CEFET/RJ
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO

STCONVS2S: SPATIOTEMPORAL CONVOLUTIONAL SEQUENCE TO SEQUENCE NETWORK FOR WEATHER FORECASTING

Rafaela de Castro do Nascimento

Dissertation submitted to the Graduate Program in Computer Science of the Federal Center for Technological Education of Rio de Janeiro, CEFET/RJ, as partial fulfillment of the requirements for the degree of master.

Advisors:
Eduardo Bezerra, D.Sc.
Fábio Porto, D.Sc.

Rio de Janeiro,

July 2020

STConvS2S: Spatiotemporal Convolutional Sequence to Sequence Network for Weather Forecasting

Master's Dissertation in Computer Science, Federal Center for Technological Education of Rio de Janeiro, CEFET/RJ.

Rafaela de Castro do Nascimento

Approved by:

President, Eduardo Bezerra da Silva, D.Sc. (CEFET/RJ) (Advisor)

Fábio André Machado Porto, D.Sc. (LNCC) (Co-advisor)

Eduardo Soares Ogasawara, D.Sc. (CEFET/RJ)

José Antônio Fernandes de Macedo, D.Sc. (UFC)

Yania Molina Souto, D.Sc. (LNCC)

Rio de Janeiro,

July 2020

Ficha catalográfica elaborada pela Biblioteca Central do CEFET/RJ

N244 Nascimento, Rafaela de Castro do
STconvS2S: spatiotemporal convolutional sequence to sequence network for Weather Forecasting / Rafaela de Castro do Nascimento — 2020.
57f : il. color. , enc.

Dissertação (Mestrado) Centro Federal de Educação Tecnológica Celso Suckow da Fonseca , 2020.
Bibliografia : f. 52-57
Orientador: Eduardo Bezerra da Silva
Coorientador: Fábio André Machado Porto

1. Redes neurais (Computação). 2. Arquitetura de computador. 3. Previsão do tempo. 4. Meteorologia – Processamento de dados. I. Silva, Eduardo Bezerra da (Orient.). II. Porto, Fábio André Machado (Coorient.). III. Título.

CDD 006.3

Elaborada pela bibliotecária Tania Mello – CRB/7 nº 5507/04

DEDICATION

For the five most important people in my life,
reasons for a happier journey.

ACKNOWLEDGMENTS

Firstly, I would like to thank Prof. Eduardo Bezerra for his supervision, availability and all the help throughout the dissertation. Also, I would like to highlight his competence in making Machine Learning classes enjoyable, even those filled with mathematical formulations. Then, Yania Souto, who patiently shared her knowledge and made this work possible. I really appreciate our discussions on the many topics surrounding this project. Furthermore, I would like to thank Fabio Porto for his valuable suggestions and careful reviews.

To all my friends. In particular, I would like to express my gratitude to my leader at TRF 2ª Região, Markenson, for his understanding and support over the past 2 years. Next, my co-worker Chrystinne, for our many conversations about the academic path (which I insistently denied for the first 4 years). In addition, I would like to thank Ana Carolina, Pammela, Rickson and Emanoele, for their encouragement and patience in listening to my complaints.

Last but not least, I am infinitely thankful to my family. My parents, Sonia and Romero, and dear sisters Paloma, Carolina and Camila, I will be forever grateful for your love and positivity. Thank you for always being there, for the many conversations and knowledge sharing. You know that for me it is all about the journey, not the accomplishment itself. It was definitely quite a journey, and your continued support has helped me through this path.

I will be calmly active, actively calm.

Paramahansa Yogananda

RESUMO

STConvS2S: rede convolucional espaço-temporal para tarefa de sequência a sequência aplicada à previsão do tempo

Rafaela de Castro do Nascimento

Orientadores:
Eduardo Bezerra, D.Sc.
Fábio Porto, D.Sc.

Resumo da Dissertação submetida ao Programa de Pós-graduação em Ciência da Computação do Centro Federal de Educação Tecnológica Celso Suckow da Fonseca CEFET/RJ como parte dos requisitos necessários à obtenção do grau de mestre.

Aplicar modelos de aprendizagem de máquina em dados meteorológicos proporcionam muitas oportunidades na área da Geociência, como prever a condição do tempo de forma mais precisa. Recentemente, a modelagem dos dados meteorológicos com redes neurais profundas tem se tornado uma área de investigação relevante. Alguns trabalhos aplicam redes neurais recorrentes (RNN) ou uma abordagem híbrida usando RNN e redes neurais convolucionais (CNN). Neste trabalho, propusemos `STConvS2S` (rede convolucional espaço-temporal para tarefa de sequência a sequência), uma arquitetura de aprendizagem profunda construída para aprender as dependências espaciais e temporais dos dados usando somente camadas convolucionais. A arquitetura proposta resolve duas limitações das redes convolucionais ao prever sequências usando dados históricos, sendo: (1) elas violam a ordem temporal durante o processo de aprendizagem, e (2) precisam que o tamanho das sequências de entrada e saída sejam iguais. Experimentos computacionais usando dados de temperatura do ar e de chuva da América do Sul mostram que nossa arquitetura captura o contexto espaço-temporal e que ela é capaz de superar ou ter resultados comparáveis em relação às arquiteturas consideradas estado da arte na tarefa de previsão. Em particular, uma das variações da nossa arquitetura proposta melhora em 23% a previsão das sequências futuras, sendo quase cinco vezes mais rápida no treinamento do que os modelos baseados em RNN comparados nos experimentos.

Palavras-chave:
Análise de dados espaço-temporais; Modelos de sequência a sequência; Redes Neurais Convolucionais; Previsão do tempo

Rio de Janeiro,

Julho de 2020

ABSTRACT

STConvS2S: Spatiotemporal Convolutional Sequence to Sequence Network for Weather Forecasting

Rafaela de Castro do Nascimento

Advisors:
Eduardo Bezerra, D.Sc.
Fábio Porto, D.Sc.

Abstract of dissertation submitted to the Graduate Program in Computer Science of the Federal Center for Technological Education of Rio de Janeiro, CEFET/RJ, as partial fulfillment of the requirements for the degree of master.

Applying machine learning models to meteorological data brings many opportunities to the Geosciences field, such as predicting future weather conditions more accurately. In recent years, modeling meteorological data with deep neural networks has become a relevant area of investigation. These works apply either recurrent neural networks (RNN) or some hybrid approach mixing RNN and convolutional neural networks (CNN). In this work, we propose `STConvS2S` (Spatiotemporal Convolutional Sequence to Sequence Network), a deep learning architecture built for learning both spatial and temporal data dependencies using only convolutional layers. Our proposed architecture resolves two limitations of convolutional networks to predict sequences using historical data: (1) they violate the temporal order during the learning process and (2) they require the lengths of the input and output sequences to be equal. Computational experiments using air temperature and rainfall data from South America show that our architecture captures spatiotemporal context and that it outperforms or matches the results of state-of-the-art architectures for forecasting tasks. In particular, one of the variants of our proposed architecture is 23% better at predicting future sequences and almost five times faster at training than the RNN-based model used as a baseline.

Keywords:
Spatiotemporal data analysis; Sequence-to-Sequence models; Convolutional Neural Networks; Weather Forecasting

Rio de Janeiro,

July 2020

Contents

List of Figures

List of Tables

List of Abbreviations

AR	Autoregressive	6
ARIMA	Autoregressive Integrated Moving Average	6, 17, 18
CFSR	Climate Forecast System Reanalysis	4, 33
CHIRPS	Climate Hazards Group InfraRed Precipitation With Station Data	4, 33
CNN	Convolutional Neural Networks	10, 12, 19
CONVLSTM	Convolutional LSTM	18
FC-ANN	Fully-connected ANN	20
LSTM	Long Short-term Memory	14, 17
MA	Moving Average	6
MAE	Mean Absolute Error	5
MIM	Memory In Memory	18
MSE	Mean Squared Error	9
RMSE	Root Mean Squared Error	5
RNN	Recurrent Neural Networks	13, 14
ST-LSTM	Spatiotemporal LSTM	18
STARIMA	Space-time Autoregressive Integrated Moving Average Model	51

Chapter I Introduction

I.1 Contextualization

Many opportunities arise in the application of deep learning models to Earth-based tasks. Models originally designed for object recognition in images, video prediction, and language translation can be applied to classification of weather patterns in satellite images, spatiotemporal forecasting of meteorological variables, and time series modeling, respectively [Reichstein et al., 2019]. As a result, the number of researches in geoscience area using deep learning approaches is increasing and improving the ability to model this type of data.

Weather forecasting plays an essential role in resource planning in cases of severe natural phenomena such as heat waves (extreme temperatures), droughts, and hurricanes. Such prediction task helps to mitigate the unavoidable consequences of these phenomena and to calculate disaster risks, which is essential to support decisions in emergency response plans. The creation of weather prediction models involves knowledge of the current state of weather, the nearby spatial environment, and the changes in the atmosphere that rely on physical laws, such as thermodynamics [Rolnick et al., 2019].

Analyzing patterns in meteorological data is not only relevant to managing natural disaster risks, but is also sensitive information for business activities. Weather forecasting influences decision-making in agriculture, aviation, retail market, and other sectors, since unfavorable weather negatively impacts corporate revenues. Some examples are how weather events affect the growth of seeds and can lead to losses for farmers [Vedenov and Sanchez, 2011], or influence the decisions of consumers purchasing in the retail market [Štulec et al., 2019].

Meteorological data can be collected from weather stations, remote sensing, or climate models. For example, the remote sensing data collected by satellites provide meteorological data about the entire globe at specific time intervals (e.g., every 6 hours or daily) and with a regular spatial resolution (e.g., 1 Km or 5 Km). Figure I.1 illustrates three possible visualizations of the same meteorological data (temperature in this case), showing the spatial and temporal aspects of this type of data. Although it is possible to focus the analysis and understanding only on the patterns of temporal ordering (time series problems), a predictive system should also consider the geographical distribution to achieve a better representation of the data. In this work, we propose a predictive

model capable of learning the spatial and temporal contexts of meteorological data.

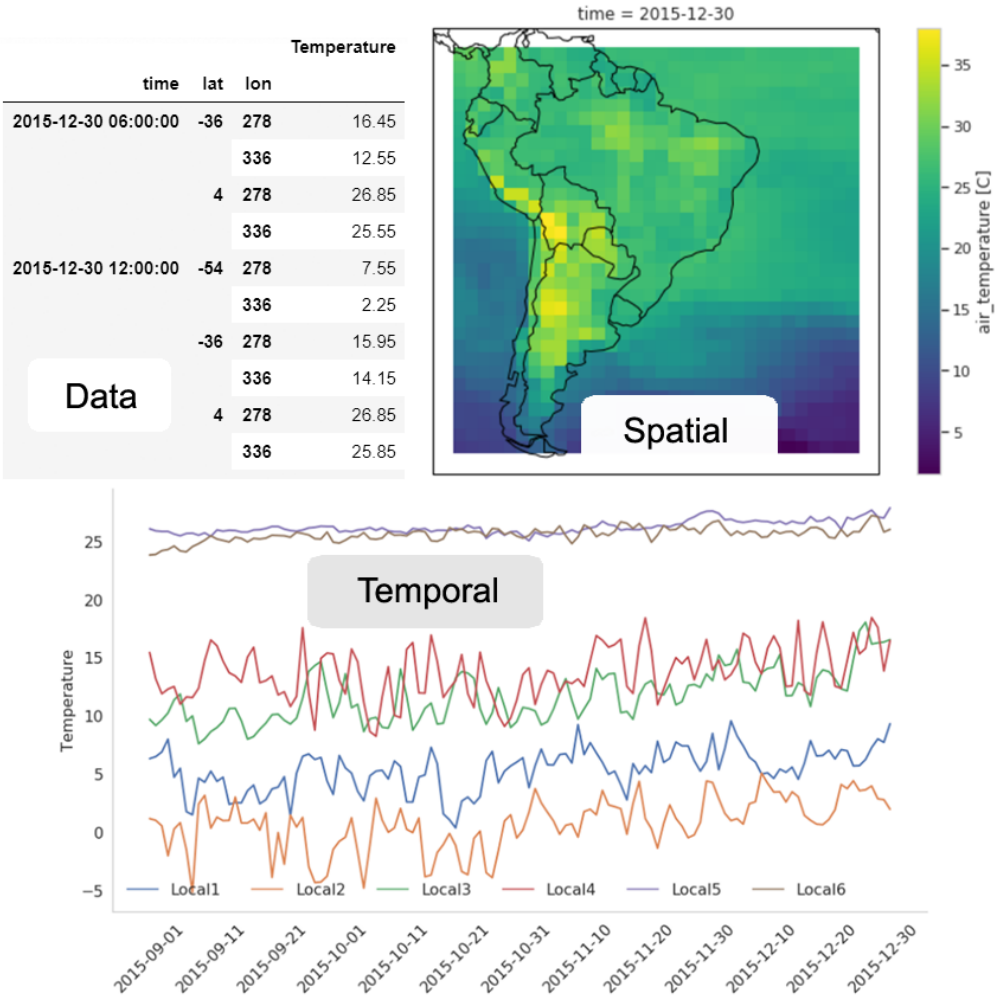


Figure I.1: Visualizations of temperature data. The raw data show variations in time and spatial coverage (latitude and longitude). The spatial view highlights different temperature values in each geographic point for a fixed day. The time view shows daily values measured in a time window (three months) for six locations.

I.2 Motivation

Over the years, with technological development, predictions of meteorological variables are becoming more accurate. However, due to the stochastic behavior of the Earth system, which is governed by physical laws, traditional forecasting requires complex, physics-based models to predict the weather [Karpatne et al., 2018]. In recent years, a large volume of data about the Earth systems has become available. This availability of historical data allows researchers to design deep learning models that can make more accurate predictions about the weather [Reichstein et al., 2019].

Even though meteorological data exhibits both spatial and temporal structures, weather forecasting can be modeled as a sequence-to-sequence problem. For example, given the measurements of a meteorological variable over time for a delimited region (input sequence), the goal is to predict

future values steps ahead for all spatial coverage simultaneously (output sequence) using its past observations (Figure I.2). In sequence modeling tasks, an input sequence is encoded to map the representation of the output sequence, which may have a different length than the input. In Shi et al. [2015], the authors proposed the ConvLSTM architecture to solve the sequence prediction problem using a radar echo dataset for precipitation forecasting. They integrated the convolution operator, adopted by the convolutional neural network (CNN), into a recurrent neural network (RNN) to simultaneously learn the spatial and temporal context of input data to predict the future sequence. Although ConvLSTM architecture has been considered the potential approach to build prediction models for geoscience data [Reichstein et al., 2019], new opportunities have emerged from recent advances in deep learning. In Wang et al. [2017, 2019], authors proposed improved versions of the long short-term memory (LSTM) unit for memorizing spatiotemporal information, and Souto et al. [2018] present an ensemble approach using ConvLSTM to improve predictions.

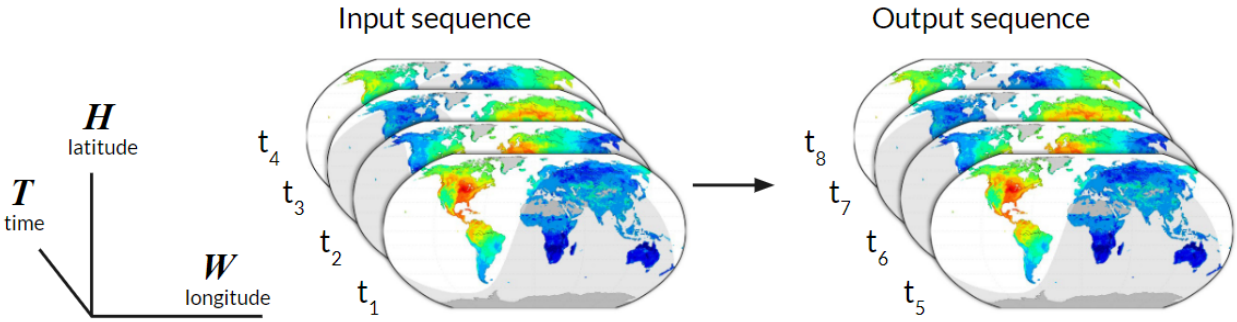


Figure I.2: Weather forecasting as a sequence-to-sequence problem [Reichstein et al., 2019].

RNN-based architectures may be ideal for multi-step forecasting tasks using spatiotemporal data [Shi et al., 2015; Souto et al., 2018; Wang et al., 2017, 2019], due to the ability to respect the temporal order (causal constraint) and predict long sequences. However, these architectures maintain the information from previous time steps to generate the output, which consequently leads to a high training time. Taking this as motivation, we address the spatiotemporal forecasting problem by proposing a new architecture using entirely 3D CNN. CNN are an efficient method for capturing spatial context and have attained state-of-the-art results for image classification using a 2D kernel [Krizhevsky et al., 2012]. In recent years, researchers expanded CNN actuation field, such as machine translation tasks [Gehring et al., 2017] using a 1D kernel, which is useful to capture temporal patterns in a sequence. 3D CNN-based models are commonly used for video analysis and action recognition [Yuan et al., 2018; Tran et al., 2018] or climate event detection [Racah et al., 2017]. However, CNN-based models are generally not considered for multi-step forecasting tasks, because of two intrinsic limitations. They violate the temporal order, allowing future information during temporal reasoning [Singh and Cuzzolin, 2019], and they cannot generate a predictive output sequence longer than the input sequence [Bai et al., 2018].

I.3 Objectives

To tackle the limitations presented previously, and inspired by the successful applications of convolutional models in other domains, the objective of this work is to employ deep learning techniques to develop a more accurate weather forecasting methodology at the study location. We introduce STConvS2S (*Spatiotemporal Convolutional Sequence to Sequence Network*), a spatiotemporal predictive model for multi-step forecasting task. To our knowledge, STConvS2S is the first 3D CNN-based architecture built as an end-to-end trainable model, suitable to satisfy the causal constraint and predict flexible length output sequences (i.e., not limited to be equal to the input sequence length).

The contributions of this work are twofold. Firstly, we provide two variants of the STConvS2S architecture that satisfy the causal constraint. One adapts the causal convolution in 3D convolutional layers, and the other introduces a new approach that strategically applies a reverse function in the sequence. Secondly, we devise a temporal generator block designed to extend the length of the output sequence, which encompasses a new application of the transposed convolutional layers.

I.4 Methodology

Machine learning models have a life cycle to build and validate hypotheses in order to solve a specific problem. This life cycle guided us through the exploratory process of creating a new deep learning architecture for weather forecasting (Figure I.3). The first activities are intended for pre-processing, aiming to extract knowledge from the selected data. The following activities are directed at the construction of deep learning architecture. Finally, the last activities evaluate the results obtained with the experiments. Further details on the activities are described below.

The data selection phase aims to extract the data from its source. The data comes from the Climate Forecast System Reanalysis (CFSR), a global dataset with associated meteorological variables. We also use data from Climate Hazards Group InfraRed Precipitation with Station data (CHIRPS), which is an almost global precipitation dataset. For both datasets, only data from South America and the surrounding ocean were selected to delimit the scope of the project. The data preparation activity aims to guarantee the quality of the data before being introduced in any learning algorithm. It focuses mainly on the cleaning and identification of the appropriate data used in the study.

Model creation is an activity designed to study and create different forecasting models. In this phase, we built a new spatiotemporal sequence to sequence network for weather forecasting. In the learning phase, we train our proposed model with the data previously processed in the initial phase. This activity is essential to identify the most suitable parameters in order to reduce errors

or training time.

The last activities are evaluation and validation. The former applies the previously trained model to the test data. If the algorithm presents an error that is not expected, it is necessary to return to the learning phase for new configurations of the hyperparameters. To evaluate the model, the following performance metrics are used: root mean squared error (RMSE) and mean absolute error (MAE). The objective of the last activity is to validate the predictive capacity of the model. The validation phase compares the accuracy of the prediction model guided in this work against state-of-the-art architectures and baselines.

An important point in the validation phase is to make a fair comparison between the models. In this way, we perform experiments defining initial hyperparameter configurations for deep learning models and propose variations in this configuration to observe the models' behavior. Another key configuration is the number of steps in the past considered in the analysis, defined equally for all experiments.

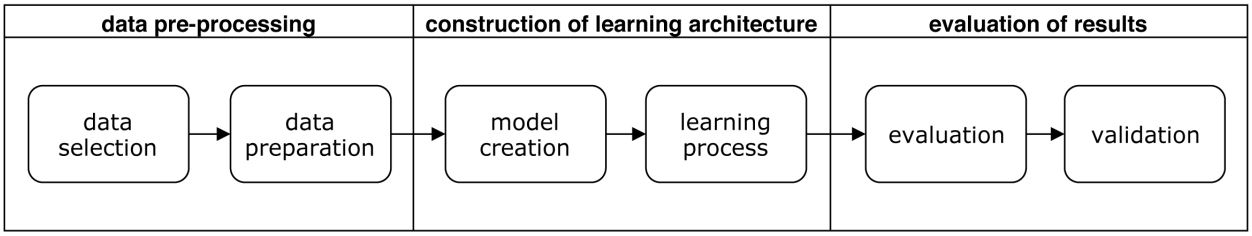


Figure I.3: Model construction life cycle

I.5 Organization

The rest of this dissertation is organized into five chapters. Chapter II presents an overview of the main concepts related to neural networks, especially deep neural networks. Chapter III discusses works related both to weather forecasting and spatiotemporal architectures. Chapter IV describes our proposed deep learning architecture. Chapter V presents our experiments and results. Chapter VI summarizes the conclusions and future work of the dissertation.

Chapter II Background

This chapter presents some background needed to understand further chapters. It starts with the basic concepts of ARIMA models, adopted as a baseline in the experiments (Section II.1). Next, it presents the main concepts to understand how deep learning models perform the training phase, and also delves into more advanced models that are used as the base networks for our proposed deep learning model and state-of-the-art models (Section II.2). Finally, it explains the concept of causal constraint in sequence-to-sequence tasks, providing examples in different domains (Section II.3).

II.1 ARIMA models

In temporal domain, time series is a sequence of historical measurements of an observable process taken at equal time intervals, where the measurements can be traffic flow, meteorological variables, for example. Autoregressive integrated moving average (ARIMA) [Box and Jenkins, 1970] are traditional methods for modeling stochastic process and one of the most used forecasting approaches for univariate time series [Hyndman and Athanasopoulos, 2018]. They are composed of an autoregressive (AR) model, a differencing method (I), and a moving average (MA) model, represented by the parameters p, d, and q, respectively. These statistical methods can handle stationarity in time series (mean and variance remain constant over time) and nonstationarity [Gujarati, 2002].

Given a integer p, the AR(p) model specify that the future estimation y_t depends linearly on its own previous values $[y_{t-p}, \ldots, y_{t-1}]$ in the form defined in Equation II.1, where δ is an intercept term (constant), p is the number of steps into past needed to predict y_t, ϕ are the parameters related to each of the past values and ε is the error term (white noise [1]) with $\varepsilon_t \sim \mathcal{N}(0, \sigma^2)$ [Brockwell and Davis, 2016].

$$y_t = \delta + \sum_{r=1}^{p} \phi_r y_{t-r} + \varepsilon_t \tag{II.1}$$

Rather than using past values, in the MA(q) model, for a given integer q, the y_t is described as a function of its last q values of white noise $[\varepsilon_{t-q}, \ldots, \varepsilon_{t-1}]$. The ARMA($p$,$q$) models are the

[1] An independent random variable

combination of these previous models, and the series is modeled using both its past values and errors. Formally, they are defined in Equation II.2, where θ are the parameters related to each of the past errors.

$$y_t = \delta + \sum_{r=1}^{p} \phi_r y_{t-r} + \varepsilon_t + \sum_{r=1}^{q} \theta_r \varepsilon_{t-r} \tag{II.2}$$

As ARMA modeling expects a stationary time series, in the way to handle nonstationarity, the differencing method can be adopted, forming the ARIMA(p,d,q) models. Nonstationarity (variations in mean and variance in time series) are often due to deterministic trends, but it can be manifested by other conditions as well [Salles et al., 2019]. Application of differencing method preliminary and, repeatedly, if necessary, is capable of generating a stationary time series. For instance, a first differencing transformation can eliminate a linear trend and is formulated as [Salles et al., 2019]:

$$y'_t = \nabla y_t = y_t - y_{t-1} \tag{II.3}$$

A particular model is the ARIMA(0,1,0), called random walk model, which is especially useful for time series exhibiting a random behavior against time. A random walk model, defined as in Equation II.4, assumes that the value of a time series y_t at a time t can be explained by the value of the series at the time $t-1$ and a random movement represented by ε_t (white noise) [Salles et al., 2019].

$$y_t = y_{t-1} + \varepsilon_t \tag{II.4}$$

II.2 Deep learning models

The ability to the system acquire knowledge by extracting patterns from raw data is known as machine learning [Goodfellow et al., 2016]. Deep learning can be seen as a subarea of machine learning, where one of the main differences is the capability to model complex data by increasing the number of layers for specific architectures. In this section, we provide the idea behind the learning process of these models and also the foundations of two deep neural networks.

II.2.1 Learning process

Machine learning models are algorithms that can learn from data and are suitable for a variety of tasks, such as classification, regression, anomaly detection, machine translation, etc. Based on the scope of this work, our explanation and example focus on the regression task, in which the model attempts to predict continuous data (numerical value) given an input. One type of learning

adopted by the model in the regression task is supervised learning, where for each input vector x, there is an associated target[2] vector (expected result) y. Thus, the process aims to learn how to predict y from x.

Figure II.1 illustrates an overview of the supervised learning process. The learning step can be split into two: forward and backward. During the forward pass, the model is fed with the input data to generate the prediction. The target and prediction data are compared by a loss function that evaluates their differences by computing a measure of error. This error is then applied to optimize the parameters (weights) of the model. The change in weights is computed by the backpropagation algorithm in the backward pass [Stevens and Antiga, 2019]. The learning process during training is iterative and its goal is to improve the parameters of the model in order to minimize the error and make better predictions.

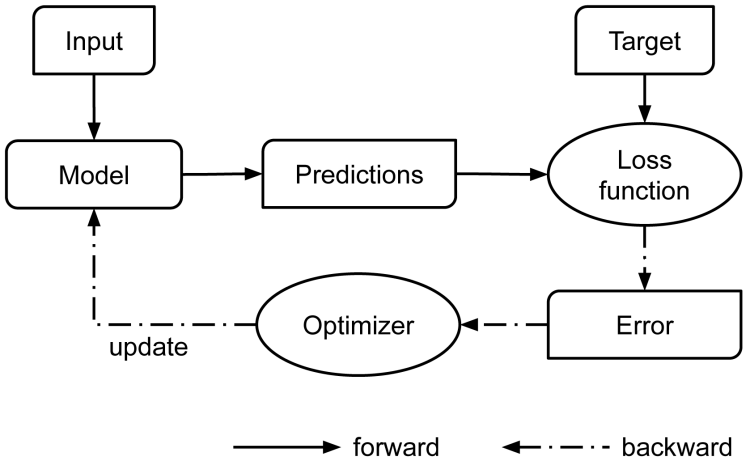


Figure II.1: Overview of the supervised learning process

In the following sections, the fundamental components used in the learning process are briefly presented.

Basic concepts and activation functions

Considering a machine learning model with one artificial neuron, the input value provided to this simple model are known as feature x_i, and for each feature, the neuron has an associated weight $\bar{w}_i$, where $1 \leqslant i \leqslant n$, and n is the number of features. Another parameter of the model aside from the weights are the bias b. Inside the neuron, pre-activation and activation steps are performed. In pre-activation, the model applies a linear transformation, which can be defined as a weighted sum of features (Equation II.5). Following, its output z is passed through the activation function (or nonlinearity) $g(\cdot)$. The nonlinearity allows the model to approximate more complex functions, in order to better represent the input data [Stevens and Antiga, 2019]. Figure II.2 schematically

[2]Also known as ground truth. These terms are used interchangeably in this work.

shows the basic structure of the artificial neuron. Formally, the predicted output $\hat{y}$ of the neuron can be expressed as:

$$z = \sum_{i=1}^{n} \bar{w}_i x_i + b \tag{II.5}$$

$$\hat{y} = g(z) \tag{II.6}$$

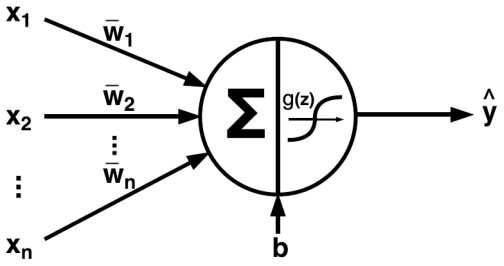


Figure II.2: Neuron with n features x_i, the learnable parameters (weights $\bar{w}_i$ and bias b), the activation function $g(\cdot)$, and at the end, the neuron generates the prediction $\hat{y}$ [Sautermeister, 2016].

Besides the nonlinear behavior, the activation functions need to be differentiable[3] so that the errors can be propagated (Section II.2.1) in the backward pass through them [Stevens and Antiga, 2019]. Figure II.3 presents some activation functions used in neural networks.

Optimization problem

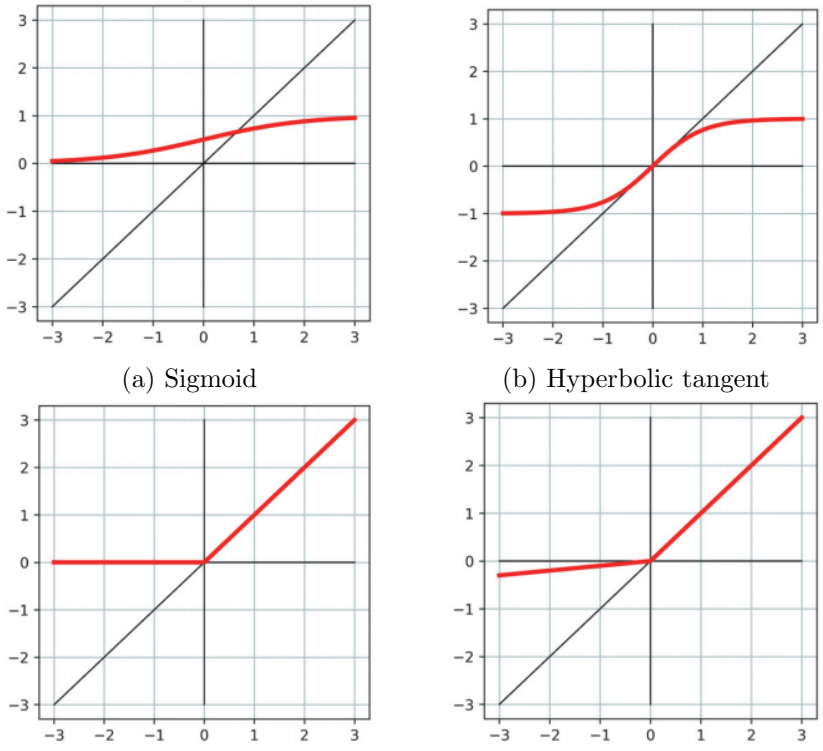
Previous section provides explanations about a simple neural network with a single artificial neuron. Complex architectures have many layers, each with many neurons inside. During the forward pass, features x is the initial information that is manipulated and propagated through each layer to produce the final output $\hat{y}$. Among the forward and backward pass, the error (loss) is calculated using a loss function[4]. This function computes a numerical value that the optimization process attempts to minimize. To measure how close to the true response the model is, the loss calculation involves taking the difference between the predicted output $\hat{y}$ and the desired output y. There are several loss functions for the regression task[5]. Equation II.7 shows mean squared error (MSE), one of the most used functions [James et al., 2013], given by

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y - \hat{y}_i)^2 \tag{II.7}$$

[3]At least in in most points of its domain, since ReLU is not differentiable at 0.

[4]Also referred to as cost function.

[5]Section V.3 provides the functions we have adopted in our work. It is important to note that in this section we refer to the accuracy of the predictions in the training phase, where the learning process takes place. Section V.3 refers to the evaluation phase applied to the unseen test data (not used during training).



(a) Sigmoid (b) Hyperbolic tangent

(c) Rectified linear unit (d) Leaky ReLU

Figure II.3: Activation functions [Stevens and Antiga, 2019]. (a) Sigmoid - the output is generated into range [0,1]. Basically, large negative numbers become 0 and large positive numbers become 1. (b) Hyperbolic tangent (tanh) - the output is generated in the range [-1,1] and is zero-centered. (c) Rectified Linear Unit (ReLU) - it computes the function $f(x) = max(0, x)$ and nowadays is the most used activation function in CNN. (d) LeakyReLU - variation of ReLU function, it has a small positive slope rather than being strictly zero for negative inputs.

where n is the number of samples used as input in the machine learning model.

In the backward pass the information (gradient) of the loss function J must be propagated back through the layers. Backpropagation is the algorithm to calculate the gradient $\nabla_{\bar{w},b}J$, with respect to the parameters, using the chain rule [Goodfellow et al., 2016]. This gradient is passed to an optimization method that uses it to adjust the parameters ($\bar{w}$ and b) of the models. Mathematically, the gradient $\nabla_{\bar{w}_i,b}J$ of a specific neuron can be defined as:

$$\nabla_{\bar{w}_i,b}J = \left(\frac{\partial J}{\partial \bar{w}_i}, \frac{\partial J}{\partial b}\right) = \left(\frac{\partial J}{\partial x'} \cdot \frac{\partial x'}{\partial \bar{w}_i}, \frac{\partial J}{\partial x'} \cdot \frac{\partial x'}{\partial b}\right) \tag{II.8}$$

where also with respect to the input x', the chain rule is used to calculate partial derivatives $\partial J/\partial \bar{w}_i$ and $\partial J/\partial b$.

II.2.2 Convolutional Neural Networks

Convolutional neural networks (CNN) are categorized as supervised learning algorithm, due to the type of experience they have during the learning process. They were studied in detail in LeCun

and Bengio [1995] for image, speech, and time series tasks, where the architecture was designed to process data with grid-like topology. Inspired by the visual cortex, the artificial neurons in this model use convolution operation to scan the input data and detect different kinds of patterns located in a local region, called receptive field. Convolution is a set of transformations that apply the same linear transformation of a small local region across the entire input [Goodfellow et al., 2016]. Formally in Equation II.9, convolution operation of two given functions f and q in terms of a variable v at point α is typically denoted with an asterisk and defined in the discrete domain as follows [Goodfellow et al., 2016]:

$$(f * q)(\alpha) = \sum_{v=-\infty}^{\infty} f(v)q(\alpha - v) \tag{II.9}$$

Here α can be seen as the offset which allows $q(\alpha - v)$ to slide along the v-axis. Similarly, in Equation II.10 the convolution operation applied to more than one axis at a time, for instance, 2-dimensional image I as input, and a 2-dimensional kernel K, is denoted by:

$$(I * K)(\alpha, \beta) = \sum_{h=1}^{H} \sum_{w=1}^{W} I(h, w)K(\alpha - h, \beta - w) \tag{II.10}$$

where $I(h, w)$ is the pixel at location (h, w) and $K(\alpha - h, \beta - w)$ gives the kernel's values (known as weights of the network) based on the pixel offsets α and β. Before CNN, to create the kernel used in image processing tasks, such as edge detection, weights were investigated manually. For example, to detect vertical and horizontal edges of an image, weights were defined similarly to

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

respectively (these specific kernels are called Sobel operator [Sobel and Feldman, 1968]). One of the main advantages of applying CNN in image processing is that the weights are learned by the network itself, removing the manual investigation of these values. Figure II.4 shows an example of a convolution operation [Burkov, 2019]. The output value of the first convolution, -3, was obtained as $(0 \times -1) + (0 \times 5) + (0 \times 4) + (1 \times -3)$. In the example, the input size is 4×4, the kernel size is 2×2, and the output will consist of a 3×3 matrix, result of sliding the kernel over the input. Thus, the output of each neuron after the convolution forms the feature map, and the kernel size defines the receptive field in the input. Since the convolution operation is a linear transformation, each convolutional layer is usually followed by an activation function.

The dimension of the feature map depends on two convolution properties: stride and padding. Stride controls how the kernel slides through the input, i.e., the position at which convolution

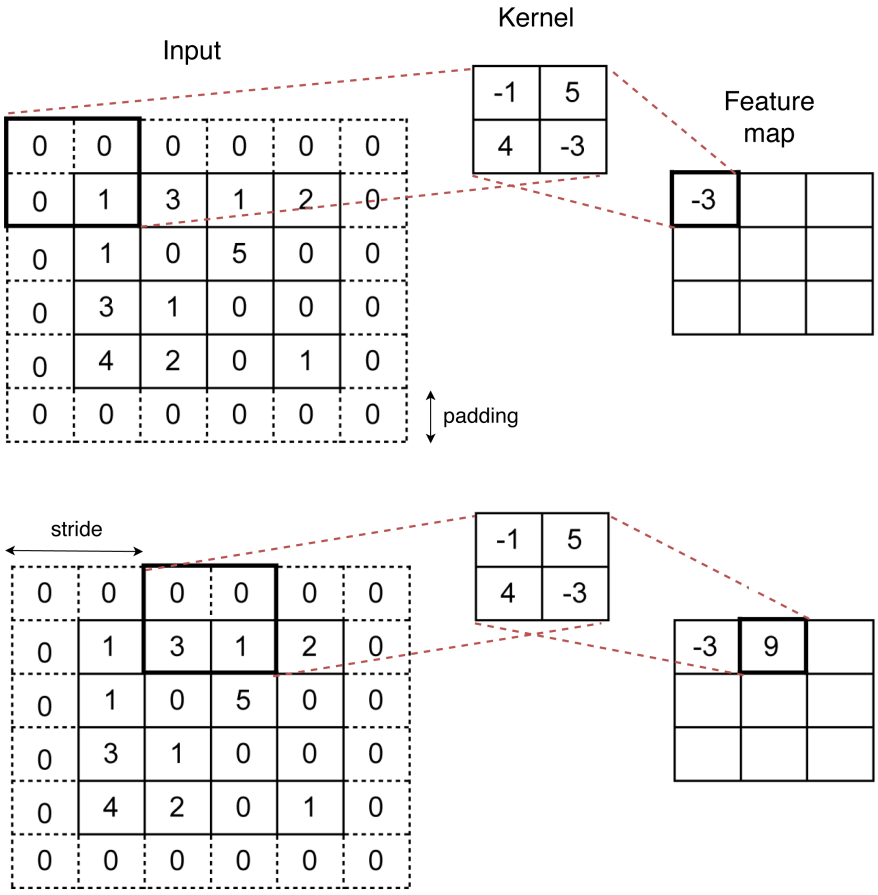


Figure II.4: A kernel convolving across an input. The size of the receptive field in the input is equal to the kernel size (2×2), the stride $= 2$ and padding $= 1$.

operation must begin for each element. The padding technique can be applied to ensure that the output feature map has the same dimension as the input data. This technique surrounds each slice of the input volume with p cells containing zeros. Thus, the size of the output can be defined in terms of input size r_{in}, stride s, kernel size k and padding p as $(r_{in} - k + 2p)/s + 1$.

Spatiotemporal Convolutional Neural Networks

CNN applied to images operates 2D convolutions, as explained in the previous section, capturing features in the spatial dimension only. When CNN is addressed to the spatiotemporal problem, such as video analysis, it also needs to capture motion information in the multiple frames. To accomplish this, 3D convolution is adopted instead of 2D convolution, and the layer can be named as 3D CNN [Tran et al., 2015]. Figure II.5 illustrates the difference in input and output representation when applying 2D and 3D convolution.

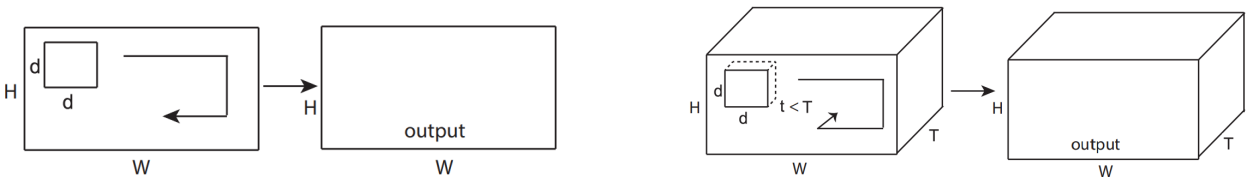


Figure II.5: Comparison of 2D and 3D convolution. (a) 2D convolution applied to an image, where H and W are height and width, generates 2D output. The 2D kernel is defined by $d \times d$, where d represents the size of the kernel (k) in the spatial dimension (H and W). (b) 3D convolution preserves the temporal information of the input resulting in an output volume. The input has 3 dimensions $T \times H \times W$, where H and W have the same definition as in (a), and T is the size of temporal dimension, i.e., the number of frames if the input is video clips. The kernel is $t \times d \times d$, where d is defined in (a) and t is the size of temporal depth. [Tran et al., 2015].

Transposed convolutional layer

Transposed convolutional layer[6] is widely used for semantic segmentation [Noh et al., 2015] and object detection [Racah et al., 2017] to reconstruct the shape of the input image after applying convolutional layers. Convolution operation can be interpreted as a many-to-one relationship, i.e., it associates multiple elements of input feature map with a single element of the generated feature map. In contrast, in a transposed convolutional layer, transposed convolution operation forms a one-to-many relationship, thus it can be used to generate an upsampled output [Noh et al., 2015] as shown in Figure II.6. Formally, the size of the output can be defined as $s(r_{in} - 1) + k - 2p$, for input size r_{in}, stride s, kernel size k and padding p [Dumoulin and Visin, 2016]. See Section II.2.2 for stride, kernel size and padding details.

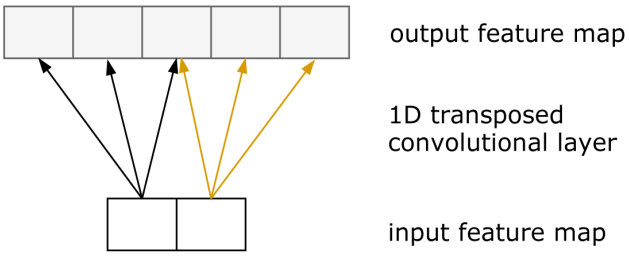


Figure II.6: Example of 1D transposed convolution operation where 2×1 input feature map ($r_{in} = 2$) is upsampled to 5×1 output feature map. This 1D transposed convolutional layer has as hyperparameters: $k = 3$, $s = 2$, $p = 0$.

II.2.3 Recurrent Neural Networks

Recurrent neural networks (RNN) are a type of neural networks in which their units are similar to the artificial neuron described in Section II.2.1, but with additional connections between layers (recurrent connections). This type of connection provides a chain-like structure, where the output

[6] Also termed as deconvolution in previous works of the literature.

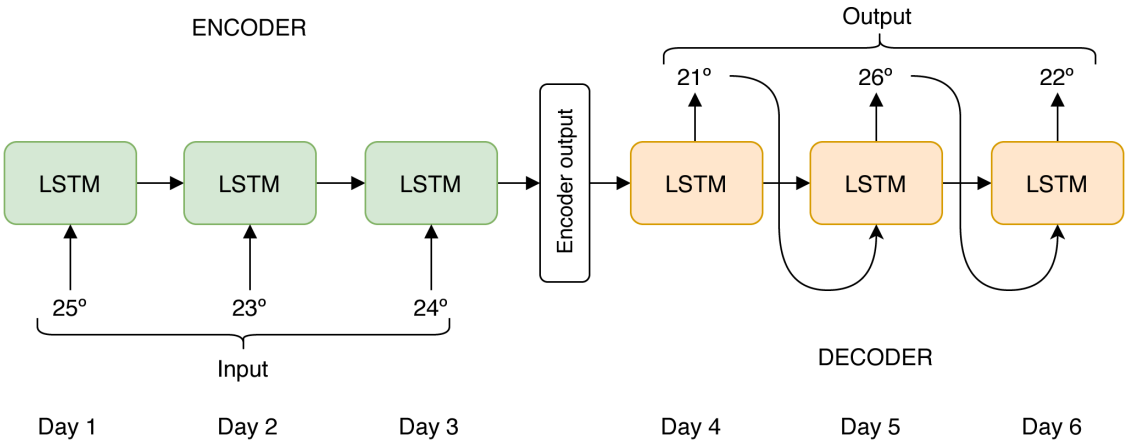


Figure II.7: The encoder formed by a stack of LSTM layers reads the input sequence one temperature at a time and then maps it to an intermediate representation called "encoder output". Another stack of LSTM layers decode the output sequence from the encoder output, generating the forecast for the next few days.

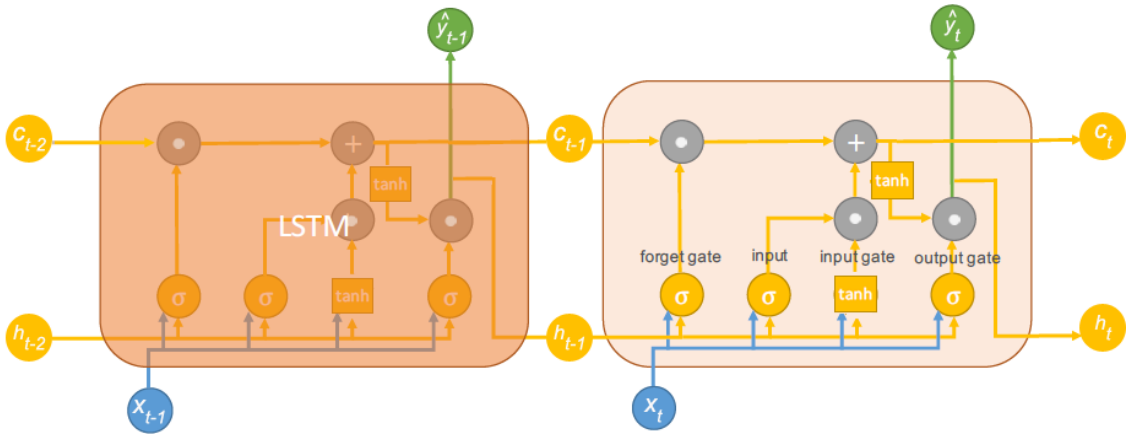


Figure II.8: Illustration of LSTM layer inside with the three gates: forget, input, and output. Among the LSTM layers are shown the recurrent connections [Zhang, 2017].

from one step is passed to the next step and so on, making it follow the causal constraint and suitable for sequence problems. Sequence modeling (or sequence-to-sequence learning) can be defined as a way of generating a model that maps an input sequence vector $x = [x_1, \ldots, x_n]$ of n elements to an output sequence vector $y = [y_1, \ldots, y_m]$, where the size of the sequences may be different. The encoder-decoder architecture, ideally designed for sequence modeling, was first proposed by Sutskever et al. [2014] for machine translation tasks using long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997], a type of RNN. This architecture can also be applied to other domains, such as forecasting tasks using time series data, as shown in Figure II.7.

Nowadays, LSTM are the standard RNN-based architecture to deal with higher temporal depth (long sequences) [Goodfellow et al., 2016]. They has a memory state (or cell state) c_t to keep the information from previous time steps. To control the flow of information from the current input x_t to memory, the layer uses a memory gate (or input gate). Other gating mechanisms are the

forget gate and output gate. The forget gate uses sigmoid function σ to regulate if it removes information from memory. The output gate controls the output of the layer $\hat{y}_t$, which is the also the information passed onwards, defined as the hidden state h_t. The output $\hat{y}_t$ is based on the new memory state after the operation on the forget and memory gate, on the previous hidden state h_{t-1} and on the current input x_t. This structure is shown schematically in Figure II.8 and the output can be formalized as [Shi et al., 2015]:

$$\hat{y}_t = \sigma(\bar{w}_x x_t + \bar{w}_h h_{t-1} + \bar{w}_c \odot c_t + b) \odot tanh(c_t) \tag{II.11}$$

where $\bar{w}_x$, $\bar{w}_h$ and $\bar{w}_c$ are the weights in relation to each state x, h and c, respectively, b is the bias, $tanh$ is the hyperbolic tangent function (see Section II.2.1) and $\odot$ operator denotes the Hadamard product[7].

A drawback of the information dependency from previous step is that LSTM does not allow parallel computation, leading to a slow training phase. Gehring et al. [2017] successfully propose a new architecture using only 1D CNNs for sequence modeling, designed with causal convolutions in decoder to capture temporal dependencies in sequences and, compared to LSTM models, computations can be completely parallelized during training.

II.3 Causal constraint in sequence-to-sequence tasks

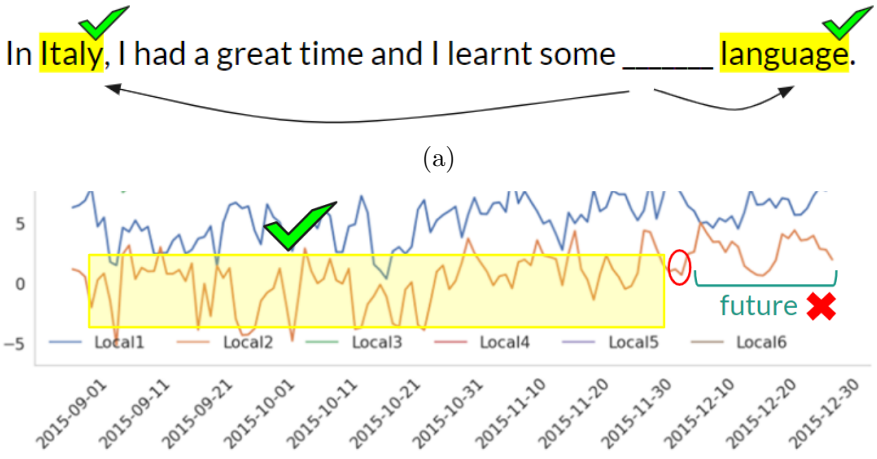


Figure II.9: Causal constraint in sequence-to-sequence tasks. (a) Speech recognition task: this model does not have to be causal, because to obtain the correct interpretation of the current word (represented as a blank space), it is necessary to look in both directions to understand the context. (b) Forecasting task: this model needs to be causal because, the model can only look in one direction to obtain the representation of the current time step (highlighted in the red circle), which means, it must not look to the future.

[7]It defines the element-wise product of two matrices with the same dimension, producing another matrix of the same dimension

Structuring data as a sequence is essential for modeling several real problems, since many observed tasks are sequential, such as communication, which is based on word sequences, or the measurement of a variable over time. In some sequence-to-sequence tasks, such as speech recognition, the desired output prediction may depend on the entire input sequence. Thus, to get the correct interpretation of the current word is necessary to look at both previous and next words in the sequence.

On the other hand, in tasks like weather forecasting, one of the main constraints is the temporal order. In other words, to properly learn the representation of the current time step, the model must not look at the next observations (future), because it violates the temporal order. In this sense, the domain in which the model is applied defines the need to satisfy the causal constraint or not. Figure II.9 provides examples of sequence-to-sequence tasks and shows which model needs to be causal.

Chapter III Related Work

Many studies have addressed weather forecasting as a time-series problem using types of neural networks [Corchado and Fyfe, 1999; Baboo and Shereef, 2010; Mehdizadeh, 2018; Zaytar and Amrani, 2016] (Section III.1). However, meteorological variables have variations in their dynamics also in space. Thus, spatiotemporal deep learning models are investigated in this chapter to tackle this problem (Section III.2). At the end, we present the main characteristics of these works and summarize the conclusions of this chapter (Section III.3).

III.1 Time series approach

Several statistical methods and machine learning techniques have been applied to historical data about temperature, precipitation, and other meteorological variables to predict the weather conditions. ARIMA are traditional statistical methods for times series analysis [Babu and Reddy, 2012]. Other studies have also applied artificial neural networks (ANN) to time series prediction in weather data, such as temperature measurements [Corchado and Fyfe, 1999; Baboo and Shereef, 2010; Mehdizadeh, 2018].

Recently, some authors have been developing new approaches based on deep learning to improve time series forecasting results, in particular, using LSTM networks. Traffic flow analysis [Yang et al., 2019], displacement prediction of landslide [Xu and Niu, 2018], petroleum production [Sagheer and Kotb, 2019] and sea surface temperature forecasting [Zhang et al., 2017] are some applications that successfully use LSTM architectures. In Zaytar and Amrani [2016], the authors build a model with stacked LSTM layers to map sequences of weather values (temperature, humidity, and wind speed) of the same length for 9 cities in Morocco and show that their results are competitive with traditional methods. However, these approaches (addressed to time series) are unable to capture the spatial dependencies in the observations.

III.2 Spatiotemporal approach

We separate this approach into two sections (Section III.2.1 and Section III.2.2) based on the type of deep learning layer adopted and in them we briefly discuss design decisions in the architecture of each work. Architectures presented in this section [Shi et al., 2015; Wang et al.,

2017, 2019; Tran et al., 2018; Racah et al., 2017] are used in the experimental evaluation (see Chapter V).

III.2.1 RNN-based models

Spatiotemporal deep learning models deal with spatial and temporal contexts simultaneously. In Shi et al. [2015], the authors formulate rainfall forecasting as a sequence-to-sequence problem, where the input and output are 2D radar map sequences. In addition, they introduce the ConvLSTM architecture to build an end-to-end model for precipitation nowcasting. The proposed model includes the convolution operation into LSTM network to capture spatial patterns. Formally, the output $\hat{y}$ is expressed as in Equation III.1. Comparing with Equation II.11, the difference between them is the replacement of multiplication by the convolution operation, denoted by $*$. In details, the future representation (state) of a cell is defined by the receptive field (green and red squares in Figure III.1) in the previous layer. This is the result of the convolution operation in the ConvLSTM layer that allows the encoding of spatial information from a neighborhood. Kim et al. [2019] also define their problem as a sequence task and adopt ConvLSTM for extreme climate event forecasting (Figure III.1). Their model uses hurricane density map sequences as spatiotemporal data.

$$\hat{y}_t = \sigma(\bar{w}_x * x_t + \bar{w}_h * h_{t-1} + \bar{w}_c \odot c_t + b) \odot tanh(c_t) \tag{III.1}$$

The work proposed in Souto et al. [2018] implements a spatiotemporal aware ensemble approach adopting ConvLSTM architecture. Ensemble approaches combine predictors to generate predictions that are weighted and possibly better than the predictions obtained by each predictor separately. The authors combine different meteorological models as channels in the convolutional layer to predict the next expected rainfall values for each location. This can be seen as a multiple regression model, since the variable of interest is predicted by a combination of predictors. Our approach, instead, can be seen as an autoregression model, because the variable of interest is predicted based on its own past data.

Based on Shi et al. [2015], Wang et al. [2017] present a new LSTM unit for ConvLSTM, called Spatiotemporal LSTM (ST-LSTM), that memorizes spatial and temporal variations in a unified memory pool. Their memory states flow through the network along a zigzag direction as shown in Figure III.2. ST-LSTM unit for PredRNN is able to deliver memory states both vertically (between layers) and horizontally (between time steps).

In Wang et al. [2019], they present an improved memory function in ST-LSTM unit adding non-stationarity modeling. They proposed an architecture, called Memory In Memory (MIM), which is inspired by the ARIMA models. They proposed two recurrent modules, where the first

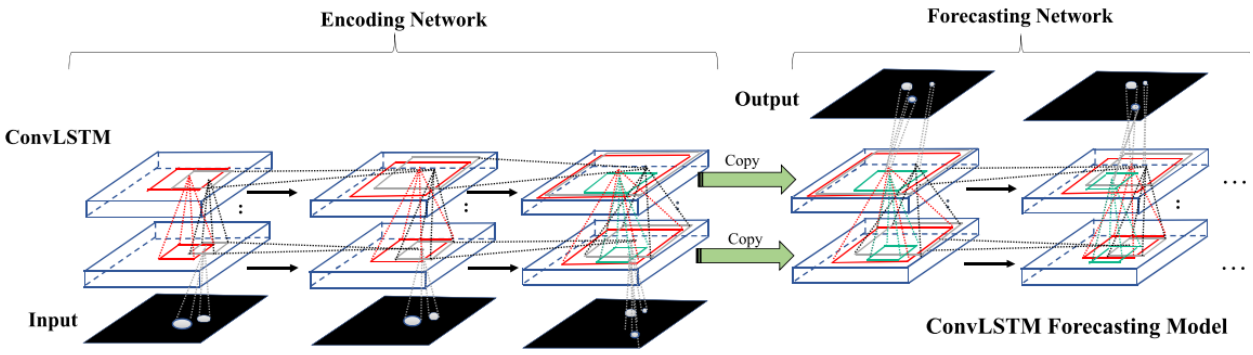


Figure III.1: The encoder (encoding network) maps the entire input sequence and its representation is passed to the decoder (forecasting network) as its initial representation [Kim et al., 2019].

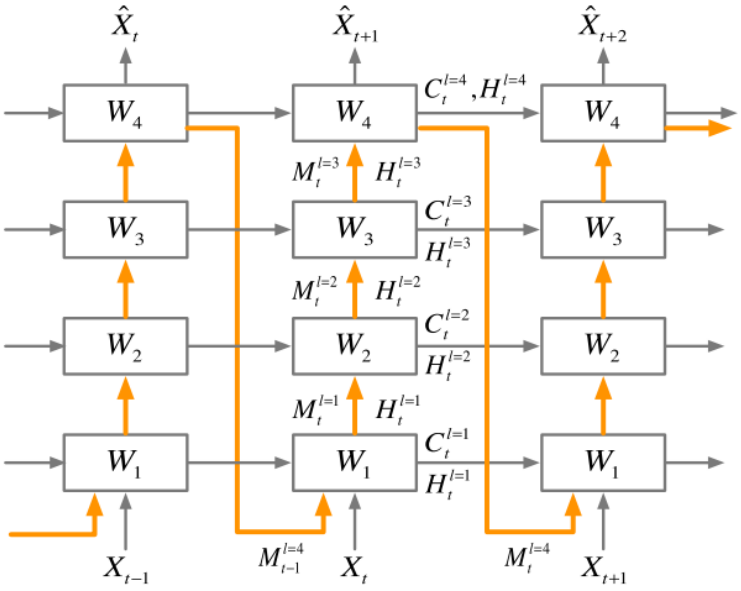


Figure III.2: PredRNN architecture with ST-LSTM. The gray arrows show the flow of conventional ConvLSTM architecture and the orange ones denote the spatiotemporal memory flow proposed in PredRNN architecture [Wang et al., 2017].

are designed to replace the temporal forget gate and capture the non-stationary variations between two consecutive hidden representations (previously termed as output state) $h_t - h_{t-1}$, through the differencing method [Box and Jenkins, 1970]. The other recurrent module takes as input the output of the non-stationary module and memory c_{t-1} to capture the approximately stationary variations in spatiotemporal sequences.

Although related to the use of deep learning for climate/weather data, our model adopts only CNN rather than a hybrid approach that combines CNN and LSTM.

III.2.2 CNN-based models

Some studies have applied spatiotemporal convolutions [Yuan et al., 2018; Tran et al., 2018] for video analysis and action recognition. In Tran et al. [2018], the authors compare several spatiotemporal architectures using only 3D CNN and show that factorizing the 3D convolutional kernel into

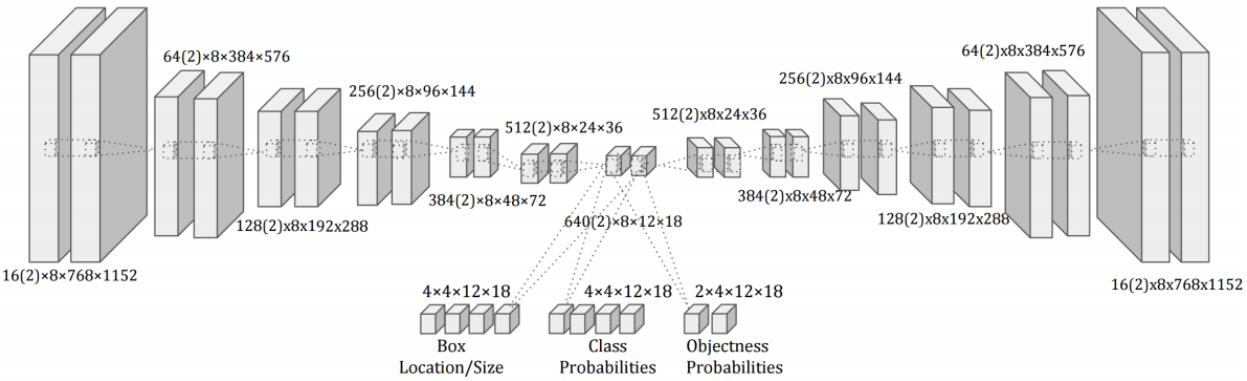


Figure III.3: A stack of 3D convolutional layers in the encoder and a stack of 3D transposed convolutional layers in the decoder are used for the recognition task [Racah et al., 2017].

separate and successive spatial and temporal convolutions produces accuracy gains. They use as input a sequence of video frames with specific height and width, and RGB channels. Their architecture focuses on layer factorization, i.e., factorizing each convolution into a block of a spatial convolution and a temporal convolution. A top layer performs a average pooling over the entire spatiotemporal volume to reduce the output dimensions, and a fully-connected ANN (FC-ANN) is responsible for the final classification prediction. Moreover, in comparison to the full 3D convolution, they indicate advantages: an increase in the complexity of the functions that can be represented, and a facility in the optimization of spatial or temporal components. Inspired by Tran et al. [2018], we also adopt a factorized 3D CNN, but with a different implementation. Figure IV.3 highlights this difference.

A limitation of both 3D CNN or factorized 3D CNN [Tran et al., 2018] is the lack of causal constraint, violating the temporal order. Singh and Cuzzolin [2019] and Cheng et al. [2019] factorize the 3D convolution as Tran et al. [2018]. Singh and Cuzzolin [2019] propose a recurrent convolution unit based on RNN to address causal constraint in temporal learning for action recognition tasks, and Cheng et al. [2019] satisfy the causal constraint by adopting causal convolution in separate and parallel spatial and temporal convolutions. We also adopt a factorized 3D CNN, but with a different implementation, where Figure IV.3 highlights our approach. In contrast to Singh and Cuzzolin [2019], we use an entirely CNN approach, and to Cheng et al. [2019], besides not using parallel convolutions when adopting a causal convolution, we introduce a new method to not violate the temporal order (details in Section IV.2.2).

Following the success of 2D CNN in capturing spatial correlation in images, Xu et al. [2019] propose a model to predict vehicle pollution emissions using 2D CNN to capture temporal and spatial correlation separately. However, unlike our work, they also do not satisfy the causal constraint when adopting 2D CNN in temporal learning. Racah et al. [2017] use a 3D CNN in an encoder-decoder architecture for extreme climate event detection (Figure III.3). They use as input 8 time

steps (sequence length) and 16 climate variables, where each variable corresponds to one channel of the tensor (likewise as in Souto et al. [2018]). Their model reconstruct the 8 time steps and predict bounding boxes[1] for 4 labelled time steps. Their architecture consists of a downsampling path in the encoder using a stack of convolutional layers, and an upsampling path in the decoder using a stack of transposed convolutional layers. Their model adopts the typical use of transposed convolutional layers to reconstruct the output to match the entire input dimension. Instead, we use these layers to generate an output with a larger dimension, different from the dimensions of the input.

III.3 Overall conclusion

To sum up, our proposed STConvS2S architecture aims to fill this gap and departs from the previous approaches, either in the manipulation of spatial and temporal dependencies or in the use of different deep learning layers to learn features from the data. Table III.1 highlights the main features of our proposed architecture as well as each work that uses the spatiotemporal approach.

In this chapter, we presented studies related to weather forecasting and spatiotemporal architectures. Firstly, we provided studies on time series approaches, in which they indicate that LSTM models performed better in forecasting tasks than the ARIMA models. Further on, we extended the related work to spatiotemporal architectures, focusing on deep learning approaches.

Table III.1: Overview of deep learning approaches for spatiotemporal data

Model	**Domain**	**Layer**	**Task**	**Output type**
ConvLSTM [Shi et al., 2015]	weather (rainfall)	ConvLSTM	forecasting	sequence
Deep-Hurricane-Tracker [Kim et al., 2019]	climate events (hurricane)	ConvLSTM	forecasting	sequence
Ensemble ConvLSTM [Souto et al., 2018]	weather (rainfall)	ConvLSTM	forecasting	single value
PredRNN [Wang et al., 2017]	weather (rainfall); video	ST-LSTM	forecasting; action prediction	sequence
MIM [Wang et al., 2019]	weather (rainfall); video	ST-LSTM + MIM	forecasting; action prediction	sequence
R(2+1)D [Tran et al., 2018]	video	3D CNN + FC-ANN	action recognition	class label
3D Encoder-Decoder [Racah et al., 2017]	climate events (hurricane)	3D CNN	object detection	bounding boxes + class label
STConvS2S (ours)	weather (temperature and rainfall)	3D CNN	forecasting	sequence

[1]Rectangular box determined by specific coordinates and used to identify an object

When dealing with sequence as output using spatiotemporal data, especially for forecasting tasks many-steps ahead, RNN-based architectures are widely explored, achieving state-of-the-art results for these tasks. On the other hand, CNN-based architectures are more applied to tasks such as action recognition and object detection, when dealing with spatiotemporal data. From the most recent research, it is evident that comparative studies between the RNN and CNN models for forecasting tasks using spatiotemporal data are still lacking.

Chapter IV Deep learning model for spatiotemporal data forecasting

This chapter first gives a formal definition of the sequence-to-sequence problem using spatiotemporal data (Section IV.1). Further on, it introduces the proposed architecture, providing a detailed description of each component (Section IV.2).

IV.1 Problem Statement

Spatiotemporal data forecasting can be modeled as a sequence-to-sequence problem. Thus, the observations of spatiotemporal data (e.g. meteorological variables) measured in a specific geographic region over a period of time serve as the input sequence to the forecasting task. Since the observations are related to a single variable and its observations at previous time steps are useful to predict the next time steps, this problem can be treated as an autoregression. More formally, we define a spatiotemporal dataset as $[\widetilde{X}^{(1)}, \widetilde{X}^{(2)}, \ldots, \widetilde{X}^{(m)}]$ with m samples of $\widetilde{X}^{(i)} \in \mathbb{R}^{T \times H \times W \times C}$, where $1 \leq i \leq m$. Each training example is a tensor $\widetilde{X}^{(i)} = [X_1^{(i)}, X_2^{(i)}, \ldots, X_T^{(i)}]$, that is a sequence of T observations containing historical measurements. Each observation $X_j^{(i)} \in \mathbb{R}^{H \times W \times C}$, for $j = 1, 2, \ldots, T$ (i.e. the length of input sequence), consists of a $H \times W$ grid map that determines the spatial location of the measurements, where H and W represent the latitude and longitude, respectively. In the observations, C represents how many meteorological variables (e.g. temperature, humidity) are used simultaneously in the model. This structure is analogous to 2D images, where C would indicate the amount of color components (RGB or grayscale).

Modeled as sequence-to-sequence problem in Equation IV.1, the goal of spatiotemporal data forecasting is to apply a function f that maps an input sequence of past observations, satisfying the causal constraint at each time step t, in order to predict a sequence $\hat{X} \in \mathbb{R}^{H \times W \times C}$, where the length T'' of output sequence may differ from the length T of input sequence.

$$\hat{X}_{t+1}, \hat{X}_{t+2}, \ldots, \hat{X}_{t+T''} = f(X_{t-T+1}, \ldots, X_{t-1}, X_t) \tag{IV.1}$$

IV.2 STConvS2S architecture

STConvS2S is an end-to-end deep neural network suited for learning spatiotemporal predictive patterns, which are common in weather forecasting domains. Our approach makes multi-step

(sequences) prediction without feeding the predicted output back into the input sequence. This is possible due to a specific component within the STConvS2S model designed to generate all future time steps T'' at once. Thus, our model can produce a sequence of any length without repeating the entire STConvS2S model up to the number of steps ahead T'' that it is desired to predict. Avoiding repetition of the model in a chain form provides computational cost savings, especially if T'' is large enough.

Although some methods for weather forecasting using a radar echo dataset apply a hybrid approach, combining CNN 2D (to learn spatial representations) and LSTM (to learn temporal representations) [Shi et al., 2015; Wang et al., 2017, 2019], our method uses only 3D convolutional layers to learn spatial and temporal contexts. Distinct from conventional convolution applied in some 3D CNN architectures [Tran et al., 2015; Tran et al., 2018; Racah et al., 2017], during temporal learning, STConvS2S takes care not to depend on future information, a crucial constraint on forecasting tasks. Another core feature of our designed network is the capability to allow flexible output sequence length, which means the possibility to predict many time-steps ahead, regardless of the fixed-length of the input sequence.

Figure IV.1 presents an overview of our proposed deep learning architecture, highlighting the spatiotemporal format of the input, feature maps and output. Figure IV.2, instead, is a simplified diagram of our architecture, mainly illustrating its components and the convolutional layers within them. In the following, we provide more details about the components which comprise our architecture.

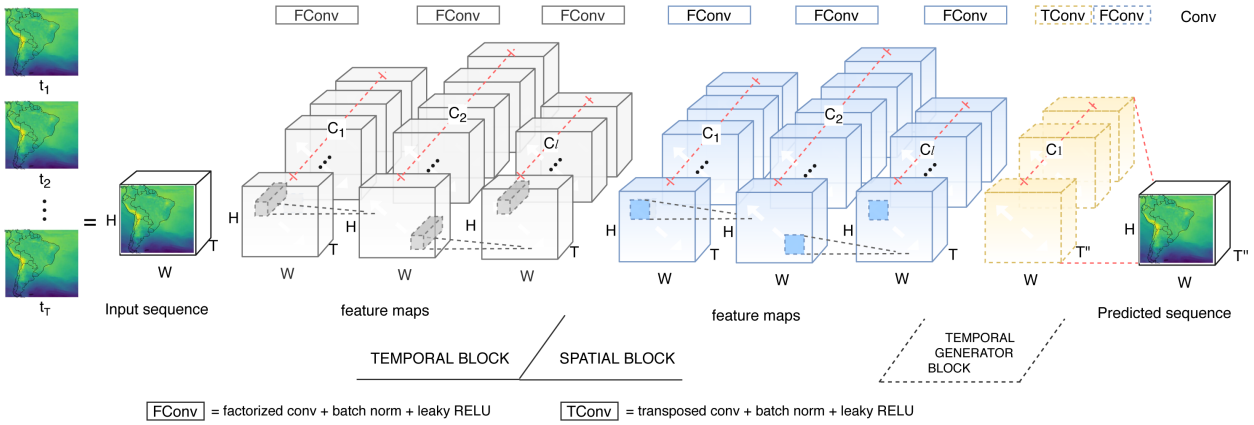


Figure IV.1: An illustration of STConvS2S architecture highlighting the spatiotemporal format. The architecture comprises three components: temporal block, spatial block, and temporal generator block. Each block is a set of layers. The temporal block learns a temporal representation of the input sequence, the spatial block extracts spatial features from the output of the previous block. On top of the spatial block, there is the temporal generator block designed to increase the sequence length T if the task requires a longer predictive horizon, where $T'' \geqslant T$. Finally, the output of this block is further fed into a final convolutional layer to complete the prediction

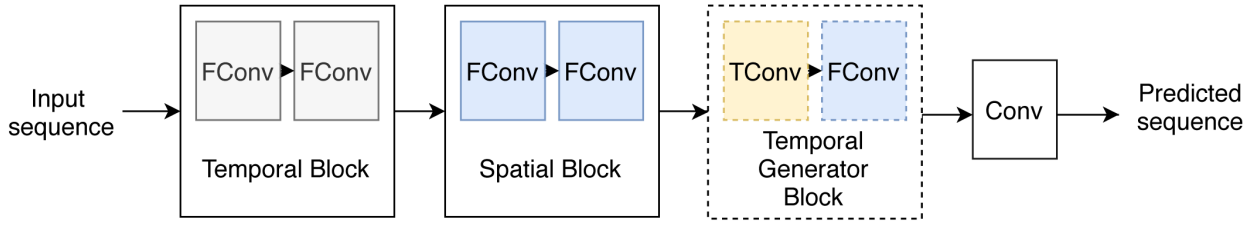


Figure IV.2: An illustration of STConvS2S architecture highlighting its components. Each block is shown with two layers, but it is possible to increase this number to suit a specific problem solution. Besides, each color indicates a distinction between layers. Gray represents the factorized convolutional layers ($FConv$) that use temporal kernels and blue, the $FConv$ that use spatial kernels. Since the temporal generator block is not always used (only when it requires extending the output sequence), this block is presented with dashed lines to distinguish it from other blocks.

IV.2.1 Factorized 3D convolutions

Instead of adopting a conventional $\bar{t} \times d \times d$ kernel for 3D convolutional layers, where d and $\bar{t}$ are the kernel size in space ($H \times W$) and time (T) dimensions, respectively, we use a factorized 3D kernel adapted from R(2+1)D network, proposed in Tran et al. [2018]. The factorized kernel $1 \times d \times d$ and $\bar{t} \times 1 \times 1$ split the convolution operation of one layer into two successive operations, named as a spatial convolution and a temporal convolution in their work. In our new architecture, we take a different approach: operations are not successive inside each convolutional layer. Instead, the factorized kernels are separated into two blocks, giving them specific learning skills. The temporal block applies the $\bar{t} \times 1 \times 1$ kernel in its layers to learn only temporal dependencies, while the next component, the spatial block, encapsulates spatial dependencies using $1 \times d \times d$ kernel. Figure IV.3 schematically illustrates the difference between these three approaches and Figure IV.4 shows a representation of the factorized 3D convolutions in the temporal and spatial blocks of STConvS2S architecture.

Compared to the full 3D kernel applied in standard convolutions, the kernel decomposition used in STConvS2S offers the advantage of increasing the number of nonlinearities in the network (additional activation functions between factorized convolutions), which leads to an increase in the complexity of representable patterns [Tran et al., 2018]. An advantage of our proposed approach over the (2+1)D block is flexibility since temporal and spatial blocks can have a distinct number of layers, facilitating their optimization.

IV.2.2 Temporal Block

In STConvS2S, the temporal block is a stack of 3D convolutional layers which adopt $\bar{t} \times 1 \times 1$ kernel during convolutions. Each layer receives a 4D tensor with dimensions $T \times H \times W \times C_{l-1}$ as input, where C_{l-1} is the number of filters used in the previous layer $(l-1)$, T is the sequence length (time dimension), H and W represent the size of the spatial coverage for latitude and longitude,

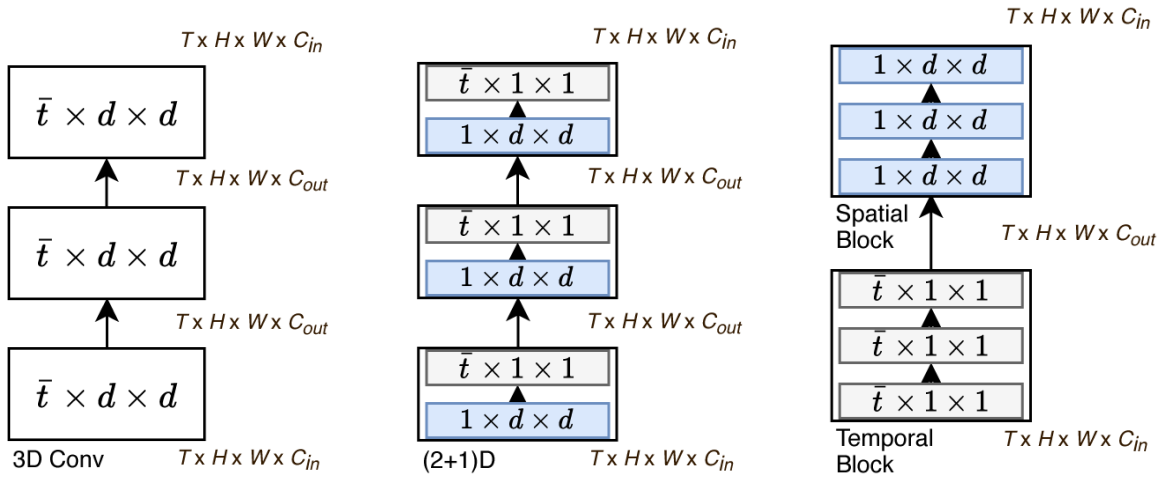


Figure IV.3: Comparison of convolution operations applied in three convolutional layers. The spatial kernel (k_s) is defined as $1 \times d \times d$ and the temporal kernel ($k_{\bar{t}}$) as $\bar{t} \times 1 \times 1$, where d and $\bar{t}$ are the kernel size in spatial ($H \times W$) and time (T) dimensions, respectively. (a) Representation of the standard 3D convolution operation using the $\bar{t} \times d \times d$ kernel. (b) Factorized 3D kernels proposed in Tran et al. [2018] as successive spatial and temporal convolution operations in a unique block called (2+1)D. (c) Our proposal for the factorized 3D kernels usage is in separate blocks. First, the temporal block stacks three convolutional layers, each performing convolutions using only the temporal kernel. Likewise, the spatial block applies the spatial kernel to its layers.

respectively. Within the block, the filters C are increased twice in the feature maps as the number of layers increases, but the final layer reduces them again to the number of filters initially defined.

In detail, this block uses batch normalization and LeakyReLU after each convolutional layer. This block discovers patterns over the time dimension T exclusively. Besides, since we are using 3D convolutional layers to analyze historical series of events, we must prevent data leakage from happening. That is, the model should not violate the temporal order and should ensure that, at step t, the learning process uses no future information from step $t+1$ onward. To satisfies this constraint, we propose two variants of the temporal block.

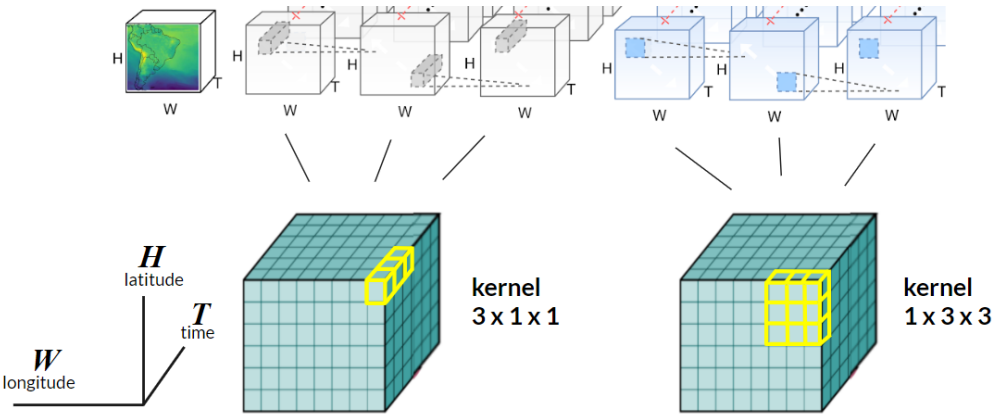


Figure IV.4: Example of representation of factorized 3D convolutions in temporal and spatial block when $\bar{t} = d = 3$. The kernel format allows each block to learn temporal and spatial dependencies separately, providing blocks with specific learning skills.

Temporal Causal Block

We name our architecture as *STConvS2S-C* when it adopts this block to learn the temporal patterns. We apply causal convolutions within the block to incorporate the ability to respect the temporal order during learning in convolutional layers. Causal convolution was originally presented in WaveNet [van den Oord et al., 2016] for 1D CNN and applied with factorized 3D convolutions in Cheng et al. [2019]. This technique can be implemented by padding the input by $k-1$ elements, where k is the kernel size. Since we use factorized convolutional layers with temporal kernel $(\bar{t} \times 1 \times 1)$, the convolution operation on a 3D input is similar to 1D, as shown in Figure IV.4 through the feature map on the left. Figure IV.5 shows the operation in details.

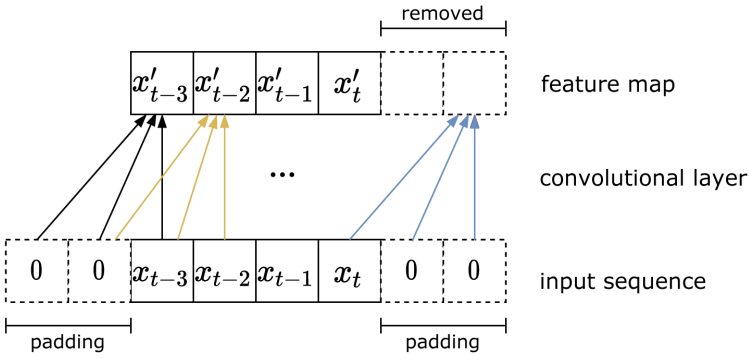


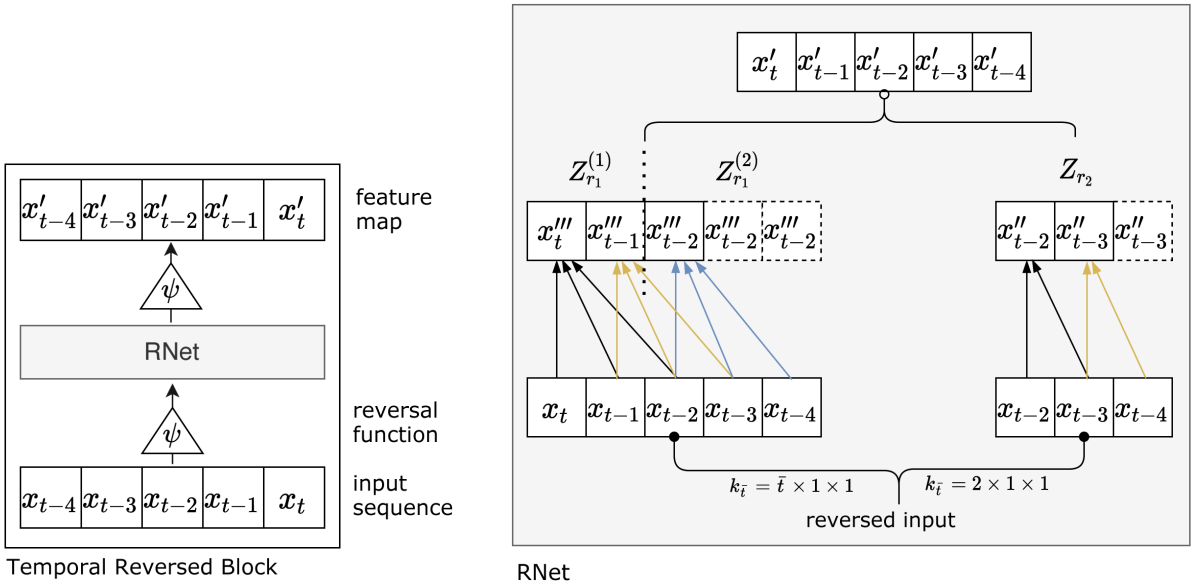
Figure IV.5: Causal convolution operation in a factorized convolutional layer showing only the time dimension, with $\bar{t} = 3$ (temporal kernel size). Input is padded by $\bar{t} - 1$ elements to avoid learning future information. To ensure that the output feature map has the same length as the input, the last $\bar{t} - 1$ elements are removed since they are related to the zeros added to the right of the input.

Temporal Reversed Block

When dealing with historical data, respecting the temporal order (causal constraint) is an essential behavior of deep learning models. This because, in real applications, future information is not available in forecasting. The common approach in the literature for adapting convolutional layers to satisfy this constraint is through causal convolutions. Here, we introduce a better alternative [1] to avoid violating the temporal order. The architecture is named as *STConvS2S-R* when composed with this block.

Unlike STConS2S-C, the STConvS2S-R architecture does not apply padding in the input, a strategy used by the former to respect the causal constraint. Instead, in the STConvS2S-R, we use a sequence of operations to fill the output feature map. Further on, we will provide details about these operations, but first, we schematically illustrate the structure of the temporal reversed block in Figure IV.6.

[1] Detailed analysis of the experimental results are demonstrated in the Ablation study (Section V.6), more specifically in the topic "Comparison of strategies to satisfy the causal constraint".



(a) (b)

Figure IV.6: Temporal Reversed Block. (a) Structure of the temporal reversed block. Here, the block is shown with one RNet layer for simplicity, but it is possible to stack several layers. (b) RNet structure in detail with two factorized convolutional layers: on the left, the layer with $\bar{t} \times 1 \times 1$ kernel ($\bar{t} = 3$); on the right, the layer with fixed kernel ($2 \times 1 \times 1$). The operations for joining the outputs of these layers are shown in Equation IV.5.

Within the temporal reversed block, we start by applying a function ψ in the time dimension of the input to reverse the sequence order, where this function is a linear transformation $\psi : \mathbb{R}^{T \times H \times W \times C} \rightarrow \mathbb{R}^{T \times H \times W \times C}$. Next, we stack several *RNet*, a carefully designed network composed of two factorized convolutional layers, applying only a temporal kernel ($k_{\bar{t}}$).

RNet generates an output that matches the input dimensions without using the padding technique in the input [2]. In this way, the first factorized convolutional layer of RNet (left in Figure IV.6) applies a $\bar{t} \times 1 \times 1$ kernel, generating an output with size of $T - \bar{t} + 1$ in the time dimension, where $\bar{t}$ is the temporal kernel value (a hyperparameter previously defined in the execution) and T is the size of time dimension (length of sequence) of the input. The second layer is similar to the first one, but always adopts the $2 \times 1 \times 1$ kernel instead, which is the smallest temporal kernel to capture correlation among the input sequence values.

One way to understanding the RNet operations is to think about padding the output by repeating the last value until it reaches the same size as the input in the time dimension (i.e., $\bar{t} - 1$ elements). This strategy helps to keep the learned values in the right position to ensure the causal constraint. Another operation is the usage of the output of the second layer ($k_{\bar{t}} = 2 \times 1 \times 1$) to generate variability in the padded values of the first layer for each time step.

Take x''_{t-3} in Z_{r_2} (Figure IV.6) as an example. First, the learning process only take account

[2]The input dimensions after each convolution operation are reduced, thus the padding technique is usually applied to handle this issue (see Section II.2.2)

x_{t-3} and x_{t-4}, thus respecting the causal constraint of not using future information. Then, we use x''_{t-3} to create variability (through the subtraction operation) for the padded value x'''_{t-2} in $Z^{(2)}_{r_1}$ in the same position (time step). Formally, STConvS2S-R computes the output feature map R of a temporal reversed block using

$$R'_{1:l_r} = \begin{cases} rnet(\psi(I_u)), & \text{if } u = 1 \\ rnet(I_u), & \text{if } 2 \leqslant u \leqslant l_r \end{cases} \tag{IV.2}$$

$$R = \psi(R') \tag{IV.3}$$

where l_r is the number of RNet layers ($rnet$) within the temporal reversed block. For the first layer, I_1 is the input sequence $\widetilde{X}$ previously defined in Section IV.1, and for the subsequent layers, $I_{2:l_r}$ is the feature map calculated in the previous RNet layer R'_{l_r-1}.

The factorized convolutional layers in RNet are defined as in Equation IV.4, where $\bar{W}_{r_{1:2}}$ and $b_{r_{1:2}}$ are the learnable weight tensor and bias term, respectively, $*$ denotes a convolution operator and $g(\cdot)$ is a non-linear activation function. $I_{r_{1:2}}$ determines the inputs for the two convolutional layers in RNet. Specifically for the second layer, $I_{r_2} = [I_{r_1(e)}]$, $\bar{t} \leqslant e \leqslant T$ in which e represents the *e-th* elements of I_{r_1} in the time dimension. Thus, We use a part of the input for the second layer, since we only need to create variability for padded values in the output feature map of the first convolutional layer.

$$Z_{r_{1:2}} = g(\bar{W}_{r_{1:2}} * I_{r_{1:2}} + b_{r_{1:2}}) \tag{IV.4}$$

The final output feature map of the RNet R' is calculated as in Equation IV.5, where $\oplus$ denotes a concatenation operator in the time dimension, $Z^{(1)}_{r_1} = [Z_{r_1(i)}]$, $1 \leqslant i \leqslant T - \bar{t}$ and $Z^{(2)}_{r_1} = [Z_{r_1(j)}]$, $\bar{t} \leqslant j \leqslant T$ in which i and j represent *i-th* and *j-th* elements of Z_{r_1} in the time dimension.

$$R' = Z^{(1)}_{r_1} \oplus (Z^{(2)}_{r_1} - Z_{r_2}) \tag{IV.5}$$

IV.2.3 Spatial Block

The spatial block is built on top of the temporal block and has a similar structure with batch normalization and LeakyReLU as non-linearity. In contrast, each 3D convolutional layer of this block extracts only spatial representations since kernel decomposition allows us to analyze the spatial and temporal contexts separately. In the STConvS2S, each feature map generated has a fixed-length in $H \times W$ dimensions and, to ensure this, the input in the spatial block is padded following $p = \frac{k_s-1}{2}$, where k_s is the size of spatial kernel. This design choice differentiates our

model from 3D encoder-decoder architecture [Racah et al., 2017], which needs to stack upsample layers after all convolutional layers, due to the downsampling done in the latter.

IV.2.4 Temporal Generator Block

In addition to ensuring that our model satisfies the causal constraint, another contribution of our work is generating output sequences with longer lengths than the length of the input sequence. When CNNs are used for sequence-to-sequence learning, such as multi-step forecasting, the length of the output sequence must be the same size or shorter than the input sequence [Gehring et al., 2017; Bai et al., 2018]. To tackle this limitation, we designed a component placed on top of the spatial block used when the task requires a more extended sequence (e.g., from the previous 5 grids, predict the next 15 grids). First, we compute the intermediate feature map G':

$$G'_{1:l_{g_t}} = tconv(I_{1:l_{g_t}}) \tag{IV.6}$$

where $l_{g_t} = \left\lceil \frac{T''-T}{2T} \right\rceil$ is the number of transposed convolutional layers ($tconv$) necessary to guarantee that G' has the size of time dimension $T_{G'} \geqslant T'' - T$. The kernel size, stride and padding of $tconv$ are fixed and extend the feature map by a factor of 2 in time dimension only. For the first layer, I_1 is the output of the spatial block S and for the other layers, $I_{2:l_{g_t}}$ is the feature map calculated in the previous layer $G'_{l_{g_t}-1}$. Follow, given G' and S, we can compute G'':

$$G'' = \rho(S \oplus G') \tag{IV.7}$$

In the equation above, $\oplus$ denotes a concatenation operator in the time dimension and $\rho(\cdot)$ is a function to ensure that the feature map G'' matches exactly the length T'' of the desired output sequence. Figure IV.7 illustrates the feature maps used in Equation IV.7. Notice that we initially preserve the output of the spatial block S, which contains the learning patterns from the early layers of the architecture, and we use the transposed convolutional layer to extend the time dimension, generating G'.

Finally, the output feature map G of this block can be defined as

$$G_{1:l_{g_c}} = g(\bar{W}_{1:l_{g_c}} * I_{1:l_{g_c}} + b_{1:l_{g_c}}) \tag{IV.8}$$

where $l_{g_c} = \left\lfloor \frac{T''}{T} \right\rfloor$ is the number of convolutional layers that use factorized kernels as in the spatial block. $\bar{W}_{1:l_{g_c}}$ and $b_{1:l_{g_c}}$ are the learnable weight tensor and bias term, respectively, in l_{g_c} layers, $*$ denotes a convolution operator and $g(\cdot)$ is a non-linear activation function. For the first convolutional layer, I_1 is G''. Unlike the temporal and spatial block, where the number of layers is a defined hyperparameter to execute the model, in the temporal generator block l_{g_t} and l_{g_c} are

calculated based on the length T'' of the desired output sequence and the length T of the input sequence (the size of time dimension).

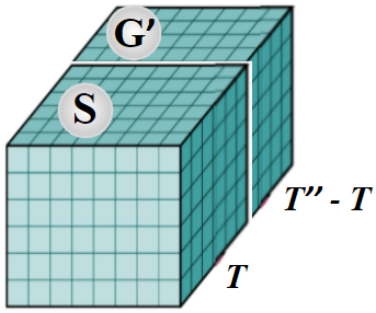


Figure IV.7: Feature maps in the Temporal Generator Block. Feature map S is the output of the spatial block, and G' is the main point of this block - to extend the sequence so that the sequence predicted at the end of the STConvS2S architecture meets the length T''.

Chapter V Experiments

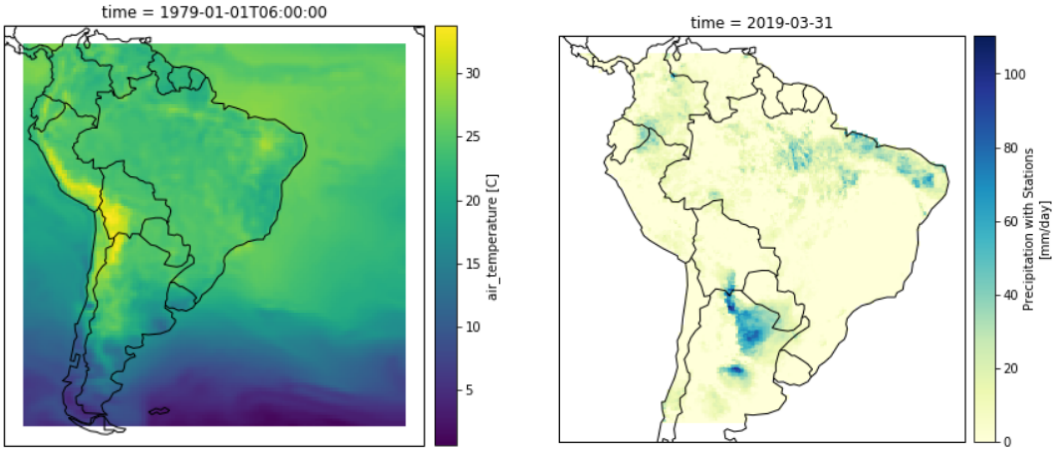
We perform experiments on two publicly available meteorological datasets containing air temperature and precipitation values to validate our proposed architecture. We start by detailing the environment built to perform these experiments, providing information about the hardware and software settings (Section V.1). We also explain the datasets (Section V.2) and the evaluation metrics adopted to compare the models (Section V.3). Further, we describe the main results of the experiments for each dataset (Section V.4 and V.5) and present the results of ablation studies (Section V.6).

V.1 Hardware and Software settings

Deep learning experiments were conducted on a server with a single Nvidia GeForce GTX1080Ti GPU with 11GB memory. We executed the ARIMA methods on 8 Intel i7 CPUs with 4 cores and 66GB RAM. We use PyTorch[1], an open source machine learning framework, in version 1.0 with Python 3.6 and CUDA 10.0 to implement and run the deep learning experiments. PyTorch is developed by Facebook's AI Research lab (FAIR) and enables fast experimentation due to its ecosystem of tools and libraries. Our source code is publicly available at `https://github.com/MLRG-CEFET-RJ/stconvs2s`.

For baselines, we use statsmodels[2] in version 0.9, which is a Python package that provides classes and functions for the estimation of many statistical models. Another relevant package is xarray[3], used in version 0.11 to manipulate and analyze the spatiotemporal dataset in netCDF4[4] format. Our datasets can be downloaded at `http://doi.org/10.5281/zenodo.3558773`, an open source online data repository.

[1] `https://pytorch.org`
[2] `https://www.statsmodels.org`
[3] `http://xarray.pydata.org`
[4] `https://www.unidata.ucar.edu/software/netcdf/`



(a) CFSR-temperature dataset (b) CHIRPS-rainfall dataset

Figure V.1: Spatial coverage of the datasets used in all experiments. (a) It shows the selected grid on January 1, 1979 with air temperature values.(b) It shows the selected grid of the sequence on March 31, 2019 with rainfall values.

V.2 Datasets

CFSR[5] is a reanalysis[6] product that contains high-resolution global land and ocean data [Saha et al., 2014]. The data contain a spatial coordinate (latitude and longitude), a spatial resolution of 0.5 degrees (i.e., $0.5° \times 0.5°$ area for each grid cell) and a frequency of 6 hours for some meteorological variables, such as air temperature and wind speed.

In the experiments, we use a subset of CFSR with the air temperature observations from January 1979 to December 2015, covering the space in 8°N-54°S and 80°W-25°W as shown in Figure V.1 (a). As data preprocessing, we scale down the grid to 32×32 in the H and W dimensions to fit the data in GPU memory. The other dataset, CHIRPS[7], incorporates satellite imagery and in-situ station data to create gridded rainfall times series with daily frequency and spatial resolution of 0.05 degrees [Funk et al., 2015]. We use a subset with observations from January 1981 to March 2019 and apply interpolation to reduce the grid size to 50×50. Figure V.1 (b) illustrates the coverage space 10°N-39°S and 84°W-35°W adopted in our experiments.

CFSR dataset has no missing values and, as CHIRPS dataset has only continent data, to make this dataset capable of being trained on grid-like models, it was necessary to imput the missing values from the ocean. We adopt as an imputation strategy the filling with the most frequent value (zero, in this case, for the rainfall dataset). As a final strategy, we apply a mask (continental data filled with 1 and ocean data filled with 0) to the predicted data generated by the models in order not to consider the ocean forecast when calculating the loss.

[5] `https://climatedataguide.ucar.edu/climate-data/climate-forecast-system-reanalysis-cfsr`

[6] Scientific method used to produce best estimates (analyses) of how the weather is changing over time [Fujiwara et al., 2017].

[7] `https://chc.ucsb.edu/data/chirps`

Similar to Shi et al. [2015], we define the input sequence length as 5, which indicates the use of the previous five grids to predict the next T'' grids. Thus, the input data shapes to the deep learning architectures are $5 \times 32 \times 32 \times 1$ for CFSR dataset and $5 \times 50 \times 50 \times 1$ for CHIRPS dataset. The value 1 in both dataset shapes indicates the one-channel (in this aspect similar to a grayscale image), 5 is the size of the sequence considered in the forecasting task, and 32 and 50 represent the numbers of latitudes and longitudes used to build the spatial grid in each dataset.

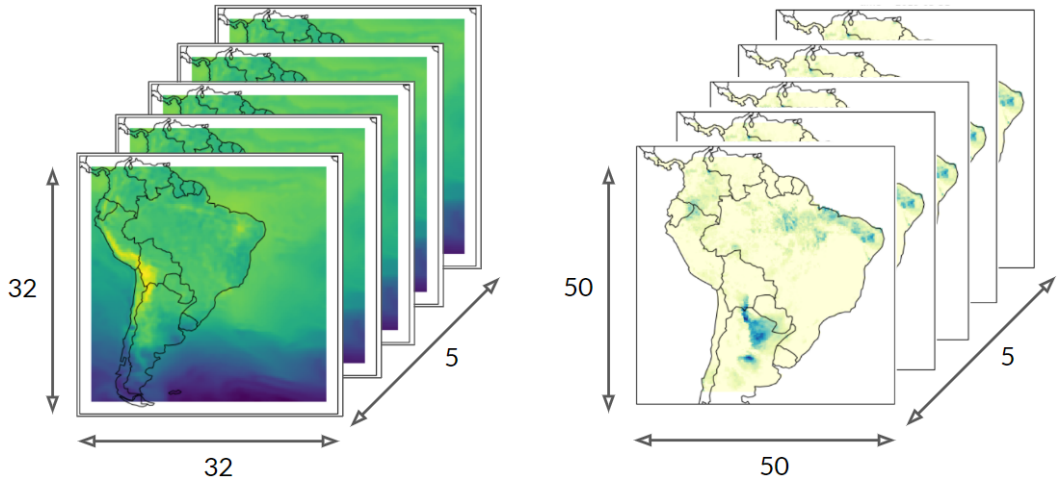
We create 54,041 and 13,960 grid sequences from the temperature dataset and rainfall datasets, respectively. Since we are using a CNN-based architecture to solve a sequence-to-sequence problem, more specifically multi-step spatiotemporal forecasting, we need to manipulate the data in order to feed it into a convolutional layer. Taking CHIRPS dataset as example, we transform this sequence of daily rainfall observations from 1981 to 2019, into several small sequences (samples), each containing an input-target association, as the type of learning is supervised. Thus, for each sample, the input is the past T observations, and the target is the future T'' observations. Mapping CNN terminology to the forecasting domain, a convolutional layer with temporal kernel (see Section IV.2.1) with size of K means that the convolution operation will analyze the input sequence K time steps at a time.

Finally, we divide both datasets into non-overlapping training, validation, and test set following 60%, 20%, and 20% ratio, in this order. In detail, considering a the index to divide the entire dataset, the index for the training set can be defined as $a \in [0, b[$, for the validation set as $a \in [b, d[$, and for test set as $a \in [d, e[$, where $b = \lfloor 0.6 \times e \rfloor$ and $d = \lfloor 0.8 \times e \rfloor$ are the split limits (considering the ratios in our case), and e is the length of the entire dataset before the division. As the name suggests, the training set is used to train the model, which means learning the patterns in data based on the space and time contexts. The model must evaluate its predictive accuracy in each iteration during the learning phase and, for this purpose, the validation set is applied. The validation set helps us to see whether the model generalizes its learning to unseen data as well or whether it is overfitting to training data. On the other hand, the test set is not considered during the training phase. It is used to evaluate the model, which means, to compute only the error without the backpropagation and optimization process.

Table V.1: Properties of CFSR and CHIRPS datasets

	CFSR dataset	CHIRPS dataset
Variable	air temperature	rainfall
Period	from Jan. 1979 to Dec. 2015	from Jan. 1981 to Mar. 2019
Frequency	of 6 h	daily
Size of temporal sequence	$T = 5$	$T = 5$
Size of spatial coverage	$H = W = 32$	$H = W = 50$
Samples	54,041 sequences	13,960 sequences

The adoption of temperature and rainfall datasets in our experimental evaluation relies on the fact that they are the two main meteorological variables. Research about their spatiotemporal representation is relevant to short-term forecasting and improves the understanding of long-term climate variability [Rahman and Lateh, 2017]. However, the proposed architecture is suitable for other meteorological variables or other domains, as long as the training data can be structured as defined in Section IV.1. Table V.1 summarizes the properties of the CFSR and CHIRPS datasets presented throughout this section and Figure V.2 illustrates the example of a small sequence (sample) for each dataset, highlighting its dimensions.



(a) CFSR sequence (b) CHIRPS sequence

Figure V.2: Example of a sequence for each dataset. In detail, the size of temporal sequence (5) and the size of spatial coverage.

V.3 Evaluation metrics

To evaluate the proposed architecture, we compare our results against ARIMA models, traditional statistical approaches for time series forecasting, and state-of-the-art models for spatiotemporal forecasting. To accomplish this, we use the two evaluation metrics presented in Equation V.1 and V.2.

RMSE, denoted as E_r, is based on MSE metric, which is the average of squared differences between real observation and prediction. The MSE square root gives the results in the original unit of the output, and is expressed at a specific spatiotemporal volume as:

$$E_r(T,H,W) = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\sum_{t\in T}\sum_{h\in H}\sum_{w\in W}[x(t,h,w)-\hat{x}(t,h,w)]^2} \tag{V.1}$$

where N is the number of test samples, $x(t,h,w)$ and $\hat{x}(t,h,w)$ are the real and predicted values at the location h and w at time t, respectively.

MAE, denoted as E_m, is the average of differences between real observation and prediction, which measures the magnitude of the errors in prediction. MAE also provides the result in the original unit of the output, and is expressed at a specific spatiotemporal volume as:

$$E_m(T,H,W) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t\in T}\sum_{h\in H}\sum_{w\in W}|x(t,h,w) - \hat{x}(t,h,w)| \tag{V.2}$$

where N, t, h, w are defined as shown in Equation V.1.

The results of these metrics can be very different from each other, as the RMSE (square difference) penalizes the wildly wrong results more than the MAE (absolute difference).

V.4 CFSR Dataset: results and analysis

We first conduct experiments with distinct numbers of layers, filters, and kernel sizes to investigate the best hyperparameters to fit the deep learning models. As a starting point, we set the version 1 based on the settings described in Shi et al. [2015] with two layers, each containing 64 filters and a kernel size of 3[8]. To make fair comparisons using the chosen datasets, we explored variations of the hyperparameters for our architecture (STConvS2S-C and STConvS2S-R) and the following state-of-the-art methods: ConvLSTM [Shi et al., 2015], PredRNN [Wang et al., 2017], and MIM [Wang et al., 2019]. Thus, for versions 2-4, we defined the number of layers (L), kernel size (K) and the number of filters (F) in a way that would help us understand the behavior of the models during the learning process by increasing L (versions 1 and 3), K (versions 2 and 4) or F (versions 2 and 3). In the training phase, we perform for all models mini-batch learning with 50 epochs, and RMSprop optimizer with a learning rate of 10^{-3}.

We applied dropout after convolutional layers during the training of PredRNN and MIM models [9] to reduce the model complexity and avoid overfitting. Without dropout, these models do not generalize well for this dataset and make less accurate predictions in the validation set. We adopt 0.5 as the dropout rate for both models after evaluating the best rate employing the grid search technique, which performed several experiments changing the rate by {0.3, 0.5, 0.8}. Figure V.3 (a) and (b) illustrate the differences in the learning curve, where the former shows a high error on the validation set early in the training stage for both models, and the latter indicates the learning curve with dropout applied.

As a sequence-to-sequence task, we use the previous five grids, as we established before in Section V.2, to predict the next five grids (denoted as $5 \rightarrow 5$). Table V.2 provides the models considered in our investigation with four different settings, the values of the RMSE metric on the

[8] 3×3 kernel for ConvLSTM, PredRNN, and MIM. $3 \times 1 \times 1$ temporal kernel and $1 \times 3 \times 3$ spatial kernel for STConvS2S.

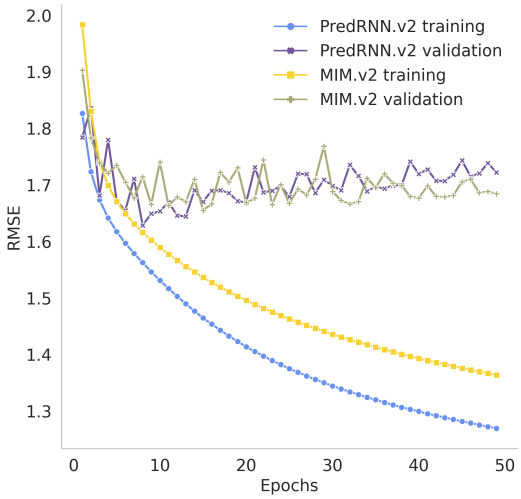
[9] STConvS2S and ConvLSTM models do not overfit during training on any version.

test set, the training time, and the memory usage by GPU. The results present the superiority of version 4, which has the highest values of L and K, reaching the lowest RMSE for all models, except PredRNN, where version 2 is superior. Another aspect to note is that when increasing the number of filters (versions 2 and 3) the impact is more significant in the training time than when increasing the number of layers (versions 1 and 3) for state-of-the-art models, indicating that version 2 is faster for these RNN-based architectures. This impact is not seen in our models (CNN-based) compared to version 3, as in versions 1 and 2 they spend almost the same time during training, showing that STConvS2S models have more stability when increasing the hyperparameters.

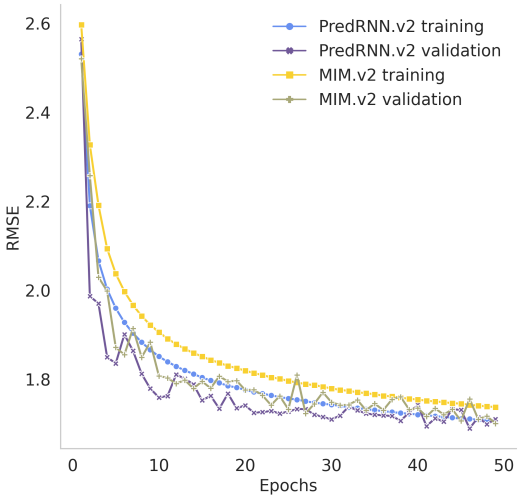
Table V.2: Evaluation of different settings on the CFSR dataset for STConvS2S and state-of-the-art methods, where the best version has the lowest RMSE value.

			$5 \rightarrow 5$		
Model	Version	Setting	RMSE	Training time	Memory usage (MB)
ConvLSTM [Shi et al., 2015]	1	L=2, K=3, F=64	2.1306	01:49:16	1119
	2	L=3, K=3, F=32	2.0090	**01:12:16**	**920**
	3	L=3, K=3, F=64	1.9607	02:53:00	1358
	4	L=3, K=5, F=32	**1.8770**	01:58:52	922
PredRNN [Wang et al., 2017]	1	L=2, K=3, F=64	1.7497	06:59:45	3696
	2	L=3, K=3, F=32	**1.6928**	**04:55:19**	**2880**
	3	L=3, K=3, F=64	1.7004	10:44:25	5242
	4	L=3, K=5, F=32	1.7028	06:41:14	2892
MIM [Wang et al., 2019]	1	L=2, K=3, F=64	1.7623	09:37:07	4826
	2	L=3, K=3, F=32	1.7199	**07:31:59**	**4124**
	3	L=3, K=3, F=64	1.7163	16:27:42	7789
	4	L=3, K=5, F=32	**1.6621**	09:49:40	4145
STConvS2S-C (ours)	1	L=2, K=3, F=64	1.6355	**01:16:40**	**991**
	2	L=3, K=3, F=32	1.5681	01:22:42	1021
	3	L=3, K=3, F=64	1.5459	03:14:25	1554
	4	L=3, K=5, F=32	**1.3791**	02:25:26	1040
STConvS2S-R (ours)	1	L=2, K=3, F=64	1.6001	**01:22:57**	**975**
	2	L=3, K=3, F=32	1.5384	01:29:18	1000
	3	L=3, K=3, F=64	1.5149	03:25:44	1493
	4	L=3, K=5, F=32	**1.2823**	02:13:37	956

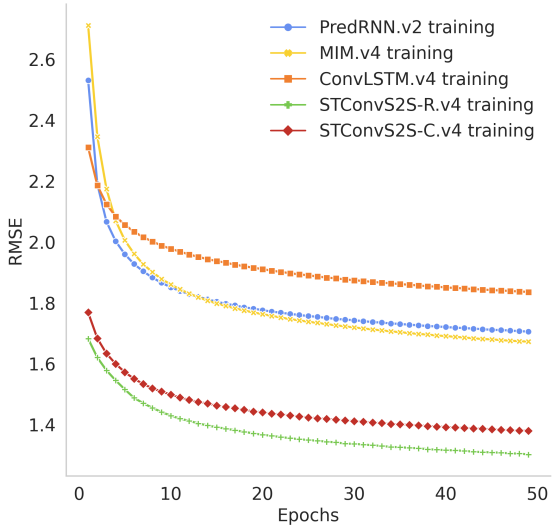
To improve the comprehension of the analysis, Figure V.4 highlights the differences between the performances of RMSE metric and training time for the models. As shown, STConvS2S-R and STConvS2S-C models perform favorably against the state-of-the-art models for the CFSR dataset in all versions, demonstrating that our architectures can simultaneously capture spatial and temporal correlations. Comparing the best version of each model, our models significantly outperform the state-of-the-art architectures for spatiotemporal forecasting. In detail, STConvS2S-R (version 4) takes only 1/4 of memory space, is almost 5x faster in training, and has achieved a



(a) Learning curve - overfitting



(b) Learning curve - dropout 0.5



(c) Training curve for all models

Figure V.3: Learning curves after running 50 epochs on temperature dataset (CFSR). (a) To exemplify, we select version 2 to illustrate the overfitting observed when analyzing the training and validation curve of PredRNN and MIM models. (b) The same version and models using dropout to improve its generalization. (c) Comparison of training curve for the best version for each model. Our models (STConvS2S-R and STConvS2S-C) achieved lower RMSE and thus, better ability to learn spatiotemporal representations.

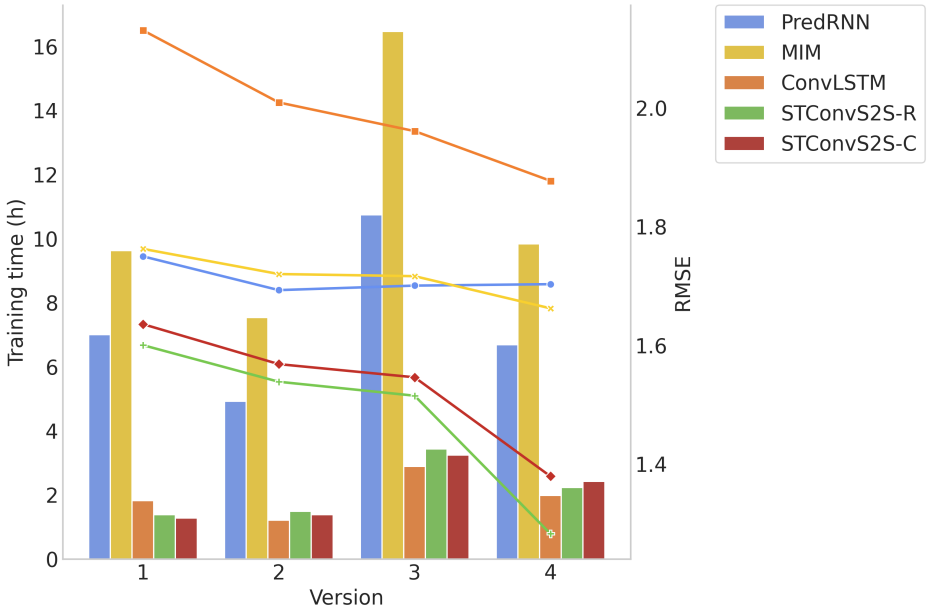


Figure V.4: Comparison between training time in hours (bar plot) and RMSE (line plot) for each model version on temperature dataset (CFSR).

23% improvement in RMSE over MIM (version 4), the RNN-based model with better performance. These results reinforce that our models have fewer parameters to optimize than MIM and PredRNN. Furthermore, our model can be completely parallelized, speeding up the learning process, as the output of the convolutional layers does not depend on the calculations of the previous step, as occurs in recurrent architectures. Figure V.3 (c) illustrates that STConvS2S-R has a lower training error in 50 epochs compared to other models, including STConvS2S-C, proving to be a better alternative to make CNN-based models respect the temporal order.

To further evaluate our models, we chose the most efficient version for each model to perform new experiments. For STConvS2S-R, STConvS2S-C, ConvLSTM and MIM models, version 4 was chosen with 3 layers, 32 filters, and kernel size of 5, and for PredRNN, version 2 with the same number of layers and filters, but with kernel size of 3. We also included a comparison with two variations of the ARIMA methods to serve as baseline for deep learning models, since they are a traditional approach to time series forecasting. The experiment for the baseline takes into account the same temporal pattern and spatial coverage. Thus, predictions were performed throughout all the 1,024 time series, considering in each analysis the previous 5 values in the sequence.

In this phase, we have not defined a specific number of epochs for each deep learning model's execution. Therefore to avoid overfitting during the training of models, we apply the early stopping technique with patience hyperparameter set to 16 on the validation dataset. As the models run with different numbers of epochs, we include the training time/epoch to be able to compare the time efficiency of the models. We train and evaluate each deep learning model 3 times and compute the mean and the standard deviation of RMSE and MAE metrics on the test set. This time, we

evaluate the models in two horizons: 5-steps ($5 \rightarrow 5$) and 15-steps ($5 \rightarrow 15$). These experiments are relevant to test the capability of our model to predict a long sequence.

In Table V.3, the results show that all models outperform the random walk model (ARIMA $(0, 1, 0)$), and this provides evidence that the temperature forecast does not follow a random behavior over time. In detail, STConvS2S-R and STConvS2S-C perform much better than the ARIMA$(5, 0, 1)$ baseline on RMSE and MAE metrics, indicating the importance of spatial dependence on geoscience data since ARIMA models only analyze temporal relationships. They also outperform the state-of-the-art models in these evaluation metrics in both horizons, demonstrating that our models can be efficiently adopted to predict future observations. However, beyond that, the designed temporal generator block in STConvS2S architecture can convincingly generate a more extended sequence regardless of the fixed-input sequence length. In a closer look at the best CNN-based architecture and RNN-based architecture in the task $5 \rightarrow 15$, STConvS2S-R takes less memory space and is faster than PredRNN. To provide an overview, Figure V.5 illustrates the cumulative error based on both horizons.

Table V.3: Performance results for temperature forecasting using the previous five observations (grids) to predict the next five observations ($5 \rightarrow 5$), and the next 15 observations ($5 \rightarrow 15$).

	$5 \rightarrow 5$				
Model	RMSE	MAE	Memory usage (MB)	Mean training time	Training time/epoch
ARIMA(0,1,0)	2.3413	1.9777	—	—	—
ARIMA(5,0,1)	2.1880	1.9005	—	—	—
ConvLSTM [Shi et al., 2015]	1.8555 ± 0.0033	1.2843 ± 0.0028	**922**	**02:38:27**	**00:02:21**
PredRNN [Wang et al., 2017]	1.6962 ± 0.0038	1.1885 ± 0.0020	2880	06:59:34	00:05:52
MIM [Wang et al., 2019]	1.6731 ± 0.0099	1.1790 ± 0.0055	4145	11:05:37	00:10:43
STConvS2S-C (ours)	1.3699 ± 0.0024	0.9434 ± 0.0020	1040	03:34:52	00:02:48
STConvS2S-R (ours)	**1.2765** ± 0.0044	**0.8648** ± 0.0051	956	03:26:15	00:02:32
	$5 \rightarrow 15$				
Model	RMSE	MAE	Memory usage (MB)	Mean training time	Training time/epoch
ARIMA(0,1,0)	2.9241	2.5736	—	—	—
ARIMA(5,0,1)	2.2481	1.9077	—	—	—
ConvLSTM [Shi et al., 2015]	2.0728 ± 0.0069	1.4558 ± 0.0076	1810	5:29:30	00:07:32
PredRNN [Wang et al., 2017]	2.0237 ± 0.0067	1.4311 ± 0.0149	7415	11:45:48	00:17:03
MIM [Wang et al., 2019]	2.0287 ± 0.0361	1.4330 ± 0.0250	10673	19:19:00	00:31:19
STConvS2S-C (ours)	1.8739 ± 0.0107	1.2946 ± 0.0061	1457	**03:12:24**	**00:05:17**
STConvS2S-R (ours)	**1.8242** ± 0.0089	**1.2538** ± 0.0070	**1373**	03:31:46	00:05:23

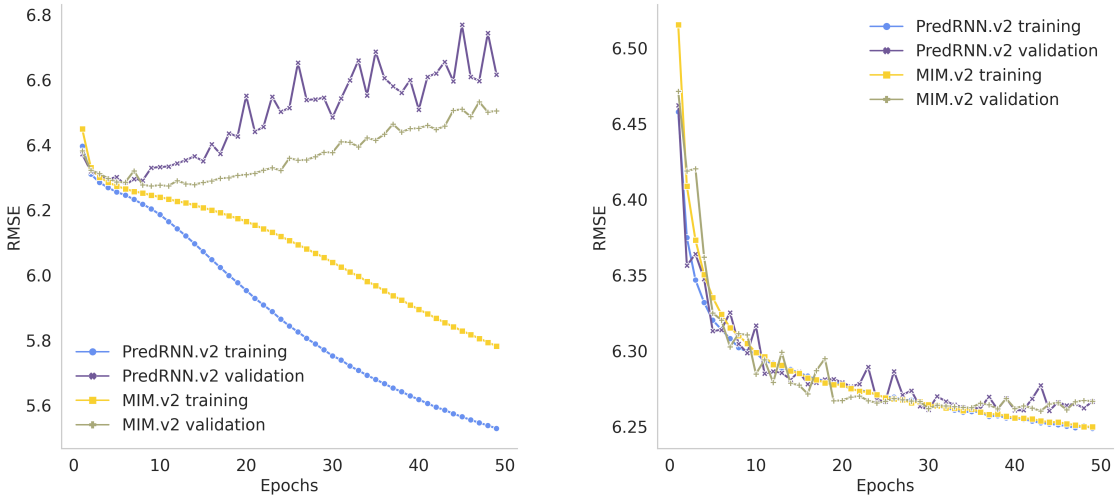
Figure V.5: Cumulative error based on both horizons ($5 \rightarrow 5$ and $5 \rightarrow 15$) using temperature dataset (CFSR). Evaluations on RMSE and MAE metrics.

V.5 CHIRPS Dataset: results and analysis

Similar to what we did with CFSR dataset, we divide the experiments into two phases. The first phase aims to investigate the best hyperparameter settings to adjust the models. In the second, we take into account the best version of each model and perform experiments with different initialization values for weight and bias to consolidate the analysis. In detail, we first set the hyperparameters as in the previous experiments on CFSR dataset. However, as the CHIRPS dataset is almost 4x smaller, all models overfit with those configurations. Thus, as an initial method to address this problem, we reduce the models' complexity by decreasing the number of layers (L) and the number of filters (F). However, we ensure fair comparability in the way we analyze the learning process when changing L (versions 1 and 3), K (versions 2 and 4) or F (versions 2 and 3). Again, all models were trained in mini-batch learning with 50 epochs, and RMSprop optimizer with a learning rate of 10^{-3}.

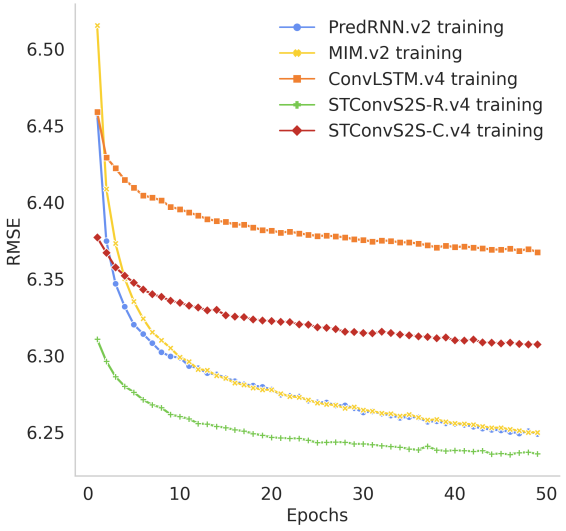
Although we reduced the complexity, the overfitting problem remained with PredRNN and MIM models. We apply dropout to improve their performance in comparison with our proposed models. As before, we apply a search to find the best dropout rate among 0.2, 0.5, 0.8. Figure V.6 (a) and (b) show the learning curves of these models with overfitting and with a dropout rate of 0.5 applied, respectively. Table V.4 shows the experimental results of predicting five grids into the future by observing five grids ($5 \rightarrow 5$). For all models, version 1 with the fewest layers has the lowest memory usage and was faster in training than the other versions. Another notable analysis is that version 2 an 4 consume the GPU memory equally for RNN-based models, thus increasing the kernel size affects only computational time.

Figure V.7 compares these results version by version. STConvS2S models outperform ConvL-



(a) Learning curve - overfitting

(b) Learning curve - dropout 0.5



(c) Training curve for all models

Figure V.6: Learning curves after running 50 epochs on the rainfall dataset (CHIRPS). (a) To exemplify, we select version 2 to illustrate the overfitting observed when analyzing the PredRNN and MIM models' training and validation curves. (b) The same version and models using dropout to improve its generalization. (c) Comparison of training curve for the best version for each model. STConvS2S-R achieved lower RMSE and, thus, better ability to learn spatiotemporal representations.

Table V.4: Evaluation of different settings on the CHIRPS dataset for STConvS2S and state-of-the-art methods, where the best version has the lowest RMSE value.

Model	Version	Setting	$5 \rightarrow 5$		
			RMSE	Training time	Memory usage (MB)
ConvLSTM [Shi et al., 2015]	1	L=1, K=3, F=16	6.4321	**00:06:39**	**746**
	2	L=2, K=3, F=8	6.4076	00:08:01	756
	3	L=2, K=3, F=16	6.3963	00:13:35	989
	4	L=2, K=5, F=8	**6.3681**	00:10:19	756
PredRNN [Wang et al., 2017]	1	L=1, K=3, F=16	6.2787	**00:30:31**	**1673**
	2	L=2, K=3, F=8	**6.2572**	00:34:19	1740
	3	L=2, K=3, F=16	6.2638	01:01:00	2775
	4	L=2, K=5, F=8	6.2600	00:39:24	1740
MIM [Wang et al., 2019]	1	L=1, K=3, F=16	6.3126	**00:25:32**	**1447**
	2	L=2, K=3, F=8	**6.2586**	00:43:52	2231
	3	L=2, K=3, F=16	6.2634	01:19:18	3521
	4	L=2, K=5, F=8	6.2626	00:48:00	2231
STConvS2S-C (ours)	1	L=1, K=3, F=16	6.3991	**00:06:16**	**609**
	2	L=2, K=3, F=8	6.3660	00:08:23	654
	3	L=2, K=3, F=16	6.3623	00:12:45	807
	4	L=2, K=5, F=8	**6.3131**	00:13:16	662
STConvS2S-R (ours)	1	L=1, K=3, F=16	6.3881	**00:07:03**	**607**
	2	L=2, K=3, F=8	6.3592	00:09:16	656
	3	L=2, K=3, F=16	6.3490	00:14:30	796
	4	L=2, K=5, F=8	**6.2269**	00:12:54	639

STM with compatible training time on all versions. Comparing the best version of each model (version 4 for STConvS2S models and ConvLSTM, and version2 for PredRNN and MIM), STConvS2S-R has the lowest prediction error and, compared to PredRNN, it is 3x faster and occupies only 1/3 of memory space. Besides, Figure V.6 (c) illustrates training in 50 epochs and indicates that STConvS2S-R learns the spatiotemporal representation of rainfall better than RNN-based architectures.

For the second phase, we train the models in the same set up as previously indicated for the CFSR dataset. We also include ARIMA methods as a baseline and evaluate the proposed architectures and state-of-the-art models in two tasks: feeding only five observations (grids) into the network and predicting the next 5 and 15 observations, denoted as $5 \rightarrow 5$ and $5 \rightarrow 15$, respectively. For both ARIMA models, predictions were performed throughout all the 2,500 time series, and for ARIMA$(5, 0, 1)$ was considered the previous five values in the sequence in each analysis. Results in Table V.5 indicate, similarly to CFSR dataset, that the rainfall forecast does not follow a random behavior against time, since ARIMA$(0, 1, 0)$ (random walk model) was lowest performing model. Also, demonstrate that STConvS2S-R achieves a better trade-off between computational cost and

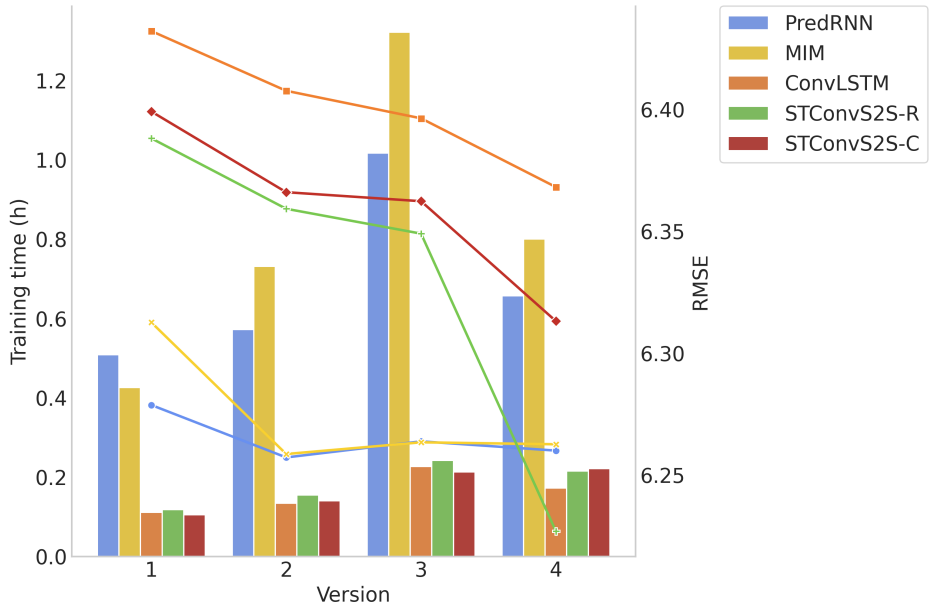


Figure V.7: Comparison between training time in hours (bar plot) and RMSE (line plot) for each model version on the rainfall dataset (CHIRPS).

Table V.5: Performance results for rainfall forecasting using the previous five observations (grids) to predict the next five observations ($5 \rightarrow 5$), and the next 15 observations ($5 \rightarrow 15$).

	$5 \rightarrow 5$				
Model	RMSE	MAE	Memory usage (MB)	Mean training time	Training time/epoch
ARIMA(0,1,0)	9.5538	8.0628	—	—	—
ARIMA(5,0,1)	7.4377	6.1694	—	—	—
ConvLSTM [Shi et al., 2015]	6.3666 ± 0.0019	2.9074 ± 0.0185	752	**00:15:15**	**00:00:13**
PredRNN [Wang et al., 2017]	6.2625 ± 0.0039	2.7880 ± 0.0110	1740	00:39:59	00:00:43
MIM [Wang et al., 2019]	6.2621 ± 0.0051	2.7900 ± 0.0178	2231	00:52:13	00:00:52
STConvS2S-C (ours)	6.3091 ± 0.0029	2.8487 ± 0.0280	662	00:15:54	00:00:15
STConvS2S-R (ours)	**6.2264** ± 0.0016	**2.7732** ± 0.0024	**639**	00:16:23	00:00:14
	$5 \rightarrow 15$				
Model	RMSE	MAE	Memory usage (MB)	Mean training time	Training time/epoch
ARIMA(0,1,0)	9.8396	7.1049	—	—	—
ARIMA(5,0,1)	7.9460	5.9379	—	—	—
ConvLSTM [Shi et al., 2015]	6.3244 ± 0.0025	2.8972 ± 0.0264	1308	00:44:30	**00:00:33**
PredRNN [Wang et al., 2017]	**6.2600** ± 0.0013	**2.7850** ± 0.0067	4115	01:53:30	00:01:58
MIM [Wang et al., 2019]	6.2722 ± 0.0020	2.7935 ± 0.0246	5276	02:48:38	00:02:34
STConvS2S-C (ours)	6.2962 ± 0.0039	2.8452 ± 0.0130	**916**	**00:35:43**	00:00:41
STConvS2S-R (ours)	6.2642 ± 0.0018	2.7971 ± 0.0187	928	00:42:50	00:00:40

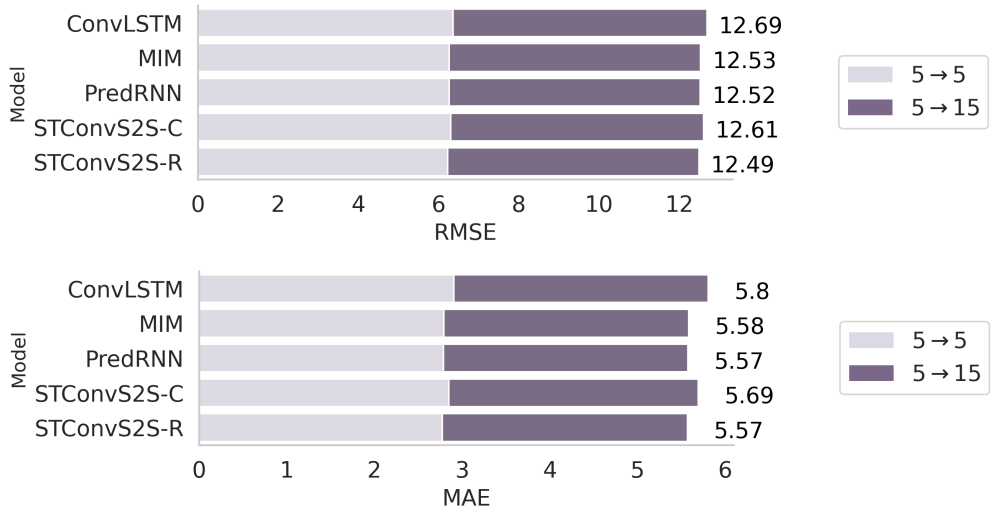


Figure V.8: Cumulative error based on both horizons ($5 \rightarrow 5$ and $5 \rightarrow 15$) using the rainfall dataset (CHIRPS). Evaluations on RMSE and MAE metrics.

prediction accuracy than state-of-the-art models in both tasks.

Figure V.8 summarizes these results in an overview of the cumulative error based on the two forecast horizons. Trained on the rainfall dataset, our proposed architecture equipped with the temporal reversed block achieves performance comparable to RNN-based architecture. Besides, it can predict short and even long sequences in the spatiotemporal context. Such a statement is confirmed by Figure V.9, which shows the observations at each time step for STConvS2S-R and PredRNN models. STConvS2S-R can predict in the long-term without many distortions since it presents a predictive result similar to the PredRNN. Given the high variability of rainfall, both models have difficulties in making an accurate forecast.

V.6 Ablation Study

We conduct a series of ablation studies[10], where the goal is to understand our architecture by removing/changing its main components and observing the impact on the evaluation metrics. For fair comparisons, we trained all models with the same settings of version 4 for CFSR (see Table V.2) and CHIRPS (see Table V.4) datasets. Besides analyzing the structure of our model, we also compare it with three models: vanilla 3D CNN, 3D Encoder-Decoder architecture [Racah et al., 2017], and a CNN model using (2+1)D blocks [Tran et al., 2018]. In Table V.6 and V.7, the results of these comparisons are shown on rows 1-3, rows 4-11 show the ablation experiments and on rows 12-13 the proposed models in this work. Following, we discuss the experimental results in detail.

- **Removal of the factorized convolutions.** In these experiments, there is no separation in temporal and spatial blocks in our architecture, since this split is only possible due to

[10] A neuroscience-inspired approach, which was used to uncover structure and organization in the brain [Meyes et al., 2016].

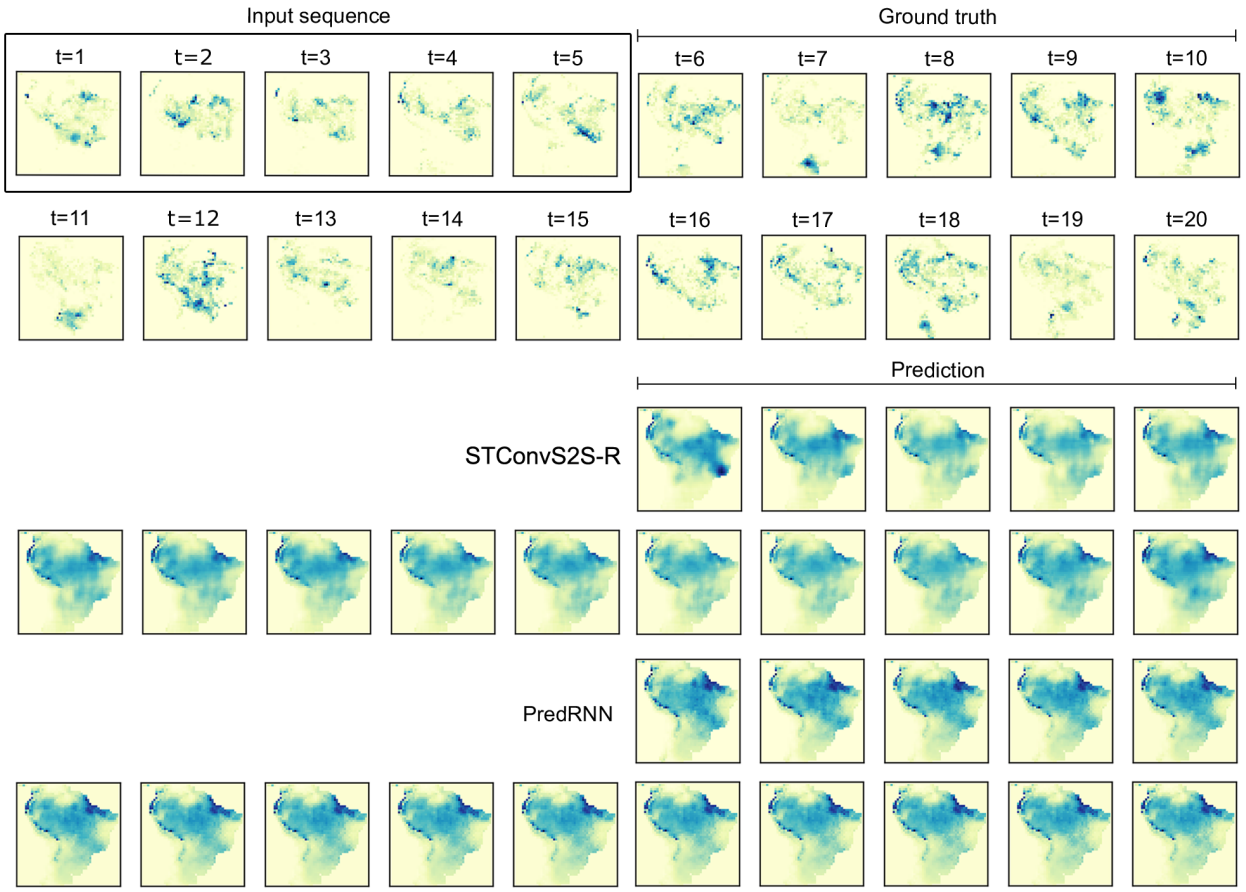


Figure V.9: Prediction example on test set of rainfall dataset (CHIRPS). Comparison between the best CNN-based and RNN-based models: STConvS2S-R and PredRNN, respectively

factorized convolutions. These models in relation to layers are similar to 3D CNN (see Figure IV.3 (a) and (c)). Removing factorized convolutions from the models underperform the proposed models in RMSE and MAE metrics (see rows 4 and 12; rows 5 and 13). To reinforce that the performance gain comes from design options rather than increased model parameters, we also performed experiments removing the progressive filters from our models (rows 10-11) for comparison. The models without factorized convolutions still performed worse.

- **Removal of the causal constraint.** In general, respecting the temporal order is more a restriction of the problem domain than an additional feature to improve models' performance. However, the STConvS2S-R model slightly improves the results on both evaluation metrics for CHIRPS datasets (comparison between rows 6 and 13 in Table V.7). On the other hand, the STConvS2S-C model results do not show the same contribution to performance.

- **Removal of the temporal block.** This experiment analyzes the importance of this component in our network since we propose two variations of STConvS2S based on the temporal block adopted. The results on row 7 indicate that although faster than the proposed models, this removal has a critical impact on RMSE and MAE, especially for the CFSR dataset.

- **Inverted blocks.** Understanding the influence of the temporal and spatial blocks on each other is not straightforward. Thus, to analyze the model structure, we change the blocks from (temporal $\Rightarrow$ spatial) to (spatial $\Rightarrow$ temporal). Both GPU memory usage and training time are very similar, comparing each STConvS2S model with the respective inverted version. Regarding the evaluation metrics, the inverted versions have the worst performance on both datasets.

- **Comparison with baselines methods.** There is no significant differentiation among STConvS2S models and the baselines on the CHIRPS dataset concerning memory usage and training time. These metrics almost increase twice on the CFSR dataset compared to 3D CNN and 3D Encoder-Decoder, but as a result, STConvS2S-R achieves a 4% improvement in RMSE over those same baselines. STConvS2S-R had a favorable or matching performance in the results of the experiments on both datasets, except for the MAE metric on the CHIRPS dataset comparing against (2+1)D Conv. STConvS2S-C does not perform as well as STConvS2S-R in these comparisons.

- **Comparison of strategies to satisfy the causal constraint.** The results show the superiority of STConvS2S-R compared to STConvS2S-C in the evaluation metrics in all the experiments. The hypothesis of STConvS2S-R to better predict is that this model does not add extra information in the input before performing the convolution operation on each layer inside the temporal block (STConvS2S-C add $\bar{t} - 1$ elements through zero-padding in the input). This study demonstrates the relevance of our original method (STConvS2S-R) to make convolutional layers satisfy the causal constraint during the learning process.

Table V.6: Quantitative comparison of ablation experiments, baseline methods using 3D convolutional layers, and our proposed models on temperature dataset (CFSR) for $5 \rightarrow 5$ task.

	Model	Factorized Conv.	Causal const.	RMSE	MAE	Memory usage (MB)	Training time
1.	3D CNN	—	—	1.3307	0.9015	578	01:13:54
2.	3D Encoder-Decoder	—	—	1.3327	0.9291	544	01:11:26
3.	(2+1)D Conv	✓	—	1.2944	0.8763	847	01:51:24
4.	STConvS2S-C	—	✓	1.4450	1.0068	605	01:39:47
5.	STConvS2S-R	—	✓	1.3477	0.9159	620	00:57:22
6.	STConvS2S	✓	—	1.2811	0.8645	884	02:00:52
7.	STConvS2S*	✓	—	1.6780	1.1828	740	01:14:47
8.	STConvS2S-C**	✓	✓	1.4152	0.9750	1000	02:15:46
9.	STConvS2S-R**	✓	✓	1.3052	0.8873	979	02:12:07
10.	STConvS2S-C***	✓	✓	1.4218	0.9821	698	01:04:58
11.	STConvS2S-R***	✓	✓	1.3315	0.9027	679	01:04:27
12.	STConvS2S-C	✓	✓	1.3791	0.9492	1040	02:25:26
13.	STConvS2S-R	✓	✓	1.2823	0.8719	956	02:13:37

* No temporal block
** Inverted (spatial ⇒ temporal)
*** No filter increase

Table V.7: Quantitative comparison of ablation experiments, baseline methods using 3D convolutional layers, and our proposed models on rainfall dataset (CHIRPS) for $5 \rightarrow 5$ task.

	Model	Factorized Conv.	Causal const.	RMSE	MAE	Memory usage (MB)	Training time
1.	3D CNN	—	—	6.2519	2.8519	534	00:09:27
2.	3D Encoder-Decoder	—	—	6.2540	2.7977	513	00:09:26
3.	(2+1)D Conv	✓	—	6.2323	2.7243	660	00:12:09
4.	STConvS2S-C	—	✓	6.3310	2.9161	553	00:12:38
5.	STConvS2S-R	—	✓	6.2576	2.8009	557	00:09:16
6.	STConvS2S	✓	—	6.2281	2.8134	609	00:10:19
7.	STConvS2S*	✓	—	6.3539	2.8980	572	00:06:57
8.	STConvS2S-C**	✓	✓	6.3255	2.8594	656	00:12:10
9.	STConvS2S-R**	✓	✓	6.2422	2.8342	633	00:12:21
10.	STConvS2S-C***	✓	✓	6.3171	2.8418	591	00:10:23
11.	STConvS2S-R***	✓	✓	6.2430	2.7996	582	00:10:10
12.	STConvS2S-C	✓	✓	6.3131	2.8327	662	00:13:16
13.	STConvS2S-R	✓	✓	6.2269	2.7765	639	00:12:54

* No temporal block
** Inverted (spatial ⇒ temporal)
*** No filter increase

Chapter VI Conclusions

VI.1 Retrospective Analysis

In this work, we propose a deep learning architecture to model spatiotemporal data, more specifically meteorological variables from South America, for short and long-term predictions. Our goal was to demonstrate that a 3D CNN-based model is applicable to sequence-to-sequence tasks, such as multi-step forecasting, and can predict more accurately the weather at the study location than RNN-based models. To this end, we conduct comparative studies with state-of-the-art methods [Shi et al., 2015; Wang et al., 2017, 2019] for spatiotemporal predictive learning, and traditional methods for time series forecasting [Box and Jenkins, 1970]. Based on the outlined objective, it was not the scope of this dissertation to research the best configurations (optimal values of hyperparameter) for our proposed model, neither for state-of-the-art nor the baseline models. Instead, we perform a fair comparison and use as initial hyperparameters configurations the values defined in Shi et al. [2015]. Further on, we proposed variations on this initial configuration to observe the models' behavior. Two datasets with different distributions were used in the experiments since the model predictive capability is strongly related to the data.

We implemented a framework using PyTorch 1.0 that builds the deep learning models, runs the experiments, saves checkpoints for reproducibility, and logs the results. We have performed 1146 experiments since we started monitoring on April 8, 2019, in which 631 were dedicated to testing the many hypotheses around the structure of our architecture. In detail, at the beginning, we adopted linear layers[1] as final layers. These layers are used to decrease the dimension C of the final feature map so that it corresponds to the dimension of the ground truth (expected result). This first decision was made based on CNN architectures commonly applied to image classification, but after many experiments, the results were not promising for our regression task. Recently, more studies were conducted to investigate the best location of the new temporal generator block, if it was in the middle just after the temporal block, or on the top of spatial block at the end of the architecture. Overall, the high number of experiments shows that, in the machine learning life-cycle, building the right model requires a lot of trial and error.

[1] Same as artificial neural network or multilayer perceptron

VI.2 Contributions

Predicting future information many steps ahead can be challenging, and a suitable sequence-to-sequence architecture to better represent spatiotemporal data for this purpose is still open for research. However, RNN-based models have been widely adopted in these cases [Shi et al., 2015; Wang et al., 2017, 2019]. Previously to this work, CNN-based architectures were not considered for this task, due to two limitations. Firstly, they do not respect the temporal order in the learning process. They also cannot generate an output sequence of length that is higher than the input sequence. Considering these limitations, we proposed STConvS2S, an end-to-end trainable deep learning architecture. STConvS2S can do spatiotemporal data forecasting by only using 3D convolutional layers.

First, we address the problem of causal constraint, proposing two variations of our architecture, one using the temporal causal block and the other, the temporal reversed block. The former adopts causal convolution, which is commonly used in 1D CNN. We also introduced a new technique in the temporal reversed block that makes it not violate the temporal order by applying a reverse function in the sequence. These implementations are essential for a fair comparison with state-of-the-art methods, which are causal models due to the chain-like structure of LSTM layers. To overcome the sequence length limitation, we designed a temporal generator block at the end of the architecture to extend the spatiotemporal data only in the time dimension.

We further compared our models with state-of-the-art models (ConvLSTM, PredRNN, MIM) through experimental studies in terms of both performance and time efficiency on meteorological datasets. The results indicate that STConvS2S models manage to analyze spatial and temporal data dependencies better since they have achieved superior performance in temperature forecasting. More specifically, in the first phase of the experiments, STConvS2S-R took only 1/4 of memory space, was almost 5x faster at training and improved the RMSE metric by 23% compared to MIM model. In the rainfall forecasting, STConvS2S-R had comparable results with the advantage of being 3x faster than PredRNN model and occupying only 1/3 of memory space. Based on the results, STConvS2S could be a natural choice for sequence-to-sequence tasks when using spatiotemporal data. We evaluate our architecture in the weather forecasting problem, but it is not limited to this domain. We expect that the results presented in this work will foment more research with comparisons between convolutional and recurrent architectures.

VI.3 Future work

Some proposals for future work can be explored, as there is space for improvement in terms of hyperparameter configuration and data manipulation. Firstly, it would be interesting to perform

an extensive parameter search to obtain better predictive results. Tuning machine learning models is an arduous and time-intensive task. Thus a possible choice should explore hyperparameters through automatic optimization methods, such as Random Search [Bergstra and Bengio, 2012] or Bayesian Optimisation [Klein et al., 2017]. For baselines, the *auto.arima* function [Hyndman and Khandakar, 2008] should be considered to search the best values for the parameters p, d and q.

Another important hyperparameter investigation would be to change the padding mode adopted in the STConvS2S-C architecture. We performed the experiments using PyTorch 1.0, which was the latest version when this research started. At that time, the only padding mode embedded in the convolutional layer was the zero-padding. Since version 1.5, released in April 2020, other options are available, such as the replicate mode [2].

It would be worth searching for ways to decrease the error in the rainfall dataset. Directions may include applying preprocessing techniques to sparse data and adding data from other geographic regions. The former is relevant since rainfall data are unbalanced with many periods without rain. The latter would be helpful to avoid overfitting the models.

Since the dataset size and other characteristics have a notable effect on the results of the experiments, a suggestion for future work is to use a diverse dataset, which means, to compose the meteorological dataset with other complementary data such as social and environmental variables. This extends the work from a univariate problem to a multivariate and can be achieved using the C dimension presented in the problem statement (Section IV.1). Another variation in the experiments would be to evaluate the models with longer time intervals to improve the sensitivity analysis.

The domain used to evaluate the architecture can be extended even further so that the model can predict video, finance or energy consumption. Moreover, a comparison with space-time autoregressive integrated moving average model (STARIMA) [Min et al., 2009], a spatiotemporal statistical approach, should be examined to serve as a strong baseline. Other architectures based on Transformer [Vaswani et al., 2017] would be an interesting point of investigation and comparison. These additional experiments would provide a more comprehensive evaluation to consolidate our architecture.

Finally, future work should look into more architectures for spatiotemporal data forecasting. A potential suggestion for this is to apply convolutional layers with attention mechanisms [Gehring et al., 2017] or tensor decomposition [Bahadori et al., 2014].

[2]See documentation at `https://pytorch.org/docs/stable/generated/torch.nn.Conv3d.html#conv3d`

Bibliography

Baboo, S. and Shereef, K. (2010). An efficient weather forecasting system using artificial neural network. *International Journal of Environmental Science and Development*, 1(4):321–326. 17

Babu, C. N. and Reddy, B. E. (2012). Predictive data mining on Average Global Temperature using variants of ARIMA models. *IEEE-International Conference on Advances in Engineering, Science and Management, ICAESM-2012*, pages 256–260. 17

Bahadori, M. T., Yu, Q., and Liu, Y. (2014). Fast Multivariate Spatio-temporal Analysis via Low Rank Tensor Learning. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (NeurIPS)*, pages 3491–3499. Curran Associates, Inc. 51

Bai, S., Zico Kolter, J., and Koltun, V. (2018). An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. arXiv:1803.01271v2. 3, 30

Bergstra, J. and Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13(10):281–305. 51

Box, G. and Jenkins, G. (1970). *Time Series Analysis: Forecasting and Control*. Holden-Day. 6, 19, 49

Brockwell, P. J. and Davis, R. A. (2016). *Introduction to Time Series and Forecasting*. Springer International Publishing, 3 edition. 6

Burkov, A. (2019). *The Hundred-Page Machine Learning Book*. Andriy Burkov. `http://themlbook.com/wiki/doku.php`. 11

Cheng, C., Zhang, C., Wei, Y., and Jiang, Y.-G. (2019). Sparse Temporal Causal Convolution for Efficient Action Modeling. In *Proceedings of the 27th ACM International Conference on Multimedia (MM'19)*, pages 592–600. ACM. 20, 27

Corchado, J. M. and Fyfe, C. (1999). Unsupervised neural method for temperature forecasting. *Artificial Intelligence in Engineering*, 13(4):351–357. 17

Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. arXiv:1603.07285. 13

Fujiwara, M., Wright, J. S., Manney, G. L., Gray, L. J., Anstey, J., Birner, T., Davis, S., Gerber, E. P., et al. (2017). Introduction to the sparc reanalysis intercomparison project (s-rip) and overview of the reanalysis systems. *Atmospheric Chemistry and Physics*, 17(2):1417–1452. 33

Funk, C., Peterson, P., Landsfeld, M., Pedreros, D., Verdin, J., Shukla, S., Husak, G., Rowland, J., Harrison, L., Hoell, A., and Michaelsen, J. (2015). The climate hazards infrared precipitation with stations - A new environmental record for monitoring extremes. *Scientific Data*, 2. 33

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, volume 3, pages 2029–2042. 3, 15, 30, 51

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`. 7, 10, 11, 14

Gujarati, D. N. (2002). *Basic Econometrics*. McGraw-Hill/Irwin, 4 edition. 6

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780. 14

Hyndman, R. and Khandakar, Y. (2008). Automatic Time Series Forecasting: The forecast Package for R. *Journal of Statistical Software, Articles*, 27(3):1–22. 51

Hyndman, R. J. and Athanasopoulos, G. (2018). *Forecasting: principles and practice*. OTexts, 2 edition. `https://otexts.com/fpp2`. 6

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning: with applications in R*. Springer. `http://faculty.marshall.usc.edu/gareth-james/ISL`. 9

Karpatne, A., Ebert-Uphoff, I., Ravela, S., Babaie, H. A., and Kumar, V. (2018). Machine Learning for the Geosciences: Challenges and Opportunities. *IEEE Transactions on Knowledge and Data Engineering*, PP(c):1. 2

Kim, S., Kim, H., Lee, J., Yoon, S., Kahou, S. E., Kashinath, K., and Prabhat (2019). Deep-hurricane-tracker: Tracking and forecasting extreme climate events. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision, WACV 2019*, pages 1761–1769. 18, 19, 21

Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. (2017). Fast bayesian hyperparameter optimization on large datasets. *Electronic Journal of Statistics*, 11(2):4945–4968. 51

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems (NeurIPS)*, pages 1097–1105. Curran Associates Inc. 3

LeCun, Y. and Bengio, Y. (1995). Convolutional Networks for Images, Speech, and Time Series. In Arbib, M. A., editor, *The Handbook of Brain Theory and Neural Networks*, pages 1–14. MIT Press, Cambridge, MA, USA. 10

Mehdizadeh, S. (2018). Assessing the potential of data-driven models for estimation of long-term monthly temperatures. *Computers and Electronics in Agriculture*, 144:114–125. 17

Meyes, R., Lu, M., de Puiseau, C. W., and Meisen, T. (2016). Ablation studies in artificial neural networks. arXiv:1901.08644. 45

Min, X., Hu, J., Chen, Q., Zhang, T., and Zhang, Y. (2009). Short-term traffic flow forecasting of urban network based on dynamic starima model. In *12th International IEEE Conference on Intelligent Transportation Systems*, pages 1–6. 51

Noh, H., Hong, S., and Han, B. (2015). Learning Deconvolution Network for Semantic Segmentation. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1520–1528. IEEE Computer Society. 13

Racah, E., Beckham, C., Maharaj, T., Ebrahimi Kahou, S., Prabhat, M., and Pal, C. (2017). ExtremeWeather: A large-scale climate dataset for semi-supervised detection, localization, and understanding of extreme weather events. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NeurIPS)*, pages 3402–3413. Curran Associates, Inc. , 3, 13, 18, 20, 21, 24, 30, 45

Rahman, M. R. and Lateh, H. (2017). Climate change in Bangladesh: a spatio-temporal analysis and simulation of recent temperature and rainfall data using GIS and time series analysis model. *Theoretical and Applied Climatology*, 128:27–41. 35

Reichstein, M., Camps-Valls, G., Stevens, B., Jung, M., Denzler, J., Carvalhais, N., and Prabhat (2019). Deep learning and process understanding for data-driven Earth system science. *Nature*, 566(7743):195–204. 1, 2, 3

Rolnick, D., Donti, P. L., Kaack, L. H., Kochanski, K., Lacoste, A., Sankaran, K., Ross, A. S., Milojevic-Dupont, N., et al. (2019). Tackling Climate Change with Machine Learning. arXiv:1906.05433. 1

Sagheer, A. and Kotb, M. (2019). Time series forecasting of petroleum production using deep lstm recurrent networks. *Neurocomputing*, 323:203 – 213. 17

Saha, S., Moorthi, S., Wu, X., Wang, et al. (2014). The NCEP Climate Forecast System Version 2. *Journal of Climate*, 27(6):2185–2208. 33

Salles, R., Belloze, K., Porto, F., Gonzalez, P. H., and Ogasawara, E. (2019). Nonstationary time series transformation methods: An experimental review. *Knowledge-Based Systems*, 164:274–291. 7

Sautermeister, B. (2016). Deep Learning Approaches to Predict Future Frames in Videos. Master's thesis, Technische Universitat Munchen, Germany. 9

Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-k., and Woo, W.-c. (2015). Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 802–810. 3, 15, 17, 18, 21, 24, 34, 36, 37, 40, 43, 44, 49, 50

Singh, G. and Cuzzolin, F. (2019). Recurrent Convolutions for Causal 3D CNNs. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*, pages 1–10. 3, 20

Sobel, I. and Feldman, G. (1968). A 3x3 isotropic gradient operator for image processing. In *Pattern Classification and Scene Analysis*, pages 271–272. John Wiley & Sons. 11

Souto, Y. M., Porto, F., Moura, A. M., and Bezerra, E. (2018). A Spatiotemporal Ensemble Approach to Rainfall Forecasting. In *Proceedings of the International Joint Conference on Neural Networks*, pages 574–581. 3, 18, 21

Stevens, E. and Antiga, L. (2019). *Deep Learning with PyTorch: Essential Excerpts*. Manning Publications. `https://pytorch.org/assets/deep-learning/Deep-Learning-with-PyTorch.pdf`. 8, 9, 10

Štulec, I., Petljak, K., and Naletina, D. (2019). Weather impact on retail sales: How can weather derivatives help with adverse weather deviations? *Journal of Retailing and Consumer Services*, 49(February):1–10. 1

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (NeurIPS)*, pages 3104–3112. Curran Associates, Inc. 14

Tran, D., Bourdev, L., Fergus, R., Torresani, L., and Paluri, M. (2015). Learning spatiotemporal features with 3d convolutional networks. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4489–4497. 12, 13, 24

Tran, D., Wang, H., Torresani, L., Ray, J., LeCun, Y., and Paluri, M. (2018). A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 6450–6459. IEEE Computer Society. 3, 18, 19, 20, 21, 24, 25, 26, 45

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. arXiv:1609.03499. 27

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is All you Need. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NeurIPS)*, pages 5998–6008. Curran Associates, Inc. 51

Vedenov, D. V. and Sanchez, L. (2011). Weather Derivatives as Risk Management Tool in Ecuador: A Case Study of Rice Production. 2011 Annual Meeting, February 5-8, 2011, Texas 98747, Southern Agricultural Economics Association. 1

Wang, Y., Long, M., Wang, J., Gao, Z., and Yu, P. S. (2017). Predrnn: Recurrent neural networks for predictive learning using spatiotemporal lstms. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, pages 879–888. Curran Associates, Inc. 3, 17, 18, 19, 21, 24, 36, 37, 40, 43, 44, 49, 50

Wang, Y., Zhang, J., Zhu, H., Long, M., Wang, J., and Yu, P. S. (2019). Memory in memory: A predictive neural network for learning higher-order non-stationarity from spatiotemporal dynamics. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 9146–9154. IEEE Computer Society. 3, 18, 21, 24, 36, 37, 40, 43, 44, 49, 50

Xu, S. and Niu, R. (2018). Displacement prediction of Baijiabao landslide based on empirical mode decomposition and long short-term memory neural network in Three Gorges area, China. *Computers and Geosciences*, 111:87–96. 17

Xu, Z., Cao, Y., and Kang, Y. (2019). Deep spatiotemporal residual early-late fusion network for city region vehicle emission pollution prediction. *Neurocomputing*, 355:183–199. 20

Yang, B., Sun, S., Li, J., Lin, X., and Tian, Y. (2019). Traffic flow prediction using LSTM with feature enhancement. *Neurocomputing*, 332:320–327. 17

Yuan, Y., Zhao, Y., and Wang, Q. (2018). Action recognition using spatial-optical data organization and sequential learning framework. *Neurocomputing*, 315:221–233. 3, 19

Zaytar, M. A. and Amrani, C. E. (2016). Sequence to Sequence Weather Forecasting with Long Short-Term Memory Recurrent Neural Networks. *International Journal of Computer Applications*, 143(11):7–11. 17

Zhang, J. (2017). Short-Term Traffic Prediction: Modelling Temporal-Spatial Features in Local Highway Networks with Deep Neural Networks. Master's thesis, Imperial College London, England. 14

Zhang, Q., Wang, H., Dong, J., Zhong, G., and Sun, X. (2017). Prediction of Sea Surface Temperature Using Long Short-Term Memory. *IEEE Geoscience and Remote Sensing Letters*, 14(10):1745–1749. 17