Machine Learning Technologies for Enabling Earth Observation

Benjamin Lucas

A thesis submitted for the degree of Doctor of Philosophy at Monash University in 2021

Clayton School of Information Technology

Copyright Notice

Abstract

Earth observation satellites are collecting data about our world at an unprecedented rate and scale. One such satellite constellation, the European Space Agency's Sentinel-2, captures a high-resolution multi-spectral image of the whole of the Earth's land surface every 5 days. As the satellites frequently revisit locations, the images can be collated temporally to create a multi-variate time series describing the change in the Earth's surface over time. The resulting Satellite Image Time Series (SITS) dataset consists of trillions of time series.

Given the magnitude of the data, machine learning plays a vital role in ensuring that they are utilised to their full potential. The research presented in this thesis explores machine learning technologies to better facilitate land cover classification from SITS. Specifically, I look at two challenges of applying machine learning to SITS: (1) Finding a time series classifier capable of scaling to the magnitude of the SITS dataset; and (2) How to train an accurate classifier when labelled data are scarce.

In response to the first challenge, I developed Proximity Forest, a tree-based classifier that uses a novel time series-specific splitting criterion at the node. My experiments show that Proximity Forest can learn from a SITS dataset of size 1 million in 17 hours, compared to the more than 200 years that would be required by the state of the art. Further experiments demonstrate Proximity Forest to be competitive with the state of the art in overall accuracy on the UCR repository of time series datasets (the benchmark in the field). My results show that Proximity Forest represents the new state of the art for time series classification as it is one of the most accurate classifiers while also being capable of scaling into the millions (and beyond).

For the second challenge, I present two different semi-supervised domain adaptation (DA) techniques for producing land cover maps from SITS when there is only a small amount of training data available in the study area. The first technique, Encoda, is based on co-learning and is designed for use when the target domain sample has an even distribution across all classes. My experiments show that Encoda is competitive with state-of-the-art techniques on the benchmark DomainNet dataset and also has a wide range of potential applications beyond Earth observation. The second technique, Sourcerer, is a deep learning-based technique utilising the principles of Bayesian inference through a novel regularisation method. Sourcerer is designed specifically for

the classification of SITS and my experiments show it to outperform the other state-of-the-art deep learning-based methods on two separate study areas. Sourcerer excels when the data are collected at random across the study area, in particular when the sample excludes some of the land cover classes.

Declaration

This thesis is an original work of my research and contains no material that has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Signature: 

Print Name: Benjamin Lucas

Date: 4 April 2021

Publications During Enrolment

Journal Articles:

1. Lucas, B., Pelletier, C., Schmidt, D., Webb, G.I., Petitjean, F.: A Bayesian-inspired, deep learning, semi-supervised domain adaptation technique for land cover mapping. Machine Learning. [In press] (2021). DOI 10.1007/s10994-020-05942-z.

2. Lucas, B., Shifaz, A., Pelletier, C., O'Neill, L., Zaidi, N., Goethals, B., Petitjean, F., Webb, G.I.: Proximity forest: an effective and scalable distance-based classifier for time series. Data Mining and Knowledge Discovery 33(3), 607–635 (2019). DOI 10.1007/s10618-019-00617-3.

3. Fawaz, H.I., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D.F., Weber, J., Webb, G.I., Idoumghar, L., Muller, P.A., Petitjean, F.: InceptionTime: Finding AlexNet for time series classification. Data Mining and Knowledge Discovery 36(6), 1936–1962 (2020). DOI 10.1007/s10618-020-00710-y.

Conference Papers:

4. Lucas, B., Pelletier, C., Schmidt, D., Webb, G.I., Petitjean, F.: Unsupervised domain adaptation techniques for classification of satellite image time series. In: IGARSS 2020 - 2020 IEEE International Geoscience and Remote Sensing Symposium, pp. 1074–1077 (2020). DOI 10.1109/IGARSS39084.2020.9324339.

5. Lucas, B., Pelletier, C., Inglada, J., Schmidt, D., Webb, G., Petitjean, F.: Exploring data quantity requirements for domain adaptation in the classification of satellite image time series. In: F. Bovolo, S. Liu (eds.) MultiTemp 2019, 10th International Workshop on the Analysis of Multitemporal Remote Sensing Images, August 5-7, 2019 – Shanghai, China. IEEE, Institute of Electrical and Electronics Engineers, United States of America (2019). DOI 10.1109/Multi-Temp.2019.8866898.

Acknowledgements

This thesis marks both an end of a journey and the commencement of a new one. There are so many people to thank for every piece of support and advice that I have received over the last three years. First and foremost, I wish to thank you to my supervisors, whose backgrounds complemented each other perfectly. To Geoff, thank you for your endless ideas and invaluable opinions, for someone with so much on his plate it's truly admirable how available you are to your students. To Daniel, I learnt more about probability and statistics from you in 3 years than from 2 degrees prior. You are a gifted teacher and your passion for the subject matter is evident from the moment you begin talking. To Francois, you gave me more time and energy than I could have asked for and you made me a computer scientist. To Charlotte, you taught me almost everything I know about remote sensing and always managed to find the bugs in my code. I one day hope to be as dedicated to supervising my students as you were to me. Outside of my supervisors, I could not have completed this thesis without Julie Holden. Thank you for teaching me research and writing skills, helping clarify my thoughts, piloting me through the complexities of the university systems, and providing friendship and counselling me on those countless occasions. I would also like to thank Lan Du and Christoph Bergmeir for their comments during my yearly reviews, my co-authors, and everyone else in the Faculty of IT for the adventure.

Thank you to my family in particular my mother, who always put her children before herself and whose support is unconditional. Raising two boys as a single mother while on minimum wage and having them both earn PhDs is far more impressive of an accomplishment than any degree.

Finally, I must thank Kylie. I began this thesis before we met and by the end of it we're married. Thank you for the emotional support; the love, laughter and friendship; and letting me work 7 days a week when I wanted to. I look forward to being able to be the one to support your dreams in the future.

This research was supported by an Australian Government Research Training Program (RTP) Scholarship.

For me,
The satellite,
Orbiting the world alone,
Tell me I'm coming home.

—Matt Pryor, The Get Up Kids

Contents

List of Figures

List of Tables

Introduction

1

Introduction to Machine Learning for Earth Observation

Machine learning plays a vital role in the field of Earth observation. This role is sometimes a direct application of an existing machine learning methods such as segmentation or classification, but more often than not, a novel approach is required owing to the individual nature of the Earth observation data and application. In this thesis I explore and develop machine learning technologies required by Earth observation applications. In particular, I look at the case of using machine learning to produce land cover maps from satellite image data, and how this can be performed successfully under various research constraints.

This chapter will introduce the essential background information required throughout the remainder of the thesis. It begins by describing what land cover maps are, and then proceeds to discuss how they can be produced from satellite images. The latter sections of the chapter discuss the research aims of my thesis and provide an overview of the remainder of the work.

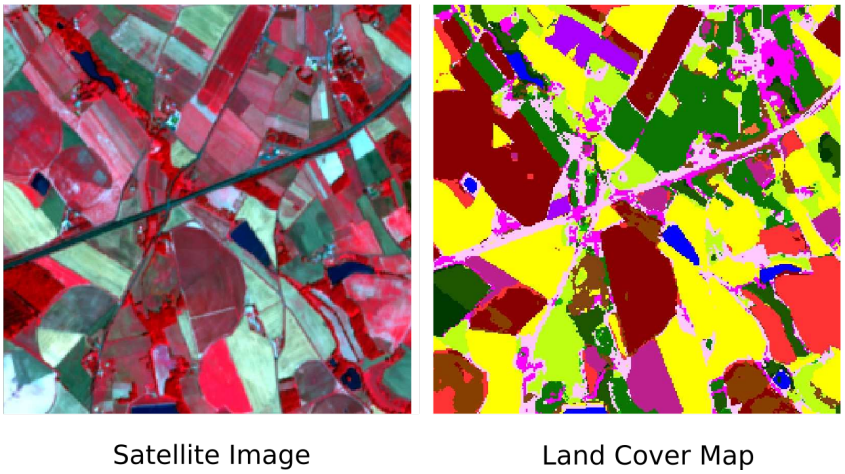


Figure 1.1: A land cover map and an aerial image of the corresponding land area being mapped. The colours of the land cover map represent different land cover classes such as agricultural, urban, forest, or water.

1.1 Land Cover Maps

Land cover maps are a spatial representation of the observed coverage of the Earth's surface. They can detail both biophysical coverage such as forests, grasslands or lakes, and anthropogenic coverage such as agriculture or urban environments. Figure 1.1 shows an example of a land cover map and an aerial image of the area being mapped where each colour of the map represents a different land cover class, eg. maize crop, road, soy crop, water, etc.

Land cover information enables us to observe and analyse the evolution of the Earth at various spatial and temporal scales. Spatially, maps can be at the scale of the continental down to the local, while temporally, they can represent the present land cover, the year-on-year land cover change, or land cover changes over decades. These broad scales both lend themselves to, and are the result of, the wide range of applications land cover maps are produced for. Maps of agricultural land covers have been used to analyse crop growing trends at a national level [21] and also to assess the relationship between land cover and stream dynamics in a local catchment area [148].

A change in land cover is a significant contributor to many larger changes in the Earth's biophysical system [53] and consequently land cover maps are considered a vital element of analysing many types of environmental processes [170]. For example, maps showing current vegetation types are used for bushfire management [95] and biodiversity conservation [163, 4], while maps showing changes in land cover have been used to analyse the deforestation [5] and urbanisation [191].

Another notable current application of land cover maps is in climate science [118, 52]. As there is an inextricable link between the land and atmosphere, land cover maps have been labelled an "essential climate variable" in support of climate research [18].

The wide-spread recognition of the importance of land cover maps from researchers and

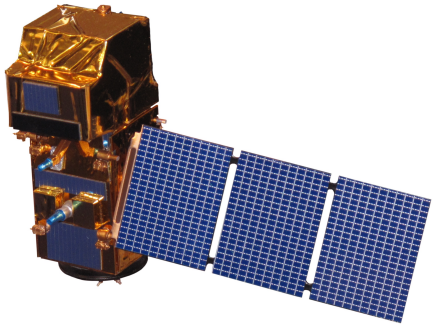


Figure 1.2: An illustration of one of the Sentinel-2 twin satellites used for land cover observation and vegetation monitoring [47].

practitioners has driven a vast improvement in related technologies and analysis over the past few decades [69].

1.2 Remote Sensing

Modern land cover maps are produced using remotely-sensed data, that is data collected from space via sensors mounted on satellites [165]. The first satellite launched specifically for this purpose was Landsat 1 launched by NASA in 1972, which returned just over 100 000 images at a spatial resolution of 80 metres [119]. Landsat 1 was the first of what has since become the longest-running Earth observation satellite program, with Landsat 7 (launched in 1999) and Landsat 8 (launched in 2013) both currently in orbit. The images from Landsat 1 represented approximately 75% of the Earth's surface, a figure which is collected in less than a week by modern satellites. One such modern satellite constellation, Sentinel-2, is the major source of data for the research comprising this thesis.

The Sentinel-2 satellite constellation is a part of the Copernicus Programme of Earth observation missions coordinated by the European Space Agency [47]. An illustration of one of the Sentinel-2 satellites in shown in Figure 1.2. The satellites take optical images of the Earth's surface at a resolution of 10-60 metres, with a revisit frequency of 10 days, and as the constellation consists of a twin-pair of satellites, the result is a high-resolution image of the entire Earth's land cover captured every five days.

In recent times, most satellite images acquired for land cover and vegetation monitoring have been made freely available for download. This was not the case with early missions and it is believed that this action has contributed greatly to advances in the field [184].

The data are available as multi-spectral images, which are recorded as an array of decimal values. An image is captured, as a value against multiple spectral bands representing the intensity of the light within a specific wavelength.

Where an image from a standard camera is comprised of the light across the three wavelength bands of the visible spectrum–blue, green and red, a multi-spectral image includes light with wavelengths outside of the visible spectrum such as infrared, ultraviolet or x-rays. The bands are not necessarily continuous and can be overlapping, so light of some wavelengths may be recorded across two bands or not recorded at all. Table 1.1 shows the wavelengths of light captured by the 13 spectral bands of the Sentinel-2 satellites. Bands 2-4 represent the spectrum of visible light. Bands 8 and 8a are both Near Infrared (NIR) however one has a narrower bandwidth. Bands 9-10 are primarily used in correcting the effect of the atmosphere on measurements.

Table 1.1: Spectral bands for the imaging sensors mounted on the Sentinel-2A satellite

Spectral band	Central wavelength (nm)	Bandwidth (nm)	Spatial resolution (m)
Band 1 – Coastal aerosol	442.7	21	60
Band 2 – Blue	492.4	66	10
Band 3 – Green	559.8	36	10
Band 4 – Red	664.6	31	10
Band 5 – Vegetation red edge	704.1	15	20
Band 6 – Vegetation red edge	740.5	15	20
Band 7 – Vegetation red edge	782.8	20	20
Band 8 – Near Infrared	832.8	106	10
Band 8A – Narrow Near Infrared	864.7	21	20
Band 9 – Water vapour	945.1	20	60
Band 10 – Short wave Infrared	1373.5	31	60
Band 11 – Short wave Infrared	1613.7	91	20
Band 12 – Short wave Infrared	2202.4	175	20

Land cover maps can be produced automatically by applying machine learning algorithms to these data. To do this, a model will use the spectral signature of a land cover type in order to classify each pixel of a region. For example, vegetation has a high reflectance in the near-infrared band, while urban land cover does not, and therefore a pixel with high values in Bands 8 and 8A is likely to be classified as a type of vegetation.

1.3 Satellite Image Time Series

Traditionally, land cover maps were produced by using machine learning models on single multi-spectral images, in which each pixel is represented by an array of values. However in recent times it has been found that by using a temporal sequence of the images more accurate land cover maps can be produced [80].

These data, known as Satellite Image Time Series or SITS, represent each pixel with a multivariate time series meaning that each spectral band is represented by a series of values. A land cover map is then created based on the profiles of these time series by classifying each pixel as a land cover class, such as urban, wheat crop, or deciduous forest. Figure 1.3 illustrates the

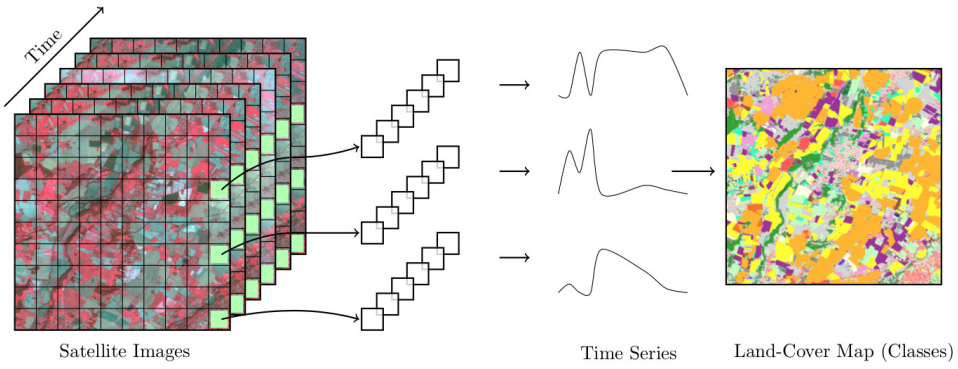


Figure 1.3: The production of time series data from satellite images [159]. Each pixel is represented by a series a multi-spectral images which can then be labelled with their land cover type using machine learning algorithms.

production of a land cover map using SITS.

Examples of the multivariate time series resulting from images of three different types of land cover—urban, soy crop, and deciduous forest—are shown in Figure 1.4.

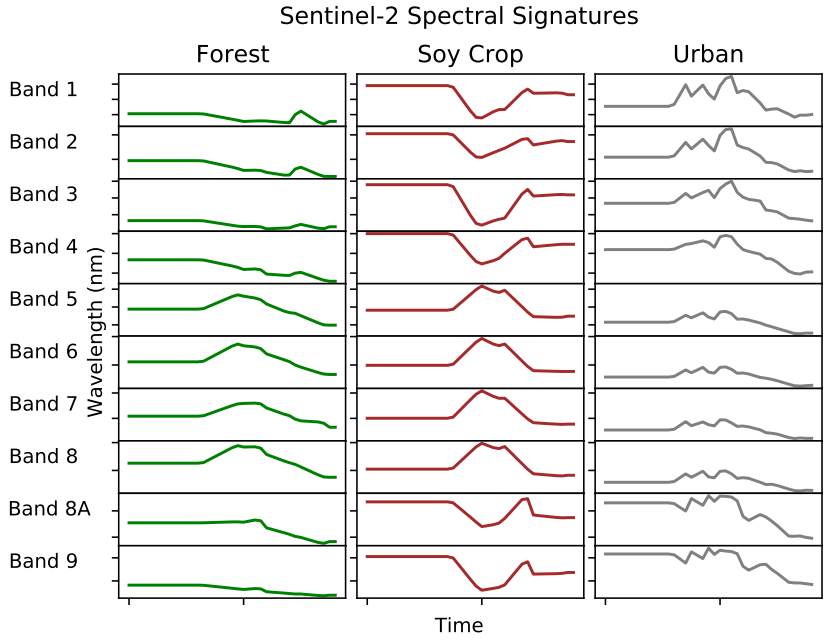


Figure 1.4: The spectral signatures of 3 different land cover classes (Forest, Soy Crop, and Urban) when imaged by Sentinel-2 satellites.

Maps produced using SITS have been found to be significantly more accurate, as these data enable classification of some land cover types that single images do not [38, 177]. For instance, soy and corn are both winter crops and will appear similar in a single image, yet their different growth rates are clearly evident using SITS data. Another noteworthy example is bare ground. It is an inevitable part of the agricultural cycle that sites will be bare during periods of the year. If an image is taken during this period, a classifier will only be able to label this site as bare

ground, alternately SITS data allow us to observe the crop growing as time progresses and classify it accordingly.

1.4 Time Series Classification

Using SITS to create land cover maps is an application of Time Series Classification (TSC), a field of machine learning concerning the labelling of time series data.

Formally, time series data can be defined as a series of values that is indexed in time order. Everyday examples of this would be the number of internet searches per day for a specific query or share prices on the stock exchange.

The analysis of time series requires different methods when compared with the analysis of non-time series data for a multitude of reasons. The unique properties of time series data mean that it can often be resampled, shifted, contracted or expanded in the temporal dimension without modifying its inherent meaning and any analysis performed must account for this. For example, if we consider two satellites with different revisit frequencies capturing images of a crop, while the resulting time series will each have a different number of timestamps, they will both have a similar profile when considered annually. Analyses must also take into account autocorrelation, the correlation between a point and those preceding it. For example, the density of vegetation present in a forest is highly correlated with the density observed a week prior (excluding anthropogenic intervention). That is, the dominant trends occur gradually—*i.e.*, vegetation growing gradually or the predominant colour of the vegetation changing seasonally—and therefore the qualities of the vegetation will be highly correlated with those observed recently.

TSC refers to the scenario where we have a dataset of multiple time series with each having a specific class label, and we would like to infer a model that correctly predicts the class of an unobserved time series based on the characteristics of those we have previously seen. A trivial example of TSC is shown in Figure 1.5, where the luminosity of the light emitted by individual stars (with their phases aligned) is recorded and displayed at regular time intervals. The plots (known in astronomy as light curves) for 4 individual stars are shown and represent 2 different classes of stars—Cepheids (red) and Elipsing Binaries (blue). In this example it is plain to see how the light curves can be used to differentiate between classes with Elipsing Binaries being defined by a high intensity peak around time 250, and Cepheids having a gradual increase followed by a gradual decrease over the period of measurement. As a time series classification problem, we would label an unknown type of star by comparing its light curve to these known light curves and labelling it as belonging to the class of the curves to which it is most similar. While this example is trivial and the two classes are distinguishable by eye, the majority of time series classification

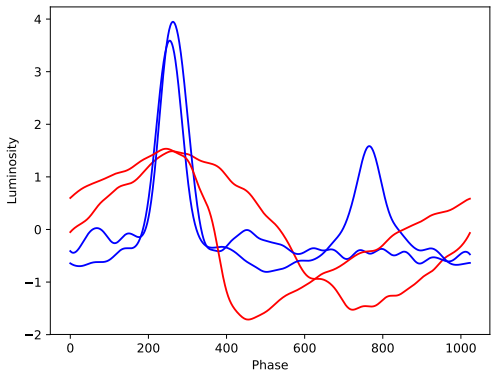


Figure 1.5: The luminosity time series emitted for two different classes of stars (known as their light curves)—Cepheids (red) and Elipsing Binaries (blue) [28]

problems are far more complex, with classes defined by subtle differences (including a third class of stars, RR Lyrae, whose light curves are very similar to Cepheids).

A TSC problem (usually) cannot be treated a classical attribute-value classification problem due to the structure of time series data. In general, if we take each of the timestamps of a series and treat them as independent attributes, a *traditional* machine learning classification algorithm will return a relatively low accuracy of predicting the classes of new data. This is because the distinguishing signal of a class is not necessarily aligned with a specific time stamp, and thus we need a classification algorithm that takes this information into account.

TSC problems can be either univariate or multivariate time series. In an m-dimensional multivariate TSC problem, the input to the classifier will be m time series and the classifier will learn to discriminate between classes based on any combination of these input time series.

1.5 Univariate Satellite Image Time Series

SITS datasets are recorded as a multivariate time series dataset, where each spectral band of the satellite is represented by a time series. However various studies have shown that for the purposes of *some* classification tasks they can be transformed into a univariate time series dataset by using the Normalized Difference Vegetative Index (NDVI) without no degradation to performance [16, 175, 57, 3].

NDVI is a dimensionless index that is used to approximate the density of green vegetation in a satellite image. It is calculated using the values of the red spectral band and the near-infrared (NIR) spectral band and the following equation:

$$NDVI = \frac{NIR - Red}{NIR + Red} \tag{1.1}$$

The resulting index theoretically has a value between -1 and 1, however negative values often only occur in the atmosphere or from specific materials. If the reflectance of the NIR spectral bands is far greater than that of the red spectral bands, the index will have a value of 0.5 or above and the image is likely to be dense vegetation such as a forest. If there is only a small difference between the reflectance of the NIR and red spectral bands, the image is likely to correspond to land with little vegetation such as tundra, grassland, or desert. Urban environments generally have NDVI values just above zero.

1.6 Research Aims and Thesis Overview

It is complicated to automate production of large scale, high-resolution land cover maps from SITS with current techniques for two major reasons: (1) there are no existing time series classifiers suitable for the scale of SITS data; and, (2) modern machine learning classifiers require very large quantities of labelled data, which is generally difficult (*i.e.*, expensive, time-consuming, laborious) to acquire at such scale.

The result is that existing maps are often a compromise between both (1) high accuracy and the number of classes defined by the map and (2) the resolution and the scale of the map. For example, the two recent global land cover maps [190, 64] each have an overall accuracy of less than 70%, likely due to the small quantity of available training data (approximately 90 000 instances). While recent maps of South America [62], Japan [149] and South-East Asia [30] are far more accurate (>80%), all display fewer than 8 land cover classes. Another global land cover map [64] was produced using a classifier that implemented decision rules defined by an expert user and were specific to each land cover class. This technique produced high accuracy but was not generalisable, and was extremely labour-intensive and therefore is unlikely to be reproduced or updated.

Addressing these two problems–finding a scalable time series classifier and producing maps where labelled data are scarce–forms the basis of the following doctoral thesis.

1.6.1 *Big* Time Series Data

The benchmark for testing and ranking time series classification algorithms is their performance on the UCR archive [37], a repository of time series datasets for classification maintained by researchers at the University of California–Riverside. The archive consists of 128 datasets of various types including images, sensors, mass spectrometry, and medical applications (the data for the light curves of different stars shown in Figure 1.5 is taken from the UCR repository).

The datasets vary in size from as few as 12 training instances for the StandWalkJump dataset to 139 883 for the MosquitoSound dataset. The lengths of the series similarly vary from 8 (PenDigits)

to 236 784 (DucksAndGeese). While the sizes of the datasets and their lengths vary greatly within the archive, even the largest is many orders of magnitude smaller than some modern time series datasets, including the SITS dataset for land cover mapping. The repercussions of this are two-fold: first, the algorithms considered state of the art in time series classification have paid scant regard to scalability, and secondly, large time series datasets are being classified with methods that are not designed for time series data.

The Sentinel-2 satellite constellation records 13 decimal values for each 10m by 10m pixel every 5 days, meaning that even at a small scale, a SITS dataset consists of millions of time series. This magnitude means that most existing large-scale maps are produced using a Random Forest model, a classifier that treats the input features as independent, discarding all benefit of having temporal data [149, 61, 62].

A scalable time series classification algorithm promises both higher resolution land cover maps–by being able to classify a greater quantity of data–and higher accuracy land cover maps–by utilising the temporal information embedded in the data and by being able to train on a greater quantity of data in a given time.

1.6.2 Scarcity of Labelled Data

The accuracy of modern machine learning algorithms is highly dependent upon the availability of a large quantity of labelled training data [71, 155]. Training data are instances that have been assigned a class label prior, which then help the algorithm learn the characteristics or features of a given class of data. This can present a major problem in land cover mapping as training data simply are not available in many areas.

This absence of data occurs for three main reasons:

1. Labelled data are both expensive and time-consuming to acquire at the resolution of the latest Earth observation satellites (10 meters in the case of the Sentinel-2 constellation).

2. The data are often specific to their location. For example, Figure 1.6 shows the mean Normalized Difference Vegetation Index of pea crops growing in 3 different regions of France. It is clear that even within one country, the same crop can take on three distinctly different profiles, meaning we cannot simply *borrow* data from a nearby region when training a model. Agricultural practices, water, soil, weather and many other factors can also contribute to variation between spectral profiles of crops.

3. Land cover changes over time, and thus we cannot reliably use old labelled data to train a new model as it may no longer be accurate.

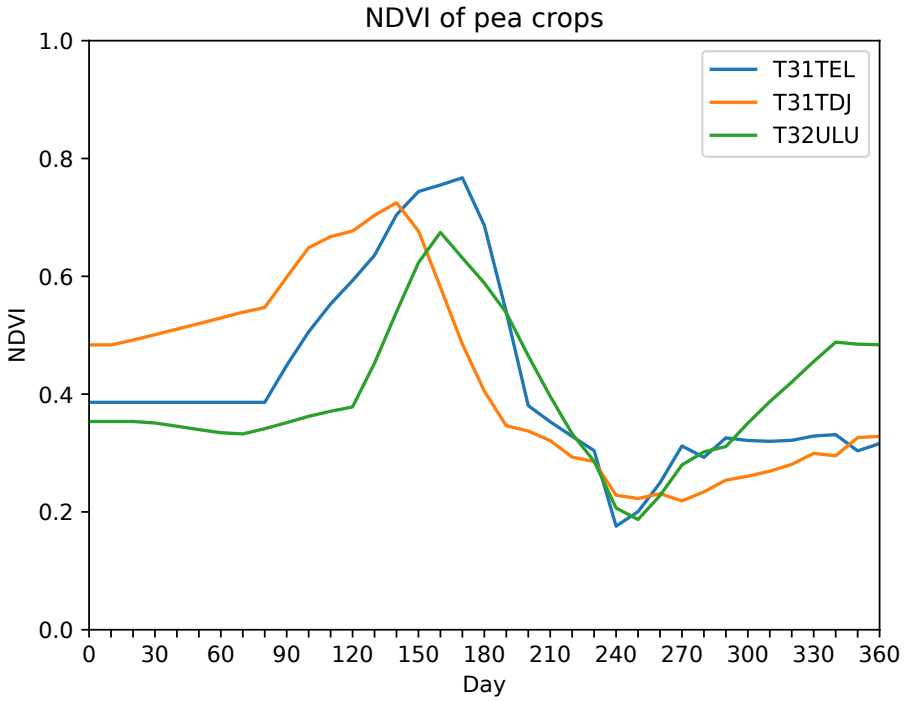


Figure 1.6: Mean NDVI and quartiles of pea crops in 2016 for three Sentinel-2 satellite tiles located in France (Figure 6.1 shows the exact locations of the tiles).

Consequently, labelled training data that are both recent and sourced from the study area are either scarce or non-existent, making utilizing the state-of-the-art machine learning methods to create land cover maps extremely challenging.

Researchers have proposed three main approaches to tackle this problem: (1) algorithms that knowingly using out-of-date reference data [161, 61]; (2) active learning strategies [128, 115]; and (3) domain adaptation (DA) [169].

The first approach best suits the scenario in which accurate historical data exist (which is often not the case for the same reasons outlined above) and state-of-the-art algorithms can be used to identify which of the historical data are now outdated [55, 35, 125, 10]. The second approach best suits the scenario where sufficient computational resources are available and where further data collection is feasible (*i.e.*, timely, affordable) [168], and therefore lends itself favorably to smaller scale applications. The third approach, DA, is best suited to a situation where labelled training data from a different geographical region or a previous year *are* available and can be used to train a classifier [169].

DA holds great potential for land cover mapping as labelled training data are available for *some* geographical regions and therefore a successful DA technique would mean that users could produce high-accuracy land cover maps of areas where no training data are available *and* without having to perform large scale data collection.

For this reason I chose to focus a period of my Ph.D candidature on developing DA methods for land cover mapping.

1.6.3 Research Aims

The main objective of my Ph.D research is to applying machine learning to better facilitate Earth observation. I aim to develop methods to address the two major issues in land cover mapping discussed in Sections 1.6.1 and 1.6.2, respectively. Specifically, my research will have the following aims[1]:

1 **To develop a time series classification algorithm with the ability to learn from and classify large datasets in a practical time frame.** This aim will be developed with, and demonstrated on, the SITS dataset for land cover mapping. This scalable algorithm must also be competitive with the state of the art in terms of classification accuracy as measured on the UCR archive.

2 **To successfully investigate and implement deep learning algorithms for time series classification.** While deep learning has transformed many fields of machine learning, most notably computer vision, it has not yet been shown to be competitive in TSC. I also seek to both identify the elements of the model architecture and features of datasets that cause the models to perform optimally or poorly.

3 **To compare and contrast the leading unsupervised domain adaptation methods for use with the SITS dataset.** This aim will assess the suitability of existing DA methods in the situation where no labelled data from the area we wish to map is available.

4 **To develop a domain adaptation method to facilitate the production of land cover maps from SITS data in areas where labelled training data are scarce or unavailable.** This aim seeks to develop a DA method to produce land cover maps for regions that have little-to-no available reference data. It is envisioned that the method will be able to generalise to other time series data also.

1.6.4 Thesis Overview

The remainder of this Ph.D thesis is divided into two parts: (1) Scalable Time Series Classification and (2) Domain Adaptation for Time Series data.

Part I examines scalable classification algorithms and their use with SITS data. It begins with Chapter 2 where I introduce the current state-of-the-art algorithms in time series classification,

[1] I note that these refer to the state of knowledge at my time of commencement, not at the time of writing, and that some of these areas (*e.g.*, deep learning for TSC) have advanced rapidly during the period of my candidature.

with particular attention being paid to their computational complexity. In the next chapter I present Proximity Forest: a scalable and accurate time series classification algorithm, and one of the major contributions of my research. Chapter 4 investigates deep learning algorithms and their application to time series classification. It will begin with a description of deep learning and then draw parallels between the fields of computer vision and time series classification. In the final section of the chapter I will discuss InceptionTime, a class-leading deep learning algorithm for time series classification, and present the concept of receptive fields for one-dimensional convolutional neural networks for time series.

Part II of this thesis begins by discussing DA, a technique used in situations where training data are scarce. Chapter 5 then presents a review of the current literature in both unsupervised DA—the case where labelled data are completely unavailable—and semi-supervised DA—where some, but not substantial, labelled data are available. Following this, I present experiments in Chapters 6 and 7 motivating the specific need for DA for classifying SITS data and demonstrating the value of labelled target data for this task at present, respectively. In Chapter 8 I present a general semi-supervised DA method in named Encoda, based on combining model ensembling and co-training. In the final Chapter of Part II I will present a deep learning-based, semi-supervised DA technique specifically designed for land cover mapping called Sourcerer. This work represents another major contribution of my Ph.D research.

In the conclusion to this thesis I will summarise the research produced during my candidature and highlight the contributions of my work. I will also discuss the limitations of my research and the scope for future work in these areas.

Part I

Scalable Time Series Classification

2

Related Work and Background

Researchers have approached the task of classifying time series datasets in a variety of ways including Nearest Neighbours-based algorithms, bag-of-words methods, interval-based methods, decision tree-based methods, deep learning models, and ensemble methods. Traditionally, the performance of algorithms was assessed solely on classification accuracy, in particular their accuracy on the UCR repository (presented in Section 2.1 below), however a recent trend has emerged emphasising the importance of the scalability of an algorithm when considering the massive size of modern time series datasets.

In this chapter I will present a substantial review of the literature regarding time series classification. First, I will introduce the UCR repository for time series classification, a collection of time series datasets used to compare the performance of algorithms. Then in each subsequent section of the chapter, I will present a family of time series classification algorithms, where the families are defined to include methods with a discernible commonality (*e.g.*, being tree-based methods, deep learning etc). This chapter has a focus on the particular interest of my research: scalable training and classification.

It is also noted that throughout this chapter the variable n is used to represent the number of training instances and the variable ℓ is used to represent the length of each individual time series. I note that for ease of exposition, I have made the simplifying assumption that all time series in a given dataset are of the same length but that it would be straightforward to extend the techniques discussed to variable length series.

2.1 UCR Repository for Time Series Classification

The UCR Time Series Classification Archive [37] is a repository of 128[1] time series datasets compiled and maintained by researchers at the University of California–Riverside and the University of East Anglia. The datasets have been donated by various researchers since 2012 and are made openly available online: http://www.timeseriesclassification.com. Over 1 000 papers in time series classification and its applications have used datasets from this repository [37].

The datasets originate from various sources including images, motion sensors, mass spectrometry, synthetic examples, and medical applications such as electrocardiograms. The number of classes per dataset varies: 42 of the 128 are binary problems, 7 datasets have over 40 classes, while the ShapesAll dataset is a 60-class problem. The datasets vary in size from as few as 12 training instances for the StandWalkJump dataset to 139 883 for the MosquitoSound dataset. While the test sets vary from 15 for each of the StandWalkJump and AtrialFibrillation datasets to 139 883 for the MosquitoSound dataset. The lengths of the series similarly vary from 8 timestamps for the PenDigits dataset to 236 784 timestamps for the DucksAndGeese dataset.

I highlight that the UCR archive includes a land cover classification dataset named Crop derived from Sentinel-2 SITS data. It is a 24-class problem and consists of 7 200 training and 16 800 test instances, which is a small subset of the data used to create large scale land cover maps. This data was subsampled from the original dataset, first presented in [132], specifically for inclusion in the UCR archive.

The UCR archive has created a benchmark for time series classification algorithms, where the best performers are ranked based on their performance on the whole repository, as opposed to cherry-picking datasets that produce favourable results.

2.2 Distance-based Classifiers

The most longstanding method for classification of time series is by using the k-Nearest Neighbours (NN) algorithm with a distance measure to determine the similarity of two series. In this algorithm,

[1]The work presented in Chapter 3 was completed prior to the UCR archive expanding to 128 datasets. At the time the majority of the work was completed, the repository consisted of 85 datasets

query data (with unknown class) is compared to each data in a training set (with known classes), and the query data is classified as belonging to the class that appears most frequently amongst the k datapoints that it is most similar to. For time series classification, k is generally chosen to be 1, that is to say that the query time series is assigned to the same class as the class of the single time series from the training data that it is most similar to.

While k is generally consistent amongst time series classifiers, the algorithms in this family differ by the distance measure used to quantify the similarity (or dissimilarity) between time series. Time series have particular properties that have led to the development of specific similarity measures: they are often auto-correlated (the value of the time series at a timestamp is likely to be close to the ones just before and after), and often include non-linear distortions in the time axis (for example, because the start of the phenomenon of interest is delayed, or because sections of the phenomena are faster or slower). This has led to the development of specific similarity measures, of which most have an ability to realign the series along a common intrinsic timeline. These measures include Dynamic Time Warping (DTW) [139, 140], Derivative DTW (DDTW) [87, 67], Weighted DTW (WDTW) [82], Longest Common Subsequence (LCSS) [176], Edit Distance with Real penalty (ERP) [25, 26], Time Warp Edit distance (TWE) [112] and Move-Split-Merge (MSM) [153].

The most well-known and widely used similarity measure is DTW. In fact, for more than a decade, the k-NN algorithm combined with the DTW measure was extremely difficult to beat for most time series classification problems [179]. DTW finds the distance between two time series by finding the optimal alignment between them, under restriction of parameter w. It does this by stretching and contracting one series to best match the other.

For example, Figure 2.1 shows the difference between using Euclidean Distance and DTW to measure the similarity of two time series. The Euclidean distance measure simply aligns the timestamps from each series and calculates the distance, whereas DTW allows for timestamp i on the green time series to align with its closest timestamp on the blue time series, which is timestamp $i + 2$.

DTW has a parameter w, which controls the maximum length of the warping window, *i.e.*, the maximum time lag allowable between matching points, to prevent pathological alignments and improve the time complexity of a single comparison (from $O(\ell^2)$ to $O(w \cdot \ell)$. All time series distance measures use one or more parameters and optimal values for these are often first learned on the training data via cross-validation. Thus, the training complexity of this family of methods is usually quadratic with the size of the training data due to the tuning of the measures' parameters. Furthermore, when using a k-NN classifier, the complexity of classification is at least linear with the size of the training data. Compounding these issues further is the fact that most measures themselves have a computational complexity that sits between linear and quadratic with the length

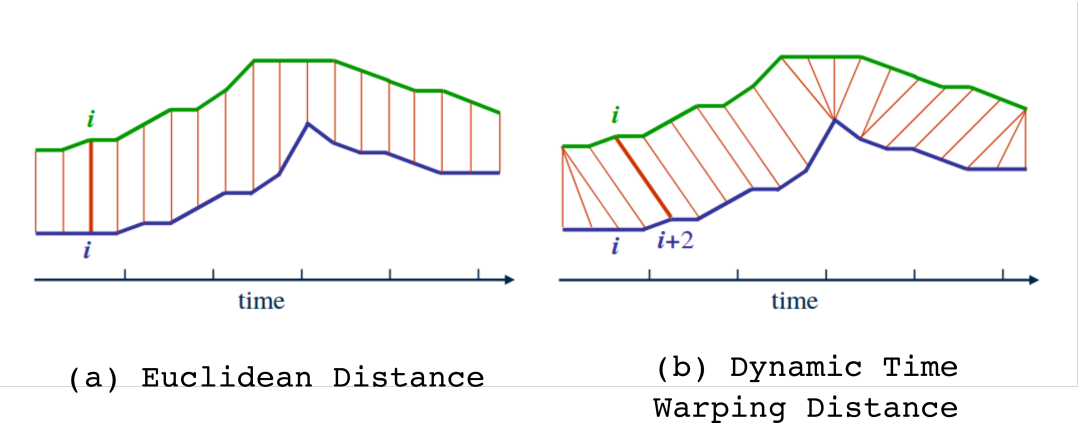


Figure 2.1: An illustration of the difference between two time series distance measures: (a) Euclidean distance and (b) Dynamic Time Warping distance (DTW)

of the series [7].

Targeted techniques have been developed in attempt to alleviate the second issue of scalable classification. Data reduction techniques aim at simplifying the training database without penalizing the classification quality; either by directly removing objects from the original database [123, 173] or by summarizing the database and replacing sets of time series with representatives using average time series [131, 129, 130, 113]. Indexing is more difficult on time series than it is on traditional data, mostly because time series measures do not obey the triangular equality (at most obeying a relaxed p-triangular inequality [100]), which makes exact pruning very inefficient (however, a general approach to improving the efficiency of NN searches in non-Euclidean space is available in [101]). To perform exact indexing, the main research effort has been put onto developing lower bounds (and mostly for DTW) [86, 100, 159, 74].

While most distance measure algorithms are many years (even decades) old, on some time series classification tasks they still return state-of-the-art classification accuracy. When compared to each other, it has been found that no single distance measure paired with a NN classifier significantly outperforms all others [103]. This observation lead researchers to the development of the Elastic Ensemble (EE) [103], an ensemble model of 11 different k-NN classifiers, each learned with a different time series distance measure (with their parameters tuned accordingly). EE significantly outperforms all individual distance measure-based k-NN classifiers but its training complexity is quadratic with the number of training instances ($O(n^2 \cdot \ell^2)$) and is therefore not practical for use on large datasets.

2.3 Shapelets

The aim of shapelet algorithms is to find consecutive subsets of time series, subseries, that can discriminate between different classes. To classify a time series, the learned shapelet is placed at the

best position in the time series (based on Euclidean distance), and the *matching* of the shapelet to the time series correspond to its distance at this best position. The original shapelet-based classifier [188] inserted this algorithm at the node of a decision tree as a splitting criterion. This algorithm has a high training complexity ($O(n^2 \cdot \ell^4)$) due to the large number of candidate shapelets and the repeated scanning of the data. Subsequent research has focused on optimising the original algorithm to address both classification accuracy and scalability, notably Fast Shapelets [134], Learning Time Series Shapelets [68] and Shapelet Transforms [75].

Shapelet Transforms (ST) is the highest performing classifier from this family. It begins by identifying the best k shapelets in a single scan of the data (the number of shapelets k can be reduced afterwards). The data is then transformed, or discretised, by defining an attribute to represent each shapelet with the value being the distance (usually Euclidean distance) between the shapelet and the best position in the time series. The transformed dataset can now be used with any classifier or ensemble of classifiers (such as in [8]). While ST is considered a state-of-the-art classifier based on accuracy, it has little potential to scale to large datasets given its training complexity of $O(n^2 \cdot \ell^4)$.

A recent shapelet-based method, EXplainable Convolutional Neural Networks (XCNN) [180], incorporates deep learning and a novel regularisation method to find shapelets that are both discriminative of a class and explainable to the user. The authors demonstrate that shapelets discovered by other methods, although useful for classification, often do not represent recognisable subseries and therefore provide little insight into understanding the problem.

2.4 Bag-of-words Approaches

Bag-of-words algorithms work by transforming the time series into a bag-of-words model by discretising subseries of the overall series into letters. The relative frequency of strings of these letters (*i.e.*, words) is then used to differentiate between classes. The time series can be discretised in the time domain, such as in Bag of Patterns [102], the frequency domain, such as in the Symbolic Aggregate Approximation-Vector Space Model (SAX-VSM) [147], the Bag of SFA Symbols (BOSS) [142], and Word eXtrAction for time SEries cLassification (WEASEL) [144], or both, such as in Multiple Representation Sequence Learner (MrSEQL) [97].

The BOSS algorithm transforms the time series into words using a Symbolic Fourier Approximation (see [143]), which makes it robust to noise and delivers high classification accuracy. It is however of limited use on large datasets as it has a training complexity of $O(n^2 \cdot \ell^2)$ [7]. The authors identified this as a weakness in the original work and subsequently produced similar approaches with improved scalability. The Bag of SFA Symbols in Vector Space (BOSS-VS) [146]

significantly improves the efficiency of BOSS but this comes at a great decrease in classification accuracy. The same authors also proposed WEASEL, which improves on the computation time of BOSS and on the accuracy of BOSS-VS, but has a very high memory complexity (my experiments in Section 3.3 show that it does not scale beyond 10 000 time series). In this way, WEASEL is more optimised for speed on small datasets than for scalability. MrSEQL is a recently published classifier from this family that was shown to be more accurate than WEASEL while also requiring significantly less memory on the single example presented in the paper[2].

2.5 Tree-based Approaches

A number of decision tree approaches have been developed for time series classification. This section provides a greater depth of explanation of the methods than other sections to enable comparison to Proximity Forest (my method that will be presented in Chapter 3) as it is also a tree-based algorithm.

Time Series Forest (TSF) [42] first derives summary features for all time series by dividing them into intervals and summarising each interval by its mean, standard deviation and gradient. Then a strategy similar to a random forest is employed to select between a random subset of features at each node in each of an ensemble of trees. A novel selection criterion is used that considers both entropy gain and the margin by which a feature separates the classes. This continues until the entropy gain ceases to improve, at which stage the node is defined as a leaf. TSF has been shown to be a reasonably accurate classifier, ranking behind EE in overall accuracy without being significantly worse [7]. However, its main virtue is computational efficiency as TSF learns in $O(n \cdot \log(n) \cdot \ell \cdot k)$ for a forest of k trees, which is a much lower complexity than the current state of the art.

Generalized Random Shapelet Forest (gRSF) [85] extracts a shapelet from a randomly chosen time series and finds the distance between this time series and each other time series. The data is then split according to whether it is above or below a threshold distance to the representative shapelet. This is applied recursively until the node is either pure or the number of instances remaining at a node is less than 3. As mentioned in Section 2.3, the main pitfall of shapelet-based methods is the high computational cost of finding candidate shapelets and comparing shapelets to other time series. The gRSF minimises this issue by randomising many of the model choices–a candidate shapelet is generated from a randomly chosen time series by choosing a random starting point and random length, this is repeated r times and the best candidate is chosen for a given split. This algorithm returns moderate classification accuracy on the UCR repository [7].

[2]The publication did not apply the classifier to the UCR archive and therefore we have no broader comparison of it against the state of the art.

The first tree-based algorithm to reach the performance of the state of the art is my method—Proximity Forest. Proximity Forest is an ensemble of decision trees with a novel time series-specific splitting criterion at the node. The splits utilise the 11 distance measures that comprise EE, however they are parametrised at random to reduce computational cost. This randomisation also aids to increase variance between individual classifiers and consequently improve overall performance of the classifier. One major benefit of using a tree-based method is that it is a divide-and-conquer algorithms and is therefore inherently scalable to large datasets. Its training complexity is quasilinear with training size and quadratic with length of the time series ($O(n \cdot \log(n) \cdot s \cdot c \cdot \ell^2 \cdot k)$, where s is the number of candidate splits, c is the number of classes).

Since its publication, Proximity Forest has been improved upon by TS-CHIEF [150], which improves its overall accuracy by incorporating new dictionary-based and interval-based splitting criteria at the node. Like Proximity Forest, the training complexity is quasilinear with the number of instances and quadratic in length of the time series. TS-CHIEF was the first method published to have an average ranking higher than HIVE-COTE on the UCR repository.

A recent method, Canonical Interval Forest (CIF) [117], upgrades TSF by replacing its three summary features with a set of 22 features, referred to as the CAnonical Time-series CHaracteristics (catch22) [109]. Their experiments show that, as a tree-based method, CIF is inherently quick to learn and classify and is also competitive with many high performing algorithms (including Proximity Forest and BOSS). The paper suggests that the classifier's main purpose is not as an individual classifier but rather as a component of HIVE-COTE ensemble (presented in Section 2.6).

2.6 Ensemble Approaches

Ensemble approaches are combinations of multiple classifiers, where each contributing algorithm can be weighted to maximise classification accuracy. The time complexity of ensemble methods is that of the slowest constituent classifier. I note that some of these approaches have been discussed in prior sections as they are based around one main type of classifier, *e.g.*, EE is based around distance measures.

The Collective of Transformation-based Ensembles (COTE) [8] is an ensemble comprising 35 classifiers across four time series domains: time, frequency, change and shapelet transformation. For the time domain, COTE uses the 11 distance measures of EE, while in the other three domains, classifiers are recruited from outside time series classification – k-nearest neighbours, naive Bayes, decision trees, random forest, rotation forest, support vector machines (two models) and a Bayesian network approach. On the benchmark UCR datasets, COTE is considered "clearly superior to other

published techniques" as it has the highest average accuracy [7]. However, its time complexity is bound by that of the Shapelet Transform, which is $O(n^2 \cdot l^4)$ and the parameter searches for the distance measures comprising EE, some of which are $O(n^2 \cdot l^3)$.

In subsequent years, the COTE algorithm was superseded by the Hierarchical Vote Collective of Transformation-based Ensembles (HIVE-COTE) [105], which adds new dictionary and interval-based classifiers, and employs a novel ensembling method to improve classification accuracy. At the time of publication, HIVE-COTE set a new benchmark for classification accuracy on the UCR repository, however its training complexity is still bound by the complexity of Shapelet Transform and the parameterisation of the distance measures in EE, and therefore is largely impractical for use on medium to large datasets. The most recent version of HIVE-COTE, called HIVE-COTE 1.0 [6], has significantly improved its computation time by removing the EE component of the ensemble, but is still behind TS-CHIEF and ROCKET (see Section 2.8) in terms of accuracy.

2.7 Deep Learning

Deep learning, in particular Convolutional Neural Networks (CNN), have had much success in image recognition in recent years [73, 157, 152], and due to the similarity of image data to time series data, researchers have started investigating their use for time series classification [182, 33, 96].

The first significant study of deep learning for time series classification [182] compared 3 models: a FFNN, a CNN, and a CNN with residual links (ResNet)[3]. The results found that the *standard* CNN performed best on the datasets of the UCR repository, with an only marginally worse overall accuracy than COTE.

An earlier study [33] used a variation of CNN by applying 3 transforms to the data, running each through a convolutional layer individually before concatenating the output and feeding this into another CNN model. The results stated that this method outperforms the state of the art on a majority of the datasets of UCR repository, however the heavy data preprocessing make it difficult to reproduce or verify [49].

A review of these architectures found that none reached the overall accuracy of the state of the art on the UCR repository [49]. It was suggested that this is most likely because deep learning models require very large training sets to be optimised [107], while the UCR repository contains relatively small datasets. For example, in computer vision—a field where deep learning has been very successful—the benchmark ImageNet dataset [43] has over 1 million images in its training set, while the largest dataset in the UCR repository has a training set size of only 139 883 time series.

Some researchers have suggested that it may be possible to address this issue through data augmentation [33, 96, 48, 54] or ensembling [49].

[3]The ResNet architecture is explained further in Section 4.1.3

The most notable deep learning model for time series classification is InceptionTime [50], which is an ensemble of CNN models based on the Inception architecture [157]. InceptionTime used ensembling to overcoming the overfitting issues experienced by previous deep learning architectures and was found to be competitive with HIVE-COTE on the UCR repository. I was a co-author on the InceptionTime publication during my PhD and it is explained and discussed in Chapter 4.

2.8 ROCKET

A significant recent advancement in time series classification is a classifier called RandOm Convolutional KErnel Transform (ROCKET) [40]. Based on the success of CNN models, ROCKET also uses convolutional kernels, but rather than learning kernel weights as per a CNN, it employs random convolutional kernels to identify discriminative patterns in time series. By using random kernels and not learning them, it allows the algorithm to use a very large number in a short timeframe. It then uses the feature maps produced by the convolution operation as the input to a multinomial regression. This simplicity makes ROCKET extremely fast, learning from 1M time series in 1 hour 15 minutes, compared to 16 hours for Proximity Forest, while also producing a comparable classification accuracy in this time. When the number of kernels is optimised ROCKET achieves state-of-the-art performance on the UCR repository.

2.9 Overall Comparison

The generally accepted method of comparing the performance of multiple classifiers on multiple datasets is by using critical differences [41]. First the accuracy of each classifier is ranked for each dataset, then these ranks are averaged for each classier. Finally, the average ranks are compared using a critical difference based on a Wilcoxon signed-rank test.

Specifically, the difference between average rankings is significantly different at level α, when the critical difference is greater than:

$$CD = q_\alpha(A) \cdot \sqrt{\frac{A(A+1)}{6 \cdot N_d}} \tag{2.1}$$

Otherwise, it cannot be said that there is a statistical difference between the classifiers.

This information is usually displayed as a critical difference diagram with a black line connecting the algorithms that are not significantly different from one another. The critical difference diagram for the highest performing time series classification algorithms is shown in Figure 2.2

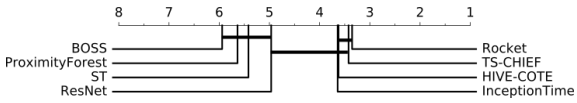


Figure 2.2: The mean rank of the state-of-the-art time series classification algorithms on the 85 datasets of the UCR repository (diagram from [40]).

3

Proximity Forest

Chapter 2 highlighted that recent decades have seen numerous classifiers and distance measures specifically designed for time series classification. It also showed that none of the algorithms with state-of-the-art accuracy are capable of scaling to large time series datasets. We have seen that some do not scale in the learning phase (ST, EE, COTE), and others require a scan of the training database to perform each classification (EE, COTE). Meanwhile, those that do scale to medium-sized datasets, such as BOSS-VS, compromise on accuracy in order to do so (as will be demonstrated in Section 3.3). Throughout the development and advancement of much of the current state of the art, scalability has usually been secondary to accuracy. This is because computation time is not a significant issue when considering datasets with only a small number of time series, such as those in the UCR repository of time series datasets (Section 2.1)[1].

In contrast, the SITS dataset for land cover mapping includes millions to billions of time series, a scale that is too large for any of the existing time series classification algorithms to learn from

[1]The work presented here was completed prior to the UCR archive expanding from 85 to 128 datasets. The statistics quoted in this chapter regarding dataset sizes and lengths refer to this earlier 85-dataset repository, thus accounting for any differences between the statistics quoted in this chapter and in Section 2.1

and classify in a functional time frame. For example, on a SITS dataset with 1 000 000 instances, the time required to learn an EE classifier (a key component of the COTE ensemble) is 73 000 days, or over 200 years. This is even more startling knowing that the series in this dataset are reasonably short–only 46 timestamps–compared to some applications.

This gap in knowledge was the motivation for my initial research: To develop a time series classification algorithm that is both competitive with the accuracy of the current state-of-the-art algorithms when measured on the UCR archive and also scales to the magnitude of the SITS dataset.

My PhD research lead to the first algorithm addressing this shortfall, a tree-based time series classification algorithm named Proximity Forest. It is comprised of an ensemble of decision tree classifiers with a novel time series based splitting criterion at the node. Proximity Forest can learn from 1 million time series in approximately 17 hours, compared to 200 years for the state of the art, while also rivalling the accuracy of the then class-leading COTE algorithm on the datasets of the UCR repository. This meant that it was the first time series specific classifier capable of classifying SITS and producing land cover maps in a reasonable time frame.

Another major legacy of Proximity Forest is the research that it inspired. Since its publication, researchers have published numerous time series classifiers with state-of-the-art accuracy and an emphasis on scalability (for *e.g.*, see [40, 150, 50]).

This chapter begins with an outline of Proximity Forest and an overview of how the algorithm filled an existing gap in knowledge. The following sections will then present the model and two key algorithms: (1) learning a Proximity Forest and (2) classifying with a Proximity Forest. In the forth section I will present experiments demonstrating the performance of Proximity Forest, including studies of its scalability, accuracy, and parameters. In the conclusion to the chapter I will provide a summary of Proximity Forest and highlight the contributions of this portion of my candidature.

I note that this chapter is heavily based on the work published in:
Lucas, B., Shifaz, A., Pelletier, C., O'Neill, L., Zaidi, N., Goethals, B., Petitjean, F., Webb, G.I.: Proximity Forest: an effective and scalable distance-based classifier for time series. Data Mining and Knowledge Discovery 33(3), 607–635 (2019).

3.1 What is a Proximity Forest?

Proximity Forest is a time series classification algorithm comprised of an ensemble of decision tree classifiers with a novel time series based splitting criterion at the node. A tree-based classifier was chosen to achieve scalability, which is an inherent property of trees due to their divide-and-conquer

nature. At each level of the tree, the data are divided into multiple subsets, and as a result the trees are on average of depth $O(\log n)$, *i.e.*, increasing sublinearly in depth relative to training set size.

Proximity Forest merges the strategy of learning decision trees where splits are based on similarity to chosen time series exemplars [11, 45, 185] with the strategy of forming forests of decision trees in which the exemplars are chosen at random [141]. Where our approach is distinguished from previous work is by the random selection of both exemplars and distance measures; randomness that serves multiple purposes. First, it avoids the high time complexity incurred by previous approaches in their search for suitable exemplars on which to split [11, 45, 185]. Second, it serves to increase diversity between the trees within the ensemble, thereby lowering bias while allowing ensembling to obviate the resulting increase in variance.

Proximity Forest incorporates three critical elements into its design:

1. It utilises 11 existing distance measures recruiting over 30 years of research into designing consistent measures for time series.

2. It is specifically designed to have a high variability between the different individual classifiers of the ensemble. This results in an improved overall classification accuracy compared with a single classifier, based on the principle of ensemble methods. In general, averaging the predictions of multiple models each having high variance and low bias results in an ensemble classifier with a lower total error than any single classifier. This is analogous to how a Random Forest classifier, another ensemble of decision trees, will only learn from a fraction of the available features for each individual node in order to introduce variability between the trees [20]. This observation is important, because the design does not learn an individual tree to maximise its accuracy; substantially different design choices would be required to design an optimal single-tree model. Thus, the learning of individual trees is designed with the intention of maximising overall classification performance.

3. Proximity Forest is designed to be extremely scalable with an average-case learning complexity of $O(n \log(n) \cdot \ell^2)$ and a classification complexity of $O(\log(n) \cdot \ell^2)$ per tree for n training time series of length ℓ. This contrasts with the state of the art learning in $O(n^2 \cdot \ell^3)$ (EE) or $O(n^2 \cdot \ell^4)$ (ST, COTE). Again here, some different choices might have occurred in the design of the algorithm if scalability was not a design objective.

At the time Proximity Forest was developed, the COTE algorithm was the state of the art in terms of classification accuracy [7], however its learning phase is bound by the runtime complexity of two of its component classifiers—ST and EE. The large runtime complexity of COTE is largely due to the fact that EE does not abstract much information during the learning phase, and therefore

has a significantly greater number of processes to complete during testing. A corollary of this is that a distance-based classifier that learns faster than EE for the same level of accuracy would also present an improvement to COTE.

It is for this reason, as well as allowing direct comparison between the two competing architectures of nearest neighbour and trees using distance-based splits, that the design of the Proximity Forest algorithm incorporates many elements of EE—11 distance measures and similar parametrisation—and why the experiments presented in Section 3.3 emphasise a direct comparison of Proximity Forest against EE.

3.1.1 Training a Proximity Forest

We seek to learn a Proximity Forest from a training set comprising n labeled time series, each of which is of length ℓ, where the labels are integers from 1 to c.

A Proximity Forest is an ensemble of k Proximity Trees; the algorithm for learning a single tree is presented in Algorithm 1. A Proximity Tree is similar to a regular decision tree, but differs in the tests applied at internal nodes. Whereas a regular decision tree applies a test based on the value of an attribute (*e.g.*, if height $>$ 160 cm, follow the left branch, otherwise follow the right branch), each *branch* of an internal node of a Proximity Tree has an associated exemplar and an object follows the branch corresponding to the exemplar to which it is *closest* according to a parameterised similarity measure. We will see later how the exemplars and measures are chosen. A tree is either a leaf or an internal node.

An internal node has two fields, `measure`, a function $object \times object \rightarrow \mathbb{R}$, and `branches`, a vector of branches. Each branch has two fields, a time series (`exemplar`) and a tree to which an object is passed if it is nearer to the branch's exemplar than any other (`subtree`).

If all data reaching a node has the same class, *i.e.*, the node is pure, the $create\_leaf$ function creates a new leaf node and assigns this class label to its field `class`. This label is then assigned to any query time series reaching this leaf during the testing phase.

Choosing the splitting criteria

At each node, a Proximity Tree creates one branch for each class that exists in the data it receives from its parent. These exemplars are chosen uniformly at random among each class. The parameterised similarity measures are also chosen uniformly at random among a pool that will be described below after the main overview of the algorithm. It will also be explained how it is possible to *learn* with randomised trees.

Each node is constructed recursively from the root node down to the leaves. If the data at the node is pure—*i.e.*, all data belongs to the same class—then the node becomes a leaf and the

Algorithm 1: build_tree(D, Δ, R)

Input: D: a time series dataset
Input: Δ: a set of parameterised distance measures
Input: $\mathcal{C}$: number of candidate splits to consider at each node
Output: T: a Proximity Tree

if is_pure(D) **then**
  return create_leaf(D)

// create tree, to be returned, represented as its root node
$T \leftarrow$ create_node()

// Creating set $\mathcal{R}$ of $\mathcal{C}$ candidate splitters
$\mathcal{R} \leftarrow \emptyset$
for $i = 1$ **to** $\mathcal{C}$ **do**
  $r \leftarrow gen\_candidate\_splitter\,(D, \Delta)$ // generate random splitter
  Add splitter r to $\mathcal{R}$
end

// select best splitter; it splits using measure $\delta^\star$ and exemplars $E^\star$
$(\delta^\star, E^\star) \leftarrow \arg\max_{r \in \mathcal{R}} \text{Gini}\,(r)$

$T_\delta \leftarrow \delta^\star$ // retain measure for root node of T
$T_B \leftarrow \emptyset$ // T_B will store the branches under root node of T
foreach exemplar $e \in E^\star$ **do**
  // $D_e^\star$ is the subset of D that are the closest to e based on $\delta^\star$
  $D_e^\star \leftarrow \left\{ d \in D \;\middle|\; \arg\min_{e' \in E^*} \delta^\star(d, e') = e \right\}$
  $t \leftarrow$ build_tree$(D_e^\star, \Delta, \mathcal{C})$ // build subtree for that branch
  Add branch (e, t) to T_B // a branch is a pair (exemplar,sub-tree)
end

return T

recursion finishes.

At each node, a pool of $\mathcal{C}$ candidate splits is generated (Algorithm 2). For each candidate, a parameterised measure is chosen uniformly at random among a pool of such measures. It then selects an exemplar for each class represented at the node and passes the data down the branches by finding the closest exemplar (one per class) for each time series in the data using the distance measure associated with the node.

Once each candidate split has been created, the chosen split is the candidate that maximises the difference between the Gini impurity of the parent node and the weighted sum of Gini impurity of the child nodes. The construction of the tree is called recursively on each branch for the successful candidate; this constructs all subtrees. Once this is done, the tree is constructed.

Increasing the number of candidate splits per node will lead to an improvement of the quality of each split. However, it will also lead to an increase of the training time. The choice for the value of $\mathcal{C}$ will be discussed later in Section 3.3.3.

Algorithm 2: gen_candidate_splitter(D, Δ)

Input: D: a time series dataset.
Input: Δ: a set of parameterized distance measures to sample from
Output: (δ, E): a parameterized distance measure and a set of exemplars

$\delta \xleftarrow{\sim} \Delta$ `// sample a parameterized measure` δ `uniformly at random from` Δ

`// Select one exemplar per class to constitute the set` E
$E \leftarrow \emptyset$
foreach class c present in D **do**
  $D_c \leftarrow \{d \in D \mid class(d) = c\}$ `//` D_c `is the data for class` c
  $e \xleftarrow{\sim} D_c$ `// sample an exemplar` e `uniformly at random from` D_c
  Add e to E
end
return (δ, E)

The case of $\mathcal{C} = 1$: is selecting at random still 'learning'?

One might wonder what the tree is actually learning when it only considers a single candidate ($\mathcal{C} = 1$). In this case, no selection of the best possible split is performed. It is interesting to note that choosing splitting criteria independently of the output value has been studied before, a key example being Extremely Randomized Trees [58]. In that work, the authors showed that splitting completely at random still ensures consistency (tending to Bayes Optimal error as the data tends to infinity). The main reason is that the exemplars are not random points in the input space, they are sampled from the data distribution of each class. Consequently, the trees *are* still learning an abstraction of the data, using the trees as a density estimator [164].

Figure 3.1 depicts a graphical representation of a simple split obtained on the `Trace` dataset. It is interesting to see that in Euclidean space, the splitting criterion is actually forming a hyperplane

that is equidistant to the exemplars. Note that this intuition is more complex for time series measures because most of them do not have properties of a metric [100].

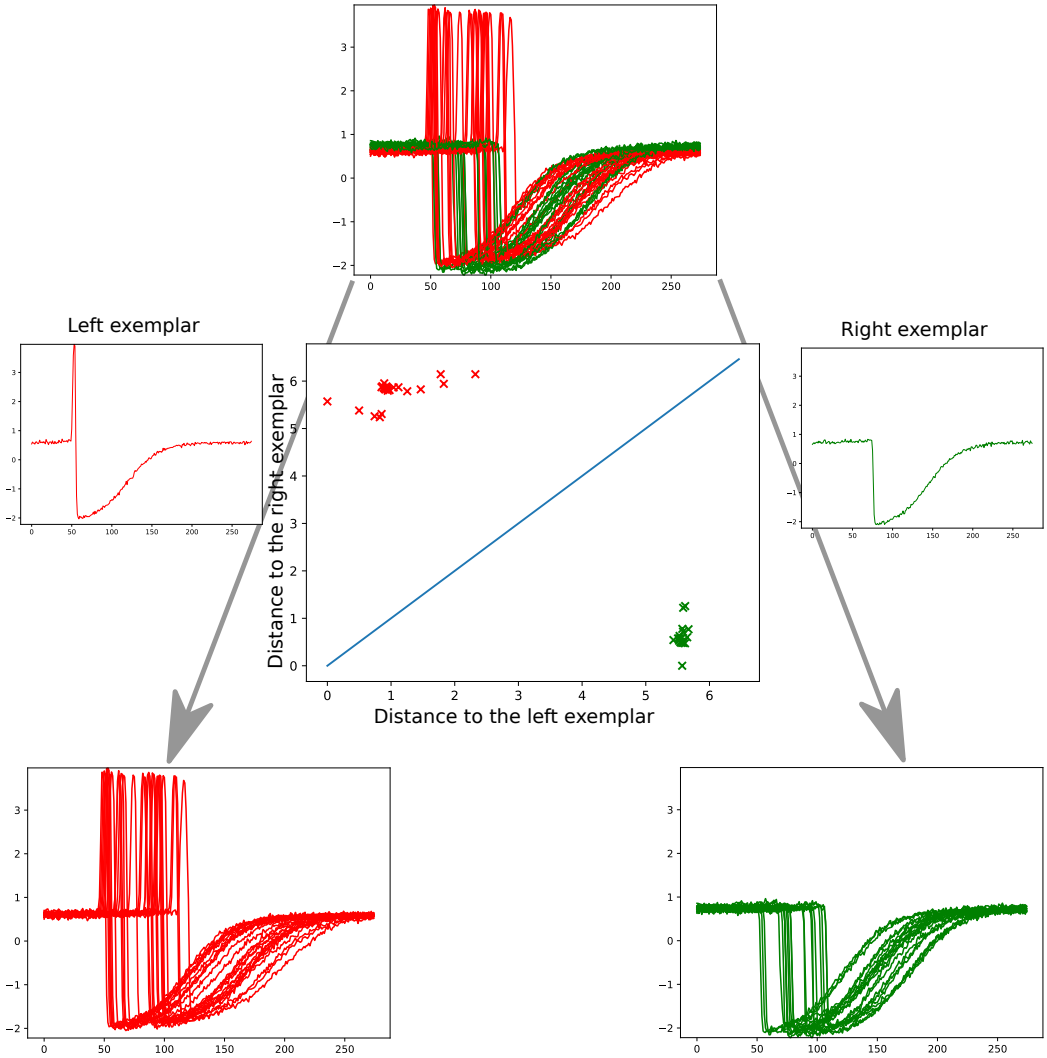


Figure 3.1: Visual depiction of the root node for the *Trace* dataset (simplified to 2 classes). The top chart represents the data at the root node (one colour per class) while the data at the bottom left and right represent the data once split by the tree. The two time series in the middle left and right are the exemplars on which the tree is splitting. The scatter plot at the center represents the distance of each time series at the root node with respect to the left and right exemplars (resp. x- and y-axes).

The scatter plot depicts each time series as a dot in this space, with the x-axis representing distance to the first exemplar and the y-axis distance to the second.

Choosing the parametrised measure to split on

The parametrised distance measure gives a measure of the similarity between the exemplar time series and the data. For each candidate split at a node, the algorithm chooses a distance measure at random from the following 11 distance measures used in EE: Euclidean Distance

(ED); Dynamic Time Warping using the full window (DTW); Dynamic Time Warping with a restricted warping window (DTW-R); Weighted Dynamic Time Warping (WDTW); Derivative Dynamic Time Warping using the full window (DDTW); Derivative Dynamic Time Warping with a restricted warping window (DDTW-R); Weighted Derivative Dynamic Time Warping (WDDTW); Longest Common Subsequence (LCSS); Edit Distance with Real Penalty (ERP); Time Warp Edit Distance (TWE); and, Move-Split-Merge (MSM). Randomising the choice of distance measure is a deliberate decision to introduce variability between each tree, for the reasons stated earlier.

Once a distance measure is chosen at random, it is then parametrised. The parametrisations are deliberately chosen to mimic as closely as possible the EE algorithm, and are addressed in turn. Even though alternative choice would likely improve either the speed of accuracy, we mimic EE's parameterisation to allow direct comparison of the different architectures, trees with proximity splitters and nearest neighbour classifiers. Euclidean distance, full DTW, and full DDTW distances have no parameters to select. DTW-R and DDTW-R only require a warping window parameter that is chosen uniformly at random in $[\![0, \lfloor \frac{l+1}{4} \rfloor]\!]$ (thus allowing a warping of elements at most $\frac{l}{2}$ apart). WDTW and WDDTW also require one parameter, the weighted value g, which controls the level of penalisation between two different timestamps—we use $g \sim U(0, 1)$. The parametrisation of ERP is a distance threshold that controls for how close elements have to be to be considered similar; we sample it uniformly at random in $[\frac{\sigma}{5}, \sigma]$, with σ being the standard deviation of the data. LCSS has as first parameter the same distance threshold value (which is sampled in the same way), and has a second parameter–the warping window size–which is chosen in the same way as for DTW-R. TWE has two parameters γ and λ, which control for the stiffness and penalty value in the alignment, respectively. Following [112], λ is sampled at random from $\cup_{i=0}^{9} \frac{i}{9}$ and γ sampled at random from an exponentially growing sequence $\{10^{-5}, 10^{-4}, 5 \text{ x } 10^{-4}, 10^{-3}, 5 \text{ x } 10^{-3}, \cdots, 1\}$. This results in 100 possible parameterisations for the TWE measure. The final measure, MSM, has a single parameter which is sampled from an exponential sequence similar to the one for γ in TWE with 100 values ranging from 10^{-2} to 10^{2}, as recommended in [153].

Choosing the parameter at random has a twofold effect: 1) it skips the cross-validation step which has a complexity that is quadratic with the number of training instances; and 2) it introduces variability between trees, which provides superior learning through lower-biased trees and ensembling. In the experiments section (Section 3.3), I will demonstrate that this method is extremely successful as Proximity Forest is not only orders of magnitude faster than EE, but its accuracy also ranks higher than EE.

3.1.2 Classifying with a Proximity Forest

The process of classification for a single Proximity Tree is detailed in Algorithm 3. A query time series begins at the root node and the distance from the query to each of the exemplar time series is calculated, by using the node's distance measure and exemplars selected when constructing the tree. The query time series is then passed down the branch of the exemplar to which it is nearest. The query time series then traverses down the tree by repeating this process until it reaches a leaf, where it is assigned the class represented by that leaf. This process is repeated for each tree constructed as part of the forest. A Proximity Forest then uses majority voting between its constituent Proximity Trees.

Algorithm 3: classification(Q, T)

Input: Q: Query Time Series
Input: T: Proximity Tree

if is_leaf(T) **then**
  return majority class of T

`// find the branch with exemplar closest to` Q `using measure` T_δ
$(e, T^\star) \leftarrow \underset{(e',T')\in T_B}{\arg\min}\ T_\delta(Q, e')$

return classification$(Q, T^\star)$ `// recursive call on subtree` $T^\star$

3.2 Comparative Complexity Analysis

During the training phase, at each node, let us assume that n' data points are present at the node. We first scan it once taking $O(n')$ time to split the data into c groups, one for each class c present at the node.

We then generate $\mathcal{C}$ candidate splits, *i.e.*, $\mathcal{C}$ sets of exemplar time series. For each such candidate set, we sample $c' \leqslant c$ exemplar time series, ie. one time series for each class represented among the n' time series available at the node–this is done in $O(1)$ given that the data is already organised by class. For the candidate split to be operational, we also require a parameterised measure to use to compare against these c' exemplars. Most of the parametrised measures can be chosen in $O(1)$, except for LCSS and ERP which calculate the standard deviation in $O(n' \cdot \ell)$ for data at the node while selecting the parameter. [2]

We now have $\mathcal{C}$ candidate splits that are ready to be evaluated. Next we pass the n' time series down the branches for all candidate splits. This is done by comparing each time series to the c' exemplars, each comparison taking from $O(c \cdot \ell)$ to $O(c \cdot \ell^2)$. Overall, this takes $O(n' \cdot c' \cdot \ell^2)$. If $r = 1$, the training process at this node is finished and we call the training function recursively for

[2] Note that these parametrisations can be performed in constant time also if the data are z-normalized, which is the case for all UCR datasets.

each of the c' children nodes. If $r > 1$, we calculate the Gini coefficient for each of the $\mathcal{C}$ candidate splits in $O(c^2)$, keep the best one, and delete other candidate splits.

As the total number of examples that reach any of the nodes at a single given level cannot be greater than the total number of examples, n, the total computation per level of the tree is thus $O(n \cdot r \cdot c \cdot \ell^2)$. In the worst case, the majority of the training data at each level will pass down a single branch and the depth of the tree will be $O(n)$, resulting in a worst training time complexity of $O(n^2 \cdot r \cdot c \cdot \ell^2)$. However, as the exemplars are following the class distribution, unless the data are in some way degenerate (for example if one class comprises only outliers), the average tree depth can be expected to be $O(\log n)$. In practice it will often be much smaller because, unlike typical divide-and-conquer approaches, the tree terminates as soon as a node is pure rather than having to separate each individual object. Thus, for non-degenerate data we can expect average case training time complexity of $O\left(n \log(n) \cdot r \cdot c \cdot \ell^2\right)$ for a single tree and thus $O\left(k \cdot n \log(n) \cdot r \cdot c \cdot \ell^2\right)$ for a full Proximity Forest comprising k Proximity Trees.

The experiments presented in the next section will include runtimes and comparison to current state-of-the-art algorithms. These confirm that this expected average case quasi-linear complexity with respect to data quantity is borne out in practice.

During classification, a time series of length ℓ will pass through an average of $\log n$ nodes on each of the k trees. At each node, the distance to at most c exemplars must be computed. For each of these distance computations, the complexity will again depend upon the chosen distance measure; the fastest being $O(\ell)$ and the slowest $O(\ell^2)$. Thus, the resulting average case complexity is $O(k \cdot \log n \cdot c \cdot \ell^2)$.

3.3 Experiments

This section describes experiments performed to evaluate Proximity Forest against the state of the art in time series classification in terms of accuracy and scalability. It starts with classification of the SITS dataset, the motivating example and an example of the large time series datasets that need a new scalable time series classification algorithm. The results show that no current state-of-the-art approach scales to this magnitude, and the classifiers that are designed for scalability, namely BOSS-VS, compromises classification accuracy to do so. The experiments presented in this section demonstrate the ability of Proximity Forest to be both scalable *and* accurate.

The second section assesses Proximity Forest on the datasets of the UCR time series classification repository, the benchmark in the field. It demonstrates that the classification accuracy of Proximity Forest is competitive with the current state of the art. The final section discusses other considerations surrounding Proximity Forest, such as hyperparameter choices (eg.

the effect of varying the number of trees), and the standard deviation of the results.

It should be mentioned that the following experiments emphasise a comparison with EE. This is because it is viewed as the closest relative to Proximity Forest amongst the current state of the art, given that neither method includes data transforms or shapelets. It is also a constituent of COTE that bounds its learning time and therefore any improvement over the runtime of EE, for the same classification accuracy, would also equate to an improvement on COTE, the current leader in the field.

3.3.1 Scalability Experiments Using Satellite Image Time Series

The SITS dataset[3] used for these experiments contains approximately 1 million time series with a train-test split of approximately 90%-10% [4]. Each time series has a length of 46 and is labelled as one of 24 possible land-use classes (*e.g.*, 'wheat', 'corn', 'plantation', 'urban'). Here the labelled data has been extracted from three sources: 1) ground field campaigns for most of the vegetation classes, 2) farmers' declaration to complete the data for some crop classes, and 3) existing map for the urban areas[5].

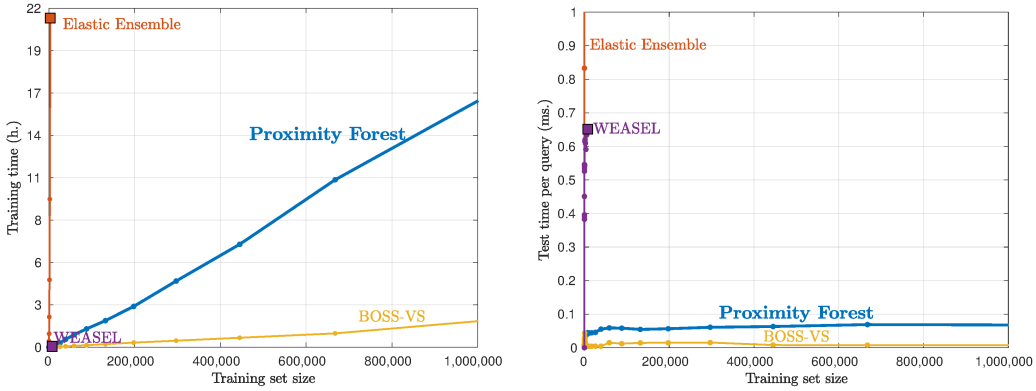
The following experiments were performed on this dataset, comparing the performance of Proximity Forest against three competitors: BOSS-VS (designed for scalability), WEASEL (designed for speed and quality), and EE (designed for quality). It uses 5 runs for each experiment of Proximity Forest and 1 run of each of the competitors–as their results are deterministic not stochastic. Throughout these experiments, Proximity Forest is set at 100 trees and 5 candidate splits per node; we will see in Section 3.3.3 that this gives a good trade-off between accuracy and computational time/memory. Although this section is primarily assessing scalability, it will also touch on the accuracy of the classifiers.

Training scalability

To assess scalability, each algorithm was trained on a subsample of the data with increasing training set size, and assessed on a constant test set. This allows training time, testing time and classification accuracy to be measured as a function of training size. Figure 3.2(a) shows training time against training size for each of the 4 algorithms.

Versus EE: First, it is evident that Proximity Forest presents a notable saving in training time over EE, confirming that it trains in linear time rather than the quadratic time for EE. Even for a small training set of about 2 000 time series, learning an EE model took about 10 hours, compared to Proximity Forest's 79 seconds. Fitting a quadratic curve through both EE and Proximity

[3]The data are presented at depth in Section 6.1
[4]The split ensures that no 2 times series come from the same plot of land.
[5]These are the same data including train/test split used in [126].



(a) (b)

Figure 3.2: Training time (a) and testing time per query (b) as a function of training set size for Proximity Forest, EE, WEASEL and BOSS-VS.

Forest is quite informative: EE returns a quadratic component of 6.3 while Proximity Forest is only $8\text{x}10^{-6}$, clearly highlighting both the quadratic complexity of EE, and also that Proximity Forest is in practice very close to its theoretical average complexity and scales quasi-linearly with n.

Versus WEASEL: WEASEL is very fast but its memory footprint did not allow it to scale beyond 8 000 time series even when given 64GB of RAM. This clearly highlights the difference with BOSS-VS: we can see that WEASEL was not developed for scalability, but rather for speed on small datasets.

Versus BOSS-VS: Proximity Forest trains slower than BOSS-VS for a given training size, however this is counteracted by the low accuracy of BOSS-VS, which is discussed further below.

Testing scalability

Figure 3.2(b) shows testing time against training size for each of the 4 algorithms.

The story here is very similar to that of training: it confirms the way Proximity Forest scales logarithmically with training set size, while EE must scan the full database many times. Here again, WEASEL becomes infeasible to apply with relatively small quantities of training data. Proximity Forest and BOSS-VS require respectively 0.0679 ms and 0.0077 ms to classify a time series with a model trained on 1M time series.

Accuracy on the SITS dataset

The SITS experiments show that Proximity Forest is highly scalable and only beaten by BOSS-VS in terms of training time. It is also important to consider how its accuracy scales with training set size. The main results are presented in Figure 3.3 which plots the accuracy as a function of training set size for Proximity Forest, EE, WEASEL and BOSS-VS.

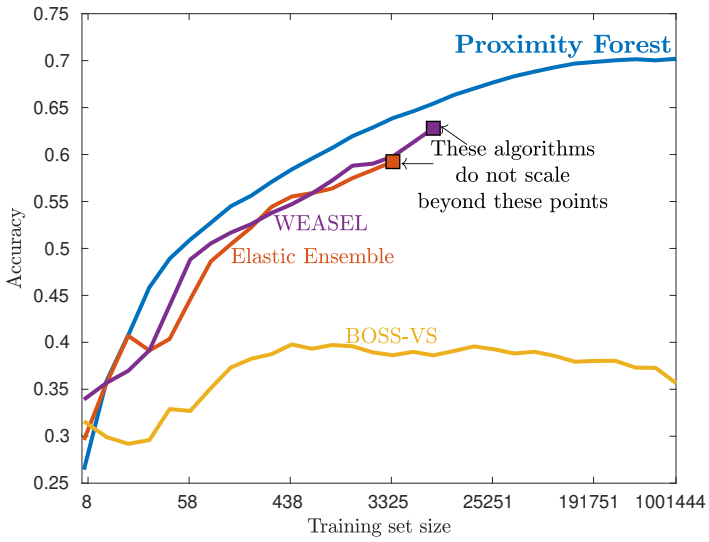


Figure 3.3: Accuracy as a function of training set size for Proximity Forest, EE, WEASEL and BOSS-VS.

The first element to observe is that Proximity Forest obtains greater accuracy than the competitors for large training sets. WEASEL and EE become infeasible to apply at relatively small data quantities and BOSS-VS–which is faster than Proximity Forest–does not learn effective classifiers on this dataset. With 63.8% accuracy at 3 400 training set size, this is 26.3 percentage points more accurate than BOSS-VS, and 4.6 and 4.7 percentage points more accurate than WEASEL and EE, respectively. Such differences are substantial in a problem comprising 24 classes. Moreover, Proximity Forest is more accurate than the other algorithms from 500 training instances upwards. This is not surprising, as trees usually have a better control over variance than NN algorithms, because of their higher bias and abstraction capabilities. Proximity Forest thus appears to be both accurate and highly scalable.

The next subsection will test whether these accuracy result holds also on the benchmark UCR archive.

3.3.2 Accuracy Experiments Using the UCR Archive

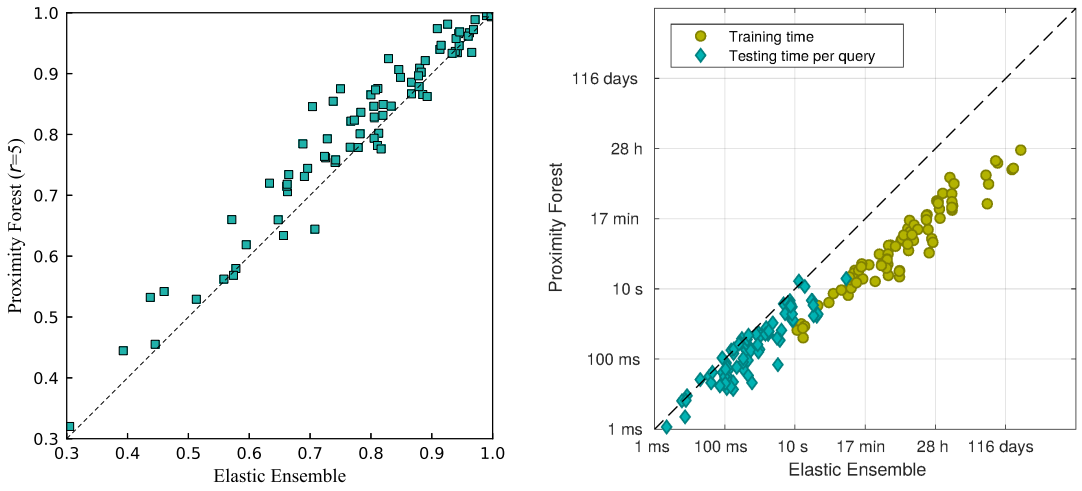
In this section, Proximity Forest is evaluated on the 85 datasets of the traditional UCR archive [28]. It is useful to remember here that the aim is not to show that Proximity Forest is more accurate than the state of the art, but only that it is competitive while being highly scalable.

We compare the mean error-rate of Proximity Forest to the error-rates on the standard train/test split for the state of the art, as tested in [7]. We average Proximity Forest results over 10 runs for each experiment. We compare Proximity Forest to five classifiers currently

representing the state of the art—DTW, EE, ST, BOSS[6], and COTE[7].

The Proximity Forest results are obtained for 100 trees with selection between 5 candidate splits per node. A detailed discussion about the Proximity Forest parameters will be presented in Section 3.3.3.

We first show the comparison with Proximity Forest's closest relative, EE. Figure 3.4 provides scatter plots of the relative accuracy, total training time and total testing time (logarithmic scale) of each of these classifiers, with each point representing a different UCR dataset.



(a) (b)

Figure 3.4: Comparison of Proximity Forest and Elastic Ensemble classifiers on UCR datasets in terms of a) accuracy and b) training and testing times (in log scale).

Figure 3.4(a) shows that Proximity Forest is more accurate on 60 datasets and less accurate on only 11 datasets, with 14 ties. Moreover, it is evident that for many datasets Proximity Forest is substantially more accurate than EE.

Figure 3.4(b) demonstrates that Proximity Forest has several orders of magnitude advantage in training time. When considering testing time, Proximity Forest has greater test time per query than EE for 12 datasets, the majority of which are small datasets (ie. less than 50 training instances). The largest such difference is observed for the 'Phonemes' dataset for which Proximity Forest takes about 17 seconds per query compared to 13 seconds per query for EE. In contrast, the test time for Proximity Forest is much smaller than EE for the biggest datasets (ie. more than 800 training instances). For example, the biggest test time difference is for the 'HandOutlines' dataset for which Proximity Forest takes about 19 seconds per query compared to 286 seconds per query for EE.

The commonly accepted method to compare multiple classifiers over multiple datasets is by

[6]This section uses the original BOSS algorithm, and not the BOSS-VS discussed in the SITS experiments. BOSS-VS is a scalable variation of BOSS, where concessions are made to accuracy in favor of training time. The original BOSS is therefore more competitive in this section.

[7]These algorithms represented the state of the art at the time Proximity Forest was developed, see [7].

comparing their average rankings using a critical difference [41] (as discussed in Section 2.9). When comparing these 6 algorithms over the 85 datasets of the UCR archive, for the rankings to be significantly different at level $\alpha = 0.05$, the critical difference (CD) between the average ranks has to be greater than:

$$CD = q_{0.05}(A) \cdot \sqrt{\frac{A(A+1)}{6 \cdot N_d}} = 2.850 \cdot \sqrt{\frac{42}{510}} \approx 0.82 \tag{3.1}$$

The average ranks and critical difference for the experiments on the UCR datasets are presented in Figure 3.5; the critical difference of 0.82 is displayed by the black line.

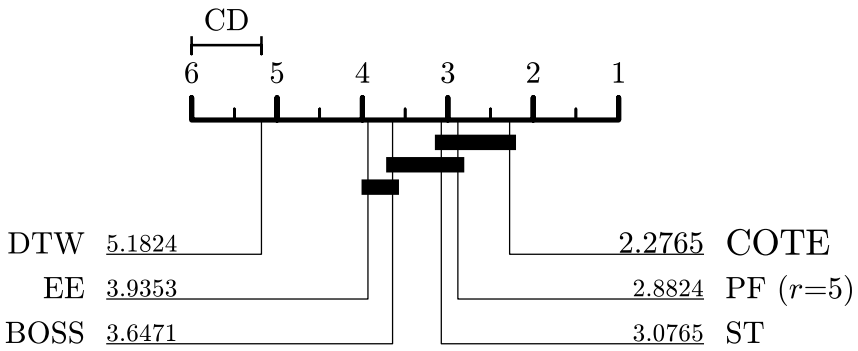


Figure 3.5: Critical difference diagram for five state-of-the-art classifiers and Proximity Forest (PF) with 5 candidates.

It can be seen that COTE ranks highest (average rank of 2.28), which is to be expected considering it incorporates the other state-of-the-art algorithms. However, COTE is not ranked significantly higher than Proximity Forest (average rank of 2.88) or ST (average rank of 3.08). Proximity Forest is ranked second overall. Its rank is not significantly different to COTE, ST or BOSS, but it is ranked significantly higher than both EE and DTW. This affirms Proximity Forest as a classifier with accuracy competitive with the then state of the art.

Proximity Forest is the most accurate classifier for 22 of the 85 datasets. However, there is no obvious commonality between these datasets to indicate conditions under which the algorithm is likely to excel.

3.3.3 Parameters and Model Variations

Proximity Forest has two main parameters that merit further investigation. First, experiments are performed exploring the sensitivity of accuracy to the number of trees in each ensemble. Then, the influence of the number of candidates assessed at each node is investigated. While not a hyperparameter, a third design choice is also explored: the random selection of similarity measure per tree as opposed to per node.

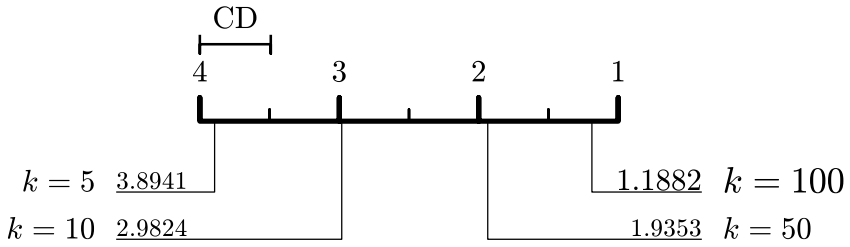


Figure 3.6: Critical difference diagram for Proximity Forest with 5, 10, 50 or 100 trees.

Number of Trees

The number of trees is the first parameter of choice when implementing the Proximity Forest algorithm, with the optimal value being large enough to provide competitive accuracy, yet small enough not to create excessive computational expense.

The UCR datasets experiments outlined above were repeated with values of 5, 10, 50, and 100 trees to analyse how the number of trees affects performance. Here, the number of candidates $\mathcal{C}$ has been fixed to 1 and the results are averaged over 50 runs. Figure 3.6 presents the critical difference diagram for accuracy with Proximity Forest models using of a different number of trees.

As would be expected, the more trees the higher the average accuracy: models with 100 trees had an average rank of 1.19 compared to 1.93, 2.98 and 3.89 for models with 50, 10, and 5 trees respectively. The difference between the highest ranked models are large enough to say that models with 100 trees are significantly better than models with 50 trees at the level of alpha equals 0.05.

Figure 3.7 compares the classification accuracy for 100 trees against 10 and 50 trees by representing them as a ratio of their error rates.

Each point represents a single dataset. This shows that having 100 trees is better on most datasets. Moreover, the fact that the data is gathered close to the line with equation $x = 1$ shows that it is unlikely that more trees would provide a very significant improvement, because the ratio of error-rates between 100 and 50 is already close to 1 (ie. the errors are only slightly reduced).

Experiments were not performed with forests comprising more than 100 trees as it was felt that the computational demands outweighed the expected benefits for our large set of experiments. Memory, training time and testing time all scale linearly with the number of trees, which means that doubling the number of trees doubles the required memory and time. However, where computational resources are not an issue, the take home message is that the more trees, the better.

As a randomized algorithm, it is interesting to study the standard deviation of the errors for Proximity Forest and how it varies with the number of trees. This is presented in Figure 3.8, where the y-axis represents the standard deviation on error-rate for 100 trees as a function of the standard deviation on k equals to 5, 10, and 50 trees. Each point represents a single dataset. One can see

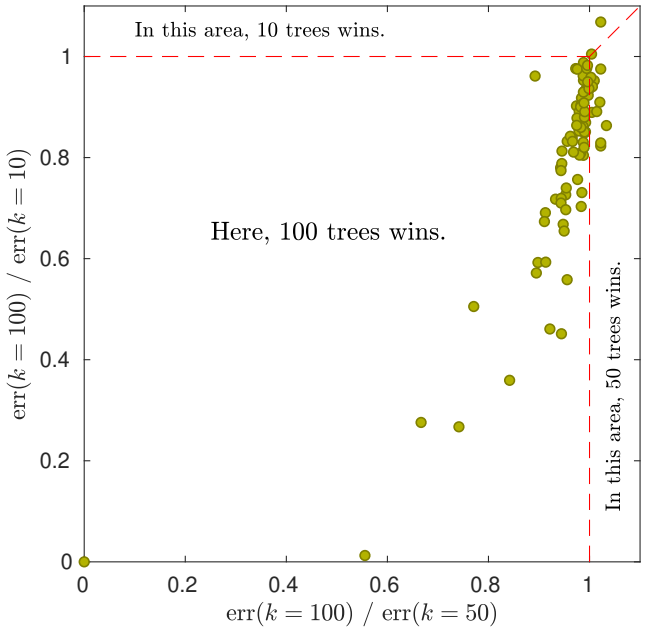


Figure 3.7: Ratio of the error rates of Proximity Forest models: 100 trees over 10 trees (x-axis) against 100 trees over 50 trees (y-axis). A value of less than 1 on either axis indicates that the model with 100 trees has higher accuracy.

that the standard deviation reduces as we increase the number of trees, and that the magnitude of this improvement reduces when increasing k. Results for 50 trees are starting to be relatively close to the $y = x$ line, showing that only marginal improvements could be expected when going to $k > 100$.

Split selection using the Gini index

This section explores the influence of the number of candidates $\mathcal{C}$ that are randomly selected at each node. As a reminder, a set of $\mathcal{C}$ candidates–exemplars and parametrised distance measures–are evaluated at each node based on the Gini index. The one maximising the Gini index is retained. To evaluate the influence of $\mathcal{C}$, the UCR experiments were repeated for 1, 2, and 5 candidates on 100 trees and the results are averaged over 10 runs.

Figure 3.9 compares the classification accuracy for 5 candidates against 1 and 2 candidates. Each point represents the ratio of the error for 5 candidates to that for the alternative on an UCR dataset.

Choosing between 5 candidates results in higher accuracy for most datasets. More precisely, selecting between 5 candidates results in greater accuracy than either 1 or 2 candidates on 61 datasets. Increasing the number of candidates lead to a reduction of the randomness on each node by discarding the worse splitters. Accordingly, the overall Proximity Forest accuracies are improved.

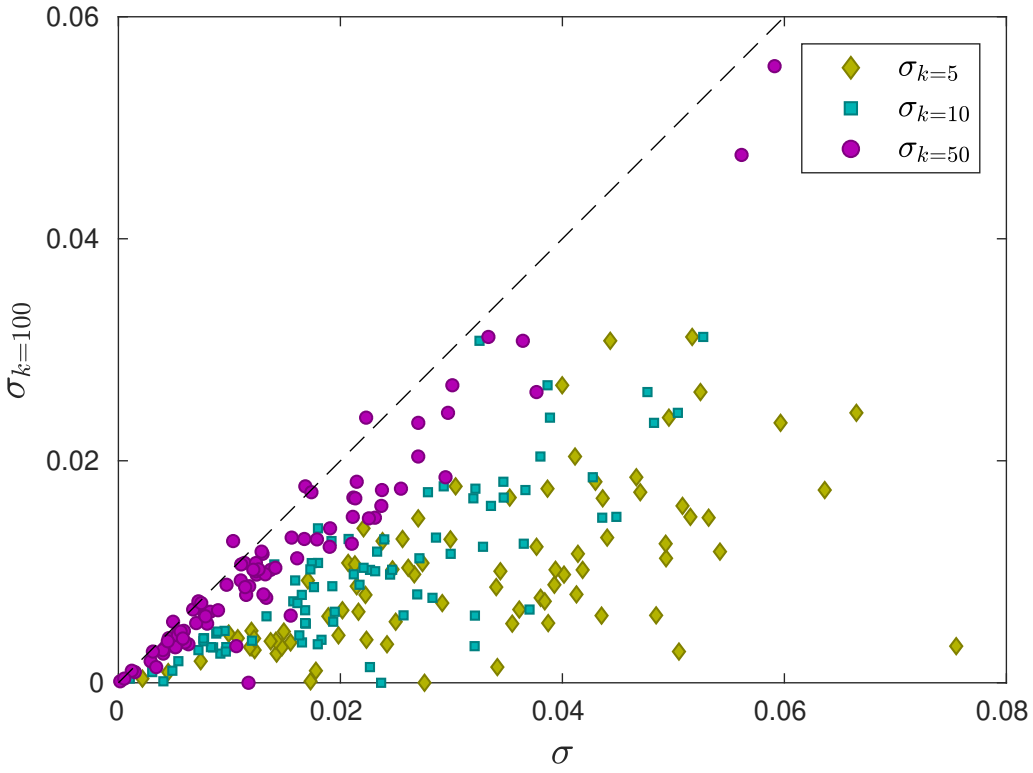


Figure 3.8: Standard deviations σ of error rates on the 85 datasets of the UCR archive for Proximity Forest models: 100 trees against 50, 10 and 5 trees.

Increasing the number of candidates to more than 5 may further improve the classification accuracy. However, increasing the number of candidates per node has substantial impact on training time. Indeed, the analysis of the Proximity Forest's computational complexity in Section 3.2 shows that the training time scales linearly with the number of candidates. To verify this analysis, a comparison of both training and testing time of Proximity Forest for 1 and 5 candidates is shown in Figure 3.10. The testing time is displayed per query, and each point represents a dataset of the UCR repository.

The results show a mean increase of 4.6 times in training time between 1 and 5 candidates, and a mean decrease of 0.93 times in testing time.

It is notable that selection between multiple alternatives both reduces testing time and increases the training time by slightly less than the expected multiple of 5 times. This is because it results in slightly shallower trees. Selection of better splits better separates the classes, requiring fewer splits to obtain pure nodes that are made into leaves.

The tuning of the number of candidates is therefore driven by a trade-off between accuracy and time.

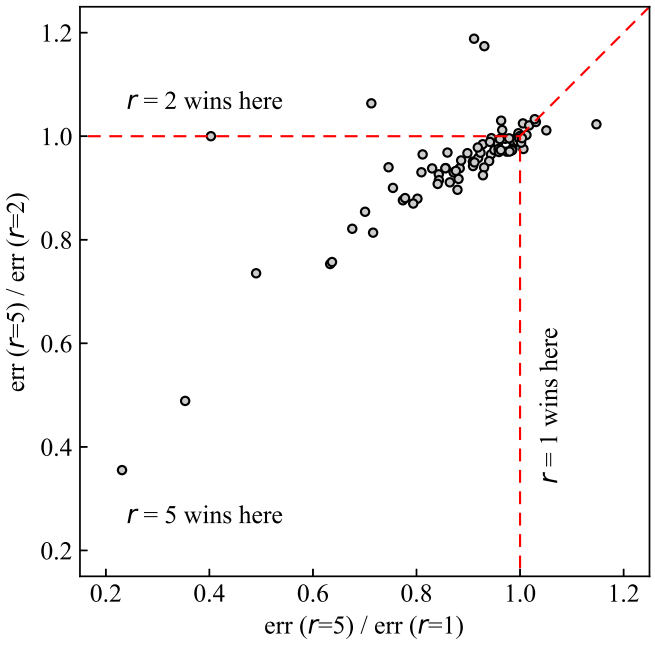


Figure 3.9: Ratio of the error rates of Proximity Forest models: 5 candidates over 1 candidate (x-axis) against 5 candidates over 2 candidates (y-axis). A value of less than 1 on either axis suggests that the model with 5 candidates has superior accuracy

Distance measure chosen per-tree versus per-node

A final variant of the Proximity Forest algorithm to explore is randomly selecting a distance measure for each tree, rather than for each node. This idea is motivated by the design of TSF [42], where one subseries is selected per tree, rather than per node. An 'on tree' variant, would be marginally quicker computationally, as only the exemplars and the parameters of the distance-metric are randomly chosen at each node. The UCR experiments were repeated for a Proximity Forest with 100 trees and 1 candidate split for this new 'on tree' variant against the standard 'on node' configuration. Each result presented is averaged over 50 runs.

Figure 3.11 compares classification accuracy for the original version 'on node' and the proposed variant 'on tree'. Each point represents a single dataset of the UCR dataset.

The results show a slight advantage for the 'on node' approach with 44 wins, 39 losses and 2 ties. Where the 'on tree' variant uses a single distance measure per tree, the 'on node' variant allows multiple combinations of measures in a single tree, thus making it more robust to inefficient metrics.

3.4 Conclusion

Research into the classification of time series has made enormous progress in the last decade. The UCR time series archive has played a significant role in challenging and guiding the development

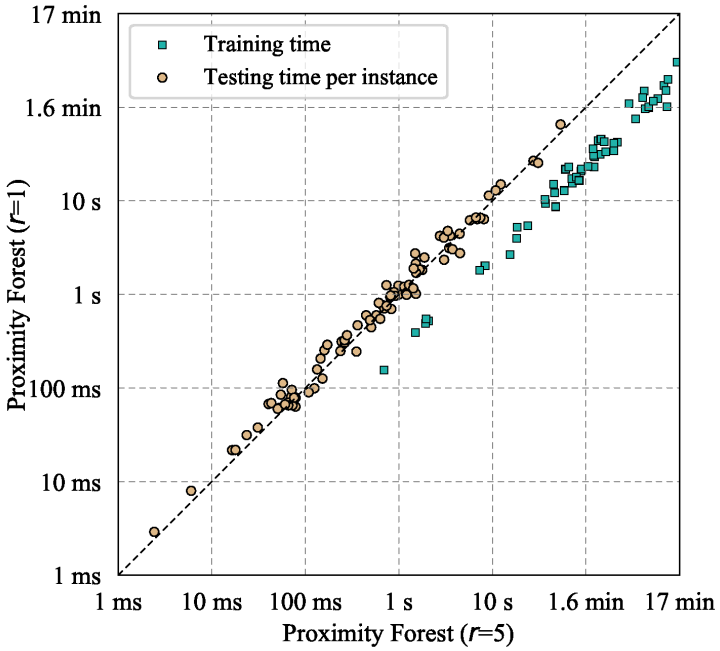


Figure 3.10: Training and testing time of Proximity Forest for 1 and 5 candidates on UCR datasets.

of new learners for time series classification. However, the largest dataset in the archive held 10 *thousand* time series only, which likely explains why the primary research focus has been on creating algorithms that have high accuracy on relatively small datasets. As a result, all of the algorithms considered state of the art in time series classification have a high computational complexity and are incapable of scaling to even medium sized datasets. This poses an issue as many modern time series datasets contain millions of instances. The motivating example for my research was the SITS dataset for land cover mapping.

As time series classifiers generally cannot scale, land cover maps are often produced from SITS using conventional machine learning techniques, treating the data as an array of independent points and forgoing any information contained in their temporal structure. My research created Proximity Forest, a model based on an ensemble of decision trees with a novel time series-specific splitting criterion at the node. On a SITS dataset of size 1 million and length 46, Proximity Forest could learn a model in 17 hours, compared to EE–then considered state of the art–requiring more than 200 years of CPU time. While the work was motivated by recent time series applications that provide orders of magnitude more time series than the UCR benchmarks, the experiments also demonstrated that Proximity Forest was highly competitive on the UCR archive. That is, Proximity Forest was one of the most accurate classifiers while being significantly faster.

While Proximity Forest has since been surpassed in terms of speed and accuracy it is notable that it changed the direction of the field to pay greater attention to the computational efficiency of algorithms.

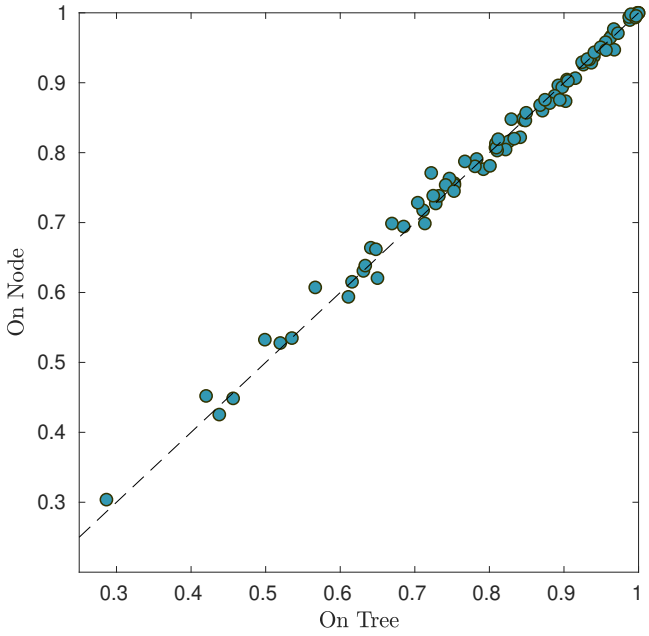


Figure 3.11: Accuracy of Proximity Forest when randomly selecting the distance measure 'on node' and 'on tree'.

3.4.1 Contributions

The contributions resulting from this chapter of my Ph.D research and the development of Proximity Forest are as follows:

1. The first time series classifier capable of learning from millions of time series in a reasonable time frame;
2. A highly scalable classifier that is also highly accurate (competitive with the accuracy of the state of the art on small datasets);
3. Helping to identify the need for scalable time series classifiers and inspiring subsequent research in the area.

4

Deep Learning for Time Series Classification

Deep Learning, also referred to as Neural Networks or Deep Neural Networks, is one of the most active areas in machine learning research. This class of models has experienced great success in a wide variety of applications including speech recognition [76], computer vision [93], bioinformatics [160, 23], and recommendation systems [194]. This success derives from the ability of deep learning models to learn hierarchical features, unlike other machine learning models, which must learn from the features that are hand-engineered[1].

While deep learning models have revolutionised many fields, they have only recently proven to be highly competitive in time series classification problems. This fact is interesting as deep learning has been extremely successful in image recognition and time series have essentially the same topology as image data, just with one fewer dimension [49, 14].

[1] I note that the counter to this argument is that the architecture of deep learning models is not always straightforward to design

It if for this reason that I decided to conduct research into using deep learning for time series classification. My research led to collaborating with colleagues to adapt the Inception deep learning model [157] to time series data, with state-of-the-art results. This research also leads to defining the concept of receptive fields for CNN for time series, an idea that assists in understanding how deep learning for time series classification works.

In this chapter I will provide an overview of deep learning, its use in computer vision problems, and its limited use in time series classification[2]. I will also include a summary of InceptionTime, a class-leading ensemble of deep learning models for time series classification, and highlight my contribution to this publication. Finally, this chapter will discuss the use of deep learning in the context of land cover mapping.

The purpose of this chapter is to provide both the requisite background of deep learning and to highlight my contributions to research on deep learning for time series and why I chose to use deep learning-based models for my future research in Part II.

4.1 Deep Learning

It is believed that deep learning models work by approximating the function of the human brain (and are therefore also called neural networks), by learning higher levels of abstraction first, and proceeding gradually to finer and finer levels of detail [14]. They replicate this function by using a sequence of connected 'layers'. For example, if the input is an image, the first layer of the model may learn edges, the second may learn the respective location of the edges, the third might learn specific features of the image, while the final layer may identify the content of the image.

There are 4 major types of deep learning models relevant for these purposes: Feed-Forward Neural Networks (FFNN), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) [145][3]. FFNN are defined by an internal structure where each node takes input from each node in the layer prior and is connected to each node in the layer subsequent. Each node has an associated weight by which the data is multiplied, resulting in a series of linear transforms of the input data. A CNN [98] is defined by layers in which a convolution function is applied to the data, allowing the model to recognise a relationship between adjoining neurons. RNN are largely used on sequential data as the output of a given element is dependent not only on input data but also the output at the previous step, which is fed back into the model as an input. This means that they are efficient in learning repeating patterns such as those that occur in sentences and are therefore highly effective in natural language processing applications.

[2]Please note that the quantity of deep learning research for time series classification has increased significantly since my research into this area began (see [49])

[3]A forth type of deep learning model—Transformers [174]—have been extremely successful in natural language processing applications but are not relevant for our purposes

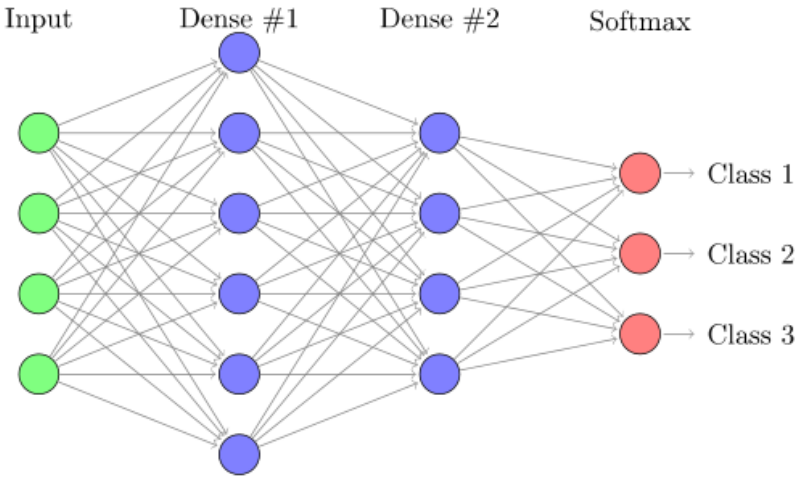


Figure 4.1: An example of a basic feed-forward neural network classifier with two dense layers and a softmax output [126].

One characteristic that all deep learning models have in common is that the size of the model must be defined by the user prior to training and finding the optimal size for a given problem can often consume valuable time and resources.

4.1.1 Feed-forward Neural Networks

A FFNN is the simplest deep learning architecture where models are composed of multiple layers of fully-connected neurons. A graphical interpretation of a FFNN with two 'dense' layers and a softmax output is shown in Figure 4.1. The neuron in a dense layer l are connected to all of the neurons in both the previous and subsequent layers. Each neuron has weight and bias parameters, the values of which are learned from the data via backpropagation [136]. The formula for the calculation of the output of layer $A^{[l]}$ is shown in Equation 4.1. The product of the input values $A^{[l-1]}$ (*i.e.*, the output values of the layer prior) and the layer weights $W^{[l]}$ are added to the layer biases $b^{[l]}$. Finally, a non-linear activation function $g^{[l]}$ is applied to the output to enable the model to fit non-linear relationships. The most common activation function for dense layers is the Rectified Linear Unit (ReLU) [99], which is computed as $\text{ReLU}(z)=\max(0, z)$.

$$A^{[l]} = g^{[l]}(W^{[l]}A^{[l-1]} + b^{[l]}) \tag{4.1}$$

The final layer differs from the dense layers by the activation function. For a classification problem with k classes the activation function would be a softmax function with k neurons. The output value of a neuron after applying the softmax function represents the probability that the instance is a member of that class. Formally, the equation for predicting class y given data $\mathbf{x}$ is:

$$p(y|\mathbf{x}) = \frac{e^{(a_y)}}{\sum_{i=1}^{K} e^{(a_i)}} \tag{4.2}$$

where $a_i = W_i^{[l]} A^{[l-1]} + b_i^{[l]}$), *i.e.*, the output value of neuron i or the i-th element of output vector A.

The weight $\mathbf{W} = \{W^{[l]}\} \; \forall \; l \in \{1, L\}$ and bias $\mathbf{b} = \{b^{[l]}\} \; \forall \; l \in \{1, L\}$ parameters of the model are generally trained using stochastic gradient descent or similar algorithms such as Adam [89]. These algorithms minimise a cost function $\mathcal{J}$, which is the average classification loss on the training data (of quantity n):

$$\mathcal{J}(\mathbf{W}, \mathbf{b}) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(\hat{y}_i, y_i), \tag{4.3}$$

where $\hat{y}_i$ is the class of instance i as predicted by the model and y_i is the correct class label of that instance.

The most common loss function $\mathcal{L}(\hat{y}_i, y_i)$ for a multi-class classification problem is the cross-entropy loss:

$$\mathcal{L}(\hat{y}_i, y_i) = -log(p(\hat{y}_i = y_i|\mathbf{x}_i)) \tag{4.4}$$

that is, the negative logarithm of the predicted probability of correctly classifying instance i.

In the case where the quantity of training data is very large, it would be quite inefficient to classify all of the data and the subsequent training loss for each step of the optimisation process (as this may be repeated thousands of times). For this reason the model is trained using mini-batches, where the total data are randomly divided into subsamples (often of 32 or 64 instances) and a parameter update is performed after the model is trained on each subsample.

Training via stochastic gradient descent (or similar) makes deep learning models highly scalable as the training complexity of these algorithms is effectively linear with the number of training instances [65]. In some instances, training times are extended due to an issue known as the 'vanishing gradient' problem [15]. Vanishing gradients occur when the change to the loss function with respect to some parameters is so small that they cannot update, and hence cannot be optimised via backpropagation. The chance of this occurring is minimised through the use of the ReLU activation function and through residual connects in the network [73] (discussed further in Section 4.2).

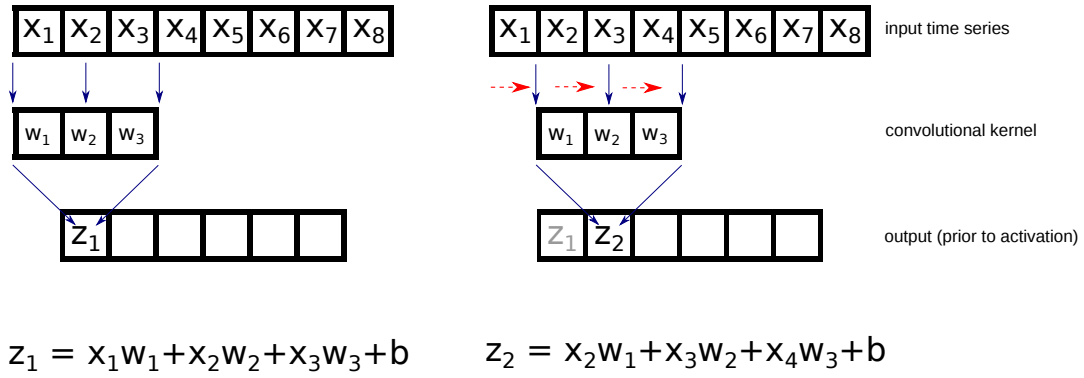


Figure 4.2: An example of a 1-dimensional convolutional kernel of length 3 applied to an input time series. A dot product is calculated between the kernel and the corresponding section of the input time series, the kernel is then shifted along the temporal axis and the calculation is repeated. A bias term (b) is also added to each calculation.

4.1.2 Convolutional Neural Networks

A CNN [98] is a deep learning architecture where layers are comprised of a set of convolutional kernels, which detect patterns in the input. A convolutional kernel is an array of weights that is applied to a section of the input data and then 'passed' across, allowing the model to recognise relationships between adjoining points in the data. The kernels detect patterns in the input by using a sliding dot product to create feature maps that distinguish between classes [49].

Each layer is comprised of $n^{[L]}$ kernels, each with dimension: $\dim^{[L]} = (\text{length}^{[L]} \times \text{height}^{[L]} \times \text{depth}^{[L]})$. Usually, CNN are most often used for image classification problems in two dimensions and in this case the depth of the kernels of layer L is set to the number of kernels in the previous layer, *i.e.*, $\text{depth}^{[L]} = n^{[L-1]}$.

For classification of time series, the convolutional layers consist of one-dimensional kernels applied along the time axis, as opposed to two-dimensional kernels used with images (*i.e.*, $\text{height}^{[L]}=1$). An example of a one-dimensional convolutional kernel applied to an input time series is depicted in Figure 4.2. A dot product is calculated between the kernel and the corresponding section of the input time series and a bias term is added to the total. The kernel is then shifted to the next position along the temporal axis and the calculation is repeated (a stride parameter determines how far the filter is passed along the axis between calculations).

As per FFNN, the output of one layer becomes the input to the subsequent layer, and by cascading multiple layers, the network is able to further extract discriminant features that are time-invariant at various levels of abstraction [49].

A CNN kernel also has a bias parameter that is added to the output value of the convolution operation. Good values for the weights and the bias of each kernel are learned from the training data by minimising a cost function as described in Section 4.1.1.

The most successful CNN architectures have one or more fully-connected layers after the

convolutional layers.

4.1.3 A History of Convolutional Neural Networks for Computer Vision

In 2012, a CNN with over 60 millions parameters named AlexNet [93] revolutionised the field of computer vision by winning the ImageNet Large Scale Visual Recognition Challenge (ILSVRC)—an annual image recognition competition–by a margin of over ten percentage points to second place. The model consisted of 8 hidden layers: the first 5 being convolutional layers and the remaining 3 being fully connected layers.

Following the success of AlexNet, much research in this area focussed on making *deeper* CNN models (*i.e.*, models with more hidden layers) and the success of deeper models gave weight to the argument that increased model depth meant increased classification performance. The VGG-16 algorithm [152] won ILSVRC 2013 with a 16 layer CNN and the GoogLeNet algorithm [157] was victorious at ILSVRC 2014 with a 22 layer CNN.

While the additional depth was novel, GoogLeNet's most significiant new characteristic was a feature called an Inception module. It worked by allowing the model to learn the best kernel size, rather than it being predefined. An Inception module takes the input data and applies three individual convolutional kernels—sizes 1x1, 3x3, and 5x5—and an average pooling layer, all in parallel, and then concatenates these four outputs prior to applying the activation function. This feature allows a model to place greater importance on the kernel weights that are more helpful in defining features of the input data.

As the depth of models increased, two issues became apparent: (1) training the number of parameters was becoming computationally expensive; and (2) the issue of vanishing gradients during backpropagation was more prevalent [15].

At ILSVRC 2015 a team from Microsoft introduced ResNet [73], a model with 152 hidden layers collected into residual blocks that help avoid the vanishing gradient problem. The residual blocks allow data to shortcut (*i.e.*, skip) layers in computation, meaning that additional layers will not increase the error rate of the model as it can easily skip layers that are not improving the accuracy. The data that has skipped computation is concatenated with the other outputs of the layers prior to the following layer.

Since its introduction, the Inception architecture has been updated multiple times [158, 156] with the most recent version combining Inception with the residual blocks of ResNet. The Inception-ResNet architecture is the current highest performing classifier on the ImageNet dataset [156].

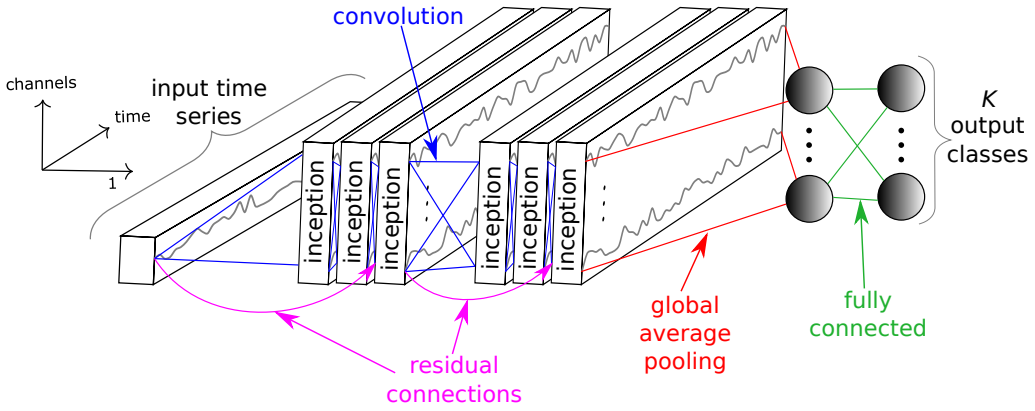


Figure 4.3: A single Inception classifier for time series classification [50].

4.2 InceptionTime

The similarity between image data and time series and their rapid advancement and success in computer vision, suggested to researchers that deep learning had potential for success in time series classification [49]. This potential was recently realised in the publication of InceptionTime [50] a class-leading CNN for time series classification based on the Inception-ResNet architecture. I was a co-author on this paper and my contribution is outlined in Section 4.2.2.

4.2.1 Architecture

A single Inception classifier for time series contains two residual blocks each comprised of three Inception modules with a linear 'shortcut' connection around the modules. The shortcuts transfer the input data straight to the input of the following block, thus mitigating the vanishing gradient problem by allowing a direct flow of the gradient [73]. Figure 4.3 depicts an Inception classifier's architecture with 6 Inception modules stacked one after the other.

Figure 4.4 shows a breakdown of an individual Inception module for time series. The first component is the 'bottleneck' layer, which slides n convolutional kernels of length 1 across the M-dimensional multivariate time series input. The result is a time series with n dimensions (where usually $n \ll M$), significantly reducing the dimensionality of the time series as well as the model's complexity and aiding to mitigate overfitting problems for small datasets. The bottleneck allows the Inception classifier to have much longer filters than the ResNet model presented in [182] while having approximately the same number of parameters to be learned. Note that for visualization purposes, Figure 4.4 illustrates a bottleneck layer with $n = 1$.

Following the bottleneck layer, are multiple convolutional layers with different kernel lengths, arranged in parallel such that each have the same time series input (the output of the bottleneck layer). Figure 4.4 shows the three convolutions with length $\in \{10, 20, 40\}$ being applied to the input.

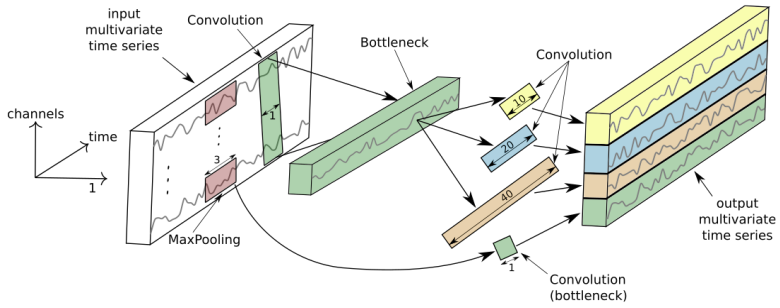


Figure 4.4: A look inside each Inception module of the Inception classifier for time series classification [50].

In addition to the above layers, the model has an additional parallel layer that consists of a pooling layer and a bottleneck. The output of a max-pooling operation over the input is the maximum value in this given window of time series, this gives the model the ability to be invariant to small perturbations. The bottleneck following the pooling layer, reduces the dimensionality of the output.

Finally, the output of each independent parallel convolution and the pooling layer are concatenated to form the new output multivariate time series.

InceptionTime is an ensemble of k Inception models, with each having the same architecture but different initial values for the weights. At prediction time, the k models are each given an even weight.

The ensembling of Inception models was necessitated by the large standard deviation exhibited by single Inception models (this was similarly experienced by researchers applying a ResNet model to time series [49]). The variability derives from both the randomly initialised kernel weights and the stochastic optimization process itself. Rather than training one, potentially very good or very poor, instance of the Inception model, InceptionTime leverages this instability and the classifier's overall accuracy through ensembling.

The following equation explains the ensembling of predictions made by a network with different initialisations:

$$\hat{y}_{i,c} = \frac{1}{k} \sum_{j=1}^{n} \sigma_c(x_i, \theta_j) \quad | \quad \forall c \in [1, C] \tag{4.5}$$

with $\hat{y}_{i,c}$ denoting the ensemble's output probability of having the input time series x_i belonging to class c, which is equal to the logistic output σ_c averaged over the k randomly initialised models.

4.2.2 Receptive Fields for 1-D Convolutional Neural Networks

My major contribution to this paper was to investigate Receptive Fields (RF) on experiments using synthetic time series data. The concept of RF is an essential tool to the understanding of deep CNN [111]. Unlike feed-forward networks a neuron in a CNN depends only on a region of

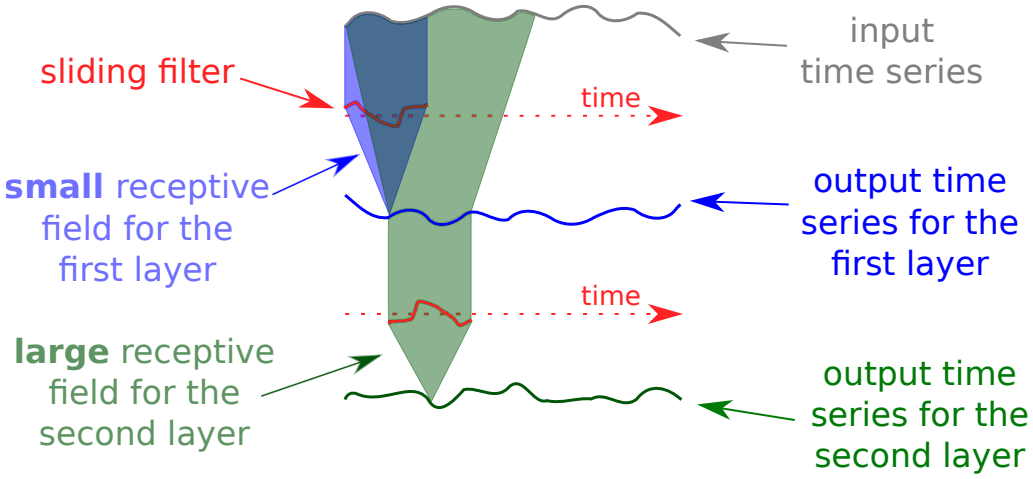


Figure 4.5: Receptive field illustration for a CNN with two layers [50].

the input signal and this region of the input space is called the receptive field of that particular neuron.

For computer vision problems this concept had been extensively studied, (for *e.g.*, [108]), however it had yet to be examined for time series CNN.

For temporal data, the RF can be considered as a theoretical value that measures the maximum field of view of a neural network in a one-dimensional space: the larger it is, the better the network becomes (in theory) at detecting longer patterns.

The formula to compute the RF for a network of depth d with each layer having a filter length equal to k_i with $i \in [1, d]$ is:

$$1 + \sum_{i=1}^{d} (k_i - 1) \tag{4.6}$$

By analyzing this equation, it is evident that adding two layers to the initial set of d layers, will increase only slightly the value of RF. In fact in this case, if the old RF value is equal to $RF^{'}$, the new value RF will be equal to $RF^{'} + 2 \times (k - 1)$. Conversely, by increasing the filter length k_i, $\forall i \in [1, d]$ by 2, the new value RF will be equal to $RF^{'} \times 2$. This is rather expected since by increasing the filter length for all layers, we are actually increasing the RF for each layer in the network. Figure 4.5 provides a further illustration of the RF for two-layers CNN.

Experiments using InceptionTime [50] demonstrated that detecting longer patterns from one-dimensional time series data, requires a larger RF. This parallels the findings from computer vision where larger RFs are used to capture more context [111].

4.2.3 InceptionTime Accuracy

Experiments showed InceptionTime to perform extremely well on the 85 datasets of the UCR repository (see Section 2.1). Figure 4.6 shows a critical difference diagram with InceptionTime

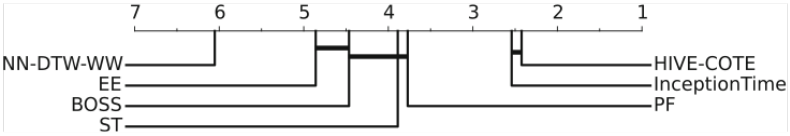


Figure 4.6: Critical difference diagram showing the performance of InceptionTime compared to the current state-of-the-art classifiers of time series data [50].

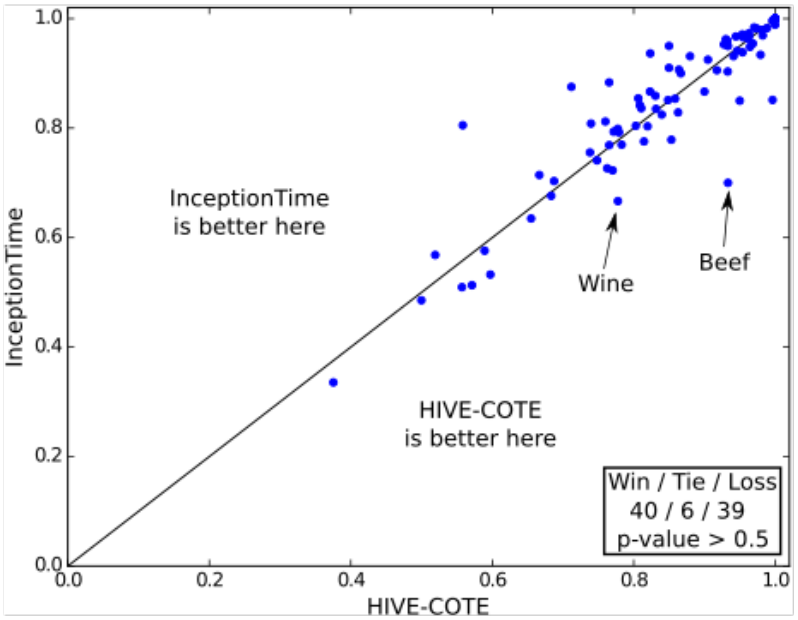


Figure 4.7: Accuracy plot for the 85 datasets of the UCR repository showing a one-on-one comparison between InceptionTime and HIVE-COTE. The vertical axis shows the test accuracy of InceptionTime while the horizontal axis shows the test accuracy of HIVE-COTE [50].

and other current state-of-the-art classifiers for time series data.

It shows that InceptionTime is competitive in accuracy—sharing the same clique—with the class-leading HIVE-COTE algorithm, an ensemble of 37 TSC algorithms with a hierarchical voting scheme [104]. A one-on-one comparison of the two classifiers is shown in Figure 4.7. The results show a Win/Tie/Loss of 40/6/39 in favor of InceptionTime, however the difference is not statistically significant, as illustrated by the critical difference diagram.

While neither algorithm is significantly better in accuracy, the capability of deep learning models to be parallelised makes learning InceptionTime a substantially easier task than training the 37 different classifiers of HIVE-COTE, whose implementation cannot trivially leverage GPU computational power. Consequently, at the time of its publication, InceptionTime was considered the new state of the art for time series classification, as its accuracy is equal to that of HIVE-COTE (the original version) while being much faster.

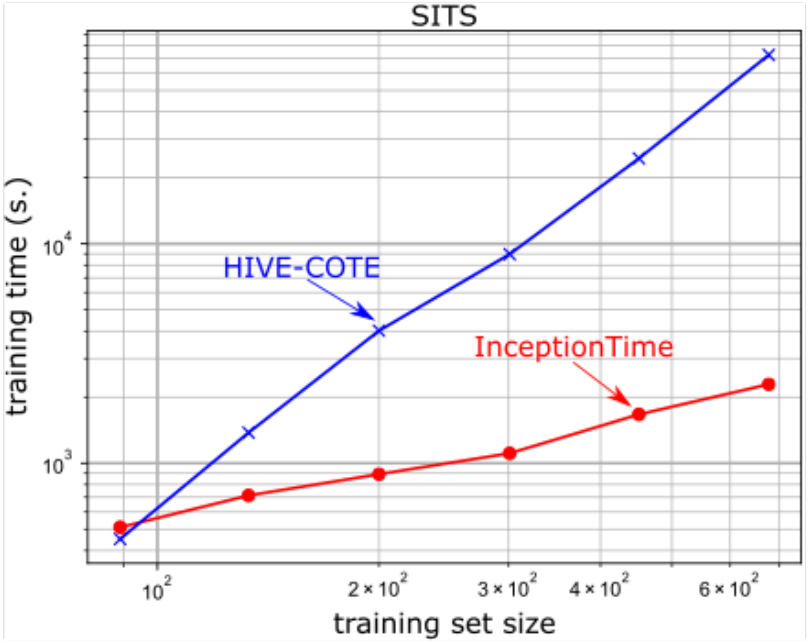


Figure 4.8: Training time of InceptionTime and HIVE-COTE against training set size for the SITS dataset. Note that each axis is shown in log scale [50].

4.3 InceptionTime and Deep Learning using SITS

As discussed substantially in previous chapters, the datasets of the UCR repository are orders of magnitude smaller than the SITS dataset, meaning that for the classification of SITS for land cover, a highly accurate classifier must also be scalable in order to train and classify in a reasonable timeframe.

Figure 4.8 shows the computation time of InceptionTime against HIVE-COTE on a subset of the SITS dataset of approximately 1 million time series[4]. This illustrates that InceptionTime is an order of magnitude faster than HIVE-COTE, and the trend suggests that this difference will only continue to grow, rendering InceptionTime (and deep learning models in general) a clear leader for large datasets.

This idea is supported in [126] where the authors find the CNN model applied along the time axis to significantly outperform the benchmark random forest algorithm and all other deep learning models on a land cover classification task. This model was further used to map the state of Victoria, Australia in [124].

Based on these reasons, I chose to utilise and explore deep learning models for the remainder of my Ph.D. research. I also note that when compared to Proximity Forest (3), deep learning models do not require all of the data to be loaded into main memory, as is the case with tree-based models. This means that the memory complexity of deep learning models is far superior.

[4]This is the same 1M-instance subsample of SITS data used in the Proximity Forest experiments in Section 3.3

4.4 Conclusion

In this chapter I have discussed the existing research into deep learning for time series classification. While the quantity of research output it still lags behind that of deep learning for image recognition, the InceptionTime classifier discussed in the first deep learning architecture to reach the state of the art. The model ensembles multiple CNN models that each utilise the most sophisticated techniques used in CNN for computer vision—Inception modules and residual connections. The InceptionTime, and other deep learning models, are not only highly accurate but also highly scalable—two orders of magnitude faster than current state-of-the-art models such as HIVE-COTE—as shown on SITS data for land cover mapping. The existing research suggests that deep learning, specifically CNN, are the best scalable classifiers for large time series datasets.

4.4.1 Contributions

The contributions resulting from this chapter of my Ph.D research are as follows:

1. Collaborating with international colleagues to implement the most successful deep learning architecture for time series classification and co-authoring its publication.
2. Exploring and defining the concept of receptive fields for time series CNN.
3. Devising and performing experiments using synthetic time series data and contributing to the writing of the InceptionTime manuscript.

Part II

Domain Adaptation for Land Cover Classification

5

Related Work and Background

The first part of this dissertation looked at classification models for large time series datasets. This was motivated by the production of land cover maps from SITS, which, until recently, were being produced by classification methods that are optimised for *standard* data, rather than time series.

The recent emphasis on scalable time series classifiers has quickly resulted in deep learning becoming the state of the art for producing land cover maps [184, 124]. However, these classifiers cannot be applied in all scenarios where land cover maps are needed and one such situation is explored in Part II of this thesis.

Modern machine learning methods, including the tree-based and deep learning methods presented in previous chapters, require large quantities of labelled training data for optimal performance. This poses a major issue for land cover classification as labelled data—*i.e.*, pixels where their land cover has been correctly identified prior—are not always readily available.

An area of machine learning research that offers great potential as a solution to this problem is domain adaptation (DA), and this is the focus of the remainder of my Ph.D research, and Part II of this thesis.

In this chapter I will begin by explaining the concept of DA including its two sub-types—unsupervised and semi-supervised DA. Then I will present the state of the art in DA with an emphasis on semi-supervised DA as this is the main focus of my research.

I note that the methods presented in this chapter are not designed for use with time series datasets and that my application to SITS data, explored in Chapter 9, is the first work I am aware of using DA for multivariate time series data.

5.1 Domain Adaptation

In DA, a labelled dataset—the *source domain*—is utilized for the purpose of classifying instances from a dataset where labelled data are scarce or unavailable—the *target domain*. These methods belong to a family of machine learning techniques that deal with data distributions that are not stationary over time or space [169].

There are three possible sources of the variation between the source and target distributions:

1. Concept drift, where $P_{source}(x, y) \neq P_{target}(x, y)$;
2. Covariate shift, where $P_{source}(y|x) = P_{target}(y|x) \; \forall \; x$, but $P_{source}(x) \neq P_{target}(x)$; and,
3. Label shift, where $P_{source}(x|y) = P_{target}(x|y) \; \forall \; y$, and $P_{source}(y) \neq P_{target}(y)$.

There is some disagreement between definitions of DA however the majority of DA research focuses on the case of a covariate shift, where the label space is the same across domains [183]. This is the definition I have also used in my work.

The main purpose of DA is to address this variation in distributions (see Figure 5.1). That is, a decision boundary learnt on source domain data will incorrectly classify target domain data due to the different distributions. A successful DA technique will realign the decision boundary between the source and target domains.

There are two general scenarios that are presented in DA research—*unsupervised* DA and *semi-supervised* DA [90]—which differ in whether labelled target data is available [122]. In unsupervised

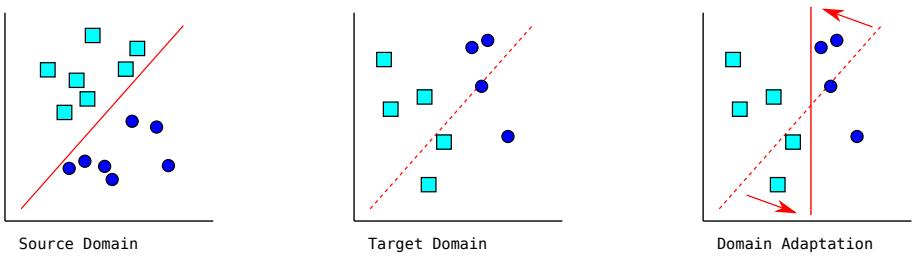


Figure 5.1: A graphical example of how domain adaptation accounts for the distribution of the domains. The decision boundary learnt on the source domain (left) does not correctly classify the data in the target domain (middle). A domain adaptation method adjusts the decision boundary to account for the shift in distributions.

DA, no labelled data are available in the target domain and the methods acquire information only from the structure of the unlabelled data. In semi-supervised DA, some labelled data are available, however there are insufficient samples to train an accurate classifier, so the labelled target data works to complement the source data in training a classifier. In accordance with the definition of DA, it is assumed that sufficient labelled source data is available in both scenarios.

Both unsupervised and semi-supervised DA are highly applicable to a land cover mapping as the study area may have either little or no labelled training data available. For my Ph.D research I chose to focus on the semi-supervised scenario as there is less research into this area at present.

5.2 Unsupervised Domain Adaptation

The main challenge of DA is the gap in feature distributions between the source and target domain. Generally speaking, DA methods approach this in one of two ways:

1. By adapting the source domain data to appear more statistically similar to the target domain, *i.e.*, aligning the distributions of the domains or finding features that are discriminant in each domain; or

2. By an end-to-end process in which a classifier is simultaneously learned from the source and adapted to the target domain data.

Early methods were largely based around domain alignment while recent deep-learning based methods are all end-to-end processes, meaning the training of the model and the adaptation occur simultaneously.

This section presents an overview of many class-leading methods in unsupervised domain adaptation [90].

5.2.1 Importance Weighting Methods

The underlying idea of these method is that some labelled source data might be more similar to the target than others, *e.g.*, forests might be similar in each domain but crops may not be. Importance weighting (IW) methods apply a weight to data from the source domain based on its relevance, and then train a traditional classifier based on this weighted source dataset. There has been various weighting methods developed of which we have chosen kernel mean matching (KMM) [78]. This method begins by applying a non-linear transform to the source and target data to map each of them individually into a Reproducing Kernel Hilbert Space (RKHS). From here, the *importance* weight vector is found through quadratic optimisation by minimising the distance between the means of the projected distributions.

5.2.2 Domain Alignment Methods

Transfer Component Analysis (TCA) [120] is similar to KMM in that it projects the source and target distributions into a RKHS and then seeks to find a transform that minimises the distance between their respective means. The technique differs however in that it uses a novel algorithm to simultaneously apply dimension reduction and minimise distance. Once the transformation is learned, it is applied to both the source and target dataset before the learning and classification tasks, respectively.

Subspace Alignment (SA) [51] performs a similar process to TCA—dimension reduction and domain alignment— but simplifies it by separating the problems and solving them independently. It first calculates the c principal components (PC) in each of the source and target domains, and then defines a transform to align the source PCs to those of the target. Once learned, the transform can be used to align the source data to the target domain before training a classifier.

Geodesic Flow Kernel (GFK) [63] extends upon SA by incorporating all of the infinite intermediate subspaces between the source and the target domains. Again here, the method projects the source and target data into a d-dimensional subspace and finds the shortest distance between the two domains (known as the geodesic flow). Then kernel function is then computed by integrating along the geodesic flow path, where the starting point is the source domain and the end point is the target domain. This kernel is then used to transform the labelled source data before training a classifier.

Feature-Level Domain Adaptation (FLDA) [91] attempts to model the shift of each feature between the source and target domains. A transfer distribution is calculated by modelling the features of the source domain based on their prevalence in the target domain. Then, a linear classifier is learned by minimising the expected value of a specified loss function under this transfer distribution.

5.2.3 Optimal Transport for Domain Adaptation

Optimal Transport for DA (OT) [32] finds the transformation matrix between the source and target distributions based on minimising a cost function. Where OT differs from other methods is that the transform can be different for each point of the source data, however a constraint is included to ensure that points from the same class have similar transforms. A classifier is then trained on the transformed source data. We have tested two different cost functions—earth mover's distance (EMD) and sinkhorn distance (SH) [32]. The potential for using OT in remote sensing has been highlighted by researchers previously (see [31]), however no experiments have been performed on SITS data at present.

5.2.4 Deep-learning based Unsupervised DA methods

Deep-learning based methods are end to end methods in which the classifier is trained and adapted in a single process. Deep learning based unsupervised DA methods can be categorised into 4 categories [195]: (1) Discrepancy-based methods; (2) Adversarial discriminative methods; (3) Adversarial generative methods; and (4) Self-supervision-based methods.

Discrepancy-based methods feed both labelled source and unlabelled target domain data into a model and measure the discrepancy between the output of the activation layers. Methods differ by whether they share weights between the source and target models and how the measure of discrepancy used. The most well-known method in this family is Deep Correlation alignment (DeepCORAL) [154], which uses a non-linear transform to align the correlations of the activation layers of two deep learning models (one for each domain). A sub-family of these methods is optimal transport, which attempts to minimise the discrepancy using optimal transport [31]. One method, Deep Joint Distribution Optimal Transport (DeepJDOT) [36], uses a deep learning model to learn a common latent space for the source and target distributions, which consequently conveys discriminant information for both domains.

Adversarial discriminative models use adversarial methods to create confusion between the domain distributions. The primary example of this is Domain-Adversarial Neural Networks (DANN) [56] uses a deep learning model with two outputs to simultaneously minimise the loss of predicting the class of the instance while maximising the loss of predicting whether the instance came from the source or target domain. Consequently, the model learns to accurately classify classes while having increasing difficulty distinguishing between domains. Adversarial Discriminative DA (ADDA) [171] uses a loss function based on a generative adversarial network (GAN) model [66] and splits the optimization process into two separate objectives for the generator and discriminator.

Rather than separating the two components as in ADDA, adversarial generative models combine the discriminative model with a GAN. There are various high performing methods in this category—*e.g.*, Coupled GAN [106], Sim GAN [151], and CyCADA [77]—however, they have low data scalability and are therefore less useful on modern *big* datasets [195].

Self-supervised-based methods recruit learning from self-supervised learning to reduce the distance between the source and target domain. This can be performed using autoencoders such as in Multi-task Auto-encoders (MTAE) [59], or parallel convolutional and deconvolutional networks as in Deep Reconstruction-Classification Networks (DRCN) [60].

5.3 Semi-supervised Domain Adaptation

While the majority of DA research has focused on unsupervised DA, it has been shown that adding a small amount of labelled target data—as few as one instance per class—in conjunction with a semi-supervised DA method can markedly improve classification accuracy [2, 187, 44, 137]. This suggests that semi-supervised DA methods might be more important to, and have more success on real-world DA problems.

Notwithstanding its potential importance and success, much of the research that has occurred in semi-supervised DA over the last decade has simply mirrored the trajectory of unsupervised DA with minor modifications added to utilise additional labelled target data [56, 172]. Most early methods worked by mapping the source and target domain data to a new feature space, ensuring that instances from the same class map to a similar area of the space (regardless their originating domain) [63, 181]. Recently it has been shown that despite the seemingly small difference between semi-supervised and unsupervised DA contexts, techniques focused on aligning the domains generally perform poorly in the semi-supervised setting [138]. In fact, in many cases it has been shown that training a classifier naively, *i.e.*, training on the labelled source and target data pooled together, will outperform these domain alignment-based methods [193, 83]. This suggests that there are likely design principles specifically beneficial to semi-supervised DA.

5.3.1 Kernel Manifold Alignment

In general, it has been shown that domain alignment-based methods do not handle non-linear deformations or high-dimensional data problems particularly well and are therefore of less relevance to many modern DA problems (including the field of remote sensing) [167]. Kernel manifold alignment (KEMA) [167] was developed to combat the issue of dealing with high-dimensional data, by creating the data transform based on only a few labelled samples from each domain. However when KEMA was used in a land cover mapping problem [10] the results were unsatisfactory, yielding only 70 percent accuracy on a 7-class classification problem.

5.3.2 Domain-adversarial Neural Networks

Domain-adversarial neural networks (DANN) [56] were presented in Section 5.2.4 as an unsupervised method, however it can also be utilized for semi-supervised DA. This method learns class labels that are domain-independent by training a CNN with a loss function comprised of two components—a class-specific component and a domain-specific component. The function simultaneously minimises the loss of predicting class labels while maximising the loss of predicting whether the instance came from the source or target domain. Consequently, the model learns

to accurately classify classes while having increasing difficulty distinguishing between domains. DANN has been successfully applied to land cover mapping in the unsupervised DA case [13] but its method of aligning domains has been shown to be less successful in the semi-supervised case [137].

5.3.3 Minimax Entropy

Minimax entropy (MME) [137] was the first deep learning-based method specifically designed for semi-supervised DA. This method learns a prototype (a representative datapoint) for each class in the labelled data and then minimizes the distance between these prototypes and the unlabelled data, thus learning discriminating features. As the labelled data is dominated by instances from the source domain, the method uses a novel adversarial method to shift the class prototypes towards the target domain data. This shift means that MME performs best in cases where the classes are represented approximately evenly in the labelled target data, and particularly in the case when they can be chosen selectively. This publication was significant in the field as it also introduced the benchmark dataset—DomainNet—used for comparison of semi-supervised DA algorithms in computer vision (see Section 8.2). As such, the accuracy of MME is often the baseline in performance that newer techniques, presented in the following sections, are compared to.

5.3.4 Unified Learning of Opposite Structures

Unified Learning of Opposite Structures (UODA) [133] attempts to align the cross-domain features using a generator and two domain-specific classifiers with opposing objective functions. The source classifier is used to *scatter* the source features to reduce the bias of the decision boundary towards the source domain. The target classifier clusters target features to enhance intra-class density and inter-class divergence. By alternating the source and target classifiers, the cross-domain features are aligned across the source and target domains.

5.3.5 Attraction, Perturbation, Exploration

Attraction, Perturbation, Exploration (A.P.E.) [88] is a semi-supervised technique consisting of three steps: attraction, perturbation, and exploration. The attraction step aligns the unlabelled target distribution with the labelled target distribution, minimising the intra-domain discrepancy within the target domain. The perturbation step then perturbs the given target samples in a way that reduces the intra-domain discrepancy, that is blurring the alignment of the labelled target to the labelled source distributions. Finally, the exploration step aligns features by class, a manner

that is complementary to the attraction step, by selectively aligning unlabelled target features complementary to the perturbation scheme.

5.3.6 Deep Co-Training with Task Decomposition

Deep Co-Training with Task Decomposition (DeCoTa) [186] is the current state of the art in semi-supervised DA. The technique decomposes the semi-supervised DA task into two separate and smaller tasks: (1) a semi-supervised learning task using the labelled and unlabelled data from target domain, and (2) an unsupervised DA task across the domains using the labelled source data and the unlabelled target data. A classifier is trained independently for each of these two tasks and the highly confident predictions from each model are then used as pseudo-labelled data to train the alternate model. This is performed iteratively until the discrepancy between models is negligible. The authors show that this method performs substantially better than any other existing method for semi-supervised DA on the benchmark DomainNet dataset[1].

5.4 Domain Adaptation for Satellite Image Time Series

When considering land cover mapping from SITS, DA can be applied across two different domains: temporal and spatial. DA across the temporal domain occurs when an existing map is in need of updating but labelled data from the present timeframe is unavailable or scarce. In this case, the map from a previous year (or years) is be used as the source domain and adapted to map the present day land cover [161, 39, 162]. DA across the spatial domain occurs when there are little or no labelled training data available in the study region (the target domain), so data from a different geographical region is used as the source domain. Depending on the DA approach used, either the classifier is trained on the labelled data from the source domain and then adapted to target domain, or the labelled data from the source domain are adapted to appear similar to the distribution of the target domain and are then used to train a classifier.

There is little existing research into using DA with SITS for land cover. One study [161] applies temporal DA to update existing land cover maps where up-to-date labelled training data are not available. The authors found that using maps from previous years and DA, can produce highly accurate maps of the present day land cover ($> 80\%$ accuracy). It was also found that the present maps increased in accuracy with the number of previous maps available to adapt data from.

At present, there are no studies of spatial DA for SITS that I have knowledge of, which is interesting as research using SITS has frequently been limited by the availability of training instances. I note that there is existing research into DA for land cover mapping concerning the

[1] This method is currently available as a preprint on arXiv, however the source code is not provided and therefore the results have not been replicated or verified

use of single images [169, 114, 72, 12, 9], as opposed to a temporal series of images, but given this difference in data distribution, this is a very different machine learning problem and requires different techniques.

5.5 Conclusion

This chapter presents the machine learning field of DA and a review of the existing literature in both unsupervised and semi-supervised DA. Early methods focussed on aligning the source and target domains, while modern methods have favoured end-to-end processes.

The majority of the methods presented in this chapter have been developed for computer vision, and to the best of my knowledge they have not been applied to land cover mapping or time series data. This may be due to a lack of performance in this specific application, or practical reasons such as data storage or computational cost.

6

The Need for Domain Adaptation using SITS

The application of DA to time series data is notably absent from the existing body of research into DA. While it is difficult to determine exactly why this might be the case, the logical starting point is to investigate whether DA is required at all. This section will present experiments on SITS data for two geographical domains to explore whether it is possible to produce accurate land cover maps of an area where labelled data are scarce, without the use of DA.

The experiments implement 4 variations of training CNN models (*e.g.*, pre-training on the labelled source data and then incrementally learning on the labelled target data). The results identify that DA would be beneficial for all small quantities of labelled target domain data.

This chapter also introduces the SITS dataset and deep learning architecture in depth, as they will be used in the experiments presented in the remaining chapters of this thesis also.

The work in this section is based on that presented at the International Workshop on the Analysis of Multitemporal Remote Sensing Images 2019 and published in:

Lucas, B., Pelletier, C., Inglada, J., Schmidt, D., Webb, G., Petitjean, F.: Exploring data quantity requirements for domain adaptation in the classification of satellite image time series. In: F. Bovolo, S. Liu (eds.) MultiTemp 2019, 10th International Workshop on the Analysis of Multitemporal Remote Sensing Images, August 5-7, 2019 – Shanghai, China. IEEE, Institute of Electrical and Electronics Engineers, United States of America (2019).

6.1 Satellite Image Time Series

The experiments were performed using the SITS data acquired by the Sentinel-2A satellite, starting on 1 January 2016 and running through to 26 December 2016 (its twin satellite Sentinel-2B was launched in March 2017). Table 6.1 shows the dates of the images for each satellite tile used in the experiments (tiles discussed further in Section 6.1.3).

6.1.1 Preprocessing

All Sentinel-2A data have been collected and prepared by colleagues from the CESBIO lab using `iota2` software [79]. The key steps in this process are outlined below:

- Atmospheric, adjacency and slope effects are corrected for using the MAJA processing chain [70]. The output of this are top-of-canopy images with associated clouds masks. I note that only the images with a cloud-cover of less than 80% are processed by MAJA.

- Each image is comprised of 13 spectral bands—four of which are recorded at a spatial resolution of 10 metres; six that are recorded at a resolution of 20 metres, which are then reinterpolated at 10 metres; and three that are recorded at a resolution of 60 metres, which are discarded at this stage as they are only used in atmospheric correction and cloud detection.

- The images are gapfilled using a linear temporal interpolation with a time gap of 10 days, resulting in 37 dates for each pixel [80, 38]. Ten days is a natural choice for the time gap as it represents the revisit frequency of one Sentinel 2 satellite. However, the orbit of the satellite results in some *overlapping* between areas and therefore some pixels are imaged more frequently than others. Thus, gapfilling is a vital processing step to ensure that each pixel has the same number of timestamps. It also allows for the correction of images that are compromised by cloud-cover. Table 6.1 shows the dates of the original images for each satellite tile and the interpolated dates.

After the MAJA processing, the resulting instances (pixels) are each represented by a multivariate time series with 10 variables (one for each spectral band) of length 37. The data has been normalized per spectral band using values from the source domain data. Following [126],

Table 6.1: Original image dates for each tile used in the experiments and the interpolated dates after pre-processing (all from 2016)

T31TEL	**T31TDJ**	**T32ULU**	**Interpolated Dates**
12-MAR	12-JAN	26-JAN	01-JAN
22-MAR	12-MAR	05-FEB	11-JAN
08-APR	22-MAR	09-MAR	21-JAN
28-APR	29-MAR	26-MAR	31-JAN
08-MAY	08-APR	29-MAR	10-FEB
18-MAY	08-APR	08-APR	20-FEB
21-MAY	11-APR	28-APR	01-MAR
28-MAY	18-APR	05-MAY	11-MAR
07-JUN	28-APR	08-MAY	21-MAR
20-JUN	01-MAY	25-MAY	31-MAR
27-JUN	18-MAY	28-MAY	10-APR
30-JUN	21-MAY	07-JUN	20-APR
07-JUL	28-MAY	24-JUN	30-APR
10-JUL	07-JUN	24-JUN	10-MAY
17-JUL	10-JUN	07-JUL	20-MAY
20-JUL	20-JUN	17-JUL	30-MAY
30-JUL	27-JUN	27-JUL	09-JUN
06-AUG	07-JUL	13-AUG	19-JUN
16-AUG	10-JUL	16-AUG	29-JUN
19-AUG	17-JUL	23-AUG	09-JUL
26-AUG	20-JUL	26-AUG	19-JUL
29-AUG	27-JUL	02-SEP	29-JUL
05-SEP	30-JUL	12-SEP	08-AUG
08-SEP	06-AUG	22-SEP	18-AUG
25-SEP	16-AUG	25-SEP	28-AUG
28-SEP	19-AUG	02-OCT	07-SEP
05-OCT	26-AUG	05-OCT	17-SEP
15-OCT	29-AUG	12-OCT	27-SEP
18-OCT	05-SEP	22-OCT	07-OCT
18-OCT	15-SEP	22-OCT	17-OCT
18-OCT	28-SEP	01-NOV	27-OCT
18-OCT	08-OCT	01-DEC	06-NOV
07-NOV	15-OCT	04-DEC	16-NOV
17-NOV	18-OCT	11-DEC	26-NOV
27-NOV	18-OCT	14-DEC	06-DEC
04-DEC	04-NOV	21-DEC	16-DEC
07-DEC	14-NOV	31-DEC	26-DEC
14-DEC	17-NOV		
17-DEC	27-NOV		
24-DEC	07-DEC		
27-DEC	14-DEC		
	17-DEC		
	27-DEC		

a variation on min-max normalization has been used, replacing the absolute minimum and maximum values with the 2nd and 98th percentile values, respectively. The percentiles used are estimated using all of the values of the series at each individual timestep. This normalization differs from the usual method for time series classification [7] but deliberately avoids two potential pitfalls in using standard methods. First, it retains the relative scale of the spectral bands as this is important to SITS data (for instance, in the calculation of normalized difference vegetation

index). Second, if the data were normalized per image, the ability to track changes over time would be lost. The normalization method used preserves both the capacity to combine band values and to track changes through time.

It should be noted that all of the models in my experiments require all data to be of the same spatial resolution and be of the same length (number of time steps). This is a current limitation of using deep learning models on time series data in general, and not otherwise related to my work specifically.

6.1.2 Reference Data

The reference data are the same as those used previously to produce a land cover map of France in 2016 with the methodology presented in [80]. The reference data originate from four sources:

1. The Agricultural Land Parcel Information System (2016) (*Registre Parcellaire Graphique*): a compilation of data gathered from farmers' declarations of agricultural land [22].
2. Urban Atlas (2012): a land cover dataset gathered by the European Environment Agency (EEA) detailing the land cover of cities in continental Europe at a very high resolution (2.5 metres) using 27 urban classes [94].
3. The CORINE Land Cover Inventory (CLC 2012): an inventory of land cover information gathered by the EEA using 44 land cover classes at a spatial resolution of 250 metres [19].
4. French National Geographic Institute 'BD-Topo': a national topographical map of produced by the government of France [116].

Information from these sources has been amalgamated to create a dataset using a nomenclature of 30 land cover classes:

- Five urban classes: High-density Urban, Low-Density Urban, Industrial, Parking, Roads;
- Fourteen vegetation classes: Rapeseed, Winter Wheat and Barley, Spring Barley, Pea, Soy, Sunflower, Corn, Corn silage, Rice, Beetroot, Potatoes, Grassland, Orchards, Vineyards;
- Seven natural and semi-natural classes: Deciduous Forest, Coniferous Forest, Lawn, Woodlands, Surface Minerals, Beaches and Dunes, Glaciers; and,
- Four *other* classes: Peat, Marshland, Inter-tidal Land, Water.

6.1.3 Source and Target Tiles

The SITS data available originate from three study areas, with each area representing a Sentinel-2 tile (110km $\times$ 110km). One tile is used as a source domain for all experiments and the remaining

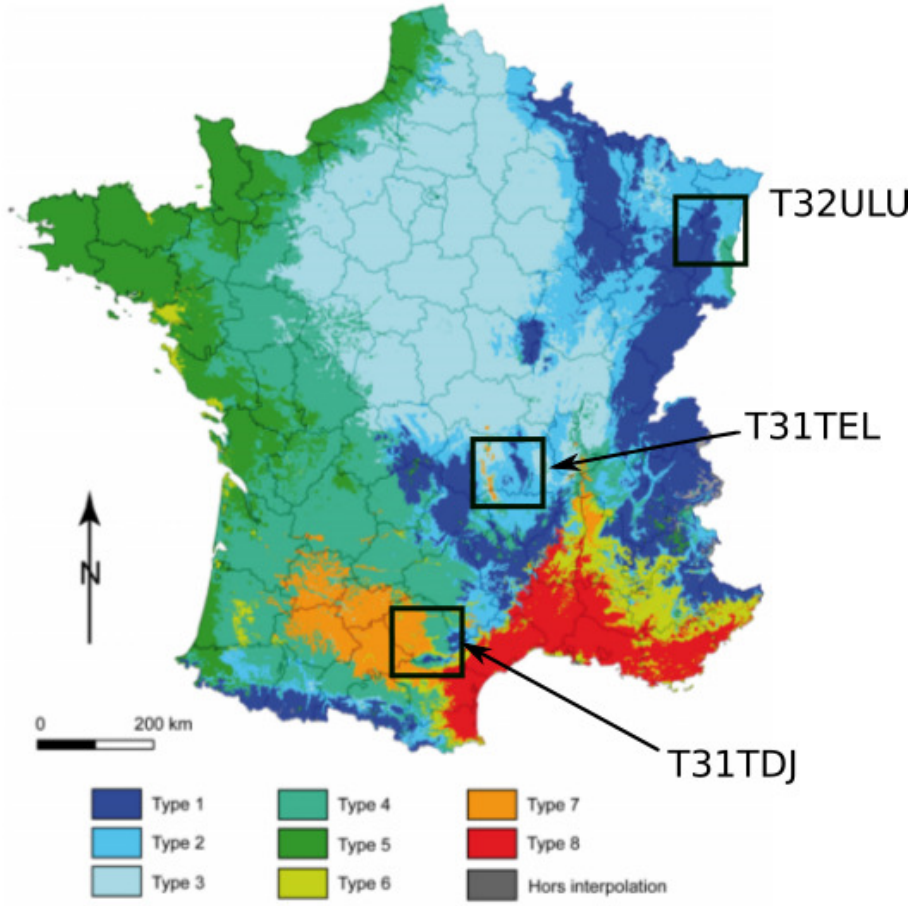


Figure 6.1: Climate map of France from [84] with the three study regions identified.

two as target domains (however some exploratory experiments only use one target domain). The regions are all located in France (see map in Figure 6.1) as there is full reference data available as outlined in Section 6.1.2. The source domain (tile T31TEL) was chosen at random amongst the available Sentinel-2 tiles and is located within a highland region known as Massif Central (45.1°N, 2.6°E). The two target tiles were chosen specifically to observe the variation in results between a target region with a similar climatic profile (T32ULU) and a target region with a very different climatic profile (T31TDJ). Target domain T31TDJ is located near the city of Toulouse in south-west France (43.6°N, 1.4°E), and T32ULU is located in the north-eastern region of France called Grand Est, which includes the city of Strasbourg (48.4°N, 7.5°E).

The CESBIO colleagues who provided the preprocessed data also provided it with predefined train and test sets per tile. I chose not to modify this split as it has been performed such that instances that belong to the same polygon are in the same set. A polygon is a contiguous area with the same land cover—*e.g.*, a farm, forest or residential area—and consequently instances from the same polygon have near-identical profiles. For example, Figure 6.2 depicts three of the spectral bands from three different pixels of sunflower from within the same polygon. The similarity between these instances demonstrates that if the data were split at random, rather than blocked

by polygon, and these instances were distributed to both the train and test sets, the problem of classifying them would be trivial. Retaining the train-test split as provided therefore ensures independence between training and testing sets (as per [135]).

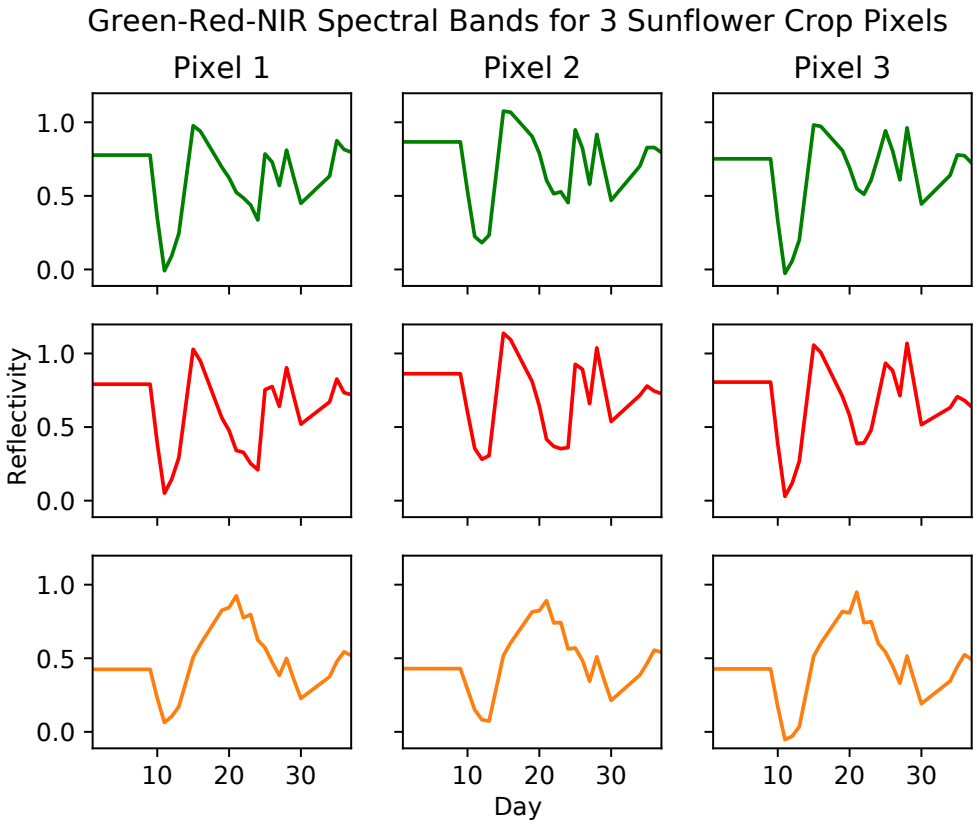


Figure 6.2: The green, red and near infrared reflectance time series for 3 different sunflower pixels located in the same polygon.

Table 6.2 displays the total number of instances per domain and per set. I note that while this shows *all* of the target training data, I have conducted experiments under the condition that only a predetermined quantity is available (per experiment), and the evolution of test accuracy is studied for increasing quantities of target training data.

Table 6.2: Total number of train and test instances (pixels) available for each domain.

	Source T31TEL	**Target** T31TDJ	**Target** T32ULU
Train	12,647,452	8,758,196	15,122,125
Test	-	3,371,843	5,599,461

A comparison of the land cover classes of the training data for the regions is displayed in Table 6.3.

It is noted that the experiments in this Chapter use tile T31TEL as the source domain and tile T31TDJ as the target domain. The displayed test accuracy is the performance of the classifier on the target test data only.

Table 6.3: Distribution of land cover classes across each domain.

Label	Description	**Source** T31TEL	**Target 1** T31TDJ	**Target 2** T32ULU
1	Urban (high density)	16709 (0.13%)	18242 (0.14%)	9871 (0.08%)
2	Urban (low density)	740326 (5.85)	307343 (2.43)	942652 (7.45)
3	Industrial	502479 (3.97)	188150 (1.49)	649285 (5.13)
4	Parking	9198 (0.07)	2779 (0.02)	20469 (0.16)
5	Road	57634 (0.46)	8898 (0.07)	74980 (0.59)
6	Rapeseed	79401 (0.63)	247425 (1.96)	291462 (2.30)
7	Wheat & Barley (winter)	509236 (4.03)	773027 (6.11)	444315 (3.51)
8	Barley (spring)	22135 (0.18)	45397 (0.36)	81282 (0.64)
9	Pea	15298 (0.12)	103890 (0.82)	49802 (0.39)
10	Soy	6296 (0.05)	262310 (2.07)	177084 (1.40)
11	Sunflower	298067 (2.36)	1823222 (14.42)	106120 (0.84)
12	Corn	609941 (4.82)	305467 (2.42)	2204111 (17.43)
13	Corn silage	827715 (6.54)	339535 (2.68)	644489 (5.10)
15	Beetroot	207575 (1.64)	9636 (0.08)	302543 (2.39)
16	Potatoes	26617 (0.21)	3465 (0.03)	40855 (0.32)
17	Grassland	2277897 (18.01)	533604 (4.22)	1119211 (8.85)
18	Orchards	527 (<0.01)	16434 (0.13)	7337 (0.06)
19	Vineyards	2578 (0.02)	357489 (2.83)	19145 (0.15)
20	Deciduous forest	1088129 (8.60)	926583 (7.33)	1972989 (15.60)
21	Coniferous forest	4732777 (37.42)	1091930 (8.63)	5373252 (42.48)
22	Lawn	128711 (1.02)	682709 (5.4)	148231 (1.17)
23	Woodlands	347466 (2.75)	368759 (2.92)	52902 (0.42)
24	Minerals	381 (<0.01)	8483 (0.07)	2072 (0.02)
27	Peat	0 (0)	0 (0)	5324 (0.04)
28	Marshland	0 (0)	71985 (0.57)	7130 (0.06)
30	Water	140359 (1.11)	261436 (2.07)	375212 (2.97)
	TOTALS	12,647,452 (100)	8,758,196 (100)	15,122,125 (100)

6.2 Model Architecture

The CNN model for the experiments presented here and in Chapters 9 and 7 is the TempCNN [126], which has been shown to be a highly accurate model for pixel-based analysis of SITS data. It has been demonstrated to significantly outperform other types of deep learning models, including recurrent neural networks, at large geographical scale.

The model comprises 3 convolutional 'blocks', followed by 1 fully-connected block and a softmax layer (see Figure 6.3). The convolutional block consists of 64 convolutional filters of length 5, followed by a batch normalization layer, a dropout layer with a rate of 0.5, ending with a ReLU activation function. The convolutions are 1-dimensional and are performed along the temporal axis only. The fully-connected layer has 256 neurons, followed by the same batch normalization layer, dropout and ReLU function. The final layer in this case is a softmax with 30 units representing the 30 land cover classes of the classification problem (see Section 6.1.2).

The experiments in this chapter were implemented using the Keras library [29].

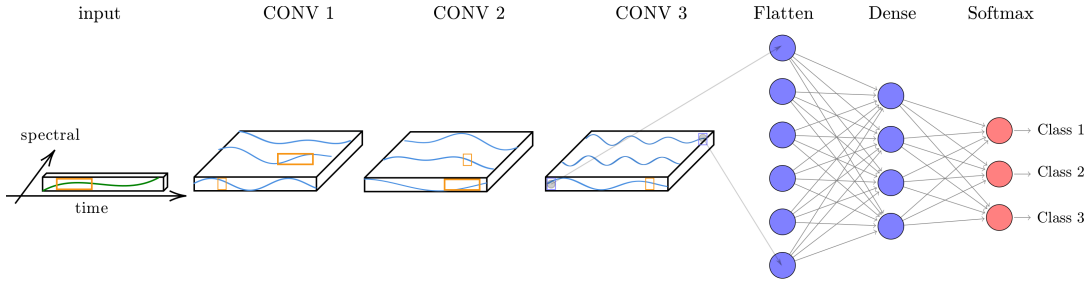


Figure 6.3: The TempCNN model architecture, as presented in [126].

6.3 Experimental Design

In order to assess the number of labelled instances required from the target domain to improve the classification performance, I evaluate four training configurations, which includes two baseline configurations:

1. *Source only:* In this configuration the model is trained on the complete source train data only. This configuration will set the lower bound on test accuracy by learning wholly on data with a different distribution.

2. *Target only:* In this configuration the model is trained on the available target train data only. This configuration will act as a control by demonstrating the test accuracy possible from a given amount of test data if no source domain data is utilised.

3. *All-weights:* In this configuration the model is first trained on the complete source train data, then the model is incrementally finetuned on the available target train data.

4. *Softmax-only:* In this configuration, once the model is trained on the source domain, the weights are frozen with the exception of the softmax layer, and only this final layer is retrained on the target train data. In theory, this will allow the model to learn the general features of the classes from the source data and the more discriminant features from the target train data.

Experiments in configurations 2, 3 and 4 are performed using an exponentially increasing quantity of labelled target data (by polygon), starting with 1 polygon of data (278 instances on average) and increasing to 31 473 polygons (representing all 8.7M available target training instances). Increasing by polygon, rather than data quantity, replicates data availability in a real-world scenario, as data are provided by farm or forest, rather than by pixel. This also makes for a more difficult problem, as rather than having an increasing random sample, the data are not distributed across the whole domain and do not represent the accurate class distribution of the area.

6.3.1 Results

The experimental results are presented in Figure 6.4, where the target domain overall test accuracy is plotted against the quantity of target training data for each configuration.

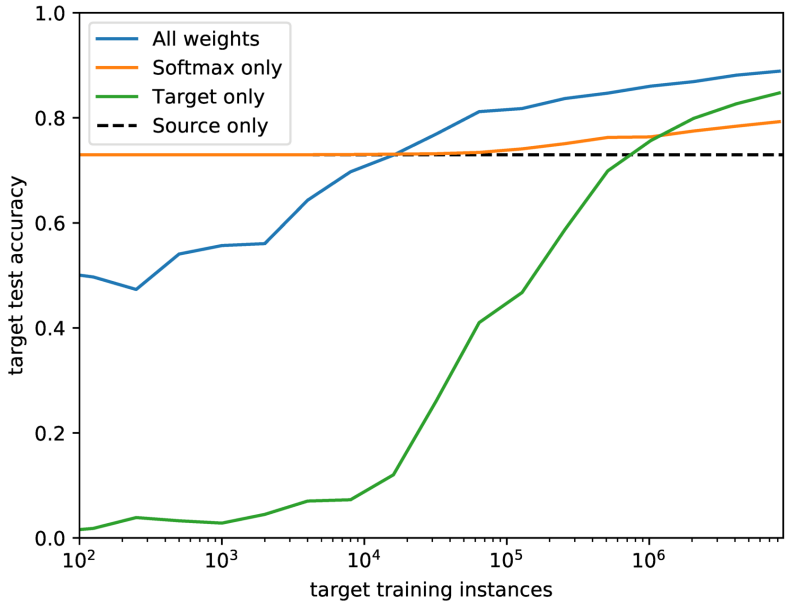


Figure 6.4: Overall test accuracy against target training data for four training configurations: Source only, Target only, All-weights, and Softmax-only.

The first baseline model, Source only, performs quite well with an overall test accuracy of 73.0% on target data. The Softmax-only model returns a higher accuracy than the source model for all quantities of target data (see Figure 6.5). It also outperforms the All-weights model up to 16 000 instances (approximately 90 polygons). Above 16 000 instances, the All-weights model is the most accurate classifier, including the situation where all target train data are available (8.7M instances)—88.9% versus 84.7% compared to the Target only model.

The All-weights configuration begins by decreasing in accuracy below the Source only model when presented with little target training data. It is likely that this initial dip is due to the available target training data having low variability, being from few polygons, and consequently *pulling* the model towards the few classes and polygons present. After the initial dip, the accuracy then increases and becomes the most accurate model for moderate to large data quantities (16k-8M instances).

A visual analysis of the results is presented in Figure 6.6, where I compare the classification of the Target only, All-weights and Softmax-only models for 512 polygons (108 893 instances) of target data. The Target only model predicts noticeably fewer classes, suggesting that not all classes are present in the training data. There is also some difference in the models with both

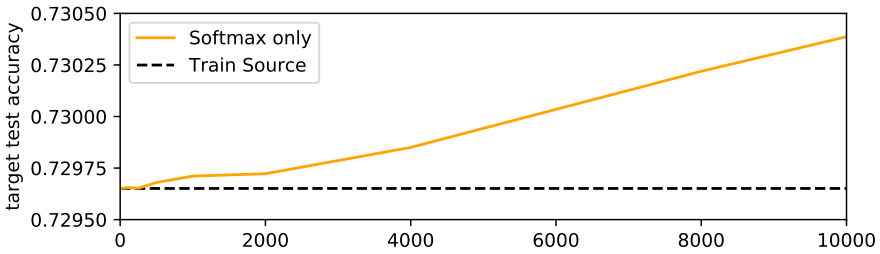
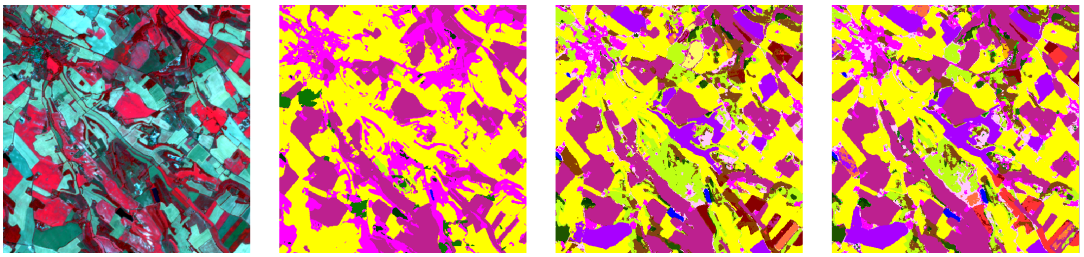


Figure 6.5: A close-up of overall test accuracy against target training data for Source only and Softmax-only.



Sentinel-2 | Target only | Softmax-only | All-weights

Figure 6.6: A Sentinel-2 false-color image and three land cover maps obtained by training on 100 000 target instances.

Source and Target data available, with the All-weights model outperforming the Softmax-only model in accordance with the analysis discussed prior.

6.3.2 Discussion

These experiments demonstrate the need for DA for classifying SITS data in regions where little training data are available. The results show that if no labelled training data is available, a CNN trained on data from a similar region will return moderate test accuracy of 73.0%. This model performs quite well with over 16 000 instances required to improve upon its accuracy. When moderate to large amounts of data are available, using a model pre-trained on a source domain will result in better overall test accuracy when compared to a model trained on target domain data only—88.9% versus 84.7%. When few labelled target instances are available however, training a model on the source domain and then continuing to learn it on target domain data will be less successful than simply using a model trained on source only. This suggests that when low quantities of labelled target data are available, applying a successful DA technique could be highly advantageous.

6.4 Conclusion

In this chapter I have presented a experiments to explain why DA has great potential for land cover mapping in regions where labelled training data are scarce or unavailable. It has shown that

the potential benefit of a DA method increases as the source and target domains areas differ in distribution. Based on this background, the remaining chapters of my Ph.D will be dedicated to researching DA for SITS.

7

Unsupervised Domain Adaptation

Unsupervised DA is used when there are no labelled data available in the target domain, and therefore a classifier must be trained using data from a source domain and unlabelled data from the target domain.

This area has been the focus of the majority of DA research and, if a successful method for Earth observation were to be developed, it could be used to map the land cover of areas without any local data collection. This represents the basis for this chapter: To explore the suitability of existing unsupervised DA methods for use with SITS to produce land cover maps of regions where no labelled data are available.

In this chapter I will present experiments comparing existing unsupervised DA methods using SITS data, where each of these methods proves unsuccessful for our purposes. After presenting the experiments, the chapter will detail synthetic examples exploring the potential reasons why each method underperforms on time series data. These experiments form preliminary work to better inform my intended future work on producing an unsupervised DA method for land cover mapping. After presenting my research I will outline my direction for future work on unsupervised DA for land cover mapping with SITS.

I note that this chapter is based on work presented at the 2020 IEEE International Geoscience and Remote Sensing Symposium and published in:
Lucas, B., Pelletier, C., Schmidt, D., Webb, G.I., Petitjean, F.: Unsupervised domain adaptation techniques for classification of satellite image time series. In: IGARSS 2020 - 2020 IEEE International Geoscience and Remote Sensing Symposium, pp. 1074–1077 (2020).

7.1 Experiments with Unsupervised Domain Adaptation

The aim of these experiments is to explore whether existing unsupervised DA methods can be applied to SITS to produce accurate land cover maps where no reference data is available. The methods were applied standard, *i.e.*, they were not adapted in any way to work with the temporal structure of SITS data. The methods used in these experiments are:

1. Importance weighting using kernel mean matching (IW-KMM) [78];
2. Transfer Component Analysis (TCA) [120];
3. Subspace Alignment (SA) [51];
4. Geodesic Flow Kernel (GFK) [63];
5. Feature-Level Domain Adaptation (FLDA) [91];
6. Optimal Transport using earth mover's distance (OT-EMD) [32]; and,
7. Optimal Transport using sinkhorn distance (OT-SH) [32].

A full description of each of these methods is provided in Section 5.2.

7.1.1 Data

The experiments performed in this Chapter use the SITS data as presented Chapter 6. The following provides a brief overview of the dataset, however please refer to Section 6.1 for more detail.

The data were acquired by the Sentinel-2A satellite between 1 January 2016 and running through to 26 December 2016. The revisit frequency of a single Sentinel-2 satellite is approximately 10 days and therefore the resulting time series have a length of 37.

The training data are labelled with one of 30 land cover classes:

- Five urban classes: High-density Urban, Low-Density Urban, Industrial, Parking, Roads;
- Fourteen vegetation classes: Rapeseed, Winter Wheat and Barley, Spring Barley, Pea, Soy, Sunflower, Corn, Corn silage, Rice, Beetroot, Potatoes, Grassland, Orchards, Vineyards;

- Seven natural and semi-natural classes: Deciduous Forest, Coniferous Forest, Lawn, Woodlands, Surface Minerals, Beaches and Dunes, Glaciers; and,
- Four *other* classes: Peat, Marshland, Inter-tidal Land, Water.

The experiments were conducted using two Sentinel-2 tiles (110km × 110km) as study areas—one as the source domain and the other as the target domains The source domain, tile T31TEL, is located within a highland region known as Massif Central (45.1°N, 2.6°E). The target domain, tile T32ULU, is located in the north-east near the city of Strasbourg (48.4°N, 7.5°E). The pair of tiles (each 110 km × 110 km) were chosen as they are climatically similar [80].

Table 6.2 displays the total number of instances available per domain and per set. The source domain has approximately 12.6 million training instances while the target domain has 5.6 million (unlabelled) test instances. These methods in these experiments use the target data without the class labels, so although the correct labels are known, they are only used to calculate test accuracy and not during the training process.

7.1.2 Classifiers

The DA methods tested in these experiments are independent of the classifier chosen. As such, I have defined three *standard* classifiers to use in conjunction with these methods: (1) A Random Forest (RF) classifier with 100 trees and a number of random selected features per node equal to the square root of the total number of features; (2) A Support Vector Machine (SVM) classifier with a linear kernel and squared-hinge loss function; and (3) A Neural Network (NN) classifier with 4 hidden layers.

The NN classifier was a fully-connected neural network for all methods with the exception of IW-KMM, as this method retains the temporal aspect of the data, and therefore a TempCNN [126] model was more applicable (see Section 6.2 for more information on TempCNN).

The classifiers were also run without any DA—applied by training on source and testing on target. These results are referred to as naive RF, naive SVM, and naive CNN.

7.1.3 Method Parameters

Each method was run using the maximum data quantities within the capacity of a machine with 64Gb of RAM. For methods that required subsampling in order to converge (because, for instance, their process might be quadratic with the quantity of data), this meant learning the transfer on a quantity of 5 000 randomly selected source and target instances, and training the classifier on 100 000 training (source) instances. Where the method included dimensionality reduction, 30 components were chosen in order to make sure that most important information was retained.

Every attempt was made to keep all parameters as close to those used in each of the respective publications.

7.1.4 Results and Discussion

The overall test accuracy of each DA method is shown in Table 7.1. I note that each method shown in the table was run 5 times and the overall test accuracy averaged, with the exceptions of SA which only completed 3 runs within the 3 day experiment period, and FLDA, which failed to finish a single run in this same timeframe.

Table 7.1: Average overall accuracy (%) on the target domain (tile T32ULU) for each unsupervised domain adaptation method (based on 5 runs)

DA method	RF	SVM	(C)NN
(Naive)	81.96	80.36	**83.72**
IW KMM	64.59	60.23	33.60
TCA	61.01	54.40	58.61
SA	72.99	58.74	63.83
FLDA		*****	
GFK	78.62	47.73	79.32
OT-EMD	41.77	45.01	49.27
OT-SH	56.33	41.27	58.80

The results show that all of the existing techniques for unsupervised DA are unsuccessful when applied to the SITS data, performing significantly worse than even naive implementations of classifiers with no DA applied. From these experiments, GFK appears to be the method with the most potential for this data, and the RF classifier appears to be the strongest classifier when used with the DA methods.

Two possible explanations for these results were explored further:

1. *Temporal shifts in classes:* As the DA methods are not designed for time series data, they treat each timestamp as a feature. This means that if the change in domains is represented by a shift in the classes along the temporal axis—eg. if a crop ripens at different times of the year—the methods are unlikely to adapt for this, because different timestamps are seen like different spectral bands without trying to link them together. To test this, I have created a synthetic time series dataset with only two classes, as shown in Figure 7.1. The change between source and target domains is represented by a shift along the temporal axis. The instances are distributed as 150 from class 1 and 150 class 2. Random noise was added to the basic shapes to create individual instances for training and testing.

Table 7.2 shows the results of applying each DA method to data with temporal shifts of increasing magnitudes. Where it is not otherwise stated the classifier used for these experiments was a random forest as this was the most successful above.

The results show that GFK and both OT methods can handle small temporal shifts between

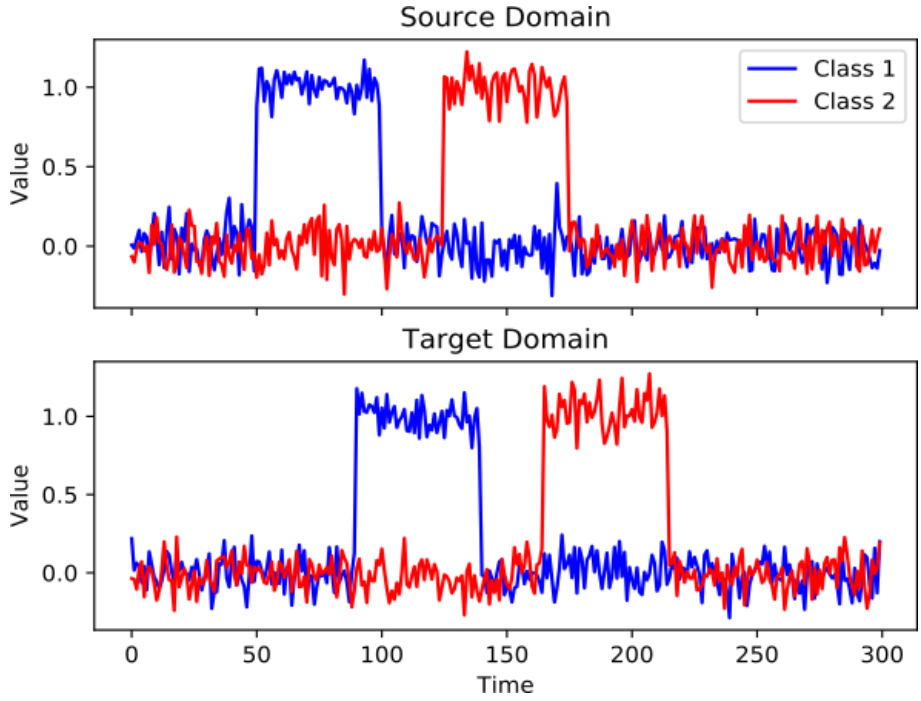


Figure 7.1: An example of instances of the synthetic data used for experiments where the source and target differ along the temporal axis.

Table 7.2: Overall classification accuracy on a synthetic dataset with source and target domains defined by a temporal shift

DA method	**Domain shift** (timestamps)			
	10	20	30	40
Naive RF (no DA)	1	1	1	0.59
Naive CNN (no DA)	1	1	0.94	0.60
SA	0.68	0.68	0.58	0.48
GFK	1	1	1	0.64
TCA	0.90	0.86	0.91	0.37
OT-EMD	1	1	1	0.75
OT-SH	1	1	1	0.73

domains, however once the shift reaches 40 timestamps all methods are scarcely better than a coin toss.

2. *Change in class distributions:* Another possible reason for the inaccuracy of the methods could be the difference in class distribution between domains. As there are no target labels available, the methods inherently assume a similar class distribution between the source and target domains. However in our data, and as would be the case for most regions of the world, various classes differ in proportion from source to target—*e.g.*, corn represents only 4.82% of the land cover of the source but 16.14% of the target, while coniferous forest makes up 18.01% of the source compared to only 8.67% of the target.

To test this, I create another synthetic binary dataset (similar to that of the previous experiment but without temporal shift) where the source and target differ only by the prevalence of each class. The results of applying each DA method to this data is shown in Table 7.3.

Table 7.3: Overall classification accuracy on a synthetic dataset with source and target domains defined by different class distributions

DA method	Proportion of class 1 in source / target			
	0.5 / 0.5	0.1 / 0.7	0.9 / 0.1	0.01 / 0.99
Naive RF (no DA)	1	1	1	1
Naive CNN (no DA)	1	1	1	0.85
SA	1	1	1	0.85
GFK	1	1	1	1
TCA	1	1	1	0.71
OT-EMD	1	0.40	0.20	0.02
OT-SH	1	0.80	0.81	0.05

The results show that OT is the method most affected by the change in class distribution, whereas the other methods are generally robust to the change except in the most extreme case.

When we combine the results of the two synthetic experiments, GFK appears to be the most robust method to the two possible situations. Although GFK outperforms both naive RF and CNN on these experiments, this is not the case for experiments on real dataset, meaning that there is possibly another reason that prevents GFK from outperforming naive approaches.

7.2 Conclusion

The first major outcome of the experiments presented in this chapter is to reinforce that, at present, target labels are invaluable in producing accurate land cover maps with SITS. This is a large contributor to exploring semi-supervised methods in the following chapters of this thesis.

Secondly, the experiments found that many existing unsupervised DA methods are not useful for use on time series data, and the additional experiments show that this is likely due to the nature of the SITS data and the class distribution change between domains. This suggests that a method that is specific to Earth observation applications will likely be required.

7.2.1 Future Work

The work covered by this chapter is an introductory exploration of unsupervised DA for SITS. In the future I would like to expand upon this work to assess the newer deep learning-based unsupervised DA methods and work towards developing an accurate unsupervised DA method for the classification of SITS.

7.2.2 Contributions

The preliminary research presented in this chapter motivates the need to develop an unsupervised DA technique specifically designed for producing land cover maps from SITS data.

8

Ensemble Co-training for Domain Adaptation

The findings of the unsupervised experiments of the previous chapter highlighted the value of labelled target data to DA for SITS at present. Based on these findings, I focused my subsequent research on developing semi-supervised DA methods.

Semi-supervised DA is used when the quantity of data available in the target domain is insufficient to train an accurate model, but ample labelled data is available in the source domain. Recent semi-supervised DA methods such as UODA [133], A.P.E. [88], and DeCoTA [186] have marked a significant shift from viewing semi-supervised DA as 'an unsupervised DA problem with some additional data to make it simpler' to a difficult problem in its own right [186]. These methods approach semi-supervised DA as a problem located at the intersection of semi-supervised learning and DA [186]. This new outlook of treating semi-supervised DA as its own research field has also resulted in the establishment of a benchmark dataset—DomainNet—to compare semi-supervised DA methods.

Motivated by these recent advancements, I sought to incorporate ideas from semi-supervised learning into a method for semi-supervised DA. The resulting method is Encoda—Ensemble Co-training for DA—a method that incorporates the semi-supervised learning concept of co-training and beneficially leverages the variability of CNN to improve accuracy through ensembling. My experiments show that it is competitive with other methods on the benchmark DomainNet dataset, while having the added advantage of being able to be implemented in the scenario where the source data is not available—*i.e.*, only models pre-trained on the source domain are required (akin to the benefits of Sourcerer presented in Chapter 9).

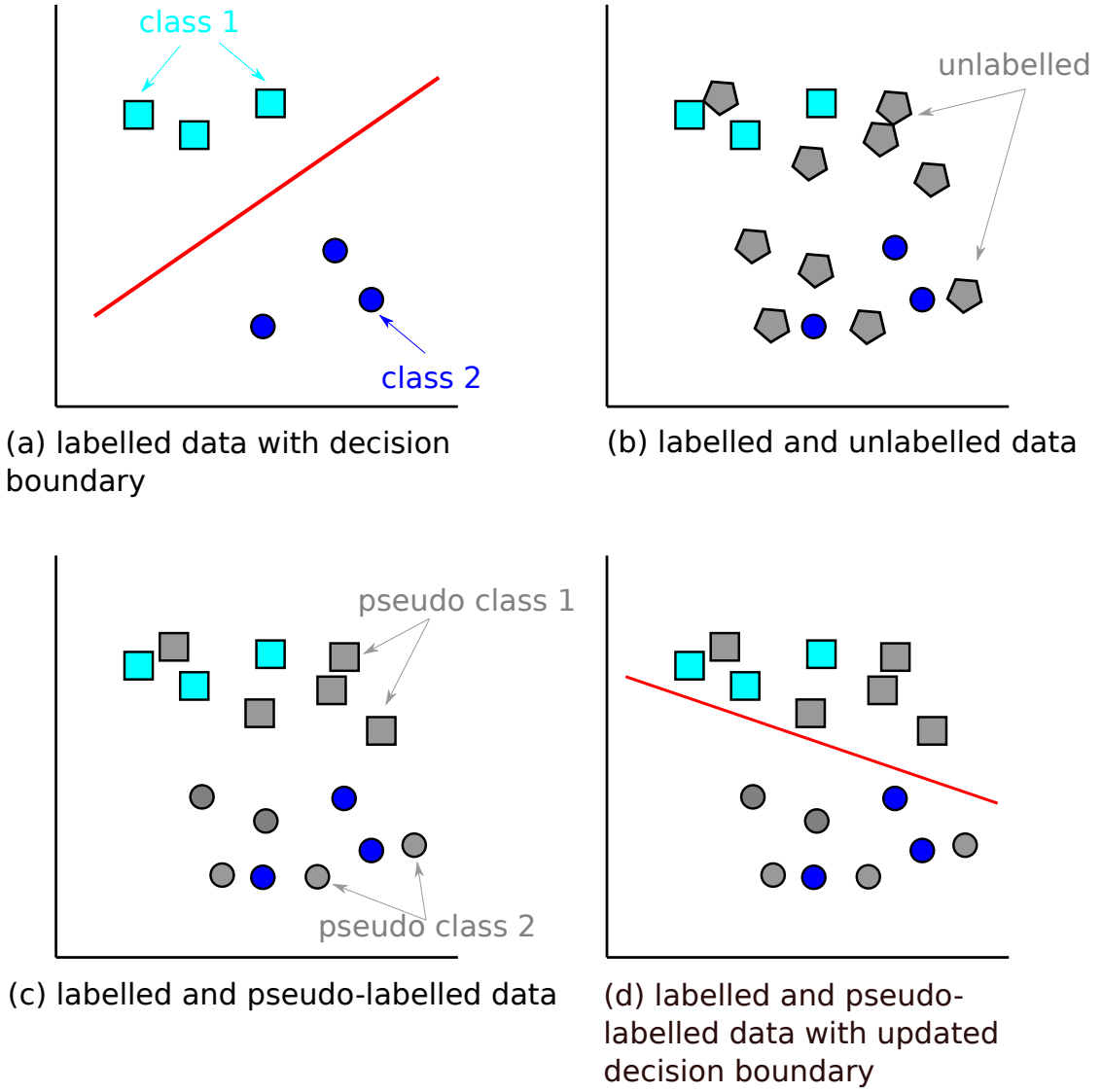
In this chapter I begin by discussing semi-supervised learning and introducing the benchmark DomainNet dataset. Then, I present how Encoda works and follow this with experiments demonstrating its performance against the state of the art. Finally, I discuss my intentions for future work with this model.

8.1 Semi-Supervised Learning

The field of semi-supervised learning shares many commonalities with semi-supervised DA (including its adjective) and thus it is not surprising that researchers have begun to share knowledge between the two areas. Semi-supervised learning is the field of machine learning where we are attempting to train a classifier using a small amount of labelled data and a large amount of unlabelled data [24]. This scenario bridges the gap between supervised machine learning—where labels are readily available—and unsupervised learning—where instances are not labelled. This is an important task as often in real-world applications, a small amount of labelled data is available but obtaining large amounts is time-consuming and/or costly [46, 24].

One common group of techniques employed in semi-supervised learning algorithms is self-training and co-training [17, 166] (see Figure 8.1). Self-training starts by learning a classifier on the available labelled data and then using this to predict on the unlabelled data. Next a decision metric is used to determine which of the predictions are most likely to be correct and these data, referred to as *pseudo-labelled data*, are added to the existing labelled training data. This can be repeated multiple times over. Co-training expands this process to include multiple classifiers, each usually using different sets of features from the data. In this case, the pseudo-labelled data from one classifier are used to train a different classifier, reducing the probability of reinforcing errors resulting from incorrectly predicted pseudo-labelled data [178].

The other common technique appearing in semi-supervised learning methods is data augmentation. The premise of these methods is to add perturbations to the labelled data and to train on both the augmented data in addition to the original labelled data. The MixUp



(a) labelled data with decision boundary

(b) labelled and unlabelled data

(c) labelled and pseudo-labelled data

(d) labelled and pseudo-labelled data with updated decision boundary

Figure 8.1: The concept of co-training is a semi-supervised learning technique involving assigning pseudo-labels to unlabelled data to train a classifier.

algorithm [192] applies a perturbation towards the decision boundary, which acts to regularise the model by preventing it from overfitting to the few available labelled samples.

Both of these strategies have potential to be effective tools for use in semi-supervised DA and while I have proceeded with using co-training for my research, I would suggest that data augmentation strategies also need to be explored fully in the future.

8.2 DomainNet

The recent emphasis and improvement in semi-supervised DA algorithms has facilitated the need for an open access dataset to use for comparison. The dataset that has emerged as the benchmark is a subset of the DomainNet [127] dataset specifically for semi-supervised DA. The original DomainNet dataset contains approximately 600 000 images from six domains—sketch, real,

quickdraw, painting, infograph, and clipart—and 345 possible class labels. The semi-supervised variant of this dataset has 4 domains—sketch, real, painting, and clipart—and 126 class labels, and was first used in experiments demonstrating the MME algorithm [138]. An example of the bird class and the strawberry class from each of the four domains is shown in Figure 8.2.

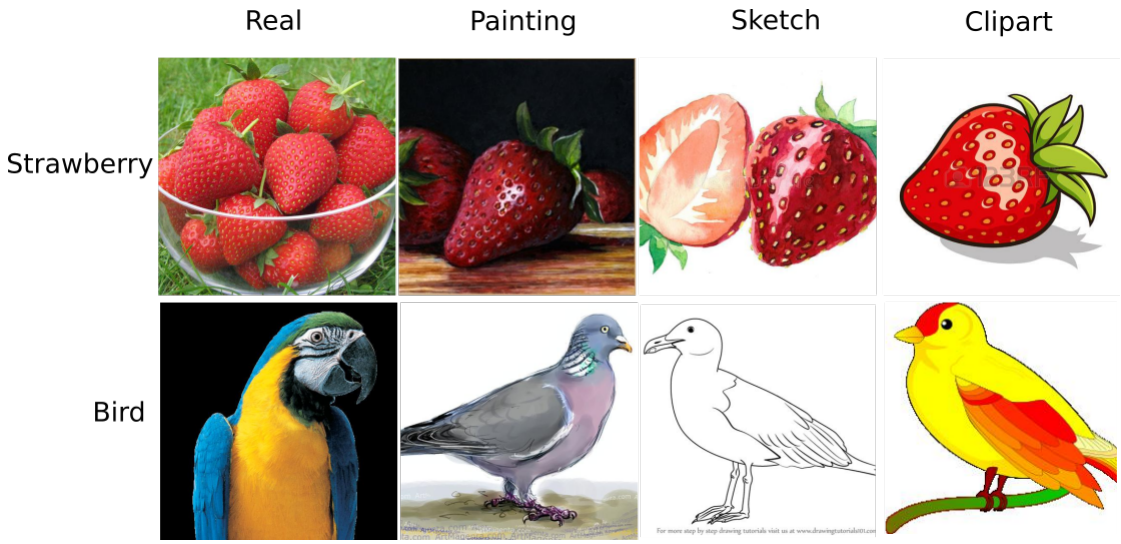


Figure 8.2: Example images from each of the 4 domains (real, sketch, painting, and clipart) in the semi-supervised variant of the DomainNet dataset for 2 classes (bird and strawberry) [138].

Each domain in the semi-supervised DomainNet dataset has predefined subsets for: Labelled Source, Labelled Target, and Unlabelled Target (Target Test). The labelled target data are 3 instances from each class in the target domain, while the unlabelled target have no labels available for training but use them for testing purposes. The quantity of each subset of the domains is shown in Table 8.1.

Table 8.1: Quantity of each subset of the semi-supervised DomainNet dataset

Domain	Labelled Source	Labelled Target	Unlabelled Target
Real	70 358	378	69 980
Sketch	24 582	378	23 826
Clipart	18 703	378	18 325
Painting	31 502	378	31 124

8.3 How Does Encoda work?

Encoda is a semi-supervised DA method that utilises the semi-supervised learning technique of co-training and combines it with the performance benefits of model ensembling. Previous research in this area showed models to have a high variability in overall accuracy with most techniques publishing results averaged over multiple runs [138]. Instead, I have used this variability to my advantage by ensembling multiple models (trained via co-training).

The process of implementing Encoda for semi-supervised DA is shown in Algorithm 4. First, we initialise an ensemble of k models. Then each model is trained on the labelled source domain data, stopping when the training accuracy reaches 100% or 50 epochs, whichever occurs sooner. Once this is completed for all k models, the source domain data is no longer required. An alternate application is to obtain or download k models pre-trained on the source domain, meaning the user does not have to be in possession of the source data at any stage.

The next stage is to *finetune* the models on the small amount of labelled target data that is available. To do this, each model of the ensemble is trained on the labelled target, until the training accuracy is greater than 90%[1]. Stopping at this point ensures that the models do not 'forget' the features learned on the source domain and by overfitting to the small amount of labelled target data.

Then the ensemble model is used to predict the unlabelled target domain data. The predicted label of an instance is assigned by averaging the softmax outputs of each model per class. The label associated with the maximum average output value across all classes is then assigned to the instance, and we think of this value as how 'confident' we are in this prediction. Using this method, as opposed to each model having an equal vote, means that one model that is very confident in its prediction, *i.e.*, has a high value for the output, can outweigh multiple models all predicting a different class but with less confidence. The confidence is saved alongside the class prediction for use in the next step.

A new target training dataset has now been created by pairing the unlabelled target data with the pseudo-labels. However, we know that many of these pseudo-labels are incorrect, because if a model trained on only the source data and a small amount of target data were enough, we would not need DA. To address this, all predictions with a confidence of less than a predefined threshold are discarded, leaving only labelled instances that we are confident are correctly labelled [27]. We then pool this data with the labelled target data to create the final training dataset.

The final step is to train the models of the ensemble on the pooled pseudo-labelled and labelled target data.

After the training the ensemble with Encoda, it is ready to be used for classification. The ensemble predicts an unlabelled instance by taking an average of the softmax output value of each model in the ensemble and assigning the label associated with the maximum value.

8.4 Experiments

To demonstrate the performance of Encoda, I have tested it on the DomainNet dataset against the state of the art in semi-supervised DA (as described in Section 5.3). The methods used for the

[1] This value is a hyperparameter that can be adjusted/chosen as required.

Algorithm 4: *ensemble_cotraining*$(X_s, y_s, X_t, y_t, y_u, k, t)$

Input: S, y_s: Source domain data and labels, respectively
Input: T, y_t: Target domain data and labels, respectively
Input: T_u: Unlabelled target domain data
Input: k: number of models in the ensemble
Input: α: minimum confidence threshold

```
// Initialise ensemble of size k
```
$M \leftarrow \emptyset$
for $i = 1$ **to** k **do**
  $m_i \leftarrow initialise\,()$
  Add model m_i to M
end

for $i = 1$ **to** k **do**
```
  // train model on source
```
  $m_i \leftarrow train\,(m_i, S, y_s)$
```
  // finetune model on labelled target
```
  $m_i \leftarrow finetune\,(m_i, T, y_t)$
end

```
// create pseudo-labels
```
$y_{t'} \leftarrow predict\,(M, T_u)$

```
// filter by confidence threshold
```
$T_u, y_{t_\alpha} \leftarrow filter\,(T_u, y_{t'}, \alpha)$

```
// train ensemble on labelled and pseudo-labelled target
```
for $i = 1$ **to** k **do**
  $m_i \leftarrow train\,(m_i, T + T_u, y_t + y_{t_\alpha})$
end

return M

comparison are:

1. Minimax Entropy (MME) [138]
2. Domain-adversarial Neural Networks (DANN) [56]
3. Unified Learning of Opposite Structures (UODA) [133]
4. Attraction, Perturbation, Exploration (APE) [88]
5. Deep Co-training with Task Decomposition (DeCoTa) [186]

The results are also published for a 'naive' model where the model is trained on all of the labelled data—with the labelled source and target domain data pooled together and shuffled.

8.4.1 Model Architecture and Hyperparameters

The architecture used for each model in the Encoda ensemble is ResNet34 [73], a 34-layer CNN with residual connections (see Section 4.1.3). This is also the architecture used in the work of all of the state-of-the-art methods facilitating an even comparison of methods, ensuring that increases in accuracy are attributable to the DA method applied and not simply to a larger architecture.

Table 8.2: Source-Target domain pairings for the semi-supervised experiments using Encoda.

Source	Target
Clipart	Sketch
Painting	Clipart
Painting	Real
Real	Clipart
Real	Painting
Real	Sketch
Sketch	Painting

While this architecture does not produce the highest overall accuracy on DomainNet [127, 138], it does allow for timely prototyping and research when compared to the training time of larger ResNet models. ResNet34 has *only* 21M parameters to train compared to over 58M for the larger (and often more accurate) ResNet152.

The results of the experiments presented here are for Encoda comprised of 20 models. The optimal threshold α was found via the best validation accuracy by training the models on the pseudo-labelled data only and testing on the labelled target data as an ensemble. This process resulted in most source-target pairings using a threshold of 0.99, with the exception of clipart to sketch, which returned 0.95. I highlight that the optimal test accuracy for the sketch to painting was returned using a threshold of 0.95, however cross-validation suggested that the optimal value was 0.99.

An analysis of the number of models in the ensemble k is presented in Section 8.4.5 and the confidence threshold in Section 8.4.4.

8.4.2 Data

The dataset used for the following experiments is the semi-supervised variant of DomainNet, as discussed in Section 8.2. The dataset consists of images from four domains: Real, Clipart, Painting, and Sketch, and 126 class labels. When used as a target domain, the labelled target data includes 3 labelled instances per class. Following other publications in the area, the results are examined for 7 different source-target domain pairings (as shown in Table 8.2).

8.4.3 Overall Accuracy

The classification accuracy of each method is shown in Table 8.3. I note that the overall accuracy reported for methods other than Encoda is the one published by the authors for the comparable ResNet34 architecture. This ensures that any question of hyperparameter choice and/or optimisation is minimised. First, the results show that DeCoTa is clearly the leading method for each of the 7 source-target pairings[2]. After this, the next three methods—a group

[2]I note that at the time of writing these results have not been published and the authors have not released the source code for verification

that includes Encoda—all return similar performance in each of the seven settings. UODA and APE are recent state-of-the-art methods and Encoda is competitive with these while having the added benefit of not requiring the user to download the source data. That is, Encoda produces state-of-the-art accuracy with the most user-friendly process.

Table 8.3: Overall Accuracy for semi-supervised DA methods on DomainNet dataset—4 domains and 126 classes.

Method	C to S	P to C	P to R	R to C	R to P	R to S	S to P	MEAN
Naive	55.6	60.8	74.5	60.8	63.6	53.3	59.5	61.2
DANN	55.1	59.1	67	62.3	63	57.4	59.7	60.5
MME	59	69.7	79	72.1	69.2	62.2	64.7	68
APE	63.1	76.7	79.4	76.6	72.1	67.8	66.1	71.1
Encoda	**64.2**	**73.5**	**80.6**	**73.6**	**71.2**	**66.1**	**68.5**	**71.1**
UODA	64.1	73.2	80.8	75.4	71.5	64.2	69.4	71.2
DeCoTa	68.6	78.7	81.5	80.4	75.2	71.9	72.7	75.6

8.4.4 Confidence Threshold

The confidence threshold α is one of the two user-defined hyperparameters of Encoda. The choice of alpha in the above results was selected via cross-validation on the labelled target data, but this section presents the results for multiple values to demonstrate the variance in accuracy resulting from this choice.

The choice of α is a balance of having a high enough value that the pseudo-labels are generally correct and the model is not learning incorrect labels, yet low enough that the quantity of pseudo-labelled data is large and variable enough to train a classifier without overfitting to just a few instances.

In the clipart to sketch experiment, a threshold of 0.9999 resulted in having 153 pseudo-labelled instances at 100% accuracy, a threshold of 0.999 resulted in 699 instances at 99.2% accuracy, and a threshold of 0.99 resulted in 2094 instances at 98.1% accuracy.

The full results of using Encoda on DomainNet with four different values of α are shown in Table 8.4. The results show that while a value of 0.99 for α is generally the best, there are situations where a lower values returns marginally better results.

Table 8.4: Overall Accuracy on DomainNet dataset for Encoda with varying confidence threshold α.

Method	C to S	P to C	P to R	R to C	R to P	R to S	S to P	MEAN
Encoda (0.95)	64.2	71.9	79.8	72.4	70.9	65.8	68.6	70.5
Encoda (0.99)	64.1	73.5	80.6	73.6	71.2	66.1	68.5	71.1
Encoda (0.999)	63.5	71.5	79.8	72.5	70.1	65.7	67.7	70.1
Encoda (0.9999)	63.3	71.1	79.4	72.1	69.8	65.6	67.5	69.8

8.4.5 Ensemble Size

The second user-defined hyperparameter of Encoda is the ensemble size. Adding a model to an ensemble will improve accuracy so long as its accuracy is as strong as the other models and the additional model provides some variability in the predictions. Therefore, going from a 3-model to a 5-model ensemble will often be highly beneficial but going from a 15-model to a 20-model ensemble may not be. A larger ensemble size also comes at a cost of computation time and resources.

The results of using Encoda on DomainNet with four different ensemble sizes are shown in Table 8.5. They show that the 20-model Encoda is marginally better than the 10-model version in most but not all cases. The benefit of increasing from 10-models to 20-models is generally far less than from 5-models to 10-models, suggesting that there is diminishing returns from adding more models at this point for this data[3].

Table 8.5: Overall Accuracy on DomainNet dataset for Encoda with varying ensemble size.

Method	C to S	P to C	P to R	R to C	R to P	R to S	S to P	MEAN
Encoda (5 models)	63.2	72.9	80.1	72.9	70.2	65.5	68	70.4
Encoda (10 models)	63.8	73.4	80.4	73.6	70.9	66.0	68.4	70.9
Encoda (20 models)	64.1	73.5	80.6	73.6	71.2	66.1	68.5	71.1

8.5 Conclusion and Future Work

In recent times, multiple new semi-supervised DA techniques have been published [133, 88, 186] marking a significant shift from viewing semi-supervised DA as an easier unsupervised DA problem to approaching it as a research problem in its own right, being located in the intersect of a semi-supervised learning and a DA. My research in this chapter was motivated by these recent advancements and sought to incorporate ideas from semi-supervised learning into a method for semi-supervised DA. The resulting method is Encoda, a method that incorporates co-training from semi-supervised learning and beneficially leverages the variability of CNN for accuracy through ensembling. My experiments show that it is competitive with other methods on the benchmark DomainNet computer vision dataset [127], while having the added advantage of being able to be implemented in the scenario where the source data is not available—*i.e.*, only models pre-trained on the source domain are required.

In the future I intend would like to apply Encoda and other recent semi-supervised DA methods on a land cover classification problem using SITS. The performance on this tasks could inform researchers how best to collect labelled data for the target domain—whether to collect data at random or to seek out a small number of selected samples from each land cover type.

[3] I note that for some applications there may be a significant benefit in using Encoda with greater than 20 models

8.5.1 Contributions

The contributions resulting from this chapter of my Ph.D research and the development of Encoda are as follows:

1. A novel semi-supervised DA method incorporating co-training, a technique borrowed from the field of semi-supervised learning;
2. A semi-supervised DA method with performance competitive with the state of the art on the benchmark DomainNet dataset;
3. A semi-supervised DA method that can be applied to pre-trained models, which may be of potential significance to the classification of SITS data.

9

Sourcerer

The previous chapter developed a semi-supervised DA method based on ensemble co-training. Semi-supervised DA has great potential for the land cover mapping application as resources are rarely available for a data collection campaign at large scale. For example, in creating a national scale land cover map of France researchers required approximately 35 million training instances [80], a quantity that is unfeasible to obtain in many nations, particularly those that are resource-poor. Therefore, a state-of-the-art semi-supervised DA method would allow for the production of high-accuracy large-scale maps at a small fraction of the cost. However, Encoda—the method presented in Chapter 8—proved to be quite computationally expensive to apply to SITS data due to the training of up to 20 classifiers on a training set of over 12 million instances.

This presented the motivation for this chapter: To develop an accurate semi-supervised DA method that uses large scale SITS data to produce land cover maps of regions where labelled data are scarce.

My research into this problem lead to Sourcerer—the first semi-supervised DA technique for time series data that can produce highly accurate land cover maps. Sourcerer is a deep learning-based method for semi-supervised DA based on a Bayesian-inspired, novel regularizer for the

weights of a CNN. My experiments with a 30-class land cover classification problem using Sentinel-2 image time series data demonstrate that Sourcerer outperforms the current state-of-the-art methods in semi-supervised DA, regardless of the quantity of labelled data available in the target domain.

I begin this chapter by presenting Sourcerer, its motivation and how it works as a DA method. I then present experiments demonstrating its effectiveness on SITS data and comparing it against current state-of-the-art semi-supervised DA methods[1]. The final section highlights the contributions to knowledge resulting from Sourcerer and future directions I work like to explore.

The research presented in this chapter is heavily based on the work published in:
Lucas, B., Pelletier, C., Schmidt, D., Webb, G.I., Petitjean, F.: A Bayesian-inspired, deep learning, semi-supervised domain adaptation technique for land cover mapping. *Accepted for publication in Machine Learning in December 2020*, arXiv preprint available: arXiv:2005.11930. (2020)

9.1 How does Sourcerer work?

Sourcerer is a novel Bayesian-inspired deep learning-based semi-supervised DA method specifically designed for use with SITS. It uses a CNN model trained on the source domain as a starting point and improves upon it by further training it on the available labelled target data. The critical and distinguishing feature of the approach is a novel regularizer, *SourceRegLoss*, that is used while training on the target data. This regularizer tunes the amount of *trust* placed on the updates. That is, as the quantity of labelled target data increases, the model places gradually more trust on what is learnt from this data (and consequently, relies less upon the weights learnt from the source data).

Sourcerer not only delivers excellent performance, but is also widely applicable as it does not require access to the source data, but instead works on a model previously trained on that data. In the experiments presented in Section 9.2, the flexibility of the approach is demonstrated by training a model on a source domain once, and then utilising this pre-trained model and Sourcerer to classify two different target domains.

9.1.1 Architecture

The CNN model used to demonstrate Sourcerer is the TempCNN, as presented in Section 6.2. This model has been shown to be a highly accurate model for pixel-based analysis of SITS data and significantly outperforms other types of deep learning models for this application [126].

[1] I note that the research leading to Sourcerer was performed prior to the publication of UODA [133], APE [88], and DeCoTA [186] and therefore these methods are not included in this chapter as comparison methods

9.1.2 Source-regularized Loss Function

To utilise Sourcerer, one must first either train a model using the labelled source data with a standard loss function (for example, categorical cross-entropy loss for a classification problem), or obtain a pre-trained model. Let $\hat{\theta}_s$ denote the estimates of the parameters of the source model. Then, using these estimates as a reference point, the new target model is trained on the labelled target data using the following source-regularized loss function:

$$\text{SourceRegLoss}(X_t, y_t, f_\theta, \hat{\theta}_s, \lambda) = L\big(f_\theta(X_t), y_t\big) + \lambda||\theta - \hat{\theta}_s||^2 \tag{9.1}$$

where:

L is the average loss (calculated per sample);

f_θ is the current model with parameters θ;

X_t, y_t are the target data and labels, respectively;

$\hat{\theta}_s$ are the estimated parameters of a model trained on the source data; and

λ is the regularization hyperparameter.

During training, the proposed regularizer acts to *shrink* the values of the estimated parameters towards those that were learned on the source data. This is done by adding the squared difference between the parameters of the target model, θ, and the estimated parameters of the source model, $\hat{\theta}_s$, to the loss function, penalizing parameter estimates that deviate substantially from the source model. This approach is motivated by the more general ideas of Bayesian inference. In Bayesian inference one formally specifies a prior guess at the likely population values of a model through the mechanism of a prior distribution. The resulting posterior distribution combines the information contained in the sample with the information in the prior. In our case, the use of the source model parameter estimates $\hat{\theta}_s$ as a reference point mimics the use of a prior distribution. The correspondence is even closer than this, due to the relationship between squared-penalties and normal distributions (see Section 9.1.4 for further discussion). A similar approach has been previously used in transfer learning [34, 1], but to the best of my knowledge this is the first time it has been adapted for time series classification, the field of Earth observation and to CNNs in general.

The hyperparameter λ controls the degree to which deviations of the target model from the source model are penalized. In standard Bayesian inference, the information contained in the prior distribution is outweighed by the information contained in the sample as the sample size grows, so that the effects of regularization are greater for small amounts of target data and correspondingly reduced as the amount of target data increases. I discuss in Section 9.1.3 a simple technique for choosing λ that mimics this behaviour. The regularization is applied to all learnable parameters

of the model; for the TempCNN this includes the weights and biases of the convolutional layers, the fully-connected layers, and the batch normalization layers.

I note that optimum results were found by freezing the running mean and running variance parameters of the batch normalization layer after training on the source data. This is because the available target data have low variability as they are from a limited number of polygons and therefore the batch mean and batch variance of these data are not representative of the data as a whole. That is, the batch means are skewed towards the classes present and the batch variance will be lower given the limited number of classes present in the target data.

It is also important to emphasize that the proposed loss function adds no additional computational cost to the training of the target model.

9.1.3 Determining the Regularization Hyperparameter

The amount of regularization applied by Sourcerer is determined entirely by the choice of λ. In this section I propose a heuristic choice that automatically balances the amount of regularization against the amount of available labelled target data. When the quantity of labelled target data is small, we would like the procedure to use a large value for λ, making the values of the weights tend toward the parameters of the source model. To see that such a schedule is sound, I note that as the amount of target data, n_t, grows the average loss is of order $O(1)$, *i.e.*, it does not grow in magnitude with n_t. To ensure that for large amounts of target data the regularization has little effect we require that $\lambda(n_t) = o(1)$, *i.e.*, tends to zero as $n_t \to \infty$. This is a necessary condition for the learning procedure to be statistically consistent.

To achieve this desired behavior, I propose a simple heuristic schedule for λ. We fix the value of λ at the two extreme points: (i) when we have a minimum quantity of target data (t_{min}, $\lambda_{t_{min}}$) and (ii) when we have some large amount of target data (t_{max}, $\lambda_{t_{max}}$). We then fit a concave-up power curve between these points. The usual form of a power curve is:

$$\lambda = A\, {n_t}^{-k} \tag{9.2}$$

where:

λ is the regularization hyperparameter;

n_t is the quantity of labelled target data available; and,

A, k are constants.

Using the properties of a power curve, this formula can also be represented as a linear equation on a log-log scale. Therefore, to find the schedule for λ we find the line that passes through the

log transform of our two points: $(\log(t_{min}), \log(\lambda_{t_{min}}))$ and $(\log(t_{max}), \log(\lambda_{t_{max}}))$, respectively. The slope of the resulting line is:

$$k = \frac{\log \lambda_{t_{max}} - \log \lambda_{t_{min}}}{\log t_{max} - \log t_{min}}. \tag{9.3}$$

Using this slope and the point $(\log t_{min}, \log \lambda_{t_{min}})$, we can define the equation of the line as:

$$\log \lambda - \log \lambda_{t_{min}} = k \left(\log n_t - \log t_{min}\right)$$

and solve for λ, yielding

$$\lambda = \left(\frac{\lambda_{t_{min}}}{{t_{min}}^k}\right) {n_t}^k. \tag{9.4}$$

Some simple and reasonable heuristic choices have been made for some of the free variables. In the (unlikely) case in which only one labelled target instance is available, a very large value for λ will ensure that the model uses the source parameters. By similar reasoning, when a significant amount of labelled target data is available, a suitably small value of λ will allow the model to learn from the target data and largely ignore the source model. Following this argument, we set $t_{min} = 1$, $\lambda_{t_{min}} = 10^{10}$, and $\lambda_{t_{max}} = 10^{-10}$ and Equation 9.4 reduces to:

$$\lambda = 10^{10} {n_t}^k \tag{9.5}$$

where k is now

$$k = -\frac{20 \log 10}{\log t_{max}}. \tag{9.6}$$

This leaves t_{max} as the only free, user-specified hyperparameter of the procedure. I note that as long as $t_{max} > 1$ (a reasonable choice), then $k < 0$ and the schedule (9.5) satisfies the condition $\lambda = o(1)$, as prescribed above. I have performed a sensitivity analysis of this parameter in Section 9.2.6.

9.1.4 Connection to Bayesian Inference

This section will examine the close connection between Sourcerer and Bayesian inference. While this has been noted previously, here I make the connection more explicit.

First I briefly review Bayesian statistics. In a Bayesian approach, we have a probabilistic model of data, $p(y|\theta)$, with unknown parameters θ that we would like to fit to some observed dataset. We further must propose a probability distribution $\pi(\theta)$ that describes our belief about which values of θ are likely to be the (unknown) population value, before seeing the data (*i.e.*, *a priori*). This is called a prior distribution. Bayesian inference proceeds by forming a posterior distribution using

Bayes' rule:

$$p(\theta|y) = \frac{p(y|\theta)\pi(\theta)}{p(y)},$$

where $p(y)$ denotes the marginal distribution of the data. The posterior distribution describes the likelihood of certain values of θ being the true (unknown) population value of θ, after observing data y, and is used as a basis for statistical inference. In practice, computing the normalizing term $p(y)$ is usually infeasible, particularly for complex models such as neural networks, and instead of using the complete posterior it is common practice to estimate θ by maximizing the unnormalized posterior

$$\hat{\theta} = \arg\max_{\theta} \{p(y|\theta)\pi(\theta)\}.$$

A particular strength of the Bayesian framework is that it allows us to formally encode our prior beliefs, or previous information, into the learning process.

Sourcerer, and the source-regularized loss (Eq. 9.1) on which it is based, can be connected to Bayesian inference by noting several equivalencies. First, I note that maximizing the posterior is equivalent to minimizing the negative logarithm of the posterior. The choice of cross-entropy loss for categorical regression is equivalent to choosing our data model $p(y|\theta)$ to be an appropriate neural network with a multinomial logistic regression output layer, and our choice of ℓ_2 regularization is equivalent to assuming a normal prior distribution for the parameters of the form

$$\theta_j \sim N\left([\hat{\theta}_s]_j, \frac{2}{\lambda}\right),$$

that is, assuming that each of the model parameters is *a priori* normally distributed with a mean equal to the estimated value of corresponding parameter in the source model, and a variance inversely proportional to λ. In this way we can interpret $[\hat{\theta}_s]_j$ as setting our "best guess" for the value of our parameter, and λ as determining how much weight we place on our prior beliefs. Large values of λ lead to small prior variance, and a concentration of probability around our prior guess $[\hat{\theta}_s]_j$, and small values spread probability more diffusely, placing less importance on our prior guess.

I note that this idea of using a prior guess and regularizing a loss for estimation of (high dimensional) parameter vectors is itself certainly not new. In fact, the concept dates back as early as 1961 [81], where in a ground-breaking piece of work the authors propose the first formal shrinkage estimator. The James-Stein procedure was designed to estimate the mean of a multivariate normal, and was shown to uniformly improve on regular least-squares (*i.e.*, equivalent in my setting to using the target data only) by shrinking the estimates towards a reference point (equivalent to the existence of a source model). This is essentially the same idea that underlies my proposal.

The Bayesian connection of my method also offers the possibility for further improvements to

Sourcerer. The authors of [81] also show that the optimal choice of λ is inversely proportional to an unbiased estimate of the Kullback–Leibler divergence between the target only model and the reference (source) model; that is, the more the target only model differs from the reference model, the less weight should be placed on the reference. Though accurate estimation of Kullback–Leibler divergences between neural networks is difficult, a similar idea could potentially be adapted for use in Sourcerer to refine the selection of λ.

Another way of choosing λ, would be in a more formal, and data-driven manner, with a prior distribution placed on λ, and it integrated directly into the posterior distribution. In this manner an appropriate value for λ could be estimated directly from the target data by a straightforward integration into a posterior sampling scheme or a variational Bayes approach, both of which are gaining popularity in the neural network community.

9.2 Experiments

This section presents experiments comparing Sourcerer to the state of the art in semi-supervised DA for the classification of SITS. In the following, a given run was performed by training a model using all of the available source training data and a fixed quantity of target training data. The target data represents a fixed number of polygons, rather than a fixed amount of data. Recall from Section 6.1 that a polygon is a contiguous area with the same land cover—*e.g.*, a farm, forest or residential area—meaning that the number of instances in a polygon can be as few as 7 to well over 1 000. On average, tile T31TDJ has 336 instances per polygon and tile T32ULU has 279 instances per polygon.

Treating the target data in this manner makes the problem more realistic, but also more difficult. More realistic because in practice reference data is collected per site (*i.e.*, per polygon), and not per satellite pixel. More difficult because rather than having an increasing random sample, the data are not distributed across the whole domain and do not represent the accurate class distribution of the area. For example, if an experiment is performed with 10 polygons of target data this will equate to approximately 3 000 training instances, but it will represent at most 10 classes from the target domain and all the instances within one of these classes will be quite similar. A sample of this nature is more difficult to learn from than a sample of the same quantity that is randomly selected across the whole target domain.

The number of polygons used for the experiments was increased according to the following schedule:

$$\textit{no. of polygons: } \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1000, 2000\}$$

Each experiment was repeated five times, and to enable comparison between runs, a linear

interpolation of the test accuracies was applied to give the test accuracy for specific quantities of training data (number of pixels). To enable comparison between methods, the interpolated results from each of the five runs were averaged. All experiments in this Chapter were implemented in PyTorch 1.3.1 [121].

9.2.1 Data

The experiments performed in this Chapter use the SITS data as presented Chapter 6. While I provide a brief summary here, the reader is encouraged refer back to Section 6.1 for more detail.

The data were acquired by the Sentinel-2A satellite between 1 January 2016 and running through to 26 December 2016. The revisit frequency of this individual Sentinel-2 satellite is approximately 10 days and therefore the resulting time series have a length of 37.

The training data are labelled with one of 30 land cover classes:

- Five urban classes: High-density Urban, Low-Density Urban, Industrial, Parking, Roads;
- Fourteen vegetation classes: Rapeseed, Winter Wheat and Barley, Spring Barley, Pea, Soy, Sunflower, Corn, Corn silage, Rice, Beetroot, Potatoes, Grassland, Orchards, Vineyards;
- Seven natural and semi-natural classes: Deciduous Forest, Coniferous Forest, Lawn, Woodlands, Surface Minerals, Beaches and Dunes, Glaciers; and,
- Four *other* classes: Peat, Marshland, Inter-tidal Land, Water.

The experiments were conducted using three Sentinel-2 tiles (110km × 110km) as study areas: one as a source domain and two as target domains. The source domain, tile T31TEL, is located within a highland region known as Massif Central (45.1°N, 2.6°E). The first target domain, tile T31TDJ, is located near the city of Toulouse in south-west France (43.6°N, 1.4°E), while the other, tile T32ULU is located in the north-east near the city of Strasbourg (48.4°N, 7.5°E).

Table 6.2 displays the total number of instances available per domain and per set. The source domain has approximately 12.6 million training instances and the experiments have been conducted under the condition that only a predetermined quantity of target train data is available per experiment. The test set for target domain T31TDJ has 3.4 million instances, while the test set for T32ULU has 5.6 million. The performance of all methods have been measured on the test data, which have been held out during the training process.

A comparison of the land cover classes of the training data for the regions is displayed in Table 6.3. There is an evident difference in the distributions of classes between domains, however the most notable point is that areas of Peat and Marshland both occur in the target domain but not in the source domain. This means that the model cannot learn the profile of these land cover classes from the source domain only.

9.2.2 Experimental Settings

The following section will begin by describing each of the following seven experimental settings that were compared in my experiments:

- Sourcerer;
- 4 baseline configurations: Source Only, Target Only, Naive TempCNN, and Finetuned TempCNN;
- 2 state-of-the-art semi-supervised DA methods: MiniMax Entropy (MME), and Domain-adversarial Neural Networks (DANN).

These configurations are detailed below.

Sourcerer

Sourcerer starts with a TempCNN model (Section 6.2) trained on the source domain data. The weights of this model are used as the initial values for training on the labelled target data with the amount the model is allowed to vary from these values (λ), based on the quantity of labelled target data. I emphasise that this is a significant benefit of Sourcerer—that its prerequisites for use are only to have available a pre-trained model and the labelled target data. That is, the practitioner does not have to be in possession of the source data to apply this method, as opposed to the state-of-the-art methods, where all labelled instances (source and target) are pooled and the model is trained on this pooled data. Therefore methods such as DANN and MME (both presented below) require all of the labelled source data to be available. This feature of Sourcerer can be of significant practical benefit as labelled training data from a Sentinel-2 tile is of the order of hundreds of gigabytes and using Sourcerer means that this only has to be stored and used for training on one occasion.

The value of λ is a function of a single hyperparameter: t_{max} (see Section 9.1.3), which is set to 10^6 for all of these experiments. I believe this to be a reasonable choice as a training set of 10^6 instances provides enough variation to train an accurate model and thus, it is unlikely that restricting the model's learning by regularizing towards the source parameters will be beneficial to the overall accuracy (I note that a sensitivity analysis is provided in Section 9.2.6). Substituting this value into equation 9.6, gives $k = -3.3333$ and the final schedule for our λ values as:

$$\lambda = 10^{10} \cdot \ n_t^{-3.3333} \tag{9.7}$$

Once λ is calculated the model is trained with SourceRegLoss (Equation 9.1) using the Adam optimizer [89]. The number of epochs used for training varies with the quantity of training data such that 5 000 gradient updates have been performed or 1 epoch is completed (see Equation 9.8), as due to the large quantity of training data, little learning occurs beyond this point (and this was shown specifically for TempCNN in [126]).

$$\text{NoEpochs} = \max\left(1, \frac{\text{NoGradUpdates} \cdot \text{BatchSize}}{\text{TargetTrainQty}}\right) \tag{9.8}$$

Baseline Configurations

The following four settings correspond to methods that can map the target domain without using any DA methods, and in doing so, represent various lower bounds for my method to compare to (also see [110] for a discussion of the performance of baseline CNN configurations).

Source Only: This is the baseline configuration in which a model is trained on labelled source data only. This is the simplest setting as it is independent of the amount of labelled target data available, and hence returns only a single value for test accuracy. This configuration sets the lower bound that would be expected for test accuracy when no target data is available and no DA method is applied. For comparative purposes, I have used the TempCNN model, the same categorical cross-entropy loss function, and the same number of training epochs as used for Sourcerer.

Target Only: This is a baseline configuration in which the only labelled data used for training the model is from the target domain. This configuration also acts as a lower bound on the test accuracy for when DA is no longer required, that is, enough target data is available to train an accurate classifier. As per the *Source only* configuration I have used the TempCNN model, categorical cross-entropy loss function and number of training epochs as used for Sourcerer.

Naive TempCNN: This is a baseline configuration in which a TempCNN model is first trained on the labelled source data and then trained on the labelled target data, without applying a particular DA method.

Finetuned TempCNN: This is a baseline configuration where the TempCNN is first trained on the labelled source data, at which point the weights of the convolutional layers are frozen, and only the fully-connected layer(s) and the softmax are finetuned by training on the labelled target data.

This technique is common in transfer learning for computer vision problems [189] as the convolutional layers of a CNN learn general features of data while the fully-connected layers

learn specifics. This configuration allows for comparison as to whether the general features of SITS data can first be learnt from a source domain and then finetuned using the available labelled target data.

State-of-the-art Methods

The two methods presented here—MME and DANN—represented the state of the art in semi-supervised DA at the time Sourcerer was developed[2]. The following details describe their specific implementation for these experiments, for more information on the methods themselves, please see Section 5.3.

I note that a comparison to Encoda is shown in Section 9.2.8 using the DomainNet dataset.

MME: This method [137] is based on training a CNN model using 2 loss functions. It learns a prototype of each class from the labelled data and then minimizes the distance between these prototypes and the unlabelled data, in the process learning discriminating features of the data. It can be implemented on any CNN model, so for comparison purposes I have implemented it using the TempCNN architecture also used in my model, so as to control for model choice. Training occurs in two steps: first, a batch of labelled data (pooled from both source and target domains) is passed forward through the model, a *standard* loss function is calculated and the weights are updated via backpropagation; then, a batch of unlabelled data is passed forward through the model and an entropy loss function is calculated using the output of the convolutional layers of the model. Once trained, unlabelled target data is tested as per a standard CNN, via a forward pass of the model. The MME model was trained using 1 epoch of the labelled training data with a batch size of 32 instances, and the Adam optimizer. This results in greater than 500 000 gradient updates to the model.

DANN: This method [56] is based on maximizing accuracy in predicting the class label of an instance while not being able to tell whether it was from the source or target domain. Learning in this manner discourages the model from learning features that are specific to a domain. To achieve this it uses a CNN model with two output layers—one for the class and one for the domain. For my experiments, each instance has a class label (land cover: 0-29) and a domain label (binary: source/target) and the loss function is the addition of the loss calculated using the class labels and the inverse of the loss calculated using the domain labels. The model learns from unlabelled target instances by using the domain label only (as the class label is unavailable). The architecture used is a TempCNN with two outputs following the convolutional layers, each with a fully connected layer and softmax. Once

[2]The Sourcerer publication was submitted prior to the publication of UODA [133] and APE [88] and therefore these methods are not present in these experiments

trained, unlabelled target data is tested via a forward pass of the model and the class labels are recorded (the domain labels are ignored). The DANN model was trained using 1 epoch of the labelled and unlabelled training data with a batch size of 32 instances, and the Adam optimizer.

Like Sourcerer, each of these methods are deep learning-based, however unlike my method, each of these require the labelled source data to be available to train the model for classifying the target domain. These methods concatenate the labelled source and labelled target data and learn from this pooled dataset. While I note that using the labelled data in this manner creates a different problem (an easier one), these methods have been included to illustrate that my method is competitive in accuracy with the current state of the art (or outperforming them), with the additional benefit of only requiring a model pre-trained on the source domain, not all of the source data. In each case it was attempted to tune the parameters of state-of-the-art method to achieve optimal accuracy for the given method and model architecture.

9.2.3 Overall Accuracy

The following section presents the results of semi-supervised DA experiments performed on two source-target domain pairs:

1. T31TEL (source)–T31TDJ (target); and
2. T31TEL (source)–T32ULU (target).

The results are presented for Sourcerer, the two state-of-the-art methods, as well as the four baseline configurations. In each instance, the overall accuracy is presented for an increasing amount of labelled target training data. The results of the experiments are presented in Tables 9.1 and 9.2.

The following subsections will analyse these results further, then move on to a comparison of the computation time of Sourcerer against the state of the art. It then analyses the per-class accuracy of the methods and this section concludes with a sensitivity analysis of the one hyperparameter of my method t_{max}.

Sourcerer Versus the Baseline Configurations

I begin with a comparison of Sourcerer against the baseline configurations. Figure 9.1 shows the average overall accuracy for Sourcerer against the baseline configurations–Naive TempCNN, Finetuned TempCNN, Target Only and Source Only–for each target tile. It shows that for each quantity of target data available, Sourcerer is either equal to or exceeds the performance of all baseline configurations. This aligns with the intuition of how Sourcerer is designed—for small

Table 9.1: Overall accuracy on target tile T31TDJ for semi-supervised methods and baseline models trained on source tile T31TEL and an increasing quantity of labelled target data.

Target Qty	Naive	Finetuned	Target Only	DANN	MME	Sourcerer
0	**0.7418**	**0.7418**	<0.001	0.6802	0.6671	**0.7418**
20	0.7415	0.6923	0.0061	0.6856	0.6671	**0.7417**
50	0.7373	0.6834	0.0150	0.6838	0.6665	**0.7417**
100	0.7303	0.6830	0.0420	0.6857	0.6689	**0.7418**
250	0.7076	0.6402	0.1085	0.7042	0.6791	**0.7425**
500	0.6719	0.6324	0.1267	0.6906	0.6910	**0.7473**
1000	0.6461	0.6631	0.2208	0.6951	0.7035	**0.7537**
5000	0.7114	0.7373	0.4509	0.7004	0.7202	**0.7538**
10000	0.7518	0.7513	0.5387	0.6955	0.7294	**0.7520**
25000	0.7623	0.7676	0.6337	0.6941	0.7496	**0.7680**
50000	0.7810	**0.7868**	0.7054	0.7104	0.7689	0.7835
100000	0.8059	**0.8063**	0.7534	0.7112	0.7845	0.8043
500000	0.8434	0.8419	0.8266	0.7491	0.8265	**0.8455**
1000000	**0.8507**	0.8504	0.8419	0.7686	0.8378	**0.8507**

Table 9.2: Overall accuracy on target tile T32ULU for semi-supervised methods and baseline models trained on source tile T31TEL and an increasing quantity of labelled target data.

Target Qty	Naive	Finetuned	Target Only	DANN	MME	Sourcerer
0	**0.8747**	**0.8747**	<0.001	0.8411	0.8022	**0.8747**
20	0.8704	0.8340	0.0020	0.8429	0.8022	**0.8747**
50	0.8656	0.7854	0.0065	0.8430	0.8040	**0.8747**
100	0.8582	0.7919	0.0186	0.8396	0.8103	**0.8750**
250	0.8341	0.7950	0.0548	0.8380	0.8133	**0.8751**
500	0.8252	0.8082	0.1046	0.8401	0.8088	**0.8753**
1000	0.8128	0.8232	0.1960	0.8431	0.8141	**0.8753**
5000	0.8432	0.8320	0.5436	0.8311	0.8367	**0.8754**
10000	0.8584	0.8549	0.6646	0.8342	0.8383	**0.8755**
25000	0.8609	0.8590	0.7022	0.8368	0.8417	**0.8776**
50000	0.8607	0.8609	0.7531	0.8376	0.8491	**0.8787**
100000	0.8707	0.8761	0.8108	0.8465	0.8556	**0.8799**
500000	0.8977	**0.8986**	0.8733	0.8509	0.8844	0.8942
1000000	0.9033	0.9037	0.8895	0.8655	0.8919	**0.9054**

quantities of target data, the model parameters will be heavily regularized towards those learned on the source data, and hence returns the same accuracy as Source Only; while for large quantities (where DA is not necessary), the model is allowed to learn from the available data and hence returns the same accuracy as Target Only. The model gradually increases in accuracy between these two extreme situations.

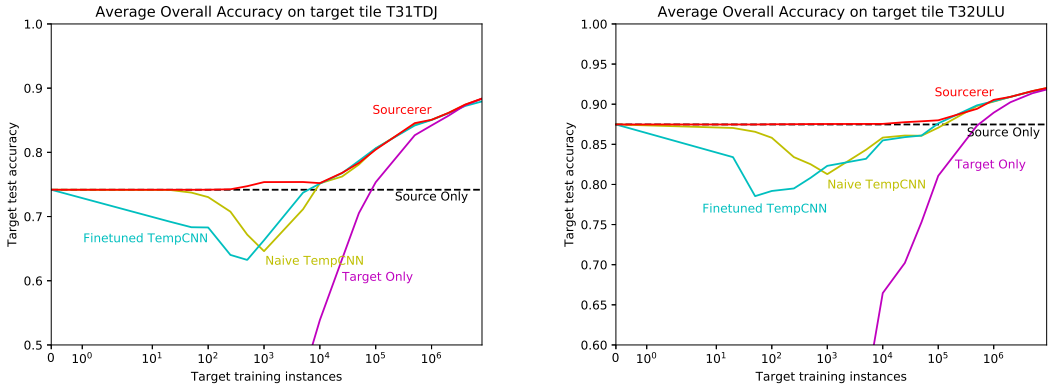
Comparing the performance of Sourcerer on the two tiles, it is evident that the magnitude of its benefit is dependent on the similarity of the source and target domains. For example, when 25 000 labelled target instances are available Sourcerer outperforms Source Only by 2.5% on target tile T31TDJ (74.2% to 76.8%) where the domains are less similar climatically, compared to 0.3% (87.5% to 87.8%) on tile T32ULU, where the domains are more similar.

An interesting result is also present in the Naive TempCNN and Finetuned TempCNN

experiments. In these setting, it was found that the model initially decreases in accuracy when trained only with labelled target data (see Figure 9.1). On target tile T31TDJ, the Source Only achieves test accuracy of 0.744, while the Naive TempCNN dips to as low as 0.646 when 1 000 labelled target instances are available (approximately 3-4 polygons), before increasing again and growing to be more accurate when moderate-to-large quantities of data are available. Similarly, tile T32ULU begins at 0.8750 test accuracy and drops to 0.812 before increasing again. A similar pattern is observed in the Finetuned TempCNN on each target tile.

This dip occurs for two reasons: (1) The available target training data originates from few polygons, and consequently the model overfits the classes present in the target data; and (2) There are some classes present in the target domain that were absent from the source, which significantly shifts the weights of the TempCNN model when they are presented for the first time [110]. These results demonstrate that the convolutions of a TempCNN cannot overcome the domain shift alone and that a semi-supervised DA method like Sourcerer is necessary for optimal accuracy.

When considering the Target Only configuration, it takes well over 1M training instances to reach the performance of Sourcerer for each target tile, thus re-emphasizing the case for an accurate semi-supervised DA method. On target tile T31TDJ, Target Only learns from 100 000 labelled instances before achieving 75% test accuracy, whereas Sourcerer uses only 1 000. On tile T32ULU, Target Only requires 500 000 training instances to achieve the starting accuracy of Sourcerer (87.5%).



(a) (b)

Figure 9.1: Average overall accuracy for Sourcerer against the Baseline configurations, trained on the source domain (T31TEL) and increasing quantities of labelled target data for domains T31TDJ (a) and T32ULU (b).

Sourcerer Versus the State of the Art

I turn now from comparison against baselines to comparison against state of the art: MME and DANN. Figure 9.2 shows the average overall accuracy for Sourcerer against the state-of-the-art

methods–DANN and MME–for each target tile. It is evident from these plots that Sourcerer, produces a higher test accuracy than either DANN or MME, for any given quantity of labelled target data. In fact, when considering tile T31TDJ the best possible accuracy achieved by DANN–76.8% (when training on 1M labelled instances) is achieved by Sourcerer when training on only 25 000 instances. On tile T32ULU, DANN achieves 86.5% accuracy using 1M labelled target instances, which is below the initial test accuracy of Sourcerer (87.5%), that is, without having done any adaptation.

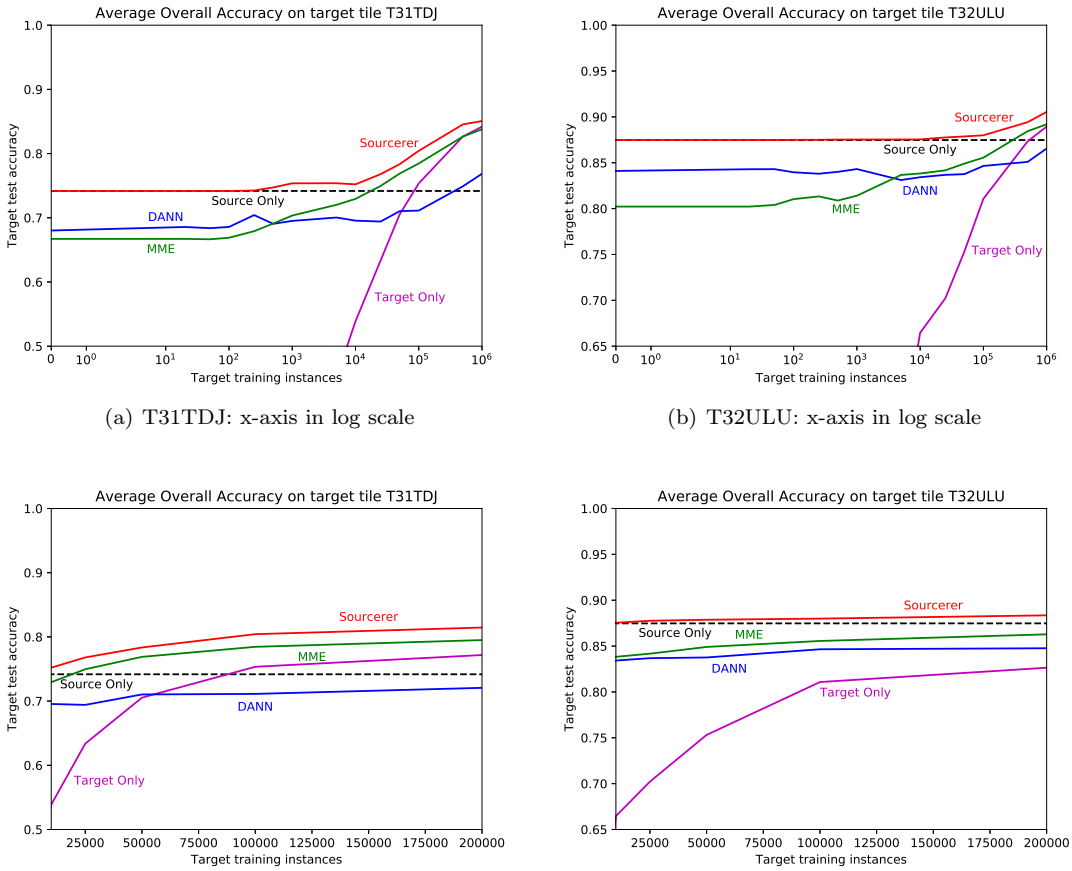
For each experiment, MME starts with the lowest overall accuracy but increases noticeably as more target data become available. When around 1M target instances are available, it produces a test accuracy within 0.5% of Sourcerer, for each target tile. The improvement indicates that the MME method *is* learning the difference between the domains, however it is not learning quick enough for our application purposes, where greater than 1M instances are unlikely to be available.

I reiterate that not only is Sourcerer outperforming each of these methods, but it is doing so in a more convenient manner. Each of MME and DANN use the labelled source data in the training process, whereas once a model is trained on the source data, Sourcerer can use this pre-trained model and the target data to map any target region.

9.2.4 Computation Time

The average time for training Sourcerer and the two state-of-the-art configurations is shown in Table 9.3. The TempCNN model, which is the starting point for Sourcerer, completed training on the source domain data in 156.60 minutes. When training on small quantities of training data, it would take just minutes longer—for example, to train on 1 000 target training instances Sourcerer took only an additional 1.43 minutes on average. This means that when a source model is available to download, a model can be trained on labelled target data using Sourcerer in minutes.

Interestingly, the models using DANN took the longest time to train when only source data was available, but scarcely increased (and even declined on average) in training time as more labelled target data was used. This is because each mini-batch used in DANN matches source data with either labelled or unlabelled target data. That is, each mini-batch of 32 instances is comprised of 16 labelled source instances and 16 target instances (either labelled or unlabelled). Consequently, as the source data is far larger than the target in quantity, the training time is a function of the amount of source data available. MME was generally the slowest to train due to the cost of the entropy calculations and performing two gradient updates for each mini-batch. The high computation cost is expected as this method is designed to maximise accuracy with very small samples of target data with even class distributions (1-3 instances per class), rather than hundreds of instances from few classes (as in our case).



(a) T31TDJ: x-axis in log scale

(b) T32ULU: x-axis in log scale

(c) T31TDJ: x-axis in linear scale

(d) T32ULU: x-axis in linear scale

Figure 9.2: Average overall accuracy for Sourcerer against the state-of-the-art methods, DANN and MME, and 2 baseline configurations. Models were trained on the source domain (T31TEL) and increasing quantities of labelled target data. Results are show for target domains T31TDJ in (a)&(c) and T32ULU in (b)&(d).

The models trained via each method had comparable testing time as the classification process for each is the same—one forward pass of the trained model. In my experiments, this took approximately 22 minutes on average for target tile T31TDJ (3.4M instances), and 35 minutes on average for target tile T32ULU (5.6M instances).

9.2.5 F1-score Accuracy

The above comparisons all centre around overall classification accuracy to compare methods, and while this is the most commonly used accuracy measure, it is far from the only one. In this section, I present another accuracy measure, the F1-score, to analyse the results of the experiments. The average F1-score is calculated by:

Table 9.3: The average training time (minutes) for Naive TempCNN, Sourcerer, DANN, and MME, using 12M source instances (Tile T31TEL) and increasing quantities of target data.

Target Qty	Naive TempCNN	DANN	MME	Sourcerer
0 (source only)	156.6	352.2	310.9	156.6
100	157.0	350.9	315.1	157.1
1000	157.1	369.9	383.6	158.0
10000	159.2	364.9	535.9	162.0
100000	183.6	368.2	1301.9	185.8
1000000	262.8	365.3	5737.8	268.0

$$F1 = \frac{1}{|classes|} \sum_{x \in classes} 2 \frac{precision_x \cdot recall_x}{precision_x + recall_x}$$

The F1-scores for Sourcerer, the two state of the art, and the three baseline configurations that use target domain data are shown in Tables 9.4 and 9.5.

Table 9.4: F1-score on target tile T31TDJ for semi-supervised methods and baseline models trained on source tile T31TEL and an increasing quantity of labelled target data.

Target Qty	Naive	Finetuned	Target Only	DANN	MME	Sourcerer
0	**0.4183**	**0.4183**	<0.001	0.3690	0.3364	**0.4183**
20	**0.4279**	0.3736	0.0006	0.3700	0.3364	0.4273
50	0.4246	0.3661	0.0016	0.3678	0.3391	**0.4282**
100	0.4163	0.3647	0.0034	0.3668	0.3462	**0.4184**
250	0.3926	0.3228	0.0092	0.3698	0.3499	**0.4193**
500	0.3845	0.3240	0.0231	0.3738	0.3704	**0.4283**
1000	0.3650	0.3445	0.0614	0.3754	0.3805	**0.4336**
5000	0.4064	0.4143	0.2014	0.3651	0.4094	**0.4418**
10000	0.4238	0.4191	0.2357	0.3660	0.4166	**0.4307**
25000	0.4607	0.4652	0.3199	0.3684	0.4414	**0.4695**
50000	0.4707	**0.4757**	0.3912	0.3811	0.4756	0.4756
100000	0.4920	0.4922	0.4360	0.3803	0.4841	**0.4936**
500000	0.5559	0.5468	0.5345	0.4324	0.5495	**0.5588**
1000000	0.5707	0.5686	0.5612	0.4564	0.5471	**0.5760**

In general, Sourcerer outperforms all of the baselines methods and the state of the art in terms of F1-score accuracy. However the F1-scores are significantly lower than the overall accuracy values for a given experiment and method—eg. when training on 1 000 labelled T31TDJ instances, Sourcerer achieves an overall accuracy of 0.7520 but only an F1-score of 0.4307. This indicates that the methods are poorly predicting some classes in the target domain.

A review of the per-class F1-scores indicates that Sourcerer performs less accurately on the classes where the distributions differ between the source and target domains. For example, Coniferous Forest covers 32.42% of the source domain but only 8.63% of target domain T31TDJ (see Table 6.3). Table 9.6 shows the Coniferous Forest F1-score for Sourcerer and the state of the

Table 9.5: F1-score on target tile T32ULU for semi-supervised methods and baseline models trained on source tile T31TEL and an increasing quantity of labelled target data.

Target Qty	**Naive**	**Finetuned**	**Target Only**	**DANN**	**MME**	**Sourcerer**
0	**0.5897**	**0.5897**	<0.001	0.5064	0.5151	**0.5897**
20	0.5824	0.5507	0.0006	0.5012	0.5151	**0.5897**
50	0.5768	0.4650	0.0013	0.5003	0.5186	**0.5897**
100	0.5708	0.4783	0.0028	0.4949	0.5164	**0.5896**
250	0.5470	0.5041	0.0084	0.4815	0.5087	**0.5891**
500	0.5228	0.4953	0.0251	0.4955	0.4958	**0.5884**
1000	0.4838	0.5098	0.0555	0.5010	0.5129	**0.5867**
5000	0.5298	0.5144	0.2019	0.4961	0.5276	**0.5681**
10000	0.5359	0.5423	0.2778	0.4929	0.5462	**0.5691**
25000	0.5417	0.5450	0.3396	0.5040	0.5362	**0.5503**
50000	0.5574	0.5612	0.3963	0.5005	0.5514	**0.5652**
100000	0.5764	**0.5829**	0.4373	0.4998	0.5539	0.5821
500000	0.6449	0.6493	0.5877	0.5114	0.6301	**0.6494**
1000000	0.6605	0.6614	0.6190	0.5374	0.6435	**0.6632**

art for one shuffle of the labelled training data from target tile T31TDJ.

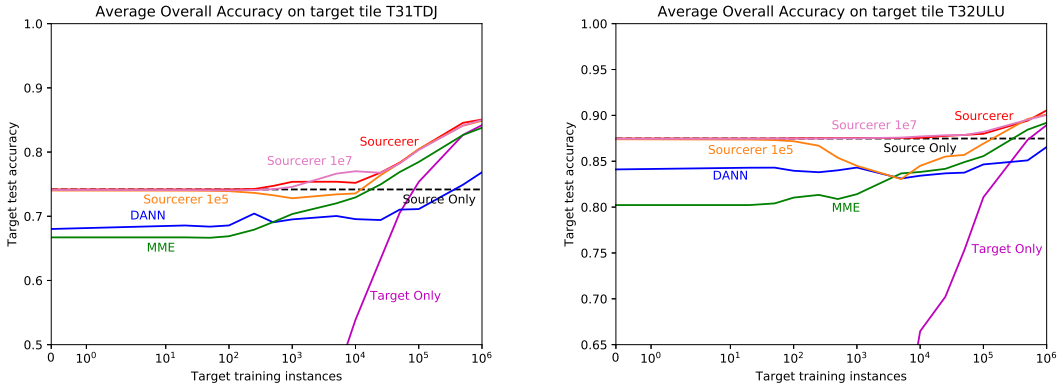
Table 9.6: F1-score for Coniferous Forest class for a model trained using Sourcerer, DANN, and MME. These results represent one shuffle of the labelled training data from target tile T31TDJ, while the source model has been trained on tile T31TEL.

Target Domain			**F1-score (Conif. Forest)**		
Labelled Polygons	Labelled Quantity	Conif. Forest Quantity	MME	DANN	Sourcerer
1	297	297	**0.0390**	0.0001	0.0006
2	394	297	**0.0252**	<0.0001	0.0005
4	644	297	**0.0230**	0.0001	0.0002
8	1198	297	**0.0154**	<0.0001	0.0003
16	1786	297	**0.1230**	<0.0001	0.0198
32	4836	297	**0.1178**	<0.0001	0.0427
64	29195	10884	**0.0450**	0.0001	0.0087
128	42504	10884	**0.0451**	0.0002	0.0012
256	73688	12498	**0.0992**	0.0001	0.0714
512	172746	47467	**0.0875**	0.0005	0.0415
1000	283090	53251	0.2762	0.0001	**0.3482**
2000	546778	82507	0.3020	0.0002	**0.3623**
4000	1094517	152598	0.3097	0.0002	**0.4102**

These results highlight a limitation of both my technique and the state of the art, as all methods perform poorly on this class. This issue is not specific to this problem however, as learning from unbalanced data is a large area of research in machine learning (for example, see [92]). One note that can be taken away from these results is that the choice of source domain is important for optimal results when using Sourcerer.

9.2.6 Hyperparameter Sensitivity Analysis

Sourcerer has only one user-defined hyperparameter, t_{max}, which represents the quantity of labelled target data at which the regularization applied to the model approaches zero (as discussed in Section 9.1.3)— it represents the quantity of target data at which we would no longer require source data and DA to learn an accurate model. I have performed experiments on each target tile with three different values of t_{max}—10^5, 10^6 (the default value), and 10^7. Figure 9.3 shows the average overall accuracy for the three models for each target tile.



(a) (b)

Figure 9.3: Average overall accuracy for Sourcerer with different values for the hyperparameter t_{max}, trained on the source domain (T31TEL) and increasing quantities of labelled target data for domains T31TDJ (a) and T32ULU (b).

On tile T31TDJ, each of the three models of Sourcerer outperform the state of the art for all quantities of labelled target data. This is also the case for tile T32ULU for models with t_{max} of 10^6 and 10^7. The model with t_{max} of 10^5 does dip below the performance of MME when around 8 000 target instances are used. This *dip* in performance is the same as that displayed by the Naive TempCNN in Section 9.2.3, and indicates that a value of 10^5 for the t_{max} does not regularize the model sufficiently.

The results for the other two models show that the choice of t_{max} being either 10^6 or 10^7 will produce similar performance, and thus any attempt to optimize this value further is not likely to be necessary.

9.2.7 Visual Analysis of Results

In this section I will illustrate what the differences in overall accuracy mean for the resulting land cover maps. Figure 9.4 shows two land cover maps produced by using DANN and Sourcerer and trained on 64 labelled target polygons (approximately 12 000 instances); in comparison with the ground truth polygons from the test data. When comparing the maps of the two methods, there

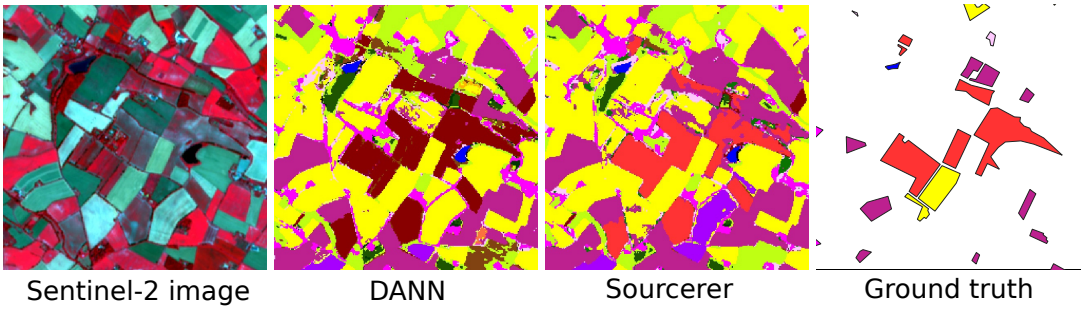


Figure 9.4: A false-color Sentinel-2 image, land cover maps produced using DANN and Sourcerer and the ground truth land cover classes. Maps were created with 64 polygons of labelled target data available (approximately 12 000 instances). Legend provided in Table 9.7.

is disagreement between large areas of agricultural land with the DANN-based model classifying large amounts of corn where Sourcerer classified soy. As soy and corn are both winter crops, their spectral profiles appear similar and an accurate classifier is required to separate them correctly. In this case, we can see from the test data that the correct land cover for these polygons are soy as predicted by Sourcerer.

When more data are available the differences between maps produced are more subtle, however still present. Figure 9.5 shows land cover maps produced by training on 512 labelled target polygons (approximately 99 000 instances) using MME and Sourcerer as well as a Sentinel-2 false color image and the ground truth. If we compare the results of each method of classifying the rapeseed crop (crop A in the ground truth subfigure), it can be seen that MME correctly classifies few pixels of this crop while in comparison, Sourcerer accurately classifies almost the whole crop. When we consider the corn crop (B in the ground truth subfigure) located just below the image's center, the MME-based model classifies approximately half of this crop as corn silage, while Sourcerer classifies almost the complete polygon correctly.

Table 9.7: Legend of land cover classes for Figures 9.4 and 9.5 (only classes present in the data shown)

Color	Class	Color	Class	Color	Class
	Urban (high density)		Soy		Deciduous forest
	Urban (low density)		Sunflower		Coniferous forest
	Industrial		Corn		Lawn
	Parking		Corn silage		Woodlands
	Road		Beetroot		Minerals
	Rapeseed		Potatoes		Peat
	Wheat & Barley		Grassland		Marshland
	Barley (spring)		Orchards		Water
	Peas		Vineyards		

9.2.8 Comparison Against Encoda

Sourcerer was specifically designed to perform well on the classification of SITS, in particular when the labelled target data is randomly collected/distributed, including the scenario where not

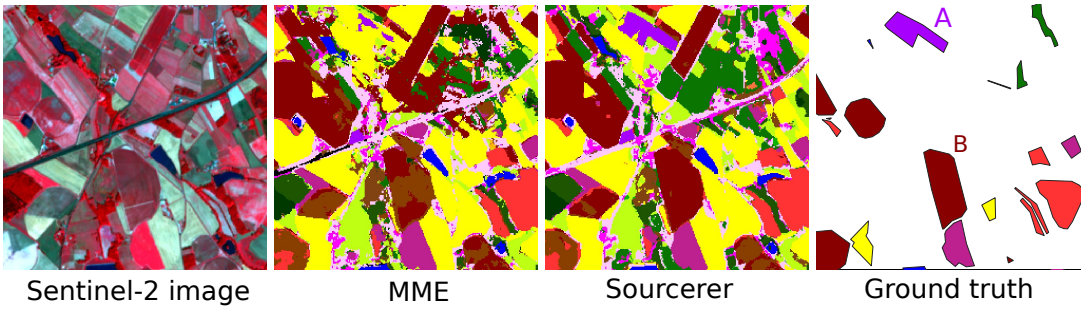


Figure 9.5: A false-color Sentinel-2 image, land cover maps produced using MME and Sourcerer and the ground truth land cover classes. Maps were created with 512 polygons of labelled target data available (approximately 99 000 instances). Legend provided in Table 9.7.

all classes are present in the target data. Conversely, Encoda (presented in Chapter 8) is designed to be a general semi-supervised DA method with the labelled target data evenly distributed between the classes. While these differences are notable, it is an interesting experiment to observe how Sourcerer performs on the DomainNet dataset against Encoda.

To ensure an even comparison, a ResNet34 model was also used as the backbone architecture for the Sourcerer experiments. The largest labelled dataset in DomainNet has approximately 60 000 instances, I therefore set Sourcerer's hyperparameter t_{max} to 50 000. This means that if we were to have this quantity of target data available, Sourcerer would place full trust in the target data and there would be very little regularisation applied when learning. A value of t_{max} to 50 000, equates to a *lambda* value of 0.0315.

The results comparing Sourcerer to Encoda are shown in Table 9.8. The results demonstrate that Sourcerer does not learn much from the labelled target data as it is far below the performance of Encoda in this application.

Table 9.8: Overall Accuracy on DomainNet dataset for Encoda against Sourcerer.

Method	C to S	P to C	P to R	R to C	R to P	R to S	S to P	MEAN
Sourcerer	55.8	60.9	74.8	61	63.8	53.5	59.9	61.4
Encoda	64.1	73.5	80.6	73.6	71.2	66.1	68.5	71.1

These results are not surprising as the experimental design presented here is notably different to the conditions under which Sourcerer was developed. As noted, Sourcerer is based around the situation where labelled target data is collected at random and hence, unevenly distributed between the classes (as this is often how it is collected in remote sensing). Conversely in these experiments, labelled target data are available for every class, so the ability of Sourcerer to be robust to rare or missing classes in the target domain data is of no benefit.

9.3 Conclusion

Land cover maps can be produced automatically by modern machine learning, however these techniques require substantial training data to achieve high levels of accuracy, which are not always available. One technique researchers use when labelled training data are scarce is DA—where data from an alternate region, known as the source domain, are used to assist the training of a classifier to map the study region, or target domain. The scenario addressed in this chapter is known as semi-supervised DA, where there are some labelled samples available in the target domain, but not sufficiently many to train a classifier.

This chapter presented a novel semi-supervised DA technique for producing land cover maps from SITS data, named Sourcerer. The Bayesian-inspired technique takes a CNN trained on a source domain and treats this as a prior distribution for the weights of the model, with the degree to which the model is modified to fit the target domain limited by the quantity of labelled target data available.

The experiments section demonstrate the performance of Sourcerer on Sentinel-2 time series images, where it outperforms all other methods for any quantity of labelled target data available on two different source-target domain pairings. On the more difficult target domain, the starting accuracy (when no labelled target data are available) of Sourcerer is 74.2%, and this is greater than the next-best state-of-the-art method when trained on 20 000 labelled target instances.

Sourcerer's high accuracy is also complemented by its straight-forward manner of application as it only requires a model pre-trained on the source domain, rather than all of the source data. In this case, a model can be trained on 10 000 labelled target instances using Sourcerer in under six minutes. This offers great promise to efficiently map resource-poor areas as the practitioner only has to download a model, not millions of instances of source domain data, and spend only a very short time training on the target domain data.

In the future, I would like to see how Sourcerer performs in other DA contexts, in particular temporal DA, used to update maps with recently-acquired data. I would also like to experiment with using Sourcerer for heterogenous DA, where domains have different resolutions or different modes of acquisition.

I would also like to compare Sourcerer to the newer semi-supervised DA methods—UODA, A.P.E., and DeCoTa (discussed in Section 5.3). At present, none of these methods have been applied to time series, however as they are highly successful on image data there is potential that they will also be successful on time series DA problems.

9.3.1 Contributions

The contributions resulting from this chapter of my Ph.D research and the development of Sourcerer are as follows:

1. A novel semi-supervised DA method for producing land cover maps from SITS data;
2. A semi-supervised DA method with state-of-the-art performance, demonstrated on two separate Sentinel-2 source-target domain pairings;
3. A semi-supervised DA method emphasising a user-friendly implementation, as it has only one hyperparameter and can be applied to a pre-trained model and does not require the user to possess the source domain training data.
4. A semi-supervised DA method that still learns effectively when not all classes are present in the labelled target data.

Conclusion

10

Research Summary

Land cover maps are a spatial representation of the biophysical and anthropogenic coverage of the Earth's surface. Currently, the most accurate large-scale land cover maps are produced automatically by applying machine learning algorithms to SITS data. However, this application is not always straight-forward and consequently presents many potential areas of interest for researchers. My Ph.D. research presented in this thesis investigates two of these research scenarios: (1) Finding a scalable time series classifier; and (2) Training a classifier when labelled data are scarce.

In Part I, I presented the field of time series classification and explained how the existing state-of-the-art classifiers cannot scale to the magnitude required to learn from the SITS dataset. This provided the motivation for the development of Proximity Forest, the first scalable and accurate time series classification algorithm. Proximity Forest is an ensemble model of decision tree classifiers that use a novel time series-specific splitting criterion at the node. My experiments showed that Proximity Forest can learn from a SITS dataset of size 1 million in 17 hours, compared to the more than 200 years that would be required by the state of the art. While the work is

motivated by scalable time series applications, my experiments also demonstrate that Proximity Forest is not significantly worse than the best existing classifiers on the benchmark UCR archive of datasets. Proximity Forest proved to be one of the most accurate classifiers while being significantly faster and therefore represents the new state of the art for time series classification. In the next chapter I looked at deep learning for time series classification, with a particular focus on an architecture called InceptionTime. Experiments proved it to be the first deep learning model competitive with the state of the art in accuracy on 'small' time series problems. Deep learning has subsequently become the state of the art for land cover mapping from SITS and time series-based Earth observation tasks.

Modern machine learning methods require large quantities of labelled training data to obtain optimal accuracy; however these quantities of data are not always available for a given application. In Part II of this thesis, I addressed this issue in the context of the land cover classification using SITS data, *i.e.*, how to produce highly accurate land cover maps without a substantial quantity of labelled data. My research in this area lead to the creation of two semi-supervised DA methods. The first, Encoda, is an ensemble of deep learning models trained with co-training, a tool borrowed from the related field of semi-supervised learning. My experiments show that Encoda is competitive with state-of-the-art techniques on the DomainNet dataset, the benchmark in DA for computer vision, and has a wide range of potential applications, including many beyond Earth observation. Encoda is designed around the scenario that the sample has an even distribution across all classes; conversely, the second technique presented—Sourcerer—is designed for use when the data have a random class distribution across the target domain. Sourcerer is a Bayesian-inspired deep learning-based method specifically designed for land cover mapping from SITS. The technique uses a model trained on a source domain as a prior distribution for the model parameters and adjusts the degree of information learned about the target domain according to the quantity of labelled target data available. My experiments showed Sourcerer to outperform two other state-of-the-art deep learning-based methods on two separate source-target domain pairings. Both methods emphasise a straight-forward manner of application as they each only require only models pre-trained on the source domain, rather than for all of the source data to be available to the user.

10.1 Original Contributions to Knowledge

The specific contributions to knowledge resulting from the research presented in this thesis are:

1. Identifying the need for scalable time series classifiers and inspiring substantial subsequent research in the area;

2. Proximity Forest—The first time series classifier with state-of-the-art accuracy while also

being capable of learning from millions of time series in a reasonable time frame;

3. Co-authoring the publication of InceptionTime—the first successful deep learning architecture for time series classification—specifically, devising the experiments investigating receptive fields for time series CNN;

4. Encoda—A novel semi-supervised DA method with state-of-the-art performance on the benchmark DomainNet dataset;

5. Sourcerer—A novel semi-supervised DA method for producing land cover maps from SITS data, emphasising a user-friendly implementation and the ability to learn effectively when not all classes are present in the labelled target data.

Three of these pieces of work—Proximity Forest, InceptionTime, and Sourcerer—have been recognised with publication in leading academic journals.

My research also has an impact across a broader scales. Proximity Forest has raised awareness of the need for scalable time series classifiers and has lead the path for multiple subsequent publications in the area [150, 50, 40]. The work also highlights the benefits of using time series-specific algorithms when studying SITS data, a practice that traditionally has not occurred. The semi-supervised DA methods demonstrate that it is still possible to produce highly accurate land cover maps in areas where resources are scarce. Both methods also draw links between DA and other areas of machine learning—Sourcerer with Bayesian inference and Encoda with semi-supervised learning—potentially giving the research problem a greater audience.

10.2 Limitations

Despite the contributions presented in this thesis, my work does have limitations, which should be noted.

1. SITS data requires substantial preprocessing including cloud detection, atmospheric correction, resampling, and normalisation. Thus, despite being freely available to download, their use requires substantial expertise, which will be outside the resources of some teams who would otherwise benefit from their use.

2. PF requires all of the data to be loaded into main memory to build each tree, meaning that there *is* a limit on the amount of training data it can learn from. Deep learning models do not have this limitation as they learn in mini-batches, which can be loaded as they are required.

3. Sourcerer does not return high accuracy on classes where there is a significant change in distribution between the source and target domains. This problem is not specific to our technique however, as learning models with unbalanced data is an active research area in machine learning [92].

4. Encoda requires specific collection of data as it is based on having examples from each class present in the labelled data.

5. Both Sourcerer and Encoda are semi-supervised methods meaning that they use labelled data from the target domain. This means that *some* resources are still required to collect and label data, which may not be available in some situations.

10.3 Future Work

In the future, one aspect of research that I would like to explore is to modify Sourcerer to actively learn its regularisation parameter from the data. Ideally, the model would be able to recognise how similar the distribution of target data is to that of the source, and thus learn the weight of trust to place on the data from the source domain. One way this could possibly be achieved is by incorporating an expectation-maximisation process into the algorithm.

A second stream of DA research I would like to investigate is to help facilitate a better choice of source domain by users by testing the performance of different source-target pairings based on various attributes of the locations.

In the future I intend to apply Encoda to a SITS data classification problem to compare it to both Sourcerer and the newer semi-supervised DA methods. If Encoda is successful, it could mean that high accuracy maps of any location could be produced with only a small number of examples from each land cover class.

Finally, as discussed in Chapter 7, I would like to extend my exploratory work on unsupervised DA to find a technique that will produce accurate land cover maps of areas where no labelled data are available. This would mean that producing maps of any location would now be possible, representing a truly monumental accomplishment in the field.

Bibliography

[1] Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., Tuytelaars, T.: Memory aware synapses: Learning what (not) to forget. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 139–154 (2018)

[2] Ao, S., Li, X., Ling, C.X.: Fast generalized distillation for semi-supervised domain adaptation. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI'17, p. 1719–1725. AAAI Press (2017)

[3] Arabameri, A., Pourghasemi, H.R.: Spatial modeling of gully erosion using linear and quadratic discriminant analyses in gis and r. In: H.R. Pourghasemi, C. Gokceoglu (eds.) Spatial Modeling in GIS and R for Earth and Environmental Sciences, pp. 299 – 321. Elsevier (2019). DOI https://doi.org/10.1016/B978-0-12-815226-3.00013-2

[4] Armsworth, P.R., Daily, G.C., Kareiva, P., Sanchirico, J.N.: Land market feedbacks can undermine biodiversity conservation. Proceedings of the National Academy of Sciences **103**(14), 5403–5408 (2006)

[5] Asner, G.P., Knapp, D.E., Broadbent, E.N., Oliveira, P.J.C.: Selective logging in the Brazilian Amazon. Science **310**(5747), 480–482 (2005)

[6] Bagnall, A., Flynn, M., Large, J., Lines, J., Middlehurst, M.: On the usage and performance of the hierarchical vote collective of transformation-based ensembles version 1.0 (hive-cote v1.0). In: V. Lemaire, S. Malinowski, A. Bagnall, T. Guyet, R. Tavenard, G. Ifrim (eds.) Advanced Analytics and Learning on Temporal Data, pp. 3–18. Springer International Publishing, Cham (2020)

[7] Bagnall, A., Lines, J., Bostrom, A., Large, J., Keogh, E.: The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Mining and Knowledge Discovery **31**(3), 606–660 (2017)

[8] Bagnall, A., Lines, J., Hills, J., Bostrom, A.: Time-series classification with COTE: the collective of transformation-based ensembles. IEEE Transactions on Knowledge and Data Engineering **27**(9), 2522–2535 (2015)

[9] Bahirat, K., Bovolo, F., Bruzzone, L., Chaudhuri, S.: A novel domain adaptation bayesian classifier for updating land-cover maps with class differences in source and target domains. IEEE Transactions on Geoscience and Remote Sensing **50**(7), 2810–2826 (2012). DOI 10.1109/TGRS.2011.2174154

[10] Bailly, S., Giordano, S., Landrieu, L., Chehata, N.: Crop-rotation structured classification using multi-source Sentinel images and LPIS for crop type mapping. In: IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium, pp. 1950–1953 (2018)

[11] Balakrishnan, S., Madigan, D.: Decision trees for functional variables. In: IEEE International Conference on Data Mining 2006), pp. 798–802 (2006)

[12] Banerjee, B., Bovolo, F., Bhattacharya, A., Bruzzone, L., Chaudhuri, S., Buddhiraju, K.M.: A novel graph-matching-based approach for domain adaptation in classification of remote sensing image pair. IEEE Transactions on Geoscience and Remote Sensing **53**(7), 4045–4062 (2015). DOI 10.1109/TGRS.2015.2389520

[13] Bejiga, M.B., Melgani, F., Beraldini, P.: Domain adversarial neural networks for large-scale land cover classification. Remote Sensing **11**(10) (2019). DOI 10.3390/rs11101153

[14] Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence **35**(8), 1798–1828 (2013)

[15] Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. (special issue on dynamic recurrent neural networks). IEEE Transactions on Neural Networks **5**(2) (1994)

[16] Bhandari, A., Kumar, A., Singh, G.: Feature extraction using normalized difference vegetation index (ndvi): A case study of jabalpur city. Procedia Technology **6**, 612 – 621 (2012). DOI https://doi.org/10.1016/j.protcy.2012.10.074. 2nd International Conference on Communication, Computing and Security [ICCCS-2012]

[17] Blum, A., Mitchell, T.: Combining labeled and unlabeled data with co-training. In: Proceedings of the eleventh annual conference on computational learning theory, COLT' 98, pp. 92–100. ACM (1998)

[18] Bojinski, S., Verstraete, M., Peterson, T.C., Richter, C., Simmons, A., Zemp, M.: The concept of essential climate variables in support of climate research, applications, and policy. Bulletin of the American Meteorological Society **95**(9), 1431–1443 (2014)

[19] Bossard, M., Feranec, J., Otahel', J.: Corine land cover technical guide. Tech. rep., European Environment Agency, Copenhagen, Denmark (2000)

[20] Breiman, L.: Random forests. Machine Learning **45**(1), 5–32 (2001)

[21] Brown, D.G., Johnson, K.M., Loveland, T.R., Theobald, D.M.: Rural land-use trends in the conterminous united states, 1950–2000. Ecological Applications **15**(6), 1851–1863 (2005)

[22] Cantelaube, P., Carles, M.: Le registre parcellaire graphique : des données géographiques pour décrire la couverture du sol agricole. Cahier des Techniques de l'INRA pp. 58–64 (2015)

[23] Cao, Y., Geddes, T., Yang, J., Yang, P.: Ensemble deep learning in bioinformatics. Nature Machine Intelligence **2**, 1–9 (2020). DOI 10.1038/s42256-020-0217-y

[24] Chapelle, O., Scholkopf, B., Zien, A.: Semi-supervised learning. Adaptive computation and machine learning. The MIT Press (2019)

[25] Chen, L., Ng, R.: On the marriage of lp-norms and edit distance. In: Proceedings of the Thirtieth international conference on Very large data bases-Volume 30, pp. 792–803. VLDB Endowment (2004)

[26] Chen, L., Özsu, M.T., Oria, V.: Robust and fast similarity search for moving object trajectories. In: Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pp. 491–502. ACM (2005)

[27] Chen, M., Weinberger, K.Q., Blitzer, J.: Co-training for domain adaptation. In: J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, K.Q. Weinberger (eds.) Advances in Neural Information Processing Systems, vol. 24. Curran Associates, Inc. (2011)

[28] Chen, Y., Keogh, E., Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G.: The UCR time series classification archive (2015). www.cs.ucr.edu/~eamonn/time_series_data/

[29] Chollet, F., et al.: Keras. https://keras.io (2015)

[30] Choudhary, K., Kupriyanov, A.: 2019 land cover map of southeast asia at 30 m spatial resolution with changes since 2010.(report). Optical Memory and Neural Networks **29**(3), 257 (2020)

[31] Courty, N., Flamary, R., Tuia, D., Corpetti, T.: Optimal transport for data fusion in remote sensing. In: 2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), pp. 3571–3574 (2016). DOI 10.1109/IGARSS.2016.7729925

[32] Courty, N., Flamary, R., Tuia, D., Rakotomamonjy, A.: Optimal transport for domain adaptation. arXiv preprint (2015). URL http://arxiv.org/abs/1507.00504

[33] Cui, Z., Chen, W., Chen, Y.: Multi-scale convolutional neural networks for time series classification. arXiv preprint (2016). URL http://arxiv.org/abs/1603.06995

[34] Dalessandro, B., Chen, D., Raeder, T., Perlich, C., Han Williams, M., Provost, F.: Scalable hands-free transfer learning for online advertising. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1573–1582 (2014)

[35] Damodaran, B.B., Flamary, R., Seguy, V., Courty, N.: An entropic optimal transport loss for learning deep neural networks under label noise in remote sensing images. Computer Vision and Image Understanding **191**, 102,863 (2020)

[36] Damodaran, B.B., Kellenberger, B., Flamary, R., Tuia, D., Courty, N.: Deepjdot: Deep joint distribution optimal transport for unsupervised domain adaptation. In: Computer Vision – ECCV 2018, *Lecture Notes in Computer Science*, vol. 11208, pp. 467–483. Springer International Publishing, Cham (2018)

[37] Dau, H.A., Keogh, E., Kamgar, K., Yeh, C.C.M., Zhu, Y., Gharghabi, S., Ratanamahatana, C.A., Yanping, Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G., Hexagon-ML: The UCR time series classification archive (2018). URL https://www.cs.ucr.edu/~eamonn/time_series_data_2018/

[38] Defourny, P., Bontemps, S., Bellemans, N., Cara, C., Dedieu, G., Guzzonato, E., Hagolle, O., Inglada, J., Nicola, L., Rabaute, T., et al.: Near real-time agriculture monitoring at national scale at parcel resolution: Performance assessment of the sen2-agri automated system in various cropping systems around the world. Remote sensing of environment **221**, 551–568 (2019)

[39] Demir, B., Bovolo, F., Bruzzone, L.: Updating land-cover maps by classification of image time series: A novel change-detection-driven transfer learning approach. IEEE Transactions on Geoscience and Remote Sensing **51**(1), 300–312 (2013)

[40] Dempster, A., Petitjean, F., Webb, G.I.: Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. Data mining and knowledge discovery **34**(5), 1454–1495 (2020)

[41] Demšar, J.: Statistical comparisons of classifiers over multiple data sets. Journal of Machine learning research **7**(Jan), 1–30 (2006)

[42] Deng, H., Runger, G., Tuv, E., Vladimir, M.: A time series forest for classification and feature extraction. Information Sciences **239**, 142–153 (2013)

[43] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255. IEEE (2009)

[44] Donahue, J., Hoffman, J., Rodner, E., Saenko, K., Darrell, T.: Semi-supervised domain adaptation with instance constraints. In: 2013 IEEE Conference on Computer Vision and Pattern Recognition, pp. 668–675 (2013). DOI 10.1109/CVPR.2013.92

[45] Douzal-Chouakria, A., Amblard, C.: Classification trees for time series. Pattern Recognition **45**(3), 1076–1091 (2012)

[46] van Engelen, J.E., Hoos, H.H.: A survey on semi-supervised learning. Machine learning **109**(2), 373–440 (2019)

[47] European Space Agency: Sentinel-2 overview. https://sentinel.esa.int/web/sentinel/about-sentinel-online, accessed 2019-03-10

[48] Fawaz, H.I., Forestier, G., Weber, J., Idoumghar, L., Muller, P.: Data augmentation using synthetic data for time series classification with deep residual networks. arXiv preprint **abs/1808.02455** (2018). URL http://arxiv.org/abs/1808.02455

[49] Fawaz, H.I., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A.: Deep learning for time series classification: a review. Data mining and knowledge discovery **33**(4), 917–963 (2019)

[50] Fawaz, H.I., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D.F., Weber, J., Webb, G.I., Idoumghar, L., Muller, P.A., Petitjean, F.: Inceptiontime: Finding alexnet for time series classification. Data Mining and Knowledge Discovery **36**(6), 1936–1962 (2020). DOI 10.1007/s10618-020-00710-y

[51] Fernando, B., Habrard, A., Sebban, M., Tuytelaars, T.: Unsupervised visual domain adaptation using subspace alignment. In: 2013 IEEE International Conference on Computer Vision, pp. 2960–2967 (2013). DOI 10.1109/ICCV.2013.368

[52] Findell, K.L., Berg, A., Gentine, P., Krasting, J.P., Lintner, B.R., Malyshev, S., Santanello, J.A., Shevliakova, E.: The impact of anthropogenic land use and land cover change on regional climate extremes. Nature communications **8**(1), 1–10 (2017)

[53] Foley, J.A., DeFries, R., Asner, G.P., Barford, C., Bonan, G., Carpenter, S.R., Chapin, F.S., Coe, M.T., Daily, G.C., Gibbs, H.K., et al.: Global consequences of land use. science **309**(5734), 570–574 (2005)

[54] Forestier, G., Petitjean, F., Dau, H.A., Webb, G.I., Keogh, E.: Generating synthetic time series to augment sparse datasets. In: 2017 IEEE International Conference on Data Mining (ICDM), pp. 865–870 (2017). DOI 10.1109/ICDM.2017.106

[55] Frenay, B., Verleysen, M.: Classification in the presence of label noise: A survey. IEEE Transactions on Neural Networks and Learning Systems **25**(5), 845–869 (2014). DOI 10.1109/TNNLS.2013.2292894

[56] Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., Lempitsky, V.: Domain-adversarial training of neural networks. The Journal of Machine Learning Research **17**(1), 2096–2030 (2016)

[57] Gessesse, A.A., Melesse, A.M.: Temporal relationships between time series chirps-rainfall estimation and emodis-ndvi satellite images in amhara region, ethiopia. In: A.M. Melesse, W. Abtew, G. Senay (eds.) Extreme Hydrology and Climate Variability, pp. 81 – 92. Elsevier (2019). DOI https://doi.org/10.1016/B978-0-12-815998-9.00008-7

[58] Geurts, P., Ernst, D., Wehenkel, L.: Extremely randomized trees. Machine learning **63**(1), 3–42 (2006)

[59] Ghifary, M., Kleijn, W.B., Zhang, M., Balduzzi, D.: Domain generalization for object recognition with multi-task autoencoders. In: 2015 IEEE International Conference on Computer Vision (ICCV), vol. 11-18-, pp. 2551–2559. IEEE (2015)

[60] Ghifary, M., Kleijn, W.B., Zhang, M., Balduzzi, D., Li, W.: Deep reconstruction-classification networks for unsupervised domain adaptation. In: B. Leibe, J. Matas, N. Sebe, M. Welling (eds.) Computer Vision – ECCV 2016, pp. 597–613. Springer International Publishing, Cham (2016)

[61] Ghorbanian, A., Kakooei, M., Amani, M., Mahdavi, S., Mohammadzadeh, A., Hasanlou, M.: Improved land cover map of iran using sentinel imagery within google earth engine and a novel automatic workflow for land cover classification using migrated training samples. ISPRS journal of photogrammetry and remote sensing **167**, 276–288 (2020)

[62] Giri, C., Long, J.: Land cover characterization and mapping of south america for the year 2010 using landsat 30 m satellite data. Remote Sensing **6**(10), 9494–9510 (2014)

[63] Gong, B., Shi, Y., Sha, F., Grauman, K.: Geodesic flow kernel for unsupervised domain adaptation. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, pp. 2066–2073. IEEE (2012)

[64] Gong, P., Wang, J., Yu, L., Zhao, Y., Zhao, Y., Liang, L., Niu, Z., Huang, X., Fu, H., Liu, S., Li, C., Li, X., Fu, W., Liu, C., Xu, Y., Wang, X., Cheng, Q., Hu, L., Yao, W., Chen, J.: Finer resolution observation and monitoring of global land cover: First mapping results with landsat tm and etm+ data. International Journal of Remote Sensing **34**, 48 (2013). DOI 10.1080/01431161.2012.748992

[65] Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT Press (2016). http://www.deeplearningbook.org

[66] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, K.Q. Weinberger (eds.) Advances in Neural Information Processing Systems, vol. 27, pp. 2672–2680. Curran Associates, Inc. (2014)

[67] Górecki, T., Łuczak, M.: Using derivatives in time series classification. Data Mining and Knowledge Discovery **26**(2), 310–331 (2013)

[68] Grabocka, J., Wistuba, M., Schmidt-Thieme, L.: Fast classification of univariate and multivariate time series through shapelet discovery. Knowledge and Information Systems **49**(2), 429–454 (2016)

[69] Gutman, G., Janetos, A.C., Justice, C.O., Moran, E.F., Mustard, J.F., Rindfuss, R.R., Skole, D., Turner, B.L., Cochrane, M.A.: Land Change Science: Observing, Monitoring and Understanding Trajectories of Change on the Earth's Surface, *Remote Sensing and Digital Image Processing,*, vol. 6. Springer Netherlands, Dordrecht (2004)

[70] Hagolle, O., Huc, M., Villa Pascual, D., Dedieu, G.: A multi-temporal and multi-spectral method to estimate aerosol optical thickness over land, for the atmospheric correction of FormoSat-2, LandSat, VENμS and Sentinel-2 images. Remote Sensing **7**(3), 2668–2691 (2015)

[71] Halevy, A., Norvig, P., Pereira, F.: The unreasonable effectiveness of data. IEEE Intelligent Systems **24**(2), 8–12 (2009)

[72] Ham, J., Chen, Y., Crawford, M.M., Ghosh, J.: Investigation of the random forest framework for classification of hyperspectral data. IEEE Transactions on Geoscience and Remote Sensing **43**, 492–501 (2005)

[73] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016). DOI 10.1109/CVPR.2016.90

[74] Herrmann, M., Webb, G.I.: Early abandoning pruned DTW and its application to similarity search. arXiv preprint (2020). URL http://arxiv.org/abs/2010.05371

[75] Hills, J., Lines, J., Baranauskas, E., Mapp, J., Bagnall, A.: Classification of time series by shapelet transformation. Data Mining and Knowledge Discovery **28**(4), 851–881 (2014)

[76] Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., Kingsbury, B.: Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine **29**(6), 82–97 (2012). DOI 10.1109/MSP.2012.2205597

[77] Hoffman, J., Tzeng, E., Park, T., Zhu, J.Y., Isola, P., Saenko, K., Efros, A., Darrell, T.: CyCADA: Cycle-consistent adversarial domain adaptation. In: J. Dy, A. Krause (eds.) Proceedings of the 35th International Conference on Machine Learning, *Proceedings of Machine Learning Research*, vol. 80, pp. 1989–1998. PMLR, Stockholmsmässan, Stockholm Sweden (2018)

[78] Huang, J., Gretton, A., Borgwardt, K., Schölkopf, B., Smola, A.J.: Correcting sample selection bias by unlabeled data. In: B. Schölkopf, J.C. Platt, T. Hoffman (eds.) Advances in Neural Information Processing Systems 19, pp. 601–608. MIT Press (2007)

[79] Inglada, J., Vincent, A., Arias, M., Tardy, B.: iota2-a25386 (2016). DOI 10.5281/zenodo.58150. URL https://doi.org/10.5281/zenodo.58150

[80] Inglada, J., Vincent, A., Arias, M., Tardy, B., Morin, D., Rodes, I.: Operational high resolution land cover map production at the country scale using satellite image time series. Remote Sensing **9**(1), 95 (2017)

[81] James, W., Stein, C.: Estimation with quadratic loss. In: Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics, pp. 361–379. University of California Press, Berkeley, Calif. (1961). URL https://projecteuclid.org/euclid.bsmsp/1200512173

[82] Jeong, Y.S., Jeong, M.K., Omitaomu, O.A.: Weighted dynamic time warping for time series classification. Pattern Recognition **44**(9), 2231–2240 (2011)

[83] Johansson, F.D., Sontag, D., Ranganath, R.: Support and invertibility in domain-invariant representations. In: K. Chaudhuri, M. Sugiyama (eds.) Proceedings of Machine Learning Research, *Proceedings of Machine Learning Research*, vol. 89, pp. 527–536. PMLR (2019)

[84] Joly, D., Brossard, T., Cardot, H., Cavailhes, J., Hilal, M., Wavresky, P.: Les types de climats en france, une construction spatiale. Cybergeo: European Journal of Geography (2010)

[85] Karlsson, I., Papapetrou, P., Boström, H.: Generalized random shapelet forests. Data Mining and Knowledge Discovery **30**(5), 1053–1085 (2016)

[86] Keogh, E., Wei, L., Xi, X., Lee, S.H., Vlachos, M.: LB_Keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures. In: Proceedings of the 32nd international conference on Very large data bases, pp. 882–893. VLDB Endowment (2006)

[87] Keogh, E.J., Pazzani, M.J.: Derivative dynamic time warping. In: Proceedings of the 2001 SIAM International Conference on Data Mining, pp. 1–11. SIAM (2001)

[88] Kim, T., Kim, C.: Attract, perturb, and explore: Learning a feature alignment network for semi-supervised domain adaptation. In: A. Vedaldi, H. Bischof, T. Brox, J.M. Frahm (eds.) Computer Vision – ECCV 2020, pp. 591–607. Springer International Publishing, Cham (2020)

[89] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Y. Bengio, Y. LeCun (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)

[90] Kouw, W.M., Loog, M.: A review of domain adaptation without target labels. IEEE Transactions on Pattern Analysis and Machine Intelligence **43**(3), 766–785 (2021). DOI 10.1109/TPAMI.2019.2945942

[91] Kouw, W.M., van der Maaten, L.J., Krijthe, J.H., Loog, M.: Feature-level domain adaptation. Journal of Machine Learning Research **17**(171), 1–32 (2016)

[92] Krawczyk, B.: Learning from imbalanced data: open challenges and future directions. Progress in Artificial Intelligence **5**(4), 221–232 (2016)

[93] Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. Communications of the ACM **60**(6), 84–90 (2017)

[94] Lavalle, C., Demicheli, L., Kasanko, M., et al.: Towards an urban atlas. assessment of spatial data on 25 European cities and urban areas. Environmental issue report. European Environment Agency, Copenhagen (2002)

[95] Lavorel, S., Flannigan, M.D., Lambin, E.F., Scholes, M.C.: Vulnerability of land systems to fire: Interactions among humans, climate, the atmosphere, and ecosystems. Mitigation and Adaptation Strategies for Global Change **12**(1), 33–53 (2007)

[96] Le Guennec, A., Malinowski, S., Tavenard, R.: Data Augmentation for Time Series Classification using Convolutional Neural Networks. In: ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data. Riva Del Garda, Italy (2016)

[97] Le Nguyen, T., Gsponer, S., Ilie, I., O'Reilly, M., Ifrim, G.: Interpretable time series classification using linear models and multi-resolution multi-domain symbolic representations. Data mining and knowledge discovery **33**(4), 1183–1222 (2019)

[98] Lecun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., Jackel, L.: Backpropagation applied to handwritten zip code recognition. Neural Computation **1**(4), 541–551 (1989)

[99] LeCun, Y.A., Bottou, L., Orr, G.B., Müller, K.R.: Efficient BackProp, pp. 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). DOI 10.1007/978-3-642-35289-8_3

[100] Lemire, D.: Faster retrieval with a two-pass dynamic-time-warping lower bound. Pattern Recognition **42**(9), 2169 – 2180 (2009). DOI https://doi.org/10.1016/j.patcog.2008.11.030

[101] Lifshits, Y.: Nearest neighbor search: algorithmic perspective. SIGSPATIAL Special **2**(2), 12–15 (2010)

[102] Lin, J., Khade, R., Li, Y.: Rotation-invariant similarity in time series using bag-of-patterns representation. Journal of Intelligent Information Systems **39**(2), 287–315 (2012)

[103] Lines, J., Bagnall, A.: Time series classification with ensembles of elastic distance measures. Data Mining and Knowledge Discovery **29**(3), 565 (2015)

[104] Lines, J., Taylor, S., Bagnall, A.: Hive-cote: The hierarchical vote collective of transformation-based ensembles for time series classification. In: 2016 IEEE 16th International Conference on Data Mining (ICDM), pp. 1041–1046 (2016). DOI 10.1109/ICDM.2016.0133

[105] Lines, J., Taylor, S., Bagnall, A.: Time series classification with hive-cote: The hierarchical vote collective of transformation-based ensembles. ACM transactions on knowledge discovery from data **12**(5), 1–35 (2018)

[106] Liu, M.Y., Tuzel, O.: Coupled generative adversarial networks. In: D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, R. Garnett (eds.) Advances in Neural Information Processing Systems, vol. 29, pp. 469–477. Curran Associates, Inc. (2016)

[107] Liu, S., Deng, W.: Very deep convolutional neural network based image classification using small training sample size. In: 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR), pp. 730–734 (2015). DOI 10.1109/ACPR.2015.7486599

[108] Liu, Y., Yu, J., Han, Y.: Understanding the effective receptive field in semantic image segmentation. Multimedia Tools and Applications **77**(17), 22,159–22,171 (2018)

[109] Lubba, C.H., Sethi, S., Knaute, P., Schultz, S., Fulcher, B., Jones, N.: catch22: Canonical time-series characteristics: Selected through highly comparative time-series analysis. Data Mining and Knowledge Discovery **33** (2019). DOI 10.1007/s10618-019-00647-x

[110] Lucas, B., Pelletier, C., Inglada, J., Schmidt, D., Webb, G., Petitjean, F.: Exploring data quantity requirements for domain adaptation in the classification of satellite image time series. In: F. Bovolo, S. Liu (eds.) MultiTemp 2019, 10th International Workshop on the Analysis of Multitemporal Remote Sensing Images, August 5-7, 2019 – Shanghai, China. IEEE, Institute of Electrical and Electronics Engineers, United States of America (2019). DOI 10.1109/Multi-Temp.2019.8866898

[111] Luo, W., Li, Y., Urtasun, R., Zemel, R.: Understanding the effective receptive field in deep convolutional neural networks. In: D.D. Lee, M. Sugiyama, U.V. Luxburg, I. Guyon, R. Garnett (eds.) Advances in Neural Information Processing Systems, pp. 4898–4906. Curran Associates, Inc. (2016)

[112] Marteau, P.F.: Time warp edit distance with stiffness adjustment for time series matching. IEEE Transactions on Pattern Analysis and Machine Intelligence **31**(2), 306–318 (2009)

[113] Marteau, P.F.: Times series averaging and denoising from a probabilistic perspective on time-elastic kernels. arXiv preprint arXiv:1611.09194 (2016)

[114] Matasci, G., Longbotham, N., Pacifici, F., Kanevski, M., Tuia, D.: Understanding angular effects in vhr imagery and their significance for urban land-cover model portability: A study of two multi-angle in-track image sequences. ISPRS Journal of Photogrammetry and Remote Sensing **107**, 99 – 111 (2015)

[115] Matasci, G., Tuia, D., Kanevski, M.: SVM-based boosting of active learning strategies for efficient domain adaptation. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing **5**(5), 1335–1343 (2012). DOI 10.1109/JSTARS.2012.2202881

[116] Maugeais, E., Lecordix, F., Halbecq, X., Braun, A.: Dérivation cartographique multi échelles de la BDTopo de l'IGN france: mise en œuvre du processus de production de la nouvelle

carte de base. In: Proceedings of the 25th International Cartographic Conference, Paris, pp. 3–8 (2011)

[117] Middlehurst, M., Large, J., Bagnall, A.: The canonical interval forest (cif) classifier for time series classification (2020)

[118] Mölders, N.: Land-Use and Land-Cover Changes: Impact on Climate and Air Quality, *Atmospheric and Oceanographic Sciences Library*, vol. 44. Springer Netherlands (2012)

[119] National Aeronautics and Space Administration: Landsat 8. https://landsat.gsfc.nasa.gov/landsat-8, accessed 2019-03-10

[120] Pan, S.J., Tsang, I.W., Kwok, J.T., Yang, Q.: Domain adaptation via transfer component analysis. IEEE Transactions on Neural Networks **22**(2), 199–210 (2011). DOI 10.1109/TNN.2010.2091281

[121] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019)

[122] Patel, V.M., Gopalan, R., Li, R., Chellappa, R.: Visual domain adaptation: A survey of recent advances. IEEE Signal Processing Magazine **32**(3), 53–69 (2015)

[123] Pękalska, E., Duin, R.P., Paclík, P.: Prototype selection for dissimilarity-based classifiers. Pattern Recognition **39**(2), 189–208 (2006)

[124] Pelletier, C., Ji, Z., Hagolle, O., Morse-McNabb, E., Sheffield, K., Webb, G.I., Petitjean, F.: Using Sentinel-2 image time series to map the state of Victoria, Australia. In: IEEE 10th International Workshop on the Analysis of Multitemporal Remote Sensing Images (MultiTemp), pp. 1–4 (2019)

[125] Pelletier, C., Valero, S., Inglada, J., Dedieu, G., Champion, N.: Filtering mislabeled data for improving time series classification. In: 2017 9th International Workshop on the Analysis of Multitemporal Remote Sensing Images (MultiTemp), pp. 1–4 (2017)

[126] Pelletier, C., Webb, G.I., Petitjean, F.: Temporal convolutional neural network for the classification of satellite image time series. Remote Sensing **11**(5), 523 (2019)

[127] Peng, X., Bai, Q., Xia, X., Huang, Z., Saenko, K., Wang, B.: Moment matching for multi-source domain adaptation. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pp. 1406–1415. IEEE Computer Society, Los Alamitos, CA, USA (2019). DOI 10.1109/ICCV.2019.00149

[128] Persello, C., Bruzzone, L.: Active learning for domain adaptation in the supervised classification of remote sensing images. IEEE Transactions on Geoscience and Remote Sensing **50**(11), 4468–4483 (2012). DOI 10.1109/TGRS.2012.2192740

[129] Petitjean, F., Forestier, G., Webb, G.I., Nicholson, A.E., Chen, Y., Keogh, E.: Dynamic time warping averaging of time series allows faster and more accurate classification. In: 2014 IEEE International Conference on Data Mining, pp. 470–479. IEEE (2014)

[130] Petitjean, F., Forestier, G., Webb, G.I., Nicholson, A.E., Chen, Y., Keogh, E.: Faster and more accurate classification of time series by exploiting a novel dynamic time warping averaging algorithm. Knowledge and Information Systems **47**(1), 1–26 (2016)

[131] Petitjean, F., Gançarski, P.: Summarizing a set of time series by averaging: From Steiner sequence to compact multiple alignment. Theoretical Computer Science **414**(1), 76–91 (2012)

[132] Petitjean, F., Inglada, J., Gançarski, P.: Satellite image time series analysis under time warping. IEEE transactions on geoscience and remote sensing **50**(8), 3081–3095 (2012)

[133] Qin, C., Wang, L., Ma, Q., Yin, Y., Wang, H., Fu, Y.: Opposite structure learning for semi-supervised domain adaptation. arXiv preprint (2020). URL http://arxiv.org/abs/2002.02545

[134] Rakthanmanon, T., Keogh, E.: Fast shapelets: A scalable algorithm for discovering time series shapelets. In: Proceedings of the 13th SIAM international conference on data mining, pp. 668–676. SIAM (2013)

[135] Roberts, D.R., Bahn, V., Ciuti, S., Boyce, M.S., Elith, J., Guillera-Arroita, G., Hauenstein, S., Lahoz-Monfort, J.J., Schröder, B., Thuiller, W., et al.: Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. Ecography **40**(8), 913–929 (2017)

[136] Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature **323**(6088), 533–536 (1986)

[137] Saito, K., Kim, D., Sclaroff, S., Darrell, T., Saenko, K.: Semi-supervised domain adaptation via minimax entropy. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 8050–8058 (2019)

[138] Saito, K., Ushiku, Y., Harada, T., Saenko, K.: Strong-weak distribution alignment for adaptive object detection. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6949–6958 (2019). DOI 10.1109/CVPR.2019.00712

[139] Sakoe, H., Chiba, S.: A dynamic programming approach to continuous speech recognition. In: Proceedings of the seventh international congress on acoustics, vol. 3, pp. 65–69. Budapest, Hungary (1971)

[140] Sakoe, H., Chiba, S.: Dynamic programming algorithm optimization for spoken word recognition. IEEE transactions on acoustics, speech, and signal processing **26**(1), 43–49 (1978)

[141] Sathe, S., Aggarwal, C.C.: Similarity forests. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '17, pp. 395–403. ACM, New York, NY, USA (2017). DOI 10.1145/3097983.3098046

[142] Schäfer, P.: The BOSS is concerned with time series classification in the presence of noise. Data Mining and Knowledge Discovery **29**(6), 1505 (2015)

[143] Schäfer, P., Högqvist, M.: SFA: A symbolic fourier approximation and index for similarity search in high dimensional datasets. In: Proceedings of the 15th International Conference on Extending Database Technology, EDBT '12, pp. 516–527. ACM, New York, NY, USA (2012). DOI 10.1145/2247596.2247656

[144] Schäfer, P., Leser, U.: Fast and accurate time series classification with WEASEL. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 637–646. ACM (2017)

[145] Schmidhuber, J.: Deep learning in neural networks: An overview. Neural Networks **61**, 85 – 117 (2015). DOI https://doi.org/10.1016/j.neunet.2014.09.003

[146] Schäfer, P., Schäfer, P.: Scalable time series classification. Data mining and knowledge discovery **30**(5), 1273–1298 (2016)

[147] Senin, P., Malinchik, S.: SAX-VSM: Interpretable time series classification using SAX and vector space model. In: 2013 IEEE 13th International Conference on Data Mining, pp. 1175–1180. IEEE (2013)

[148] Seo, B., Bogner, C., Poppenborg, P., Martin, E., Hoffmeister, M., Jun, M., Koellner, T., Reineking, B., Shope, C.L., Tenhunen, J.: Deriving a per-field land use and land cover map in an agricultural mosaic catchment. Earth System Science Data **6**(2), 339 (2014)

[149] Sharma, R., Tateishi, R., Hara, K., Iizuka, K.: Production of the japan 30-m land cover map of 2013-2015 using a random forests-based feature optimization approach. Remote Sensing **8**(5) (2016)

[150] Shifaz, A., Pelletier, C., Petitjean, F., Webb, G.I.: Ts-chief: a scalable and accurate forest algorithm for time series classification. Data mining and knowledge discovery **34**(3), 742–775 (2020)

[151] Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W., Webb, R.: Learning from simulated and unsupervised images through adversarial training. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2242–2251 (2017). DOI 10.1109/CVPR.2017.241

[152] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: Y. Bengio, Y. LeCun (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015). URL http://arxiv.org/abs/1409.1556

[153] Stefan, A., Athitsos, V., Das, G.: The move-split-merge metric for time series. IEEE transactions on Knowledge and Data Engineering **25**(6), 1425–1438 (2013)

[154] Sun, B., Saenko, K.: Deep coral: Correlation alignment for deep domain adaptation. In: G. Hua, H. Jégou (eds.) Computer Vision – ECCV 2016 Workshops, pp. 443–450. Springer International Publishing, Cham (2016)

[155] Sun, C., Shrivastava, A., Singh, S., Gupta, A.: Revisiting unreasonable effectiveness of data in deep learning era. In: 2017 IEEE International Conference on Computer Vision (ICCV), vol. 2017-, pp. 843–852. IEEE (2017)

[156] Szegedy, C., Ioffe, S., Vanhoucke, V.: Inception-v4, inception-resnet and the impact of residual connections on learning. arXiv preprint (2016). URL http://arxiv.org/abs/1602.07261

[157] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–9 (2015). DOI 10.1109/CVPR.2015.7298594

[158] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), vol. 2016-, pp. 2818–2826. IEEE (2016)

[159] Tan, C.W., Webb, G.I., Petitjean, F.: Indexing and classifying gigabytes of time series under time warping. In: Proceedings of the 2017 SIAM International Conference on Data Mining, pp. 282–290 (2017). DOI 10.1137/1.9781611974973.32

[160] Tang, B., Pan, Z., Yin, K., Khateeb, A.: Recent advances of deep learning in bioinformatics and computational biology. Frontiers in Genetics **10**, 214 (2019). DOI 10.3389/fgene.2019.00214

[161] Tardy, B., Inglada, J., Michel, J.: Fusion approaches for land cover map production using high resolution image time series without reference data of the corresponding period. Remote Sensing **9**(11) (2017)

[162] Tardy, B., Inglada, J., Michel, J.: Assessment of optimal transport for operational land-cover mapping using high-resolution satellite images time series without reference data of the mapping period. Remote Sensing **11**(9), 1047 (2019)

[163] Tesfaw, A., Pfaff, A., Golden Kroner, R., Qin, S., Medeiros, R., Mascia, M.: Land-use and land-cover change shape the sustainability and impacts of protected areas. Proceedings of the National Academy of Sciences of the United States of America **115**(9), 2084 (2018)

[164] Ting, K.M., Zhu, Y., Carman, M., Zhu, Y., Zhou, Z.H.: Overcoming key weaknesses of distance-based neighbourhood methods using a data dependent dissimilarity measure. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1205–1214. ACM (2016)

[165] Townshend, J.G.: Land cover. International Journal of Remote Sensing **13**(6-7), 1319–1328 (1992)

[166] Triguero, I., Triguero, I., García, S., García, S., Herrera, F., Herrera, F.: Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. Knowledge and information systems **42**(2), 245–284 (2015)

[167] Tuia, D., Camps-Valls, G.: Kernel manifold alignment for domain adaptation. PLOS ONE **11**(2), 1–25 (2016). DOI 10.1371/journal.pone.0148655

[168] Tuia, D., Pasolli, E., Emery, W.: Using active learning to adapt remote sensing image classifiers. Remote Sensing of Environment **115**(9), 2232 – 2242 (2011)

[169] Tuia, D., Persello, C., Bruzzone, L.: Domain adaptation for the classification of remote sensing data an overview of recent advances. Ieee Geoscience And Remote Sensing Magazine **4**(2), 41–57 (2016)

[170] Turner, B.L., Lambin, E.F., Reenberg, A.: The emergence of land change science for global environmental change and sustainability. Proceedings of the National Academy of Sciences of the United States of America **104**(52), 20,666–20,671 (2007)

[171] Tzeng, E., Hoffman, J., Saenko, K., Darrell, T.: Adversarial discriminative domain adaptation. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2962–2971 (2017). DOI 10.1109/CVPR.2017.316

[172] Tzeng, E., Hoffman, J., Zhang, N., Saenko, K., Darrell, T.: Deep domain confusion: Maximizing for domain invariance. arXiv preprint (2014). URL http://arxiv.org/abs/1412.3474

[173] Ueno, K., Xi, X., Keogh, E., Lee, D.J.: Anytime classification using the nearest neighbor algorithm with applications to stream mining. In: Data Mining, 2006. ICDM'06. Sixth International Conference on, pp. 623–632. IEEE (2006)

[174] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u., Polosukhin, I.: Attention is all you need. In: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (eds.) Advances in Neural Information Processing Systems, vol. 30. Curran Associates, Inc. (2017)

[175] Viana, C.M., Oliveira, S., Oliveira, S.C., Rocha, J.: Land use/land cover change detection and urban sprawl analysis. In: H.R. Pourghasemi, C. Gokceoglu (eds.) Spatial Modeling in GIS and R for Earth and Environmental Sciences, pp. 621–651. Elsevier (2019). DOI https://doi.org/10.1016/B978-0-12-815226-3.00029-6

[176] Vlachos, M., Hadjieleftheriou, M., Gunopulos, D., Keogh, E.: Indexing multidimensional time-series. The VLDB Journal—The International Journal on Very Large Data Bases **15**(1), 1–20 (2006)

[177] Vuolo, F., Neuwirth, M., Immitzer, M., Atzberger, C., Ng, W.T.: How much does multi-temporal Sentinel-2 data improve crop type classification? International Journal of Applied Earth Observation and Geoinformation **72**, 122–130 (2018). DOI 10.1016/j.jag.2018.06.007

[178] Wang, W., Zhou, Z.H.: A new analysis of co-training. In: Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML'10, p. 1135–1142. Omnipress, Madison, WI, USA (2010)

[179] Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., Keogh, E.: Experimental comparison of representation methods and distance measures for time series data. Data Mining and Knowledge Discovery **26**(2), 275–309 (2013)

[180] Wang, Y., Emonet, R., Fromont, E., Malinowski, S., Tavenard, R.: Adversarial regularization for explainable-by-design time series classification. In: 2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI), pp. 1079–1087 (2020). DOI 10.1109/ICTAI50040.2020.00165

[181] Wang, Y.C., Feng, C.C.: Patterns and trends in land-use land-cover change research explored using self-organizing map. International Journal of Remote Sensing **32**(13), 3765–3790 (2011)

[182] Wang, Z., Yan, W., Oates, T.: Time series classification from scratch with deep neural networks: A strong baseline. In: Neural Networks (IJCNN), 2017 International Joint Conference on, pp. 1578–1585. IEEE (2017)

[183] Wilson, G., Cook, D.J.: A survey of unsupervised deep domain adaptation. ACM Trans. Intell. Syst. Technol. **11**(5) (2020). DOI 10.1145/3400066

[184] Wulder, M.A., Coops, N.C., Roy, D.P., White, J.C., Hermosilla, T.: Land cover 2.0. International Journal of Remote Sensing **39**(12), 4254–4284 (2018)

[185] Yamada, Y., Suzuki, E., Yokoi, H., Takabayashi, K.: Decision-tree induction from time-series data based on a standard-example split test. In: Proceedings of the Twentieth International Conference on International Conference on Machine Learning, ICML'03, pp. 840–847. AAAI Press (2003)

[186] Yang, L., Wang, Y., Gao, M., Shrivastava, A., Weinberger, K.Q., Chao, W.L., Lim, S.N.: Deep co-training with task decomposition for semi-supervised domain adaptation (2020)

[187] Yao, T., Yingwei Pan, Ngo, C., Houqiang Li, Tao Mei: Semi-supervised domain adaptation with subspace learning for visual recognition. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2142–2150 (2015). DOI 10.1109/CVPR.2015.7298826

[188] Ye, L., Keogh, E.: Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. Data mining and knowledge discovery **22**(1), 149–182 (2011)

[189] Yosinski, J., Clune, J., Bengio, Y., Lipson, H.: How transferable are features in deep neural networks? In: Advances in Neural Information Processing Systems, pp. 3320–3328 (2014)

[190] Yu, L., Wang, J., Gong, P.: Improving 30 m global land-cover map from-glc with time series modis and auxiliary data sets: A segmentation-based approach. International Journal of Remote Sensing **34**, 5851–5867 (2013). DOI 10.1080/01431161.2013.798055

[191] Yu, X., Zhang, B., Li, Q., Chen, J.: A method characterizing urban expansion based on land cover map at 30 m resolution. Science China.Earth Sciences **59**(9), 1738–1744 (2016)

[192] Zhang, H., Cisse, M., Dauphin, Y.N., Lopez-Paz, D.: mixup: Beyond empirical risk minimization. In: International Conference on Learning Representations (2018)

[193] Zhang, L.: Transfer adaptation learning: A decade survey. arXiv preprint (2019). URL http://arxiv.org/abs/1903.04687

[194] Zhang, S., Yao, L., Sun, A., Tay, Y.: Deep learning based recommender system: A survey and new perspectives. ACM Computing Surveys **52**(1) (2019). DOI 10.1145/3285029

[195] Zhao, S., Yue, X., Zhang, S., Li, B., Zhao, H., Wu, B., Krishna, R., Gonzalez, J.E., Sangiovanni-Vincentelli, A.L., Seshia, S.A., Keutzer, K.: A review of single-source deep unsupervised visual domain adaptation. IEEE Transactions on Neural Networks and Learning Systems pp. 1–21 (2020). DOI 10.1109/TNNLS.2020.3028503