Neural controlled differential equations for crop classification

Master Thesis

Joaquin Gajardo
joaquin.gajardocastillo@epfl.ch

Environmental Sciences and Engineering Section
School of Architecture, Civil and Environmental Engineering
EPFL

Host institution:
Ecovision Lab
Institute of Geodesy and Photogrammetry
Department of Civil, Environmental and Geomatic Engineering
ETH Zürich

Supervisors:

Devis Tuia
ECEO Lab
EPFL

Jan Dirk Wegner
Ecovision Lab
University of Zürich
& ETH Zürich

May 9, 2021

I start by thanking my supervisors Devis and Jan, for this great opportunity to learn with them, which was a very fruitful learning experience for me. I thank Ozgur Turkoglu, and Nando Metzger from my hosting lab for their active support in the project with many helpful discussions and exchanges of ideas, and in special, to Stefano D'Aronco for his active involvement and for carefully reviewing the manuscript of this thesis. I am grateful to Patrick Kidger, for his kind collaboration with many clarifications and ideas which greatly helped me during my work. Last and most importantly, I want to thank my parents for their unconditional love, unwavering support and for always fostering my curiosity, which has all gifted me a life filled with opportunities and good fortune. This work is dedicated to them.

Abstract

Accurate and scalable crop classification is important for food security and sustainable resources management. The temporal development of crops, i.e., their phenology, is a continuous phenomena that if properly captured and analyzed, can help to recognize them. The novel model, *Neural Controlled Differential Equations* (NCDE), extends the disruptive *Neural Ordinary Differential Equations* (NODEs) for processing time series, and allows their hidden state to be dynamically controlled by a continuous representation of the data. This thesis' objective is to explore if a model with a fully continuous hidden state, such as NCDEs, can achieve improved performance in crop classification. The experimental results obtained showed that their performance is still inferior to the state-of-the-art models for this task, however a careful study of their weaknesses suggested potential opportunities for further improving them. In particular it was found that the continuous data representation method chosen for the data may severely affect the performance of the model. Therefore, evaluating improved interpolation and regression methods with respect to those evaluated here are a promising direction to boost their performance.

Keywords: Remote sensing, crop classification, crop phenology, deep learning, time series modelling, neural controlled differential equations, recurrent neural networks.

Résumé

La classification précise et à grande échelle des cultures est importante pour la sécurité alimentaire et la gestion durable des ressources. Le développement temporel des cultures, c'est-à-dire leur phénologie, est un phénomène continu qui, s'il est correctement saisi et analysé, peut aider à les reconnaître. Le nouveau modèle, Équations Différentielles Neurales Contrôlées (NCDE), étend les Equations Différentielles Neurales Ordinaires (NODE) pour traiter des séries temporelles, et permet à leur état caché d'être dynamiquement contrôlé par une représentation continue des données. L'objectif de cette thèse est donc d'explorer si un modèle qui définit un état caché entièrement continu, tel que les NCDE, peut atteindre une meilleure performance dans la classification des cultures. Les résultats obtenus ont montré que leur performance est encore inférieur à l'état de l'art pour cette tâche, mais une étude attentive de leurs faiblesses a suggéré des opportunités potentielles pour les améliorer. En particulier, il a été constaté que la méthode choisie pour le représentation continue des données peut affecter sévèrement la performance des NCDEs. Ainsi l'évaluation de méthodes d'interpolation et de régression améliorées par rapport à celles évaluées ici donnent une direction prometteuse pour augmenter leur performance.

Mot-clés: Télédétection, classification des cultures, phénologie des cultures, apprentissage profond, modélisation des séries temporelles, équations différentielles neurales contrôlées, réseaux neuronaux récurrents.

Contents

Acknowledgements **iii**

Abstract **v**

Résumé **vi**

1. Introduction **1**
1.1. Context . . . 1
1.2. Motivation . . . 1
1.3. Contribution . . . 3
1.4. Outline . . . 4

2. Related work **5**
2.1. Traditional machine learning approaches . . . 5
2.2. Deep learning approaches . . . 6

3. Methods **9**
3.1. Deep learning and neural networks . . . 9
3.2. Recurrent neural networks . . . 12
3.2.1. Long short-term memory networks . . . 12
3.2.2. Gated recurrent units . . . 14
3.3. Neural ordinary differential equations . . . 14
3.3.1. Background . . . 15
3.3.2. Definition . . . 15
3.3.3. Characteristics . . . 16
3.3.4. The ODERNN model . . . 17
3.4. Neural controlled differential equations . . . 18
3.4.1. NCDE definition . . . 18
3.4.2. Stacked NCDE . . . 20
3.4.3. Continuous data representation methods . . . 21
3.4.4. Properties . . . 24
3.5. Software . . . 24

4. Data **25**
4.1. TU Munich dataset . . . 25
4.2. Swisscrop . . . 27
4.3. Pre-processing . . . 27

5. Results **31**
5.1. Experimental setup . . . 31
5.2. Model performance . . . 33
5.2.1. Base case . . . 33
5.2.2. Per-crop performance analysis . . . 34
5.3. Ablation studies . . . 36
5.3.1. Observational mask importance . . . 36
5.3.2. Interpolation methods . . . 37
5.3.3. High-dimensional dataset . . . 43
5.3.4. Cloud cover study . . . 44

6. Discussion **47**
6.1. Analysis of results . . . 47
6.2. Limitations of NCDEs . . . 48
6.3. Outlook . . . 50

7. Conclusion **53**

A. Hyperparameters analysis **57**

Chapter 1

Introduction

In this chapter the context of this work is first introduced, followed by its motivation and contribution. An outline of the structure of the rest of the work is additionally provided at the end of the chapter.

1.1. Context

Earth's biophysical surface can affect the climate system, habitat's biodiversity and ecosystem's capacity for supporting human needs [1]. Therefore, having an up-to-date and accurate account of the planet's land cover is essential for environmental research, resources management, monitoring climate change and disaster prevention [2]. Remote sensing is a field intrinsically related to land cover monitoring [1], and can provide an affordable and accessible way of doing so at a large scale using satellite images [2], [3].

Agricultural land monitoring in particular, is of high relevance for securing a sustainable food production around the world [3]–[5]. There are several factors that put pressure in agricultural land, such as the intense growth in population anticipated for this century [6], and the expansion of urban land and bio-fuel production [3]. On the other hand, agriculture has been linked to environmental systems degradation [3], biodiversity loss [7] and desertification when not properly managed [8], and as a result, the potential for expanding crop land in exchange of range land and forest is limited [5]. All these factors establish the necessity for agricultural monitoring. In particular, precise knowledge about crop distribution provides crucial information for agricultural monitoring [4], which can be used for the investigation of land management practices [5] and for the design and application of resources management and agrarian policy, such as subsidies [9]. Information about the crop distribution can be obtained via ground visits or surveys, however remote sensing provides a more cost-effective way of doing so from a distance. Such problem is known in the field as *crop type identification* or *crop type classification* [9].

1.2. Motivation

When attempting to monitor and classify agricultural land, in order for a model to be able to distinguish the different crops, a complete and accurate representation of their

distinctive attributes is helpful. In this regard, the phenological development of a crop (i.e. its biological progression towards maturity) is a characteristic signature and thus useful for their identification [10]. Single-date spectral information has proven to be unsuitable for crop classification in practice, due to plants having similar spectral attributes at certain stages of their growth [4], [5]. For decades researchers have attempted to characterize crops by extracting their phenological information from multi-temporal data [4], [5], [10]–[12], usually by computing vegetation indexes from the spectral values at the different points in time. These studies and more recent ones [13], [14] have confirmed the added benefit of multi-temporal monitoring and modelling for crop classification. Nowadays many modern satellites have both a fine spatial resolution and a small temporal revisit time, e.g. Sentinel-2 has 10-60 m spatial resolution bands and only a 5 days revisit time. Therefore the chances of acquiring many cloudless images throughout the year are increased, allowing for effective time series modelling and plot-level crop classification [13], [15].

Machine learning methods have been extensively used in remote sensing literature for automating crop classification, with models such as *Maximum Likelihood Classifiers* (MLC) [12], [16], [17], *Random Forests* (RF) [3], [13], [18] and *Support Vector Machines* (SVM) [19]–[21]. However, nowadays deep learning methods such as neural networks have gained widespread popularity due to remarked success in the technological industry sector and in many academic research fields, such as computer vision and machine translation, and are increasingly being adopted by the remote sensing community as a toolbox for a variety of problems [22], [23]. This kind of methods are well designed for fitting large amounts of high-dimensional data and can learn highly non-linear relationships between the input data and the target variable [23]. In particular, *Convolutional Neural Networks* (CNN) have successfully been applied in remote sensing in areas such as object & wildlife detection [24], [25] and semantic segmentation [25], [26], while *Recurrent Neural Networks* (RNN) have been applied for modelling time series of satellite images for land cover and crop classification [15], [27], [28]. CNNs are effective when dealing with satellite images, as they can aggregate and extract contextual information from a pixel in an image by applying convolutions to its neighbour pixels, and therefore can model spatial and spectral dependencies in the data. On the other hand RNNs are especially effective in modelling sequential data such as time series, thus being able to learn temporal and spectral dependencies in the data for each individual pixel of a time series of satellite images of a given area. In this thesis, RNNs are considered as baselines models.

Models such as RNNs, encode relevant information learnt from input data in a hidden state variable. Very recently, a novel family of deep learning models, called *Neural Ordinary Differential Equations* (NODE) [29], has been introduced. This model also encodes information in a hidden state, but the evolution of the hidden state is defined by an ordinary differential equations whose right hand side (i.e., its vector field) can be parameterized using a neural network, rather than an analytical function. In contrast to RNNs, these models have a hidden state that is continuously defined [29], but they are unable to process sequential input data by their own, such as time series. However, a

variation of the model, called ODERNN has been proposed [30], which combines NODEs with an RNN for reading new input data, and thus can effectively model time series. The ODERNN model has been recently applied for the first time in remote sensing [31], outperforming other non-contextual models for crop classification, such as RNNs and *Temporal Convolutional Neural Networks* (TempCNN).

1.3. Contribution

Given the successful application of an ODERNN model for crop classification, this thesis focuses on the exploration of a related model for this same problem. In 2020, a similar model to NODE called *Neural Controlled Differential Equations* (NCDEs) was proposed [32]. This model is a slight, but robust and elegant modification to a NODE model, drawn from controlled differential equations [33]. Roughly speaking, this model is like a NODE that is guided by a continuous-representation of the input data, which can be built by interpolation or curve fitting [32]. Unlike an ODERNN model, a NCDE does not require an RNN for reading the time series data, therefore the hidden state is allowed to evolve continuously throughout the time series processing, while still being controlled by the data, and it does not present "jumps" of the hidden state, unlike those observed in an ODERNN model [31].

The reason behind preferring a smooth representation of the input, and hidden state, stems from phenology. Phenology models have become an important component in fields such as agriculture and ecology [34] and it is an active area of research [35]–[37]. These models normally rely only on meteorological data [34], [36], but can also incorporate satellite-based remote sensing products, like vegetation indexes [37]–[40]. The trend in this line of work seems to recognize the value of continuous modelling of plant phenology, because the underlying phenomenon itself is continuous [35], [37], [38], [40]. In particular, continuous time models that represent the phenological state through a continuous latent state (i.e. hidden state) have been proposed, such as the use of a continuous *Hidden Markov Model* (HMM) [35] and a spatio-temporal Bayesian approach [37]. The methods of this thesis are in line with this trend, and posits that having a model with a continuously defined hidden state, such as a NCDE, is better suited for modelling the continuous underlying phenomena that determine a crop's signature, i.e., its phenological development. This work is the first attempt to use a NCDE approach to implicitly model a phenological state, in this case using only remote sensing data coming from satellite image and for the purpose of crop classification.

To this end the main focus of this thesis is to thoroughly analyze the operation and performance of NCDEs with multi-temporal remote sensing datasets for crop classification and to propose and validate modifications to the model. In particular, two modifications are proposed, an alternative method for creating the continuous-input data representation than those proposed in the original paper [32] and also a stacked variant of the model (S-NCDE). Selected state-of-the-art baselines models, together with the NCDE and the S-NDCE models are evaluated by comparing their performance on two datasets, the TU

Munich dataset [15] and the Swisscrop dataset [41].

1.4. Outline

The thesis is structured as follows. Chapter 2 presents the previous work with machine and deep learning for crop classification. Chapter 3 draws a basic introduction to deep learning and defines the baselines and NCDE models evaluated. Chapter 4 presents the two datasets used to evaluate the models. Chapter 5 summarizes the experimental results of this work. Chapter 6 puts the results in perspective and discusses the advantages and limitations of NCDEs as well as an outlook of future work. Finally, Chapter 7 presents the conclusions and takeaways from this work.

Chapter 2

Related work

Crop classification via satellite images is a topic that has been extensively studied in remote sensing literature since the late 60s. In fact, according to [42], the first work looking to automate land use classification [16] in remote sensing used agricultural data. In this chapter, traditional machine learning approaches for crop classification will be presented, followed by the state-of-the-art deep learning approaches.

2.1. Traditional machine learning approaches

The classical approaches for crop classification do not usually consider temporal information, and are normally composed of three sequential steps, namely, data pre-processing, feature extraction, classification, and data post-processing [42]. In the pioneering work by [16], multi-spectral information was considered by fitting the spectral bands to a Gaussian distribution and using them as features, followed by a MLC, however the importance of also including spatial and temporal was already acknowledged. Agricultural land has common specific traits such as distinctive geometries [43] and a high vegetation content, but differentiating between crops may be difficult [44]. Spectral features design in the form of vegetation indexes, such as the *Normalized Vegetation Difference Index* (NDVI) [45], are effectively used until this day for crop classification [3], [9], [20]. [9] used a range of vegetation indexes and textural features such as the entropy and homogeneity to classify crops in California using ASTER satellite images via *Decision Trees (DT)*, a rule-based machine learning algorithm. [3] used RFs, which are an ensemble of DTs and a popular model for crop classification, for producing crop maps on several sites around the world. [20] used NDVI and other vegetation indexes such as the *Normalized Difference Red Edge Index* (NDRE) with SVMs, which are also a popular choice in remote sensing [21].

Many authors have additionally extracted temporal information about the phenological state through profiling vegetation indexes in time and deriving statistics that can be used as features for classification [4], [11], [12], [17], [18]. [11] used NDVI profiles from MODIS satellite images and DTs as a classifier. [4] used bi-temporal information of vegetation indexes and a ruled-based system with ASTER satellite images of Central Asia. Both [12] and [17] built temporal NDVI profiles and extracted features from them, respectively fed to an MLC and a *k-Nearest-Neighbours* classifier (KNN). [18] used a dynamical masking system to obtain temporal NDVI from which several features are extracted by statistics

and given to a RF classifier. RFs and SVMs are general purpose algorithms, and even SVMs may be able to directly process time series input if an appropriate kernel function is used [19]. [3] also used time series of vegetation indexes as features, but given directly to an RF classifier, without explicitly deriving statistics from them. There is also some studies that prove the additional benefit of having time information for crop classification [13], [46]. [46] used NDVI profiles with a RF classifier and focused on finding an optimal amount of dates and time windows to acquire the data. On the other hand, [13] also used a RF classifier for crop classification, but without vegetation index profiling and rather concatenated all the raw bands at the different times and fed them directly to the classifier.

All this aforementioned approaches mainly rely on rule-based systems, including time information through vegetation indexes, and do not explicitly model time series input. Nevertheless, some authors have used models that were specifically applied for their capabilities of modelling time series input in their architecture [5], [6], [47], [48]. [5] used HMM, which is a generative type of graph model, that encodes information in a hidden state. [47] used *Conditional Random Fields* (CRF), a discriminative type of graph model, which as HMMs, also encodes prior information in a hidden state. [48] used CRF for incorporating data into a HMM model and a final RF classifier, on inter-annual images with the aim of modelling crop rotation. [6] used *Time-Weighted Dynamical Time Warping* (TWDTW) while still relying on a temporal NDVI profile. This later model is yet another example of the increasing application of models specifically designed to process time series, for crop classification.

2.2. Deep learning approaches

In recent years, deep learning approaches have been successfully applied for crop classification. These models usually avoid manual feature extraction, such as building statistics from NDVI profiles, as was common with traditional machine learning approaches, and rather allow the models to figure out their own feature representations from the data. Perhaps the first application of neural networks to crop classification was done by [15] using a *Long-Short Term Memory* (LSTM) neural network, a particular type of RNN, for modelling temporal relationships in the raw spectral bands of 26 satellite images from Bavaria, Germany. RNNs are in a way, comparable to a HMM, as they can explicitly model time series and have an internal hidden state that holds the feature representations [49].

On two follow-up papers to [15], [49], [50] used *convolutional RNN* (convRNN) models [51], which have identical mechanics to a normal RNN but with convolutional operations instead, thus effectively accounting for spatial information in the images. [52] also used a convRNN model, but preceded by a U-net (i.e. a CNN variant) to first encode feature representations in the data and then fed them to the convRNN model. [41], [53] created a new variant of RNN cell, and used it as a convRNN model in crop classification problems, achieving state-of-the-art results.

[2], [54], [55] used TempCNNs, which are CNNs that apply a convolution filter only in the temporal dimension rather than a 2-D one in space, to learn time dependencies for crop classification. [56], [57] used *self-attention Transformers* [58], which are a new type of models that have revolutionized the field of natural language processing, to extract relevant temporal information. These models compute an importance score for each element of a time series, which allows them to focus their attention on the informative portions of the time series.

[31] used a ODERNN model [30], for crop classification, achieving improved results compared to simple RNN models. As stated in Chapter 1, this algorithm can continuously model the dynamics of the hidden state of an RNN between its updates with input data, which is thought to be the reason for their better performance. This last work serves as an inspiration for this thesis, where the ODERNN method is considered as a baseline model and the use of NCDEs for crop classification is explored, as a model that defines a fully-continuous hidden state, rather than a semi-continuous one like ODERNNs do.

Chapter 3

Methods

In this chapter all the models used in this thesis are described. First, background information on deep learning is provided, so that readers unfamiliar to the topic may still follow the contents of this work. Then, RNNs are defined and described, with a detailed explanation of the two RNN models considered as baselines, the *Gated Recurrent Units* (GRU) and the LSTM models. Subsequently, NODEs are defined and described in addition to the ODERNN method, which is the third baseline evaluated in this work. Finally, NCDEs are defined and their operation and characteristics are described in detail, as well as the interpolation and curve fitting methods used for creating continuous input data representations for NCDEs.

3.1. Deep learning and neural networks

Machine and deep learning are fields of artificial intelligence that study how models can learn tasks through experience. In practice, instead of explicitly instructing a program with a particular solution to a task, the models learn to extract this information from data itself. Deep learning is a subfield of machine learning, that is concerned with the study of *Artificial Neural Networks* (ANN) or *Neural Networks* (NN) for short, which are a family of models originally inspired by early attempts of modelling the neural circuits in a brain [59]. In general, NNs may be mathematically defined as f_θ, where θ denotes their parameters. A neural network can be seen as a $\mathbf{x} \to \hat{\mathbf{y}}$ mapping, with $\hat{\mathbf{y}} = f_\theta(\mathbf{x})$. Thus, they are functions whose parameters can be learnt from data. There are several theoretical findings [60]–[63] (i.e., the universal approximation theorem), which show that, even shallow neural networks, can approximate any mapping, provided that they include non-linear functions in their design and a sufficiently large amount of learnable parameters.

The simplest form of neural networks are called *Feed-Forward Neural Networks* (FFNN). A representative diagram of their design in presented in Figure 3.1. This type of networks always have at least one layer for taking in the input data and an *output layer* for giving the predictions, in which case the network is said to be of one layer (i.e., the input layer is not counted). They are called feed-forward because the information flows only from left to right when making predictions, i.e., from the input data to the prediction. Additionally, they can have *hidden layers*, for additional representational power. Therefore, Figure 3.1 represents a FFNN of three layers, for instance. Each layer consists of several *units*

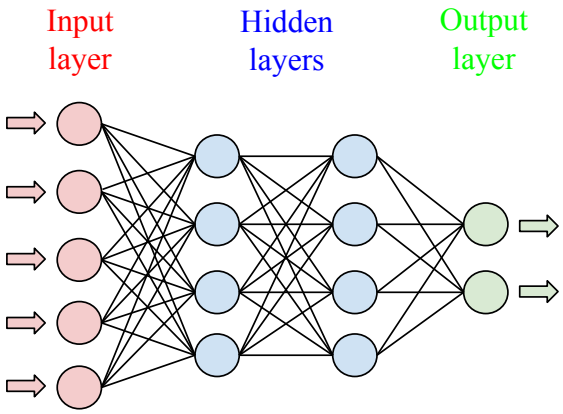


Figure 3.1.: Schematic diagram of a feed-forward neural network, with two hidden layers of four units each, and where each sample in the input data is five-dimensional and has two possible output classes.

or artificial neurons that compute a linear combination of the output of every unit in the previous layer, followed by a non-linear function, called the *activation function* [64]. The amount of units in the input layer matches the input data dimensionality, while for the output layer it matches the output's dimensionality, e.g., two units for a binary classification problem (the definition of a classification problem is provided later in this section). The amount of hidden layers is also know as the *depth* of network, and if the hidden layers have all the same amount of units, this latter number is known as the *width* of the network. The name deep learning stems from this terminology [65]. Each connection between pairs of neurons (i.e., the connecting lines in Figure 3.1), has a *weight* associated to it, which is a parameter representing the strength of the connection and each neuron has an additional additive *bias* term. If l denotes the l^{th} layer of a FFNN, the output of that layer is referred to as its *activation*, and is defined as follows:

$$\mathbf{a}^l = g^l(\mathbf{W}^l\mathbf{a}^{l-1} + \mathbf{b}^l), \tag{3.1}$$

Here, g^l is the non-linear activation function applied after the l^{th} layer, $\mathbf{W}^l$ is a matrix that contains the weights of every connection with the previous layer, and $\mathbf{b}^l$ is a vector with a bias term for each unit in the layer. In this case, the collection of learnable parameters θ, consists of all the different $\mathbf{W}^l$ and $\mathbf{b}^l$ in each layer. The last layer L is the output layer and computes $\hat{\mathbf{y}}$ (i.e., $\mathbf{a}^L = \hat{\mathbf{y}}$). Conversely, the first layer takes in the input data $\mathbf{x}$, which means that: $\mathbf{a}^0 = \mathbf{x}$.

There are several types of experiences or learning paradigms that a machine learning model can learn from [65]. The most common one, is called *supervised learning* and this work's methods fall in that category. In supervised learning tasks the data consists of $\mathbf{x}$ and $\mathbf{y}$ pairs, where $\mathbf{x}$ is the high-dimensional real-valued data fed to the model and

$\mathbf{y}$ is the target variable, which can be a a single value or a sequence of them, which in turn can be either discrete or real numbers. When the values in the target variable $\mathbf{y}$ are part of a discrete set, the task is defined as a classification task. Since the classification problem here has only a single-valued target variable, $\mathbf{y}$ is henceforth denoted simply as y, and it is referred to as the class or label that a data point belongs to.

In its most common form, the learning process is governed by a gradient-based *optimization algorithm*, which can be generically summarized as follows. The parameters of the network are initialized and then batches of the input data are passed through the model, in what is called a *forward pass*, making a prediction $\hat{y}$ about the class of each sample in the batch. A scalar loss is the computed, e.g., mean squared error, w.r.t. the true label y. A *backward pass* then computes the gradient of the network's parameters w.r.t the loss by *backpropagation* (i.e., applying the chain rule). The parameters are then updated by taking a step according to their gradients. This process is repeated until the loss converges to a minimum and this is why it is also referred to as *gradient descent*. Several optimization methods that perform gradient descent are usually readily implemented and available in any deep learning framework [66], e.g., *Stochastic Gradient Descent* (SGD) and Adam [67].

Another key concept of machine learning in general are the *hyperparameters*, which are design parameters not learnt by the network itself, and that are rather defined in forehand, e.g., the width and depth of a FFNN. They will affect a model's capability to properly fit the data, so an important part of machine learning is to find appropriate hyperparameters via trial and error. Hyperparameters can be model specific or training-algorithm specific. The two most important training hyperparameters that will appear throughout this work are the *learning rate* and the *batch size*, which are common to all gradient-based learning algorithms. The first determines the size of the update step of the network parameters and the second one is how many samples of the data are in one batch. Additionally, an epoch is when the network has seen all the batches that make up the training set and the training may require several epochs before the model can fit the data, so the maximum number of epochs to let the model run is also a hyperparameter. Model specific hyperparameters are, for instance, the aforementioned width and depth of a FFNN. Other relevant model-specific hyperparameters are described in each model's respective section.

Finally, in order to practically train a model (i.e., fit its parameters to data) and evaluate its performance, the data is often split in three subsets, called the *training*, *validation* and *test sets*. The objective is to learn a function f_θ with the training set, that can make predictions $\hat{y}$ that approximate the true label y as closely as possible for the test set, which is not revealed during training. The validation set is seen during training, but the model does not explicitly learn from it. It is mainly used for finding appropriate hyperpameters by evaluating the performance of the model on this set, while also controlling that the model indeed generalizes well and does not overfit the training data (i.e., by checking that the gap between the training and validation loss does not widen too much and that the validation loss does not start to increase).

3.2. Recurrent neural networks

Recurrent neural networks [68], are a special type of neural networks specifically designed for handling sequential data, such as text or time series. Essentially, they are a type of dynamical system that can read a sequence of data values one at a time and update their internal state recursively by a non-linear transformation of its previous state and the corresponding input value in the sequence. This internal state is called the *hidden state* and it is usually denoted by $\mathbf{h}$. These models, as any neural network, have trainable parameters which are learnt through backpropagation and as a characteristic trait, are shared across time. This trait allows these type of models to be represented as a single computational unit or recurrent cell that is applied recursively for reading a data sequence [65]. Examples of two different types of recurrent cells are depicted in Figure 3.2. Any RNN cell, follows the same principle, and reads a sequence by applying itself as many times as the sequence length. Mathematically, a generic RNN cell can be defined as:

$$\mathbf{h}_t = f_\theta(\mathbf{x}_t, \mathbf{h}_{t-1}), \tag{3.2}$$

where f_θ is the recurrent cell with learnable parameters. Each sample in the data $\mathbf{x}$ is a collection of $T+1$ ordered points: $\{\mathbf{x}_0, ..., \mathbf{x}_T\}$ such that $\mathbf{x}_t \in \mathbb{R}^v$ for $v \in \mathbb{N}$. Conversely, $\mathbf{h} \in \mathbb{R}^p$, where the dimensionality $p \in \mathbb{N}$ of the hidden state is a hyperparameter. An initial state $\mathbf{h}_0$ must also be provided, which usually consists of just zeros or small random numbers.

In its most basic form, an RNN cell [69] is represented as:

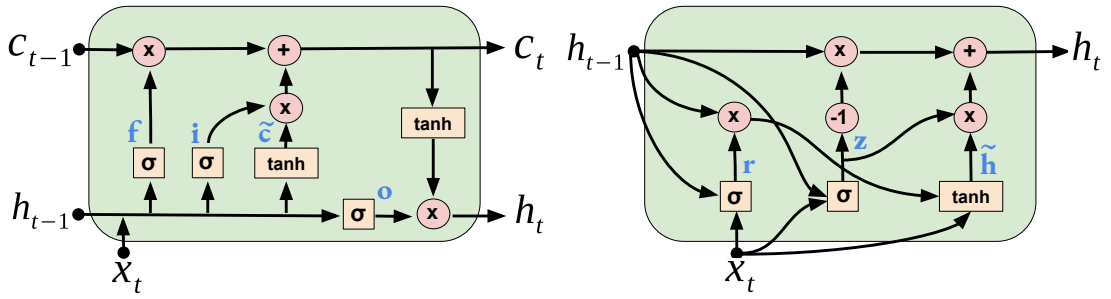
$$\mathbf{h}_t = \tanh(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}) \tag{3.3}$$

where tanh denotes the *hyperbolic tangent* as the activation function, $\mathbf{W}$ and $\mathbf{U}$ are respectively the input and recurrent parameters and $\mathbf{b}$ is the bias term [69]. In here, $\mathbf{W} \in \mathbb{R}^{p \times v}$, $\mathbf{U} \in \mathbb{R}^{p \times p}$, $\mathbf{b} \in \mathbb{R}^p$ and the activation function is applied element-wise, thus preserving dimensionality. Note that, as previously stated, $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{b}$ do not depend on time. To decode the information encoded in the final hidden state, $\mathbf{h}_T$, and making a prediction $\hat{y}$, often a simple learnable linear mapping (i.e., a singe linear layer; an analogy for this is to apply a final FFNN, as in Figure 3.1, without any hidden layers and without activation functions) is used as an output layer, which is external to the recurrent cell. To provide a prediction that can be interpreted as the probability of each sample in the data to be in each of the possible classes, the output is usually followed by a *softmax* function (*i.e.* normalized exponential).

The RNN baselines models used in this work have slightly more sophisticated architectures of recurrent cells, so each will have a dedicated subsection in this chapter.

3.2.1. Long short-term memory networks

The basic form of RNNs as presented before, can encounter difficulties while training with a gradient-based optimization method and struggle to remember long-term relationships



a LSTM b GRU

Figure 3.2.: Schematic diagram of an LSTM and a GRU cell structure. The yellow boxes denote the layers in each cell finalized by their respective activation function and the red circles denote point-wise operations, hence the $\odot$ product in the equations is denoted here by x for simplicity. Their respective gates are written in blue.

in the data when the sequences get longer [70]. These difficulties are namely the *vanishing gradient problem* and *exploding gradient problem*, which may as well occur in deep FFNNs. Each problem refers, respectively, to an exponential decrease and an exponential increase in the norm of the gradients, which make impossible for the network to learn correlations between long-term dependencies, because the backward flow of information about the loss becomes insufficient [71].

LSTM networks were specifically designed to address these limitations by adding another internal state called the *memory cell*, in addition to gates that can learn to control the inflow and outflow of information from the memory cell [72]. In the original LSTM design, these gates are the *input* and *output gates*. The input gate is designed to protect the memory cell from irrelevant perturbations in the input data by learning to override the input or to allow it to update the memory of the network. Conversely, the output gate protects the memory cell from perturbations in the output flowing back into the network's memory. Subsequent developments included a *forget gate* in the architecture. Such gate prevents the hidden and cell states to grow unbounded, by allowing the network to learn to reset its memory when needed [73]. Recent studies have confirmed the importance of the forget gate and also of the final activation function [74]. The following set of equations defines the architecture of a modern LSTM network:

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\tilde{\mathbf{c}}_t &= \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}
\tag{3.4}
$$

where $\odot$ denotes the Hadamard product (*i.e.* element-wise product), which preserves dimensionality; σ is the *logistic sigmoid* function; i, f and o respectively denote the input, forget and output gates and are all defined similarly to Equation 3.3, but with a different activation function; and $\mathbf{c}$ denotes the memory cell state variable. Essentially, the memory cell is updated by a weighted sum (modulated by the forget and input gate) of a new memory state candidate, $\tilde{\mathbf{c}}_t$, and the previous memory state $\mathbf{c}_{t-1}$, where $\tilde{\mathbf{c}}_t$ is calculated in the same way as a simple RNN (Equation 3.3). A representative diagram of the set of Equations 3.4 is presented in Figure 3.2a. Note how each gate directly affects the output of the respective point-wise multiplication that succeeds them, by controlling the amount of information that can continue its path for computing the new cell and hidden states.

3.2.2. Gated recurrent units

Gated recurrent units are a recently proposed simplified version of an LSTM [75]. They have only two gates, an *update gate* z and a *reset gate* r, thus reducing the number of parameters compared to a LSTM network, which has one more gate. Despite of their lower number of parameters, they have been shown to perform equally well or better than LSTMs for certain tasks [75]. A GRU architecture is defined as follows:

$$\begin{aligned}
\mathbf{z}_t &= \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1} + \mathbf{b}_z) \\
\mathbf{r}_t &= \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{b}_r) \\
\tilde{\mathbf{h}}_t &= \tanh(\mathbf{W}_h\mathbf{x}_t + \mathbf{U}_h(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h) \\
\mathbf{h}_t &= (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t
\end{aligned} \tag{3.5}$$

The update and reset gate are defined in the same way as the gates of an LSTM and there is no separate memory cell, thus the hidden state is directly exposed. The hidden state is a linear combination of the previous state $\mathbf{h}_{t-1}$ and a candidate activation or state $\tilde{\mathbf{h}}_t$, which in turn can have a varying access to the memory of the hidden state depending on the reset gate influence. As for the LSTM cell, a representative diagram is presented in Figure 3.2b. Note how the structure has less elements than LSTMs, but a more intricate circuit, which allows them to extract information from data more efficiently.

Although LSTM and GRU networks are effective in attenuating the vanishing and exploding gradient problems compared to a simple RNN cell, complex and long input sequences may still pose a threat to a proper training procedure. As a result, other techniques such as gradient clipping may be required to counteract it [71], [74].

3.3. Neural ordinary differential equations

Neural ordinary differential equations are a novel and increasingly popular type of deep learning model. In this section, first some background information is given about their

origin, then their definition is presented and their characteristics are described, followed by the definition of the ODERNN model.

3.3.1. Background

Although NODEs were only formally proposed in 2018 [29], the intuitions that led to their development come from observations made by different authors in recent years [76]–[79]. These observations pointed that the sequence of transformations performed by the layers of a *Residual Network* (ResNets [80], a successful recent type of neural networks), can be interpreted as a simple discretization of an *Ordinary Differential Equation* (ODE) [29], [81]. ResNets are FFNNs, but in addition to the current layer's transformation done by f_θ they also directly add a previous layer's state or activation, which is the key for them to be interpretable as an ODE. Equation 3.6 summarizes the dynamics of a ResNet block, in which $l \in \{0, ..., L\}$, represents a layer of the network [29], [80]. Note that, in the same way as for FFNNs, a key difference of Resnets with RNNs (Equation 3.2) is that the learnable parameters are not shared among layers (i.e., θ^l instead of θ).

$$\mathbf{h}_l = \mathbf{h}_{l-1} + f_{\theta^l}(\mathbf{h}_{l-1}) \tag{3.6}$$

With a simple manipulation of Equation 3.6 and by increasing the number of layers, or conversely by taking the discretization step to its infinitesimal limit, the classic definition of a derivative can be recovered, thus defining a NODE model [29], [82].

3.3.2. Definition

As previously described, NODE models can be regarded as a continuous-time analogue of ResNets [83]. Then, a neural ordinary differential equation can be defined as:

$$\frac{\mathbf{dh}(t)}{dt} = f_\theta(\mathbf{h}(t)) \tag{3.7}$$

where for some $p \in \mathbb{N}$, $f_\theta : \mathbb{R}^p \to \mathbb{R}^p$ is a neural network with parameters θ, hence the name: *neural* ODE. Similarly to RNNs, $\mathbf{h}$ defines the model's hidden state and its dimensionality p is a hyperparameter. As f_θ is usually just a FFNN, its width and depth will be additional hyperparameters in this model. Note that Equation 3.7 is like a regular ODE definition, with the exception that f here, is parameterized by a neural network instead of being an analytical function like in normal ODEs. Borrowed from ODE nomenclature, f is often referred to as the *vector field* in the literature. Even though f_θ does not necessarily needs the time to be included as an argument, it can be done, so that it may be included as an extra dimension in $\mathbf{h}$ as a form of augmentation if desired [29], [32].

Given an initial hidden state $\mathbf{h}(0)$, the solution $\mathbf{h}(T)$ to this differential equation, can be

obtained by solving the following initial value problem (IVP):

$$\begin{aligned} \mathbf{h}(0) &= \xi_\phi(\mathbf{x}) \\ \mathbf{h}(T) &= \mathbf{h}(0) + \int_0^T f_\theta(\mathbf{h}(\tau))d\tau \\ \hat{y} &= l_\psi(\mathbf{h}(T)) \end{aligned} \tag{3.8}$$

which can be solved with a fixed step ODE solver or with an adaptative-step ODE solver up to a desired accuracy. An example of their operation can be found in each particular segment in Figure 3.3 (i.e., without a final ξ_ϕ and l_ψ, as in the set of Equations 3.8), which are part of the ODERNN model introduced later in this section. The ODE solver in a NODE generates a continuous hidden state on-the-fly, while integrating Equation 3.7. The mathematical formalities in Equation 3.8 are the following. First, the variable τ in the integral represents the internal time variable of the ODE that is used for taking the discretization steps. Then for some $v, p \in \mathbb{N}$ representing the input and hidden dimensions respectively: $\xi_\phi : \mathbb{R}^v \to \mathbb{R}^p$ and $l_\psi : \mathbb{R}^p \to \mathbb{R}$ define additional learnable linear mappings with parameters ϕ and ψ, that are needed to respectively encode the input data into hidden dimensions and to decode the hidden state to make a prediction, and additionally, to ensure that NODEs does not present expressivity constraints, i.e., limits in their approximation capabilites [81], [82]. In particular, an initial linear layer acts as a form of augmentation for NODEs, that allows them to learn any $\mathbf{x} \to \hat{y}$ mapping [84]. Note that since in order to solve an ODE one must solve an IVP, in the set of Equations 3.8 the input data is forced to be encoded only in the initial value of the hidden state $\mathbf{h}(0)$.

3.3.3. Characteristics

NODEs do not have a depth explicitly defined like traditional FFNNs, but a good analogy is the number of function evaluations of the vector field that are performed by the ODE solver [29]. When using a fixed-step solver, a NODE is similar to a ResNet with as many layers as the number of time steps that are taken by the ODE solver. However, when using an adaptative-step ODE solver, it is free to evaluate the vector field f_θ wherever it needs to find the solution of the integral. In this latter case, a NODE can be referred to as a *continuous-depth model*, where the model becomes "increasingly deep" as it tries to fit the data [29].

The main advantages of having a model evaluated by an ODE solver, such as in NODEs, are the following:

- **Memory efficient backpropagation.** The gradients can be computed using the *adjoint sensitivity method* [85], which is more memory efficient (i.e., constant memory use regardless of the batch size) than traditional backpropagation, as it does not need to store the forward pass activations for the backward pass. Conversely, it can be slow, as it needs to solve a second ODE backwards in time to find the

gradients of the solution of the first ODE's loss w.r.t the parameters θ and its initial value $\mathbf{h}(0)$ (i.e., $\frac{\partial L}{\partial \theta}$ and $\frac{\partial L}{\partial \mathbf{h}(0)}$) [29], [82].

- **Continuously defined dynamics.** The model is defined by an ODE, thus the hidden state is continuously defined. This means that this type of model is theoretically well suited for modelling phenomena that follow a continuously evolving underlying process (e.g., the phenological development of crops). Even though the variable t in an ODE (Equation 3.7) represents time, it is just an internal detail of the model and has no relationship to the timestamps of some time series input data for example. In fact, as previously mentioned, a NODE model on its own cannot model sequential input, such as time series. Nevertheless, the model can be used as an independent building block and coupled with other models that are able to process sequential data, in order to take advantage of its continuously-defined dynamics [29].

3.3.4. The ODERNN model

NODEs are similar to RNNs in the sense that they have shared parameters among their "layers", but they lack the ability to incorporate input data at each step and are entirely defined by the initial conditions of the ODE. In the other hand, RNNs only have discrete dynamics, which means that their hidden state is not defined between the update steps, whereas as previously stated, a NODE's hidden state is continuously defined.

A model that couples both of these types of models, called ODERNN, has been recently proposed [30]. This model is comparable to other modelling attempts that allow an RNN's hidden state to evolve continuously between updates of incoming sequential data, such as exponentially decaying the hidden state between observations [86]. An ODERNN model can be summarized as follows:

$$\begin{aligned} \tilde{\mathbf{h}}_t &= \text{ODEsolve}(f_\theta, \mathbf{h}_{t-1}, [0, \Delta_t]) \\ \mathbf{h}_t &= \text{RNN}(\mathbf{x}_t, \tilde{\mathbf{h}}_t) \end{aligned} \tag{3.9}$$

where RNN symbolically denotes Equation 3.2 with any variant of RNNs, which in this work is a GRU cell. Complementary, ODEsolve denotes an ODE solver that solves an IVP with integrand f_θ and initial value $\mathbf{h}_{t-1}$ between the limits $[0, \Delta_t]$, where Δ_t represents the time difference between observations $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$. An explanatory diagram can be found in Figure 3.3. The simplest version of an ODERNN model is constituted of one RNN cell and subsequently solves one NODE (as in the set of Equations 3.8, but without the input and output linear mappings) between each of the RNN input data readings, i.e., where the discontinuities in the hidden state are. Note that the hidden state must be shared between each NODE and the RNN, which means that the hidden state gets interrupted every time there is incoming data and gets overridden by the output of the RNN. From a mathematical perspective, their definition and operation is like the one of an RNN model, with the sole difference that between each of the RNN's update steps,

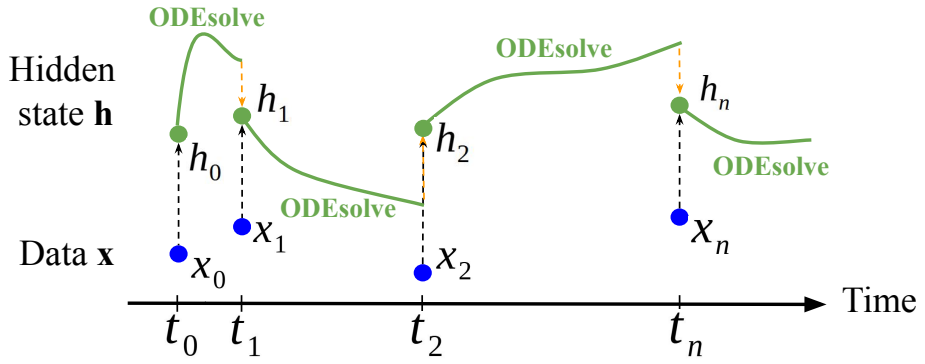


Figure 3.3.: Schematic diagram of a ODERNN model. The hidden state that is continuously defined by a NODE, gets interrupted by an RNN update when there is incoming data, which are represented by the black dotted arrows. After each RNN update is made, a new NODE is defined and solved.

$\mathbf{h}_{t-1}$ is allowed to evolve as a NODE and then the result of this evolution $\tilde{\mathbf{h}}_t$ is fed to the RNN, instead of $\mathbf{h}_{t-1}$ as usual.

Similarly to the LSTM and GRU models, the final hidden state $\mathbf{h}(T)$, encodes all information about the input sequence and can be decoded to make predictions, with an additional final linear layer, as l_ψ in the set of Equations 3.8.

3.4. Neural controlled differential equations

Neural controlled differential equations were recently introduced as a related model to NODEs that can address the limitation of incorporating incoming information, and thus are able to process sequential data by themselves [32], [83]. *Controlled differential equations* (CDEs) belong to the relative small field of mathematics called *rough analysis* [33] and they are a type of differential equations whose vector field trajectories continuously depend on a time-varying function $\mathbf{X}$ (i.e., the *control*), rather than only on the initial value like in an ODE does.

3.4.1. NCDE definition

In order to define a NCDE, the control $\mathbf{X}$ or also called the *data path*, must first be defined. Given a time series of multi-dimensional data points, $\mathbf{X}$ is a continuous approximation of the values of each of the separate data dimensions in time, which can be obtained by simple linear interpolation, for instance. This means that given a collection of data points

and their corresponding timestamps $\mathbf{x} = ((t_0, \mathbf{x}_0), (t_1, \mathbf{x}_1), ..., (t_n, \mathbf{x}_n))$, where $\mathbf{x}_i \in \mathbb{R}^{v-1}$ and $t_i \in \mathbb{R}$, then $\mathbf{X}$ is defined as a continuous function that maps any time t between t_0 and t_n to values in the data dimensions, i.e., $[t_0, t_n] \to \mathbb{R}^v$. Methods for constructing $\mathbf{X}$ and some details about their requirements are discussed in section 3.4.3.

The definition of a NCDE in differential form is a straightforward modification to that of a NODE (recall Equation 3.7) and is denoted as:

$$\frac{\mathbf{dz}(t)}{dt} = f_\theta(\mathbf{z}(t))\frac{\mathbf{dX}(t)}{dt} \tag{3.10}$$

where f_θ is also a vector field parameterized by a neural network. In here, $\mathbf{z}$ denotes the hidden state, and the change of notation from $\mathbf{h}$ is just done to keep consistency with the original paper and to differentiate it from the other models, despite of representing the same. Note that NCDEs relate to CDEs in the same way as NODEs do to ODEs, so a CDE is defined identically to Equation 3.10, but with f not necessarily being learnt from data.

After creating $\mathbf{X}$, the full mapping from input data to prediction with a NCDE can be defined in a similar way to NODEs, by integrating Equation 3.10 and including input and output mappings as follows:

$$\begin{aligned} \mathbf{z}(t_0) &= \xi_\phi(\mathbf{X}(t_0)) \\ \mathbf{z}(t_n) &= \mathbf{z}(t_0) + \int_{t_0}^{t_n} f_\theta(\mathbf{z}(s))\mathbf{dX}_s \\ \hat{y} &= l_\psi(\mathbf{z}(t_n)) \end{aligned} \tag{3.11}$$

where similarly to the set of Equations 3.8, $\xi_\phi : \mathbb{R}^v \to \mathbb{R}^p$ and $l_\psi : \mathbb{R}^p \to \mathbb{R}$ are learnable linear mappings for a hidden state $\mathbf{z}$ of size $p \in \mathbb{N}$, and $f_\theta : \mathbb{R}^p \to \mathbb{R}^{p \times v}$ is a FFNN [32], [83]. Therefore, as in an ODERNN model, a NCDE has at least three model-specific hyperparameters: the dimensionality p of the hidden state, and the width and depth of the vector field f_θ. Note that for NCDEs, time must be appended as a channel in the data before creating $\mathbf{X}$, which is needed to avoid translational invariance [32] and to prevent loss of information, because the model does not necessarily need the original timestamps as explained in section 3.4.4.

The integral in the set of Equations 3.11 is a generalized case of a classic integral, in which the discretization variable goes through a function, and it is called a Riemann-Stieltjes integral [87]. When $\mathbf{X}$ is differentiable, like in the considered case, it can be directly evaluated as: $\mathbf{dX}_s = \frac{\mathbf{dX}}{ds}(s)ds$. Then, Equation 3.11 may be redefined as follows:

$$\begin{aligned} \mathbf{z}(t_0) &= \xi_\phi(\mathbf{X}(t_0)) \\ \mathbf{z}(t_n) &= \mathbf{z}(t_0) + \int_{t_0}^{t_n} f_\theta(\mathbf{z}(s))\frac{\mathbf{dX}}{ds}(s)ds \\ \hat{y} &= l_\psi(\mathbf{z}(t_n)) \end{aligned} \tag{3.12}$$

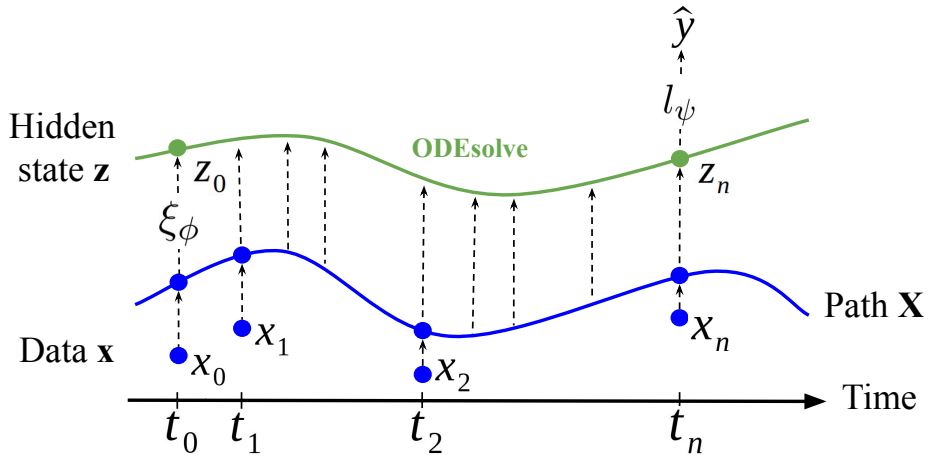


Figure 3.4.: Schematic diagram of a NCDE model. The continuous input data path $\mathbf{X}$ is constructed by approximating the time series data points, by cubic splines interpolation for instance. The hidden state is defined by a CDE, which is an ODE with a vector field that is continuously controlled by the derivative of the data path $\mathbf{X}$.

The integral here now is like a regular one in a NODE, but where instead of a vector field f_θ, a *controlled vector field* is integrated, which can be denoted as $g_{\theta,\mathbf{X}}(\mathbf{z}, s) = f_\theta(\mathbf{z})\frac{\mathbf{dX}}{ds}(s)$. This last equation is a matrix-vector product, which after computing may be directly solved with the same tools as for solving NODEs and just requires passing $g_{\theta,\mathbf{X}}$ rather than f_θ to the ODE solver. Note that $g_{\theta,\mathbf{X}}$ is also equivalent to the right hand side of Equation 3.10. Figure 3.4 presents a simplified visual representation of the operation of a NCDE. First, a continuous input path is constructed from time series data. Then, the NCDE is initialized by sampling the data path $\mathbf{X}$ at the first timestamp value, before it proceeds to solve an ODE with a controlled vector field, i.e, controlled by the derivative or slope of the data path $\mathbf{X}$. Finally, the prediction $\hat{y}$ is obtained by applying a linear mapping to the terminal hidden state value of the NCDE.

3.4.2. Stacked NCDE

The S-NCDE model is implemented in this work as a novel extension to a simple NCDE. This model implies solving two CDEs. In a differential form, this accounts to solve the following system of equations:

$$\begin{aligned} \frac{\mathbf{dz}(t)}{dt} &= f(\mathbf{z}(t))\frac{\mathbf{dX}(t)}{dt} \\ \frac{\mathbf{du}(t)}{dt} &= g(\mathbf{u}(t))\frac{\mathbf{dZ}(t)}{dt} \end{aligned} \tag{3.13}$$

This means that, w.r.t. the set of Equations 3.12, one needs to solve two different integrals rather than one. To this end, the chosen implementation consists of solving the two CDEs subsequently by interpolating the entire solution sequence $\mathbf{z_t} = (\mathbf{z}_0, ..., \mathbf{z}_{t_n})$ of the first CDE to build a new continuous path or control $\mathbf{Z}$ in a similar way than the first control $\mathbf{X}$ is built from the input data. This second control is then used as a continuous-time path to control the second CDE, whose terminal value is decoded into a prediction as usual. In particular, a S-NCDE model can be defined for $t \in [t_0, t_n]$ as follows:

$$\begin{aligned} \mathbf{z}(t_0) &= \xi_\phi(\mathbf{X}(t_0)) \\ \mathbf{z}(t) &= \mathbf{z}(t_0) + \int_{t_0}^{t} f_\theta(\mathbf{z}(s)) \frac{\mathbf{dX}}{ds}(s) ds \\ \mathbf{u}(t_0) &= \zeta_\upsilon(\mathbf{Z}(t_0)) \\ \mathbf{u}(t_n) &= \mathbf{u}(t_0) + \int_{t_0}^{t_n} g_\omega(\mathbf{u}(s)) \frac{\mathbf{dZ}}{ds}(s) ds \\ \hat{y} &= l_\psi(\mathbf{u}(t_n)) \end{aligned} \tag{3.14}$$

In here, the new terms w.r.t. Equation 3.12 are: $\mathbf{Z} : [t_0, t_n] \to \mathbb{R}^p$, which is a continuous-time representation of the solution sequence $\mathbf{z_t}$ of the first CDE; $g_\omega : \mathbb{R}^p \to \mathbb{R}^{p \times p}$, which is the vector field of the second CDE, parameterized by a FFNN with learnable parameters ω; and $\zeta_\upsilon : \mathbb{R}^p \to \mathbb{R}^p$ is an additional linear mapping that learns the initial value of the second CDE in a similar way to what ξ_ϕ does for the first CDE. Figure 3.5 provides a visual representation of the model. The second CDE is stacked on top of the first one, and operates in the same way as the bottom one, but with the discrete solution sequence of $\mathbf{z}$ taking the role of the data points on the first CDE. A continuous path of $\mathbf{z}$ must be reconstructed, in order to control the second CDE.

Finally, note that for simplicity, this particular implementation defines the same dimensionality for the hidden states $\mathbf{z}$ and $\mathbf{u}$, however this does not need to be the case in general. Additionally, the method considered for constructing $\mathbf{Z}$ can be the same as the one for creating $\mathbf{X}$, like it is the case for the experiments of this thesis, but also does not need to be the case in general.

3.4.3. Continuous data representation methods

The possible methods for constructing $\mathbf{X}$ that are considered in this work are: linear interpolation, natural cubic splines interpolation and squared exponential kernel regression. For every method the goal is to obtain a continuous representation of the input data such that $\mathbf{X}(s)$ is defined $\forall s \in [t_0, t_n]$, where $\{t_0, ..., t_n\}$ is the set of all timestamps of the input observations. The multi-dimensional bold notation for X is purposely left aside henceforth in this section, as each method is applied only to 1-D data, i.e., applied separately in each dimension of the data. A brief summary and references for each

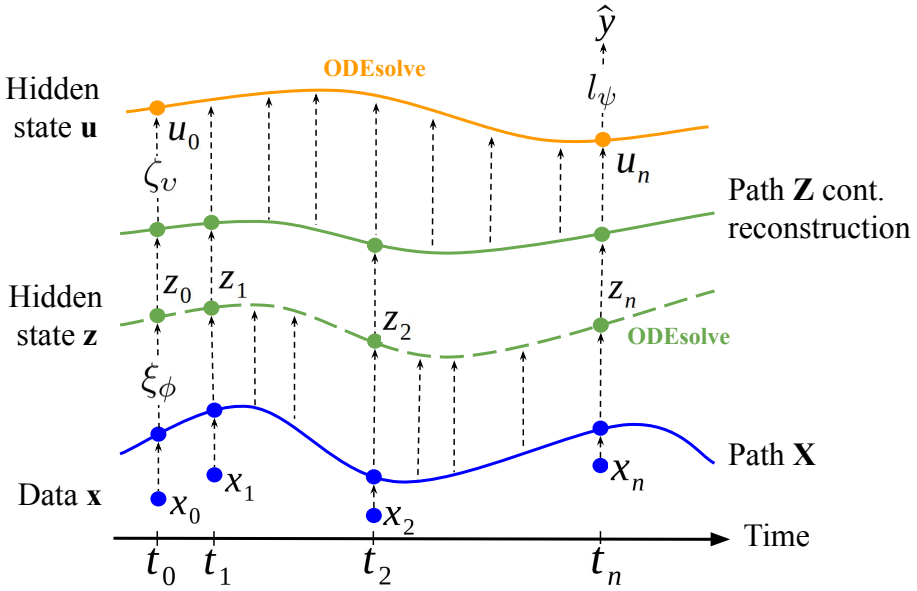


Figure 3.5.: Schematic diagram of a S-NCDE model. Two hidden states coexist and are defined by two different CDEs. The data path **Z** takes samples of **z** in alignment with the timestamps, to build a continuous representation of it to control **u**.

method are given in what follows.

- **Piece-wise linear interpolation.** This is the simplest form of interpolation, it produces a linear segment between each pair of consecutive points in each time series. The continuous linear segments are defined by Equation 3.15, where its derivative is just the slope a_i of each segment (i.e., $\frac{dX_i}{ds}(s) = \frac{x_{i+1}-x_i}{t_{i+1}-t_i} = a_i$).

$$X_i(s) = x_i + (s - t_i)\frac{x_{i+1} - x_i}{t_{i+1} - t_i} = x_i + (s - t_i)a_i \qquad \text{, for } s \in [t_i, t_{i+1}] \tag{3.15}$$

- **Piece-wise natural cubic spline interpolation.** Natural cubic splines are a third-order polynomial interpolation, computed piece-wise like the linear interpolation. Each segment or piece i of the interpolation can be defined by Equation 3.16 between each pair of knots, which exactly match the original data, like it is the case of linear interpolation. The coefficients of each segment can be found by solving a tri-diagonal system of equations [88]. Its derivative is straightforward to compute and it is given by Equation 3.17.

$$X_i(s) = a_i + b_i s + c_i s^2 + d_i s^3 \qquad \text{, for } s \in [t_i, t_{i+1}] \tag{3.16}$$

$$\frac{dX_i}{ds}(s) = b_i + 2c_i s + 3d_i s^2 \qquad \text{, for } s \in [t_i, t_{i+1}] \tag{3.17}$$

- **Squared exponential kernel regression.** This is a form of curve fitting[i] via *Gaussian Process* (GP). The objective of this method is to model the data points, perturbed by some noise which is normally distributed, by the following equation:

$$x = f(t) + \epsilon \qquad \text{, } \epsilon \overset{i.i.d}{\sim} \mathcal{N}(0, \sigma_n^2) \tag{3.18}$$

  It does so by modelling a joint likelihood distribution of the points with a *covariate function* or *kernel*, which, in this case, is the squared exponential kernel (i.e., SE kernel, also known as radial basis function kernel). Other than being a very common choice, the key advantage of this kernel is the fact of being differentiable [89]. The SE kernel for this setting is defined as:

$$\kappa(t_i, t_j) = \sigma_f^2 \exp\left\{ \frac{(t_i - t_j)^2}{2L^2} \right\} + \sigma_n I \qquad \text{, } \forall\, t_i, t_j \in \{t_0, ..., t_n\} \tag{3.19}$$

  where σ_f, L and σ_n are hyperparameters defining the signal variance, the length-scale and the noise variance, respectively. The curve fitting coefficients will be given by α in Equation 3.20, where $\mathbf{K}$ is the covariance matrix built from all the observations at timestamps $\{t_0, ..., t_n\}$ and $\mathbf{I}$ is the identity matrix.

$$\alpha = (\mathbf{K} + \sigma_n \mathbf{I})^{-1} \mathbf{x} \tag{3.20}$$

  Note that after fitting the data points, the result is a posterior probability in the space of continuous functions whose point-wise values follows a multivariate normal distribution defined by a mean and a covariance matrix. However, since NCDEs can not process probabilistic inputs, in this case only the mean of the posterior distribution is recovered. Therefore, the mean approximation of the data path X is defined by Equation 3.21 [89] and its derivative by Equation 3.22.

$$X(s) = \sum_{i=0}^{n} \alpha \kappa(t_i, s) \qquad \text{, for } s \in [t_0, t_n] \tag{3.21}$$

$$\frac{dX}{ds}(s) = \sum_{i=0}^{n} \alpha \nabla \kappa(t_i, s) = \sum_{i=0}^{n} \alpha \sigma_f^2 \kappa(t_i, s) \frac{(t_i - s)}{L^2} \qquad \text{, for } s \in [t_0, t_n] \tag{3.22}$$

A separate interpolation has to be done for each time series of every example in the

[i] Although it could also be enforced to exactly match the data points, in which case it would be strictly speaking an interpolation.

input data, which means that if the batch of input data is of size B and has v dimensions or features, there will be $B \times v$ different interpolations to be performed for that batch. In addition, there are some considerations to uphold depending on the type of methods used. For example, the control $\mathbf{X}$ of a NCDE should be twice-differentiable if the adjoint sensitivity method is being used for backpropagation [32]. When this is not the case $\mathbf{X}$ will not be smooth (e.g., like with linear interpolation) and the adjoint backpropagation method with an adaptive ODE solver can be undesirable slow. However this can be avoided in practice by explicitly telling the ODE solver where the kinks are, so it avoids evaluating the function at this points where the derivative has discontinuities [32]. Additionally, all this methods except for the linear interpolation method are non-causal, which means that they need to look at future points to approximate the whole signal. When using NCDEs for inference, this means that only the linear interpolation method can be used, as long as there are no missing values in the data. In the case there are missing values, normal linear interpolation would also not work, and some minor adjustments to this method must be made.

3.4.4. Properties

A relevant property that CDEs possess is *reparametrization invariance* [90], which implies that the terminal value or solution of a CDE is invariant to reparametrizations of the continuous data path. This means that, theoretically speaking, one could build the input data path with any array of timestamps, regardless of the original time series regularity, as long as the original timestamps are appended as a channel or feature in the data, to ensure that no information is lost. In fact, this is the standard practice with NCDEs.

3.5. Software

All the code is written in Python and the models are implemented in *Pytorch* [91], a deep learning library which offers automatic differentiation. NODEs are solved with *torchdiffeq* library [29] and NCDEs with *torchcde* library [32], which is a wrapper library around torchdiffeq to compute NCDEs and that also offers built-in controls such as linear, rectilinear and natural cubic splines interpolation. All the code implemented for this work can be found in GitHub: `https://github.com/JoaquinGajardo/NeuralCDEcrops`.

Chapter 4

Data

The two multi-temporal remote sensing datasets considered for evaluating the models are presented in this chapter. A detailed separate description is provided for both of the datasets, followed by a section describing the common data pre-processing methodology.

4.1. TU Munich dataset

This dataset, henceforth referred to as the TUM dataset, was proposed in [15] and consists of a time series of 26 multi-spectral satellite images from Sentinel-2A, collected over a 102 km $\times$ 42 km area in the north of Munich, Germany, which has homogeneous agricultural, geographical and climate conditions. The images were retrieved over the same area for 26 different dates, between December 31$^{\text{st}}$, 2015 and August 29$^{\text{th}}$, 2016. There are four bands or channels originally available at 10 m *ground sampling resolution* (GSR), the B2 (blue), B3 (green), B4 (red), B8 (near infrared) bands and two 20 m GSR bands, which are the B11 and B12 bands (short-wave-infrared 1 and 2 respectively) and were upsampled to 10 m GSR by nearest-neighbour interpolation. The original data includes 19 ground truth classes, provided by the *Bavarian Ministry of Agriculture* in the form of field geometries: 18 different crop types with their respective names, plus an additional class *other* for the case where no field geometry was available. The crop classes are: *corn, meadow, asparagus, rape, hops, summer oats, winter spelt, fallow, winter wheat, winter barley, winter rye, beans, winter triticale, summer barley, peas, potatoes, soybeans* and *sugar beets*. The data was atmospherically corrected with the SEN2COR toolbox [92] for obtaining the bottom-of-atmosphere (BOA) reflectance and to additionally obtain a clouds or bad weather binary mask. Patches of 3×3 pixels are considered as additional features, which after flattening makes up a total dataset of roughly 350000 pixels time series of time length 26 and feature dimension 54. Labels are one-hot soft-assignments (i.e. a probability for each class), because bordering pixels were weighted, for preventing hard-assignments and loss of information, although most of them are zeros and ones.

The dataset is divided into a train, validation and test sets and was further pre-processed by [31], from whom the data was directly sourced for this work. In particular, data points without any labels and those with only one valid observation in the time series were removed. The uninformative pixels were masked out as zeros with the help of the clouds mask (i.e. multiplying the input data with the clouds mask element-wise). Additionally,

the dataset contained a label for each pixel at each different time, so they were reduced to one label per pixel for the entire time series by summing in the time dimension and normalizing, which is possible because of the one-hot encoding format of the labels. The final train, validation and test sets sizes are 287858, 55740 and 1499, respectively, thus accounting for roughly a 85%/15% split between the train and validation sets. The test set was discarded due to its small size, and because it does not contain all of the classes. Figure 4.1 presents the classes distribution for each set.

The dataset is further pre-processed in this work by removing the last time step, which only contains cloudy observations. In addition, a reduced-features version of the dataset is considered in most of the experiments, which excludes the neighbourhood and thus only considers the 6 bands from the central pixel as features. The reason for this is discussed in Chapter 5.

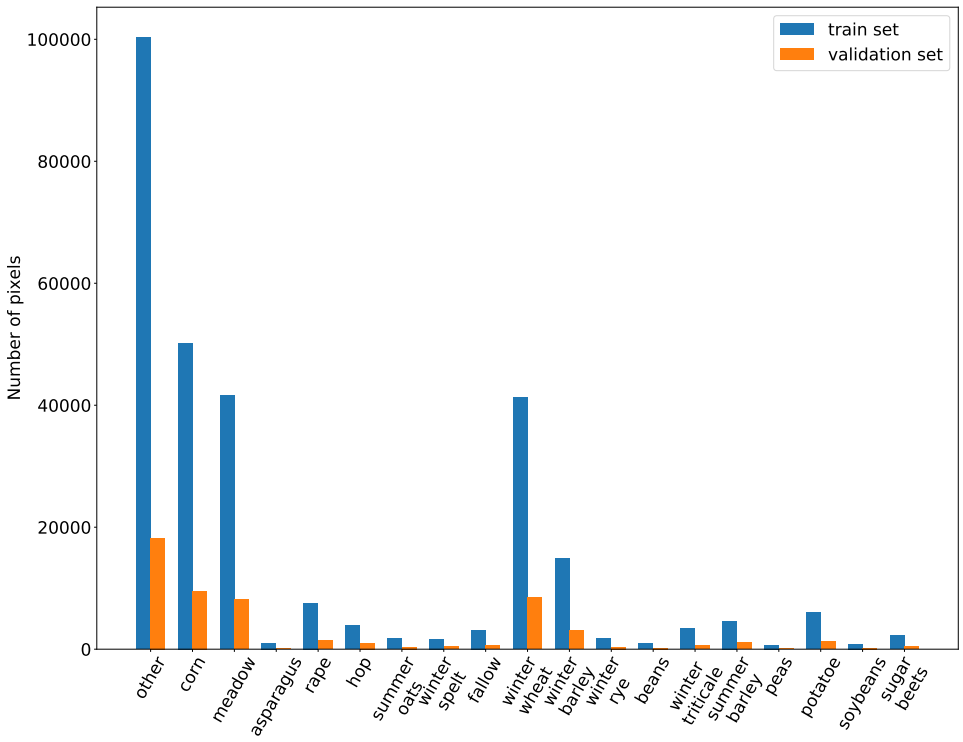


Figure 4.1.: TUM dataset class distribution.

4.2. Swisscrop

This dataset was gathered by [41] and used in [31]. It contains multi-spectral Sentinel-2A satellite images collected over an area of 50 km × 48 km in the Cantons of Zürich and Thurgau, Switzerland, at 142 different times between January 2019 and December 2019. The original dataset contains over a 100 different crop classes, with ground truth provided by the *Swiss Federal Office of Agriculture* and it is highly imbalanced, meaning that there are many classes which have few instances. Additionally, the dataset is of massive size, accounting for more than 400 GB of memory. Therefore, since the aim of this work is exploratory, just a sample of the dataset is used in order to speed up the development. The dataset considered has 52 possible labels, and only four channels per pixel, the red, green, blue and near-infrared bands, without considering a neighbourhood. As with TUM dataset, the data has been previously atmospherically corrected with the SEN2COR toolbox [92]. Moreover, the majority classes have been down-sampled to 10000 instances, thus producing a fairly balanced dataset, with a few exceptions. The train and validation sets splits are 75%/25% and the classes distribution for both sets are shown in Figure 4.2. The class names are: {0: *unknown class 0*, 1: *apples*, 2: *beets*, 3: *berries*, 4: *biodiversity area*, 5: *buckwheat*, 6: *chestnut*, 7: *chicory*, 8: *einkorn wheat*, 9: *fallow*, 10: *field bean*, 11: *forest*, 12: *gardens*, 13: *grain*, 14: *hedge*, 15: *hemp*, 16: *hops*, 17: *legumes*, 18: *linen*, 19: *lupine*, 20: *maize*, 21: *meadow*, 22: *mixed crop*, 23: *multiple*, 24: *mustard*, 25: *unknown class 1*, 26: *oat*, 27: *pasture*, 28: *pears*, 29: *peas*, 30: *potatoes*, 31: *pumpkin*, 32: *rye*, 33: *sorghum*, 34: *soy*, 35: *unknown class 2*, 36: *spelt*, 37: *stone fruit*, 38: *sugar beet*, 39: *summer barley*, 40: *summer rapeseed*, 41: *summer wheat*, 42: *sunflowers*, 43: *tobacco*, 44: *tree crop*, 45: *vegetables*, 46: *vines*, 47: *unknown class 3*, 48: *wheat*, 49: *winter barley*, 50: *winter rapeseed* and 51: *winter wheat*}. The four unknown classes (i.e., 0, 25, 35 and 47) are entirely missing in this sample dataset, and some classes such as chestnuts and gardens have very few examples.

Given that the length of the time series is fairly large, the time series are down-sampled in time for the experiments by a given factor, e.g., a time down-sample factor of two, means that only every second step in the time series is considered, thus only considers 71 steps. A mask with the % of cloud coverage is also provided and a threshold of 10% is used for considering a pixel as cloudy, thus producing a binary clouds mask. As with the TUM dataset, uninformative pixels due to clouds or bad weather are represented as zeros by masking the data with the clouds mask.

4.3. Pre-processing

The common pre-processing for both datasets consists of appending the timestamps as an extra channel, as well as adding an additional channel with the observational mask. The observational mask consists of only one channel because, for either dataset, whenever there is a missing value in one of the channels at a given time in a given sample, every other channel was also missing. Finally, the data is normalized by subtracting the mean

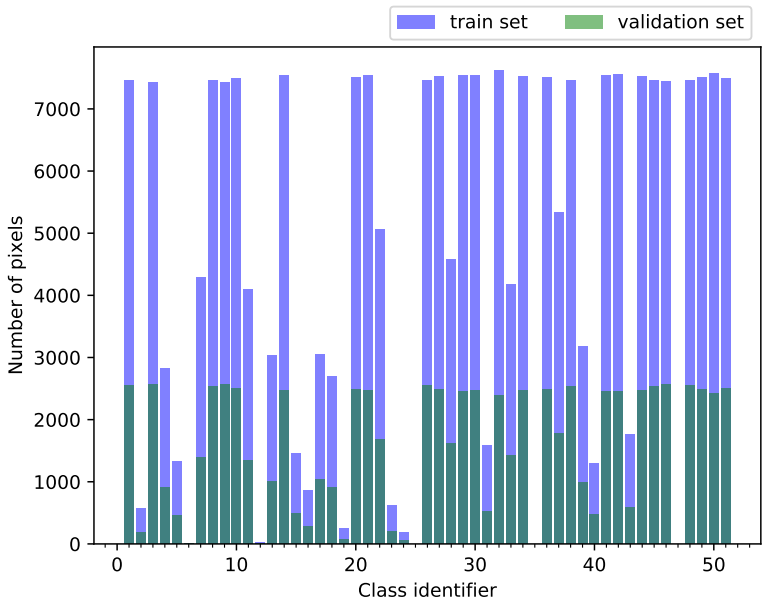


Figure 4.2.: Swisscrop dataset class distribution.

and dividing by the standard deviation on a per-channel basis. For NCDEs, the pre-processing requires some additional steps, detailed in the torchcde examples in GitHub, at `https://github.com/patrick-kidger/torchcde`. In particular, missing values are represented as NaNs (not-a-number), so that they can be detected by torchcde and skipped during the interpolation, and the observational mask that is appended as an extra channel is a cumulative one, because it will be reduced to a regular binary observational mask when taking the derivative of the interpolation path $\mathbf{X}$ as in Equation 3.10, which finally determines the vector field. There is a final caveat, whenever there are missing values at the beginning or end of the time series. It is important that $\frac{\mathbf{dX}}{dt}$, is as close to zero as possible at this locations, because the hidden state $\mathbf{z}$ should not continue evolving in this cases as there is no real observations, and a value of $\frac{\mathbf{dX}}{dt}$ equal to zero would ensure it. Whenever there are missing values at the end or at the beginning of a time series, the interpolation methods available at torchcde automatically pad (i.e., propagate) the last observed value towards the end and the first observed value towards the beginning before proceeding with the interpolation. For linear interpolation, even when just copying the last and first observed value and inserting them respectively at the end and the beginning of the time series, it is ensured that the derivative would be exactly zero, because the interpolation there would be an horizontal line. However, for natural cubic splines, padding all missing time step values located at the beginning and end of the time series is particularly important, as, otherwise, in order to ensure smoothness the spline derivative can remarkably deviate from being zero. With this procedure, the derivative of cubic splines is still not guaranteed to be exactly zero towards both ends,

but it is in general much closer to it. Therefore, since SE kernel regression method is not part of torchcde, the pre-processing includes a final step to pad forward and backwards the data as it was just described, to mimic the operation of the library. Hence, the pre-processing implemented can be summarized as follows:

Step 1: Represent missing values as NaNs for NCDEs, while keeping them as zeros for the baselines.

Step 2: Append time as the first channel.

Step 3: Append observational mask as an extra channel: a cumulative one for NCDEs and a regular binary one for the baselines.

Step 4: Normalize each channel to zero mean and standard deviation equal to one.

Step 5: For NCDEs pad forward the last observed value of each time series and backwards the first one, including the time channel.

Chapter 5

Results

In this chapter the experimental findings of this work are presented. In section 5.1 the experimental setup is described, including the hyperparameters selected for each model and the different metrics used for comparing the results. In section 5.2 the base results of each model for each dataset are exposed, with an additional analysis of the per-crop performance obtained by the models. In section 5.3, the experimental results of four ablation studies are presented, which highlight the effect of important configuration choices for the NCDE and S-NCDE models, such as the interpolation method used, the inclusion of an observational mask as a feature in the data, the impact of a high-dimensional dataset on the NCDE and S-NCDE models' architecture and performance, and the effect of including cloudy observations in the data.

5.1. Experimental setup

The baselines models used for the experiments are the GRU, LSTM and ODERNN methods, which are all RNN-based models, as described in Chapter 3. The models evaluated against the baselines are the NCDE model, and a stacked variant of a NCDE, i.e. the S-NCDE model. The main metrics used for comparing the models on the different experiments are the overall accuracy and the average per-class F1-score (i.e. the harmonic mean between precision and recall) of each individual pixel classified by the models. The overall accuracy provides a general metric of performance, but it tends to be biased to the dominant classes, whereas the average per-class F1-score allows for a better assessment of the models' capabilities for distinguishing every class in the dataset. Additionally, the number of parameters of each model is reported in each table of results. The results are reported on the validation set, because there was no appropriate test set available for both datasets, as described in Chapter 4. Note that this induces a potential bias in the results, but since no exhaustive hyperparameter optimization is performed, this bias is considered to be rather small and the validation set can serve the purpose of comparing the models in the different studies. All the experiments, with the exception of those in section 5.3.4, have cloudy pixels represented as NaNs for the NCDE and S-NCDE models, and as zeros for the baselines, as detailed in Chapter 4.

The configuration used for training the models is the following. The Adam optimizer [67] was used, which is a robust stochastic gradient-based optimization algorithm. A cross-entropy loss function was used, applied to the softmax of the output of each model.

For all the models, unless otherwise stated, an initial learning rate of 0.001 was used on the optimizer. Following the setup of [32], the learning rate was reduced when the performance failed to improve after a certain number of epochs and here the learning rate was multiplied with a constant factor of 0.1 when the F1-score on the validation set did not improve after two epochs. The training was done for a maximum of 40 epochs, with *early stopping* [93] when the validation F1-score failed to improve after five epochs. The model parameters at the epoch that achieved the best validation F1-score are the ones kept and saved as the trained model, and thus the ones used for reporting the results. The use of this methodology reduces training time and acts as a form of regularization, i.e., prevents overfitting [65].

A simple per-model hyperparameter analysis for each dataset was performed and can be found in the Appendix A. This analysis lead to the following hyperparameters choices for each model. For the TUM dataset, the GRU and LSTM baselines used a batch size of 256 and a hidden state size of 256. The ODERNN baseline had a batch size of 512, and a hidden state size of 256, while the vector field is implemented as a FFNN with width set to 256 units and with a tanh activation function after each linear layer, as implemented in [30] and [32]. The NCDE model used a batch size of 1024 and its hidden state size is set to 128, while the vector field is implemented as in [32], with a FFNN, here of three hidden layers of 256 units. A ReLU (i.e. rectified linear unit) activation function was used after every layer, with exception of the final one which was followed by a tanh activation function. Lastly, the S-NCDE model used a batch size of 512, while the two vector fields implemented were similar to those of a NCDE model, i.e., FFNNs of two hidden layers with width set to 128. For the Swisscrop dataset, the hyperparameters influenced the performance much more so than for the TUM dataset, and since only one run was performed for each combination of values, the following hyperparameters choices may not necessarily be as robust as for the TUM dataset. For the GRU, LSTM and ODERNN baselines an initial learning rate of 0.01 was used. As for TUM dataset, the GRU and LSTM model used a hidden state size of 256, while the batch size was set to 1024 and 512 for GRU and the LSTM models, respectively. The vector field of the ODERNN, NCDE and S-NCDE models are implemented in the same way as for the TUM dataset, however there are some slight differences in the hyperparameters values. For the ODERNN model, a batch size of 1024 and a hidden state size of 256 were used, while the vector field had one hidden layer with a width of 128. The NCDE model used a batch size of 1024, a hidden state size of 128, and its vector field was made of two hidden layers and 256 units. Finally, the S-NCDE model used a batch size of 512, a hidden state size of 32 for both hidden states, while both of the vector fields had one hidden layer with 128 units.

The dopri5 adaptative ODE solver (i.e., a variant of the Dormand-Prince method [94], which is the default option of torchcde and torchdiffeq) was used for ODERNN, NCDE and S-NCDE, with a relative error tolerance of 10^{-4} and an absolute error tolerance of 10^{-6}, while the adjoint method was used for computing the gradients in the backward pass. Additionally, a semi-norm [95] was used for the adaptative ODE solver as it allowed

for a faster training in most cases. All the hidden states of the baselines were zero-initialized. All models parameters were initialized with the default Pytorch settings, which uses a uniform He initialization [96] for linear layers and a uniform initialization for all the RNN network's parameters.

5.2. Model performance

In this section the main performance results of the models on each dataset are reported. First, the base case results are exposed, followed by an analysis of the performance of the models on each crop class. For the NCDE and S-NCDE models the linear interpolation method is used with an equally spaced timestamps array (i.e., the default option of torchcde). Additionally, the reduced-features version of the TUM dataset is used for all the experiments in this section and in the rest of the thesis, with exception of the study in 5.3.3. This is done because NCDEs present severe problems when the dataset is high-dimensional, which is put in evidence in section 5.3.3 and discussed in Chapter 6. Due to time constraints, every time series of the Swisscrop dataset was downsampled to length 36, as it was the case in the hyperparameters analysis too. Finally, the observational mask was always appended as an additional feature, as explained in Chapter 4. All these particular choices are later validated in section 5.3.

5.2.1. Base case

The main results for each model on the TUM dataset are presented in Table 5.1. As with every other table of results from henceforth, the model that achieved the best performance is marked in bold numbers. The best results are obtained by the ODERNN model, outperforming the other baselines, and the NCDE and S-NCDE models. The results of the baselines are in agreement with the results obtained by [31]. The ODERNN model achieves a mean overall validation accuracy of 86.4% and a mean F1-score of 75.9%. The NCDE and S-NCDE models achieve almost identical results between each other and roughly a 0.6% lower accuracy and a 1.6% lower F1-score than the ODERNN model. Note that the S-NCDE model can achieve similar results to the NCDE model with half the number of parameters, but taking a considerably longer training time. Additionally, Figure 5.1 summarizes the evolution of the validation loss of all models during training. It can be observed that the baselines are able to achieve a lower loss on the validation set than the NCDE and S-NCDE models, throughout the training. The GRU model observes some considerable degree of overfitting towards the end, however the final results do not reflect this due to the early stopping methodology that is applied.

The main results for each model on Swisscrop dataset are reported in Table 5.2. For this dataset, the best results are obtained by the GRU model with an overall accuracy of 87.9% and an average per-class F1-score of 90%. Surprisingly, the ODERNN model had a lower performance than both of the other baselines, contrarily of what was seen on the TUM dataset. However, this is thought to be due in part to the limited hyperparameter analysis

Table 5.1.: Base case results on TUM dataset. The $\mu \pm \sigma$ of the overall accuracy and average F1-score across three runs is reported for each model, with the average training time.

Model	Accuracy (%)	F1-score (%)	Training time	Parameters
GRU	86.2 ± 0.2	75.4 ± 0.7	4 min	210K
LSTM	86.2 ± 0.1	75.1 ± 0.4	5 min	275K
ODERNN	**86.4 ± 0.2**	**75.9 ± 0.2**	3.7 h	405K
NCDE	85.8 ± 0.1	74.3 ± 0.2	4 h	430K
S-NCDE	85.7 ± 0.1	74.3 ± 0.2	20 h	210K

performed for this dataset. The NCDE and S-NCDE models had again, nevertheless, the lower performance of all models. The NCDE model achieved an accuracy of 85.5 % and a F1-score of 86.6 %, while the S-NCDE model had a similar accuracy, but a F1-score than was 1 % lower in average. The training times of the NCDE and S-NCDE models are fairly high compared to the baselines and are higher than those observed on the TUM dataset, presumably due mainly to the longer time series. As for the TUM dataset the GRU and LSTM models can be successfully trained in under five minutes. Finally, Figure 5.2 summarizes the evolution of the validation loss during training for the results presented in Table 5.2. As it was observed for the TUM dataset, the baselines achieved a lower validation loss than the NCDE and S-NCDE models, throughout the training.

Table 5.2.: Base case results on Swisscrop dataset. The $\mu \pm \sigma$ of the overall accuracy and average F1-score across three runs is reported for each model, with the average training time.

Model	Accuracy (%)	F1-score (%)	Training time	Parameters
GRU	**87.9 ± 0.4**	**90.0 ± 0.3**	4.5 min	215K
LSTM	87.7 ± 0.2	89.4 ± 0.1	4 min	285K
ODERNN	86.4 ± 0.6	87.9 ± 0.6	3.5 h	280K
NCDE	85.5 ± 0.8	86.6 ± 0.8	13 h	305K
S-NCDE	85.4 ± 0.8	86.5 ± 0.8	1 d 9 h	170K

5.2.2. Per-crop performance analysis

An analysis of the crop predictions made by the ODERNN and NCDE models for the TUM and Swisscrop dataset is presented in this section. In Figure 5.3, the per-class F1 scores obtained on average by the runs in section 5.2.1 are presented for the TUM dataset. Both models were able to identify most of the classes with a similar performance, but there are a few classes that are specially difficult for them to recognize. Fallow was the most critical case, with a F1-score close of 29% for the ODERNN model and only 23% for

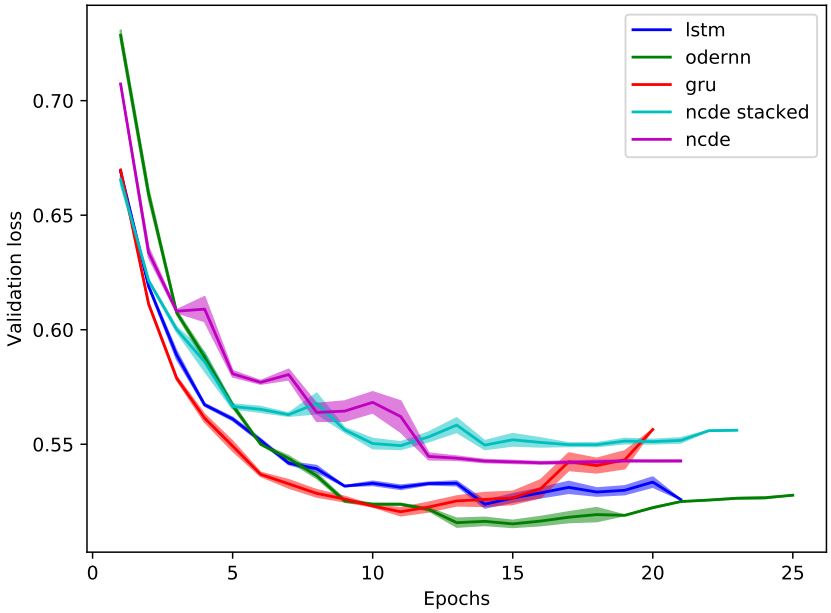


Figure 5.1.: Validation loss for all models in base case scenario on the TUM dataset. For each model the $\mu \pm \sigma$ at every epoch and across three runs is visualized.

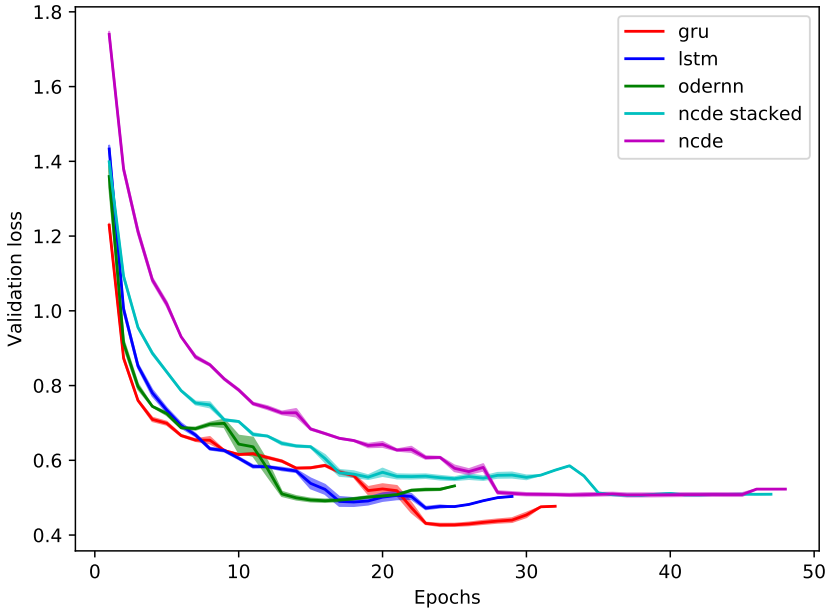


Figure 5.2.: Validation loss for all models in base case scenario on the Swisscrop dataset. For each model the $\mu \pm \sigma$ at every epoch and across three runs is visualized.

the NCDE model. Other problematic classes were winter triticale, winter rye, asparagus and winter spelt. As it can be seen in the respective confusion matrices in Figures 5.5 and 5.6, fallow was often mistaken by the other or the meadow classes, while winter triticale was confused mostly with the similar classes winter wheat and winter barley. Fallow is a technique where arable land is let uncovered for a cultivation season, i.e., without any crop or vegetation cover. For this reason, the variation of the reflectances values throughout the year compared to any crop, which follow a phenological cycle, are likely to be limited. The models seem to understand that it is not a normal crop but predict that it is either a meadow or other, presumably because there is no sufficient observable differences in the data, to help them discern between the non-crop classes.

For the Swisscrop dataset, the per-class F1-scores obtained by the ODERNN and the NCDE models on the base case, are presented in Figure 5.4. For this dataset the non-crop classes, such as gardens, pasture, meadow and hedge, are the most difficult for both models to identify. The performance of the ODERNN and NCDE model is similar for most of the classes, with exception of the garden and chestnut classes. As it was noted in Chapter 4, this classes have very few examples, so the ODERNN model is more effective at extracting information from this limited amount of examples. Finally, a visualization of the kind of plot-level predictions that can be obtained with an NCDE model on this dataset is presented in Figure 5.7. Unfortunately, only the relative position of the fields is available without the pixels geo-localization, thus a reference background layer could not be incorporated.

5.3. Ablation studies

In this section the results of four ablation studies, that evaluate the effect of the models' configuration choices taken in section 5.2, are presented. All the studies in this section were performed only with the TUM dataset. The ablation studies are: (*i*) neglecting the use of the observation mask in the data, (*ii*) the interpolation method and time spacing used, (*iii*) the effect of using all available features in the data, and (*iv*) the effect of including cloudy observations.

5.3.1. Observational mask importance

This experiment evaluates the performance of the models when the observational mask is not included as an additional feature in the data. The results are presented in Table 5.3. The performance of all models decreases, compared to the base case. However, the effect is more distinct for the NCDE model, where for example the mean F1-score obtained dropped by 1.7%, while it was only reduced by 0.3% for the GRU model or 0.6% for the ODERNN model, for instance. The effect is weaker in a S-NCDE model, where there is only a 0.9% reduction on the F1-score. This results are in agreement with those obtained by [32], who also observed that including the observational mask as a channel is particularly important for a NCDE model. This may be attributed to a

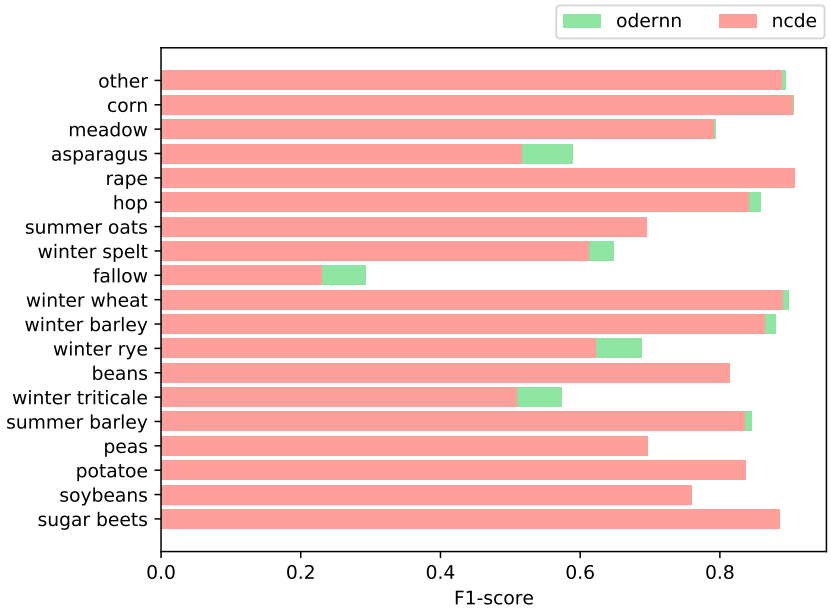


Figure 5.3.: Best validation F1-scores obtained on the TUM dataset classes for the base case of the ODERNN and NCDE models. For each class, the average across three runs is visualized.

case of informative missingness [86], which means that in this case, a NCDE model loses relevant information about the location of missing values after interpolating the data, if its not included as a channel. In contrast, the baselines can have an awareness of where the missing values are, by just directly learning that they are represented by zeros. For the NCDE and S-NCDE models, it is possible that when the observational mask is present as a feature, they use it to learn to be more skeptical about longer segments of $\mathbf{X}$, such as those built by skipping several invalid observations, and to rely more on segments were there was more valid observations. Note that when using a linear interpolation method, the NCDE and S-NCDE models may still be able to capture this information by looking for where the kinks in the interpolation are. Thus, since a S-NCDE model is in theory a more powerful version of the NCDE model, it may be more effective at learning these relationships, which could explain the weaker reduction in performance w.r.t the NCDE model in Table 5.3.

5.3.2. Interpolation methods

In this section, the performance of the NCDE and S-NCDE models are studied when using different interpolation and regression methods, and when using the original timestamps to build them or not. The methods evaluated are linear and cubic splines interpolation, and SE kernel regression. Using the original timestamps (i.e., *irregularly spaced*) in

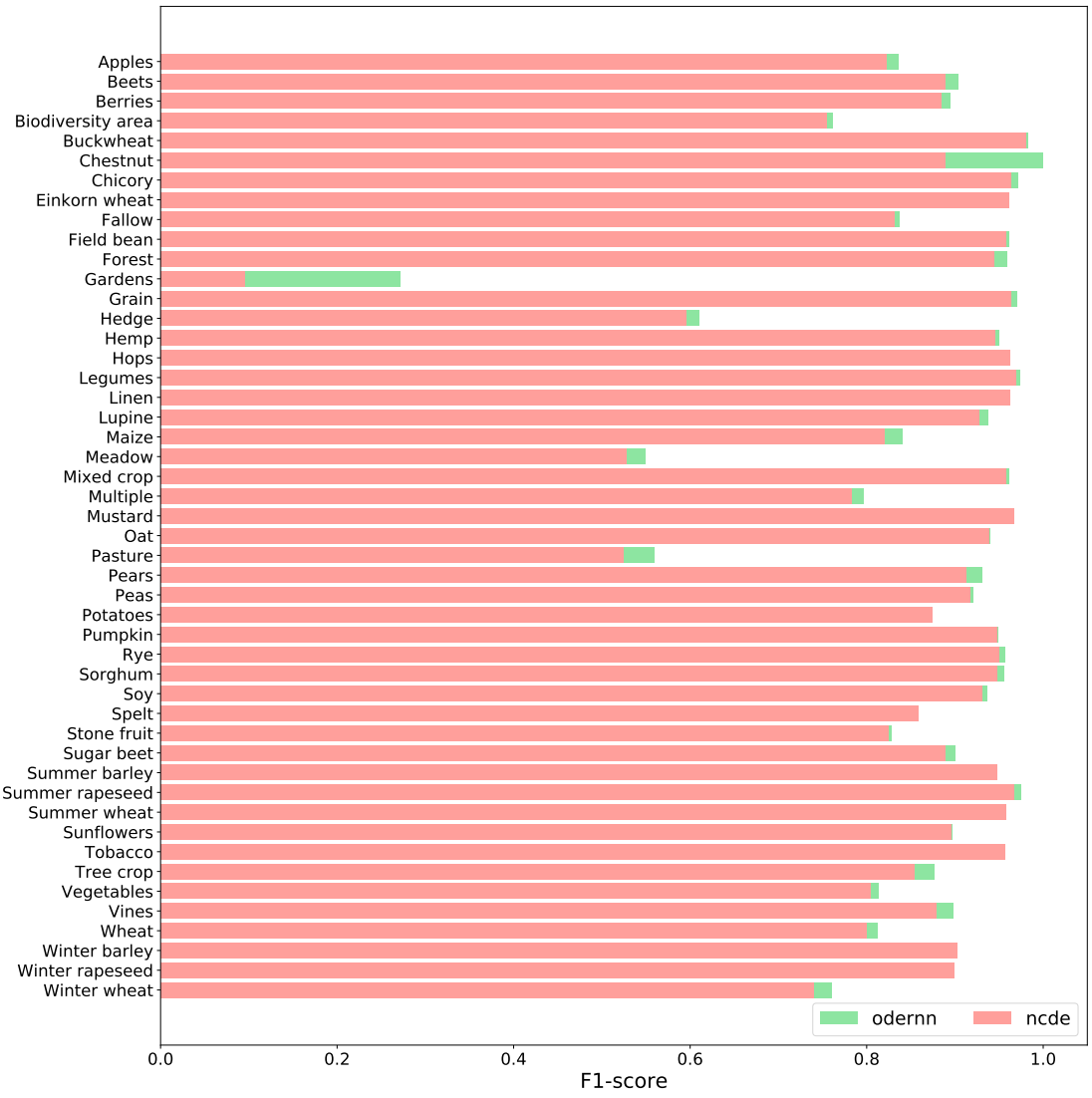


Figure 5.4.: Best validation F1-scores obtained on the Swisscrop dataset classes for the base case of the ODERNN and NCDE models. For each class, the average across three runs is visualized.

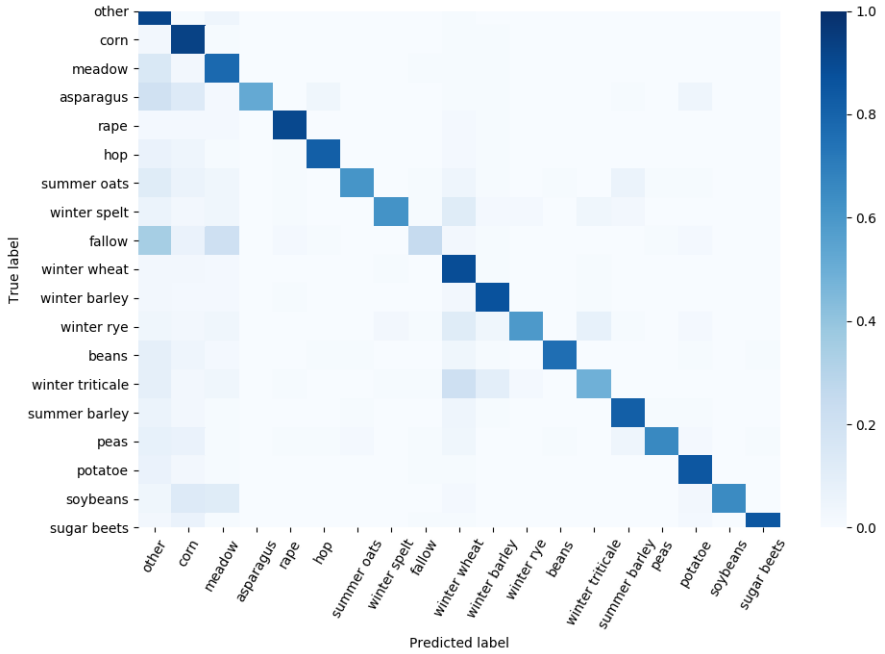


Figure 5.5.: Confusion matrix of ODERNN model on TUM dataset, normalized by rows.

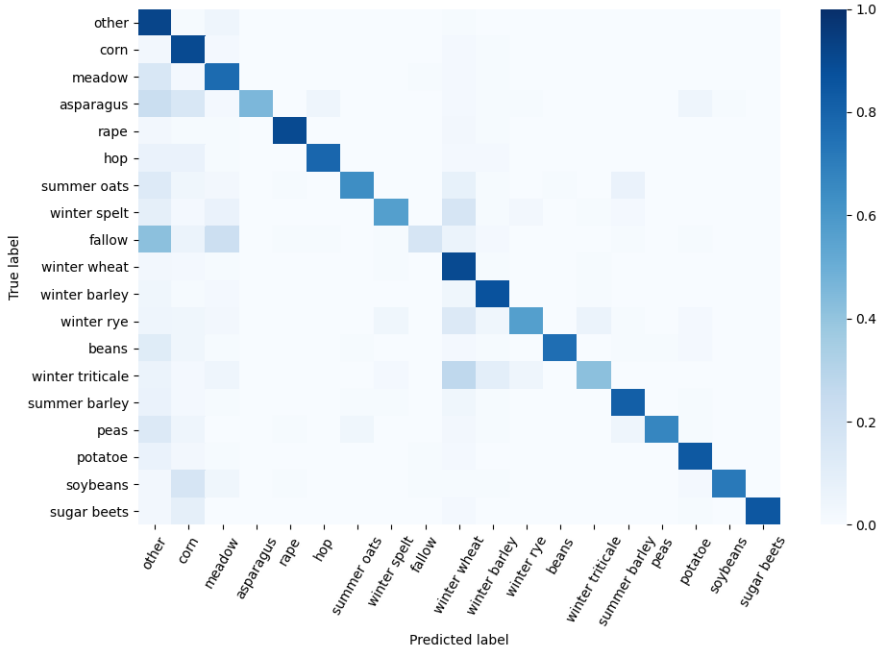


Figure 5.6.: Confusion matrix of NCDE model on TUM dataset, normalized by rows.

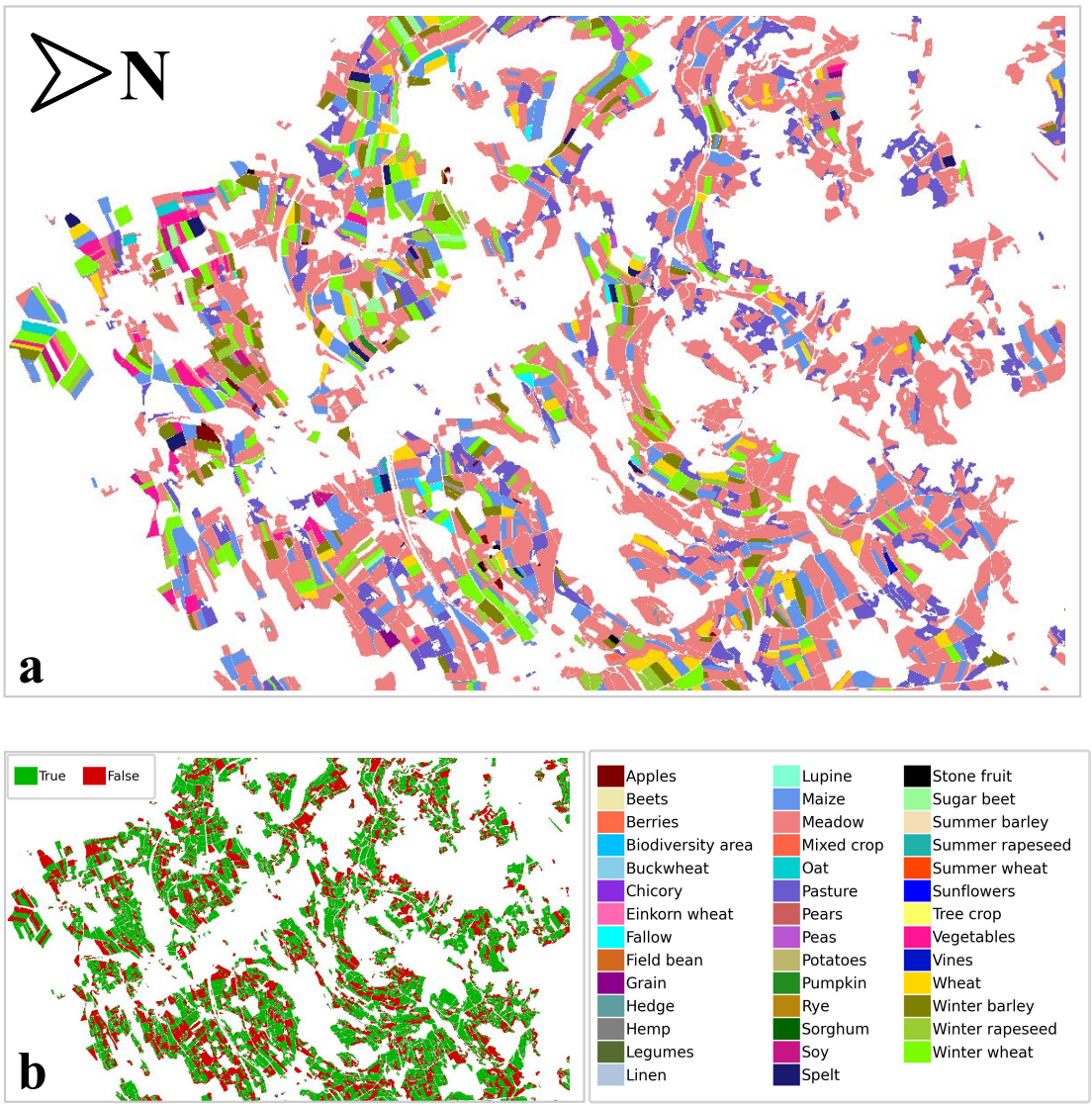


Figure 5.7.: Crop predictions (a) and accuracy map (b) for the same test area of Swisscrop dataset with the NCDE model.

Table 5.3.: No observational mask on TUM dataset. The $\mu \pm \sigma$ across three runs is reported for each model.

Model	Accuracy (%)	F1-score (%)	Parameters
GRU	86.1 ± 0.2	75.1 ± 0.3	210K
LSTM	86.0 ± 0.1	74.7 ± 0.9	275K
ODERNN	**86.1 ± 0.2**	**75.3 ± 0.2**	405K
NCDE	85.4 ± 0.3	72.6 ± 0.8	400K
S-NCDE	85.5 ± 0.2	73.4 ± 0.4	205K

the interpolation is evaluated here, in contrast to the default implementation choice on torchcde, which is to not build the interpolation with the original timestamps and rather use an *equally spaced* time array. This is possible, as detailed in section 3.4.4, because of the reparametrization invariance property of NCDEs, and as long as the original timestamps are appended in the data to ensure that no information is lost. Visualization examples of the methods are presented in Figure 5.8 for linear interpolation, Figure 5.9 for cubic splines, and Figure 5.10 for SE kernel regression. Note how the topology of the continuous path and the derivative values may considerably change for the same sample, depending on the method used. The results for the NCDE and S-NCDE models when using each method are presented in Table 5.4. The results show that for the NCDE and S-NCDE models, a linear interpolation method is always better than the cubic interpolation and the SE kernel regression. Additionally, it makes little difference in terms of performance for both models if the equally spaced or irregularly spaced schemes are used for the linear interpolation method, however it does makes an appreciable difference in the F1-score when using the cubic splines interpolation. The SE kernel regression method is only implemented for using the original timestamps, and has the lowest performance of the three methods. However, since no search is made for the SE kernel method's hyperparameters, as disccused in Chapter 6, its performance is expected to be improved with some additional work. In particular, its performance may be improved to at least that of a cubic spline interpolation, if the regression is enforced to exactly fit the data points. In terms of speed, for the NCDE model the SE kernel methods is the fastest with an average run time of 1.5 hours, while the linear interpolation methods and cubic spline methods take roughly 4.5 and 7 hours to train respectively, regardless of the timing scheme used. On the other hand, for the S-NCDE method, linear interpolation is the fastest with an average run time of 11 hours when using equally spaced timestamps and 17 hours when using the original irregularly spaced timestamps, while taking between 19 and 22 hours on average for all the other options. Overall, using the linear interpolation method for both models with the default equally spaced timing scheme seems to give the best trade-off between performance and speed of computation, hence the choice of it in the rest of the experiments presented in this chapter.

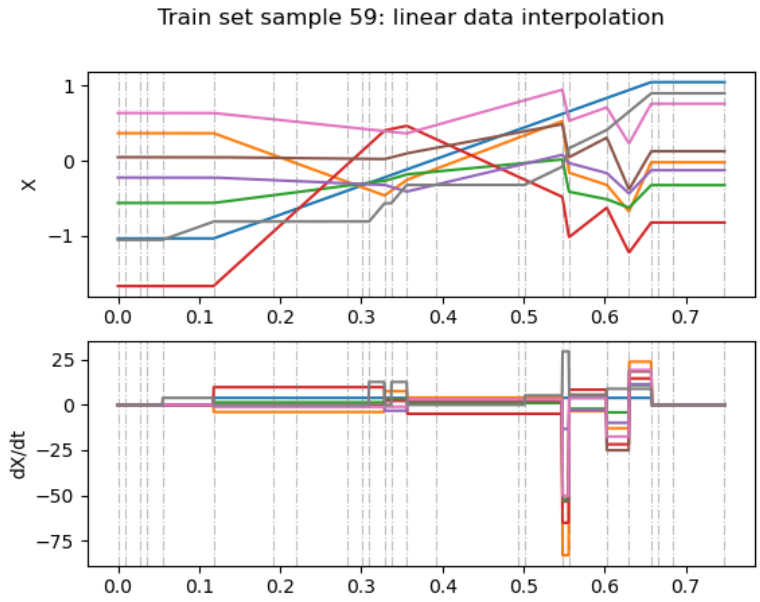


Figure 5.8.: Sample of linear interpolation on the TUM dataset. The dashed vertical lines denote the timestamps used for interpolation. A thousand points are used for each figure for visualization purposes.

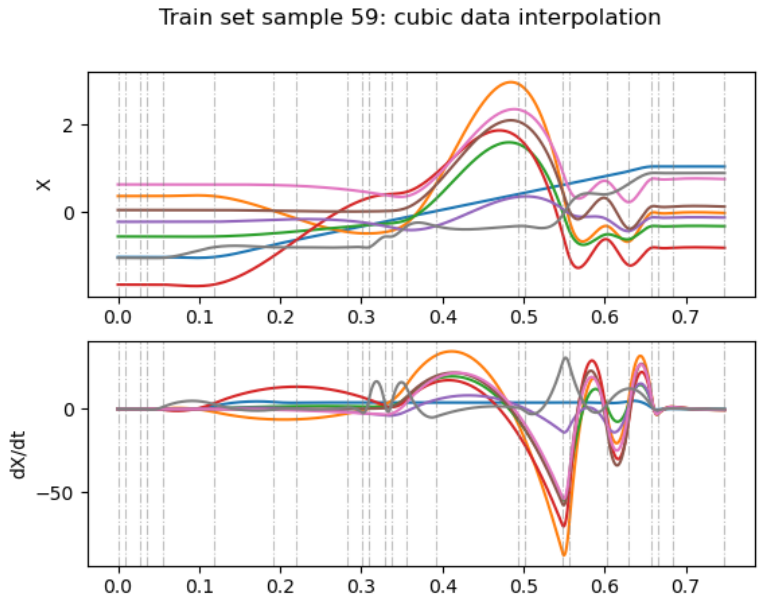


Figure 5.9.: Sample of cubic splines interpolation on the TUM dataset. The dashed vertical lines denote the timestamps used for interpolation. A thousand points are used for each figure for visualization purposes.

Table 5.4.: Comparison of interpolation methods for NCDE model on TUM dataset. The $\mu \pm \sigma$ across three runs is reported for each model.

		Equally spaced		Irregularly spaced	
Model	Method	Accuracy (%)	F1-score (%)	Accuracy (%)	F1-score (%)
NCDE	Linear	**85.7 ± 0.1**	**74.2 ± 0.4**	85.7 ± 0.1	74.1 ± 0.4
	Cubic	85.4 ± 0.1	73.1 ± 0.2	85.2 ± 0.0	72.5 ± 0.1
	SE kernel	—	—	81.7 ± 0.8	60.3 ± 4.3
S-NCDE	Linear	85.6 ± 0.4	73.5 ± 1.0	**85.6 ± 0.3**	**73.7 ± 0.9**
	Cubic	85.3 ± 0.2	72.7 ± 0.7	85.0 ± 0.2	71.8 ± 0.5
	SE kernel	—	—	84.2 ± 0.3	69.4 ± 1.1

5.3.3. High-dimensional dataset

This study evaluates the models' performance when using all the 54 original features of the TUM dataset, plus one feature for original timestamps and one for the observational mask as usual. The results are presented in Table 5.5. The best results are obtained by the ODERNN model, with an accuracy of 87.4% and a F1-score of 77.5%. All the baseline models observe an increase of at least 1% in accuracy and 1.6% in F1-score w.r.t. the base case, thus they are able to effectively use the contextual information provided in the additional features. In contrast, both the NCDE and the S-NCDE models observe a sharp decrease in performance. In particular, the F1-score of the NCDE model decreases by almost 6%, and the S-NCDE model does the same by 2.7%. Note also, that the number of parameters of the NCDE model quintuplicate w.r.t. the base case, which is prohibitively large compared to the other models, while the S-NCDE model only doubles its number of parameters. As discussed further in Chapter 6, this is due to the fact that the size of the final layer of the vector field is a product of the number of input channels, the size of the hidden state and the width of the vector field. Therefore, as the S-NCDE model can attain a similar performance w.r.t. the NCDE model, with a much lower size of the hidden state without reducing its performance, the number of parameters does not dramatically increase.

Table 5.5.: Performance of each model when using all features of TUM dataset. The $\mu \pm \sigma$ across three runs is reported for each model.

Model	Accuracy (%)	F1-score(%)	Parameters
GRU	87.3 ± 0.1	77.2 ± 0.2	245K
LSTM	87.2 ± 0.2	76.9 ± 0.3	325K
ODERNN	**87.4 ± 0.1**	**77.5 ± 0.2**	445K
NCDE	84.6 ± 1.0	68.4 ± 3.5	2015K
S-NCDE	85.4 ± 0.8	71.6 ± 1.5	410K

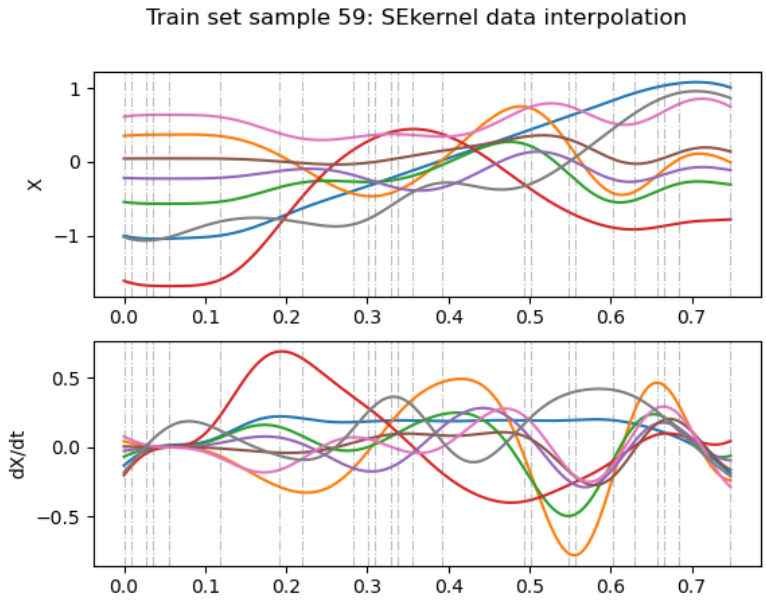


Figure 5.10.: Sample of SE kernel regression on the TUM dataset. The dashed vertical lines denote the timestamps used for interpolation. A thousand points are used for each figure for visualization purposes.

5.3.4. Cloud cover study

In this experiment the effect of including cloudy observations on the model's performance is studied. This is achieved by avoiding to mask out cloudy pixels in the original datasets during data pre-processing. This study evaluates the practical ability of the models to learn to disregard noisy uninformative observations, such as those usually observed in raw satellite images. The results of this experiment are presented in Table 5.6.

Table 5.6.: Results for each model when using TUM dataset without masking out cloudy observations. The $\mu \pm \sigma$ across three runs is reported for each model.

Model	Accuracy (%)	F1-score (%)	Parameters
GRU	85.5 ± 0.1	71.6 ± 0.3	210K
LSTM	85.7 ± 0.0	72.2 ± 0.2	275K
ODERNN	**85.8 ± 0.1**	**72.5 ± 0.4**	405K
NCDE	80.7 ± 1.1	58.8 ± 0.7	430K
S-NCDE	83.3 ± 0.9	64.0 ± 3.2	210K

The performance of all models drops remarkably w.r.t the base case, but it is especially distinctive for the NCDE model. For instance, the ODERNN model's accuracy is re-

duced by 0.6% and its F1-score drops by 3.4%, whereas the NCDE model experiences a reduction of up to 5% and 15.5 % in accuracy and F1-score, respectively. On the other hand, the S-NCDE again does show some degree of improvement compared to the NCDE model, with only a 2.4% and 10.3% drop in accuracy and F1-score, respectively. Overall, the NCDE and S-NCDE models are far more sensitive to the presence of cloudy observations, than the baselines models. As in the experiments of section 5.3, the results for the S-NCDE model are improved compared to those of the NCDE model, presumably because it can learn more relevant temporal relationships from the data, due to the increased processing, thus understanding and remembering better the characteristics of uninformative cloudy observations.

Chapter 6

Discussion

In this chapter, first the collection of results of the previous chapter are analyzed and interpreted, followed by an exposition of the advantages and limitations of the NCDE model. Finally, the outlook of future research with NCDEs for crop classification is presented.

6.1. Analysis of results

Although for most of the experiments the performance of the S-NCDE model improves compared to a NCDE model, all of them show that both NCDE model and S-NCDE model have a lower performance than those of the baseline models. The NCDE and S-NCDE model showed similar performance on the base case, but the training time of the latter, may take up to a day or more. Meanwhile, the GRU and LSTM models could be trained up to a good performance in under five minutes. Additionally, it was observed in the per-crop performance analysis, that the ODERNN and the NCDE models had similar performance on most classes, but the ODERNN model did better at predicting minority classes in the data.

The ablation studies put in perspective several important considerations for the NCDE models. In particular, while the performance of the baselines is only somewhat affected when the observational mask is not considered, it is confirmed to be a crucial addition for increasing the performance of NCDE models. NCDE models rely on a continuous-time path of the input data, which skips the missing values, so it does not necessarily know what part of the path has a better connection to the true observed values. Therefore they trust every part of the interpolation path in the same way. On the other hand, the baselines can learn how the missing values are represented and may be able to count them. NCDE models struggle more without an observational mask channel in the data, because then they do not know where the last true observation was made or even how many of them they were.

As it was found in section 5.3.2, the interpolation or regression method chosen can also have a considerable impact in the model's performance. It is observed that linear interpolation provides better empirical results. One straightforward reason to explain this may be the previously exposed matter of having a flat end and beginning of the

interpolation, whenever there is no valid observations there, like it can be observed in Figure 5.8. This may be effective in reducing the degree of freedom of the ODE solver to change the value of the hidden state, because as explained in Chapter 4, in those cases the derivative of the interpolation is exactly zero and thus the hidden state will not change when the solver evaluates the interpolation there. The previous is not the case for the other interpolation and regression methods, therefore they might be systematically misleading the solution of the hidden state when the ODE solver evaluates the data path in those regions. As the data interpolation methods will unavoidably influence the hidden state evolution, and given that the linear interpolation is only the simplest interpolation method available, this raises the question as if other less naive and smooth interpolation or regression methods could model the evolution of the data in a way that benefits NCDEs more. In particular, SE kernel regression was evaluated in this work, but with unsatisfactory results. However, given that this method itself has some hyperparameters, if enough data time points are provided perhaps its hyperparameters could be learnt from the data, potentially leading to better results than when using simple linear interpolation.

Additionally, it was shown that when using high-dimensional input data, the performance of the NCDE and S-NCDE models was reduced (although less distinctively so for the S-NCDE model) and their number of parameters rose considerably compared to the baselines models. The previous means that the baselines were able to successfully extract additional information on this features to improve their performances. This rise in the number of parameters for the NCDE models when dealing with a high-dimensional dataset has practical implications which will be discussed in the following section. Finally, an acute sensitivity of NCDEs to observations that are noisy and unrelated to crop phenology, like the presence of clouds, was revealed in this work. Even though filtering out cloudy observations is the common practice in remote sensing, the drop in the performance of NCDEs when cloudy observations are not filtered out from the data, means that the baselines models may still be used as end-to-end models on raw satellite image data with an acceptable accuracy, while NCDEs probably would not be the preferred choice in this scenario. Additionally, this provides further evidence for the belief that the continuous data representation needed for the NCDE and S-NCDE models imposes an additional inductive bias into these models w.r.t. the baselines.

6.2. Limitations of NCDEs

The exploration of a NCDE model for crop classification was motivated by the hypothesis that a continuously defined hidden state may be more effective in modelling the phenological development of a crop. However, the practical results obtained in this work did not corroborate this hypothesis, due to their lower performance compared to an ODERNN model or the RNN baselines. Assuming that no implementation error was made, the main reasons behind this are thought to be either in the continuous data representation, or in the ODE solver and are discussed in the following. First, a straightforward option is that a more refined data interpolation method could further improve the results of

NCDE models. This is believed so, as it was empirically observed that the interpolation method chosen indeed induces a strong prior in the model and can affect the performance dramatically (e.g., with an uncalibrated SE kernel regression). Some ideas for a more refined continuous data representation method are mentioned in section 6.3. Second, an adaptative ODE solver method was used in the experiments, whose operation may be particularly affected by a noisy dataset. The current implementation of this type of solvers in torchdiffeq has batch-dependant time steps, because for evaluating a candidate step they compute a norm of the hidden state over the hidden state dimension and the samples in the batch. This may lead to particularly noisy time steps taken by the solver when the data is highly heterogeneous, such as that of Swisscrop dataset which has many different classes (i.e., in a random batch some classes may be missing w.r.t a different random batch). This could be a potential reason as of why it is observed that the ODERNN method performs worst than the RNN baselines in the Swisscrop dataset. In this case the ODEs in the ODERNN model that allow the hidden state to continuously evolve, are actually hampering the performance of a GRU cell w.r.t. its performance on a stand-alone GRU model (i.e., recall that a ODERNN model is implemented here with a GRU cell). Some options for investigating this hypothesis and overcoming this potential issue are given in section 6.3.

Whichever the case may be, unless better performances can be obtained with the NCDE model, their practical advantages (e.g., less memory use than a ODERNN model [32]) are deemed limited compared to the baseline models, and there are several considerations that either limit the model's practicality or that require special attention when training NCDE models:

Number of parameters. When the number of input features in the dataset rises, the number of parameters in a NCDE model will rise proportionally on the last layer of the vector field, which may become of great size compared to its neighbouring layers. The dramatic drop in the NCDE model's performance observed in section 5.3.3 is likely due to this reason, either because of the disproportionate architectural design or a because of a severe overfitting problem, due to the five-fold rise in the number of parameters w.r.t. the base case. In any case, this is a major problem with NCDEs, because it means that if a dataset is high dimensional, one must recur to feature reductions techniques, which will inevitably lose some information and thus affect the performance of the model. Furthermore, if the dataset is also fairly big in terms of number of samples and input dimensions, there are limited options for increasing the number of parameters of the model without recurring in a disproportionate size of the final layer in the vector field f_θ.

Stability. Model's stability in training (i.e., having a training loss that is decreasing without big prominences) has been found by some authors, to be dependant on the learning rate (lr) to batch-size (BS) ratio [97], [98], and the appropriate ratio depends on the amount of parameters that the model has. In practice, it was found in some preliminary experiments in this work that the bigger the network, the lower the $\frac{lr}{BS}$ ratio had to be to continue observing a stable training. This

can be achieved by either lowering the learning rate or by increasing the batch size, to ensure that update steps taken by the optimizer are not big or at least the gradients are averaged out over more samples, thus preventing noisy parameter updates. With that said, and bearing in mind the previously mentioned hypothesis of potentially noisy steps taken by an adaptative ODE solver as well, one alternative to prevent the latter, while also keeping a stable training loss, is to considerably increase the batch size, and to increase the learning rate in linear proportion [97]. In this work, having a larger batch size of 1024 (i.e., w.r.t. to 512), was found to be beneficial to the NCDE model. In some exploratory tests on the TUM dataset, the batch size was further increased while still producing similar and stable results, however having larger batch sizes on the Swisscrop dataset was not evaluated due to time constraints.

Speed of computation. Even if using a semi-norm [95] for the ODE solver, did increase the speed of computation for NCDEs, it was repeatedly observed across the experiments that NCDEs are very slow to train compared to the GRU and LSTM models. NCDEs can take several hours to fit the data and the S-NCDE model up to days, while RNNs can do it in just a couple of minutes. This is big practical constraint for developing new ideas and testing them quickly. In addition, one must consider the computation time of the interpolation coefficients in the data pre-processing. When missing values are largely present in the data, such as those representing cloudy observations in this work, the available interpolation methods in torchcde become slow. In spite of a particular model being slow to train, if they would achieve considerable better results than other models in practice, trained models could be applied at inference time for applied use, so it may not matter if they were slow to train. However, given that NCDEs would additionally need to interpolate the new time series at inference time, their practicality for creating crop distribution maps over large areas is questionable, as it would take them a much longer computation time compared to other models that do not require interpolation such as RNNs or the ODERNN model. For a reference, the predictions with the trained models over a fifth of the total area of the Swisscrop dataset (i.e., 480 km^2), from which a sample was used for visualization in Figure 5.7, took about 10 hours for the NCDE model and only one minute for the GRU model, using a single GPU. This is a considerable 600 times difference, that could make the use of NCDE models for making country-level crop maps, problematic depending on the computational resources available.

6.3. Outlook

In this section, some possible directions for future work looking to improve the performance and speed of NCDE models for crop classification are given.

- **Vector field design.** Improving the design of the vector field in terms of its final

layer. For a vector field that is a FFNN, as implemented in the original NCDE paper and in this work, its output layer is a linear layer that maps $\mathbb{R}^r \to \mathbb{R}^{p \times v}$, when there are r hidden units. This means that this layer has $r \times p \times v$ parameters, so when the input dimension v is large, it may cause an ill-defined architecture. So the development of ideas on how to circumvent this issue is required. Additionally, the design of the controlled vector field $g_{\theta,\mathbf{X}}$ could be improved by squeezing $\frac{d\mathbf{X}}{dt}$ values in a similar way that the final tanh activation function does so with the vector field f_θ, which may be the reason of NCDEs' sensitivity to noisy data. Finally, other combinations of hidden state sizes between the two CDEs could be used when using a S-NCDE model.

- **Continuous data representation.** As discussed in section 6.2, more sophisticated interpolation or regression methods may provide improved results w.r.t. to linear interpolation. In particular, a smooth method would be preferable, because the underlying phenological process is smooth. One option is to improve SE kernel regression by calibrating its hyperparameters with the data. Another option is to hybridize a cubic spline interpolation to have a flat end and beginning if there are missing values there, like it is the case for linear interpolation. In this case the hidden state would be ensured to remain unchanged when the vector field is evaluated in those regions.
- **ODE solver.** Adaptative ODE solvers in torchdiffeq calculate a batch-dependant norm, which is thought to affect the models performance on certain datasets. The most straightforward approach to test this hypothesis would be to do a hyperparameter search with fix-step solvers and evaluating different step size values, for the ODERNN and NCDE models on the Swisscrop dataset and see if better results can be obtained. Other options may be to adjust the error tolerances of adaptative solvers to smaller values, however this would incur into even larger training times; or fix the current torchdiffeq algorithm for adaptative ODE solvers, to compute only a sample-wise norm, in order to have independent norms used to solve the ODEs on each sample. Alternatively, *Hypersolvers* [99] may be evaluated, which add a learnable term to the solver step in order to enhance the solver and have presumably a notable reduction on speed of computation.
- **Uncertainty.** NCDEs and NODEs, are not able to process stochastic input nor to give any uncertainty estimates about their predictions, because ODEs and CDEs are deterministic. *Neural ODE processes* [100], which define a stochastic process over a distribution of NODEs, appears like a promising alternative that can process and dynamically adapt to incoming data points from time series input like NCDEs do, while also modelling uncertainty in the data, the hidden state and the predictions.

Chapter 7

Conclusion

In this work the use of Neural Controlled Differential Equations, a novel deep learning model, was explored for the problem of crop classification with time series of satellite images. Modelling the temporal dimension is crucial for a better classification, since changes on the reflectance values of the crops throughout the year holds distinctive phenological information of each crop type. Deep learning methods that are able to process time series data, are currently the state-of-the-art for crop classification, largely due to their flexibility and scalability for dealing with data. Recurrent Neural Networks in particular, have recently shown outstanding empirical results for crop classification. This type of methods however, holds information in a hidden state that is not continuously defined, which is thought to be a disadvantage when attempting to model continuous phenomena such as the phenological development of a crop. To address this issue, a recently proposed enhancement to an RNN model, the ODERNN model, has been successfully applied for crop classification. This model is able to continuously define portions of the hidden state of an RNN, and its empirical performance validate the added value of a continuous hidden state model as a proxy for crop phenology. For this reason, and also motivated by research in ecology on continuous phenology models, it is believed that a model that could rather define an entirely continuous hidden state would be more beneficial for the task of crop classification. Neural Ordinary differential equations, are a type of deep learning models that have recently entered the stage for continuous-state modelling and are a rapid-pace area of research, with application to numerous fields. The ODERNN model efficiently uses NODEs between an RNN cell data readings, for a semi-continuous modelling of its hidden state. In the last year, NCDEs have been proposed as a related model to a NODE that can model a fully continuous hidden state, unlike the ODERNN model, and that can dynamically incorporate time series data by by defining a Controlled Differential Equation. In contrast, NODEs behave like ODEs, thus are fully defined by their initial value and have no mechanism for incorporating incoming time series data.

To evaluate NCDEs for crop classification, the model's performance was methodologically assessed on a series of experiments, in comparison to selected state-of-the-art baseline models, namely, the aforementioned ODERNN model and two types of RNNs, the GRU and LSTM models. In addition, a more complex stacked version of NCDEs, i.e., the S-NCDE model, was developed and evaluated. All the models were studied with two multi-temporal crop classification datasets, the TU Münich dataset, which consists of 19

crop classes, and the Swisscrop dataset from Zürich and Thurgau, and has 52 crop classes. First, the models were evaluated in a base case scenario for both datasets, consisting on the configuration choices that better suited NCDEs, such as only using a reduced-features version of TUM dataset or a shorter time series version of Swisscrop dataset, and including the observational mask as a channel in the data. The best results were obtained by the ODERNN baseline, closely followed by the two RNN baselines. The NCDE and S-NCDE models showed similar performance, but the S-NCDE model could manage to do so with fewer parameters at a cost of a higher computation time. In this latter aspect, the RNN baselines were better than the other models by a considerable margin, and the NCDE training time was always higher than that of the ODERNN model. In addition the ODERNN and NCDE models' performance at detecting each crop type was further analyzed. It was observed in general, that both models were able to identify most of the classes similarly well, however for difficult non-crop classes like fallow or pasture, minority classes like chestnuts or gardens, and very similar classes such as winter triticale and winter rye, the ODERNN methods was consistently better than the NCDE model.

Subsequently, NCDEs and the other models were further scrutinized with a series of ablation studies, in order to better understand their strengths and weaknesses. A study where the addition of the observational mask as a channel was omitted, showed a decrease in performance for all models, but had a particularly adverse effect for the NCDE model and the S-NCDE model, although to a lesser degree on the latter. Hence, it was found that this is indeed an important consideration that must be taken with NCDE models, as they first need to create a continuous data path with the data points, thus losing the sense of location of the missing values in the data. When the observational mask is not included, the NCDE models struggle to find out which portions of the continuous data path are more reliable. Later, the effect of using different continuous data representation methods such as linear and cubic interpolation, in addition to the squared exponential kernel regression, was explored for the NCDE and S-NCDE models. The results confirmed the importance of the method selected and showed an overall better performance with the simpler linear interpolation. Another study regarding the use of a a higher dimensional input data revealed that the NCDE models are particularly affected in this scenario, due to the effect that this has on the vector field's network architecture, whose parameters on the final layer depend directly on the input dimension size, thus originating a disproportionate number of parameters. The S-NCDE model suffered this problem to a lesser degree, and the baseline models could conversely benefit from the additional features, thus increasing their performance. Finally, a study where the cloudy observations in the dataset were not neglected, showed that every model experienced a decrease in performance due to the presence of uninformative observations. The NCDE model experienced the largest decrease in performance, followed by the S-NCDE model. This result exposes the sensitivity of NCDE models to uninformative observations. Additionally, it further lays in evidence that the need of creating a continuous data path of real physical phenomena imposes an inductive bias or prior assumption to the NCDE models, which is not present for the baselines models.

Finding high performing and practical-to-use models for crop classification is of great relevance for agricultural monitoring, as it allows for a scalable, effective and accurate way of mapping crop distributions on a country-scale. This information is valuable for resources management, agrarian policy making, ecosystems protection and food security, among others. In this work it was shown that NCDE models are not yet at the level of other state-of-the-art models for this task, and are slow to train or to test, compared to RNN models. However, their performance is not far behind, specially considering that they do not rely on an RNN model like the ODERNN model does, and it is believed that there are solvable practical issues that hindered their performance and prevented them from fully expressing the benefits of continuous-time modelling. In particular, smooth methods for continuous data representation that comply with some desirable qualities that linear interpolation shows, are worth to be explored, such as a cubic interpolation with a flat end and beginning when there are missing values there, or to learn SE kernel regression's hyperparameters with the data. In addition, it is believed that an adequate numerical solver configuration is indeed important for finding appropriate hidden state solutions. It was observed on the Swisscrop dataset, that even an ODERNN model would do worse-off than an RNN model, presumably because of the negative effect of a batch-dependant adaptative ODE solver on a dataset with many classes. Therefore, further work is needed to develop and test these and more ideas. What is certain, is that the irruption of NODEs are a scientific breakthrough, where the ruling physical modelling paradigm of differential equations and the new and powerful modelling paradigm of machine learning have been combined. With the current pace of research in this topic, these models are likely to influence and drive applications on many fields that have inherently close connections to physical systems, such as remote sensing and environmental modelling, in the years to come.

Appendix A

Hyperparameters analysis

A separate grid analysis was performed for each model on each dataset, with one run per combination of hyperparameters. For all models, an initial learning rate of 0.001 is used (unless otherwise stated), which was found to be a robust choice in general. The batch size is always part of the analysis in order to find appropriate learning dynamics for the optimizer. In the same way as for all the experiments in the results section, the learning rate was reduced with a factor of 0.1 when the overall validation F1-score failed to increase after one epoch.

For TUM dataset, the reduced-features version of the dataset, explained in Chapter 4, was used. Additionally, for the NCDE and S-NCDE models, linear interpolation method with equally spaced intervals was used for the continuous data representations, as it was the case too for Swisscrop dataset. The results for the GRU, LSTM, ODERNN, NCDE and S-NCDE models are presented in Tables A.1 A.2, A.3, A.4, A.5, respectively. The hyperparameters analyzed for GRU and LSTM models are only batch size and hidden state size, and the values tried for both models are identical. The hyperparameters analyzed for the ODERNN, NCDE and S-NCDE models are the same, i.e., batch size, hidden state size, and width and depth of the vector field. The values used are similar, with only slight differences in the values tried for the hidden state size. In each table, the combination that was selected is marked in bold. For the NCDE and S-NCDE methods this was not necessarily the combination that achieved the best performance on the metrics, but one that performed acceptably, did not severly overfit, and had a similar amount of parameters than the selected baselines models.

For Swisscrop dataset, the results for the GRU, LSTM, ODERNN, NCDE and S-NCDE models are presented in Tables A.6 A.7, A.8, A.9, A.10, respectively. Due to time constraints, the analysis was performed only on a fraction of the time series length with a time downsampling factor of four, i.e., the length of the time series was downsampled from 142 to 36, with the proceeding detailed in section 4.2. Additionally, for speeding up the analysis with the ODERNN, NCDE and S-NCDE models, some information from the analysis on TUM dataset is considered. Since the GRU and LSTM models are fast to train, the learning rate was additionally considered in the analysis, as it was observed that these models could improve their performance with a higher learning rate. For the ODERNN the hyperparameters analyzed are the same than for the TUM dataset, with the exception of the initial learning rate, for which a value of 0.01 was used, and for the case of three hidden layers on the vector field, which was omitted because it was

observed in the TUM dataset that the results were worse overall in this case. For the NCDE model, the only change w.r.t the analysis on the TUM dataset is that the hidden state size was always set to 128, which was found to work better on the TUM dataset. Finally, for the S-NCDE model the analysis is substantially reduced, due to the long training times, and only the batch size and the depth of the vector field are analyzed, while the hidden state size is set to 32 and the hidden units of the vector field to 128, like in the TUM dataset.

Table A.1.: Grid search over batch size (BS) and hidden state size (HC) for GRU model on TUM dataset.

BS	HC	Accuracy (%)	F1-score (%)	Parameters
256	64	85.8	73.6	15K
	128	86.2	74.6	55K
	256	**86.3**	**75.6**	210K
512	64	85.1	71.3	15K
	128	86.0	74.8	55K
	256	86.2	75.0	210K
1024	64	84.8	69.2	15K
	128	85.4	71.6	55K
	256	86.1	74.1	210K

Table A.2.: Grid search over batch size (BS) and hidden state size (HC) for LSTM model on TUM dataset.

BS	HC	Accuracy (%)	F1-score (%)	Parameters
256	64	85.3	71.7	20K
	128	86.2	74.6	75K
	256	**86.4**	**75.0**	275K
512	64	84.6	69.2	20K
	128	85.9	73.7	75K
	256	86.1	74.8	275K
1024	64	84.6	69.3	20K
	128	84.8	70.0	75K
	256	85.9	74.2	275K

Table A.3.: Grid search over batch size (BS), hidden state size (HC), and width (HU) and depth (HL) of the vector field for ODERNN model on TUM dataset.

BS	HC	HU	HL	Accuracy (%)	F1-score (%)	Parameters
512	128	128	1	86.0	73.7	90K
			2	86.4	75.4	105K
			3	86.3	75.3	120K
		256	1	86.3	74.4	120K
			2	86.5	75.7	185K
			3	86.3	75.1	250K
	256	128	1	86.6	76.0	275K
			2	86.6	75.5	290K
			3	86.6	75.8	310K
		256	1	86.4	76.1	340K
			2	**86.6**	**76.1**	405K
			3	86.4	75.8	470K
1024	128	128	1	86.0	73.8	90K
			2	85.8	72.3	105K
			3	86.2	74.8	120K
		256	1	85.7	72.9	120K
			2	85.8	73.3	185K
			3	85.8	73.4	250K
	256	128	1	86.5	75.9	275K
			2	86.5	75.5	290K
			3	86.6	75.6	310K
		256	1	86.5	75.8	340K
			2	86.5	75.9	405K
			3	86.3	74.6	470K

Table A.4.: Grid search over batch size (BS), hidden state size (HC), and width (HU) and depth (HL) of the vector field for NCDE model on TUM dataset.

BS	HC	HU	HL	Accuracy (%)	F1-score (%)	Parameters
512	64	128	1	85.1	72.4	75K
			2	85.2	72.4	90K
			3	85.2	72.2	110K
		256	1	85.3	72.9	150K
			2	85.2	72.4	215K
			3	85.5	72.8	280K
	128	128	1	85.2	72.9	150K
			2	85.5	73.6	170K
			3	85.7	73.8	185K
		256	1	85.4	73.3	300K
			2	85.2	72.7	365K
			3	85.0	72.3	430K
1024	64	128	1	85.3	72.9	75K
			2	85.1	72.4	90K
			3	85.2	72.2	110K
		256	1	85.4	73.1	150K
			2	85.5	73.6	215K
			3	85.7	74.1	280K
	128	128	1	85.3	73.2	150K
			2	85.0	72.0	170K
			3	85.6	73.4	185K
		256	1	85.5	73.8	300K
			2	85.6	73.9	365K
			3	**85.7**	**74.2**	430K

Table A.5.: Grid search over batch size (BS), hidden state size (HC), and width (HU) and depth (HL) of the vector field for the S-NCDE model on TUM dataset.

BS	HC	HU	HL	Accuracy (%)	F1-score (%)	Parameters
512	32	128	1	85.8	74.4	175K
			2	**86.0**	**74.8**	210K
			3	83.5	67.5	240K
		256	1	85.6	74.4	350K
			2	85.9	74.4	480K
			3	85.8	74.2	610K
	48	128	1	85.9	74.7	365K
			2	85.9	74.8	395K
			3	84.9	71.6	430K
		256	1	85.7	74.7	720K
			2	86.0	75.1	815K
			3	84.6	70.4	985K
1024	32	128	1	85.3	72.8	175K
			2	85.6	73.7	210K
			3	85.6	73.8	240K
		256	1	85.5	73.5	350K
			2	85.6	74.2	480K
			3	84.7	71.5	610K
	48	128	1	85.0	72.2	363K
			2	85.7	74.3	395K
			3	85.5	73.4	430K
		256	1	85.6	73.8	720K
			2	85.7	73.7	850K
			3	85.2	72.3	985K

Table A.6.: Grid search over batch size learning rate (lr), batch size(BS) and hidden state size (HC) for GRU model on Swisscrop dataset.

lr	BS	HC	Accuracy (%)	F1-score (%)	Parameters
0.001	256	64	75.9	74.0	15K
		128	83.1	82.4	60K
		256	87.0	88.3	215K
	512	64	71.5	68.4	15K
		128	80.1	81.0	60K
		256	86.4	87.7	215K
	1024	64	66.1	61.3	15K
		128	76.9	75.0	60K
		256	86.0	87.6	215K
0.01	256	64	80.1	81.0	15K
		128	86.6	87.9	60K
		256	82.3	84.5	215K
	512	64	79.0	78.9	15K
		128	85.2	87.1	60K
		256	86.1	87.8	215K
	1024	64	78.7	79.4	15K
		128	85.2	87.1	60K
		256	**87.4**	**89.3**	215K

Table A.7.: Grid search over batch size learning rate (lr), batch size(BS) and hidden state size (HC) for LSTM model on Swisscrop dataset.

lr	BS	HC	Accuracy (%)	F1-score (%)	Parameters
0.001	256	64	76.4	74.6	20K
		128	83.0	83.9	75K
		256	86.8	88.9	285K
	512	64	72.5	70.4	20K
		128	80.8	82.1	75K
		256	86.2	85.9	285K
	1024	64	67.9	64.7	20K
		128	77.1	75.8	75K
		256	82.9	82.4	285K
0.01	256	64	80.0	81.1	20K
		128	85.1	86.7	75K
		256	82.6	82.7	285K
	512	64	78.6	79.4	20K
		128	85.2	86.1	75K
		256	**88.0**	**89.4**	285K
	1024	64	79.3	79.5	20K
		128	85.2	84.6	75K
		256	79.5	78.3	285K

Table A.8.: Grid search over batch size (BS), hidden state size (HC), and width (HU) and depth (HL) of the vector field for ODERNN model on Swisscrop dataset. The learning rate (lr) is set to 0.01.

BS	HC	HU	HL	Accuracy (%)	F1-score (%)	Parameters
512	128	128	1	85.3	86.0	90K
			2	84.3	85.5	110K
		256	1	85.4	87.0	125K
			2	85.2	86.9	190K
	256	128	1	75.2	74.0	280K
			2	73.3	74.5	300K
		256	1	74.6	73.2	350K
			2	73.4	74.6	415K
1024	128	128	1	85.4	86.8	90K
			2	85.5	86.8	110K
		256	1	86.5	88.2	125K
			2	85.1	86.9	190K
	256	128	1	**87.0**	**88.6**	280K
			2	86.2	88.3	300K
		256	1	86.8	87.7	350K
			2	85.8	87.7	415K

Table A.9.: Grid search over batch size (BS), and width (HU) and depth (HL) of the vector field for NCDE model on Swisscrop dataset. The hidden channels (HC) are set to 128.

BS	HU	HL	Accuracy (%)	F1-score (%)	Parameters
512	128	1	85.2	86.2	125K
		2	80.6	81.1	140K
		3	79.7	80.9	155K
	256	1	84.8	86.2	240K
		2	85.3	84.2	305K
		3	79.8	81.0	370K
1024	128	1	83.9	84.9	125K
		2	80.9	82.2	140K
		3	81.9	81.9	155K
	256	1	86.1	87.8	240K
		2	**86.1**	**87.9**	305K
		3	74.2	75.0	370K

Table A.10.: Grid search over batch size (BS) and depth (HL) of the vector field for the S-NCDE model on Swisscrop dataset. The hidden channels (HC) are set to 32 and the hidden units (HU) of the vector field to 128.

BS	HL	Accuracy (%)	F1-score (%)	Parameters
512	1	**86.1**	**87.2**	170K
	2	79.9	81.0	200K
1024	1	83.2	84.2	170K
	2	75.6	74.0	200K

Bibliography

[1] C. Gómez, J. C. White, and M. A. Wulder, "Optical remotely sensed time series data for land cover classification: A review," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 116, pp. 55–72, 2016, ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2016.03.008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271616000769.

[2] C. Pelletier, G. I. Webb, and F. Petitjean, "Temporal convolutional neural network for the classification of satellite image time series," *Remote Sensing*, vol. 11, no. 5, 2019, ISSN: 2072-4292. DOI: 10.3390/rs11050523. [Online]. Available: https://www.mdpi.com/2072-4292/11/5/523.

[3] J. Inglada, M. Arias, B. Tardy, O. Hagolle, S. Valero, D. Morin, G. Dedieu, G. Sepulcre, S. Bontemps, P. Defourny, and B. Koetz, "Assessment of an operational system for crop type map production using high temporal and spatial resolution satellite optical imagery," *Remote Sensing*, vol. 7, no. 9, pp. 12 356–12 379, 2015, ISSN: 2072-4292. DOI: 10.3390/rs70912356. [Online]. Available: https://www.mdpi.com/2072-4292/7/9/12356.

[4] C. Conrad, S. Fritsch, J. Zeidler, G. Rücker, and S. Dech, "Per-field irrigated crop classification in arid central asia using spot and aster data," *Remote Sensing*, vol. 2, no. 4, pp. 1035–1056, 2010, ISSN: 2072-4292. DOI: 10.3390/rs2041035. [Online]. Available: https://www.mdpi.com/2072-4292/2/4/1035.

[5] S. Siachalou, G. Mallinis, and M. Tsakiri-Strati, "A hidden markov models approach for crop classification: Linking crop phenology to time series of multi-sensor remote sensing data," *Remote Sensing*, vol. 7, no. 4, pp. 3633–3650, 2015, ISSN: 2072-4292. DOI: 10.3390/rs70403633. [Online]. Available: https://www.mdpi.com/2072-4292/7/4/3633.

[6] M. Belgiu and O. Csillik, "Sentinel-2 cropland mapping using pixel-based and object-based time-weighted dynamic time warping analysis," *Remote Sensing of Environment*, vol. 204, pp. 509–523, 2018, ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2017.10.005. [Online]. Available: https://doi.org/10.1016/j.rse.2017.10.005.

[7] K. Henle, D. Alard, J. Clitherow, P. Cobb, L. Firbank, T. Kull, D. McCracken, R. F. Moritz, J. Niemelä, M. Rebane, D. Wascher, A. Watt, and J. Young, "Identifying and managing the conflicts between agriculture and biodiversity conservation in europe–a review," *Agriculture, Ecosystems & Environment*, vol. 124, no. 1-2, pp. 60–71, Mar. 2008. DOI: 10.1016/j.agee.2007.09.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167880907002319?via3Dihub.

[8] L. Horrigan, R. S. Lawrence, and P. Walker, "How sustainable agriculture can address the environmental and human health harms of industrial agriculture," *Environmental Health Perspectives*, vol. 110, no. 5, pp. 445–456, May 2002. DOI: 10.1289/ehp.02110445. [Online]. Available: https://doi.org/10.1289%2Fehp.02110445.

[9] J. M. Peña-Barragán, M. K. Ngugi, R. E. Plant, and J. Six, "Object-based crop identification using multiple vegetation indices, textural features and crop phenology," *Remote Sensing of Environment*, vol. 115, no. 6, pp. 1301–1316, 2011, ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2011.01.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0034425711000290.

[10] J. B. Odenweller and K. I. Johnson, "Crop identification using landsat temporal-spectral profiles," *Remote Sensing of Environment*, vol. 14, no. 1, pp. 39–54, 1984, ISSN: 0034-4257. DOI: https://doi.org/10.1016/0034-4257(84)90006-3. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0034425784900063.

[11] B. D. Wardlow and S. L. Egbert, "Large-area crop mapping using time-series modis 250 m ndvi data: An assessment for the u.s. central great plains," *Remote Sensing of Environment*, vol. 112, no. 3, pp. 1096–1116, 2008, ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2007.07.019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0034425707003458.

[12] S. Foerster, K. Kaden, M. Foerster, and S. Itzerott, "Crop type mapping using spectral–temporal profiles and phenological information," *Computers and Electronics in Agriculture*, vol. 89, pp. 30–40, 2012, ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2012.07.015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169912002013.

[13] F. Vuolo, M. Neuwirth, M. Immitzer, C. Atzberger, and W.-T. Ng, "How much does multi-temporal sentinel-2 data improve crop type classification?" *International Journal of Applied Earth Observation and Geoinformation*, vol. 72, pp. 122–130, 2018, ISSN: 0303-2434. DOI: https://doi.org/10.1016/j.jag.2018.06.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0303243418303167.

[14] Q. Hu, W.-b. Wu, Q. Song, M. Lu, D. Chen, Q.-y. Yu, and H.-j. Tang, "How do temporal and spectral features matter in crop classification in heilongjiang province, china?" *Journal of Integrative Agriculture*, vol. 16, no. 2, pp. 324–336, 2017, ISSN: 2095-3119. DOI: https://doi.org/10.1016/S2095-3119(15)61321-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2095311915613211.

[15] M. Rußwurm and M. Körner, "Temporal vegetation modelling using long short-term memory networks for crop identification from medium-resolution multi-spectral satellite images," in *2017 IEEE Conference on Computer Vision and*

Pattern Recognition Workshops (CVPRW), 2017, pp. 1496–1504. DOI: 10.1109/CVPRW.2017.193.

[16] K. S. Fu, D. A. Landgrebe, and T. L. Phillips, "Information processing of remotely sensed agricultural data," *Proceedings of the IEEE*, vol. 57, no. 4, pp. 639–653, 1969. DOI: 10.1109/PROC.1969.7019.

[17] N. Matton, G. S. Canto, F. Waldner, S. Valero, D. Morin, J. Inglada, M. Arias, S. Bontemps, B. Koetz, and P. Defourny, "An automated method for annual cropland mapping along the season for various globally-distributed agrosystems using high spatial and temporal resolution time series," *Remote Sensing*, vol. 7, no. 10, pp. 13 208–13 232, 2015, ISSN: 2072-4292. DOI: 10.3390/rs71013208. [Online]. Available: https://www.mdpi.com/2072-4292/7/10/13208.

[18] S. Valero, D. Morin, J. Inglada, G. Sepulcre, M. Arias, O. Hagolle, G. Dedieu, S. Bontemps, P. Defourny, and B. Koetz, "Production of a dynamic cropland mask by processing remote sensing image series at high temporal and spatial resolutions," *Remote Sensing*, vol. 8, no. 1, 2016, ISSN: 2072-4292. DOI: 10.3390/rs8010055. [Online]. Available: https://www.mdpi.com/2072-4292/8/1/55.

[19] G. Camps-Valls, L. Gomez-Chova, J. Munoz-Mari, J. L. Rojo-Alvarez, and M. Martinez-Ramon, "Kernel-based framework for multitemporal and multisource remote sensing data classification and change detection," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 46, no. 6, pp. 1822–1835, 2008. DOI: 10.1109/TGRS.2008.916201.

[20] M. Ustuner, F. B. Sanli, S. Abdikan, M. T. Esetlili, and Y. Kurucu, "Crop type classification using vegetation indices of rapideye imagery," *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XL-7, pp. 195–198, 2014. DOI: 10.5194/isprsarchives-XL-7-195-2014. [Online]. Available: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XL-7/195/2014/.

[21] G. Mountrakis, J. Im, and C. Ogole, "Support vector machines in remote sensing: A review," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 66, no. 3, pp. 247–259, 2011, ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2010.11.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271610001140.

[22] X. X. Zhu, D. Tuia, L. Mou, G. Xia, L. Zhang, F. Xu, and F. Fraundorfer, "Deep learning in remote sensing: A comprehensive review and list of resources," *IEEE Geoscience and Remote Sensing Magazine*, vol. 5, no. 4, pp. 8–36, 2017. DOI: 10.1109/MGRS.2017.2762307.

[23] M. Reichstein, G. Camps-Valls, B. Stevens, M. Jung, J. Denzler, N. Carvalhais, and Prabhat, "Deep learning and process understanding for data-driven earth system science," *Nature*, vol. 566, no. 7743, pp. 195–204, Feb. 2019. DOI: 10.1038/s41586-019-0912-1. [Online]. Available: https://doi.org/10.1038%2Fs41586-019-0912-1.

[24] B. Kellenberger, M. Volpi, and D. Tuia, "Fast animal detection in uav images using convolutional neural networks," in *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, 2017, pp. 866–869. DOI: 10.1109/IGARSS.2017.8127090.

[25] N. Audebert, B. Le Saux, and S. Lefèvre, "Segment-before-detect: Vehicle detection and classification through semantic segmentation of aerial images," *Remote Sensing*, vol. 9, no. 4, 2017, ISSN: 2072-4292. DOI: 10.3390/rs9040368. [Online]. Available: https://www.mdpi.com/2072-4292/9/4/368.

[26] M. Volpi and D. Tuia, "Dense semantic labeling of subdecimeter resolution images with convolutional neural networks," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 2, pp. 881–893, 2017. DOI: 10.1109/TGRS.2016.2616585.

[27] E. Ndikumana, D. Ho Tong Minh, N. Baghdadi, D. Courault, and L. Hossard, "Deep recurrent neural network for agricultural classification using multitemporal sar sentinel-1 for camargue, france," *Remote Sensing*, vol. 10, no. 8, 2018, ISSN: 2072-4292. DOI: 10.3390/rs10081217. [Online]. Available: https://www.mdpi.com/2072-4292/10/8/1217.

[28] Z. Sun, L. Di, and H. Fang, "Using long short-term memory recurrent neural network in land cover classification on landsat and cropland data layer time series," *International Journal of Remote Sensing*, vol. 40, no. 2, pp. 593–614, 2019. DOI: 10.1080/01431161.2018.1516313. [Online]. Available: https://doi.org/10.1080/01431161.2018.1516313.

[29] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Advances in Neural Information Processing Systems*, vol. 31, 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

[30] Y. Rubanova, R. T. Q. Chen, and D. K. Duvenaud, "Latent ordinary differential equations for irregularly-sampled time series," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019, pp. 5320–5330. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/42a6845a557bef704ad8ac9cb4461d43-Paper.pdf.

[31] N. Metzger, M. O. Turkoglu, S. D'Aronco, J. D. Wegner, and K. Schindler, "Crop classification under varying cloud cover with neural ordinary differential equations," 2020. arXiv: 2012.02542 [cs.CV].

[32] P. Kidger, J. Morrill, J. Foster, and T. Lyons, "Neural Controlled Differential Equations for Irregular Time Series," in *Advances in Neural Information Processing Systems*, vol. 33, 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/4a5876b450b45371f6cfe5047ac8cd45-Paper.pdf.

[33] T. J. Lyons, M. Caruana, and T. Lévy, *Differential Equations Driven by Rough Paths*. Springer Berlin Heidelberg, 2007. DOI: 10.1007/978-3-540-71285-5. [Online]. Available: https://doi.org/10.1007%2F978-3-540-71285-5.

[34] I. Chuine and J. Régnière, "Process-based models of phenology for plants and animals," *Annual Review of Ecology, Evolution, and Systematics*, vol. 48, no. 1, pp. 159–182, 2017. DOI: 10.1146/annurev-ecolsys-110316-022706. [Online]. Available: https://doi.org/10.1146/annurev-ecolsys-110316-022706.

[35] J. S. Clark, J. Melillo, J. Mohan, and C. Salk, "The seasonal timing of warming that controls onset of the growing season," *Global Change Biology*, vol. 20, no. 4, pp. 1136–1145, 2014. DOI: https://doi.org/10.1111/gcb.12420. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1111/gcb.12420.

[36] B. Drepper, A. Gobin, S. Remy, and J. V. Orshoven, "Comparing apple and pear phenology and model performance: What seven decades of observations reveal," *Agronomy*, vol. 10, no. 1, p. 73, Jan. 2020. DOI: 10.3390/agronomy10010073. [Online]. Available: https://doi.org/10.3390%2Fagronomy10010073.

[37] T. Qiu, C. Song, J. S. Clark, B. Seyednasrollah, N. Rathnayaka, and J. Li, "Understanding the continuous phenological development at daily time step with a bayesian hierarchical space-time model: Impacts of climate change and extreme weather events," *Remote Sensing of Environment*, vol. 247, p. 111 956, 2020, ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2020.111956. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0034425720303266.

[38] J. I. Fisher, A. D. Richardson, and J. F. Mustard, "Phenology model from surface meteorology does not capture satellite-based greenup estimations," *Global Change Biology*, vol. 13, no. 3, pp. 707–721, 2007. DOI: https://doi.org/10.1111/j.1365-2486.2006.01311.x. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2486.2006.01311.x.

[39] Y. Fu, H. Zhang, W. Dong, and W. Yuan, "Comparison of phenology models for predicting the onset of growing season over the northern hemisphere," *PLOS ONE*, vol. 9, no. 10, pp. 1–12, Oct. 2014. DOI: 10.1371/journal.pone.0109544. [Online]. Available: https://doi.org/10.1371/journal.pone.0109544.

[40] R. Stöckli, T. Rutishauser, D. Dragoni, J. O'Keefe, P. E. Thornton, M. Jolly, L. Lu, and A. S. Denning, "Remote sensing data assimilation for a prognostic phenology model," *Journal of Geophysical Research: Biogeosciences*, vol. 113, no. G4, Nov. 2008. DOI: 10.1029/2008jg000781. [Online]. Available: https://doi.org/10.1029%2F2008jg000781.

[41] M. O. Turkoglu, S. D'Aronco, G. Perich, F. Liebisch, C. Streit, K. Schindler, and J. D. Wegner, "Crop mapping from image time series: Deep learning with multi-scale label hierarchies," 2021. arXiv: 2102.08820 [cs.CV].

[42] C. Ünsalan and K. L. Boyer, "Review on land use classification," in *Multispectral Satellite Image Understanding*, Springer London, 2011, pp. 49–64. DOI: 10.1007/978-0-85729-667-2_5. [Online]. Available: https://doi.org/10.1007%2F978-0-85729-667-2_5.

[43] J. R. Anderson, E. E. Hardy, J. T. Roach, and R. E. Witmer, "A land use and land cover classification system for use with remote sensor data," *USGS*, 1976. DOI: 10.3133/pp964. [Online]. Available: https://doi.org/10.3133%2Fpp964.

[44] D. J. Belcher, E. E. Hardy, and E. S. Phillips, "Land use classification with simulated satellite photography," United States Department of Agriculture, Economic Research Service, Agricultural Information Bulletins 309024, Apr. 1971. DOI: 10.22004/ag.econ.309024. [Online]. Available: https://ideas.repec.org/p/ags/uersab/309024.html.

[45] J. W. Rouse, "Monitoring vegetation systems in the great plains with erts," *Third ERTS Symposium, NASA, Washington, DC., 1973*, vol. 1, pp. 309–317, 1973. [Online]. Available: https://ntrs.nasa.gov/citations/19740022614.

[46] C. Conrad, S. Dech, O. Dubovyk, S. Fritsch, D. Klein, F. Löw, G. Schorcht, and J. Zeidler, "Derivation of temporal windows for accurate crop discrimination in heterogeneous croplands of uzbekistan using multitemporal rapideye images," *Computers and Electronics in Agriculture*, vol. 103, pp. 63–74, 2014, ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2014.02.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169914000386.

[47] T. Hoberg, F. Rottensteiner, R. Q. Feitosa, and C. Heipke, "Conditional random fields for multitemporal and multiscale classification of optical satellite imagery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 2, pp. 659–673, 2015. DOI: 10.1109/TGRS.2014.2326886.

[48] S. Bailly, S. Giordano, L. Landrieu, and N. Chehata, "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*, 2018, pp. 1950–1953. DOI: 10.1109/IGARSS.2018.8518427.

[49] M. Rußwurm and M. Körner, "Multi-temporal land cover classification with sequential recurrent encoders," *ISPRS International Journal of Geo-Information*, vol. 7, no. 4, 2018, ISSN: 2220-9964. DOI: 10.3390/ijgi7040129. [Online]. Available: https://www.mdpi.com/2220-9964/7/4/129.

[50] M. Rußwurm and M. Körner, "Convolutional lstms for cloud-robust segmentation of remote sensing imagery," *Proceedings of the Conference on Neural Information Processing Systems Workshops (NeurIPSW)*, 2018.

[51] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, Curran Associates, Inc.,

2015. [Online]. Available: https://proceedings.neurips.cc/paper/2015/file/07563a3fe3bbe7e3ba84431ad9d055af-Paper.pdf.

[52] R. M Rustowicz, R. Cheong, L. Wang, S. Ermon, M. Burke, and D. Lobell, "Semantic segmentation of crop type in africa: A novel dataset and analysis of deep learning methods," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2019. [Online]. Available: https://www.gov.uk/research-for-development-outputs/semantic-segmentation-of-crop-type-in-africa-a-novel-dataset-and-analysis-of-deep-learning-methods.

[53] M. O. Turkoglu, S. D'Aronco, J. Wegner, and K. Schindler, "Gating revisited: Deep multi-layer RNNs that can be trained," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021. DOI: 10.1109/tpami.2021.3064878. [Online]. Available: https://doi.org/10.1109%2Ftpami.2021.3064878.

[54] N. D. Mauro, A. Vergari, T. Basile, F. Ventola, and F. Esposito, "End-to-end learning of deep spatio-temporal representations for satellite image time series classification," in *DC@PKDD/ECML*, 2017. [Online]. Available: https://www.semanticscholar.org/paper/End-to-end-Learning-of-Deep-Spatio-temporal-for-Mauro-Vergari/49eb731ef1d29d75f52c9b1ea538565da963d1ad.

[55] L. Zhong, L. Hu, and H. Zhou, "Deep learning based multi-temporal crop classification," *Remote Sensing of Environment*, vol. 221, pp. 430–443, 2019, ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2018.11.032. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0034425718305418.

[56] M. Rußwurm, S. Lefèvre, and M. Körner, "BreizhCrops: A Satellite Time Series Dataset for Crop Type Identification," in *Time Series Workshop of the 36th International Conference on Machine Learning (ICML)*, Long Beach, United States, 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02343898.

[57] M. Rußwurm and M. Körner, "Self-attention for raw optical satellite time series classification," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 169, pp. 421–435, 2020, ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2020.06.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271620301647.

[58] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[59] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, Dec. 1943. DOI: 10.1007/bf02478259. [Online]. Available: https://doi.org/10.1007%2Fbf02478259.

[60] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, Dec. 1989. DOI: 10.1007/bf02551274. [Online]. Available: https://doi.org/10.1007%2Fbf02551274.

[61] K.-I. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, no. 3, pp. 183–192, 1989, ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90003-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900038.

[62] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, no. 6, pp. 861–867, Jan. 1993. DOI: 10.1016/s0893-6080(05)80131-5. [Online]. Available: https://doi.org/10.1016%2Fs0893-6080%2805%2980131-5.

[63] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 930–945, 1993. DOI: 10.1109/18.256500.

[64] S. Agatonovic-Kustrin and R. Beresford, "Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research," *Journal of Pharmaceutical and Biomedical Analysis*, vol. 22, no. 5, pp. 717–727, 2000, ISSN: 0731-7085. DOI: https://doi.org/10.1016/S0731-7085(99)00272-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0731708599002721.

[65] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[66] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016. arXiv: 1609.04747. [Online]. Available: http://arxiv.org/abs/1609.04747.

[67] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980.

[68] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362, ISBN: 026268053X.

[69] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990. DOI: https://doi.org/10.1207/s15516709cog1402_1. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1.

[70] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994. DOI: 10.1109/72.279181.

[71] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of the 30th International Conference on Machine Learning*, S. Dasgupta and D. McAllester, Eds., ser. Proceedings of Machine Learning Research, vol. 28, Atlanta, Georgia, USA: PMLR, Jun. 2013, pp. 1310–1318. [Online]. Available: http://proceedings.mlr.press/v28/pascanu13.html.

[72] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI: 10.1162/neco.1997.9.8.1735.

[73] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2000. DOI: 10.1162/089976600300015015.

[74] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017. DOI: 10.1109/TNNLS.2016.2582924.

[75] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179. [Online]. Available: https://www.aclweb.org/anthology/D14-1179.

[76] E. Weinan, "A proposal on machine learning via dynamical systems," English, *Communications in Mathematics and Statistics*, vol. 5, no. 1, pp. 1–11, Mar. 2017. DOI: 10.1007/s40304-017-0103-z.

[77] Y. Lu, A. Zhong, Q. Li, and B. Dong, "Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations," in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, Stockholmsmässan, Stockholm Sweden: PMLR, Jul. 2018, pp. 3276–3285. [Online]. Available: http://proceedings.mlr.press/v80/lu18d.html.

[78] E. Haber and L. Ruthotto, "Stable architectures for deep neural networks," *Inverse Problems*, vol. 34, no. 1, p. 014004, Dec. 2017. DOI: 10.1088/1361-6420/aa9a90. [Online]. Available: https://doi.org/10.1088/1361-6420/aa9a90.

[79] L. Ruthotto and E. Haber, "Deep neural networks motivated by partial differential equations," *Journal of Mathematical Imaging and Vision*, vol. 62, pp. 352–364, 2019. [Online]. Available: https://link.springer.com/article/10.1007%2Fs10851-019-00903-1.

[80] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7780459.

[81] E. Dupont, A. Doucet, and Y. W. Teh, "Augmented neural odes," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/21be9a4bd4f81549a9d1d241981cec3c-Paper.pdf.

[82] H. Zhang, X. Gao, J. Unterman, and T. Arodz, "Approximation capabilities of neural ODEs and invertible residual networks," in *Proceedings of the 37th International Conference on Machine Learning*, H. D. III and A. Singh, Eds., ser. Proceedings of Machine Learning Research, vol. 119, PMLR, Jul. 2020, pp. 11 086–11 095. [Online]. Available: http://proceedings.mlr.press/v119/zhang20h.html.

[83] J. Morrill, P. Kidger, C. Salvi, J. Foster, and T. Lyons, "Neural cdes for long time series via the log-ode method," 2020. arXiv: 2009.08295 [cs.LG].

[84] S. Massaroli, M. Poli, J. Park, A. Yamashita, and H. Asama, "Dissecting neural odes," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 3952–3963. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/293835c2cc75b585649498ee74b395f5-Paper.pdf.

[85] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko, *The mathematical theory of optimal processes*. New York, NY: Wiley, 1962. [Online]. Available: http://cds.cern.ch/record/234445.

[86] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent neural networks for multivariate time series with missing values," *Scientific Reports*, vol. 8, no. 1, Apr. 2018. DOI: 10.1038/s41598-018-24271-9. [Online]. Available: https://doi.org/10.1038%2Fs41598-018-24271-9.

[87] T. J. Stieltjes, "Recherches sur les fractions continues," fr, *Annales de la Faculté des sciences de Toulouse : Mathématiques*, vol. 1e série, 8, no. 4, J1–J122, 1894. [Online]. Available: www.numdam.org/item/AFST_1894_1_8_4_J1_0/.

[88] E. W. Weisstein, *Cubic Spline*. [Online]. Available: https://mathworld.wolfram.com/CubicSpline.html, accessed on 10-February-2021.

[89] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005, ISBN: 026218253X. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-28650-9_4.

[90] T. Lyons, "Rough paths, signatures and the modelling of functions on streams," in, S. Jang, Y. Kim, D. Lee, and I. Yie, Eds. Kyung Moon SA, 2014. [Online]. Available: https://ora.ox.ac.uk/objects/uuid:05191b9a-24f1-4407-9d94-fb1a96f63b9d.

[91] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[92] J. Louis, V. Debaecker, B. Pflug, M. Main-Knorn, J. Bieniarz, U. Mueller-Wilm, E. Cadau, and F. Gascon, "Sentinel-2 sen2cor: L2a processor for users," in *ESA Living Planet Symposium 2016*, L. Ouwehand, Ed., ser. ESA Special Publications (on CD), vol. SP-740, Spacebooks Online, Aug. 2016, pp. 1–8. [Online]. Available: https://elib.dlr.de/107381/.

[93] L. Prechelt, "Early stopping — but when?" In *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2012, pp. 53–67. DOI: 10.1007/978-3-642-35289-8_5. [Online]. Available: https://doi.org/10.1007%2F978-3-642-35289-8_5.

[94] J. Dormand and P. Prince, "A family of embedded runge-kutta formulae," *Journal of Computational and Applied Mathematics*, vol. 6, no. 1, pp. 19–26, 1980, ISSN: 0377-0427. DOI: https://doi.org/10.1016/0771-050X(80)90013-3. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0771050X80900133.

[95] P. Kidger, R. T. Q. Chen, and T. Lyons, ""Hey, that's not an ODE": Faster ODE adjoints with 12 lines of code," 2020. arXiv: 2009.09457 [cs.LG]. [Online]. Available: https://openreview.net/forum?id=bzVsk7bnGdh.

[96] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, IEEE, Dec. 2015. DOI: 10.1109/iccv.2015.123. [Online]. Available: https://doi.org/10.1109%2Ficcv.2015.123.

[97] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: training imagenet in 1 hour," *CoRR*, vol. abs/1706.02677, 2017. arXiv: 1706.02677. [Online]. Available: http://arxiv.org/abs/1706.02677.

[98] S. Jastrzębski, Z. Kenton, D. Arpit, N. Ballas, A. Fischer, A. Storkey, and Y. Bengio, "Three factors influencing minima in SGD," 2018. [Online]. Available: https://openreview.net/forum?id=rJma2bZCW.

[99] M. Poli, S. Massaroli, A. Yamashita, H. Asama, and J. Park, "Hypersolvers: Toward fast continuous-depth models," in *Advances in Neural Information Processing Systems*, vol. 33, 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/f1686b4badcf28d33ed632036c7ab0b8-Paper.pdf.

[100] A. Norcliffe, C. Bodnar, B. Day, J. Moss, and P. Liò, "Neural ODE processes," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=27acGyyI1BY.