi-source domain adaptation (AMDA) was developed to address this problem. AMDA was also capable of dealing more effectively with the imbalanced data distribution among the sources. Two techniques originally proposed for domain expansion were also extended to the task of multi-source domain adaptation. AMDA and the extended domain expansion techniques were implemented and evaluated on the LCZ classification problem. Despite its simplicity, AMDA was able to achieve more than a 12% improvement over the baseline.
Acknowledgements

First and foremost, I thank Allah Almighty for giving me the ability and strength to complete this task successfully.

I would like to express my deepest gratitude to my mother, my father, and my wife for their persistent support, encouragement, patience, and confidence in me.

I would like to express my gratitude and special appreciation to my supervisors Dr. Shawki Areibi and Dr. Graham Taylor for their help and support during my work and for the guidance they provided me while carrying out this research. The opportunity to work with them has been extremely rewarding. I would like also to thank Dr. Gary Grewal for his valuable discussions and feedback that greatly helped me enhance the quality of my work and my thesis. I would like to also thank the faculty and staff in the School of Engineering for their support and help, and to my colleagues in the ISLAB and MLRG and my friends at the University of Guelph for their encouragement, support and the great time we spent together.

Special thanks to my brothers and sister, I am grateful for all the support, encouragement, and sacrifices you made for me. Finally, I would like to thank both the Canadian and the Libyan governments for their financial support.
To

my mother, father, wife, children, brothers, and sister

whose love and encouragement helped accomplish this

thesis.
List of Publications

• Journal Papers:

   A. Elshamli, G. W. Taylor, A. Berg, and S. Areibi,

   Elshamli, A., Taylor, G., & Areibi, S.
   IEEE Transactions on Geoscience and Remote Sensing (accepted).

• Conference Papers:

1. “A simple and effective semi-supervised learning framework for hyperspectral image classification”
   A. Elshamli, G. W. Taylor, and S. Areibi,

2. ”A Deep Learning Framework to Predict Routability for FPGA Circuit Placement”
   A. Al-Hyari, A. ELshamli, Z. Abouwaimer, G. Grewal, S. Areibi,
## Contents

Abstract .............................................................. ii
Acknowledgements ............................................... iii
List of Publications ............................................. v
Table of Contents ............................................... vi
List of Tables .................................................... x
List of Figures .................................................... xii
List of Abbreviations ........................................... xiv

1 Introduction .................................................. 1
   1.1 Domain Adaptation ...................................... 3
   1.2 Motivations and Challenges ............................ 4
   1.3 Thesis Statement ........................................ 5
   1.4 Thesis Objective ........................................ 5
   1.5 Research Contributions ................................. 6
   1.6 Thesis Organization ................................... 7

2 Background .................................................. 8
   2.1 Remote Sensing .......................................... 8
      2.1.1 Characteristics of Remote Sensing Images .... 11
   2.2 Machine Learning ....................................... 13
      2.2.1 Supervised Learning .............................. 14
### CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.2</td>
<td>Unsupervised Learning</td>
<td>17</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Semi-supervised Learning</td>
<td>17</td>
</tr>
<tr>
<td>2.2.4</td>
<td>Feature Extraction and Dimensionality Reduction</td>
<td>17</td>
</tr>
<tr>
<td>2.2.5</td>
<td>Representation Learning</td>
<td>19</td>
</tr>
<tr>
<td>2.3</td>
<td>Deep Neural Networks (DNNs)</td>
<td>20</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Neural Network Training Procedure</td>
<td>20</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Representation Learning using DNNs</td>
<td>23</td>
</tr>
<tr>
<td>2.4</td>
<td>Domain Adaptation</td>
<td>28</td>
</tr>
<tr>
<td>2.4.1</td>
<td>DA Formulation and Notations</td>
<td>28</td>
</tr>
<tr>
<td>2.4.2</td>
<td>DA Theory</td>
<td>31</td>
</tr>
<tr>
<td>2.5</td>
<td>Remote Sensing Image Classification Challenges</td>
<td>32</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Pixel-based Task</td>
<td>33</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Image Quality</td>
<td>34</td>
</tr>
<tr>
<td>2.5.3</td>
<td>Spectral Mixture</td>
<td>35</td>
</tr>
<tr>
<td>2.5.4</td>
<td>Data Labeling</td>
<td>35</td>
</tr>
<tr>
<td>2.6</td>
<td>Summary</td>
<td>36</td>
</tr>
</tbody>
</table>

### 3 Literature Review

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Domain Adaptation Categories</td>
<td>37</td>
</tr>
<tr>
<td>3.2</td>
<td>Domain Adaptation in Remote Sensing</td>
<td>38</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Selection of Invariant Features</td>
<td>38</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Representation Learning</td>
<td>39</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Adaptation of Classifiers</td>
<td>40</td>
</tr>
<tr>
<td>3.3</td>
<td>Deep Learning Techniques For Domain Adaptation</td>
<td>42</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Autoencoder-based Approaches</td>
<td>44</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Distribution Matching Approaches</td>
<td>48</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Adversarial Training Approaches</td>
<td>51</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Multi-source Domain Adaptation</td>
<td>55</td>
</tr>
</tbody>
</table>
3.3.5 Domain Expansion .................................................. 57
3.4 Datasets and Benchmarks ............................................. 58
3.4.1 Remote Sensing Domain Adaptation Benchmarks ................. 58
3.4.2 Deep Learning Domain Adaptation Benchmarks .................. 60
3.5 Summary ................................................................. 66

4 Single Source Domain Adaptation ...................................... 67
4.1 Introduction ............................................................. 67
4.2 Domain Adaptation Paradigms ........................................ 68
4.2.1 Two Stage DA ...................................................... 69
4.2.2 DANN: End-to-End DA ........................................... 70
4.3 Experimental Setup .................................................... 71
4.3.1 Baselines and DA Settings ....................................... 72
4.3.2 Origin of Samples Used for DA .................................... 72
4.3.3 DAE and DANN Architecture and Hyper-parameters .......... 73
4.3.4 Classification task .................................................. 74
4.4 Datasets ................................................................. 75
4.4.1 Hyperspectral Images .............................................. 75
4.4.2 Multispectral Images .............................................. 76
4.5 Results ................................................................. 79
4.5.1 HSI Images .......................................................... 80
4.5.2 Multispectral Images: Scenario 1 (Spatial Adaptation) ........... 80
4.5.3 Multispectral Images: Scenario 2 (Temporal Adaptation) ........ 81
4.5.4 Computation Time .................................................. 83
4.6 Discussion .............................................................. 84
4.6.1 Challenges Encountered For Domain Adaptation .................. 84
4.6.2 Availability of Samples From The Target Domain ............... 86
4.6.3 Scalability .......................................................... 86
5 Multi-source Domain Adaptation using DNNs

5.1 Introduction

5.2 Methodology

5.2.1 Architecture

5.2.2 Adaptation by Knowledge Expansion Networks

5.2.3 Adaptive Multi-source DA (AMDA)

5.3 Experimental setup

5.3.1 Dataset

5.3.2 Data Pre-processing

5.3.3 Pixel-based to Patch-based Prediction

5.3.4 Baseline Network Architecture and Hyper-parameters

5.3.5 Network Training

5.4 Results and discussion

5.4.1 Single-source Single-target Scenario

5.4.2 Multi-source DA Results

5.4.3 Network Complexity and Scalability

5.5 Summary

6 Conclusions and Future Work

6.1 Conclusions

6.2 Lessons Learned

6.3 Future Work
List of Tables

2.1 Characteristics of remote sensing satellites currently orbiting the Earth. . . . . . . 13
2.2 Basic characteristics for commonly used supervised algorithms. . . . . . . . . . . 16
2.3 Notations used in this thesis. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
2.4 Taxonomy of learning types. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
2.5 Classical vs. RS image classification tasks. . . . . . . . . . . . . . . . . . . . . . . 32
3.1 Characteristics of the DA deep learning techniques. . . . . . . . . . . . . . . . . . 43
3.2 LCZs data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
3.3 Digit recognition DA classification accuracy . . . . . . . . . . . . . . . . . . . . . 66
4.1 Representation learning and classification settings. . . . . . . . . . . . . . . . . . 72
4.2 Classes in the Pavia city dataset. . . . . . . . . . . . . . . . . . . . . . . . . . . . 75
4.3 Classes used for Landsat dataset. . . . . . . . . . . . . . . . . . . . . . . . . . . . 79
4.4 DA classification results for the Pavia dataset. . . . . . . . . . . . . . . . . . . . . 81
4.5 Crop mapping classification results from June. . . . . . . . . . . . . . . . . . . . . 81
4.6 Crop mapping classification results from July. . . . . . . . . . . . . . . . . . . . . 82
4.7 Crop mapping multi-temporal classification. . . . . . . . . . . . . . . . . . . . . . 82
4.8 Training and mapping time of the existing and proposed techniques. . . . . . . . . 84
4.9 User and producer accuracy. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 85
5.1 LCZs data - DFC2017 provided ground-truth (# of pixels). . . . . . . . . . . . . . 100
5.2 LCZs classes considered in this experiment (# of patches). . . . . . . . . . . . . . 103
5.3 Baseline network architecture and hyper-parameters. ................. 109
5.4 Comparison between the training procedure of the proposed methods. .... 110
5.5 Single-source single-target baseline network accuracy ...................... 112
5.6 LCZ classification accuracy results ......................................... 113
5.7 Network complexity and training time. ...................................... 114
5.8 LCZ map generation time ..................................................... 115
List of Figures

2.1 Remote Sensing - Basic concepts. .............................. 9
2.2 Remote Sensing - Acquisition process. .......................... 10
2.3 Electromagnetic spectrum ........................................ 11
2.4 Machine learning and knowledge discovery. ....................... 13
2.5 Artificial neural network. .......................................... 21
2.6 Basic autoencoder. ..................................................... 23
2.7 Convolution Neural Network - Basic blocks. ...................... 25
2.8 Numerical operations in the convolution and pooling layers. ........ 26
2.9 Commonly used techniques to prepare RS data for ML. .......... 33

3.1 Representation learning using an autoencoder. ................... 44
3.2 Deep Reconstruction-Classification Networks ...................... 46
3.3 Domain Separation Networks ....................................... 51
3.4 Domain Adversarial Neural Network ................................ 52
3.5 ADDA framework. ...................................................... 54
3.6 Generate-to-adapt (G2A) architecture. .............................. 56
3.7 Pavia City: DA Scenario ............................................. 58
3.8 Pavia University: DA Scenario ....................................... 59
3.9 Indian Pine: DA Scenario ............................................. 59
3.10 Examples of the UC Merced Aerial data set. ....................... 60
LIST OF FIGURES

3.11 Berlin LCZ training data. ......................................................... 61
3.12 Paris LCZ training data. ......................................................... 61
3.13 Rome LCZ training data. ......................................................... 62
3.14 Hong Kong LCZ training data. ............................................... 62
3.15 Sao Paulo LCZ training data. .................................................. 62
3.16 Samples from Office-31 dataset. ............................................ 63
3.17 Office-31 DA results. ............................................................. 64
3.18 Examples of the digit recognition datasets. .............................. 64

4.1 Domain adaptation paradigms considered in this study. ................. 69
4.2 Pavia City: DA Scenario .......................................................... 76
4.3 Landsat 4-5TM images. ........................................................... 77
4.4 Data used in the multi-spectral experiment. ................................ 78
4.5 Source and target domains in the Landsat images ...................... 79

5.1 The proposed multi-source domain adaptation framework for LCZs mapping. 89
5.2 Our baseline CNN architecture. ................................................. 90
5.3 Learning without forgetting. ..................................................... 92
5.4 Less-forgetful Learning. .......................................................... 94
5.5 AMDA network architecture ..................................................... 98
5.6 Paris Landsat 8 scene and the provided LCZ map. ..................... 101
5.7 Patch generation process ........................................................ 102
5.8 Sample patches from Berlin. .................................................... 104
5.9 Sample patches from Paris. ..................................................... 105
5.10 Sample patches from Rome. ................................................... 106
5.11 Sample patches from Hong Kong. .......................................... 107
5.12 Sample patches from Sao Paulo .............................................. 108

6.1 Autoencoder approach for aligning the multi-sensor feature space .... 122
List of Abbreviations

RS : Remote Sensing
HSI : HyperSpectral Image
MSI : MultiSpectral Image
ML : Machine Learning
ANN : Artificial Neural Network
DNN : Deep Neural Network
CNN : Convolution Neural Network
AE : Autoencoder
DAE : Denosing Autoencoder
DA : Domain Adaptation
UDA : Unsupervised Domain Adaptation
MDA : Multi-source Domain Adaptation
LwF : Learning without forgetting
LFL : Less-forgetful Learning
AMDA : Adaptive Multi-source Domain Adaptation
Chapter 1

Introduction

Remote sensing (RS) is the science and technology of obtaining information about objects from a distance, typically from a satellite, aircraft or unmanned aerial vehicle. This technology enables the collection of valuable data that was not possible to collect in the past. For example, Precision Agriculture is an emerging farming management concept based on observing, measuring and responding to the field variability in crops and yields [1]. For such an application to be successful and affordable, a comprehensive field survey must be conducted at different times to observe crop growth and health status during the planting season. Traditional techniques usually achieve this by being physically on the field and measuring the required attributes and field parameters such as plant height, crop conditions, and soil properties. This method is tedious and expensive since it involves collecting a large number of samples from the field. Furthermore, it is not feasible in many cases due to the large areas that are involved and the accessibility issues for collecting such data. Remote sensing can overcome these issues and help in raising the precision agriculture efficiency and success to a higher level.

As technology improves and evolves, RS data becomes richer, and the resolution of the data becomes higher in all domains: spectral, spatial, and temporal. For example, Landsat 8, a NASA and USGS multispectral instrument launched in 2013, orbits the entire Earth every 16 days. Landsat 8 multi-spectral image (MSI) contains 11 spectral bands, including the visible/near-infrared
(VNIR) bands, short-wave infrared (SWIR) bands, and thermal infrared bands. Each scene covers about 33,000 km$^2$ with a ground resolution of 30 m and is publicly available as a compressed geo-referenced Tiff file (file size: $\approx 1$ GByte) [2]. There are a total of about 500 images acquired every day. Another example is SENTINEL-2, launched in 2015, with 13 spectral channels in the VNIR and SWIR bands. SENTINEL-2 has a higher spatial and temporal resolution than Landsat 8 (10 m ground resolution and a 5 day revisit frequency). The enormous future opportunities provided by such a great wealth of data in these images come also with challenges on how to analyze and extract important patterns and knowledge from these images in a timely fashion.

Classification is one of the important tasks that has been extensively studied within the Machine Learning (ML) community, and several novel algorithms and techniques have been developed. These algorithms have been successfully applied to many real-life applications, such as banking, medicine, and remote sensing [3, 4]. The aim of this task is to categorize or classify an input into one of the predefined classes. Classifying pixels from a scene of a crop field into different classes that reflect the plant growing conditions, or classifying an image of a person based on emotions are examples of these classification tasks.

The classification task is a supervised ML technique aimed at learning a mapping function (e.g., classifier) that maps every input to a correct group or class. Prior to training, this technique must have access to training examples that can be used to model the relationship between the input and the output. In order for the ML technique to be able to learn a proper mapping function, these training examples must be correctly annotated (or labelled) with the correct class label. Furthermore, the number of correctly annotated examples must be sufficient to capture the variability in the inputs. Li et al. [5] have showed in one case study that ML algorithms require 60 samples per class$^1$. The availability of high-quality and high-quantity training sets has made it possible for the state-of-the-art machine learning techniques to compete with human performance in several classification tasks [6, 7]. Nevertheless, human involvement in creating these training sets are still the

---

$^1$Generally, it is hard to quantify sufficient in this context, as it depends on the characteristics of the problem, such as the size of the input features, number of classes, etc.
most time-consuming operation in this supervised task.

In RS, the training set is created either by physically surveying the area of interest or by image interpolation. During this labelling process, a subset of the pixels in the area of interest must be carefully and correctly annotated by the user to reflect the categories/classes considered in this task. This manual work is costly and time-consuming, and this process must be repeated for every new image. Thus, another efficient and more productive labelling method must be developed to overcome this obstacle. Overcoming this obstacle will allow for more utilization of the RS images that are being captured on a daily basis, such as those from Landsat 8 and SENTINEL-2, which in turn will help in realizing enormous applications related to environmental science and geoscience that were neither possible nor practical in the past. One of the promising methods that has been proposed in the literature is transfer learning, which aims to employ the knowledge learned from previous tasks to help in solving new tasks.

1.1 Domain Adaptation

Domain Adaptation (DA) is one of the transfer learning techniques, in which the system aims to adapt the knowledge learned from one domain, called the source domain, and apply it to another related domain, called the target domain. The main challenge of the DA problem is that a significant variation exists between the source and the target data distribution; therefore, traditional supervised models trained using the source data without any adaptation will more likely fail in the target domain. Emerging applications usually follow the DA setting, which demands the use of other related domain annotation data. Multi-source DA (MDA) is another form of DA, which implies the existence of more than one source domain. The MDA problem is more relevant to RS applications. Consider, for example, the 500 new scenes provided by Landsat 8 on a daily basis. This enormous amount of data cannot be fully utilized in a timely fashion unless the labelling process is minimized or better yet, eliminated. Earth’s atmospheric conditions and seasonal changes, however, result in a significant data shift between the scenes. In fact, each scene can be considered
as a new domain, and applying DA is necessary to obtain high-quality results for the new scenes.

1.2 Motivations and Challenges

Classification is one of the essential analysis tasks in remote sensing. Completing this task efficiently will open doors for different kinds of applications, such as precision agriculture and early warnings regarding wildfires in forests. These applications can then be carried out effectively in terms of time and costs. The opportunities provided by the current earth monitoring systems, such as Landsat 8, which provide open and free access to an enormous amount of data on a daily basis, have not been utilized to a full extent. Efficient and robust analysis techniques must thus be developed for greater utilization of this stream of images. The ability to analyze these images in a timely fashion, will have a great impact on society, since it will be a useful tool for better understanding our planet, and also avoiding or at least minimizing the damages of sudden natural disasters.

In general, the classification task is difficult in terms of the time and the computational power demanded. The difficulty of this task is more challenging for RS images, compared with typical machine vision classification tasks, due to the following reasons:

- The cost of acquiring sufficient training data to obtain acceptable results is very high compared with traditional object or scene recognition.

- Compared to standard object recognition and image segmentation, the labelling process is not obvious for most RS applications.

- Unlike the traditional image classification task, which is an image based task with each image belonging to a specific class, the RS classification task is a pixel-based task.

- Significant data shift exists between the images captured at different times or from different locations.

- In addition to the input data shift, class imbalance also exists in RS classification tasks. Class imbalance is more apparent when the training data originates from multiple sources and the
CHAPTER 1. INTRODUCTION

target is from a new unseen source.

- Compared to regular images, most publicly free RS images are low resolution (e.g. > 10 m).
  At such low resolutions, object shapes (e.g., land-cover classes) are not clearly visible; thus, the discrimination task is difficult even for human experts.

1.3 Thesis Statement

Challenges introduced in the previous section have motivated this work to develop a scalable\(^2\) classification framework that is able to deal efficiently with the data variability and uncertainty in remote sensing. In supervised classification tasks, it is assumed that the training and test data represent the same domain (i.e., the distribution of the training data and the test data is similar). If this assumption does not hold, then domain shift exists, and applying classical supervised training will likely result in poor performance. The domain shift occurs when the sampling configuration and/or the sampling conditions have changed. The domain shift problem is apparent in several fields such as natural language processing, finance, medical, and remote sensing. In fact, given the nature of the image acquisition process in remote sensing, the domain shift problem is more apparent and severe. Therefore, we may consider each remote sensing scene as a unique domain. To this end, our problem statement can be summarized as: “How to efficiently and effectively perform the remote sensing pixel classification task under domain shift?”.

1.4 Thesis Objective

The objective of this thesis is to investigate, implement, and analyze domain adaptation frameworks based on deep learning for remote sensing pixel classification. Deep learning has proven to be an effective approach in handling large-scale and very complicated learning tasks. Deep learning also offers a great deal of flexibility in developing new algorithms for solving challenging problems that

\(^2\)In this context, scalable means that the model still perform well even when the problem size or complexity significantly increase.
other traditional machine learning methods can only solve with very complicated, and sometimes “impractical”, modeling [8].

1.5 Research Contributions

The main contributions of this thesis can be categorized into two major paradigms:

1. **Representation Learning Based on Deep Learning**: Two unsupervised representation learning algorithms based on deep learning are presented in Chapter 4:
   - A representation learning algorithm was proposed for RS images based on denoising autoencoders (DAE). Unlike commonly used unsupervised representation learning techniques in RS, DAE can handle large-scale DA problems. DAE is also more robust, therefore, it can better reduce the data discrepancy between the domains.
   - A domain adversarial neural network (DANN) [9] was proposed to the RS community as an end-to-end framework for solving DA problems. DANN is trained to satisfy a dual objective problem: to produce an invariant representation for both domains, and to be able to discriminate between the provided labelled samples.
   - The advantage of the representations learned from DAE and DANN were investigated by evaluating them across different single source DA scenarios and comparing their performance with commonly used techniques in RS.

2. **Multi-Source Domain Adaptation** Three MDA algorithms were proposed in Chapter 5. These algorithms could handle large-scale MDA problems efficiently and could also reduce the effect of the class imbalance among the sources by:
   - proposing a simple, yet effective DNN framework to deal with the large number of samples in remote sensing more efficiently.
CHAPTER 1. INTRODUCTION

- proposing a novel end-to-end Adaptive Multi-source Domain Adaptation (AMDA) technique that is simple, robust, and capable of handling imbalanced large-scale multi-source domain adaptation problems.
- extending two recently introduced domain expansion algorithms for multi-source DA.
- implementing and empirically evaluating the proposed techniques on a large-scale public remote sensing benchmark: the LCZ classification problem.

1.6 Thesis Organization

The remainder of the thesis is organized in the following manner. Chapter 2 provides essential background information on remote sensing. Fundamental machine learning and deep learning concepts are also provided in Chapter 2 along with the formulation and definition of the domain adaptation problem. In Chapter 3, present review to the related domain adaptation techniques proposed in the literature. The proposed single source domain adaptation techniques, the two-stage denoising autoencoder, and the end-to-end domain adversarial neural network are introduced in Chapter 4. Chapter 5 presents the proposed multi-source domain adaptation algorithms. Finally, the conclusions and potential future works are presented in Chapter 6.
Chapter 2

Background

This chapter provides the reader with essential background material that covers the different topics related to this research. The chapter is organized as follows: Section 2.1 covers the fundamental aspects of remote sensing (RS) technology and Section 2.2 covers machine learning (ML) related topics. Section 2.4 defines domain adaptation, formulates the problem, and provides theoretical background. Finally, Section 2.5 highlights the challenges related to the remote sensing classification task.

2.1 Remote Sensing

Remote sensing (RS) is the technology of measuring certain physical properties of an object without making physical contact with that object. Humans perform RS using their visual system (e.g. eyes), with which they can estimate many physical properties of surrounding objects without being in direct contact with them. Bats perform RS in the dark by generating high-pitched sonar and listening to the sound echoes. Remote sensing is based on the basic concept that the amount of energy reflected from any given material varies with the wavelength of the signal. In an ideal situation, the identity of the material or substance can therefore be identified directly from the reflected response with respect to the wavelength.

In the field of Earth observation, RS refers to obtaining information about the Earth using
CHAPTER 2. BACKGROUND

a satellite, an aircraft, or even an unmanned aerial vehicle (UAV). The information obtained is the measurement of the electromagnetic signals reflected from the Earth’s surface using a sensor attached to the aerial platform. Recent advances in sensing technology have allowed for such an application to exist and allowed for the measurement of these electromagnetic signals at different wavelengths. Sensors are classified either as passive or active based on the source of the measured energy. Passive sensors measure the natural energy levels which surround existing sources, such as the reflection of sunlight or the energy generated from the Earth itself. On the other hand, active sensors measure the reflected energy generated by a source attached to the sensor. This can be related to the initial examples of humans and bats, in which the bat RS is active, and the human RS is passive.

Figure 2.1 shows the complete process of obtaining RS images. The sensing platform (airborne/satellite) measures the reflection from the Earth surface. Basic pre-processing for these measurements is then carried out to generate the final image. The obtained image is than represented as an image cube, as illustrated in Figure 2.1

![Figure 2.1: Remote Sensing - Basic concepts.](image)

There are several physical factors control the obtained image. To illustrate this, the acquisition process must be examined in more detail. As illustrated in Figure 2.2, the on-board sensor measures the electromagnetic radiation reflected from a field of view (FOV) at different wavelengths simultaneously. The radiation (or energy) that is reflected is proportional to the surface and the
material that causes this reflection. By utilizing an onboard Global Positioning System (GPS),
the spatial coordinates of each measurement can be associated with the measured electromagnetic
reflections. The ground line covered by the sensor FOV is called the *swath*, and all the pixels in
the swath are captured simultaneously. At a given swath all the ground pixels are captured. As the
sensor moves, the FOV changes and therefore, another swath is covered. Once the image acqui-
sition process is completed, an error correction step is required to correct the geometry errors and
the radiometric errors of the captured pixels [10]. The process is essentially the same whether the
sensor is attached to an aircraft or a satellite. The difference is related to the ground resolution of
the image and the means by which the initial processing of the data was conducted. It should be
noted that RS images are usually processed at the pixel level which is considered the basic data
element. In addition to the spatial attributes that can be easily represented by the pixel index, each
pixel has a set of variables/attributes representing the reflected electromagnetic radiation measured
at each wavelength. These are referred to as the spectral measurements or bands.
CHAPTER 2. BACKGROUND

2.1.1 Characteristics of Remote Sensing Images

Unlike traditional static 2D images, which are usually characterized by the spatial resolution, the RS images are characterized by two resolutions: spatial and spectral. The spectral resolution depends solely on the sensor instrument being used. The number of bands that the sensors can measure instantaneously dictates this resolution. This change in the reflected energy with respect to the wavelengths is also known as the spectral signature.

![Electromagnetic Spectrum](image)

Figure 2.3: Electromagnetic Spectrum.

The number of bands or channels is based on the wavelength resolution used to capture the RS image. Based on this number, the RS images can be divided into two types: multispectral images (MSI), which refer to images with few bands (e.g., a dozen bands), and hyperspectral images (HSI) with a much larger number of bands (e.g., >100 bands). Due to the ability of the HSI sensing device to capture electromagnetic reflections at a wider range and at contiguous spectral wavelengths, the resulting image contains a large number of bands. Figure 2.3 shows the complete electromagnetic spectrum. As shown in this figure, the visible region covers a very small part of the whole spectrum (0.4µm - 0.7µm). The most commonly used spectrum ranges are the visible and the infrared regions (0.4µm - 12µm). These ranges are important since the strength of the energy that is measured relies on the cellular structure of the vegetation, the moisture content, the pigmentation, the mineral contents of the soil, and the level of sedimentation in the water. Furthermore, the strength of the signal measured in the far-infrared (Far-IR) region relies on the thermal properties of the surface that is sensed; consequently this portion of the IR is also known as the thermal-IR [10]. The microwave/radar range is used in active RS, and the measured signal
strength depends mainly on the roughness of the surface. It is therefore commonly used for mineral exploration.

The spatial resolution, also known as the ground resolution, depends mainly on the spatial resolution of the sensing device as well as the altitude of the scanning platforms. The mapping of the acquired RS data to the actual ground reference is a complex process, therefore pre-processing steps dealing with many distortions that occur during image acquisition have to be applied. The source of distortion can be from a range of sources [10]: 1) the Earth’s rotation, 2) the scanner view angle, 3) the wide FOV and the curvature of the Earth, and 4) variations in the altitude, attitude, and velocity of the scanning platform.

Once distortions in geometry are corrected, the image can then be registered to a map coordinate system. This registration process is known as *geo-coding* or *geo-referencing* and each pixel in the image is coded in terms of the map coordinate system. Geo-coding is an important step for many applications so that the acquired image can be precisely referenced to the available maps that are known a priori. Since these maps usually reference ground regions and points of interest, such as buildings, parks, forests, and rivers, another useful layer of information can be added on top of the information gathered by the RS instrument.

Airborne scanners from lower altitudes provide higher spatial resolution than their counterparts that are mounted on platforms at higher altitudes; however, higher altitude platforms, such as satellites, are considered one of the main sources for RS images since they provide a wider FOV and they can provide continuous scanning for the same site on each cycle. Furthermore, the advances in the technology for sensing devices has increased the spatial resolution for near orbit satellites. As of today, there are satellite systems that can acquire RS images with a spatial resolution less than 1m/pixel. An example is WorldView-3, which provides up to 0.31 meter/pixel. Furthermore, such systems can complete the full cycle in about one day\(^1\). Table 2.1 lists examples of the satellites that are currently orbiting the Earth. This enhanced spatial resolution makes such images of great use for most applications. Moreover, this temporal resolution (i.e., one day) that provides

\(^1\)https://www.digitalglobe.com/resources/satellite-information
multiple images of the same area of interest at different times is essential in many fields, such as environmental and climate change.

Table 2.1: Characteristics of remote sensing satellites currently orbiting the Earth.

<table>
<thead>
<tr>
<th>Instrument</th>
<th>Resolution (m) (# bands)</th>
<th>Image size (km)</th>
<th>Re-vist freq. (days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeoEye-1 (2008- Now)</td>
<td>1.8(4) 1.84</td>
<td>100×100</td>
<td>3</td>
</tr>
<tr>
<td>Spot 6 (2012 - Now)</td>
<td>6(4) 1.50</td>
<td>60×60</td>
<td>1</td>
</tr>
<tr>
<td>Landsat 8 (2013 - Now)</td>
<td>30(5) 15 30(2)</td>
<td>170×183</td>
<td>16</td>
</tr>
<tr>
<td>WorldView-3 (2014 - Now)</td>
<td>1.2(8) 0.31 3.7(8)</td>
<td>13.3×56</td>
<td>1</td>
</tr>
<tr>
<td>SENTINEL-2 (2015 - Now)</td>
<td>10(10) 20(3)</td>
<td>100×100</td>
<td>5</td>
</tr>
</tbody>
</table>

2.2 Machine Learning

Machine Learning (ML) is the core component for discovering knowledge and obtaining interesting patterns which are the ultimate goal for any data mining system [11]. As illustrated in Figure 2.4, the process starts within a specific field or industry by defining the problem and making raw data available to the ML algorithms. The raw data which is supplied depends on the problem at hand, and it can be in different formats, such as image, text, audio or sensory reading.
CHAPTER 2. BACKGROUND

This raw data goes through a pre-processing step such as vectorization, data cleaning, normalization, and dimensionality reduction. The pre-processed data is then supplied to the main machine learning algorithm to train/build a model to solve the given problem. The resulting model can then be used as a backbone for knowledge and problem solving by the industry. Based on the learning strategy, ML algorithms can be categorized into supervised, semi-supervised, and unsupervised learning. These categories will be presented and discussed in the subsequent sections.

2.2.1 Supervised Learning

Supervised learning is the family of ML techniques that learn from examples; therefore, these algorithms must have access to examples, called training data. Each example in this dataset consists of two parts, the input \( x \in \mathcal{X} \) and the output \( y \in \mathcal{Y} \), where \( \mathcal{X} \) is the input feature space and \( \mathcal{Y} \) is the label space. The input feature space \( \mathcal{X} \) is the numerical value that represents the input; for example, if the input is an image, then the input feature space is the intensity value of every pixel in the image. The label space \( \mathcal{Y} \) is also a numerical value that provides an accurate (Ground-truth) description for the input. Two groups of supervised algorithms are considered based on the label space: classification and regression. In classification, the label space consists of finite discrete values. They are therefore called class labels. In regression, the label space is a continuous series of real numbers. Differentiating email messages into Spam and not Spam messages is a typical classification problem, while estimating the amount of rainfall is a typical regression problem.

Using this training set, the supervised algorithm aims to learn a model: a mapping function that maps the input to the output (e.g., \( f: x \rightarrow y \)). This model is an estimate of the true unknown model that governs the input/output relationship. The performance of the final model learned by the supervised algorithm depends on the quality and the size of the training set. During training, the supervised technique aims to minimize the mapping error for this training set; however, the actual model performance will be measured against the test set, which contains a set of input instances that were not included in the training set.

One of the main challenges in supervised learning is to develop a model that can “generalize”
both the training and the test data. In this regard, two major concerns are encountered: *under-fitting*, and *over-fitting*. Under-fitting occurs when the model is incapable of capturing the variability of the data. In contrast, over-fitting is encountered when the model fits the training data perfectly, but then produces poor accuracy with the test data. Under-fitting can be easily measured using the training data which is already available; however, over-fitting cannot be measured since the test data is not available during training. Cross-validation is used to estimate the over-fitting by hiding a portion of the training set during the training stage and using it to “validate” the performance of the trained model. This validation set thus acts as a test set. Using these metrics, under-fitting and over-fitting can be avoided by monitoring the model’s performance during the training, and adjusting the training algorithm strategy or hyper-parameters as necessary.

Both the quality and the size of the training set play a key role in supervised learning. Training sets that contain inaccurate labels and small training sets that do not capture the possible diversity of the problem will likely result in a model with an unacceptable performance. Special care and a significant amount of effort must therefore be carried out to obtain a sufficiently high-quality training set. In many tasks, generating or collecting the training data is hard, time-consuming, and sometimes not feasible for certain constraints (e.g., budget, time). For this reason, *Crowdsourcing* is one of the techniques that is being employed nowadays to speed up and reduce the cost of the labelling task. Utilizing the Internet, workers are invited to perform the labelling task using a portal designed specifically for the task. Each worker is given a data instance sample from the data to annotate. Since a considerable number of workers can perform this labelling task simultaneously, the quality of these labels is questionable. To reduce this uncertainty, the annotation process must be designed to be more challenging, and a verification step must be performed as well. ImageNet benchmark [12] is an example of a successful crowdsourcing annotation system, which utilizes the platform provided by Amazon Mechanical Turk\(^2\) to efficiently perform this labelling task.

During the past few decades, several supervised algorithms have been developed. Table 2.2 presents a general comparison chart for the most commonly used supervised algorithms. The com-
CHAPTER 2. BACKGROUND

Table 2.2: Basic characteristics for commonly used supervised algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Type</th>
<th>Accuracy</th>
<th>Speed Train/Predict</th>
<th>Model Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic Regression</td>
<td>Regression</td>
<td>Low</td>
<td>Fast/Fast</td>
<td>Simple</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>Regression</td>
<td>Low</td>
<td>Fast/Fast</td>
<td>Simple</td>
</tr>
<tr>
<td>Support Vector Machines</td>
<td>Classification</td>
<td>High</td>
<td>Moderate/Moderate</td>
<td>Moderate</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>Classification</td>
<td>Moderate</td>
<td>Moderate/Fast</td>
<td>Simple</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>Both</td>
<td>High</td>
<td>Slow/Fast</td>
<td>Complex</td>
</tr>
<tr>
<td>Decision Trees</td>
<td>Both</td>
<td>Moderate</td>
<td>Fast/Fast</td>
<td>Moderate</td>
</tr>
<tr>
<td>k-Nearest Neighbours</td>
<td>Both</td>
<td>Moderate</td>
<td>Fast/Slow</td>
<td>No Model</td>
</tr>
</tbody>
</table>

Comparison is made based on the typical performance of each algorithm [11, 13]; however, different implementation details and strategies may change the performance of the algorithm. Selecting the right algorithms for a particular application is not a simple decision. Many factors may guide the selection process such as the criticality of the application, the size of the data (both training and testing), and the type of the data variable. For example, if the training set size is very small compared to the test set size, then neural network algorithms will probably not be a good choice; however, simpler algorithms, such as linear regression, and Naive Bayes, will probably perform better for this type of data. Nevertheless, in many cases, it is hard to choose the most appropriate algorithm that will result in an acceptable performance for the problem at hand. Moreover, even if the “right algorithm” has been defined, usually a hyper-parameter fine-tuning step is required to obtain the best results. For example, a basic deep feed-forward neural network will require the user to define many hyper-parameters, such as the number of layers, the number of units in each layer, and the learning rate. This problem is still an open problem and Murphy [13] uses the “no free lunch” theorem to describe the difficulty of selecting the right algorithm. However, the ML community provides useful guidelines to help the user to select the right algorithm for the problem at hand, such as the Microsoft® Azure Machine Learning Algorithm Cheat Sheet.

2.2.2 Unsupervised Learning

In contrast to supervised learning, in which labels are missing or unavailable, unsupervised learning is used to detect certain patterns based on some similarity in the data. In other words, the problem of unsupervised learning is finding hidden patterns and structure in unlabelled data. Clustering is one of the most effective unsupervised techniques, in that the data instances are clustered into groups that maintain certain similarities. Different algorithms have been developed in the past to efficiently achieve this task [14]. *K-means* clustering algorithm [15] is one of the common techniques used to cluster different types of data. The algorithm uses distance as a similarity measure. Its aim is to minimize the squared distance function of the cluster points and iteratively update the k cluster centroids to achieve this goal. In contrast to k-means, which uses hard or crisp clustering by making each example belong to only one cluster, *fuzzy c-means* (FCM) uses a relaxed method by making it possible for each example to belong to all the clusters with a different degree of membership [16]. Unsupervised learning has been applied to many important tasks, such as dimensionality reduction and representation learning.

2.2.3 Semi-supervised Learning

When the training set size is very small with respect to the unlabelled set, the learning algorithm is required to explore knowledge and patterns in the labelled and unlabelled instances. Semi-supervised learning (SSL) tackles the shortage of labelled data by making use of the massive amounts of unlabelled data through an analysis conducted by unsupervised techniques. SSL combines both supervised and unsupervised methods to reduce the negative impact of the small training set [17].

2.2.4 Feature Extraction and Dimensionality Reduction

The classification performance of a supervised model is degraded when the number of training samples is small with respect to the dimension of feature space $\mathcal{X}$. This behaviour was studied
CHAPTER 2. BACKGROUND

by Gordon F. Hughes [18], and it has been known since then as the Hughes phenomenon. This phenomenon basically explains the relation between the size of the training set and the problem dimension. Hughes concluded that in order to maintain a good classification performance, the training set size has to be much greater than the dimension of the input features. Two approaches are commonly used to reduce the dimensionality of the problem: feature selection and feature extraction. Feature selection aims to select the most discriminating sub-features from the original feature space. Different supervised and unsupervised techniques are used to achieve this task [19]. In contrast, feature extraction usually aims to transform the problem into different compact feature space, and then select the most discriminating features. Principal Component Analysis (PCA) is one of the popular feature extraction techniques used to perform this transformation. Because of its recent outstanding performance in many difficult learning tasks, deep learning has recently been used to perform dimensionality reduction [20]. The results confirmed the applicability of deep learning approaches to perform dimensionality reduction for unseen data.

2.2.4.1 Principal Component Analysis (PCA)

PCA [21] is commonly used by the RS community to perform dimensionality reduction, as well as representation learning. PCA aims to extract the most important structures from the data by projecting the raw data into a space spanned by principal components. These components express the variances in the data, and the largest possible variance in the data can be found within the first few principal components. These components are obtained using eigen decomposition, which is a process in which the projection matrix and the diagonal eigenvalue matrix are computed. The components in the projection matrix corresponding to the largest eigenvalues capture the majority of the variances in the data, making PCA one of the most popular candidates for lossy data compression and dimensionality reduction. Furthermore, PCA is often used to visualize high dimensional data in 2D or 3D by plotting the two or three largest principal components.

KPCA is an extension of PCA using the kernel trick [22], which is based on mapping the original input space into a higher dimensional space using a non-linear mapping function. KPCA
is then performed by computing PCA in the new space using the kernel function. As the new space has greater dimensionality than the original space, the computation of the kernel function (i.e., the kernel matrix) for the entire dataset is expensive, and even intractable, if the original data is highly dimensional. Consequently, the kernel matrix is often computed using samples drawn from the original dataset, and these samples must be representative of the data distribution. Commonly, these samples are selected based on either prior knowledge about the data distribution or by using cluster analysis.

PCA and KPCA are simple and effective forms of feature extraction and dimensionality reduction with an analytical solution that has been applied to enhance classification of RS images [23, 24, 25].

### 2.2.5 Representation Learning

Similar to feature extraction and dimensionality reduction, which are introduced above, representation learning also aims to transfer the original feature space into another space. It is not necessary for the new space to be compact; however, it is important for the new space to be meaningful and useful for the learning algorithm to distinguish and detect hidden patterns. The successful implementation of deep learning algorithms in many difficult tasks, such as speech recognition, object recognition, and text mining, can be attributed to their ability to detect hidden features that are not visible in the original space. Bengio et al. [26] defines representation learning as “disentangling” the hidden underlying factors that describe the original data. They further argue that artificial intelligence (AI) will not be achieved without representation learning, because AI must fully understand everything around it. This understanding can only be obtained through representation learning. Representation learning can also be used as an efficient and effective method for dimensionality reduction. Autoencoder (AE) has been used recently to find robust and rich features. AE framework is presented in the next section.
CHAPTER 2. BACKGROUND

2.3 Deep Neural Networks (DNNs)

DNNs are one of the machine learning techniques that have been adopted to solve very difficult tasks in various fields [27]. One of the most interesting attributes of DNNs is their ability to learn the input-output relationship without requiring a human to perform any type of feature engineering or hand-crafting.

A DNN is an artificial neural network with multiple layers, each of which applies a simple numerical calculation to its input. The initial raw data transforms into a richer and more abstract representation as it moves forward from one layer to the next.

2.3.1 Neural Network Training Procedure

As shown in Figure 2.5(a), a simple NN consists of an input layer, a hidden layer, and an output layer. The connections between the neurons represent the weights used to compute the output of each neuron. As illustrated in Figure 2.5(b), the output of hidden layer neurons can be expressed as follows:

$$z = g \left( \sum_i (w_i \cdot x_i) + b \right)$$

where $g$ is the activation function, $x_i$ is the $i^{th}$ input from the previous layer (e.g., input layer), and $w_i$ and $b$ are the neuron weights and the bias respectively. The weights and the bias of each neuron are the parameters that the NNs aim to optimize. The activation function $g$ of each neuron is a differentiable non-linear function. Usually, all neurons of a particular layer use the same activation function; however, different layers may employ different activation functions.

The entire parameters of the network are optimized to minimize the network objective ($\mathcal{L}$). The network loss is computed using a predefined objective ($E$), which provides a numerical value reflecting the error of the network output ($\hat{y}$) with respect to the desired output ($y$). For simplicity, it is assumed for a given input $x$ that the network output is computed as $\hat{y} = f(x; \theta)$ where $\theta$
represents the parameters of the entire network. The loss can therefore be computed as follows:

\[ \mathcal{L} = E \left( f(x, \theta), y \right) \]

The objective function \( E \) is designed to best measure the network output error according to the type of output (e.g., real numerical values or discrete numerical values). The network’s goal is to
CHAPTER 2. BACKGROUND

find the best set of parameters $\hat{\theta}$ that minimize this objective, i.e.,

$$\hat{\theta} = \operatorname{argmin}_\theta E$$  \hspace{1cm} (2.1)

This is can be viewed as approximating the distribution that governs the input-output $(x,y)$ using the network parameters such that $P(y|x; \theta)$. The network parameters are updated in an iterative manner using backpropagation [28, 8] to compute the gradient of $E$ with respect to $\theta$. This is achieved by applying the chain rule for derivatives starting from the output and moving backward to compute the gradient of the loss with respect to each weight. Once the gradient of all the layers is computed, the weights are then updated using stochastic gradient descent (SGD) such that:

$$\theta = \theta - \mu \cdot \frac{\partial L_y}{\partial \theta}$$  \hspace{1cm} (2.2)

where $\mu$ is the learning rate. The SGD process is repeated until a termination criterion is achieved.

One of the key characteristics that distinguish NNs from most other machine learning techniques is their ability to handle large-scale datasets. This is achieved by performing each SGD iteration using only a subset of the training data, called a mini-batch. Each mini-batch is generated by sampling $m$ instances from the training data. In practice, the mini-batch size ($m$) is fine-tuned based on the available computing resources and the NN’s convergence behaviour. The average gradient of this mini-batch is used to update the network parameters. For a training dataset of size $N$ ($N \gg m$), sweeping through the entire data will require $k = \frac{N}{m}$ SGD iterations. One sweep through the entire training data is called an epoch, and usually hundreds or even thousands of epochs are required to achieve good results with large datasets.

With large datasets, the training time required for DNNs was a barrier, and only a few organizations and universities had enough computing and storage requirements to be able to conduct large-scale deep learning experiments. However, recent progress made in computing and the use of graphic processing units (GPUs) have made it possible to reduce the training time from several weeks to days [29]. Furthermore, in the last decade the deep learning community has optimized
CHAPTER 2. BACKGROUND

the training procedure in different ways, and with the utilization of distributed computing, it is now possible to reduce the training time of a very large dataset such as ImageNet (> 1 million images) from days to minutes [30, 31].

2.3.2 Representation Learning using DNNs

As discussed in the previous section, a DNN is able to learn a hierarchy of rich representations from the raw data. The superiority of DNNs is the result of their ability to automatically discover hidden patterns even without any labelled instances [26]. As a result, several DNNs have been developed and applied successfully to unsupervised learning such as autoencoders, stacked restricted Boltzmann machines and deep convolutional networks [32, 8, 33].

2.3.2.1 Autoencoder (AE)

The basic AE framework [34] seeks to learn a mapping function that can reconstruct a given input via a hidden representation. Reflecting on the basic NN discussed earlier, the desired output of the network is actually the input. As shown in Figure 2.6, AE learns a mapping function from

![Figure 2.6: Basic autoencoder.](image-url)
CHAPTER 2. BACKGROUND

the input layer to the output layer such that:

\[
Z = f_{\theta_e}(X),
\]

(2.3)

\[
\hat{X} = f_{\theta_d}(Z),
\]

(2.4)

where \( f_{\theta_e}(\cdot) \), called the encoder, and \( g_{\theta_d}(\cdot) \), called the decoder, are non-linear functions governed by the network parameters, \( \theta_e \) and \( \theta_d \), and \( \hat{X} \) is the reconstructed input. Using the same SGD procedure introduced earlier, the network parameters are updated to minimize the reconstruction error. Therefore, \( \theta_e \) and \( \theta_d \) are updated such that:

\[
\arg\min_{\theta_e, \theta_d} E(X, \hat{X})
\]

(2.5)

The error function, \( E(X, \hat{X}) \), is chosen such that it is appropriate for the specific type of data under consideration (real numbers, binary, categorical, etc.). For example, mean-square-error (MSE) is usually used to model the network objective if the input is expressed as real numbers (e.g., sensor readings). The same NNs learning procedure presented earlier is applied here. The hidden representation (\( Z \)), which represents the new feature space, can be a compact representation and has a much lower dimensionality than the original data. AE can therefore be considered a tool for compression, as shown in Figure 2.6.

This basic learning technique was extended in [35], by making the learned representation more robust to noisy data. The resulting training principle, known as Denoising Auto-Encoders, is based on training the model to reconstruct the original “clean” input from a corrupted version. The corrupted version of the input is generated by introducing perturbations to the original data instances. The denoising criterion prevents the network from simply copying the input to the hidden layer.

A DAE’s learning capacity can also be extended to learn different hierarchical levels of representation by stacking DAEs on top of each other [36]. Stacked DAEs are trained separately, and the learned representation at level \( i \) is used as input to the DAE at level \( i + 1 \).

AE has been very useful for initializing the network for other tasks such as classification.
CHAPTER 2. BACKGROUND

Moreover, the learned representation (Z) has been found to be richer and more useful than the original features (X) for the classification task.

2.3.2.2 Convolution Neural Networks (CNNs)

The first CNN (called LeNet) was introduced in 1999 for object recognition by LeCun et al. [37]. LeNet was very successful for character and digit recognition, and the banking sector has adopted CNN to automate the cheque clearing process [38]. However, CNN became popular just recently when Krizhevsky and his team won the large-scale ImageNet competition in 2012 [39], and since then CNN based models have dominated the machine vision field.

Figure 2.7 illustrates the basic building blocks of a CNN model which consists of a cascade of layers, each performing a certain numerical operation to its inputs. The common type of layers used in CNN are: 1) convolution, 2) pooling, and 3) fully-connected layers.

Figure 2.7: Convolution Neural Network - Basic blocks.

Convolution and pooling layers are applied in the earlier layers which operate in the spatial domain. The convolution layer consists of a number of filters with predefined hyper-parameters such as the filter size, stride size, and padding size. As illustrated in Figure 2.8(a), each filter convolves around the input image by moving the filter according to the stride size. While the spatial size of the filter (e.g., height and width) is based on the spatial locality features to be
learned, the depth of the filter is always equal to the depth of the input image; therefore, if the input image has $k$ features (or channels), then the filter depth will also be $k$. Consequently, the total number of parameters of each filter is $k \times h_1 \times h_2^4$, where $h_1$ and $h_2$ represent the height and the width of the filter respectively. The weights are the trainable parameters of a single filter, yet each convolution layer has multiple filters (usually the number of filters is large, i.e., $>32$); therefore, the total number of weights to be learned for a particular convolution layer consisting of $n$ filters will be $n \times k \times h_1 \times h_2$. The resulting feature maps (FM) from all the filters are stacked to perform the input for the subsequent operations. Due to the convolution operation, the size of these FMs is usually smaller than the input. If, however, the stride is set to 1 and the input image is padded with zeros, then the spatial size of the FMs will be equal to the input’s size.

Figure 2.8: Numerical operations in the convolution and pooling layers.

Usually, the convolution operation is followed by applying a non-linear function to the FMs.

For simplicity and clarity, the bias term is omitted in this example; however, it should be clear that one extra parameter is required for each filter if the bias term is considered.
CHAPTER 2. BACKGROUND

This non-linear function is called the activation function. Several activation functions can be used; however, to be able to perform backpropagation, this function should be differentiable (at least in the considered range of numerical values). Rectified Linear Units (ReLU), used in Figure 2.8 (b), are commonly used as activation functions for convolution layers.

The next basic operation is subsampling or pooling. As illustrated in Figure 2.8 (c), pooling is achieved by moving a window of fixed size around the input FM and producing a new FM that is much smaller in size. The reduction in the FM size depends on the dimensions of the pooling window and the stride extent. Pooling involves very simple numerical operations, such as computing the maximum value (max pooling), the average value (average pooling), and the sum value (sum pooling).

Stages of these convolution blocks (Convolution/ReLU/Pooling) are used in deep CNN; however, for the classification task, the last layers are usually fully-connected layers and the output layer is a one-hot encoding vector representing the output of the network. The same SGD update method is used here as that discussed in Section 2.3.1, and is based on computing the gradients of all the layers using backpropagation.

In addition to the advancements made in computing and storage, which have contributed to a substantial reduction in the training time, the simplicity of the CNN building blocks and the recent advances in the algorithms aimed at further optimizing and stabilizing the training procedure, have contributed heavily to making CNN overpower all other machine vision techniques. CNN has thus become the typical tool for many machine vision tasks. Furthermore, the deep learning community has made great progress in building scalable and easy to use deep learning libraries. These libraries enable the user to start training deep CNN architecture using personal desktops/laptops with minimum efforts. In the last few years, all of these advancements have resulted in a very different and sophisticated deep CNN architectures. These architecture have been used to address challenging machine vision tasks, such as segmentation, motion detection, and face detection; for example, CNN is now playing a key role in the realization of autonomous driving [40].
CHAPTER 2. BACKGROUND

2.4 Domain Adaptation

The amount of publicly available data (e.g., images, videos, audio, and text) has been growing at a massive rate and will continue to do so in the near future. For the machine learning community, this enormous amount of data is a source of opportunities and challenges to train highly complex models and deploy intelligent systems. While the opportunities are directly tied to the availability of the data, processing this huge amount of data efficiently is a major challenge. The term BigData [41] has been recently used to address these opportunities and challenges. One particular challenge for deploying such a large-scale system is the requirement of sufficient training samples for every “data source”. For example, consider that an artificial intelligence system is being designed based on object recognition, and an abundance of high resolution training images in illumination controlled environments are available. The system to be deployed will, however, be equipped with a low-resolution camera, and the images will be captured in a realistic environment (i.e., different illuminations and clutter backgrounds). The available training data is insufficient to provide the system with an acceptable operation performance, and the high cost of collecting sufficient data with the new camera is not acceptable. The solution to this particular challenge is to utilize the available training data and adapt it to the new domain.

2.4.1 DA Formulation and Notations

The common notations, which are listed in Table 2.3 and are used in the literature [42, 43], are adopted here to formulate the domain adaptation problem. The general supervised problem is formulated, after which the domain adaptation problem is formulated. In a classification or regression

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{X}$:</td>
<td>Input feature space</td>
<td>$P(x)$:</td>
<td>Input marginal distribution</td>
</tr>
<tr>
<td>$\mathcal{Y}$:</td>
<td>Label space</td>
<td>$P(y</td>
<td>x)$:</td>
</tr>
<tr>
<td>$\mathcal{T}$:</td>
<td>Learning task</td>
<td>$P(y)$:</td>
<td>Label marginal distribution</td>
</tr>
<tr>
<td>$D_S$:</td>
<td>Source domain</td>
<td>$P(x, y)$:</td>
<td>Joint distribution</td>
</tr>
<tr>
<td>$D_T$:</td>
<td>Target domain</td>
<td>$\hat{P}$:</td>
<td>Estimated distribution</td>
</tr>
</tbody>
</table>
task \( T \), a distribution (or a domain) \( D \) is defined by the feature space \( \mathcal{X} \), a \( d \)-dimensional input feature space (\( \mathcal{X} \subset \mathbb{R}^d \)), and the label space \( \mathcal{Y} \). The observed samples (e.g., training data) from this domain are \( (X, Y) =: \{(x_i, y_i)\}_{i=1}^n \) where \( x_i \in \mathcal{X} \) and \( y_i \in \mathcal{Y} \). The domain \( D \) is governed by an unknown labelling rule, or prediction function \( f: x \rightarrow y \), which maps the input \( x \) to the label \( y \). The goal of the task \( T \) is to estimate \( f \) using the observed data. \( f \) then can be used to predict the label of the new target data (e.g., test data).

From a statistical point of view, \( f(\cdot) \) represents the conditional probability distribution, \( P(y \mid x) \), which can be formulated as follows:

\[
P(y \mid x) = \frac{P(x, y)}{P(x)} \tag{2.6}
\]

where \( P(x) \), \( P(x, y) \) are the marginal and joint probability distributions respectively.

Unfortunately only a limited number of samples \( (X, Y) \) from the domain space are observed (e.g., the training data), and the learning task will be successful in estimating \( P(y \mid x) \) only if these samples are an equitable representation of the domain population \( (x, y) \). Using Equation (2.6) to approximate the labelling rule \( P(y \mid x) \) requires approximating both the input marginal distribution \( P(x) \) and the input-output joint distribution \( P(x, y) \) from the training data \( (X, Y) \). The learning system will be able to provide accurate predictions only if the instances in the training data provide a good estimate for these two distributions. In contrast, the learning system will provide poor predictions if the training data is not sufficient to provide a good estimate for these distributions. All supervised algorithms are thus built assuming that the following conditions are satisfied:

\[
\hat{P}(x) \approx P(x) \tag{2.7}
\]
\[
\hat{P}(x, y) \approx P(x, y) \tag{2.8}
\]

The domain adaptation problem follows the above setting, but the training and the test data are sampled from two different domains or distributions. The training data is sampled from the “source” domain \( D_S \) and the test data is sampled from the “target” domain \( D_T \). As a result, the above two
conditions in Equation (2.7) and Equation (2.8) are not satisfied. The domain adaptation problem is related to the *sample selection bias* problem [44] and the *covariate shift* problem [45]. For clarity, Table 2.4 summarizes the key differences between these scenarios. Training under sample selection bias is very challenging since there are insufficient training instances to estimate either of the distributions. In contrast, the covariate shift is a special case of the sample selection bias, in which the training data is assumed to be sufficient to obtain a reasonable estimate of the joint distribution. However, the covariate shift is more common when the frequency of the labels change significantly between the training and the test data. Both phenomena are thus more likely to appear when changing domains.

<table>
<thead>
<tr>
<th>Learning Type</th>
<th>Inputs</th>
<th>Assumptions/Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>General (Supervised)*</td>
<td>$D_S = (X_S, Y_S);$ $D_T = X_T$</td>
<td>$\mathcal{T}_S = \mathcal{T}_T$, $P(X_S) \approx P(x)$, $P(Y_S) \approx P(y)$</td>
</tr>
<tr>
<td>Sample Selection Bias</td>
<td>*</td>
<td>$P(X_S, Y_S) \neq P(x, y)$, $P(Y_S) \neq P(y)$</td>
</tr>
<tr>
<td>Covariate Shift</td>
<td>*</td>
<td>$P(Y_S</td>
</tr>
<tr>
<td>Domain Adaptation</td>
<td>*</td>
<td>Sample selection bias assumptions+ $\mathcal{T}_S \neq \mathcal{T}_T$, $\mathcal{D}_S \neq \mathcal{D}_T$</td>
</tr>
<tr>
<td>Domain Expansion (Multitasking,</td>
<td>$D_T = (X_T, Y_T);$ $\Theta(D_S)$</td>
<td>$\mathcal{T}_S \neq \mathcal{T}_T$, $\mathcal{D}_S \neq \mathcal{D}_T$</td>
</tr>
<tr>
<td>Lifelong Learning)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*The inputs and the assumptions/conditions of this setting holds for other settings, unless otherwise stated

**Domain expansion** (DE) is also related to DA; however, unlike all the other learning types listed in Table 2.4, which have a fixed learning task ($\mathcal{T}$) for both domains, the new domain learning task for DE is different from the old domain learning task. This implies that the label space for both domains is not the same ($\mathcal{Y}_S \neq \mathcal{Y}_T$). With DE, it is also assumed that sufficient training data from the new domain is available to train a model using the new domain data. The old training data is, however, unavailable, and instead, a model trained using the source domain $\Theta(D_S)$ is available. DE aims to update this model such that it can perform well and adapt to the new domain while preserving the model performance with the old domain.
CHAPTER 2. BACKGROUND

2.4.2 DA Theory

Ben-David et al. [46] were the first to introduce a lower bound on the domain generalization of a classifier trained using the source data. This bound can be simplified in terms of the classifier performance on the source, and the divergence between the source and the target marginal distributions. For two domains, $D_S$ and $D_T$, the target risk bound for the target domain $D_T$:

$$R_{D_T}(\eta) \leq R_{D_S}(\eta) + \hat{d}_H(D_S, D_T)$$

(2.9)

where $\eta \in H$ is the hypothesis in space $H$, and the $\hat{d}_H(D_S, D_T)$ term is the divergence between the two domains. The divergence between two distributions, $D_S$ and $D_T$, of size $N_{D_S}$ and $N_{D_T}$, respectively, with respect to the hypothesis class $H$ is formulated as follows [46]:

$$\hat{d}_H(D_S, D_T) = 2 \left( 1 - \min_{\eta \in H} \left[ \frac{1}{N_{D_S}} \sum_{x \in D_S} I[\eta(x) = 0] + \frac{1}{N_{D_T}} \sum_{x \in D_T} I[\eta(x) = 1] \right] \right)$$

(2.10)

where $I[a]$ is a binary indicator that is 1 if $a$ is true and 0 otherwise. $H$-divergence is expressed in terms of the ability of the hypothesis class $H$ to correctly identify the origin of the samples.

$H$-divergence cannot, however, be computed for all possible subsets, and it must be approximated. Ben-David et al. [46] introduced the Proxy $A$-distance ($\hat{d}_A$), which approximates the minimum term in Equation (2.10) using finite samples from the source and the target samples:

$$\hat{d}_A = 2(1 - 2\epsilon),$$

(2.11)

where $\epsilon$ is the classification error of a binary classifier trained to distinguish between the source samples and the target samples. Therefore, a strategy to control $H$-divergence is to find a representation of samples for both domains that are as indistinguishable as possible.

Blitzer et al. [47] performed one of the first theoretical studies which considered the problem of adaptation with multiple sources. This study set a bound for the error rate of a binary classifier
obtained from a weighted combination of the sources. Mansour et al. [48] demonstrated that a uniform combination of the hypothesis learned from different sources can perform very poorly in the target domain. This study demonstrated theoretically that it is possible to combine the predictors learned from each source to achieve a loss on the target that is less than the largest loss among the combined sources. The study assumed, however, that the target distribution is some mixture of the available source distribution, and that the solution could only be applied to regression predictors. Ben-David et al. [49] extended the work done in [47] and provided a similar bound to [48] but used only a single predictor trained on the subset of data selected from the source’s mixture. Recently Hoffman et al. [50] extended the work of Mansour et al. [48] and provided a normalized solution with theoretical guarantees for both regression and classification losses. This study also proposed a solution to estimate the optimal distribution-weighted combination vector by forming the problem as a Difference of Convex functions programming (DC-programming) problem [51].

2.5 Remote Sensing Image Classification Challenges

As discussed earlier, the classification task is one of the important tasks in many fields and several machine learning algorithms have been very successful in performing image classification. However, the RS image classification task is more challenging than the classical image classification task. Table 2.5 summarizes the key differences between the classical image classification and the RS image classification. The following sections will elaborate more on these differences.

Table 2.5: Classical vs. RS image classification tasks.

<table>
<thead>
<tr>
<th>Task</th>
<th>Feature space</th>
<th>Noise</th>
<th>Evaluation</th>
<th>Labeling</th>
<th>Benchmarking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical</td>
<td>Image-based</td>
<td>Spatial × Spectral</td>
<td>Low</td>
<td>Precise</td>
<td>Trivial</td>
</tr>
<tr>
<td>RS</td>
<td>Pixel-based</td>
<td>Spectral</td>
<td>High</td>
<td>Fuzzy</td>
<td>Hard</td>
</tr>
</tbody>
</table>
2.5.1 Pixel-based Task

Unlike the classical image classification task, which classifies the whole image into one particular class, RS image classification is pixel-based. Pixel-based classification involves classifying every pixel in the image into a particular class; therefore, the RS image classification task can be considered as a segmentation task. The methods used to handle RS data for the classification task can be categorized into three families: 1) pixel based, 2) patch-to-pixel, and 3) patch-to-patch. Figure 2.9 provides an illustration of these methods as applied to an RS image.

![Flatten
\( \mathbf{y} \in \mathbb{R}^1 \) Pixel-Based RS Image \[ \mathbf{X} \in \mathbb{R}^{(W \times H \times k)} \] \( \mathbf{y} \in \mathbb{R}^1 \) Input Vector \( (\mathbf{x} \in \mathbb{R}^k) \)
Ground-truth map \[ \mathbf{Y} \in \mathbb{R}^{(W \times H)} \]
Input Vector \( (\mathbf{x} \in \mathbb{R}^{w \times h \times k}) \) Input Volume \( (\mathbf{x} \in \mathbb{R}^{5 \times 5 \times k}) \) \( \mathbf{y} \in \mathbb{R}^{3 \times 3} \) Patch-to-pixel Patch-to-patch Flatten

Figure 2.9: Commonly used techniques to prepare RS data for ML.

- **Pixel-based:** This method considers only the spectral features \( (k \) features) of each pixel without considering the spatial information or the context of the pixel. It is therefore also known as the spectral-based method, and it has been the main method of processing RS images, especially HSI. As shown in Figure 2.9, each of the pixel spectral bands are flattened and the resultant vector of size \( k \) is considered as the input features for the pixel. The corresponding pixel value from the ground-truth map is the class label of the target pixel.

- **Patch-to-Pixel:** This method considers the spatial smoothness property of the image. Information from the neighbouring pixels are concatenated with the original spectral bands of the target pixel. This method is also known as “spatial-spectral feature extraction” and “spatial and contextual features extraction” [4]. Different techniques can be considered based on how...
CHAPTER 2. BACKGROUND

the spatial information will be incorporated. One of these techniques is to consider a window surrounding the target pixel and flatten the entire extracted window cube to obtain a vector of size $h_1 \times h_2 \times k$, where $h_1$, and $h_2$ are the window dimensions as shown in Figure 2.9. However, it is common to extract some statistical data within that window, such as minimum, maximum, and mean pixel intensity within the neighbouring window. Furthermore, more sophisticated filters, such as mathematical morphology and Markov random field operations, have been found to be useful. The value of the pixel at the center of the patch is the class label of the patch. Data from overlapping patches must be extracted to generate the training data for all the labelled pixels. Similarly, the test data is generated for all the unlabelled pixels.

- **Patch-to-Patch** This method is similar to the previous method, but the target here is no longer a pixel; it is also a patch. However, the image patch size is usually larger than the map patch size. As shown in Figure 2.9, the input image patch size is $5 \times 5$, while the output patch size is $3 \times 3$. This method has been found to be very useful and reduces the computation demands, especially for very high resolution RS images [52].

Furthermore, when performing pixel-based classification, the feature space for every instance is only the spectral dimension of the pixel; therefore the input feature becomes very small. While in the classical image classification task, the feature space is the entire image. For example, each image in ImageNet is $\mathbb{R}^{256 \times 256 \times 3}$, thus the feature space of a single instance is 196,608. On the other hand, the feature space of an RS single instance is $<12$ for the multispectral and $<200$ for hyperspectral images.

2.5.2 Image Quality

Noise in RS images is much higher than their counterparts in regular images [53]. Due to the nature of the RS acquisition system, the noise can be from different sources. Although some types of noise can be minimized and corrected, such as the noise caused by the sensing instruments and
CHAPTER 2. BACKGROUND

the aircraft’s stability during image acquisition, it is very difficult to remove other types of noise caused by other effects related to the scene itself. Examples of these factors are [54, 55]: atmospheric effects, sunlight angle, radiation resulting from a secondary illumination near the scene, and shadowing. Furthermore, these factors do not affect all the image bands equally; therefore, minimizing these effects is extremely difficult for RS [56].

2.5.3 Spectral Mixture

Due to the low resolution of RS images, pixels usually contain a combination of materials such as soil, vegetation, and man-made materials. The existence of such materials within the resolution of the image (e.g., five meters) is very common and therefore, the resultant spectral signature for a particular pixel will be a combination of these substances. An ongoing active field of research in RS, known as spectral unmixing, emerged to identify the pure substance in the image, known as endmembers, and estimate their ratio in each pixel [4].

2.5.4 Data Labeling

In many traditional applications, associating labels or tags to each example is not a very complex task. In fact, the labelling process is sometimes embedded within the data collection process as is the case with financial and social datasets. However, this task is not simple in RS, and in order to obtain accurate labels, a comprehensive and precise ground survey has to be conducted or image interpolation has to be performed. Moreover, it is common to assume that some of the pixels in the ground-truth have been incorrectly labelled, which in turn will weaken the classification performance. Another issue resulting from the nature of the task and the labelling process is that the labels are sparse, and it is not very common that the labelled pixels cover only a very small portion of the image (e.g., < 5%).
CHAPTER 2. BACKGROUND

2.6 Summary

This chapter covered important and essential background material related to RS classification. Remote sensing images from the acquisition stage up to the analysis and processing stage were introduced and discussed. Fundamental topics related to machine learning and deep learning were also presented and highlighted. Domain adaptation problem formulation, and theoretical work were also presented. Finally, the task of RS image classification was introduced and compared with the classical image processing task. In the next chapter, the domain adaptation algorithms proposed by the RS community and the deep learning community will be presented and discussed.
Chapter 3

Literature Review

In this chapter, the proposed domain adaptation techniques in the literature are reviewed. In addition to presenting previous works that have explicitly targeted remote sensing (RS), deep learning algorithms proposed for domain adaptation will also be reviewed.

3.1 Domain Adaptation Categories

Based on the definition of domain adaptation (DA) introduced in Chapter 2, several DA variants can be derived [42, 43]:

- **Homogeneous vs heterogeneous**: The DA problem is considered homogeneous when all the domains share the same feature space (i.e., $X_S = X_T$); otherwise it is considered as a heterogeneous DA problem (i.e., $X_S \neq X_T$). For example, a homogeneous DA task in natural language processing (e.g., a sentiment analysis task) is one which uses the same dictionary for all domains.

- **Semi-supervised vs. unsupervised**: The DA problem is considered semi-supervised if a subset of the target domain instances is labelled, and it is considered unsupervised if all the target domain instances are unlabelled.

- **Single vs. multi-source**: When there is only one source domain, the problem is considered
as a single source DA. The problem is considered to be a multi-source DA when the number of source domains is more than one.

3.2 Domain Adaptation in Remote Sensing

As discussed in Chapter 2, remote sensing images are more likely to exhibit significant data shifts between the domains. Therefore, the RS community has addressed this problem and several DA techniques have been proposed. Tuia et al. [57] divided DA techniques used in RS classification into four groups:

1. Selection of invariant features,

2. Representation learning,

3. Adaptation of classifiers,

4. Adaptation of the classifier by active learning (AL).

Because of the similarity between the last two groups, they are combined into one family (Adaptation of Classifiers). In the next section, the main characteristics of each group will be discussed, and the proposed techniques belonging to this group will be reviewed.

3.2.1 Selection of Invariant Features

This is usually achieved by considering only a subset of the original features that are invariant between the domains. Bruzzone and Persello [58] used this technique to solve the DA problem for HSI. To find the best features subset, two objectives were introduced: class separability and spatial invariance. The class separability objective (e.g., discrimination) is computed using supervised classification on the training set. For unsupervised DA settings, the spatial invariance objective is computed using the log-likelihood of the unlabelled samples in the target domain. Assuming that all the classes in the task follow Gaussian distribution, an Expectation Maximization (EM)
algorithm is used to approximate the distribution parameters (i.e., $\mu$: mean & $\Sigma$: variance) for all the classes of the unlabelled samples. The spatial invariance is then computed as the sum of the distances between each distribution in both domains. Multi-objective optimization is considered to search for the best features subset that maximizes the class separability objective and minimizes the spatial invariance objective. This technique is further improved in [58] based on kernel methods. Instead of approximating the spatial invariance by learning the distribution parameters, a non-parametric kernel function is proposed. The probability distributions of the source and the target are represented in reproducing kernel Hilbert space (RKHS) [59], which allows for a more suitable representation of the feature space. The proposed DA technique in [58] uses this RKHS space to approximate the similarity between the domains for a given subset of features from the original space. Similar to their previous work, a multi-objective optimization algorithm was also used to find the best features from the original feature space.

Yan et al. [60] utilized TrAdaBoost [61] to find the best set of features. TrAdaBoost is a transfer learning technique based on boosting, in which the weights of the training instances from the source domain are decreased to reduce their negative impact on the classifier performance. The heuristic technique proposed in [60] found the best set of features using a modified particle swarm optimization (PSO) technique. The optimization objectives are: 1) to increase the ability of different weak classifiers to correctly classify both hard and easy instances, and 2) to improve the overall boosting performance.

### 3.2.2 Representation Learning

Data is adapted such that the distribution over the domains is more compatible. This technique is also known as feature extraction and distribution matching. Several techniques have been proposed for feature extraction; however, the key difference between these techniques is how the transformation is formed and how the transformed data between the source and the target are aligned.

Several transformation methods have been studied for RS, such as histogram matching [62], graph matching [63], principal component analysis (PCA) [64], kernel-PCA [65], canonical corre-
CHAPTER 3. LITERATURE REVIEW

lation analysis (CCA) [66], and kernel-CCA [67]. Once the transformation is computed, a classifier is trained using the transformed features. However, some techniques do further processing which is aimed at further minimizing the divergence between the domains, such as Maximum Mean Discrepancy (MMD) [68] and Manifold Alignment (MA) [69, 70].

Most of these algorithms have not been evaluated for a general domain adaptation scenario; they have only been evaluated on change detection DA problems. Furthermore, most of these techniques are not good candidates for large-scale DA problems, since they employ eigensystems and kernels which require matrix manipulation in the order of $O(N^3)$, where $N$ is the number of samples.

3.2.3 Adaptation of Classifiers

Techniques belonging to this family take a semi-supervised strategy that utilizes the unlabelled target samples to adapt a classifier trained using the labelled source samples. Bruzzone and Prieto [71] used a Maximum Likelihood Classifier (MLC), which was trained on the source image and adopted for the target image. They considered a time-series analysis (i.e., both images were for the same geographical area). Assuming that the statistical distribution of the target pixels was a mixture of the density distribution of the classes in the source domain, the problem was transferred into a mixture density estimation to update the MLC parameters using only the target data in an unsupervised setting. The EM algorithm was used to approximate the mixture parameters for all the classes of the target samples by maximizing the log-likelihood of the mixture. This work was extended to increase its robustness by exploiting the temporal correlation between the time-series [72]. Cascade classification based on Bayesian rule [73] was adopted by considering the information contained in both domains.

Ensemble techniques are also used for time-series DA problems, which involve training different base classifiers on the source domain and then adapting these classifiers for the target domain. In [74, 75], the base supervised classifiers were an RBF neural network (RBF-NN) classifier and the MLC. As in [72], the density distribution of the target’s pixel was assumed to be a mixture
of the density distribution of the classes in the source domain, and EM was used to approximate the mixture parameters for both classifiers. Once the base parameters of the classifiers were updated, the final class label for the target pixel was computed by combining the prediction of these base-classifiers. Majority voting, Bayesian averaging, and posterior probability were proposed and evaluated for the combination strategy, and they all provided similar results.

The RS community has widely adopted the Support Vector Machines (SVM) [76] for supervised and semi-supervised learning. For example, transductive SVM (TSVM), introduced by Vapnik and Vapnik [77] as a semi-supervised framework, has also been utilized to solve DA problems in RS. In [78], the available training data was used to obtain the start-up separating hyperplanes of the SVM classifier. Pseudo-labels (or semi-labels) were obtained by labelling the unlabelled instances using this classifier. Pseudo-labels closest to the separating hyperplane were selected to be included in the next iteration. The selected pseudo-labels, “transductive”, were merged with the original training set to update the separating hyperplanes using a modified SVM optimization criteria. This modified optimization criterion gave a higher weight to the original training set to reduce the negative impact of misclassified semi-labels. DASVM [79] is similar to TSVM but instead of only adding the selected pseudo-labels from the target to the training data, DASVM also gradually removes some labelled source samples from the training data, based on their distance from the separation hyperplane. At the final iteration, the training data only includes the pseudo-labels selected from the target domain. After each iteration, the inserted semi-labels with inconsistent labelling results with respect to the previous iterations are removed from the training pool. SVM has also been used by the domain transfer multiple-kernel learning (DTMKL) technique proposed in [80]. In DTMKL, the optimization procedure of the SVM is modified to include a divergence term. This divergence term uses MMD to approximate the data distribution mismatch between the domains.

The other family which adopts the base classifier is the Active Learning (AL) paradigm. AL is a machine-user interactive system, which aims to minimize the effort needed to annotate the test data and to maximize the test classification performance. With the aid of a base classifier, an automated query generating function is designed to select the most informative samples to be
CHAPTER 3. LITERATURE REVIEW

annotated by the user. These samples are then added to the source training data to update the base classifier. This process is repeated based on the predefined criteria. It is worth mentioning that AL shares an important feature with semi-supervised learning (SSL): the usage of unlabelled data in the analysis. For this reason, several researchers consider AL as a semi-supervised learning technique; however, SSL does not assume the availability of an “annotator” (e.g., user) who can accurately label the selected instances from the test data.

The main challenge here is how to select the minimal set of informative target samples that the expert user needs to annotate. Matasci et al. [81] utilized TrAdaBoost to reweight samples from the source based on the classifier error (e.g., the probability of the sample being mislabelled by the classifier). Samples from the target domain were selected based on their level of uncertainty and the user was asked to provide the true class label for these samples. These target samples, which were annotated by the user, were added to the training pool. The weights of the training pool samples were updated according to their origin and classification errors such that the weights of the misclassified source samples were decreased and the weights of the misclassified target samples (annotated by the expert) were increased. Matasci et al. [81] modified the SVM classifier to incorporate the weights of the instances into the SVM cost function. Recently, Lunga et al. [82] proposed an AL domain adaptation framework based on a deep convolutional neural network. Different metrics were used to evaluate the solution quality with guidance from a human expert.

3.3 Deep Learning Techniques For Domain Adaptation

Due to their superior performance, their ability to handle large-scale data, and their ability to extract robust and rich features [83], several deep learning (DL) approaches have been proposed to tackle the DA problem. In fact, even without explicitly defining an adaptation objective, the feature extracted from a deep CNN model trained on a large-scale dataset [84], can outperform a task-specific feature extractor technique [85]. As the discrepancy between the domains increases, however, the approach must be designed explicitly to handle this significant discrepancy. The
primary focus in this section is to survey DL approaches used to solve the DA problem. For a more comprehensive survey, the reader is referred to these recent survey papers [43, 86]. Most of the recently developed DA algorithms are based on the theoretical studies introduced earlier, and therefore they can be categorized either as a reweighting approach or as a representation learning approach. All DL techniques reviewed in this section belong to the representation learning category.

DL techniques use different choices for representation learning. These choices are: 1) autoencoder-based approaches, 2) distribution matching (i.e., minimizing domain divergence), and 3) adversarial training approaches. Many DL techniques employ a combination of these choices to tackle the DA problem. This taxonomy is similar to the taxonomy introduced by Tzeng et al. [87] for the adversarial domain adaptation. The current study has divided the methods reviewed in this work based on the DA strategy. These strategies are autoencoders, distribution matching or adversarial training. Table 3.1 summarizes the key characteristics of the reviewed approaches.

Table 3.1: Characteristics of the DA deep learning techniques.

<table>
<thead>
<tr>
<th>Technique</th>
<th>End-to-end</th>
<th>Autoencoders</th>
<th>Weight sharing</th>
<th>Distribution Matching</th>
<th>Adversarial</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>fully</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>partial</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>none</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDA [88]</td>
<td>✓</td>
<td>SDA</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MS-STS [89]</td>
<td>✓</td>
<td>SDA</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DLID [90]</td>
<td>✓</td>
<td>PSD</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDC [91]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>MMD</td>
<td></td>
</tr>
<tr>
<td>MTAE [92]</td>
<td>✓</td>
<td>SDA</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAN [93]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>MK-MMD</td>
<td></td>
</tr>
<tr>
<td>JAN [94]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>JDD</td>
<td></td>
</tr>
<tr>
<td>DRCN [95]</td>
<td>✓</td>
<td>Conv. AE</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DANN [9]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSDA [96]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>MMD</td>
<td>✓</td>
</tr>
<tr>
<td>D-CORAL [97]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>CORAL</td>
<td></td>
</tr>
<tr>
<td>DSN [98]</td>
<td>✓</td>
<td>Conv. AE</td>
<td>✓</td>
<td>MMD</td>
<td>✓</td>
</tr>
<tr>
<td>CoGAN [99]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G2A [100]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ADDA [87]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 3. LITERATURE REVIEW

3.3.1 Autoencoder-based Approaches

DA techniques of this family are generally two-stage approaches, as shown in Figure 3.1. The first stage is entirely unsupervised; thus, the test data can also be used at this stage. For the second stage, only the training data is used to train the classifier. As shown in Figure 3.1, the classifier is trained using the learned representation obtained from the encoder. Compared to methods which have an analytical solution, such as PCA and KPCA, the training time of a DAE is considerably slower. There is, however, no need to compute or store a matrix on the order of \(O(N^2)\). This is important when dealing with large-scale datasets, such as those which are typically found in RS applications. Glorot et al. [88] were one of the first to utilize this type of representation learning, in which stacked denoising autoencoders (SDA) [35] are used for the representation learning. SDA consists of several (stacked) denoising autoencoders trained separately. Each denoising autoencoder is trained to minimize the reconstruction error of a corrupted version of the input [101]. The corrupted version of the input is generated by introducing perturbations to the original data instances, which will increase the model generalization capacity of the model and prevent the net-

![Figure 3.1: Representation learning using an autoencoder.](image-url)
work from simply copying the input to the hidden layer. Glorot et al. [88] tackled the Amazon sentiment classification task by using SDA for representation learning and linear support vector machines (SVM) for classification.

AE was also utilized in the “deep learning for domain adaptation by interpolating between domains” (DLID) technique proposed in [90]. DLID consists of two stages: in the first stage, deep non-linear feature extractor models \( F_{W_i} \) \( i \in [1, ..., K] \) learn using predictive sparse decomposition (PSD) [102], which can be viewed as an AE that decodes the input into a sparse representation. Each feature extractor is trained on a different dataset \( D_i \); the first feature extractor \( F_{W_1} \) is trained using only the source data \( D_S \). A subset of the source samples is replaced with a subset of samples from the target domain for all consequent feature extractors until only the target samples are used for the last feature extractor \( F_{W_k} \), in which \( D_k = D_T \). All of these feature extractors are trained in an unsupervised fashion to minimize the reconstruction error. In the second stage, all of the extracted features are concatenated \( \{Z_j\}_{j=1}^K \), and then fed as an input to a classifier (e.g., logistic regression). During this stage, the entire model, the feature extractors and the classifier are fine-tuned using the source domain labels and the available limited labelled instances of the target domain.

Ghifary et al. [92] proposed a multi-task autoencoder (MTAE) by extending the standard denoising autoencoder [35] to reconstruct multiple source samples from a single source sample. The shared representation (encoding) learns by jointly reconstructing the class sample of a domain from the other domain sample. At the same time, a private reconstruction (decoding) learns by minimizing the reconstruction error over all the domains. Training is performed in mini-batches sampled from one domain at a time. For each mini-batch, the shared encoding and the decoding private branch for the selected domain are updated. In other words, each optimization step aims to minimize the reconstruction error of a particular domain. By iterating through the domains, MTAE seeks a shared representation that would perform well with all the domains.

The multi-task autoencoder (MTAE) and D-MATE (the denoising version of MATE) have been tested against similar autoencoder approaches, such as DAE, using different object recognition
tasks with different source-target configurations. On average, the D-MATE outperforms all other techniques; however, in all the experiments, and in order to fulfill the basic training protocol of this framework, the number of samples for each class was set to be the same across all the domains. This was performed by pairing instances from the reconstructed domain with all instances that shared the same class from all the domains (i.e., including a copy of itself). To relax this constraint, the authors proposed the use of random-selection to reconstruct the between-domain pairs in which only the minimum number of samples per-class for all the domains $m_c = \min(n^1_c, n^2_c, ..., n^M_c)$ is considered, and random selection is suggested with the domains that have more samples (i.e., $n^i_c > m_c$). Furthermore, the training time of this approach seemed to be of an order of magnitude compared to similar approaches. The authors did not, however, report or discuss the training time and the algorithm complexity.

Unlike MTAE and DLID, deep reconstruction-classification networks (DRCN) [95] jointly optimize the discrimination and the reconstruction tasks. As illustrated in Figure 3.2, DRCN consists of two pipelines: a classification pipeline and a reconstruction pipeline. The classification pipeline is a typical supervised CNN model and the reconstruction pipeline is a convolutional autoencoder for data reconstruction. The intent of the entire model is to learn the following three functions: encoder ($G_{enc}$), decoder ($G_{dec}$) and a label predictor encoder ($G_{lab}$). These functions are governed by the parameters $\Theta_{dec}$, $\Theta_{enc}$, and $\Theta_{lab}$ respectively. The decoder function is shared between the

Figure 3.2: Deep Reconstruction-Classification Networks(DRCN) [95].
two pipelines; therefore, $\Theta_{\text{dec}}$ is updated by both pipelines. DRCN has adopted denoising and data augmentation to improve the algorithm performance. For real numbers, as is the case with image data, a squared loss is used for reconstruction error. Therefore, the DRCN loss is formulated as:

$$
L = \lambda L_c(\text{G}_{\text{lab}}(\text{G}_{\text{dec}}(X^s; \Theta_{\text{dec}}); \Theta_{\text{lab}}), Y^s) + (1 - \lambda) \| \text{G}_{\text{dec}}(\text{G}_{\text{enc}}(\tilde{X}^t; \Theta_{\text{enc}}); \Theta_{\text{dec}}) - X^t \|^2_2.
$$

where $L_c$ is the cross-entropy loss and $\lambda$ is a hyper-parameter controlling the trade-off between the supervised classification objective and the unsupervised reconstruction objective. The implemented training protocol is designed to alternate between the source and the target mini-batches on every iteration. DRCN performance is tested when using the source or when using both the source and the target data for the reconstruction objective. Results suggested that DRCN performs better when using only the target data, which suggests that the labelled source data is not useful for cross-domain generalization.

Kandaswamy et al. [89] proposed a DA technique for when there are multiple sources. The work is an extension of the ”source-target-source” (STS) technique [103] developed by the same author. STS is an iterative heuristic technique that alternates between the source and the target as an exploration mechanism to optimize a stacked denoising autoencoder (SDA), trained initially on the source data in completely unsupervised settings. The model resulting from this pretraining stage is considered as a baseline to be used during the iterative process. In the subsequent cycles, the baseline SDAE parameters are updated by switching between the target and the source in every cycle. Updates to the parameters of the baseline SDA are performed in two stages: an unsupervised update, which minimizes the reconstruction error of the noisy input samples, followed by a supervised fine-tuning stage, which adds a logistic regression layer to the last layer of the SDA. The algorithm terminates if no further improvements are achieved in the validation set. An outer loop is added to the STS to accommodate the multi-source scenario (MS-STS). In every main iteration of the MS-STS, a different source from the available sources is considered for the STS subitera-
CHAPTER 3. LITERATURE REVIEW

tions. MS-STS has been tested only on the CSIP dataset [104] for iris and periocular recognition, which is a very small dataset. Due to its very high computational demands, MS-STS cannot be easily scaled to larger datasets.

3.3.2 Distribution Matching Approaches

In this paradigm, the problem is formulated to find invariant representation for both domains based on minimizing the discrepancy between the features extracted from both domains, while at the same time minimizing the classification error with respect to the labelled instances of the source. The loss of this paradigm in unsupervised settings can be simply expressed as:

\[
L = L_{\text{class}}(X^s, Y^s) + \lambda L_{\text{DA}}(X^s, X^t)
\]

where \(L_{\text{class}}\) is the classification loss on the labelled data (i.e., the source domain samples), \(L_{\text{DA}}(X^s, X^t)\) is the domain adaptation loss computed using unlabelled samples from both domains, and \(\lambda\) is a hyper-parameter that controls the influence of the domain loss. Classification loss is usually computed using standard cross-entropy loss; however, the key difference between the techniques that adopt this paradigm is how the domain adaptation loss is computed. Comparing this loss with the bound introduced earlier in Equation (2.9), it is clear that this loss is approximating \(R_D(\eta)\) as \(L_{\text{class}}\) and \(\hat{d}_H\) as \(\lambda L_{\text{DA}}\).

Unlike some of the previously introduced techniques (e.g., MTAE, DLID), the main property of the approaches that utilize this paradigm is that it is an end-to-end technique in which the two objectives are considered at the same time. The gradient is used as a mechanism to optimize the parameters of the model. In contrast, the DRCN method introduced earlier is also an end-to-end approach.

Several DA techniques proposed recently follow this paradigm; for example, the deep domain confusion (DDC) introduced by Tzeng et al. [91] uses the maximum mean discrepancy

\(^1\text{http://csip.di.ubi.pt/}\)
CHAPTER 3. LITERATURE REVIEW

(MMD) [68] to compute the domain loss. It also uses all domains (i.e., source and target) to compute this loss using MMD. MMD is computed using the features extracted at the layer just before the classification layer. $L_{DA}$ is computed over mini-batches consisting of samples from both domains; however, $L_{class}$ is computed using only the source samples, and $L_{DA}$ is computed using all the samples. DDC therefore forks the data after the adaptation layer (the 7th layer). DDC was tested on an object recognition problem by fine-tuning all the layers of the CNN pretrained on ImageNet (AlexNet) [39].

The Deep Adaptation Network (DAN) [93] follows a similar approach but the multi-kernel version of the MMD (MK-MMD)[105] was used to compute the domain loss. Furthermore, this loss is computed as the sum over the last layers ($f_{c_6}, f_{c_7}, f_{c_8}$). To make this computation efficient with the linear computation time, DAN uses the unbiased MMD estimate [106], which only considers pairs of the data without replacement and uses a convex combination,"multi", of different kernels by varying the kernel bandwidth. DAN was also tested on object recognition problems by adopting a pretrained AlexNet model; however, the first three convolution layers were frozen during training.

Instead of using MMD, Sun and Saenko [97] proposed a method (D-CORAL) to align the correlation of the source and the target distributions using the second order statistics (covariances) of the extracted features from the source and the target to compute the domain adaptation loss. This loss, denoted as $L_{CORAL}$, is expressed as:

$$L_{CORAL} = \frac{1}{4d^2} \|C_S - C_T\|_F^2$$

where $d$ is the dimension of the extracted features. For an input $X$ with $N$ samples, $C$ can be computed as:

$$C(X) = \frac{1}{N-1} \left( X^\top X - \frac{1}{N} (1^\top X)(1^\top X) \right)$$

where $1$ is a column vector with all elements set to 1. Depending on the network architecture, the $L_{CORAL}$ can be computed for different layers. Assuming that this loss will be included for a set of
layers $\Omega$, the entire network loss can be expressed as:

$$
\mathcal{L} = \mathcal{L}_{\text{class}}(X^s, Y^s) + \sum_{j \in \Omega} \lambda_j \mathcal{L}^{j}_{\text{CORAL}}(X^s, X^t)
$$

(3.3)

Rozantsev et al. [96] proposed a method that was similar to DAN and DDC (denoted here as BSDA for “Beyond Sharing for DA”). BSDA also uses MMD to find the domain invariant representation; however, it also uses a weight regularizer between the non-shared layers. The weight regularizer is introduced to ensure that the weights of the non-shared layers are related; therefore a linear transformation penalty is chosen to model this relationship. The $L_2$ norm between the non-shared layers is used to compute this penalty.

Bousmalis et al. [98] proposed the use of Domain Separation Networks (DSNs) that utilize autoencoders to compute the domain loss. As shown in Figure 3.3, DSN consists of private encoders, one for each domain, and a shared decoder. Despite the fact that DSN employs autoencoders, such as MTAE and DLID techniques introduced in the previous subsection, the representation learning and the classification are learned jointly. DSN therefore uses the same objective introduced in Equation (3.2) with multiple terms to compute $\mathcal{L}_{\text{DA}}$:

$$
\mathcal{L} = \mathcal{L}_{\text{class}} + \alpha \mathcal{L}_{\text{recon}} + \gamma \mathcal{L}_{\text{similarity}} + \beta \mathcal{L}_{\text{difference}}
$$

(3.4)

where $\mathcal{L}_{\text{recon}}$ is the sum of the reconstruction error for each domain, $\mathcal{L}_{\text{similarity}}$ is the similarity loss that encourages the encoded shared representation for the domains to be similar, and $\mathcal{L}_{\text{difference}}$ is the difference loss between the encoded shared representation and the encoded private representation. The hyper-parameters, $\alpha$, $\gamma$, and $\beta$, control the influence of each term.

DSN proposes two methods to compute the similarity loss ($\mathcal{L}_{\text{similarity}}$): 1) using the adversarial approach by attaching a domain classifier to the shared representation to classify the origin of the samples as in [9], and 2) using the MK-MMD as in [93]. The difference loss ($\mathcal{L}_{\text{difference}}$) induces differences between the shared and the private representation of each domain, which will result
in extracting unique aspects for each domain. DSN realizes this loss by promoting orthogonality between the private and the shared representation vectors for each domain. Orthogonality is computed as the Euclidean norm of the inner product of the two vectors.

Long et al. [94] proposed a Joint Adaptation Networks (JAN) technique that uses a novel method, called the joint distribution discrepancy (JDD), to compute the domain loss. JDD extends MMD, and instead of computing only the discrepancy between the marginal distributions as in MMD, i.e., \( P(X^s) \) and \( P(X^t) \), JDD approximates the discrepancy between the joint distributions \( P(X^s, Y^s) \) and \( P(X^t, Y^t) \); however, for unsupervised settings, \( Y^t \) is not available during training, and in order to approximate JDD, a transductive learning approach is considered. JAN therefore uses the predicted class label probability vector computed by the classifier for both domains (i.e., the softmax layer) instead of using the crisp labels.

### 3.3.3 Adversarial Training Approaches

Adversarial training was initially introduced in order to avoid machine learning models being fooled by fake examples. Adversarial training does, however, seem to be an excellent tool for other tasks such as domain adaptation.

Ganin et al. [9] proposed a domain adversarial neural network (DANN) technique to estimate
the domain loss by training a domain classifier to discriminate between the origin of the samples using a deep feature extractor. To obtain invariant representation for both domains, the discrimination loss of the domain classifier must be minimized with respect to the domain classifier, and at the same time, maximized with respect to the representation layers. As shown in Figure 3.4,

![Figure 3.4: Domain Adversarial Neural Network (image from [9]).](image)

DANN consists of optimizing the parameters of three functions: 1) the feature extractor $G_f(\cdot; \theta_f)$, 2) the label predictor $G_y(\cdot; \theta_y)$, and 3) the domain classifier $G_d(\cdot; \theta_d)$. Using Equation (3.2), $L_{\text{class}} = L_y(\theta_f, \theta_y)$, and $L_{\text{DA}} = -L_d(\theta_f, \theta_d)$. The ‘-ve’ sign in $L_{\text{DA}}$ is adversarial, as the domain classifier is in an ‘adversarial’ relationship with the feature extractor. In other words, $G_d(\cdot; \theta_d)$ is optimized to minimize the domain classifier loss, and $G_f(\cdot; \theta_f)$ is optimized to maximize the domain classifier loss. The aim of the DANN is to optimize the following objective/energy $E$ function:

$$E(\theta_f, \theta_y, \theta_d) = L_y(\theta_f, \theta_y) - \lambda L_d(\theta_f, \theta_d), \quad (3.5)$$

where $L_y(\theta_f, \theta_y)$ is the total loss for the class discriminator against the source domain training samples, and $L_d(\theta_f, \theta_d)$ is the total loss for the domain discriminator against both the source training samples and the target training samples. The parameter $\lambda$ controls the influence of the domain discriminator loss on the features which have been learned. The learning algorithm updates the network parameters that simultaneously minimize the class discriminator loss ($L_y$), and maximize
CHAPTER 3. LITERATURE REVIEW

the domain discriminator loss ($\mathcal{L}_d$):

$$
(\hat{\theta}_f, \hat{\theta}_y) = \arg\min_{\theta_f, \theta_y} E(\theta_f, \theta_y, \hat{\theta}_d), \quad (3.6)
$$

$$
\hat{\theta}_d = \arg\max_{\theta_d} E(\hat{\theta}_f, \hat{\theta}_y, \theta_d). \quad (3.7)
$$

Using the standard backpropagation algorithm, the following update rules are applied:

$$
\theta_f = \theta_f - \mu \frac{\partial \mathcal{L}_y}{\partial \theta_f} + \lambda \mu \frac{\partial \mathcal{L}_d}{\partial \theta_d}, \quad (3.8)
$$

$$
\theta_y = \theta_y - \mu \frac{\partial \mathcal{L}_y}{\partial \theta_y}, \quad (3.9)
$$

$$
\theta_d = \theta_d - \mu \frac{\partial \mathcal{L}_d}{\partial \theta_d}, \quad (3.10)
$$

where $\mu$ is the learning rate. With the exception of the last term in Equation (3.8), all other updates are typical stochastic gradient descent (SGD) updates. The $+\lambda$ term in Equation (3.8) represents the maximization objective of Equation (3.7), which is an essential part of the DANN. In order to use the standard SGD as a basic learning algorithm, Ganin et al. [9] introduces a special layer called the gradient reversal layer (GRL), as shown in Figure 3.4. GRL acts as the identity transform during the forward pass, and as a gradient reversal during backpropagation. These two operations can be expressed as follows:

$$
R(x) = x, \quad (3.11)
$$

$$
\frac{dR(x)}{dx} = -\lambda I, \quad (3.12)
$$

where $I$ is the identity matrix. The implementation of the GRL is relatively simple using the available DL frameworks. It is done by defining the forward function as the identity transformation. During backpropagation the gradient is multiplied by a constant with no parameter updates.

Tzeng et al. [87] also utilized adversarial training and proposed an Adversarial Discriminative Domain Adaptation (ADDA) technique. Unlike DANN, ADDA is a multi-stage approach, as
CHAPTER 3. LITERATURE REVIEW

shown in Figure 3.5. In the first stage, which is known as the pretraining stage, a source mapping function and a classifier are trained using the source data. The source mapping function or “Source CNN” seen in Figure 3.5 is then used to initialize the second stage using the target mapping function or “Target CNN” seen in Figure 3.5. In this stage, known as the “adversarial adaptation” stage, the target mapping function is fine-tuned such that the discriminator cannot distinguish the source mapping from the target mapping. In the final stage, known as the testing stage, the fine-tuned target mapping function is used along with the classifier built during the pretraining stage to predict the class labels for the target samples.

The Generative Adversarial Network (GAN) [107] has recently received a great deal of attention and has been applied to many machine learning tasks [108]. GAN consists of two models which are trained simultaneously. The first is a generative model (G), which is trained to generate data instances from Gaussian noise. The resulting data instances are similar to real instances. The second model is a discriminator (D), which is trained to discriminate between the samples generated by (G), i.e., the fake examples, and the real examples. GANs are trained in an unsupervised fashion using a two-player min-max game which minimizes the (G) loss so that the generated instances are as similar as possible to the real instances. At the same time, the discrimination loss of (D) is maximized so that the discriminator will not be able to distinguish the original correct instance from the fake instance generated by (G). Liu and Tuzel [99] have created Coupled Generative Adversarial Networks (CoGAN) that use GAN to learn a robust and invariant representation that is domain independent. CoGAN was inspired by the idea that DNN models learn a hierarchical feature representation. In this particular GAN architecture, the generative models gradually
decode information from more abstract concepts to more detailed ones, and the discrimination models gradually decode the information in the opposite hierarchy.

To this end, CoGAN is designed so that the extracted abstract concepts of the different domains must be identical while allowing each domain to maintain its unique fine-grained details. The abstract concepts of the domains are made equal using a weight sharing strategy. In both parts of each GAN, the layers which map these abstract concepts are tied with the corresponding layer of the GAN of the other domain. Although CoCAN was originally proposed to generate images, it has also been tested for DA tasks. It is achieved with CoGAN by attaching a softmax layer to the last hidden layer of the discriminative models. Training is then completed by jointly optimizing the regular GAN objective for both domains, and only minimizing the classification error on the source domain.

Sankaranarayanan et al. [100] also utilized a GAN approach to learn the shared representation between the domains. In addition to the typical GAN modules, which include the generator (G) and the discriminator (D), the G2A (abbreviation of “generate-to-adapt”) also requires a feature extractor (F) and a classifier (C) as shown in Figure 3.6. Using the typical SGD updates, the optimization goes through three steps as shown in the following sequence. In Step 1, which uses the source samples, the goal of (G) is to deceive (D), and (D) is updated such that it can distinguish the true input from the fake output generated by (G). In Step 2, (F), (G), and (C) are updated to minimize the classification error, and to maximize the ability of (G) to deceive (D). In Step 3, (F) is updated by reversing the gradient of (D) for the target data such that (F) is encouraged to provide similar features for both domains. These updates are implemented in an alternating procedure. Each update uses a fixed-size mini-batch of the data, and the entire process is repeated for a fixed number of iterations.

### 3.3.4 Multi-source Domain Adaptation

Most proposed DA techniques consider the case of single-source DA, and few techniques have been proposed for multi-source domain adaptation (MDA). Early work in MDA was based on
either reweighting the samples based on the estimated differences in the conditional probabilities across all the source domains \[109, 110\] or by reweighting the classifiers that were learned from the different sources. They both, however, assume the availability of some labelled instances from the target domain.

MDA has recently gained more attention from the ML community and several approaches have been proposed. Zhao et al. [111] extended the domain adversary neural network (DANN) [9] to enable a feature extractor to learn by creating multiple domain discriminators that align all the samples, regardless of their origin, into a space that makes it hard for a domain discriminator to determine the origin of the sample. uses a similar approach in a Deep Cocktail Network (DCTN); however, instead of using only one classifier for the task, a separate classifier was trained for each source. During testing, DCTN used the confidence score of these classifiers to classify the target samples. Despite the promising results obtained from these adversary approaches, they required samples from the target domain(s) to be available during training. This is not practical, especially for large-scale applications such as remote sensing, in which sampling from the target domain would be noisy and could result in a negative transfer.

Hoffman et al. [50] extended the work in [48] and provided a more generalized theoretical guarantee for the MDA. Similar to [48], this study confirmed that an optimal distribution-weighted combination vector of the source classifiers exists and can provide the best results for the target domain. A new algorithm was proposed for estimating this vector from the pretrained source classifiers. Hoffman et al. [50] work is related to our MDA work in Chapter 5, as it does not require the availability of the target data. Scalability, however, is not considered, and the pretrained classifiers must have a similar performance. This is not easy to satisfy when dramatic differences exist be-
CHAPTER 3. LITERATURE REVIEW

tween the class distribution in each source, which is very common in remote sensing applications.

For RS, most of the MDA proposed techniques tend to target time-series change detection problems, which assume that all the domains represent the same geographical location [112]. Furthermore, most of the MDA techniques proposed for RS are kernel-based techniques [67, 113], which involve large matrix mathematical operations, and this is not suitable for large-scale problems. Finally, RS techniques that address the MDA problem are ensemble-based techniques and do not exploit the relationship between the domains [114].

3.3.5 Domain Expansion

Domain Expansion (DE) is related to Domain Adaptation (DA); however, the key differences between DE and DA are twofold: 1) DE solves two different tasks, one for each domain, and 2) the goal of DE is to perform well on the new task, while preserving the performance of the old task. In addition to the difference in the methods of the goal, a DE framework assumes that the old domain data is not accessible; therefore, DE aims to avoid the catastrophic forgetting problem in DNNs [115].

Several techniques have been proposed for DE, such as progressive neural networks (PNN) [116], which use the intermediate activation outputs from the old network as an auxiliary input to the new network. Kirkpatrick et al. [117] added a constraint, called the elastic weight consolidation (EWC), to encourage similarity between the weights of the two networks. Li and Hoiem [118] proposed a learning without forgetting (LwF) method, which enforces a similar response to be maintained for both tasks. Finally, Jung et al. [119] proposed a less-forgetful learning (LFL), which fixes the decision boundaries by freezing the parameters of the last fully connected layer, and adds a feature of similarity loss to the network objective.
CHAPTER 3. LITERATURE REVIEW

3.4 Datasets and Benchmarks

This section presents commonly used datasets and benchmarks for the domain adaptation task. First, the RS benchmarks are presented, followed by the domain adaptation benchmarks used by the deep learning community.

3.4.1 Remote Sensing Domain Adaptation Benchmarks

As discussed in Section 3.2, early DA techniques for RS focused on solving change detection applications in which only a single area of interest (AOI) is considered. Most of the datasets used for DA, however, are MSI and HSI data that has been initially collected and designed for the classification task. Using this data, DA is usually realized by dividing the image into two disjoint regions, one for each domain [120, 121]. Figures 3.7, 3.8, and 3.9 are examples of this approach. Time-series images are considered for a particular AOI [71, 79].

Several researchers have established very high-resolution aerial images for scene detection, such as the University of California, Merced Land Use Dataset (UC Merced) [122], which consists of 21 land use categories with 100 scenes per category. Figure 3.10 shows some examples of this dataset. Recently Xia et al. [123] have created aerial land use datasets consisting of 10,000 scenes and covering 30 land use categories. Since all of these datasets consist of very high-resolution
images, DL based techniques have established state-of-the-art results on these benchmarks with a performance comparable to that of a human.

Unfortunately, there were no large-scale DA benchmarks for pixel-based image classification in RS. To encourage research and advance the RS imagery analysis, the RS community has recently been doing extensive work to establish a standard and baseline benchmark for more challenging remote sensing tasks. In particular, there is the annual Data Fusion Contest (DFC), organized by the Data Fusion Technical Committee of the IEEE Geoscience and Remote Sensing Society (GRSS). Every year, they design and prepare data to address particular fusion problem(s) for RS. The DFC2017 contest was a local climate zones (LCZs) classification using free, open data. The training data consisted of five LCZ training zones from five different cities (Berlin, Hong Kong, Paris, Rome, and Sao Paulo), and the contest’s challenge was to predict the LCZ maps for four test cities (Amsterdam, Chicago, Madrid, and Xian). In addition to the ground-truth for the training
CHAPTER 3. LITERATURE REVIEW

Figure 3.10: Examples of the UC Merced Aerial data set.

cities, multispectral images from Landsat 8 and Sentinel-2 were also provided for all the cities. Furthermore, the users were encouraged to use other types of data to achieve the classification task as long as this data was from free and open sources. An example of such data was the auxiliary data provided from OpenStreetMap (OSM) for all the cities.

This problem is a perfect and practical example for domain adaption. Because of its challenging nature, its practicality, and the benefits provided to RS and geoscience studies by solving this problem, it was decided to consider this data for the multi-source DA problem in this research.

3.4.2 Deep Learning Domain Adaptation Benchmarks

Due to the superior performance of the supervised deep CNN in the image classification task, most of the proposed DA techniques have been evaluated on image classification/recognition tasks. Unlike RS, which lacks a standard DA benchmark that can be used as a standard gauge for evaluation, the DL community has established several benchmarks in different fields. In this section, the com-
Table 3.2: LCZs data

<table>
<thead>
<tr>
<th>LCZ Class label</th>
<th>Berlin</th>
<th>Hong Kong</th>
<th>Paris</th>
<th>Rome</th>
<th>Sao Paulo</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compact high-rise (LCZ 1)</td>
<td>0</td>
<td>631</td>
<td>56</td>
<td>0</td>
<td>955</td>
<td>1,642</td>
</tr>
<tr>
<td>Compact midrise (LCZ 2)</td>
<td>1,534</td>
<td>179</td>
<td>2,705</td>
<td>1,551</td>
<td>134</td>
<td>6,103</td>
</tr>
<tr>
<td>Compact low-rise (LCZ 3)</td>
<td>0</td>
<td>326</td>
<td>0</td>
<td>104</td>
<td>5,308</td>
<td>5,738</td>
</tr>
<tr>
<td>Open high-rise (LCZ 4)</td>
<td>577</td>
<td>673</td>
<td>366</td>
<td>0</td>
<td>482</td>
<td>2,098</td>
</tr>
<tr>
<td>Open midrise (LCZ 5)</td>
<td>2,448</td>
<td>126</td>
<td>446</td>
<td>1,495</td>
<td>244</td>
<td>4,759</td>
</tr>
<tr>
<td>Open low-rise (LCZ 6)</td>
<td>4,010</td>
<td>120</td>
<td>2,419</td>
<td>480</td>
<td>1,862</td>
<td>8,891</td>
</tr>
<tr>
<td>Large low-rise (LCZ 8)</td>
<td>1,654</td>
<td>137</td>
<td>748</td>
<td>435</td>
<td>1,915</td>
<td>4,889</td>
</tr>
<tr>
<td>Sparsely built (LCZ 9)</td>
<td>761</td>
<td>0</td>
<td>60</td>
<td>0</td>
<td>335</td>
<td>1,156</td>
</tr>
<tr>
<td>Heavy industry (LCZ 10)</td>
<td>0</td>
<td>219</td>
<td>0</td>
<td>51</td>
<td>179</td>
<td>449</td>
</tr>
<tr>
<td>Dense trees (LCZ A)</td>
<td>4,960</td>
<td>1,616</td>
<td>4,497</td>
<td>284</td>
<td>6,359</td>
<td>17,716</td>
</tr>
<tr>
<td>Scattered trees (LCZ B)</td>
<td>1,028</td>
<td>540</td>
<td>394</td>
<td>555</td>
<td>302</td>
<td>2,819</td>
</tr>
<tr>
<td>Bush and scrub (LCZ C)</td>
<td>1,050</td>
<td>691</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,741</td>
</tr>
<tr>
<td>Low plants (LCZ D)</td>
<td>4,424</td>
<td>985</td>
<td>7,688</td>
<td>984</td>
<td>376</td>
<td>14,457</td>
</tr>
<tr>
<td>Bare rock or paved (LCZ E)</td>
<td>0</td>
<td>214</td>
<td>0</td>
<td>109</td>
<td>323</td>
<td>323</td>
</tr>
<tr>
<td>Bare soil or sand (LCZ F)</td>
<td>359</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>144</td>
<td>503</td>
</tr>
<tr>
<td>Water (LCZ G)</td>
<td>1,732</td>
<td>2,603</td>
<td>234</td>
<td>500</td>
<td>3,492</td>
<td>8,561</td>
</tr>
<tr>
<td>Total</td>
<td>24,537</td>
<td>8,846</td>
<td>19,827</td>
<td>6,439</td>
<td>22,196</td>
<td></td>
</tr>
</tbody>
</table>

(a) L8- True color (left). LCZ Zones (Right)

(b) City center - zoom-in of the red box in (a)

Figure 3.11: Berlin LCZ training data.

(a) L8- True color (left). LCZ Zones (Right)

(b) City center - zoom-in of the red box in (a)

Figure 3.12: Paris LCZ training data.

Monly used domain adaptation benchmarks for image classification problems will be introduced.
CHAPTER 3. LITERATURE REVIEW

(a) L8- True color (left). LCZ Zones (Right)

(b) City center - zoom-in of the red box in (a)

Figure 3.13: Rome LCZ training data.

(a) L8- True color (left). LCZ Zones (Right)

(b) City center - zoom-in of the red box in (a)

Figure 3.14: Hong Kong LCZ training data.

(a) L8- True color (left). LCZ Zones (Right)

(b) City center - zoom-in of the red box in (a)

Figure 3.15: Sao Paulo LCZ training data.

3.4.2.1 Object Recognition

Office-31 [124] is a typical object recognition dataset consisting of 31 objects that are usually found in the office environment. Office-31 is composed of three domains: Amazon, which contains images downloaded from the Amazon online store (amazon.com), and DSLR and Webcam,
CHAPTER 3. LITERATURE REVIEW

which contain images captured using a Digital SLR camera and a web camera respectively. Amazon is the dominant domain, consisting of about 2,800 instances spread across the 31 classes, (≈ 90 images/class). Acquiring annotated data from this source is easy and inexpensive. Furthermore, the illumination and contrast in the Amazon domain images are fixed, as well as the background of the images. For this reason, Amazon is the optimal domain to be considered as a source of synthetic data. The other two domains consist of images of office objects taken in a real environment under less controlled lighting conditions. The key difference between the DSLR and the Webcam domains is the image resolution; however, both consist of very small number of instances per class (≈ 3 images/class for DSLR and ≈ 5 images/class for Webcam).

![Figure 3.16: Samples from Office-31 dataset.](image)

Figure 3.16 shows sample images of the backpack class from the three domains. It should be noted that there is almost no domain shift between Webcam and DSLR as the exact same objects are used for both domains; therefore, adapting the Amazon domain to either of the other domains provides similar results. However, the main challenge with this data is that the possible target domains (DSLR, Webcam) have very few examples. Since the amount of available data is critical for training a deep CNN model, most techniques therefore fine-tune a pretrained model on ImageNet (e.g., AlexNet [39]).

To further illustrate the superior performance of the DL techniques, Figure 3.17 shows the accuracy achieved over time for an Amazon to a Webcam DA problem. In these results, other non-deep learning methods were included, such as SURF [124]. All DL methods performed better than SURF, even without being explicitly designed for DA (Green methods in the chart: DeCAF,
CHAPTER 3. LITERATURE REVIEW

AlexNet, ResNet-50). Furthermore, the results are further improved by fine-tuning a model that is trained on a large scale dataset (i.e, AlexNet), and even further improved with deeper networks such as ResNet [125]. A significant improvement has been noticed recently as the approaches have been designed to explicitly address the DA problem. It is worth mentioning that some of the introduced DL techniques have not been evaluated against this benchmark, such as DSN [98] and CoGAN [99]. In particular, Bousmalis et al. [98] criticized this dataset for being noisy and lacking diversity.

![Amazon --> Webcam](image)

Figure 3.17: Office-31 DA results (dotted line presents the results trend).

3.4.2.2 Digit Recognition

![Digit Recognition Datasets](image)

Figure 3.18: Examples of the digit recognition datasets.

Digit recognition is one of the first tasks in which DL, and especially deep CNN, show remarkable results. Furthermore, there are a variety of datasets in this task that fit the DA problem requirements nicely. Commonly used datasets are MNIST [126], MNIST-M [9], USPS [127] and SVHN [128], each of which has 10 class labels [0-9]. Figure 3.18 provides examples of these.
datasets. Various DA scenarios can be designed out of this data by considering different source → target configurations. Results of commonly considered DA configurations are listed in Table 3.3, which includes the following configurations:

**MNIST → MNIST-M:** In this problem, the MNIST dataset is used as a source. The target, MNIST-M, is constructed by blending MNIST digits with a natural scene patch randomly extracted from the Berkeley Segmentation Dataset (BSDS500). The process is as follows [9]. For every example in the MNIST dataset, a random patch from BSDS500 is extracted and the patch pixels are subtracted from the MNIST digit sample. Unlike MNIST, MNIST-M is an RGB image and therefore, MNIST digits are converted to RGB by using the original image channel for the three channels, as seen in in Table 3.3.

**MNIST ↔ USPS:** USPS is similar to MNIST but it has less diversity and fewer numbers of examples; therefore, adapting MNIST → USPS is more successful than the opposite adaptation task.

**MNIST → SVHN:** SVHN consists of house number patches extracted from RGB street view images [128]; therefore SVHN is more diverse than MNIST. Although MNIST test set performance of a supervised classification model trained on MNIST training set is almost perfect (99.5% accuracy), a dramatic drop in the performance is noticed when this model is applied to SVHN (0.603 vs. 0.260). This is due in part to the huge gap in the data distribution, and also because MNIST does not contain enough diversity to assist in the adaptation task; therefore, many approaches have failed this task [9].

**SVHN → MNIST:** For this configuration, all approaches are able to significantly reduce the gap between the source and the target baselines by at least 14%. In fact, the DSN method achieved an accuracy comparable to the target baseline (0.931 vs. 0.995). This configuration is one of the successful examples, and it shows how the adaptation can be successful when the source data is large and diverse.
Table 3.3: Digit recognition DA classification accuracy. The percentage of reduction of the gap between the source and the target baselines is also shown (in brackets).

<table>
<thead>
<tr>
<th>Method</th>
<th>SVHN → MNIST</th>
<th>MNIST → SVHN</th>
<th>MNIST → USPS</th>
<th>USPS → MNIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source baseline</td>
<td>0.603</td>
<td>0.260</td>
<td>0.752</td>
<td>0.571</td>
</tr>
<tr>
<td>DANN</td>
<td>0.739 (34.8%)</td>
<td>0.357 (14.6%)</td>
<td>0.911 (76.1%)</td>
<td>0.740 (39.9%)</td>
</tr>
<tr>
<td>DRCN</td>
<td>0.820 (55.5%)</td>
<td><strong>0.401</strong> (21.2%)</td>
<td>0.918 (39.2%)</td>
<td>0.737 (39.2%)</td>
</tr>
<tr>
<td>DSN</td>
<td><strong>0.931</strong> (83.8%)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CoGAN</td>
<td>-</td>
<td>-</td>
<td>0.912 (76.6%)</td>
<td>0.892 (75.7%)</td>
</tr>
<tr>
<td>ADDA</td>
<td>0.760 (40.1%)</td>
<td>-</td>
<td>0.894 (67.9%)</td>
<td>0.901 (77.8%)</td>
</tr>
<tr>
<td>G2A</td>
<td>0.847 (62.4%)</td>
<td>0.364 (15.7%)</td>
<td><strong>0.925</strong> (82.8%)</td>
<td><strong>0.905</strong> (79.5%)</td>
</tr>
</tbody>
</table>

Target baseline 0.995 0.924 0.961 0.995

### 3.5 Summary

In this chapter, a literature review of the state-of-the-art domain adaptation techniques for RS was presented and discussed. The deep learning algorithms which have been recently introduced for domain adaptation were also presented.

Deep learning is a very powerful tool, and it has been successfully applied to different fields, including RS. However, to our knowledge, domain adaptation based on deep learning has not been explored in this area. Furthermore, a multi-source domain adaptation problem is still an open problem and requires further investigation, especially for RS tasks.
Chapter 4

Single Source Domain Adaptation

4.1 Introduction

Unsupervised domain adaptation (UDA) is a challenging task. As discussed in Section 3.2, most of the recently proposed UDA methods for remote sensing are either iterative techniques that aim to adopt a base classifier to the new domain or two-stage approaches. In the two-stage approach, a representation learning is performed first, followed by the training a classifier in the second stage. In this work, we consider a contemporary approach which combines the two learning stages into a single stage. Such end-to-end methods have recently become popular for Domain Adaptation in the machine learning community, though to date, they have not received attention in remote sensing. Domain-Adversarial Neural Networks (DANN), introduced by Ganin et al. [9], achieves this by learning a representation that is as close as possible for both domains, and at the same time, this representation can be used by a discriminator to classify the available training samples from the source domain.

Furthermore, when using representation learning for remote sensing DA problems, liner transformation such as PCA or non-linear such as KPCA are usually used. Despite their simplicity and effectiveness, these methods are not scalable and cannot be applied for a large scale problem. To overcome the lack of scalability in the two-stage learning paradigm, we also investigated the
CHAPTER 4. SINGLE SOURCE DOMAIN ADAPTATION

effectiveness of utilizing Denoising Autoencoders (DAE) for this task.

In summary, the objectives of this study are:

1. Introduce to the remote sensing community deep learning methods to be used as a scalable representation learning method for the DA problems in remote sensing.

2. Investigate the two-stage and the end-to-end paradigms for DA in the context of pixel-based classification of remote sensing images.

3. Evaluate the performance of the proposed methods in different UDA scenarios against commonly used techniques in this field.

This study considers the problem of general land-use classification using hyperspectral images (HSI) and the problem of crop mapping using multispectral images (MSI). Crop mapping is an important activity for countries like Canada and the USA, where the agricultural industry represents a significant portion of the economy. In order for the crop map to be accurate, massive field surveys and data collection efforts must be conducted across the whole country on an annual basis [129]. In this context, DA will reduce this survey effort, and produce high-quality maps without the need for collecting ground-truth data for every season. This will result in producing maps much faster than the current practice allows.

The remainder of this chapter is organized as follows: Section 4.2 introduces the considered UDA paradigm considered in the study. Section 4.3 presents the adopted experimental design and Section 4.4 describes the data used for the experiments. results are presented in Section 4.5 and are followed by a discussion in Section 4.6. Finally, a summary is presented in Section 4.7.

4.2 Domain Adaptation Paradigms

As discussed in Section 3.2, we have grouped DA techniques into three groups. Our approach belongs to the family of representation learning techniques which have received great attention recently in the context of supervised and unsupervised learning [26].
CHAPTER 4. SINGLE SOURCE DOMAIN ADAPTATION

In this section, we compare the classical two-stage and the end-to-end (single-stage) representation learning DA paradigms.

![Figure 4.1: Domain adaptation paradigms considered in this study.](image)

(a) Two-stage DA
(b) DANN

Figure 4.1 provides a visual overview of each approach.

**4.2.1 Two Stage DA**

As illustrated in Figure 4.1a, two-stage DA classification is performed using two separate tasks: 1) representation learning, and 2) classification.

1) **Representation Learning:** This is the core component of the framework, which uses one or more input images to learn a common latent space. Several unsupervised techniques can be used here, such as PCA, Kernel PCA and, DAEs, which do not require class information (labels) to be associated with the input images. The output of this stage is a mapping function ($\Phi$) that maps any given remote sensing sample to the new representation space. The input image samples used for learning the representation are usually rescaled (e.g., $[0,1]$ or $[-1,1]$) before learning the mapping, and the same rescaling function is applied to the new samples provided during the next stage.

2) **Classification:** This task is based on supervised learning, and therefore, it assumes that labelled samples are available. First, the samples obtained by remote sensing are mapped to the new representation space using the transformation function $\Phi$. A classifier is then trained using the transformed samples and their associated ground-truth labels, which have been supplied by the user, to learning a mapping from input space to discrete classes ($\Theta$). Newly acquired remote sensing images can be mapped using $\Phi$, and then classified using $\Theta$. 
CHAPTER 4. SINGLE SOURCE DOMAIN ADAPTATION

4.2.2 DANN: End-to-End DA

The DANN introduced by Ganin et al. [9] is inspired by the theoretical work of Ben-David et al. [46], where they establish a framework to compute the divergence between two probability distributions. The reader can refer to Section 2.4.2 for further details. Ben-David et al. [46] have shown that the Proxy $\mathcal{A}$-distance $\hat{d}_{\mathcal{A}}$, which approximate the distance between two domains, can be minimized by finding a representation that “confuses” the binary classifier, preventing it from distinguishing the origin of the samples. The binary classifier can be easily established by constructing a new dataset where the source samples are labelled as 0, and the target samples are labelled as 1. Ganin et al. [9] exploited this idea to build a type of deep neural network that learns a representation invariant to source and target domain. To establish this, and as discussed in Section 3.3.3, Ganin et al. [9] proposed a domain Adversarial Neural Network (DANN) aims of finding features that minimize the classification error and maximize the domain discriminator error.

Unlike the two-stage DA approaches, which do not require the availability of samples from the target domain during training, DANN, as an end-to-end approach, requires the availability of unlabelled samples from the target domain during training. Although this might be a concern for some applications, unlabelled samples from the target domain are usually available at no cost.

DANN have recently established state-of-the-art results for various domain adaptation tasks, such as sentiment analysis and image classification [9]. However, they have yet to see adoption outside the machine learning community. To this end, our main contributions are twofold:

- We investigate the use of DANN to address the domain adaptation problem across different DA classification tasks.
- We compare the performance of DANN to several variants of representation learning within the two-stage paradigm. We focus on DAE for the latter, but also evaluate several classical approaches.

The main advantages of the DANN proposed here, over the existing representation learning techniques in the literature are: 1) it can simultaneously perform the representation learning task and
CHAPTER 4. SINGLE SOURCE DOMAIN ADAPTATION

the classification task within a coherent architecture; 2) it can be easily implemented using existing deep learning frameworks; and 3) computation scales linearly with the number of data points. The latter is very important, especially in terms of practicality, since most existing methods are based on solving a generalized eigen decomposition problem using large dense similarity and dissimilarity matrices (c.f. [67, 70]). Distance-based methods will not be efficient within an operational system that deals with large images. On the other hand, DANN as well as DAE, as techniques trained by stochastic gradient descent, can naturally handle this issue using the typical mini-batch training procedure.

The main obstacles that may prevent the utilization of deep learning models are the computational resources needed for training, as well as how to efficiently identify optimal network hyperparameters, such as the network architecture (e.g., number of layers and neurons in each layer), and the learning rate. However, the deep learning community is making tremendous progress in speeding up the learning process by utilizing GPUs and distributed systems [30], and in automating the search over hyperparameters [130]. Consequently, we anticipate that deep learning will help in realizing an operational intelligent remote sensing system.

4.3 Experimental Setup

This section describes the design of experiments used to implement and evaluate the proposed techniques. The performance of DANN and DAE has been compared to PCA and KPCA. We use the following notation: Let a source domain $D_S\{X_S, Y_S\}$ be a set of pixels ($X_S$), with the associated ground truth ($Y_S$), and the target domain, $D_T\{X_T\}$, be a set of pixels ($X_T$). The objective is to utilize the prior information provided in the source domain to predict the ground truth for the target domain ($Y_T$), which is only available for the purposes of evaluation. The initial features (e.g., spectral bands and filter bands) of the two domains have been scaled to zero mean and unit variance using the descriptive statistics of the source samples.
4.3.1 Baselines and DA Settings

To evaluate the performance of the proposed framework, we used the same settings as Matasci et al. [67]. These settings are summarized in Table 4.1. The implemented techniques are compared against the following baselines:

<table>
<thead>
<tr>
<th>Method</th>
<th>X</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training set</td>
<td>Testing set</td>
</tr>
<tr>
<td>Src</td>
<td>–</td>
<td>(X_S) (X_T)</td>
</tr>
<tr>
<td>Tgt</td>
<td>–</td>
<td>(X_T) (X_T)</td>
</tr>
<tr>
<td>1DOM</td>
<td>(X_S)</td>
<td>(\Phi(X_S)) (\Phi(X_T))</td>
</tr>
<tr>
<td>2DOM</td>
<td>(X_S \cup X_T)</td>
<td>(\Phi(X_S)) (\Phi(X_T))</td>
</tr>
<tr>
<td>DANN</td>
<td>(X_S \cup X_T)</td>
<td>(G_f(X_S)) (G_f(X_T))</td>
</tr>
</tbody>
</table>

**Source baseline (Src):** This baseline is computed using samples from the source image for training and the samples from the target image for testing. The Src baseline is considered to be the lower baseline, which we aim to outperform.

**Target baseline (Tgt):** This baseline is computed using a classifier that is trained and tested on samples from the target domain. The Tgt baseline is considered to be the upper baseline, containing information that we would normally not have in practice.

There is no representation learning for both the Src and the Tgt baselines, and the classifier is trained using the original spectral bands of each domain. The main objective of DA is to perform as well as the Tgt baseline while using only the label information provided in the source domain.

4.3.2 Origin of Samples Used for DA

Two cases are considered for the origin of the samples, \(X\), used to learn the new latent representation (i.e., learning the mapping function \(\Phi\)), as shown in Table 4.1:

- **1DOM** \(\left(X \subseteq X_S\right)\): A subset of samples from the source image are drawn randomly.
- **2DOM** \(\left(X \subseteq (X_S \cup X_T)\right)\): A subset of samples from both the source and target images
are drawn randomly.

The DANN technique conforms to the 2DOM setting, as it naturally requires unlabelled samples from the target domain during training, while two-stage approaches can be trained as 1DOM or 2DOM.

The number of samples drawn from each class is fixed for both cases. Therefore, the size of samples used for representation learning is fixed to \((C \times N)\) samples \((N\) pixels/class), where \(C\) is the number of classes. For all 1DOM DA techniques, only the samples drawn from the source are used for representation learning: \(X \subset (X_S)\). For the 2DOM DA techniques, including DANN, the samples drawn from the target domain are also used for the representation learning task, \(X \subset (X_S \cup X_T)\). However, the labels of the samples drawn from the target domain are not used during training.

Besides selecting the number of components to be used as the new representation space, PCA is a parameter-free approach; however, KPCA requires designing the kernel and specifying its parameters. We follow Matasci et al. [67] and used the Gaussian kernel\(^1\), and the median Euclidean distance computed over the samples, \(X\), is used as an estimate of \(\sigma\).

### 4.3.3 DAE and DANN Architecture and Hyper-parameters

Different architectures and denoising techniques were investigated for the DAE. We found that the best results were obtained when using an overcomplete shallow architecture and additive Gaussian noise. For all experiments, the hidden layer size and the standard deviation of the noise were set to 200 and 0.1, respectively. The sigmoid activation function was adopted for all neurons, and the mean squared error between the input and its reconstructed version was used as a cost function. Training was performed using SGD with a learning rate set to 0.1.

DANN is realized by a deep architecture to learn \(G_f\). We tested different depths and found that the performance deteriorated beyond three hidden layers. Therefore, the depth of the network was

\[ K(x_i, x_j) = e^{-\frac{(x_i - x_j)^2}{2\sigma^2}} \]
fixed to three hidden layers. Furthermore, the size of each hidden layer was fixed to 50 neurons for all layers (similar performance was observed with 20, 30 and 100 neurons). Training was performed using SGD with the following learning rate schedule:

\[ \mu_i = \mu_0 (1 + \gamma i)^{-p}, \]

where \( \mu_0 \) is the initial learning rate, which was set to 0.9, \( \gamma \) and \( P \) are the annealing parameters, which were set to \( 10^{-3} \) and 0.75, respectively. To avoid over-fitting, \( \ell_2 \)-norm restrictions on the network weights were applied with the penalty parameter set to \( 10^{-4} \).

In order to fairly compare the DANN performance in the DA setting, we computed two additional baselines using a standard deep neural network (DNN) architecture with three hidden layers, followed by a logistic regression output layer. This architecture is identical to the DANN architecture presented in Figure 4.1b when the domain discriminator part is dismissed, which can be achieved by eliminating the influence of the domain discriminator loss (i.e., setting the GRL parameter \( \lambda \) to 0). Since all neural network-based techniques (e.g., DAE, DNN and DANN) are trained using mini-batches, the mini-batch size parameter is fixed to 32 in all experiments, and training is terminated after 250 epochs (i.e., \( 250[C \times N \times (32)^{-1}] \) iterations).

### 4.3.4 Classification task

We evaluated Logistic Regression (LR), linear support vector machines (SVM), and radial basis function (RBF) SVMs. In terms of speed, LR was the fastest, followed by the linear SVM. In terms of performance, the LR and linear SVM have similar performance, while the RBF SVM has slightly better performance, but requires a longer training time and parameter tuning. Therefore, LR was adopted for all classification tasks. LR is a linear probabilistic model that computes the probability of the output based on the input \( P(x \mid y) \) in a supervised fashion. LR finds the model parameters that maximize the log likelihood of the output, \( y \), for the given input, \( x \). In this study, LR was adopted in all proposed methods including DANN, in which LR is used as the output layer.
CHAPTER 4. SINGLE SOURCE DOMAIN ADAPTATION

for both the class and domain discriminators.

For the training set size, we adopt the same sampling strategy used for representation learning (i.e, $N$ samples per class). This training set is drawn from the source domain; however, in the case of the Tgt baselines, this training set is drawn from the target domain. The test set size is fixed to 50% of the total samples of the target domain.

4.4 Datasets

The proposed framework is evaluated on HSI and MSI images.

4.4.1 Hyperspectral Images

This image was acquired using the ROSIS-03 hyperspectral sensor over the city of Pavia. This image has a spatial resolution of 1.3 m and covers about 1 km$^2$. The sensor covers the spectrum between 430 and 860 nm in 115 spectral bands. Noisy and uncalibrated bands are removed, and the remaining 102 bands are atmospherically corrected using the ground measurements and sun spectrometer data collected during the flight campaign [131]. Figure 4.2 shows a true color composite of the image.

Table 4.2: Classes in the Pavia city dataset.

<table>
<thead>
<tr>
<th>Classes</th>
<th>Source ($X_S$)</th>
<th>Target ($X_T$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># of Pixels</td>
<td># of Pixels</td>
</tr>
<tr>
<td></td>
<td>%</td>
<td>%</td>
</tr>
<tr>
<td>Roads</td>
<td>326</td>
<td>2,549</td>
</tr>
<tr>
<td></td>
<td>7.96</td>
<td>9.07</td>
</tr>
<tr>
<td>Vegetation</td>
<td>1,793</td>
<td>6,406</td>
</tr>
<tr>
<td></td>
<td>43.75</td>
<td>22.80</td>
</tr>
<tr>
<td>Shadows</td>
<td>514</td>
<td>1,638</td>
</tr>
<tr>
<td></td>
<td>12.54</td>
<td>5.83</td>
</tr>
<tr>
<td>Buildings</td>
<td>1,465</td>
<td>17,501</td>
</tr>
<tr>
<td></td>
<td>35.75</td>
<td>62.29</td>
</tr>
<tr>
<td>Total</td>
<td>4,089</td>
<td>28,094</td>
</tr>
</tbody>
</table>

Following the settings found in [67], two disjoint regions are used to simulate the source and target domains, as shown in Figure 4.2. In these two regions, four classes are used for the classification task. Table 4.2 summarizes these classes and their distribution of each region. Other pixels in the image are considered as “background pixels”, and are not included in the analysis. For this
dataset, 200 samples from each class comprise the training set, which is used to learn $\Phi$, and to train the classifier, $\Theta$. The proposed model is evaluated for each even number of extracted features (e.g. 2, 4, ..., NF where NF is the maximum number of features (NFS) in the original space).

4.4.2 Multispectral Images

Landsat-5 TM images are MSI with 6 bands and 30 m ground resolution. Landsat images of the Lake Simcoe watershed in Ontario, Canada, were acquired through the data access portal (GloVis) of the United States Geological Survey (USGS). Scenes captured on May 4th, May 20, June 21, and July 7th, 2010 were downloaded and atmospherically corrected using the semi-automatic classification plugin for QGIS [132]. The images are then cloud- and shadow-masked using the FMASK tool developed by [133], and only cloud-free areas that overlap between the images were extracted and considered for further analysis. Figure 4.3 shows the downloaded images and the study area.

The spatial information in the images is enhanced by extending the standard set of bands us-

Figure 4.2: Pavia City: DA Scenario (top) true color composite, (bottom) ground-truth map and map keys.
ing texture filters. For a specific neighboring system $\mathcal{N}$ in the image, the spatial filters compute the value of the center pixel by applying a mathematical operation using all pixels in $\mathcal{N}$. We empirically tested different filters and found that the following six filters enhanced classification accuracy:

- mean: $\text{mean}(p) \; \forall \; p \in \mathcal{N}$
- erosion: $\text{min}(p) \; \forall \; p \in \mathcal{N}$
- range: $(\text{max}(p) - \text{min}(p)) \; \forall \; p \in \mathcal{N}$
- opening: dilation + erosion over $\mathcal{N}$
- dilation: $\text{max}(p) \; \forall \; p \in \mathcal{N}$
- entropy: information entropy $\forall \; p \in \mathcal{N}$

The neighboring system, $\mathcal{N}$, used for all filters is a $3 \times 3$ car box. These filters were applied to each of the original six bands separately, and the filtered bands were stacked with the original bands to obtain a final set of 42 bands. The ground truth data for the study area is provided by Melnychuk [134], where a field survey during the 2010 cropping season was conducted, covering
commonly grown crops in the study area. Figure 4.4 shows the true color composite (Bands 3, 2, 1) of a sample region from May 21 scene, along with the crop map for the same region. The study area covers about $400 \text{ km}^2$, and it is expected that samples from different regions in this area will exhibit a higher shift than the one found in the Pavia dataset. The crops considered in this dataset are listed in Table 4.3, in which 500 randomly selected samples per class are used for representation learning. An equal number of samples per class are drawn randomly for each of the ten realizations of the classification task.

The proposed framework is applied to this dataset using the following two scenarios:

**Scenario 1 (Spatial adaptation):** This study follows a similar scenario to that of the Pavia dataset, in which only a single image (captured at one date) is used. The source and target domains are shown in Figure 4.5.

**Scenario 2 (Temporal adaptation):** In this scenario, samples from a single image are used to train a model, which is then applied to another image acquired at a different date. This scenario would be more realistic if the data for different seasons or years was available; however, since the ground truth is only available for one year (2010), domain adaptation is simulated by selecting the source and target domains from the available images. Preliminary analysis shows that among the available Landsat scenes, the June 21st and July 7th images are the best candidates for crop...
classification. This is also confirmed in [134]; therefore, these two images are used as the source and target, respectively.

![Figure 4.5: Source (white) and target (red) domains in the Landsat images.](image)

Table 4.3: Classes used for Landsat dataset.

<table>
<thead>
<tr>
<th>Classes</th>
<th>Source ($X_S$)</th>
<th>Target ($X_T$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># of Pixels</td>
<td>%</td>
</tr>
<tr>
<td>Corn</td>
<td>5,802</td>
<td>25.35</td>
</tr>
<tr>
<td>Forage</td>
<td>5,789</td>
<td>25.29</td>
</tr>
<tr>
<td>Soybean</td>
<td>8,126</td>
<td>35.50</td>
</tr>
<tr>
<td>Winter Wheat</td>
<td>3,174</td>
<td>13.87</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>22,891</strong></td>
<td><strong>55,907</strong></td>
</tr>
</tbody>
</table>

4.5 Results

In this work we used the same metrics used by the RS community to evaluate RS pixel classification. These are: overall accuracy (OA), average accuracy (AA), and Cohen’s Kappa statistic [135]. OA represents the ratio between the total correctly classified samples and the total number of samples. AA is computed by averaging the accuracy of each class. OA and AA are equal when all
classes have the same number of samples. Cohen’s Kappa statistic is an estimated measure of the agreement between the classification results and the ground truth. Kappa ranges between -1 and 1: a value of 1 indicates complete agreement, a value of -1 implies complete disagreement and a value of 0 means random agreement (i.e., agreement by chance).

Ten independent realizations are performed on the training and test sets for all configurations used in this experiment. The mean and standard deviation for these metrics are computed for each of the ten repetitions. The number of features (NF) that provide the best mean accuracy over these tests, as well as the mean and standard deviation, are reported for each experiment. It should be clear that the NF represents the dimension of the new representation learned by the associated technique, and since there is no representation learning for the Src and Tgt baselines, NF is not reported for these baselines. It should be noted that the NF for DAE, DNN, and DANN, represent the number of neurons per layer in the network. For simplicity, the number of hidden units in each layer of a multi-layer network are kept the same.

### 4.5.1 HSI Images

Classification results are presented in Table 4.4. Both PCA and KPCA are not able to extract useful representations for the target domain. However, the representation learned using DAE outperformed the Tgt baseline, as well as the semi-supervised DA technique proposed in [67]. DANN, and a regular DNN trained on the source, (Src-DNN), outperformed the Tgt-LR. However, unlike DAE, DANN was not able to outperform the deep version of the Tgt baseline.

### 4.5.2 Multispectral Images: Scenario 1 (Spatial Adaptation)

The best classification results with the associated NF for the Landsat dataset, using the source and target from the same image, are provided in Tables 4.5 and 4.6, for the June and July images, respectively. For the June DA task, all of the two-stage representation learning techniques were able to improve on the Src baseline, and DAE provided the best improvement; but, none of these techniques were able to outperform the Tgt baseline. However, for the July task, only KPCA and DAE
CHAPTER 4. SINGLE SOURCE DOMAIN ADAPTATION

Table 4.4: DA classification results for the Pavia dataset.

<table>
<thead>
<tr>
<th>Method (NF)</th>
<th>Kappa Avg</th>
<th>Kappa Std</th>
<th>OA Avg</th>
<th>OA Std</th>
<th>AA Avg</th>
<th>AA Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Src-LR</td>
<td>0.845</td>
<td>0.013</td>
<td>0.918</td>
<td>0.005</td>
<td>0.857</td>
<td>0.046</td>
</tr>
<tr>
<td>Tgt-LR</td>
<td>0.860</td>
<td>0.006</td>
<td>0.920</td>
<td>0.004</td>
<td>0.952</td>
<td>0.001</td>
</tr>
<tr>
<td>PCA-1DOM (28)</td>
<td>0.770</td>
<td>0.030</td>
<td>0.876</td>
<td>0.019</td>
<td>0.762</td>
<td>0.041</td>
</tr>
<tr>
<td>PCA-2DOM (52)</td>
<td>0.821</td>
<td>0.022</td>
<td>0.905</td>
<td>0.011</td>
<td>0.824</td>
<td>0.057</td>
</tr>
<tr>
<td>KPCA-1DOM (24)</td>
<td>0.813</td>
<td>0.012</td>
<td>0.895</td>
<td>0.006</td>
<td>0.869</td>
<td>0.030</td>
</tr>
<tr>
<td>KPCA-2DOM (18)</td>
<td>0.824</td>
<td>0.012</td>
<td>0.904</td>
<td>0.006</td>
<td>0.852</td>
<td>0.026</td>
</tr>
<tr>
<td>DAE-1DOM (200)</td>
<td>0.868</td>
<td>0.005</td>
<td>0.924</td>
<td>0.003</td>
<td>0.932</td>
<td>0.007</td>
</tr>
<tr>
<td>DAE-2DOM (200)</td>
<td>0.863</td>
<td>0.005</td>
<td>0.921</td>
<td>0.003</td>
<td>0.937</td>
<td>0.002</td>
</tr>
<tr>
<td>SSTCA <a href="17">67</a></td>
<td>0.847</td>
<td>0.010</td>
<td>0.911</td>
<td>0.006</td>
<td>0.929</td>
<td>0.004</td>
</tr>
<tr>
<td>Src-DNN (50)</td>
<td>0.862</td>
<td>0.010</td>
<td>0.923</td>
<td>0.006</td>
<td>0.845</td>
<td>0.014</td>
</tr>
<tr>
<td>Tgt-DNN (50)</td>
<td>0.878</td>
<td>0.005</td>
<td>0.931</td>
<td>0.003</td>
<td>0.866</td>
<td>0.008</td>
</tr>
<tr>
<td>DANN (50)</td>
<td>0.868</td>
<td>0.005</td>
<td>0.926</td>
<td>0.003</td>
<td>0.854</td>
<td>0.010</td>
</tr>
</tbody>
</table>

were able to achieve improvement over the Src baseline. On the other hand, DNN outperformed
the Tgt baseline for both tasks, and DANN was able to provide comparable performance, but both
were not able to outperform the deep architecture Tgt baseline (e.g., Tgt-DNN).

Table 4.5: Crop mapping classification results from June.

<table>
<thead>
<tr>
<th>Method (NF)</th>
<th>Kappa Avg</th>
<th>Kappa Std</th>
<th>OA Avg</th>
<th>OA Std</th>
<th>AA Avg</th>
<th>AA Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Src-LR</td>
<td>0.357</td>
<td>0.015</td>
<td>0.532</td>
<td>0.011</td>
<td>0.512</td>
<td>0.011</td>
</tr>
<tr>
<td>Tgt-LR</td>
<td>0.578</td>
<td>0.009</td>
<td>0.689</td>
<td>0.007</td>
<td>0.714</td>
<td>0.007</td>
</tr>
<tr>
<td>PCA-1DOM (32)</td>
<td>0.448</td>
<td>0.034</td>
<td>0.598</td>
<td>0.026</td>
<td>0.584</td>
<td>0.024</td>
</tr>
<tr>
<td>PCA-2DOM (40)</td>
<td>0.442</td>
<td>0.045</td>
<td>0.596</td>
<td>0.034</td>
<td>0.576</td>
<td>0.038</td>
</tr>
<tr>
<td>KPCA-1DOM (40)</td>
<td>0.402</td>
<td>0.044</td>
<td>0.561</td>
<td>0.034</td>
<td>0.562</td>
<td>0.023</td>
</tr>
<tr>
<td>KPCA-2DOM (40)</td>
<td>0.392</td>
<td>0.017</td>
<td>0.554</td>
<td>0.012</td>
<td>0.549</td>
<td>0.014</td>
</tr>
<tr>
<td>DAE-1DOM (200)</td>
<td>0.502</td>
<td>0.014</td>
<td>0.640</td>
<td>0.012</td>
<td>0.604</td>
<td>0.006</td>
</tr>
<tr>
<td>DAE-2DOM (200)</td>
<td>0.498</td>
<td>0.014</td>
<td>0.637</td>
<td>0.011</td>
<td>0.604</td>
<td>0.007</td>
</tr>
<tr>
<td>Src-DNN (50)</td>
<td>0.599</td>
<td>0.009</td>
<td>0.710</td>
<td>0.007</td>
<td>0.707</td>
<td>0.007</td>
</tr>
<tr>
<td>Tgt-DNN (50)</td>
<td>0.685</td>
<td>0.004</td>
<td>0.774</td>
<td>0.003</td>
<td>0.766</td>
<td>0.004</td>
</tr>
<tr>
<td>DANN (50)</td>
<td>0.599</td>
<td>0.010</td>
<td>0.712</td>
<td>0.005</td>
<td>0.711</td>
<td>0.007</td>
</tr>
</tbody>
</table>

4.5.3 Multispectral Images: Scenario 2 (Temporal Adaptation)

Table 4.7 provides results based on the June and July images as the source and target, respectively.
The 40% difference on the overall accuracy between the Src-LR and Tgt-LR baselines reflects
CHAPTER 4. SINGLE SOURCE DOMAIN ADAPTATION

Table 4.6: Crop mapping classification results from July.

<table>
<thead>
<tr>
<th>Method (NF)</th>
<th>Kappa Avg</th>
<th>Kappa Std</th>
<th>OA Avg</th>
<th>OA Std</th>
<th>AA Avg</th>
<th>AA Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Src-LR</td>
<td>0.523</td>
<td>0.028</td>
<td>0.654</td>
<td>0.019</td>
<td>0.682</td>
<td>0.020</td>
</tr>
<tr>
<td>Tgt-LR</td>
<td>0.621</td>
<td>0.008</td>
<td>0.724</td>
<td>0.006</td>
<td>0.749</td>
<td>0.004</td>
</tr>
<tr>
<td>PCA-1DOM (26)</td>
<td>0.471</td>
<td>0.086</td>
<td>0.614</td>
<td>0.061</td>
<td>0.648</td>
<td>0.059</td>
</tr>
<tr>
<td>PCA-2DOM (18)</td>
<td>0.432</td>
<td>0.068</td>
<td>0.590</td>
<td>0.044</td>
<td>0.618</td>
<td>0.048</td>
</tr>
<tr>
<td>KPCA-1DOM (40)</td>
<td>0.514</td>
<td>0.020</td>
<td>0.638</td>
<td>0.016</td>
<td>0.677</td>
<td>0.014</td>
</tr>
<tr>
<td>KPCA-2DOM (40)</td>
<td>0.529</td>
<td>0.027</td>
<td>0.653</td>
<td>0.024</td>
<td>0.682</td>
<td>0.011</td>
</tr>
<tr>
<td>DAE-1DOM (200)</td>
<td>0.609</td>
<td>0.016</td>
<td>0.713</td>
<td>0.012</td>
<td><strong>0.743</strong></td>
<td>0.011</td>
</tr>
<tr>
<td>DAE-2DOM (200)</td>
<td>0.609</td>
<td>0.015</td>
<td>0.712</td>
<td>0.012</td>
<td><strong>0.743</strong></td>
<td>0.010</td>
</tr>
<tr>
<td>Src-DNN (50)</td>
<td>0.689</td>
<td>0.006</td>
<td>0.776</td>
<td>0.004</td>
<td>0.765</td>
<td>0.005</td>
</tr>
<tr>
<td>Tgt-DNN (50)</td>
<td>0.755</td>
<td>0.005</td>
<td>0.823</td>
<td>0.004</td>
<td>0.815</td>
<td>0.005</td>
</tr>
<tr>
<td>DANN (50)</td>
<td><strong>0.692</strong></td>
<td>0.005</td>
<td><strong>0.777</strong></td>
<td>0.003</td>
<td><strong>0.766</strong></td>
<td>0.006</td>
</tr>
</tbody>
</table>

Interestingly, all of the two-stage approaches were able to improve on the Src baseline, and again, DAE provided the best improvement. Unlike previous tasks, the deep architecture approaches were not able to outperform the Tgt-LR baseline, and in particular, the Src-DNN suffered from over-fitting and was not even able to outperform PCA and DAE. Despite this poor performance for the deep architecture, the adversarial version (DANN) was able to outperform all the other techniques.

Table 4.7: Crop mapping multi-temporal classification.

<table>
<thead>
<tr>
<th>Method (NF)</th>
<th>Kappa Avg</th>
<th>Kappa Std</th>
<th>OA Avg</th>
<th>OA Std</th>
<th>AA Avg</th>
<th>AA Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Src-LR</td>
<td>0.130</td>
<td>0.031</td>
<td>0.360</td>
<td>0.026</td>
<td>0.333</td>
<td>0.019</td>
</tr>
<tr>
<td>Tgt-LR</td>
<td>0.672</td>
<td>0.012</td>
<td>0.762</td>
<td>0.009</td>
<td>0.766</td>
<td>0.008</td>
</tr>
<tr>
<td>PCA-1DOM (14)</td>
<td>0.271</td>
<td>0.017</td>
<td>0.451</td>
<td>0.014</td>
<td>0.485</td>
<td>0.015</td>
</tr>
<tr>
<td>PCA-2DOM (12)</td>
<td>0.285</td>
<td>0.014</td>
<td>0.465</td>
<td>0.011</td>
<td><strong>0.496</strong></td>
<td>0.013</td>
</tr>
<tr>
<td>KPCA-1DOM (42)</td>
<td>0.253</td>
<td>0.028</td>
<td>0.452</td>
<td>0.017</td>
<td>0.437</td>
<td>0.036</td>
</tr>
<tr>
<td>KPCA-2DOM (42)</td>
<td>0.255</td>
<td>0.030</td>
<td>0.451</td>
<td>0.022</td>
<td>0.446</td>
<td>0.027</td>
</tr>
<tr>
<td>DAE-1DOM (200)</td>
<td><strong>0.296</strong></td>
<td>0.029</td>
<td><strong>0.482</strong></td>
<td>0.020</td>
<td>0.486</td>
<td>0.028</td>
</tr>
<tr>
<td>DAE-2DOM (200)</td>
<td>0.295</td>
<td>0.029</td>
<td>0.481</td>
<td>0.020</td>
<td>0.486</td>
<td>0.029</td>
</tr>
<tr>
<td>Src-DNN (50)</td>
<td>0.261</td>
<td>0.030</td>
<td>0.454</td>
<td>0.024</td>
<td>0.436</td>
<td>0.019</td>
</tr>
<tr>
<td>Tgt-DNN (50)</td>
<td>0.800</td>
<td>0.004</td>
<td>0.856</td>
<td>0.003</td>
<td>0.848</td>
<td>0.004</td>
</tr>
<tr>
<td>DANN (50)</td>
<td><strong>0.304</strong></td>
<td>0.033</td>
<td><strong>0.496</strong></td>
<td>0.026</td>
<td><strong>0.449</strong></td>
<td>0.031</td>
</tr>
</tbody>
</table>

Surprisingly, under several conditions, Src-DNN was able to achieve a performance comparable to the DA techniques. This confirms the ability of the deep learning techniques, even without
CHAPTER 4. SINGLE SOURCE DOMAIN ADAPTATION

using any adaption mechanism, to extract useful discriminatory features that are robust and can naturally adapt to changes in the data distribution [84]. This can be observed in all multi-spatial scenarios (Tables 4.4, 4.5, 4.6) and is clearly evident with the HSI case. As the difficulty of the DA task increases, however, Src-DNN over-fits the source data and performs poorly on the target data, as in the multi-temporal case (Table 4.7). We, therefore, argue that in situations where domain mismatch is extreme, a learning approach designed explicitly for DA is required.

4.5.4 Computation Time

All reported results were obtained using an Intel Core i7 3770K Quad-core CPU (32GB RAM) machine powered with Nvidia GTX Titan Black GPUs, each with 6GB memory. GPUs are only utilized by the neural network-based approaches. Theano and Caffe deep learning libraries were used to implement the DAE and DANN, respectively, while PCA and KPCA were realized using the scikit-learn library. Table 4.8 reports the representation learning and mapping run time (in seconds) for both the HSI and the MSI experiments. Training time is defined as the time required by the algorithm to learn the mapping function ($\Phi$), while mapping time is the time required to apply the mapping function to the target samples. For easy reference, the number of samples used for training and mapping for both experiments is also included in Table 4.8. Due to its simplicity, PCA is the fastest and is mainly affected by the number of bands in the image (102 bands for HSI versus 42 bands for MSI). Unlike PCA, KPCA is not scalable, and the number of samples severely affects the run time, as well as the memory requirements, as it is required to compute a matrix of order $O(N^2)$ during both the training and mapping stages.

For the neural network-based techniques, all reported times are based on 250 training epochs. As anticipated, the training time for the DAE (one hidden layer, 200 units) is significantly higher than that required for both PCA and KPCA. However, the DAE mapping time is much faster than with KPCA (18× faster for the MSI case). Although DNN and DANN have the same network depth (3 hidden layers), DANN uses samples from the target domain during training and therefore

\[^2\]6,250 iterations for HSI, and 15,625 iterations for MSI.
CHAPTER 4. SINGLE SOURCE DOMAIN ADAPTATION

Table 4.8: Training and mapping time of the existing and proposed techniques.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>HSI (800:14,048)</th>
<th>MSI (2000:17,954)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Train Time</td>
<td>Map Time</td>
</tr>
<tr>
<td>PCA</td>
<td>0.008</td>
<td>0.008</td>
</tr>
<tr>
<td>KPCA</td>
<td>0.174</td>
<td>0.668</td>
</tr>
<tr>
<td>DAE</td>
<td>9.225</td>
<td>0.155</td>
</tr>
<tr>
<td>DNN</td>
<td>6.023</td>
<td>-</td>
</tr>
<tr>
<td>DANN</td>
<td>10.057</td>
<td>-</td>
</tr>
</tbody>
</table>

\(^a\) (Number of training samples : Number of mapping samples)

requires twice the amount of data as input during training. This resulted in a much faster DNN (where the DNN is about \(1.6\times\) faster than DANN for both experiments).

4.6 Discussion

It is clear from the results that representation learning techniques can be applied successfully to the remote sensing DA problem. However, several factors can affect the performance of such techniques, which we address in the following subsections.

4.6.1 Challenges Encountered For Domain Adaptation

It is apparent from the multi-spatial DA setting that DAE, DNN, and DANN are successful in extracting useful representations for the classification task. However, performance can be greatly affected by the level of difficulty of the task. For example, in the HSI dataset, the image’s original features are rich in both the spectral and spatial domains (102 bands versus 6 bands; 1.5 m versus 30 m). Also, the nature of the task is more challenging for the MSI, as the classes in these images belong to different families (roads, vegetation, shadows, and buildings), while the MSI image classes are from the same family (i.e., vegetation). Furthermore, the regions corresponding to the source and target are more distinct spatially in the multi-spatial case of the crop mapping problem than in HSI.

To further clarify this, we computed the producer’s accuracy (PA) and the user’s accuracy
CHAPTER 4. SINGLE SOURCE DOMAIN ADAPTATION

(UA) for the MSI tasks. The PA (i.e., precision) for a particular class is the fraction of correctly classified samples with respect to all ground truth samples, i.e., True Positive/(True Positive + False Positive). Therefore, this metric measures the ability of the classifier to correctly retrieve the corresponding class. On the other hand, UA (i.e., recall) for a particular class is the fraction of correctly classified samples with respect to the total number of samples assigned by the classifier to this class, i.e., True Positive/(True Positive + False Negative). Therefore, this metric indicates the reliability of the classifier in retrieving the class with respect to the misclassification measured for that particular class. The quality of the final classification results are affected by these metrics, and therefore variations in these metrics for a single class will result in poor overall performance, especially for Kappa and AA. Table 4.9 shows these metrics for a single run of the MSI experiment.

Table 4.9: User and producer accuracy.

<table>
<thead>
<tr>
<th>Crop</th>
<th>June UA%</th>
<th>June PA%</th>
<th>July UA%</th>
<th>July PA%</th>
<th>June→July UA%</th>
<th>June→July PA%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corn</td>
<td>69.78</td>
<td>46.39</td>
<td>73.67</td>
<td>79.57</td>
<td>30.71</td>
<td>34.16</td>
</tr>
<tr>
<td>Forage</td>
<td>72.81</td>
<td>77.74</td>
<td>75.86</td>
<td>58.69</td>
<td>52.92</td>
<td>68.01</td>
</tr>
<tr>
<td>Soybean</td>
<td>49.72</td>
<td>84.08</td>
<td>59.88</td>
<td>70.07</td>
<td>60.08</td>
<td>42.23</td>
</tr>
<tr>
<td>Winter Wheat</td>
<td>69.17</td>
<td>34.72</td>
<td>73.31</td>
<td>80.26</td>
<td>56.88</td>
<td>58.85</td>
</tr>
</tbody>
</table>

Considering the multi-spatial DA task for June and July, it is clear that most of the confusion occurs in June, rather than in July. In June, Forage is the only class that produces similar values for both the UA and PA, as this crop is maintained throughout the season and has less variability. Most of the confusion occurs in the winter wheat class, as it is fully grown by the end of June, and some fields have been already harvested, resulting in misclassification of these instances. Additionally, corn and soybeans are still in an early development stage at this time, rendering samples from both classes indistinguishable.

The multi-temporal DA task was the most challenging task, and the corn class contributed greatly to these poor results. As the corn significantly emerged after the 16 days difference between the two images, none of the DA techniques were able to cope with this massive shift in the data distribution. The same observation applies to soybeans, but to a lesser degree, as the soybean canopy does not grow as large as corn does during this period [136].
CHAPTER 4. SINGLE SOURCE DOMAIN ADAPTATION

4.6.2 Availability of Samples From The Target Domain

The availability of samples from the target domain during training is also a concern for some DA tasks, such as disaster monitoring and damage assessments, where only samples from the source domain are available for representation learning and training of a classifier. A DANN would not be suitable for such a setup; however, DAE and DNN can be used for this, as they do not require samples from the target domain during training. In fact, DAE has demonstrated robust performance in all experiments with respect to the origin of samples, as it provides stable results in both the 1DOM and 2DOM settings.

4.6.3 Scalability

In practical remote sensing applications, the number of samples is massive, and therefore scalability is a major concern. Kernel-based approaches, such as KPCA, are not an option for such large-scale tasks since they are based on solving eigen decomposition problem using dense matrices. In addition to their robust performance, deep learning techniques are a promising option for tackling this problem. Furthermore, the recent significant development in computing, storage, and deep learning libraries, have caused an enormous reduction in the training time for large-scale applications [30]. The reported computing times, although only utilizing a single GPU and not a distributed system, confirm the applicability of using deep learning for large-scale remote sensing DA tasks.

4.7 Summary

In this chapter we propose a single-source domain adaptation techniques based on deep learning. These techniques aims to find invariant representation for both domains in unsupervised fashion. The obtained results confirm the effectiveness of these techniques in different domain adaptation scenarios. In the next chapter we will address the multi-source domain adaptation problem in remote sensing.
Chapter 5

Multi-source Domain Adaptation using DNNs

5.1 Introduction

In this chapter we address the Multi-source Domain Adaptation problem (MDA). Often multiple training data sources, are available for the same task. It is sample inefficient to train separate models per domain, which motivates learning a single model from multiple sources. multi-source domain adaptation is a challenging problem and is particularly apparent when there are significant changes in the data distribution among these sources. Utilizing single source will not guarantee acceptable results, therefore, it is necessary to utilize all the sources. However, as discussed in Chapter 3, using a uniform combination of all the source to train a single model is also likely to fail, thus, it is important to develop a method to utilize all the sources and re-weighting each source contribution to the final classification model.

In this work, we propose a simple end-to-end framework that is efficient for large-scale MDA. We also design and implement different MDA techniques based on CNNs to solve remote sensing classification problems. In particular, we target the recently developed Local Climate Zone (LCZ) [137] classification problem. The LCZ mapping schema delivers generic classification maps
for urban and rural landscapes. The schema consists of 17 classes, reflecting the structural and land cover properties that affect air temperature. Currently, the World Urban Database and Portal Tool (WUDPT) [138] uses crowdsourcing to gather a census of cities around the world, and the LCZ framework has been selected as the mapping schema for characterizing cities in a consistent manner. In order to develop these LCZ maps, the WUDPT process employs supervised machine learning techniques to obtain each city’s full LCZ map. However, the system relies completely on non-expert users to label the training samples for every particular city, and a verification process must be performed to evaluate the quality of each map\(^1\).

Our work on MDA is proposed as a candidate tool for a more productive and accurate LCZ mapping system. The proposed techniques aim to reduce the time required to generate these maps, and also reduce the uncertainty resulting from crowdsourcing. The main contributions of this work can be summarized as follows:

- We propose a novel end-to-end Adaptive Multi-source Domain Adaptation (AMDA) technique that is simple, robust, and capable of handling imbalanced large-scale multi-source problems.
- We also extend two recently introduced domain expansion algorithms for multi-source DA.
- Finally, we implement and empirically evaluate the proposed techniques on a large-scale public remote sensing benchmark: the LCZ classification problem.

The remainder of this chapter is organized as follows: The proposed techniques are highlighted in Section 5.2. Section 5.3 describes the experiment details and the data used. Results and analysis are presented in Section 5.4. Finally, the conclusion remarks and possible future works are covered in Section 5.5.

\(^1\)http://www.wudapt.org/
5.2 Methodology

In this section we describe the proposed framework and our architectural choices. The proposed framework, as illustrated in Figure 5.1, consists of training and prediction activities that are found in any typical machine learning pipeline. Since we are addressing remote sensing pixel-based classification tasks, we have added a patch extraction activity in the pipeline to translate the provided image(s) into a set of image patches which can then be used to train the DNN model. This should allow users to employ Convolutional Neural Networks (CNNs) which typically operate on images.

![Diagram of the proposed multi-source domain adaptation framework for LCZs mapping.](image)

(a) Training stage.

(b) Deployment stage.

Figure 5.1: The proposed multi-source domain adaptation framework for LCZs mapping.

5.2.1 Architecture

Next, we describe the proposed DA networks, which can be trained using the extracted patches from the source images. These networks are then used for prediction, and potentially re-trained or fine-tuned when more training sources become available. An important attribute of these networks is that there are no restrictions on the number of source domains, and the network size or complexity will not be affected by the number of domains. Furthermore, these networks use common components and thus are easy to implement using any deep learning library such as PyTorch [139] or TensorFlow [140].

The baseline network we consider here is a convolutional Neural network (CNN), as shown in Figure 5.2. The network consists of multiple convolutional blocks and fully-connected layers. The output vector of the network is the softmax of the last fully-connected layer. The network
parameters \( \theta \) are optimized to minimize the classification loss \( \mathcal{L}_y(\theta) \) with respect to the network parameters \( \theta \). The optimized parameters of the network \( (\hat{\theta}) \) can be found with:

\[
\hat{\theta} = \arg\min_{\theta} \mathcal{L}_y(\theta). \tag{5.1}
\]

The classification loss, \( \mathcal{L}_y \), is computed using the classical cross-entropy given by the following expression:

\[
\mathcal{L}_y = -H(y, \hat{y}) = -\sum_{i=1}^{m} y_i \cdot \log(\hat{y}_i), \tag{5.2}
\]

\[
\hat{y} = f(x, \theta), \tag{5.3}
\]

where \( y \) and \( \hat{y} \) are one-hot encoded vectors for the ground truth and the network output, respectively, and \( f \) is a function parameterized by \( \theta \) that maps the input \( x \) to the output \( \hat{y} \). The size of the output vector is \( m \), where \( m \) is the number of class labels in the task. This loss is computed using a mini-batch of \( (x, y) \) pairs sampled from the training data. The network output is computed by applying a softmax to the last fully-connected layer output vector \( (z) \):

\[
\hat{y}(z)_j = \sigma(z)_j = \frac{\exp(z_j)}{\sum_{k=1}^{m} \exp(z_k)} \quad \forall \ j = 1,..m. \tag{5.4}
\]
CHAPTER 5. MULTI-SOURCE DOMAIN ADAPTATION USING DNNS

The network parameters ($\theta$) are updated using Stochastic Gradient Descent (SGD), as follows:

$$\hat{\theta} = \theta - \mu \cdot \frac{\partial L_y}{\partial \theta},$$  \hspace{1cm} (5.5)

where $\mu$ is the learning rate, and $\frac{\partial L_y}{\partial \theta}$ is the partial derivative of $L_y$ with respect to the network parameters $\theta$.

All of the proposed methods follow the same objective, and all use SGD to update the network parameters. Similar to the distribution matching approaches discussed earlier, we also add a domain adaptation loss term to the network objective. Using the same notation we use for the baseline network, the domain adaptation network objective can be expressed as:

$$L(x, y, \theta) = \lambda_y L_y(x, y, \theta) + \lambda_{DA} L_{DA}(x, y, \theta),$$  \hspace{1cm} (5.6)

where $\lambda_y$ and $\lambda_{DA}$ are control parameters for the influence of the classification loss and the DA loss, respectively. Although all proposed methods use Equation (5.6), they differ in how the $L_{DA}$ term is computed. However, the classification loss ($L_y$) is identical across all the methods and is computed according to Equation (5.2).

In this study, we investigate several approaches to train this network, including PNN and DANN. However, they all fail to enhance the baseline results, and we will only discuss and focus on the most promising techniques. In the next section we will introduce the domain expansion methods and how we modify them to be suitable for MDA. Our proposed technique is discussed in the following section (Sec. 5.2.3).

5.2.2 Adaptation by Knowledge Expansion Networks

In this section, we focus on two recently introduced domain expansion algorithms, which we modify to be used for DA. The common theme for these models is that the network parameters are progressively updated, and the network objective includes a term to overcome catastrophic forgetting.
CHAPTER 5. MULTI-SOURCE DOMAIN ADAPTATION USING DNNS

5.2.2.1 Learning without Forgetting

Li and Hoiem [118] proposed LwF for domain expansion (DE), which aim to avoid catastrophic forgetting and to generate a unified model that performs well with both tasks. In the typical DE setting, the old task training data is not available, however, the pretrained network that learned with the old task data is available. This is the network that DE aims to expand.

\[
\begin{align*}
\theta_s & \quad \text{Shared representation learning layers} \\
\theta_o & \quad \text{Task specific layer(s)} \\
\hat{y}_{\text{old}} & = f(x_{\text{new}}; \theta_s, \theta_o) \\
\end{align*}
\]

Figure 5.3(a) illustrates how LwF achieves this. The network trained on the old task (start-up network), \( \theta_o \), is divided into shared representation learning layers (\( \theta_s \)), and task specific layer(s) (\( \theta_o \)). LwF expands this network for the new task by creating a new layer(s) for the new task (\( \theta_n \)). First, using the new data, \( x_{\text{new}} \), the old task response is recorded as follows:

\[
y_{\text{old}} = f(x_{\text{new}}; \theta_s, \theta_o)
\]  \hspace{1cm} (5.7)

It is important to note that the recorded \( y_{\text{old}} \) values do not represent the crisp label for the old task, but rather the posterior probabilities of the task labels based on the new task data. However, it will be treated as if it is the actual old task ground-truth and is never updated. \( y_{\text{old}} \) is then added the available new task training data pairs, \( (x_{\text{new}}, y_{\text{new}}) \), to form a new training set, \( D = \)
\( \{x_{\text{new}}, (y_{\text{new}}, y_{\text{old}})\} \).

The new layers, \( \theta_n \), are randomly initialized, and the entire network parameters, \( (\theta_s, \theta_o, \theta_n) \), are optimized to jointly minimize the loss with respect to the new task ground-truth \( (y_{\text{new}}) \) and the loss with respect to the recorded ground-truth \( (y_{\text{old}}) \), which can expressed as:

\[
\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n = \arg \min_{\theta_s, \theta_o, \theta_n} \mathcal{L}_{\text{new}}(x_{\text{new}}; \theta_s, \theta_n) + \lambda_{\text{old}} \cdot \mathcal{L}_{\text{old}}(x_{\text{new}}; \theta_s, \theta_o).
\]

where \( \mathcal{L}_{\text{old}} \) is the old task loss, \( \mathcal{L}_{\text{new}} \) is the new task loss, and \( \lambda_{\text{old}} \) is a hyper-parameter to control the influence of the old loss. The cross-entropy loss in Equation (5.2) is used to compute both losses.

In our work, we adopt the same strategy used by LwF for multi-source DA as shown in Figure 5.3(b). Because the label space of all the domains is the same, \( \theta_n \) and \( \theta_o \) are identical in terms of size and architecture. Furthermore, since we will be optimizing the network over multiple domains, we need to efficiently compute the old network’s response with respect to the new domain data. Instead of re-computing and recording \( y_{\text{old}} \) whenever we change the input data, we create a separate branch for the old network’s parameters \( (\theta_{s(\text{old})}, \theta_o) \) and freeze them while optimizing the new network parameters \( (\theta_s, \theta_n) \) using the new domain data. This allows us to recompute \( y_{\text{old}} \) on the fly and there is no need to update it and record it for the entire domain data whenever we iterate to a different domain. Furthermore, for DA we would like the old network and the updated network to maintain the same response for a given input, therefore, we modify \( \mathcal{L}_{\text{old}} \), \( (\mathcal{L}_{\text{DA}} \text{ is used to replace } \mathcal{L}_{\text{old}}) \), such that it encourages the updated new network output to be the same as the frozen old network output. Therefore, \( \mathcal{L}_{\text{DA}} \) can be expressed as:

\[
\mathcal{L}_{\text{DA}} = -H(\hat{y}_{\text{new}}, \hat{y}_{\text{old}}) = - \sum_{i=1}^{m} \hat{y}_{\text{new}(i)} \log(\hat{y}_{\text{old}(i)}).
\]

Finally, we further change the LwF training protocol to re-iterate through all the source domains as we will discuss later.
5.2.2.2 Less-forgetful Learning

Unlike LwF, Less-forgetful learning (LFL) [119], shown in Figure 5.4(a), uses different objectives to avoid catastrophic forgetting. First, it freezes the last hidden layer ($\theta_{L-1}$) before applying the softmax function, which ensures fixing the classifier decision boundaries (we will refer to the last layer of the network as the logits layer in the remainder of the chapter). The second objective involves encouraging the old network’s extracted features ($F_o$) to be similar to the new network extracted features ($F_n$). Using the above notation, $F_n$ are computed as follows:

$$F_o = f(x_{\text{new}}; \theta_{L-1}^{(1..L-1)})$$ (5.10)

$$F_n = f(x_{\text{new}}; \theta_n^{(1..L-1)})$$ (5.11)

where $f$ is the mapping function parameterized by $\theta_{L-1}^{(1..L-1)}$ which maps an input $x$ to the hidden representation of layer $L-1$. $L_2$ norm is used to compute the similarity between $F_o$ and $F_n$ as follows:

$$\mathcal{L}_e = \frac{1}{2} || F_n - F_o ||_2^2.$$ (5.12)

Similar to LwF, the start-up network is used for expansion, and $\theta_n^{(1..L-1)}$ is optimized such that:

**Figure 5.4: Less-forgetful Learning.**
\[
\hat{\theta}_{n}^{(1:L-1)} = \arg\min_{\theta_{n}^{(1:L-1)}} \mathcal{L}_{\text{new}}(x_{\text{new}}; \theta_{n}^{(1:L-1)}, \theta_{o}^{(L)}) + \lambda_{e} \cdot \mathcal{L}_{e}(x_{\text{new}}; \theta_{n}^{(1:L-1)}, \theta_{o}^{(1:L-1)}) \tag{5.13}
\]

Unlike LwF, by fixing the last layer, LFL constrains the number of classes for both tasks to be identical. Furthermore, the DE scenarios used to evaluate LFL [119] can be considered as a DA problems. Therefore, LFL are adopted for DA without any modification and \( \mathcal{L}_{e} \) in Equation (5.12) will be used to replace \( \mathcal{L}_{\text{DA}} \) in our domain adaptation objective in Equation (5.6).

Although \( \mathcal{L}_{e} \) induces the extracted features of the two tasks to be similar, we consider that this objective is implicitly included when the logits layer, \( \theta_{o}^{(L)} \), is fixed. Fixing \( \theta_{o}^{(L)} \) will force the network to comply with the given decision boundaries when optimizing for the new domains, which in turn results in encouraging the new network to produce similar features to the features produced by the old network. Consequently, we also investigate the effect of discarding the DA loss (e.g. \( \mathcal{L}_{\text{DA}} = 0 \)), and just fixing the logits layer as shown in Figure 5.4(b). We refer to this method as LFL-Fix, and the method with the explicit DA loss (e.g, \( \mathcal{L}_{\text{DA}} = \mathcal{L}_{e} \)) as LFL-L2.

Since both the LwF and LFL algorithms are modified for DA, an iterative training approach is used. The basic training cycle for a given source domain data is shown in Algorithm 1. In each training cycle, only a single source is used to update the network as described earlier. During this cycle, \( k \) SGD updates are performed by randomly sampling a mini-batch from the provided source data \( S \).

The main challenge we encounter when modifying LwF and LFL for domain adaptation is the type of control signal (or indicator) to use for detecting improvements in the network with respect to all the sources. We investigate several scores (metrics) as a control signal based on a held-out validation set, such as average loss, max loss, average accuracy, and minimum accuracy. Among these scores, we find that maximizing the minimum accuracy within the sources provides the most stable and accurate control signal. Assuming that the validation accuracy of the \( i^{th} \) source is \( a_{i} \),
CHAPTER 5. MULTI-SOURCE DOMAIN ADAPTATION USING DNNS

Algorithm 1 Knowledge Expansion Networks (LwF, LFL) - Basic training cycle

**Inputs:**
- **S:** Single domain training data
- **k:** Number of SGD updates.
- **Val[i]:** validation set from all source (i:1 .. N)
- **θ_o, θ_n:** Old, new networks

**Outputs:**
- **θ_n:** Updated network parameters
- **ϕ:** Validation score

1: \( θ_n \leftarrow \text{SGD}(θ_o, θ_n, S, k) \) // Do \( k \) SGD updates using data from \( S \)
2: \( ϕ \leftarrow \text{Evaluate}(θ_n, \text{Val}) \) // Evaluate and compute the val. score
3: Return \( θ_n, ϕ \)

then the validation score, \( ϕ \), can be computed as follows:

\[
ϕ = \min_{a \in a_i} (a) \tag{5.14}
\]

The held-out validation set is extracted from all training sources and never used for training. In order to maximize the algorithm training speed, the size of this validation set should be relatively small. (e.g. 5% - 10% of the source training data). The algorithm also assumes the availability of a start-up network, \( (θ_o) \), pretrained on single source. At the end of each training cycle the network is evaluated using the validation sets (step 2 in Algorithm 1), and the validation score is computed according to Equation (5.14). If improvement is detected then the old network parameters are updated with the current network parameters, and this network will be saved and considered as the best optimized network. Finally, the algorithm terminates if no enhancement to the validation score has been reported for a consecutive predefined number of training cycles. It should be noted that the same algorithm is used for both LwF and LFL-L2, however, LFL-Fix does not require to maintain the old network parameters \( θ_o \) during adaptation and also the logits layer, \( θ_n^{(L)} \), is frozen and never updated.
5.2.3 Adaptive Multi-source DA (AMDA)

When addressing DA, it is important to recall that: 1) the data from all source domains is available during adaptation, and 2) the task is identical across all domains. These main characteristics motivated us to develop the Adaptive Multi-source DA (AMDA), where we designed a simple end-to-end multi-source DA algorithm. The AMDA network is shown in Figure 5.5, and consists of \( N \) branches, where \( N \) is the number of sources, and all branches share the same parameters. The loss of each branch is computed using the regular cross-entropy as in Equation (5.2), however, the entire network loss is computed as a weighted sum of these \( N \) losses as follows:

\[
\mathcal{L} = \sum_{i=1}^{N} \lambda_i \mathcal{L}_i,
\]

where \( \lambda_i \) is the loss weight of the \( i^{th} \) branch. The network performance on the validation set is used to compute \( \lambda_i \) as follows:

\[
\lambda_i = \frac{1}{a_i}; \quad a_i \in \mathbb{R} \mid 0 < a_i \leq 1,
\]

where \( a_i \) is the validation accuracy of the \( i^{th} \) source. Equation (5.16) sets a higher weight for low-performance branches, and therefore the resulting gradient from those branches will be magnified accordingly. On the other hand, the loss of branches with superior performance will not be magnified. As a result, branches with low-performance will have a higher influence on the updated network parameters.

The training protocol for AMDA is shown in Algorithm 2. The loss weights are initialized uniformly for all branches and are set to \( \frac{1}{N} \) (Step 1). Similar to the modified LwF and LFL algorithms, The actual training is performed in Step 5, where \( k \) steps of SGD are performed, and each SGD step optimizes the network parameters using a mini-batches sampled from all sources. The loss weights are fixed during these \( k \) SGD updates, and after these updates, the network is evaluated using the validation set of all the source. during this step, Step 6, the validation score and
CHAPTER 5. MULTI-SOURCE DOMAIN ADAPTATION USING DNNS

![Diagram of AMDA network architecture for N domains (x₁, x₂, ..., xₙ).](image)

the loss weights are computed according to Equation (5.14) and Equation (5.16) respectively. If an improvement is detected (Step 8), the current network parameters are saved and considered the best network so far (step 10). This training process is repeated as long as there are improvements in the validation score of the source domains. The algorithm terminates when no improvement is reported for a predefined number of consecutive iterations.

**Algorithm 2** Adaptive Multi-source DA (AMDA)

**Inputs:**
- \( \text{Tr}[i] \): Source domains training data
- \( \text{Val}[i] \): validation set from all source \((i: 1 \ldots N)\)
- \( k \): Number of SGD updates.

**Outputs:**
- \( \hat{\theta} \): Best network parameters
- \( \hat{\varphi} \): Validation score

1: \( \lambda_i \leftarrow \frac{1}{N} \)  // Initialize the loss weights
2: \( \theta \leftarrow \text{InitilizeNetwork()} \)  
3: \( \hat{\varphi}, \alpha_i \leftarrow \text{Evaluate}(\theta, \text{Val}) \)  // Compute the validation accuracy
4: \textbf{while} (not termination criteria) \textbf{do}
5: \( \theta \leftarrow \text{SGD}(\theta, \text{Tr}, k) \)  // Do \( k \) SGD updates using all sources
6: \( \lambda_i, \varphi \leftarrow \text{Loss}(\theta, \text{Val}) \)  // Compute the val. score & loss weights
7: \( \theta \leftarrow \text{UpdateLoss}(\lambda_i) \)  // Update the network loss weights
8: \textbf{If} (\( \varphi > \hat{\varphi} \)) \textbf{then}
9: \( \hat{\varphi} \leftarrow \varphi \)  // Update best val. score
10: \( \hat{\theta} \leftarrow \theta \)  // Create checkpoint
11: \textbf{Return} \( \hat{\theta}, \hat{\varphi} \)

98
CHAPTER 5. MULTI-SOURCE DOMAIN ADAPTATION USING DNNS

The main properties of the AMDA algorithm that distinguish it from the previously proposed algorithms are its implementation simplicity and its small number of parameters as they are shared among all the branches. Also, it is perfectly aligned with the argument made by Mansour et al. [48] that a model based on a weighted combination of the source distributions is better than a model based on a uniform combination of sources. Based on this observation, the baseline network is considered similar to AMDA, but uses a uniform combination of the sources (e.g., the loss weight for all the sources is fixed to $\lambda_i = \frac{1}{N}$).

5.3 Experimental setup

In this section, we describe the dataset and experiments used to evaluate the proposed methods. All proposed methods are implemented using the TensorFlow open source machine learning library [140], and all experimental results reported are obtained using an Intel Core i7 3770K Quad-core CPU (32GB RAM) machine powered with Nvidia GTX Titan Black GPUs, each with 6GB memory.

5.3.1 Dataset

The dataset used in this work was taken from the IEEE Data Fusion Contest in 2017 (DFC2017), which was organized by the Data Fusion Technical Committee of the IEEE Geoscience and Remote Sensing Society [114]. The contest involved classification of Local Climate Zones (LCZs) using free, open source data. The training data consists of five LCZ classes from five different cities (Berlin, Hong Kong, Paris, Rome, and Sao Paulo). The class distribution for the training cities is provided in Table 5.1. The contest challenge is to predict the LCZ maps for four test cities (Amsterdam, Chicago, Madrid, and Xi’an). In addition to the ground-truth data, multispectral images from Landsat 8 and Sentinel-2 are also provided for all the cities. Furthermore, users were encouraged to use other types of data for the classification task, as long as the data was from free and open sources. As an example of such data, the user is provided with auxiliary data from
Table 5.1: LCZs data - DFC2017 provided ground-truth (# of pixels).

<table>
<thead>
<tr>
<th>LCZ Class label</th>
<th>Berlin</th>
<th>Hong Kong</th>
<th>Paris</th>
<th>Rome</th>
<th>Sao Paulo</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compact high-rise (LCZ 1)</td>
<td>0</td>
<td>631</td>
<td>56</td>
<td>0</td>
<td>955</td>
<td>1,642</td>
</tr>
<tr>
<td>Compact midrise (LCZ 2)</td>
<td>1,534</td>
<td>179</td>
<td>2,705</td>
<td>1,551</td>
<td>134</td>
<td>6,103</td>
</tr>
<tr>
<td>Compact low-rise (LCZ 3)</td>
<td>0</td>
<td>326</td>
<td>0</td>
<td>104</td>
<td>5,308</td>
<td>5,738</td>
</tr>
<tr>
<td>Open high-rise (LCZ 4)</td>
<td>577</td>
<td>673</td>
<td>366</td>
<td>0</td>
<td>482</td>
<td>2,098</td>
</tr>
<tr>
<td>Open midrise (LCZ 5)</td>
<td>2,448</td>
<td>126</td>
<td>446</td>
<td>1,495</td>
<td>244</td>
<td>4,759</td>
</tr>
<tr>
<td>Open low-rise (LCZ 6)</td>
<td>4,010</td>
<td>120</td>
<td>2,419</td>
<td>480</td>
<td>1,862</td>
<td>8,891</td>
</tr>
<tr>
<td>Large low-rise (LCZ 8)</td>
<td>1,654</td>
<td>137</td>
<td>748</td>
<td>435</td>
<td>1,915</td>
<td>4,889</td>
</tr>
<tr>
<td>Sparsely built (LCZ 9)</td>
<td>761</td>
<td>0</td>
<td>60</td>
<td>0</td>
<td>335</td>
<td>1,156</td>
</tr>
<tr>
<td>Heavy industry (LCZ 10)</td>
<td>0</td>
<td>219</td>
<td>0</td>
<td>51</td>
<td>179</td>
<td>449</td>
</tr>
<tr>
<td>Dense trees (LCZ A)</td>
<td>4,960</td>
<td>1,616</td>
<td>4,497</td>
<td>284</td>
<td>6,359</td>
<td>17,716</td>
</tr>
<tr>
<td>Scattered trees (LCZ B)</td>
<td>1,028</td>
<td>540</td>
<td>394</td>
<td>555</td>
<td>302</td>
<td>2,819</td>
</tr>
<tr>
<td>Bush and scrub (LCZ C)</td>
<td>1,050</td>
<td>691</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1,741</td>
</tr>
<tr>
<td>Low plants (LCZ D)</td>
<td>4,424</td>
<td>985</td>
<td>7,688</td>
<td>984</td>
<td>376</td>
<td>14,457</td>
</tr>
<tr>
<td>Bare rock or paved (LCZ E)</td>
<td>0</td>
<td>0</td>
<td>214</td>
<td>0</td>
<td>109</td>
<td>323</td>
</tr>
<tr>
<td>Bare soil or sand (LCZ F)</td>
<td>359</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>144</td>
<td>503</td>
</tr>
<tr>
<td>Water (LCZ G)</td>
<td>1,732</td>
<td>2,603</td>
<td>234</td>
<td>500</td>
<td>3,492</td>
<td>8,561</td>
</tr>
<tr>
<td>Total</td>
<td>24,537</td>
<td>8,846</td>
<td>19,827</td>
<td>6,439</td>
<td>22,196</td>
<td></td>
</tr>
</tbody>
</table>

OpenStreetMap (OSM) for all cities.

The ground-truth is provided as a raster image at 100m resolution. Unfortunately, the provided data is severely imbalanced, and all provided maps are sparse. Figure 5.6 shows an example of the provided ground-truth map for the city of Paris. As reported in [114], the winners of the contest used ensemble approaches based on random forest techniques. These models are trained using the all Landsat 8 spectral bands and 43 handcrafted bands. These handcrafted bands includes: spectral indices, statistics (mean, standard deviation) over $10 \times 10$ patches, and morphological profiles extracted from the spectral bands and the OSM building layer using different structure sizes.

5.3.2 Data Pre-processing

In these experiments, the multispectral surface reflectance bands [1-7] of Landsat 8 images were downloaded from Google Earth\(^2\) at their original resolution of 30m. The visible clouds are masked using CFMask algorithm [141], and each spectral band is re-scaled between 0 and 1. The provided 100 m LCZ maps are up-sampled to 30 m to match the Landsat 8 resolution.

\(^2\)https://earthengine.google.com/
5.3.3 Pixel-based to Patch-based Prediction

The process of extracting patches from a given remote sensing image, \( I \), is illustrated in Figure 5.7. Given an image \( I \subset \mathcal{X} \in \mathbb{R}^{W \times H \times B} \), where \( W, H \) are the spatial dimension and \( B \) is the spectral dimension (number of bands). The entire image is scanned using a sliding window of size \( w \). During scanning, the sliding window is strided (e.g., shifted) using the stride scalar \( s \). Patches of
size \((p \times p)\) centered at all the scanned sliding windows are extracted. Setting \(w = (1, 1)\) and \(s = 1,\) will result in extracting all possible patches from the image. For efficiency, \(w > 1, s > 1,\) and \(p > s + w\) should be considered, which will result in more diversity in the extracted samples and will also significantly reduce the processing demand for both training and testing.

When generating the training patches, the sliding window acts as a filter and only the patches with a sliding windows that coincide with the ground-truth map \((G)\) are considered (see Figure 5.7). On the other hand, the ground-truth map is not available for the test images, therefore, all extracted patches are processed. However, the result of a single test patch will be populated to the sliding window of that patch. In this experiment patches of size 20x20 are extracted from the images using a moving window with a fixed stride of 3. We limit our experiments to the classes that exist in
CHAPTER 5. MULTI-SOURCE DOMAIN ADAPTATION USING DNNS

all training cities. Using this setting, the number of extracted samples/class (e.g., patches) for all training cities is provided in Table 5.2.

<table>
<thead>
<tr>
<th>LCZ Class label</th>
<th>Berlin</th>
<th>Hong kong</th>
<th>Paris</th>
<th>Rome</th>
<th>Sao paulo</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compact midrise (LCZ 2)</td>
<td>1,200</td>
<td>61</td>
<td>2,432</td>
<td>926</td>
<td>16</td>
<td>4,635</td>
</tr>
<tr>
<td>Open midrise (LCZ 5)</td>
<td>2,041</td>
<td>33</td>
<td>213</td>
<td>922</td>
<td>121</td>
<td>3,330</td>
</tr>
<tr>
<td>Open low-rise (LCZ 6)</td>
<td>3,587</td>
<td>13</td>
<td>1701</td>
<td>311</td>
<td>1,457</td>
<td>7,069</td>
</tr>
<tr>
<td>Large low-rise (LCZ 8)</td>
<td>1,096</td>
<td>45</td>
<td>525</td>
<td>225</td>
<td>1,314</td>
<td>3,205</td>
</tr>
<tr>
<td>Dense trees (LCZ A)</td>
<td>4,599</td>
<td>1,152</td>
<td>4,307</td>
<td>192</td>
<td>6,622</td>
<td>16,872</td>
</tr>
<tr>
<td>Scattered trees (LCZ B)</td>
<td>805</td>
<td>304</td>
<td>256</td>
<td>414</td>
<td>169</td>
<td>1,143</td>
</tr>
<tr>
<td>Low plants (LCZ D)</td>
<td>4,250</td>
<td>691</td>
<td>8,119</td>
<td>803</td>
<td>162</td>
<td>14,025</td>
</tr>
<tr>
<td>Water (LCZ G)</td>
<td>1,402</td>
<td>2,294</td>
<td>111</td>
<td>457</td>
<td>3,184</td>
<td>7,451</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>18,980</strong></td>
<td><strong>4,593</strong></td>
<td><strong>17,667</strong></td>
<td><strong>4,250</strong></td>
<td><strong>13,045</strong></td>
<td></td>
</tr>
</tbody>
</table>

Using this patch generation method, sample of the patches generated from the training images are shown in Figure 5.8- Figure 5.12.
Figure 5.8: Sample patches from Berlin.
CHAPTER 5. MULTI-SOURCE DOMAIN ADAPTATION USING DNNS

Figure 5.9: Sample patches from Paris.
Figure 5.10: Sample patches from Rome.
CHAPTER 5. MULTI-SOURCE DOMAIN ADAPTATION USING DNNS

Figure 5.11: Sample patches from Hong Kong.
CHAPTER 5. MULTI-SOURCE DOMAIN ADAPTATION USING DNNS

Figure 5.12: Sample patches from Sao Paulo
5.3.4 Baseline Network Architecture and Hyper-parameters

<table>
<thead>
<tr>
<th>Layer</th>
<th>Type</th>
<th>Settings</th>
<th>Output</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Input</td>
<td>-</td>
<td>20x20x7</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>Conv</td>
<td>64 x (3x3)</td>
<td>9x9x64</td>
<td>4,032</td>
</tr>
<tr>
<td>2</td>
<td>BN</td>
<td>-</td>
<td>9x9x64</td>
<td>256</td>
</tr>
<tr>
<td>3</td>
<td>Non-linerat</td>
<td>ReLU</td>
<td>9x9x64</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>Conv</td>
<td>64 x (3x3)</td>
<td>4x4x64</td>
<td>36,864</td>
</tr>
<tr>
<td>5</td>
<td>BN</td>
<td>-</td>
<td>4x4x64</td>
<td>256</td>
</tr>
<tr>
<td>6</td>
<td>Non-linerat</td>
<td>ReLU</td>
<td>4x4x64</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>Pooling</td>
<td>Max, (2x2)</td>
<td>3x3x64</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>Flatten</td>
<td>-</td>
<td>576</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>FC</td>
<td>200, ReLU</td>
<td>200</td>
<td>115,400</td>
</tr>
<tr>
<td>10</td>
<td>Drop-out</td>
<td>rate: 50%</td>
<td>200</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>FC</td>
<td>200, ReLU</td>
<td>200</td>
<td>40,200</td>
</tr>
<tr>
<td>12</td>
<td>FC (Logits)</td>
<td>8</td>
<td>8</td>
<td>1,608</td>
</tr>
<tr>
<td>13</td>
<td>Output</td>
<td>8, Softmax</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

**Total number of parameters:** 198,616

Conv: Convolution, BN: Batch Normalization, FC: Fully-Connected.

In this work we adopted a simple Convolutional Neural Network (CNN) architecture. Architecture and hyper-parameter tuning are performed by cross-validation on the training set and the best results are achieved with the network architecture shown in Table 5.3. The first stage of the network is the representation learning sub-network, which consists of two convolutional blocks. Each convolutional block consists of three layers: convolution, Batch Normalization (BN), and an activation layer. The ReLU activation function (where ReLU = \( \max(x, 0) \)) is used for the activation layer of the convolutional block. The output vector of this representation is then passed to the classification stage, which consists of two fully-connected layers, each of size 200 with ReLU activation functions, followed by a fully-connected layer that produces the logits’ output vector of size \( m = 8 \) classes. Finally, the logits’ outputs run through a softmax activation function \( (\sigma) \), which produces the normalized class probability vector. We train all models using the Adam optimizer [142], with a base learning rate of \( 10^{-4} \). We also fix the mini-batch size in all experiments to 64 samples. A data loader is used to feed the network with the required mini-batches. This loader samples uniformly from the training source(s) to generate the mini-batch for the next iteration.
Finally, we fix the loss control parameters ($\lambda_y$ and $\lambda_{DA}$) for LwF and LFL to 1.0.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Baseline</th>
<th>LFL-Fix</th>
<th>LFL-L2</th>
<th>LwF</th>
<th>AMDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start-up network</td>
<td>☒</td>
<td>☑</td>
<td>☑</td>
<td>☑</td>
<td>☒</td>
</tr>
<tr>
<td>$\mathcal{L}_{DA}$</td>
<td>☒</td>
<td>☒</td>
<td>☑</td>
<td>☑</td>
<td></td>
</tr>
<tr>
<td>Tuning $\lambda_y$</td>
<td>☒</td>
<td>☒</td>
<td>☑</td>
<td>☑</td>
<td>☒</td>
</tr>
<tr>
<td>Tr. Data$^a$</td>
<td>☑</td>
<td>☒</td>
<td>☒</td>
<td>☒</td>
<td>☑</td>
</tr>
</tbody>
</table>

$^a$Mini-batch training data: ☒: single-source, ☑: all-sources

### 5.3.5 Network Training

For a fair comparison, we use the same architecture and parameters described earlier for all methods (Table 5.3). Because the data comes from different sources, each with different dataset size and imbalanced classes, the epoch term becomes ambiguous. To overcome this, on every training cycle, we use the same number of SGD updates for all methods. The main difference between the methods, besides the required additional layer(s) or parameters as discussed in Section 5.2, is the training procedure. Table 5.4 summarizes the key differences between the training procedures of the implemented networks. For clarity, we describe the training protocol for each network:

**Baseline:** This network is trained by sampling from all sources, which can be viewed as a network trained using a single training dataset combining the training data from all sources. Furthermore, the baseline does not require a start-up network or any additional hyper-parameters.

**LwF:** In addition to the loss scalars $\lambda_y$, and $\lambda_{DA}$, LwF also requires a start-up network. Our strategy for selecting this start-up network involves creating a pool of source networks using the baseline network, each of which is trained using only one source. These networks are then evaluated on other training sources, and the network that achieves the best validation score is selected as a start-up network $\theta_o$. After each training cycle, the network validation score is re-evaluated, and $\theta_n$ replaces $\theta_o$ if improvement in the score is achieved.

**LFL:** The same technique used to build the start-up network for LwF is adopted for LFL as well. As discussed in Section 5.2.2, two types of LFL are implemented, LFL-L2 and LFL-Fix. In both
cases, the logits layer of the start-up network is frozen (Layer 12 in Table 5.3); however, LFL-L2 includes the DA loss of Equation (5.12), and therefore requires the control parameter $\lambda_{DA}$. $L_{DA}$ is computed using the start-up network, as shown in Figure 5.4(a). Since we are optimizing the network over multiple training sources, whenever an improvement is detected on the validation score, the fine-tuned weights of the new network $\theta_n$ will be used to replace the start-up network weights $\theta_o$, and therefore $F_o$ will be updated. However, the logits layer is never updated, and the parameters of this layer remain frozen for the entire training stage.

**AMDA:** The AMDA network is identical to the baseline network in terms of architecture and number of parameters — they only differ in how the loss is computed. In particular, the baseline network can be viewed as an AMDA network with a fixed $\lambda_i = \frac{1}{N}$ for all sources during the training stage. Furthermore, unlike LwF and LFL, AMDA does not require a start-up network at the beginning of training.

### 5.4 Results and discussion

The results obtained for the proposed techniques for solving the LCZ classification problem are presented in this section.

#### 5.4.1 Single-source Single-target Scenario

We will first present the results of the single-source single-target scenario to provide some insight on the problem difficulty, and how the sources are related. In this scenario, a baseline network is trained using a single image as the source domain, and the rest of the images are considered a part of the target domain. We further tested the single-source single-target domain adaptation case as an additional baseline. Domain Adversarial Neural Networks (DANN) [9] is used here, and for each source-target pair, the target samples are used to compute the adversarial loss. Table 5.5 shows the classification accuracy results of this scenario. If we are able to select the optimal source for a given target, then we will obtain the highlighted results in the table. Examining the single-
CHAPTER 5. MULTI-SOURCE DOMAIN ADAPTATION USING DNNS

source single-target baseline (SSBL) results also shows the relationships between these sources. For example, using Rome as a source provides the best accuracy result for Paris. On the other hand, using Paris as a source results in poor accuracy for Rome. We attribute these significant differences to the variation on the data distribution of each couple, more specifically, the ground-truth diversity of the source compared with the target. A closer look at the Rome and Paris data distributions in Table 5.2 will reveal that the balance within the ground-truth samples of Rome enables its baseline network to perform better than the Paris baseline network. Due to the drastic imbalance of the ground-truth distribution of the Paris image, its baseline network is unable to generalize to cities with different ground-truth distribution. Similar observations can be also noted with Sao Paulo which also has drastic imbalance of the ground-truth distribution. On average, Berlin, as a source, has achieved the highest average accuracy (0.761) and this can be attributed to the equitable, to a certain extent, ground-truth distribution compared to the other cities.

The Single-Source Domain Adaptation (SSDA) results in Table 5.5 did not show a significant improvement for all the cases despite the fact this method uses the target data to align the adopted features for both domains. It is also important to note the negative transfer effect for some cases which causes the DA method to significantly degrade performance. These results confirm the need to efficiently exploit the available data from more than one source for better domain adaptation.

Table 5.5: Single-source single-target baseline network accuracy. Results highlighted in bold are the best test results for each case (SSBL: Single-Source Baseline, SSDA: Single-Source Domain Adaptation).

<table>
<thead>
<tr>
<th>Train (Source)</th>
<th>TEST (Target)</th>
<th>Berlin SSBL</th>
<th>Berlin SSDA</th>
<th>Hong Kong SSBL</th>
<th>Hong Kong SSDA</th>
<th>Paris SSBL</th>
<th>Paris SSDA</th>
<th>Rome SSBL</th>
<th>Rome SSDA</th>
<th>Sao Paulo SSBL</th>
<th>Sao Paulo SSDA</th>
<th>Average SSBL</th>
<th>Average SSDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berlin</td>
<td></td>
<td>0.658</td>
<td>0.784</td>
<td>0.852</td>
<td>0.756</td>
<td>0.849</td>
<td>0.861</td>
<td>0.761</td>
<td>0.769</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hong Kong</td>
<td></td>
<td>0.525</td>
<td>0.403</td>
<td>0.643</td>
<td>0.321</td>
<td>0.662</td>
<td>0.588</td>
<td>0.789</td>
<td>0.726</td>
<td>0.655</td>
<td>0.525</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paris</td>
<td></td>
<td>0.741</td>
<td>0.754</td>
<td>0.764</td>
<td>0.757</td>
<td>0.463</td>
<td>0.518</td>
<td>0.876</td>
<td>0.879</td>
<td>0.795</td>
<td>0.757</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rome</td>
<td></td>
<td>0.684</td>
<td>0.697</td>
<td>0.459</td>
<td>0.791</td>
<td>0.860</td>
<td>0.659</td>
<td>0.816</td>
<td>0.879</td>
<td>0.705</td>
<td>0.757</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sao Paulo</td>
<td></td>
<td>0.641</td>
<td>0.529</td>
<td>0.751</td>
<td>0.794</td>
<td>0.690</td>
<td>0.763</td>
<td>0.424</td>
<td>0.384</td>
<td>0.627</td>
<td>0.618</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.4.2 Multi-source DA Results

Next, results from the proposed methods are presented. These methods are evaluated by creating five different multi-source DA scenarios. In each scenario, one of the images is considered as a target, and the rest are considered source images. The average accuracy over 10 runs for each scenario is shown in Table 5.6. We also compute the overall average accuracy of each method for these five scenarios. Finally, we include the relative improvement made by each method on the overall average accuracy with respect to the baseline overall average accuracy. “B. SSBL”, and “B. SSDA” columns in Table 5.6 represent the “best” source results for the single source scenario extracted from Table 5.5, which assumes that the optimal source is selected for each target.

Table 5.6: LCZ classification accuracy results: **bold** are best, **underline** are second best.

<table>
<thead>
<tr>
<th>Target</th>
<th>Baseline</th>
<th>B. Source</th>
<th>LFL-Fix</th>
<th>LFL-L2</th>
<th>LwF</th>
<th>AMDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berlin</td>
<td>0.703</td>
<td>0.741</td>
<td>0.794</td>
<td>0.843</td>
<td><strong>0.848</strong></td>
<td>0.827</td>
</tr>
<tr>
<td>Hong kong</td>
<td>0.658</td>
<td>0.751</td>
<td>0.685</td>
<td>0.708</td>
<td>0.758</td>
<td><strong>0.817</strong></td>
</tr>
<tr>
<td>Paris</td>
<td>0.915</td>
<td>0.860</td>
<td>0.917</td>
<td><strong>0.936</strong></td>
<td><strong>0.936</strong></td>
<td>0.927</td>
</tr>
<tr>
<td>Rome</td>
<td>0.532</td>
<td>0.619</td>
<td>0.653</td>
<td>0.592</td>
<td><strong>0.684</strong></td>
<td>0.675</td>
</tr>
<tr>
<td>Sao Paulo</td>
<td>0.868</td>
<td>0.866</td>
<td>0.863</td>
<td><strong>0.884</strong></td>
<td>0.869</td>
<td>0.875</td>
</tr>
<tr>
<td>Overall Average</td>
<td>0.735</td>
<td>0.780</td>
<td>0.778</td>
<td>0.793</td>
<td>0.813</td>
<td><strong>0.824</strong></td>
</tr>
<tr>
<td>Improvements%</td>
<td>-</td>
<td>6.13%</td>
<td>5.49%</td>
<td>7.80%</td>
<td><strong>10.55%</strong></td>
<td><strong>12.10%</strong></td>
</tr>
</tbody>
</table>

On average, classification accuracies based on the best source network(s) are better than the baseline(s). This enhancement over the baseline (≈ 6%) is a strong indicator that an efficient DA technique is required, or at least a technique that allows for the selection of the right source for a given target. The LFL-Fix network, despite its simplicity, is also able to compete with the baseline network. However, adding the DA regularization term in LFL-L2 (e.g., matching the features extracted from the networks) helps to improve the results. Also, LFL-Fix and LFL-L2 are affected by the quality of the start-up network, since the decision boundaries of this start-up network are used for the full duration of training.

While LwF is able to outperform the baseline by more than 10%, AMDA is able to outperform all techniques for overall average accuracy. Besides its simplicity, and its property of not alternating between domains, AMDA also does not require a start-up network, unlike the LwF and LFL.
CHAPTER 5. MULTI-SOURCE DOMAIN ADAPTATION USING DNNS

This is an important property for AMDA, as initializing the training with a poor quality start-up network may significantly affect the training progress and the solution quality of the network.

5.4.3 Network Complexity and Scalability

The network complexity (i.e., number of parameters) and the training time of the proposed techniques are presented in Table 5.7. Although training time and model complexity are not a major concern for this particular application, it is important to realize these factors when selecting the best method that fits within application constraints and available resources.

Table 5.7: Network complexity and training time.

<table>
<thead>
<tr>
<th>Network</th>
<th># of Parameters</th>
<th>Training Time (min)</th>
<th>Prediction Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>198,616</td>
<td>1.700</td>
<td>9.52 × 10^{-5}</td>
</tr>
<tr>
<td>LwF</td>
<td>397,232</td>
<td>3.341</td>
<td>9.52 × 10^{-5}</td>
</tr>
<tr>
<td>LFL-L2</td>
<td>395,624</td>
<td>7.019</td>
<td>9.52 × 10^{-5}</td>
</tr>
<tr>
<td>LFL-Fix</td>
<td>198,616</td>
<td>2.430</td>
<td>9.52 × 10^{-5}</td>
</tr>
<tr>
<td>AMDA</td>
<td>198,616</td>
<td>5.914</td>
<td>9.52 × 10^{-5}</td>
</tr>
</tbody>
</table>

Compared to the other proposed DA networks, LFL-Fix is the fastest, and LFL-L2 is the slowest in terms of training time. The low speed of LFL-L2 is mainly due to the additional computation of the L2-norm that forms the features vectors. In terms of network complexity, the LwF network is the largest, and is twice the size of the baseline network. However, these extra parameters did not significantly slow the training time, as half of these parameters are frozen during the adaptation process. AMDA is identical to the baseline network in terms of the number of parameters; however, the AMDA optimization process is more complicated than the baseline, which requires evaluating the validation accuracy for each domain to update their respective loss scalar. Furthermore, due to the changes in the loss weights of each branch, AMDA convergence is slower than the baseline; however, the latter still converges quickly, but to a worse solution.

In terms of scalability, all proposed methods are capable of handling data of any size. However, as the number of sources increases, the training time also increases for all methods. The domain expansion methods, (LwF and LFL), are more susceptible to data size increases. We anticipate
CHAPTER 5. MULTI-SOURCE DOMAIN ADAPTATION USING DNNS

AMDA will be the least susceptible due to the fact that its training protocol does not require to alternate between sources after every iteration. It is also important to note that when deployed, all of the networks have the same architecture and parameters as the baseline, and therefore, the inference time is identical for all methods. The inference time is found to be less than 1 sec for every $10^4$ samples. Therefore, the network will be able to produce the full LCZ map for a large city, such as Paris ($3866 \times 3292$ pixels), in less than 2 minutes. Table 5.8 shows the time required to generate the LCZ map for our method, for the 1st place (WXYZ) and the 2nd place (AGT) winners of the DFC 2017 contest. Both WXYZ and AGT have heavy pre-processing pipelines that implement hand-crafted feature extraction. Furthermore, because they are ensemble-based, they require more time to make predictions. Finally they require a post-processing stage to smooth the resulted map. Our method, on the other hand, does not require pre-processing or post-processing and therefore is significantly faster in generating LCZ maps. We also note that choosing a sliding window larger than $1 \times 1$ allows our method to significantly reduce the amount of data that needs to be processed.

Table 5.8: LCZ map generation time for AMDA, DFC 2017 winners. All times are in seconds.

<table>
<thead>
<tr>
<th>Test City</th>
<th>WXYZ TEAM (1st place)</th>
<th>AGT TEAM (2nd place)</th>
<th>AMDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amsterdam</td>
<td>33.4</td>
<td>379.8</td>
<td>1.1</td>
</tr>
<tr>
<td>Chicago</td>
<td>250.3</td>
<td>2464.5</td>
<td>8.5</td>
</tr>
<tr>
<td>Madrid</td>
<td>128.5</td>
<td>1718.7</td>
<td>4.0</td>
</tr>
<tr>
<td>Xi’an</td>
<td>59.1</td>
<td>767.4</td>
<td>2.5</td>
</tr>
</tbody>
</table>

Fea. Ex.: Feature extraction, Pred.: prediction and ensemble (if applicable), Post.: Post processing, Patching: Patch generation.

5.5 Summary

In this chapter we addressed the multi-source domain adaptation (MDA) problem for the classification of remote sensing images. We proposed an a DNN-based Adaptive Multi-source Domain Adaptation (AMDA) to efficiently solve this problem. We further modify and extend two domain expansion methods, proposed in the literature, to be used for MDA. Compared to other learning techniques, such as random forest and support vector machines, DNNs offer a great deal of scal-
ability that is more suitable for large-scale datasets. The proposed methods are implemented and evaluated using the Local Climate Zones classification problem.

The proposed methods are compared to a baseline network trained using a combination of the data from all source domains. The proposed techniques were able to consistently improve on the baseline, with the largest gain made by AMDA. AMDA is easy to implement, and is able to scale to any number of sources, and can efficiently deal with imbalanced data distribution among the source domains. Furthermore, unlike currently proposed MDA approaches for remote sensing, AMDA does not require any pre-processing, such as feature engineering and hand-crafting, or post-processing steps.
Chapter 6

Conclusions and Future Work

This thesis addresses domain adaptation (DA) for the classification of remote sensing (RS) images. In general, DA is a challenging task, although in RS images, the DA task is even more challenging due to the significant data shift between the domains. The reason for this additional difficulty is the result of the nature of the RS acquisition system, which is usually not under the user’s control.

Several techniques were proposed based on deep neural networks (DNNs) for solving the RS classification task under different DA settings:

1. **Single Source DA** In this DA setting, only one source and one target image are considered. Two frameworks were proposed for learning invariant representation from both domains using unsupervised deep learning approaches:

   • A denoising autoencoder (DAE) was employed to learn invariant representation. DAE is a two-stage DA approach, which starts with an unsupervised representation learning stage that is optimized by reconstructing the clean version of the input from a corrupted one. This representation is then used in the second stage to learn the classification model based on the source samples.

   • An end-to-end domain adversarial neural network (DANN) was employed to jointly learn the invariant representation and the classification model. A branch of the network contains a domain discriminator to distinguish between the source and the target sam-
ples using the learned representation. This representation is optimized to perform well with the classification task. At the same time, this representation should confuse the domain discriminator such that it cannot distinguish between the domains.

2. **Multi-source DA** In this DA setting, multiple sources were considered and several algorithms were proposed based on deep convolutional neural networks:

- Two domain expansion techniques were extended for multi-source DA. Learning without forgetting (LwF) and less forgetful learning (LFL) optimize a start-up network by alternating between the sources until it reaches the best scores among all the sources.
  - LwF optimizes the network by encouraging it to maintain a similar prediction for the visited domains.
  - LFL optimizes the solution by forcing the network to use the start-up decision boundaries, and by also encouraging the network to maintain a similar representation for all the visited domains.
- Developed a novel adaptive multi-source DA (AMDA) technique that was capable of reconciling the performance among all the source domains, and at the same time, performed well in the target domain.

### 6.1 Conclusions

The proposed techniques have been implemented and evaluated using different DA scenarios. The results confirmed the applicability of these techniques. However, the following should be noted:

1. **Utilization of unlabelled samples from the target domain:** Utilizing the unlabelled samples from the target domain is beneficial for DA and for classical image classification tasks if the target samples are not a concern. However, the RS classification task is a pixel-based classification task. Sampling from the target domain is therefore not a trivial process, and it
might be inefficient for some large-scale RS classification problems. From the experiments conducted during this research, it was concluded that:

- **Single source:** While DANN requires samples from the target domain to achieve better results, DAE is not constrained by this requirement. In fact, the results showed that the representation learned from the DAE trained using only the source samples is similar, and sometimes even better, than representation learned using samples from both domains.

- **Multi-source:** In this case, given that there are enough sources to capture the variability in the feature space, it is not necessary to sample from the target domain.

2. **Application:** The remote sensing data to be considered depends on the application. For example, in a crop mapping application, the imaging time is important to be able to distinguish between the different crops. Images captured at the early stage of the season will provide poor results; however, obtaining images when the crops are in the final growing stage will provide acceptable results. Another example is the land-use application, which does not impose a hard constraint on the image timing in a spatial adaptation problem. However, for spatial and temporal adaptations, the solution quality deteriorates significantly due to the significant shift and changes in the spectral bands.

3. **Labeling and benchmarking:** As indicated earlier, the labelling process for RS images is not a trivial process, and the quality of the labelling process is usually questionable. Inaccurate labels will result in an imprecise evaluation. Furthermore, with low-resolution RS images, this task becomes harder and the level of obscurity increases. However, for large-scale applications, such as local climate zones (LCZs), high precision is not necessary as long as the classification map captures the main regions of the image correctly.

4. **Scalability:** Scalability is one of the main requirements for most RS applications, especially with applications that require processing multiple images for the same task. The two main scalability requirements for RS are storage and computation. This thesis argues that deep
neural networks are suited to fulfill these requirements. The proposed adaptive multi-source DA (AMDA) can be considered as one such example of a scalable system; however, AMDA still requires images from different sources to be available at the same time, which cannot be easily satisfied.

6.2 Lessons Learned

In this section, we share some of the important lessons that have been learned during this study. They are as follows:

1. **Experimental Setup:** Designing a remote sensing experiment is a complicated task. First the remote sensing images must be carefully examined. Relying only on visual inspection is insufficient and may be misleading. At least a primary analysis should be conducted such as computing the band’s histogram, and the ground-truth distribution in the images. The result of this primary analysis should help in selecting reasonable and realistic images/regions to be used as a source(s) or as a target(s) (refer to Section 4.3 and Section 4.4 for more details on how this process can be achieved).

2. **Patch preparation and mini-batch sampler/loader for deep learning:** In order for a researcher to use CNNs for a pixel-based remote sensing classification, the task must be transformed into a patch-based task (refer to Section 5.3.3 for more details on how this process can be achieved). To be able to obtain good quality results, it is also important to explore different settings for the hyper-parameters used during this process. This include the patch size and the sampling window size. The image resolution and the nature of the classification task plays an important role in setting these hyper-parameters. For example, large patches (e.g., 50 × 50) can be considered for a crop mapping application using Landsat 8 images, since the crop fields are usually of large size. However, these large patches should not be considered for land-use application. Also, it is necessary to design and implement a mini-batch sampler within your favourite deep learning framework (e.g., a function or class). This
CHAPTER 6. CONCLUSIONS AND FUTURE WORK

module should handle the data loading and the mini-batch(s) feeder for the deep learning model and should be capable of handling multiple sources. A successful implementation for such sampler/data loader should speed up the process of evaluating different architecture and different training regimes.

3. **Deep Learning baseline model**: Designing the baseline model is not trivial and it can be considered one of the most time consuming tasks. In order to speed up this task, it is essential to start with a simple deep learning model and evaluate it using few scenarios from the designed experimental settings that have been already established. At this stage the actual final results (e.g., convergence speed, accuracy) are not expected to be acceptable. Performing design exploration of the architecture is an important exercise to achieve convergence. Once this is achieved, it is useful to further tune several hyper-parameters and increase the model capacity.

6.3 Future Work

Despite the progress which has been made in remote sensing and machine learning, many challenges still need to be addressed. Possible future directions are highlighted here to address some of these challenges.

- **Multi-sensor adaptation**: Multi-sensor adaptation has not been extensively studied, and it is necessary to establish a classification system that can cope with images captured using different sensors. For example, images from Landsat-8 and Sentinel-2 differ in spectral bands, ground-resolution, and revisit frequency. The ability to utilize a continuous stream of images from both sources at the same time would advance the earth monitoring efficiency to a higher level. Moreover, several studies have shown that combining optical sensors, such as Landsat, with non-optical sensors, such as light detection and ranging (LiDAR), synthetic aperture radar (SAR), helps the classification task [143].

Many challenges need to be addressed for multi-sensor DAs, including how to align the
images into a common feature space and if it possible to learn a new space that is common for both images. One possible approach to investigating this is to use autoencoders to establish this alignment. Unlike the work in [98], a shared encoder can be used to map the data from different sensors to the same latent space, as shown in Figure 6.1. While the shared encoder is optimized to align the spaces into a common space \( z \), the private encoder for each sensor is optimized to reconstruct the same input. The input size of the shared encoder should be equal to the maximum number of bands for the sensors under consideration. One possible way to solve this is to stack blank bands on the patches generated from the source with lesser bands. A successful training for the shared encoder would enable the establishment of a single model that would use the aligned space \( Z \) to build a unified multi-source classification model.

![Figure 6.1: Autoencoder approach for aligning the multi-sensor feature space](image)

- **Multi-source adaptation**: Multi-source adaptation is a natural problem in RS. It is anticipated that many potential benefits from RS will be fulfilled once a robust and efficient solution to this is developed. In the future, the proposed methods will be evaluated on other multi-source domain adaptation benchmarks with a particular focus on AMDA. A more in-depth analysis will be carried out regarding their performance and scalability. Another direction being pursued is a collaboration with the IEEE Geoscience and Remote Sensing Society as well as the World Urban Database and Portal Tool to create a large-scale LCZ experiment to evaluate the proposed methods.

- **DNN - Generator Unit**: One of the challenges of RS is the high storage requirement for the
images that are collected over time. The common practice is to store these images on low-cost and low-speed storage units; however, accessing this data will slow the entire modelling process (e.g., AMDA). The required images must therefore be transferred to faster storage before starting the modelling process. A possible method to overcome this would be to build and maintain a DNN that generates samples as required by the modelling process. This DNN generator could be parameterized such that the user could request specific samples from the historical data. This would allow the system to replace a massive amount of storage (in the order of hundreds of terabytes) with a DNN model that would require only a fraction of the original storage space. Moreover, sampling from this generator would be efficient in terms of speed and quality. This generator could be fine-tuned and enriched whenever new data becomes available. This generator or generators could be used to replace the source images that AMDA would demand to build the classification model.

- **Benchmarking:** One possible direction to enhance benchmarking is to establish a classification framework on the cloud. Nowadays, cloud services, such as Amazon Web Services (AWS) and Google Cloud, provide great opportunities for the RS community to build a scalable, sustainable, and efficient RS platform that can be used by researchers and industries to advance the level of utilization of the growing RS field. It is about time to realize remote sensing on-line applications that perform all the necessary processing on the cloud and deliver the final product to the user.
References


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES