remote sensing

MDPI

Article

Higher-Order Convolutional Neural Networks for Essential Climate Variables Forecasting

Michalis Giannopoulos [1,2], Grigorios Tsagkatakis [1,2,*] and Panagiotis Tsakalides [1,2]

[1] Institute of Computer Science, Foundation for Research and Technology-Hellas (FORTH), 70013 Crete, Greece; mgiannop@ics.forth.gr (M.G.); tsakalid@ics.forth.gr (P.T.)
[2] Computer Science Department, University of Crete, 70013 Crete, Greece
* Correspondence: greg@ics.forth.gr

Abstract: Earth observation imaging technologies, particularly multispectral sensors, produce extensive high-dimensional data over time, thus offering a wealth of information on global dynamics. These data encapsulate crucial information in essential climate variables, such as varying levels of soil moisture and temperature. However, current cutting-edge machine learning models, including deep learning ones, often overlook the treasure trove of multidimensional data, thus analyzing each variable in isolation and losing critical interconnected information. In our study, we enhance conventional convolutional neural network models, specifically those based on the embedded temporal convolutional network framework, thus transforming them into models that inherently understand and interpret multidimensional correlations and dependencies. This transformation involves recasting the existing problem as a generalized case of N-dimensional observation analysis, which is followed by deriving essential forward and backward pass equations through tensor decompositions and compounded convolutions. Consequently, we adapt integral components of established embedded temporal convolutional network models, like encoder and decoder networks, thus enabling them to process 4D spatial time series data that encompass all essential climate variables concurrently. Through the rigorous exploration of diverse model architectures and an extensive evaluation of their forecasting prowess against top-tier methods, we utilize two new, long-term essential climate variables datasets with monthly intervals extending over four decades. Our empirical scrutiny, particularly focusing on soil temperature data, unveils that the innovative high-dimensional embedded temporal convolutional network model-centric approaches markedly excel in forecasting, thus surpassing their low-dimensional counterparts, even under the most challenging conditions characterized by a notable paucity of training data.

Keywords: ND convolutional neural networks; tensor decompositions; stacked convolutions; time series forecasting; essential climate variables

Citation: Giannopoulos, M.; Tsagkatakis, G.; Tsakalides, P. Higher-Order Convolutional Neural Networks for Essential Climate Variables Forecasting. *Remote Sens.* **2024**, *16*, 2020. https://doi.org/10.3390/rs16112020

Academic Editor: Nicolas Baghdadi

Received: 16 March 2024
Revised: 22 May 2024
Accepted: 30 May 2024
Published: 4 June 2024

1. Introduction

Monitoring Earth's climate change is a significant contemporary challenge, and data collected from different geographical areas over time play a critical role in understanding different physical characteristics and processes. *Essential Climate Variables* (ECVs) are crucial to this work, as they provide global-scale climate change insight and help assess climate risks and causes. The *Global Climate Observing System* (GCOS) (https://gcos.wmo.int/en/essential-climate-variables) (accessed on 1 February 2024) currently specifies 54 such variables [1], including soil temperature, soil moisture, above-ground biomass, and sea surface salinity, among others. ECVs are observed according to specific climate monitoring principles (identified by the GCOS), while the typical estimation of ECVs is usually based on numerical simulation models [2].

In the past years, data-driven *Machine Learning* (ML) methods have also contributed to the field in terms of mitigation and the prediction of climate change [3]. Moreover, the

quite prominent performance of *Deep Learning* (DL) methods in numerous *Remote Sensing* (RS) tasks [4], such as multispectral [5] and hyperspectral [6] classification, as well as multitemporal land cover classification [7,8], has also paved the way for its involvement in the field [9]. While DL methods have been considered for the instantaneous retrieval of the ECV values, the problem of forecasting has yet to be explored in sufficient detail. Furthermore, existing cutting-edge DL frameworks [9] forecast each ECV independently, thus failing to capitalize on existing correlations.

In RS research, considerable efforts concentrate on DL models for supervised classification tasks, thus spanning land cover classification [10], building detection [11], and scene classification [12], among others. The interest in spatiotemporal data analysis is escalating, thus prompted by its extensive application potential [13]. Urgent global challenges, like uncontrolled deforestation [14,15] and growing environmental pollution [16,17], accentuate the necessity for pioneering research and solutions, thus attracting academia, policymakers, and decision makers.

When the problems in question involve sequential time dependence, DL methods like *Recurrent Neural Networks* (RNNs) and *Long Short Term Memory* (LSTM) networks [18] are typically employed. While such methods can capture long-range temporal dependencies, they are not able to simultaneously capture both spatial and temporal observations, which are encoded in 3D spatiotemporal structures. A very successful approach toward that goal is the Conv-LSTM [19], which is a variant of the LSTM network in which the data flow through the cells by keeping the input dimension (3D in that case) instead of being just a 1D vector with features.

Beyond the RNN/LSTM methodologies in sequence modeling tasks, a particular subset of CNN models has emerged as a noteworthy substitute, specifically the *Temporal Convolutional Networks* (TCNs). The TCNs, which are 1D models, were initially developed for detailed action segmentation tasks [20]. In this pioneering work, two variants of TCNs were presented: the encoder–decoder TCN and the dilated TCN, with the latter drawing inspiration from the Wavenet model [21]. Another model conceived by Google DeepMind [22] leveraged 1D-CNN structures, utilizing dilated convolutions and residual blocks, and it outperformed the capabilities of existing RNN models. This model underscores the effectiveness of 1D-CNN architectures in complex sequence modeling tasks.

Of noticeable interest are works aiming to compensate for multidimensional data with more than three dimensions by adopting 4D architectures. More precisely, 4D-CNNs have been introduced for semantically segmenting cardiac volumetric sequences [23] and classifying multitemporal land cover [7,8], amongst others. Both approaches construct encoder–decoder networks, but while in [23], the upsampling is performed via kNN interpolation, in [7,8], it is executed via fully learnable 4D transpose convolution.

Focusing on spatiotemporal observation analysis, [19] proposed a Conv-LSTM model for the problem of precipitation nowcasting. To predict future radar maps from past radar echo sequences, the method adopted an LSTM architecture that employs convolutions in the input-to-state and state-to-state transitions. Moreover, Conv-LSTM architectures have also been employed in similar tasks, thus dealing with radar [24] and satellite [25] time series data.

In the study conducted in [26], another form of TCN architecture that utilizes residual blocks and dilated causal convolutions was presented. This model was benchmarked against various recurrent architectures, including LSTMs, *Gated Recurrent Units* (GRUs), and RNNs, in tasks typically employed for RNN evaluations, with results indicating superior performance of the TCN. However, these outcomes are not universal; for example, ref. [27] demonstrated scenarios where LSTM and CNN outperformed TCN, specifically in forecasting day-ahead electricity prices in Spain. This suggests that the effectiveness of these architectures can significantly depend on the context and specific characteristics of the task at hand.

In a noteworthy recent study, Villia et al. introduced an innovative technique for predicting ECV values, which is detailed in their paper [9]. The proposed model, termed

ETCN, merges three distinct networks: an encoder, a TCN, and a decoder. The proposed architecture is capable of analyzing spatiotemporal sequences and predicting the next time step. We discuss the method in detail in Section 2.2.

In essence, existing state-of-the-art ECVs' forecasting architectures comprise lower-order (2D or 3D) models in the quest to accurately forecast ECVs' values. Retooling these lower-order DL architectures for handling the intrinsically 4D simultaneous ECVs' forecasting task can lead to inferior performance, as demonstrated in Figure 1.

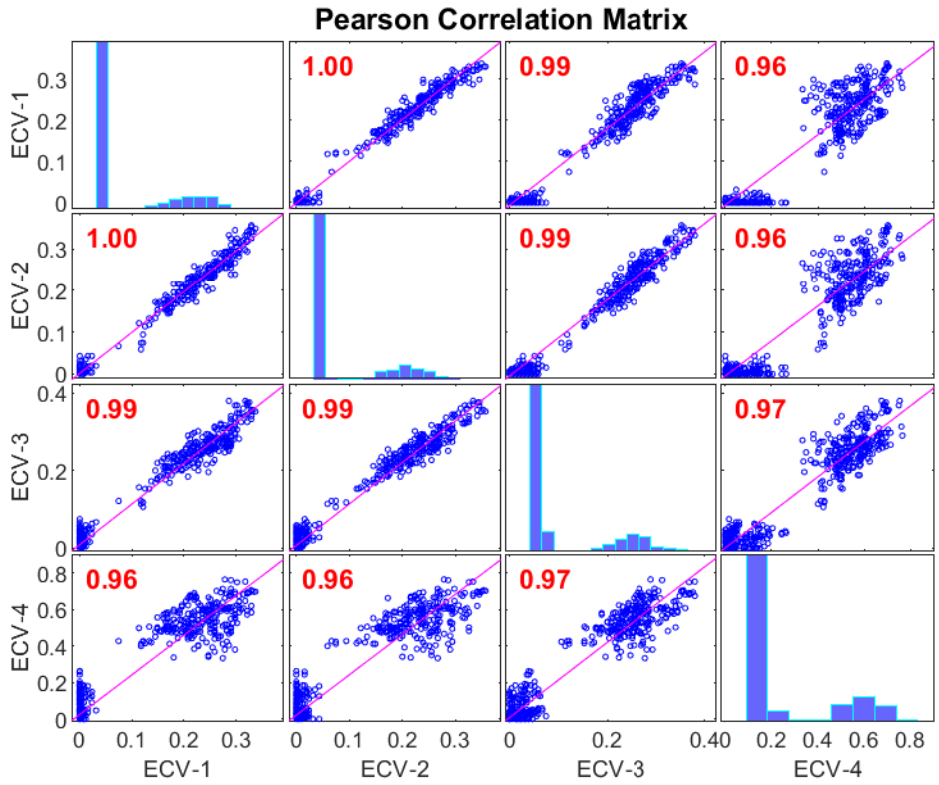


Figure 1. Time series data from various ECVs are teeming with valuable insights, thus capturing their temporal dynamics. Notably, ECVs like soil temperature and moisture at varying depths often exhibit significant correlations, which are evident in the slope of the least squares line drawn between each pair. These critical interdependencies among ECV time series can be effectively harnessed and maintained using meticulously crafted 4D model architectures, thus allowing for a more comprehensive and nuanced analysis.

Simultaneously measuring multiple ECVs such as soil temperature at different depths can provide more reliable estimations by exploiting the inherent correlations, thus mandating the need for models that are capable of performing the regression of high-dimensional structures. In this work, we diverge from traditional methodologies and introduce a sophisticated high-dimensional DL strategy for forecasting ECVs. We accomplish this by expanding the cutting-edge *Embedded Temporal Convolutional Network* (ETCN) architecture [9] into its more advanced high-dimensional variant, thus enabling the concurrent learning of spatiotemporal features across all ECVs under consideration. This holistic approach not only streamlines the forecasting process, but also leverages the intricate interdependencies present among the various climate variables.

In addition, we present two new ECV forecast datasets for the purpose of training and assessing the efficacy of the suggested model in comparison to leading approaches. We carry out comprehensive tests to measure each model's performance against various structures and multiple model parameters. The empirical outcomes from actual ECV data underscore the superiority and promise of the proposed advanced feature modeling in all cases under consideration.

The main contributions of this work include the following:

- The theoretical extension of lower-order *Convolutional Neural Networks* (CNNs) to their N-D analogs by providing a rigorous mathematical foundation for the forward and backward passes;
- The demonstration of issues and workarounds for incorporating this extension into a working DL framework;
- The design and development of a novel 4D-ETCN to tackle the ECVs' forecasting problem and its evaluation on properly designed datasets for the cause;
- The introduction of two novel datasets encoding satellite-derived geophysical parameters, specifically soil temperature at different levels, obtained on monthly periodicity over 40 years.

The paper is organized as follows: Section 2 introduces the proposed high-dimensional extension of the basic functionality of the convolutional layer for both forward and backward passes, with some parts of the detailed mathematical and conceptual study of this extension being placed in the accompanying appendices of the present work. In addition, we therein sketch the proposed 4D-ETCN architecture for tackling the ECVs' forecasting problem as an instance of the aforementioned extension for the problem at hand. Section 3 outlines the datasets formulated and the experimental frameworks established, and this is accompanied by the in-depth evaluations performed to determine the effectiveness of the introduced 4D-ETCN. This section also encompasses the consequential conclusions extracted from these investigative procedures. Furthermore, Section 4 furnishes an interpretation of the obtained results, thereby elucidating both the merits and drawbacks of the proposed approach in comparison to its counterparts. Section 5 distills the core conclusions and significant insights gained through this study.

2. Materials and Methods

2.1. Proposed Method: Higher-Order Convolutional Neural Networks

In this section, we demonstrate the case of 4D data analysis by deriving the required equations for generalizing the traditional low-order convolutional layers to their higher-order analogs. For that cause, we introduce two different alternatives, namely tensor decompositions and stacked convolutions.

2.1.1. Preliminary Definitions and Tensor Algebra

A tensor is a multidimensional array denoted by $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$. The order of $\boldsymbol{\mathcal{X}}$, N, is determined by the number of its dimensions, which are referred to as modes. Without loss of generality, in this work, we focus on 4-dimensional tensors, where two dimensions are employed for spatial encoding, one for temporal, and one for the specific modality/variable.

A typical operation on a tensor is to reorder the mode n vectors/fibers into a matrix, which is a transformation called "matricization"/unfolding/flattening. Note that this transformation is not unique in the sense that there exist many different ways of stacking up the mode n fibers of a tensor into a matrix. Formally, following the notation proposed in [28], the mode n unfolding of a tensor stacks its mode n fibers as columns to a matrix $\mathbf{X}_{(n)}$, which can be defined as follows:

Definition 1 (**Tensor Unfolding**). *An Nth order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ can be flattened into a matrix $\mathbf{X}_{(n)}$, where the tensor element $(i_1, i_2, \ldots, i_n)$ is mapped to the matrix element (i_n, j), with*

$$\begin{cases} j = 1 + \sum_{\substack{k=1 \\ k \neq n}}^{N} (i_k - 1) J_k \\ J_k = \prod_{\substack{m=1 \\ m \neq n}}^{k-1} I_m \end{cases} \tag{1}$$

An important aspect of tensor algebra is the diverse products it provides. Focusing on the relevant operations, one can define the following products, namely Kronecker product and Mode n product [29,30], as follows:

Definition 2 (**Kronecker Product**). *Given two matrices* $\mathbf{X} \in \mathbb{R}^{I \times J}$ *and* $\mathbf{Y} \in \mathbb{R}^{K \times L}$, *their Kronecker product is a matrix defined as follows:*

$$\mathbf{Z} = \mathbf{X} \otimes \mathbf{Y} = \begin{bmatrix} x_{11}\mathbf{Y} & x_{12}\mathbf{Y} & \cdots & x_{1J}\mathbf{Y} \\ x_{21}\mathbf{Y} & x_{22}\mathbf{Y} & \cdots & x_{2J}\mathbf{Y} \\ \vdots & \vdots & \ddots & \vdots \\ x_{I1}\mathbf{Y} & x_{I2}\mathbf{Y} & \cdots & x_{IJ}\mathbf{Y} \end{bmatrix} \in \mathbb{R}^{IK \times JL} \tag{2}$$

Definition 3 (**Mode** n **Product**). *Given a tensor* $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ *and a matrix* $\mathbf{Y} \in \mathbb{R}^{K \times I_N}$, *their Mode n/Tensor Times Matrix product is a tensor defined as follows:*

$$\boldsymbol{\mathcal{Z}} = \boldsymbol{\mathcal{X}} \times_{(n)} \mathbf{Y} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_{n-1} \times K \times I_{n+1} \times \ldots \times I_N} \tag{3}$$

In essence, each mode n fiber of tensor $\boldsymbol{\mathcal{X}}$ is multiplied by matrix $\mathbf{Y}$. A more eloquent interpretation though is to express the mode n product in terms of unfolded tensors as follows:

$$\boldsymbol{\mathcal{Z}} = \boldsymbol{\mathcal{X}} \times_{(n)} \mathbf{Y} \iff \mathbf{Z}_{(n)} = \mathbf{Y}\mathbf{X}_{(n)}. \tag{4}$$

Probably the most prominent advantage of tensors is the diverse decompositions they offer for data of multiple dimensions. Here, we provide a brief overview of the one that is useful in the context of this work (i.e., Tucker), while the other most famous ones (e.g., CP/PARAFAC/CANDECOMP and Tensor-Train) are explained in details in seminal works in the field [28–31]. The Tucker decomposition has taken its name from the Ledyard R. Tucker [32–34] and is widely known as an extension of the classical PCA and SVD methods to higher dimensions [35]. More precisely, we have the following definition.

Definition 4 (**Tucker Decomposition**). *Given a tensor* $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ *of order* N, *Tucker decomposition decomposes it into a core tensor* $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{R_1 \times R_2 \times \ldots \times R_N}$ *and a set of factor matrices* $\{\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}\}$, *with* $\mathbf{U}^{(N)} = [\mathbf{u}_1^{(n)}; \mathbf{u}_2^{(n)}; \ldots; \mathbf{u}_{R_N}^{(n)}] \in \mathbb{R}^{R_N \times I_N}$ *and* $n = 1, 2, \ldots, N$. *The core tensor* $\boldsymbol{\mathcal{G}}$ *is multiplied along each of its modes with the factor matrices* $\{\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)}\}$ *in the following way:*

$$\begin{aligned} \boldsymbol{\mathcal{X}} &= \boldsymbol{\mathcal{G}} \times_{(1)} \mathbf{U}^{(1)} \times_{(2)} \mathbf{U}^{(2)} \times_{(3)} \ldots \times_{(N)} \mathbf{U}^{(N)} \\ &= \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_n=1}^{R_N} g_{r_1 r_2 \ldots r_N} (\mathbf{u}_{r_1}^{(1)} \circ \mathbf{u}_{r_2}^{(2)} \circ \ldots \circ \mathbf{u}_{r_N}^{(N)}) \\ &= \left\| \boldsymbol{\mathcal{G}}; \mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \ldots, \mathbf{U}^{(N)} \right\| \end{aligned} \tag{5}$$

2.1.2. Forward Propagation in ND CNNs

In order to extend CNNs operating on low-dimensional observations to their high-order analogs, we have to generalize the basic concept or convolutions to the N-D case. For that cause, we propose two different ways of computation: one via tensor decompositions and one via stacking of lower-dimensional convolutions.

In the former case, by employing the notations proposed by Cichocki et al. in [36], we initially define the convolution between two tensors along a specific mode, namely *Partial (Mode n) Convolution*, in the following way:

Definition 5 (Partial (Mode n) Convolution). *Given two tensors $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ and $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{J_1 \times J_2 \times \ldots \times J_N}$ of order N, their Partial (Mode n) Convolution derives a tensor $\boldsymbol{\mathcal{Z}} = \boldsymbol{\mathcal{X}} \boxdot_{(n)} \boldsymbol{\mathcal{Y}}$ whose subtensors are computed as follows:*

$$\boldsymbol{\mathcal{Z}}(k_1, k_2, \ldots, :, \ldots, k_N) = \boldsymbol{\mathcal{X}}(i_1, i_2, \ldots, :, \ldots, i_N) \star \boldsymbol{\mathcal{Y}}(j_1, j_2, \ldots, :, \ldots, j_N) \tag{6}$$

where $\star$ stands for the linear convolution operation, and $k_l = \overline{i_l j_l}$ for $l = 1, 2, \ldots, N$. The multi-index $i = \overline{i_1 i_2 \ldots i_N}$ is an index which takes all possible combinations of values of indices $i_1 i_2 \ldots i_N$ with $i_n = 1, 2, \ldots, I_n$ and $n = 1, 2, \ldots, N$, and it is defined in the following way:

$$i = \overline{i_1 i_2 \ldots i_N} = i_1 + (i_2 - 1) I_1 + (i_3 - 1) I_1 I_2 + \ldots + (i_N - 1) I_1 \ldots I_{N-1} \tag{7}$$

In the context of this study, we restrict only to a specific case of Equation (6), namely partial (mode 1) convolution of matrices $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2}$ and $\mathbf{Y} \in \mathbb{R}^{J_1 \times J_2}$, which is defined as follows:

$$\begin{cases} \mathbf{Z} = \mathbf{X} \boxdot_{(1)} \mathbf{Y} \\ \mathbf{Z}(:, k_2) = \mathbf{X}(:, i_2) \star \mathbf{Y}(:, j_2) \end{cases} \tag{8}$$

Given the above definitions and notations, the N-D Tucker convolution is then defined [36] as follows:

Definition 6 (N-D Tucker Convolution). *Given two tensors $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ and $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{J_1 \times J_2 \times \ldots \times J_N}$ of order N, their N-D Tucker convolution $\boldsymbol{\mathcal{Z}} = \boldsymbol{\mathcal{X}} \star \boldsymbol{\mathcal{Y}}$ is computed in the following way:*

1. *Perform Tucker decomposition on tensor $\boldsymbol{\mathcal{X}}$, and obtain its core tensor $\boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{X}}} \in \mathbb{R}^{R_1 \times R_2 \times \ldots \times R_N}$ and its factor matrices $\{\mathbf{X}^{(1)}, \mathbf{X}^{(2)}, \ldots, \mathbf{X}^{(N)}\}$:*

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{X}}} \times_{(1)} \mathbf{X}^{(1)} \times_{(2)} \mathbf{X}^{(2)} \times_{(3)} \ldots \times_{(N)} \mathbf{X}^{(N)} = \left\| \boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{X}}}; \mathbf{X}^{(1)}, \mathbf{X}^{(2)}, \ldots, \mathbf{X}^{(N)} \right\| \tag{9}$$

2. *Perform Tucker decomposition on tensor $\boldsymbol{\mathcal{Y}}$, and obtain its core tensor $\boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{Y}}} \in \mathbb{R}^{Q_1 \times Q_2 \times \ldots \times Q_N}$ and its factor matrices $\{\mathbf{Y}^{(1)}, \mathbf{Y}^{(2)}, \ldots, \mathbf{Y}^{(N)}\}$:*

$$\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{Y}}} \times_{(1)} \mathbf{Y}^{(1)} \times_{(2)} \mathbf{Y}^{(2)} \times_{(3)} \ldots \times_{(N)} \mathbf{Y}^{(N)} = \left\| \boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{Y}}}; \mathbf{Y}^{(1)}, \mathbf{Y}^{(2)}, \ldots, \mathbf{Y}^{(N)} \right\| \tag{10}$$

3. *Derive the core tensor of the N-D Tucker convolution, $\boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{Z}}}$, by combining the core tensors $\boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{X}}}$ and $\boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{Y}}}$ via their Kronocker product:*

$$\boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{Z}}} = \boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{X}}} \otimes \boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{Y}}} \in \mathbb{R}^{R_1 Q_1 \times R_2 Q_2 \times \ldots \times R_N Q_N} \tag{11}$$

4. *Derive the factor matrices of the N-D Tucker convolution, $\{\mathbf{Z}^{(1)}, \mathbf{Z}^{(2)}, \ldots, \mathbf{Z}^{(N)}\}$, by combining the factor matrices $\{\mathbf{X}^{(1)}, \mathbf{X}^{(2)}, \ldots, \mathbf{X}^{(N)}\}$ and $\{\mathbf{Y}^{(1)}, \mathbf{Y}^{(2)}, \ldots, \mathbf{Y}^{(N)}\}$ via their partial (mode 1) convolution:*

$$\begin{cases} \mathbf{Z}^{(n)} = \mathbf{X}^{(n)} \boxdot_{(1)} \mathbf{Y}^{(n)} \\ \mathbf{Z}^{(n)}(:, s_n) = \mathbf{X}^{(n)}(:, r_n) \star \mathbf{Y}^{(n)}(:, q_n) \\ s_n = \overline{r_n q_n} = 1, 2, \ldots, R_n Q_n \end{cases} \tag{12}$$

5. *Derive the output tensor of the N-D Tucker convolution, $\boldsymbol{\mathcal{Z}}$, by combining the core tensor $\boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{Z}}}$ and the factor matrices $\{\mathbf{Z}^{(1)}, \mathbf{Z}^{(2)}, \ldots, \mathbf{Z}^{(N)}\}$ via their Tucker composition:*

$$\boldsymbol{\mathcal{Z}} = \boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{Z}}} \times_{(1)} \mathbf{Z}^{(1)} \times_{(2)} \mathbf{Z}^{(2)} \times_{(3)} \ldots \times_{(N)} \mathbf{Z}^{(N)} = \left\| \boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{Z}}}; \mathbf{Z}^{(1)}, \mathbf{Z}^{(2)}, \ldots, \mathbf{Z}^{(N)} \right\| \tag{13}$$

The definition of the N-D Tucker convolution [36] between two third-order tensors $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$ is illustrated in Figure 2.

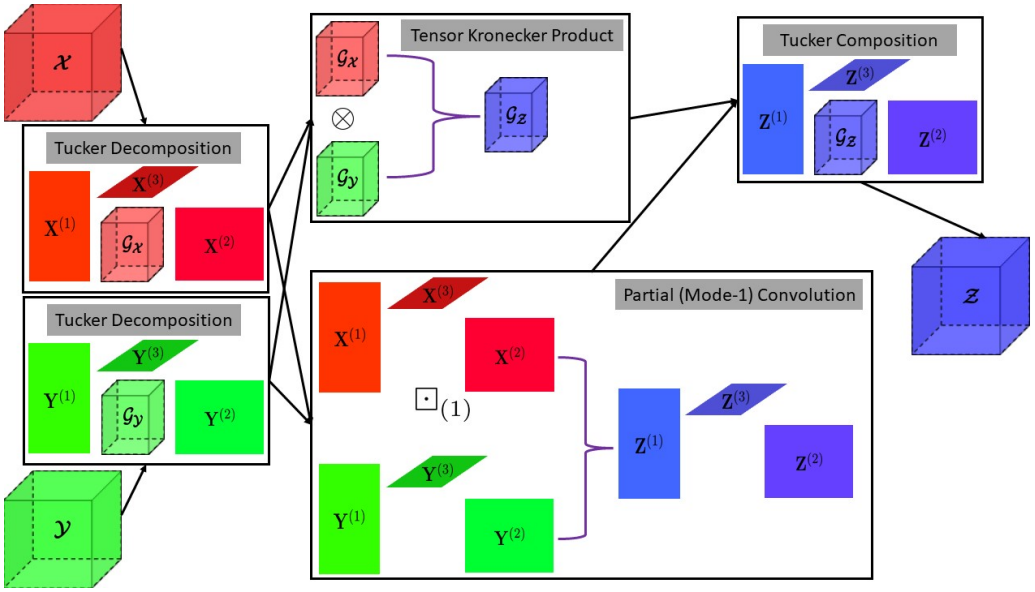


Figure 2. Tucker convolution between two third-order tensors $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$.

2.1.3. Backpropagation in ND CNNs

Based on the aforementioned discussion, the convolution of two ND tensors $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ and $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{J_1 \times J_2 \times \ldots \times J_N}$ can be applied in the context of a CNN by substituting $\boldsymbol{\mathcal{X}}$ with $\boldsymbol{\mathcal{I}}$ as the layer's input, $\boldsymbol{\mathcal{Y}}$ with $\boldsymbol{\mathcal{W}}$ as the layer's trainable weights/kernel, and adding the respective bias term $\boldsymbol{b}$. Hence, we can express the basic operation of an ND convolutional layer, l, in the "direct-sum" form as follows:

$$\begin{aligned} \mathcal{Z}_{i_1,i_2,\ldots,i_N}(l) &= \sum_{j_1}\sum_{j_2}\cdots\sum_{j_N} \mathcal{W}_{j_1,j_2,\ldots,j_N}(l)\mathcal{I}_{i_1-j_1,i_2-j_2,\ldots,i_N-j_N}(l-1) + \boldsymbol{b}(l) \\ &= \boldsymbol{\mathcal{W}}(l) \star \mathcal{I}_{i_1,i_2,\ldots,i_N}(l-1) + \boldsymbol{b}(l) \end{aligned} \tag{14}$$

where $\{j_1, j_2, \ldots, j_N\}$ span the dimensions of the kernel, $\{i_1, i_2, \ldots, i_N\}$ span the dimensions of the input, and $\boldsymbol{b}$ is the bias term. As can be seen from Equation (14), the input of the previous layer $(l-1)$ serves for the computation of the current output layer (l), and the input of the current layer l can be computed as follows:

$$\mathcal{I}_{i_1,i_2,\ldots,i_N}(l) = f(\mathcal{Z}_{i_1,i_2,\ldots,i_N}(l)) \tag{15}$$

where f is a selected activation function (e.g., ReLU, tanh). With Equations (14) and (15), every value in the forward pass of an ND convolutional layer can be computed, so what remains to be defined is the respective equations during the backward pass.

In order to be able to derive the backward pass equations of an ND convolutional layer, we should dive in its functionality depicted in Figure 3. More precisely, given the input $\boldsymbol{\mathcal{I}}$ and the weights $\boldsymbol{\mathcal{W}}$, the layer outputs its N-D convolution during the forward pass. On the contrary, during the backward pass, the los gradient from the next layer, $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{Z}}}$, must be propagated to the previous layers. For that purpose, the local gradients (i.e., $\frac{\partial \boldsymbol{\mathcal{Z}}}{\partial \boldsymbol{\mathcal{I}}}$, $\frac{\partial \boldsymbol{\mathcal{Z}}}{\partial \boldsymbol{\mathcal{W}}}$) alongside the chain rule will help us calculate the gradients with respect to the input (i.e., $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{I}}}$) and the filters (i.e., $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{W}}}$). These gradients need to be updated for the following reasons:

- Since $\boldsymbol{\mathcal{I}}$ is the output of the previous layer, $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{I}}}$ becomes the loss gradient for the previous layer.
- $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{W}}}$ is used to update the filters of the convolutional layer via the gradient step:

$$\boldsymbol{\mathcal{W}}_{New} = \boldsymbol{\mathcal{W}}_{Old} - \mu \frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{W}}} \tag{16}$$

where μ is the step size parameter, which serves as the learning rate of the optimization problem.

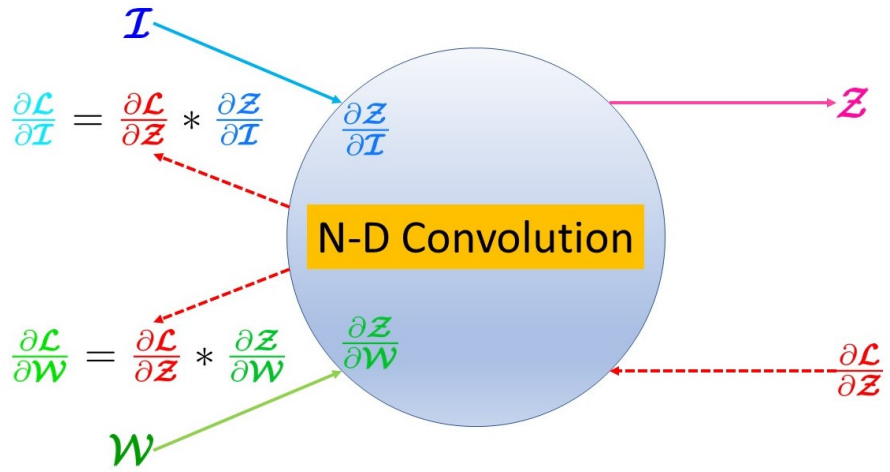


Figure 3. Computation of the loss gradients of ND convolutional layer. The loss gradient of the previous layer ($\frac{\partial \mathcal{L}}{\partial \mathcal{Z}}$) is propagated to other layers via the help of local gradients ($\frac{\partial \mathcal{Z}}{\partial \mathcal{I}}$, $\frac{\partial \mathcal{Z}}{\partial \mathcal{W}}$) and the chain rule.

More precisely, the aforementioned gradients can be computed via the chain rule, as depicted in Figure 3:

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial \mathcal{I}} = \frac{\partial \mathcal{L}}{\partial \mathcal{Z}} * \frac{\partial \mathcal{Z}}{\partial \mathcal{I}} \\ \frac{\partial \mathcal{L}}{\partial \mathcal{W}} = \frac{\partial \mathcal{L}}{\partial \mathcal{Z}} * \frac{\partial \mathcal{Z}}{\partial \mathcal{W}} \end{cases} \tag{17}$$

Due to space limitations, the precise mathematical computation of Equation (17) in the closed form is provided, for all involved gradients, in Appendix B. In order to set our notation, we summarize the results therein for each one of them as follows:

- Loss gradient from the next layer:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{Z}} = f'(\mathcal{Z}_{i_1,i_2,\ldots,i_N}(l)) \left[\delta_{i_1,i_2,\ldots,i_N}(l+1) \star rot_N(\mathcal{W}(l+1)) \right] \tag{18}$$

- Gradient with respect to the layer's input:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{I}} = conv\left(\frac{\partial \mathcal{L}}{\partial \mathcal{Z}}, \mathcal{W}(l) \right) \tag{19}$$

- Gradient with respect to the layer's weights:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{W}} = conv\left(\frac{\partial \mathcal{L}}{\partial \mathcal{Z}}, rot_N(\mathcal{I}(l-1)) \right) \tag{20}$$

- Gradient with respect to the layer's bias:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}} = \sum_{i_1} \sum_{i_2} \cdots \sum_{i_N} \left(\frac{\partial \mathcal{L}}{\partial \mathcal{Z}} \right) \tag{21}$$

Equations (19) and (20) sketch the general methodology for computing the desired gradients, and they indicate that in reality both the forward and backward passes of an ND convolutional layer are convolutions.

2.1.4. Efficient Implementation: Stacking (N-1)D Convolutions

After successfully expanding the convolution operator from the traditional 2D and 3D scenarios to the multidimensional (ND) context using the Tucker convolution model, we integrated this advanced operation into the DL paradigm. Specifically, we developed a bespoke Tucker convolution layer within the Tensorflow framework [37], thus enabling its application in the creation of higher-dimensional CNN models. Our initial experimentation with this Tensorflow adaptation was conducted on two lower-dimensional benchmark classification datasets, namely MNIST-2D and MNIST-3D. Through this exploration, our objective was to evaluate the computational efficiency of our DL adaptation vis à vis built-in convolutional layers.

To that end, we have constructed two identical model architectures for the two datasets, with only 1 convolutional layer (built-in and Tucker) before the final classification (i.e., fully connected) one. In addition, we have frozen the weights of the fully connected layer in order to measure fairly among the time needed for the respective models to train. To keep the architectures as simple as possible, we used only 4 filters in each convolution case, and we trained the models for up to only 10 epochs. Moreover, we employed only 128 samples for training purposes (i.e., out of the 60,000 available) and accordingly 128 for test purposes (i.e., out of the 10,000 available), while the batch size at each experiment was set to 32.

In Table 1, we report the computational times (in seconds) needed for training each of the aforementioned models for the MNIST classification tasks. As becomes evident, when the Tucker convolution model is incorporated with DL platforms, it faces severe computational problems, although outside this framework, it is at least comparable to its competitors.

Table 1. Computational time (in seconds) required by the Tucker convolution model and the builtin Tensorflow one, for the MNIST classification problems. When incorporated with deep learning platforms, the Tucker convolution model clearly does not scale well.

Dataset	# Trainable Parameters	Builtin	Tucker
MNIST-2D	40	10.4867	**504.861**
MNIST-3D	112	9.2061	**791.394**

By observing the results of Table 1, we notice the nonscalable required computational times when the Tucker convolution model is incorporated into the DL framework. Based on our extensive experimental studies, we concluded that this handicap is mainly attributed to TensorFlow's fast libraries implementation of convolution, which is performed across every input channel and for every desired output filter in a batch mode [38,39].

On the contrary, our implementation based on tensor decompositions has to make use of specific toolboxes/packages for the respective computations. Towards that end, we have considered the excellent review work of Psarras et al. [40], which surveys the current tensor software landscape. Based on the thorough lists provided therein, we mainly focused our efforts on Python-based Tensorly [41], which provides all tensor operations and decompositions required, as well as GPU computation capabilities. In particular, we designed our Tucker convolution layer either employing as backend Tensorflow (version: 2.3.0) [37], PyTorch (version: 1.9.0) [42], or NumPy (version: 1.18.5) [43]. Unfortunately, neither of our attempts fructified, even when we tried the MATLAB-based Tensor Toolbox package [44].

To circumvent the limitations imposed by GPU constraints, it becomes imperative to leverage Tensorflow's foundational elements. Specifically, to fully harness the entirety of the data's informational content, we present an N-dimensional (ND) convolutional layer/module that ingeniously employs the corresponding $(N-1)$D layer/module. Specifically, the computed value for a neuron in the convolved output, situated at coordinates $(i_1, \ldots, i_{N-1}, i_N)$, is articulated as follows:

$$
\begin{aligned}
y_{i_1,\ldots,i_{N-1},i_N} &= f\left(\sum_{c}^{C_{in}}\sum_{j_N=0}^{J_N-1}\sum_{j_{N-1}=0}^{J_{N-1}-1}\cdots\sum_{j_1=0}^{J_1-1} w_{j_1,\ldots,j_{N-1},j_N} x_{c,(i_1+j_1)(\ldots)(i_{N-1}+j_{N-1})(i_N+j_N)} + b_{j_1,\ldots,j_{N-1},j_N}\right) \\
&= f\left(\sum_{j_N=0}^{J_N-1}\left[\sum_{c}^{C_{in}}\sum_{j_{N-1}=0}^{J_{N-1}-1}\cdots\sum_{j_1=0}^{J_1-1} w_{j_1,\ldots,j_{N-1},j_N} x_{c,(i_N+j_N)(i_{N-1}+j_{N-1})(\ldots)(i_1+j_1)}\right] + b_{j_1,\ldots,j_{N-1},j_N}\right) \\
&= f\left(\sum_{j_N=0}^{J_N-1} C_{(N-1)-D} + b_{j_1,\ldots,j_{N-1},j_N}\right)
\end{aligned}
\tag{22}
$$

where $f(.)$ is the activation function, $w_{j_1,\ldots,j_{N-1},j_N}$ stands for the value of the kernel connected to the current feature map at position $(j_1,\ldots,j_{N-1},j_N)$, $x_{c,(i_1+j_1)\ldots(i_{N-1}+j_{N-1})(i_N+j_N)}$ represents the value of the input neuron at input channel c, $b_{j_1,\ldots,j_{N-1},j_N}$ is the bias of the computed feature map, C_{in} denotes the number of original channels (i.e., first layer) or the number of feature maps of the previous layer (i.e., intermediate layer), and $j_1,\ldots,j_{N-1},j_N$ are the kernel's dimensions across each of its modes.

The restructuring of the convolution sums as demonstrated in Equation (22) is attainable because convolution is inherently a linear operation, thus permitting the reordering of the summation processes. This adaptative strategy was previously outlined in sources such as [7,8,23,45], thus serving as a foundational pathway transitioning from 3D CNNs to more complex 4D variations. By embracing this methodology, we engage in the aggregation of numerous sequences of $(N-1)$D convolutions—represented by the term $C_{(N-1)-D}$ in (22)—extended along the final, or Nth, dimension.

From an execution standpoint, additional restructuring of the corresponding for-loop was enacted as per the frameworks suggested in [7,8,23]. This involved the convolution of $(N-1)$D input frames with their respective $(N-1)$D filter frames, which is a crucial adjustment for our custom layer to facilitate genuine (as opposed to separable) ND convolution. An exhaustive description of the respective Stacked convolution algorithm, applicable to tensors of up to 4 dimensions, is comprehensively articulated in Stacked Convolution Algorithm. The repository which contains the Python-scripts involving the implementation of ND CNNs can be found in the "Supplementary Materials" at the back matter heading after Section 5.

2.2. Proposed Simultaneous Spatiotemporal ECVs' Forecasting Architecture

Within this section, our initial focus involves outlining the state-of-the-art model architecture intended for generalization in addressing the current challenge. Following this, we delineate the specific scenario under scrutiny for the ECVs' forecasting task, thus elucidating the comprehensive mathematical elements detailing the application of the proposed 4D-ETCNs to address the task at hand. Concluding this section, we furnish implementation particulars and construct the corresponding network architecture.

2.2.1. Embedded Temporal Convolutional Network—ETCN

As previously discussed in Section 1, ETCNs [9] stand out as the leading models for the current task. These models employ a three-part architecture featuring an encoder network, a TCN, and a decoder network. To optimize the processing of the data's temporal aspect, the authors implemented the *Time-Distributed* wrapper around 2D-CNNs within the encoder and decoder segments of the model. The encoder segment is structured with convolutional and max pooling layer blocks, whereas the decoder reverses this process by utilizing 2D transpose convolutional and batch normalization layers. The TCN component of the architecture is characterized by residual blocks that include 1D dilated causal convolutional layers, dropout layers, and ReLU activation layers. This particular configuration of causal convolution ensures that the model's prediction at any specific time is solely based on past inputs.

2.2.2. 4D-ETCN for ECVs' Forecasting Modeling

In addressing the task of forecasting time series ECVs, a problem inherently involving 4D inputs (comprising 2D spatial, temporal, and ECV modality dimensions), traditional 2D-CNNs fall short, as they inevitably overlook crucial information. Even 3D models can only maintain additional temporal or ECV modality information, not both. Recognizing this limitation, we present a novel 4D model designed to concurrently harness correlations across such diverse dimensions. Our focus is on the ETCN models, and we extend the state-of-the-art architectures initially proposed in [9] to suit the specific requirements of this multidimensional forecasting challenge.

As illustrated in Figure 4, the proposed 4D-ETCN model operates on the principle of efficiency, thus directly ingesting raw time series ECV data—essentially, sequences of 3D ECV data—that are devoid of any prerequisite data manipulation. These raw data are propelled through an intricate cascade of high-dimensional operations within the ETCN framework, including convolution, pooling, and upsampling, with each stage amplifying in sophistication as data traverse further into the network's layers.

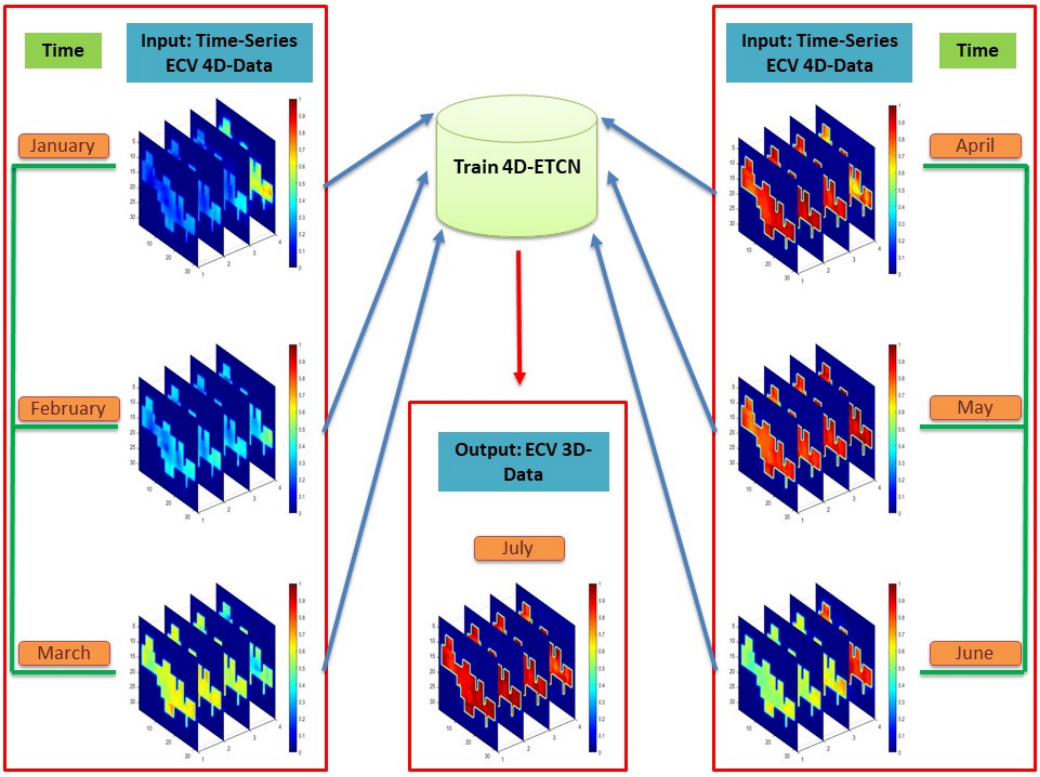


Figure 4. The proposed method involves the concurrent forecasting of multiple ECV time series data. Original 4D data, which consist of a time series of 3D data, are input into our specially devised higher-order ETCN model. This model retains all pertinent information up to the regression phase by performing all critical operations such as convolutions and transpose convolutions within a 4D framework.

In parallel, the model meticulously conducts spatial downsampling followed by upsampling, which is a strategy that is pivotal for distilling more salient, feature-rich representations from the data, all while assiduously conserving the intricacies of temporal progression and ECV modality specifics up until the culmination point of the network's regression compartment.

The proposed architecture stands out by virtue of its capacity to exploit the entirety of the high-dimensional data, thus steadfastly performing computations within the very dimensions native to the original data. This approach marks a significant departure from the methodologies employed by lower-order models, which typically impose a preprocessing

stage that inevitably simplifies or compresses data ahead of any network training, thus potentially leading to the loss of valuable information.

2.2.3. 4D-ETCN Modules for ECVs' Feature Learning

In Sections 2.1.2–2.1.4 we provided a detailed pipeline for the transition from conventional lower-order CNNs to the ND ones via tensor decompositions and stacked convolutions. Aiming to incorporate them to the ECV forecasting problem, we have to extend existing lower-order CNN models for that cause to their high-dimensional analogs. Since the vanilla ETCN model [9] consists of the state-of-the-art CNN architecture, we can directly make use of the results derived above in order to construct a higher-order ETCN model. More precisely, the ECV forecasting problem at hand can be considered as a subcase of the ND general case, where the dimensionality of the data is equal to $N = 4$ (once the time series ECV data form a 4D tensor). In that sense, we can directly extend all lower-order convolutional layers contained in vanilla ETCN architecture [9] with 4D ones.

Beyond the convolutional layers, it is also necessary to expand the max pooling and transpose convolutional layers/modules referenced in [9] to accommodate four-dimensional functionality. The justification for this enhancement aligns with the underlying logic for the convolutional layer's expansion, which is comprehensively explained in [7]. By integrating these 4D layers, we equip ourselves with the essential components to elevate the standard lower-dimensional ETCN model to a high-dimensional iteration. Consequently, our advanced 4D-ETCN framework is capable of processing time series ECV data in their original dimensions, thus adeptly navigating the complex demands of the concurrent forecasting challenge presented. This approach not only maintains the integrity of the data's dimensions but also enhances the model's capacity to interpret and learn from the high-dimensional nature of the input.

2.2.4. 4D-ETCN Architectures for ECVs' Forecasting

The primary objective of this study is to evaluate the efficacy of ETCN models in addressing the specified problem and to gain insights into the potential advantages of the proposed 4D structure over its lower-dimensional counterparts. To this end, we use the standard ETCN [9] as a reference model for comparison while developing its 4D equivalent. Our analysis concentrates on ETCN frameworks given their established superiority in forecasting tasks within the ETCN context [9], thus even surpassing the performance of Conv-LSTM structures that are specifically crafted for such challenges.

To explore the potential benefits of our advanced ETCN model compared to the standard version, our architecture retains the fundamental components (namely, the encoder, TCN, and decoder) but only expands those elements amenable to higher-dimensional processing units (i.e., the encoder and decoder). This approach ensures consistency for a fair comparison while allowing us to directly assess the impact and advantages of high-dimensional processing in forecasting tasks.

Each architecture consists of multiple layers, including convolutional, transpose convolutional, max pooling, activation, dropout, and batch normalization layers. These are arranged in "layers-stacks" within the encoder, TCN, and decoder portions of the model, thus adhering to the framework put forth in the conventional ETCN [9]. Specifically, the encoder and the TCN sections are each composed of three stacks, while the decoder section is constructed using two stacks.

Each stack within the encoder includes a convolutional layer, which is directly succeeded by a ReLU activation layer and a max pooling layer, with the exception of the final stack. Concluding the sequence of stacks in the encoder segment involves a reshape layer, which is tasked with reorganizing the data to be compatible with the subsequent 1D-TCN segment of the model. This systematic layering is instrumental in the progressive feature extraction and data transformation essential for the model's forecasting tasks.

The TCN segment of the model, drawing inspiration from the framework presented in [26], substitutes standard 1D convolutional layers with residual blocks. Each of these

blocks is composed of two 1D convolutional layers that maintain uniformity in terms of output filter size and number. Following each convolutional layer are ReLU activation and dropout layers, with weight normalization [46] applied to the convolutional filters. Unchangingly, the input for each residual block within the TCN segment is combined with its output; this sum, after undergoing ReLU activation, produces the final output for the residual block.

Consistent with the vanilla ETCN model's TCN [9], our model employs causal and dilated 1D convolutions, thus ensuring that the output at any given time is predicated solely on present and preceding inputs. Since the TCN output is a one-dimensional vector, matching in dimensions the number of output filters of the last residual block, we reintroduce a reshape layer that transforms the 1D vectors to 4D tensors, thus rendering them suitable for the decoder segment of the model. This methodical approach facilitates the nuanced processing required for accurate temporal predictions.

Conversely, each stack within the decoder segment is composed of a transpose convolutional layer, which is succeeded by ReLU activation and batch normalization layers. Culminating the architecture, the prediction layer incorporates a single-filter convolutional layer for generating the requisite 3D output. This layer is then followed by a Sigmoid activation layer. This configuration within the decoder segment is crucial for reconstructing the spatial–temporal features from the compressed representations and predicting the continuous values pertinent to the environmental conditions under observation.

Given that our samples inherently possess four dimensions, encompassing two spatial, one temporal, and one ECV modality dimension , designated as (H, W, T, ECV_M), their utilization for training within the confines of a vanilla ETCN model necessitates a form of dimensional reduction or "collapsing" for each sample. While the original ETCN model was evaluated on individual ECVs, assertions from [46] suggest its applicability to multichannel images. Leveraging this, we adapted their architecture to handle the "active information" within the ECV time series data (i.e., spatial and temporal dimensions), thus interpreting the ECV modality as distinct channel information. In stark contrast, our innovatively proposed 4D-ETCN model operates directly within the data's intrinsic dimensions, thereby fully engaging with the complete gamut of information present. This approach eases a more holistic and integrated exploitation of the data's spatiotemporal dynamics.

The encoder and decoder segments of the devised ETCN model can be construed as U-Nets, which are predominantly employed in segmentation tasks. In conventional U-Net models, both samples and labels typically share the same dimensions, i.e., 2D for image segmentation [47], and 3D for volumetric image segmentation [48]). Unlike existing approaches, the proposed high-dimensional model confronts the challenge of disparate dimensionality between samples (4D) and targets (3D).

To navigate this complexity while retaining maximal information, we adopt a tactic akin to the one outlined in [7]: all operations within the ETCN model are executed in their corresponding high-dimensional spaces up until the final regression layer. At this juncture, we utilize filters with a kernel size of 1 across all dimensions, thus maintaining unit strides across every dimension barring the temporal one slated for reduction (i.e., $(1, 1, T, 1)$). This method ensures that the supplemental, nonspatial information remains unaltered until the immediate prelude to regression, wherein it is judiciously downsampled during the ultimate step, thereby aligning the network training with the 3D ground-truth targets.

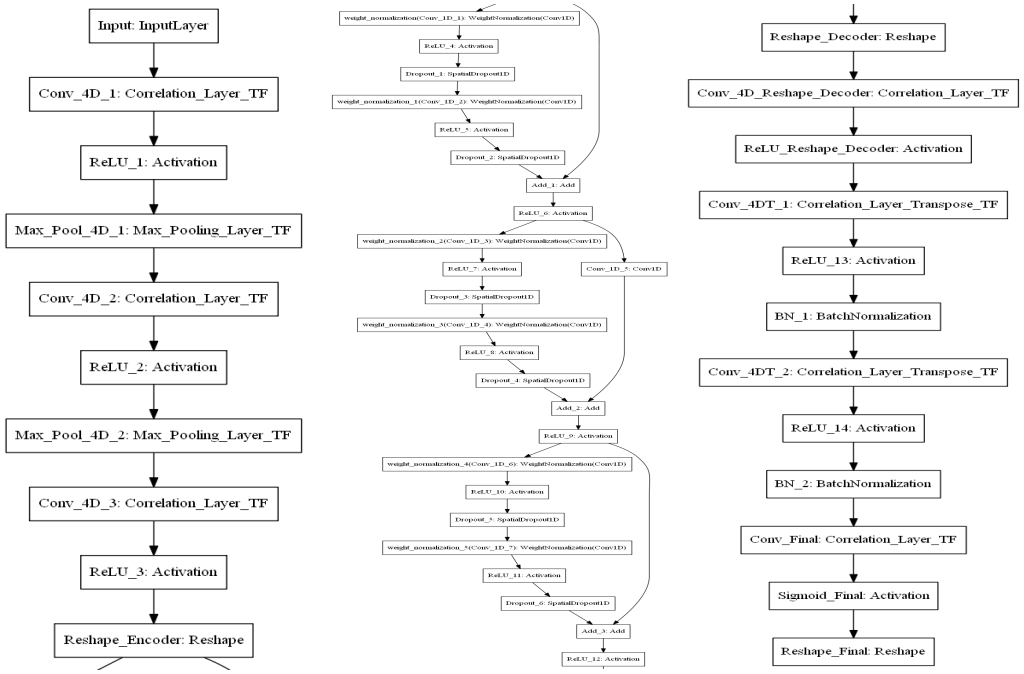
In relation to the foundational topology of the 4D-ETCN, all convolutional layers utilize a modest kernel size of 2 across every dimension combined with "same" padding. Strides are set to unity across all dimensions during all convolutions. Concurrently, the quantity of filters experiences an augmentation as the network delves deeper into the encoder segment of the ETCN: The initial stack of layers possesses a filter count equivalent to a predefined parameter (termed "starting-neurons"), which experiences a doubling for the subsequent stacks (for instance, both the second and third stacks possess filters numbering 2× "starting-neurons").

In line with our goal to perform spatial downsampling solely on input data samples, we employ max pooling layers with pool size and strides equal to 2 in spatial dimensions and 1 in the temporal and the ECV modality dimensions. "Same" padding is consistently applied across all max pooling layers, and dropout layers are established with a rate of 0.3.

For the transpose convolutional layers, we utilize compact kernels with a size of 2 across all dimensions, thus consistently applying "same" padding for all versions of the 4D-ETCN models. Due to the unique spatial downsampling in the encoder section, strides for the transpose convolutions are appropriately modified, which are designated as 2 for spatial dimensions (to accommodate the corresponding upsampling) and 1 for the other dimensions.

Regarding the TCN segment of our model, the fundamental topology incorporates 1D filters of size 2, with a filter count of 2× "starting-neurons" at the first and third stack and 64 at the second. The dilation rate of the 1D kernels within the TCN's stacks adheres to the protocol proposed in the vanilla ETCN setup [9] and is sequentially arranged at $1 - 2 - 4$.

And indicative sketch of our 4D-ETCN model architecture can be seen in Figure 5. As described above, the architecture consists of three parts: encoder (Figure 5a), TCN (Figure 5b), and decoder (Figure 5c), which are combined in the way explained earlier in the quest to derive accurate ECV forecasts.



(**a**) Model architecture—encoder. (**b**) Model architecture—TCN. (**c**) Model architecture—decoder.

Figure 5. The proposed 4D-ETCN model architecture: Each higher-order input sample undergoes processing by the encoder, TCN, and decoder parts of the model in order to be used for accurate ECV forecasting purposes.

Taking into account that we face a regression task, MSE was used as the loss function. All examined models' weights were randomly initialized using the Glorot Uniform initializer [49]. Concerning the optimization learning algorithm, we employed the Adam scheme [50] with a constant learning rate of 0.0001 and exponential decay rates for the first and second moment estimates equal to $\beta_1 = 0.9$ and $\beta_2 = 0.999$, respectively. Training of each below model was performed using a batch size of 4 samples (since the loading of high-dimensional data can become quite memory intensive) for up to 100 epochs.

3. Results

In this section, our first task is to outline the datasets introduced for our imminent experiments, followed by an explanation of the experimental framework implemented. Next, we establish the evaluation criteria used to compare various DL models. Subsequently, we delve into a comprehensive analysis of the results derived from our chosen approach across diverse experimental scenarios, thus utilizing multiple configurations to meticulously assess the impact of each pertinent parameter throughout the entire ECV forecasting process. Concluding, we juxtapose our proposed methodology against prevailing state-of-the-art techniques, thus affirming its superior efficacy.

3.1. Dataset Description

The evaluation of DL strategies and the cultivation of ML models pivot significantly on the accessibility of appropriate training and test sets. This becomes particularly challenging in the realm of high-dimensional data, where relevant datasets are notably limited. In our study, we gauged the effectiveness of the suggested system through its precision in estimating ECVs. In particular, our efforts were concentrated on training both the proposed model and established methodologies using historical observational time series with the objective of learning to forecast forthcoming values of soil temperature at various depths.

We utilized data that are openly accessible from the *ERA5-Land Monthly Averaged* dataset (https://developers.google.com/earth-engine/datasets/catalog/ECMWF_ERA5_LAND_MONTHLY) (accessed on 1 February 2024) [51]. ERA5-Land is a reanalysis dataset that offers a coherent perspective on the historical progression of terrestrial variables, spanning multiple decades, and does so with improved resolution compared to its ERA5 counterpart. By offering data that extend back over several decades, reanalysis furnishes a detailed account of historical climate conditions. This dataset encompasses all 50 variables available on the *Climate Data Store* (CDS). The ERA5-Land data, available since 1981, come with precomputed monthly mean averages, thus enhancing the ease and speed of data access for various applications.

For the purposes of our research, we focused on soil temperature variables, specifically levels 1–4. These levels represent the temperature strata within the soil, specifically layer 1 (0–7 cm), layer 2 (7–28 cm), layer 3 (28–100 cm), and layer 4 (100–289 cm) of the ECMWF Integrated Forecasting System. The surface reference point is at 0 cm, and each of these four climatic variables is gauged on the Kelvin scale. The soil temperature is determined at the midpoint of each layer, with heat transfer computations performed at the layers' interfaces. The model presupposes the absence of heat transfer beneath the base of the deepest layer.

In order to employ the aforementioned dataset, some preprocessing steps had to take place in advance. More precisely, our goal was to create a 4D imagery time series dataset comprising each of the 2D imagery of each climate variable measured at all available sampling times available. Towards that end, we made use of the Google Earth Engine platform [52], with dataset provider Copernicus CDS and dataset availability 1 January 1981–1 December 2020. With this setup, we initially selected our region of interest for downloading our time series imagery at a $\approx$9 km resolution provided by the platform. Subsequently, we selected the four different climate variables mentioned earlier for downloading, which were accompanied by the desired dimensions of the imagery. At this point, we have to mention that since the high-dimensional models' memory demands are in direct alignment with their input data size, we chose the spatial dimensions to be patches of size 32×32. In addition, given that the samples were taken on a monthly basis starting from January 1981 up to December 2020, the temporal dimension of our created time series data is equal to $12 \times 40 = 480$. The last dimension of our data is considered the climate variable under investigation, and hence its cardinality is equal to four. Since every climate variable is measured at Kelvin scale, we normalized each one of them separately in the [0–255] scale in order to be interpreted as 8-bit images from the DL models. Before feeding the input to

the network, the pixel values were further scaled to the range from 0 to 1 toward enhancing the efficiency of the imminent training process.

To ensure that our study's findings are not confined to a specific dataset, we evaluated the performance of the proposed methodologies and those for comparison on two distinct datasets. Specifically, the first *Region Of Interest* (ROI) chosen for our investigation is the province/island of Crete, which is situated in Greece. Concurrently, the second encompasses various territories in Central Italy predominantly concentrated around the Lazio region. Concerning the Crete ROI, its latitude and longitude limits, $[latlim, lonlim]$, are specified as two-element vectors of the form $[latmin, latmax] = [34.7702, 35.7035]$ and $[lonmin, lonmax] = [23.385, 26.5106]$. As far as the Italy ROI is concerned, the respective limits are formed as $[latmin, latmax] = [41.570252, 42.504503]$ and $[lonmin, lonmax] = [12.101440, 15.806824]$, respectively. Both selected areas are characterized by a rich diversity in land surfaces, thus featuring an assortment of plains, forests, and mountainous terrains.

3.2. Experimental Setup

In this study, our primary aim is to execute a one-step forward prediction by developing a model that ingests a set quantity of images from a spatiotemporal series of multiple climate variables and forecasts an identical count of images, with each displaced one timestep ahead. To achieve this, it is imperative to pre-establish the number of timesteps employed in this predictive task, thus denoting our data's temporal dimension. We opted for a timestep of six months, which is a duration sufficient to discern potential disparities among the scrutinized climate variables. By extracting sequential time series segments with overlap, 473 unique samples (with the final month of 2020 excluded), each with dimensions of (32,32,6,4), were produced from the forty-year span of the monthly data available.

Since the problem at hand is a forecasting task, we selected the first 38 years (i.e., 1981–2018) for training purposes, while the 2 most recent years (i.e., 2019–2020) were kept for testing (in a holdout split). Furthermore, we considered a validation set formed by 25% of the training samples, which is a split that was performed uniformly at random to avoid bias in favor/against specific time periods. The data splitting process ended up with a sample distribution summarized in Table 2.

Table 2. Dataset split among training–validation–test sets. The first 38 years are used for training (75%) and validation (25%) purposes, while the last 2 ones are used for testing.

Total Samples	Training Set	Validation Set	Test Set
473	342	114	17

In the forthcoming experiments, we adhered to the dataset splits previously mentioned. Each model underwent training within the designated training set, while its performance underwent validation in the validation set through the exploration of various hyperparameter configurations. Subsequently, the model's performance was evaluated based on the optimal validation loss achieved throughout training, regardless of whether this occurred at the final epoch. The models exhibiting the best performance were then used in the test set for a single evaluation, thus determining the ultimate performance of the model.

The formulated model structures were crafted within the Python programming environment, thus utilizing the TensorFlow and Keras [53] libraries. This choice was deliberate, given their extensive customization capabilities throughout various aspects of a DL model—an essential feature for our study, given the implementation of diverse custom layers. Additionally, the compatibility of both libraries with GPU acceleration significantly reduced the computational time required for the training process. In our experimental setup, we employed NVIDIA's *Quadro P4000* GPU model, equipped with 8 GB of RAM, to further enhance computational efficiency.

3.3. Evaluation Metrics for Essential Climate Variables Forecasting

Since the problem at hand is essentially a pixel-level regression task, we relied on several ML metrics for the evaluation of the designed models. For that cause, we employed reconstruction error metrics, image quantitative metrics, and correlation metrics. For their definitions, let x_i stand for the pixels of the ground truth image x, while y_i stands for the pixels of the predicted image y (n in total).

Concerning the reconstruction error metrics, the *Mean Square Error* (MSE), the *Mean Absolute Error* (MAE), and the *Root Mean Square Error* (RMSE) between x and y are defined as follows:

$$\begin{cases} MSE = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2 \\ MAE = \frac{1}{n}\sum_{i=1}^{n} \mid x_i - y_i \mid \\ RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2} \end{cases} \tag{23}$$

However, if there are biases in either the mean or the amplitude of fluctuations of the predictions, the RMSE could be severely compromised. To alleviate this, the mean bias can be removed from the RMSE, thus leading to the *Unbiased Root Mean Square Error* (ubRMSE), which is computed as follows:

$$\begin{cases} Bias = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i) \\ ubRMSE = \sqrt{RMSE^2 - Bias^2} \end{cases} \tag{24}$$

For the above reconstruction error metrics, values closer to zero indicate a better-performing model.

Since pixel-level regression can be seen as an image reconstruction task, reconstruction error metrics are by no means the only suitable ones to be measured. Indeed, among image fidelity metrics, we initially reported the *Peak Signal-to-Noise Ratio* (PSNR) metric, which is defined as follows:

$$PSNR = 10\log_{10}\left(\frac{MAX_I^2}{MSE}\right) \tag{25}$$

In Equation (25), MAX_I^2 is the maximum possible pixel value in the image (e.g., for images scaled in the range $[0, 1]$ $MAX_I^2 = 1$, while for 8-bit images, $MAX_I^2 = 255$), while MSE is defined as in Equation (23). Higher PSNR values indicate better image fidelity.

Moreover, the *Structural Similarity Index Measure* (SSIM) [54] is a perception-based metric that is mainly used for measuring the similarity between two images. The SSIM extracts structure, luminance and contrast from both images x and y, and its values range between 0 and 1 (with higher values desired). Formally, the SSIM is defined as follows:

$$SSIM = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{26}$$

where μ_x stands for the mean of ground truth image x, μ_y stands for the mean of predicted image y, σ_x^2 and σ_y^2 represent the respective variances, and σ_{xy} is the covariance between them. In addition, $c_1 = (k_1 D)^2$ and $c_2 = (k_2 D)^2$ are two variables used to stabilize the division with weak denominator, D is the dynamic range of the pixel values (i.e., $D = 1$ for images scaled in $[0, 1]$, while $D = 255$ for 8-bit images), and finally k_1 and k_2 are by default set as $k_1 = 0.01$ and $k_2 = 0.03$.

Treating ground truth and predicted images as two statistical samples, the *Pearson Linear Correlation Coefficient* (PLCC) is a measure of their linear correlation. The PLCC is essentially a normalized measurement of the covariance such that the result always lies in

$[-1, 1]$, and, as with covariance itself, the PLCC only reflects a linear correlation of variables. Mathematically, the PLCC is defined as follows:

$$PLCC = \frac{\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^{n}(x_i - \mu_x)^2}\sqrt{\sum_{i=1}^{n}(y_i - \mu_y)^2}} \tag{27}$$

A value of $PLCC = 1$ implies that the pixels of the ground truth and predicted images are perfectly positive linearly correlated, while a $PLCC = -1$ implies total negative linear correlation. Values of the PLCC close to 0 indicate that the two images under consideration are not linearly correlated.

Finally, apart from the aforementioned metrics, for each of the trained models presented below, we reported the time for its training and testing/inference as metrics of its computational complexity. The former one refers to the time required for optimizing the model's weights/parameters throughout the training process, while the latter one quantifies the time required for the model to operate in "production mode" (i.e., derive predictions) several times once the training process has concluded.

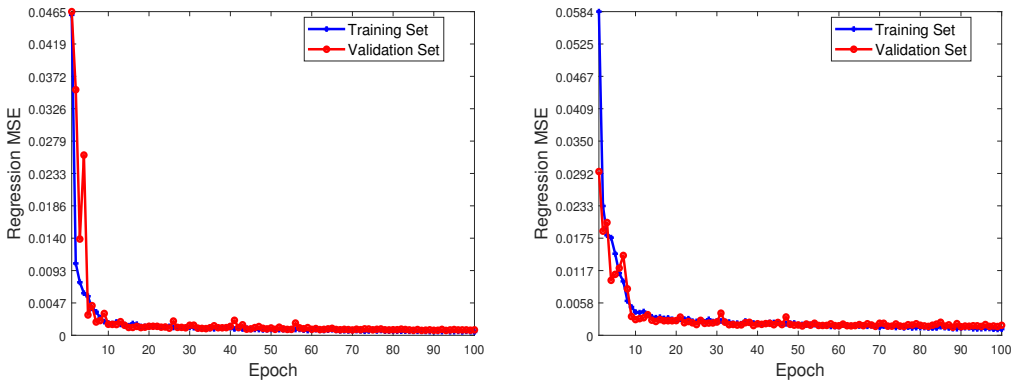
3.4. 4D-ETCN Architecture Ablation Study

To obtain a better understanding of different model choices, we explored the impact of different hyperparameters of the proposed 4D-ETCN model in terms of performance. More specifically, we focused on different hyperparameters, including the size and number of filters on 4D convolutional layers for the encoder and the decoder, as well as for the 1D kernels of the TCN. For that cause, we set as a basic architecture one with a filter size at the encoder and decoder equal to $(2, 2, 2, 2)$, a dropout rate equal to 0.3, and the size of the 1D kernel of TCN was set to 2. With the goal of pinpointing the optimal hyperparameter setup, we undertook a comprehensive ablation study. This involved scrutinizing each hyperparameter in isolation, as well as in various combinations, to guide our selection of the most effective model. In the ensuing experiments, we report the results from the two examined datasets, namely the Crete, Greece and Italy ROIs. Our primary focus was on the model's MSE loss metric, thus serving as the cornerstone for our ultimate choice of model.

Due to space limitations, the analytical experimental results of this ablation study are provided in Appendix C. As can be seen from the result tables therein, the two datasets requires quite different hyperparameter configurations for achieving optimal results. In general though, adopting architectures with more filters in the encoder part of the network leads to clear performance gains. In addition, wider receptive fields both at the encoder and the decoder parts of the architecture contributed positively to its performance, irrespective of the dataset at hand.

3.5. Convergence Analysis

Figure 6 showcases the convergence trajectory of our proposed 4D-ETCN architecture for both examined datasets. The convergence is illustrated primarily through the regression loss (i.e., MSE) plotted against the number of epochs, thus encompassing both training and validation sets. It is crucial to note that we employed the finest hyperparameters for the suggested 4D models, which were derived from the corresponding tables furnished in Appendix C. We discerned a progressive enhancement in the performance of the 4D-ETCN model coupled with an effective mitigation of overfitting. This was evidenced by the validation curve closely mirroring the training curve, thus signifying robust generalization potential. Another interesting remark is that this behavior was observed for both datasets, thus indicating that the good generalization capabiblity of the proposed approach was dataset-independent.



(**a**) Regression MSE—Crete. (**b**) Regression MSE—Italy.

Figure 6. Regression MSE of the best-proposed 4D-ETCN models for Crete and Italy datasets. As the number of epochs increases, the model's performance improves, thus avoiding overfitting for both datasets.

3.6. Evaluation against State-of-the-Art Methods

Beyond refining the proposed 4D-ETCN architecture delineated in Section 3.4, we undertook a comparative analysis between our advanced high-dimensional model and prevailing cutting-edge methods in the discipline. To ensure an equitable comparison, we selected the relatively novel method advanced by Villia et al. in [9] (previously discussed in Sections 1–2.2.1), thus serving as the foundational architecture that our proposition enhances. More specifically, the authors introduced the concept of ETCN for the problem of forecasting ECVs, thus coping primarily with land surface temperature and soil moisture. In that attempt, they performed an analogous ablation study and derived the best-performing model for the task, which makes use of 2D convolutional kernels both in the encoder and decoder parts. Since on their experimental scenarios were not contained within a temporal window equal to 6 months, we therein considered their best-performing model for 12 months, which mainly consisted of 2D filters of size $(4, 4)$, the number of filters in the encoder part equal to 32–64–64, a dropout rate set to 0.3, and the size of the 1D kernel at the TCN part of three. In addition, since in [9], the input of the time series was imagery of spatial size 28×28 rather than 32×32 in our case, we modified accordingly the number of filters in the decoder part to be 64 rather in 49 and all convolutions to "same" padding mode to match the current experimental setup. Moreover, both the encoder and the decoder parts of the network in [9] were wrapped around by the Time-Distributed Tensorflow's layer in order to fully exploit the temporal information available in the data. However, in our setup, the ETCN architecture proposed in [9] forecasts all four ECVs jointly at once and not separately in their vanilla setup.

Apart from the aforementioned state-of-the-art method described in [9], we additionally compared our proposed high-dimensional method with a 2D Conv-LSTM [19] architecture also proposed in [9] as a baseline. It consists of two layers with 64 filters per layer and a kernel size equal to $(3, 3)$ for both layers. Its output layer is a 3D convolutional layer of kernel size $(3, 3, 3)$, which maps to the desired output.

Table 3 reports the obtained results in the test set for the proposed model architecture as emerged from the ablation study of Section 3.4, as well as its state-of-the-art competing method described above in both under examination datasets. We observe that the proposed high-dimensional model outperformed its competitors in nearly all reported metrics, both in the Crete and in Italy dataset. Moreover, the ETCN model [9] outperformed the Conv-LSTM [9] one in the Crete dataset, while the two models had comparable performance in the Italy dataset. In addition, all methods performed better in the Crete dataset than in the Italy one, but in this toughest case, the proposed 4D-ETCN model widened even further the performance gap from its competitors. Moreover, concerning the unbiased RMSE and MAE measured in nominal Kelvin values (i.e., by converting the predictions

back to their prenormalization values) metrics, we observe that once more our proposed higher-order model outperformed its competitors in nearly all cases. Interpreted from a climatological context point of view, our method's predictions were on average ≈0.37 and ≈0.84 Kelvins away from the ground truth values in the two examined datasets, thus indicating its enhanced performance.

Table 3. The proposed higher-dimensional 4D-ETCN model surpasses its counterparts Conv-LSTM and ETCN, both cited in [9] across several evaluated metrics, including reconstruction error, image fidelity, and correlation.

Dataset	Model	MSE	MAE	SSIM	PSNR	PLCC	ubRMSE	MAE (K)
Crete	Conv-LSTM [9]	0.00154561	0.0201477	0.928759	28.2473	0.991675	0.0369692	0.380813
	ETCN [9]	0.00115819	0.018624	0.952923	29.4991	0.9939	0.0326225	**0.348874**
	4D-ETCN-Proposed	**0.00110941**	**0.0179513**	**0.959973**	**30.7597**	**0.994405**	**0.0298117**	0.377492
Italy	Conv-LSTM [9]	0.00287025	0.0382445	0.840449	25.5313	0.977507	0.0861671	1.05962
	ETCN [9]	0.00436746	0.0527445	0.882363	23.7782	0.987332	0.0442294	0.940987
	4D-ETCN-Proposed	**0.00181847**	**0.031803**	**0.932413**	**29.1255**	**0.990558**	**0.0329555**	**0.848434**

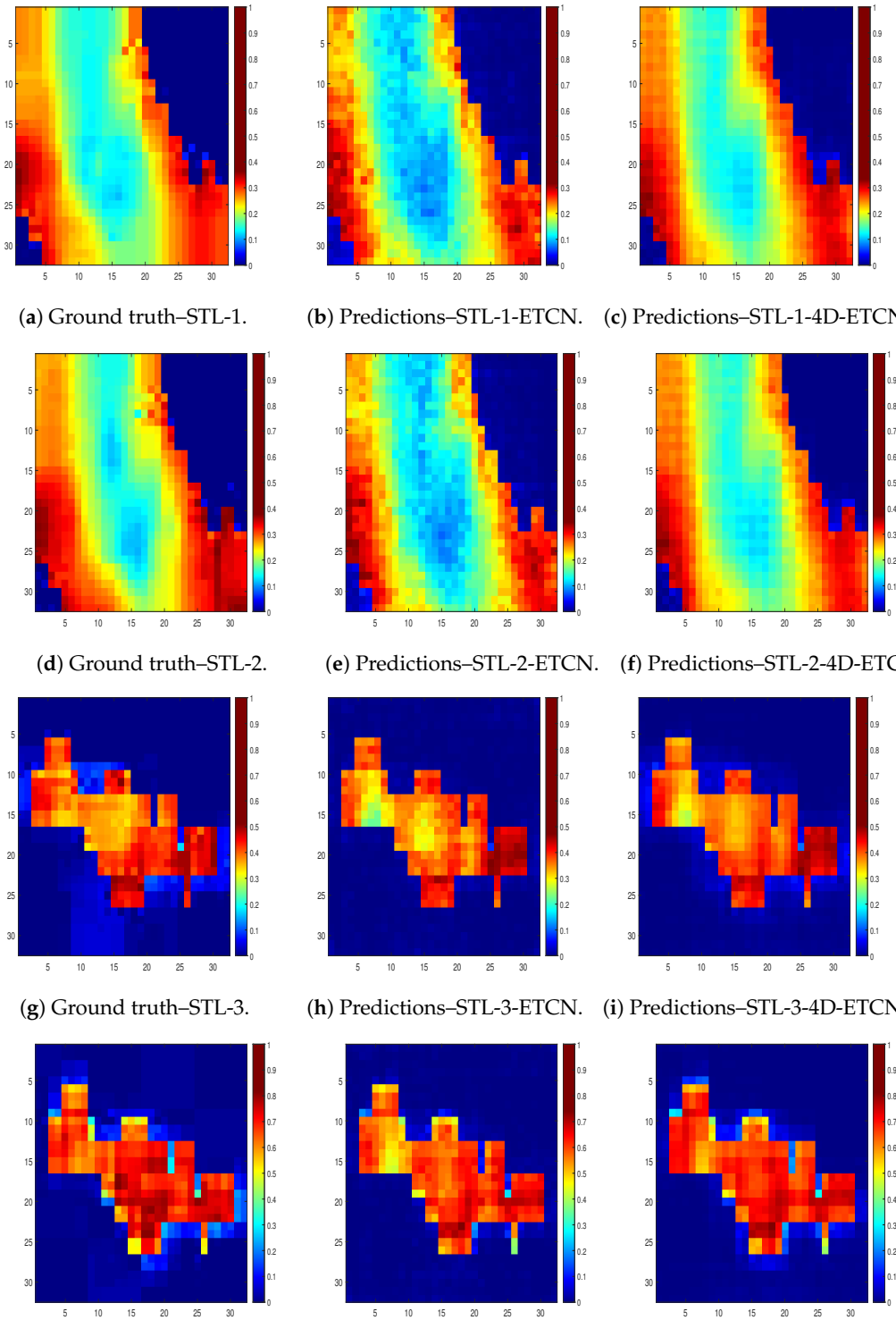
Of course, higher-dimensional models and operations in the respective spaces require more parameters to be learned, thus leading to an increase in the time needed to train the network, as dictated by the reported results in Table 4. Nevertheless, the required training times of ≈2 and ≈3 h for the Crete and Italy datasets, respectively, are surely not prohibitive, especially if more powerful hardware than the employed one is available. The most important thing to note here is that although the training times may vary among the involved models, the respective test/inference ones are nearly identical. This observation is of particular interest, thus taking into account that once the models are trained (a process which is performed only once), they can be employed in "production mode" with nearly the same minimal cost. Consequently, our superior higher-order model can derive enhanced predictions as fast as its competitors in a matter of few seconds.

Table 4. The proposed higher-dimensional 4D-ETCN model's superior performance comes at the cost of increased computational demands in terms of training, while its inference runtime for deriving predictions is negligible.

Dataset	Model	# Parameters	Time Training (Minutes)	Time Testing/Inference (Seconds)
Crete	Conv-LSTM [9]	459,012	17.6605	**0.5**
	ETCN [9]	**336,874**	**3.30542**	**0.5**
	4D-ETCN-Proposed	34,599,367	116.909	1
Italy	Conv-LSTM [9]	459,012	17.9783	**0.5**
	ETCN [9]	**336,874**	**3.44944**	**0.5**
	4D-ETCN-Proposed	47,104,583	181.733	1

To provide a visual depiction of the proposed model's efficacy, Figure 7 presents the actual values of the four distinct ECVs being studied (namely, soil temperature levels 1 through 4) for the specific test sample month of November 2020. This was juxtaposed with predictions derived from our advanced 4D model and the runner-up method, the ETCN [9]. The regression maps featured in Figure 7 indicate that our 4D-ETCN model more accurately discerned the majority of the true values across both datasets, thus resulting in distinctly delineated regions. It is also evident that the ETCN [9] model exhibited substantial "noise effects," particularly within the Italian dataset (i.e., Figure 7a–e), due to its inability to accurately forecast large segments of each test sample. Conversely, the proposed 4D-ETCN

model excelled in mirroring most of the values present in the actual regression maps thanks to its inherent capability to grasp high-dimensional interrelations among the variables, thereby closely reflecting the ground truth maps.



(**a**) Ground truth–STL-1. (**b**) Predictions–STL-1-ETCN. (**c**) Predictions–STL-1-4D-ETCN.

(**d**) Ground truth–STL-2. (**e**) Predictions–STL-2-ETCN. (**f**) Predictions–STL-2-4D-ETCN.

(**g**) Ground truth–STL-3. (**h**) Predictions–STL-3-ETCN. (**i**) Predictions–STL-3-4D-ETCN.

(**j**) Ground truth–STL-4. (**k**) Predictions–STL-4-ETCN. (**l**) Predictions–STL-4-4D-ETCN.

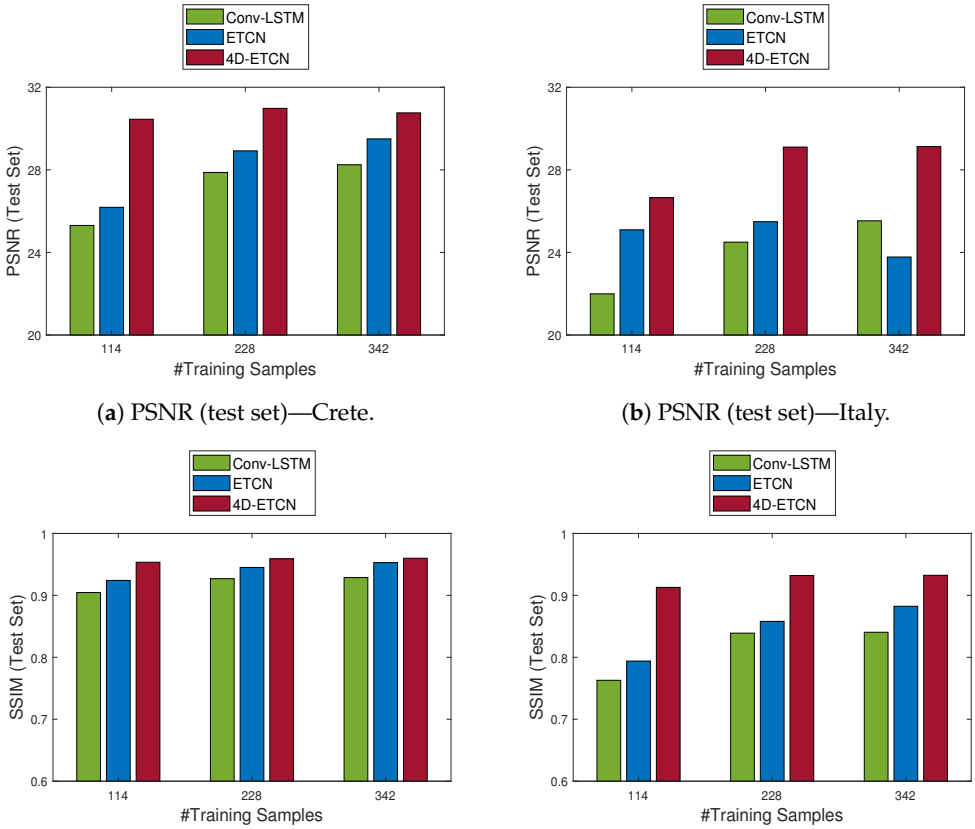
Figure 7. The regression maps for November 2020 showcasing the ground truth and predictions for the soil temperature at two distinct depths have been generated using the ETCN and 4D-ETCN models for Italy (**a**–**f**) and Crete (**g**–**l**) datasets.

4. Discussion

Within this section, we delve into the outcomes detailed in Section 3. Our discussion unfolds systematically, thus exploring the models' performance in relation to the extent of accessible training data and aiming to quantify the specific requirements of each model for accurate ECVs' forecasting. Furthermore, an examination is conducted on the efficacy of the proposed higher-order ETCN alongside its competitors, particularly in scenarios with increased temporal information. This analysis aims to illuminate the respective strengths and limitations inherent in each examined model, thus ensuring a comprehensive and lucid comparison.

4.1. Impact of Training Set Size on Forecasting Accuracy

To assess the efficiency of the proposed 4D-ETCN model concerning its dependency on the volume of training data available, we conducted tests using just 66% and 33% of the original training samples, thereby decreasing them from 342 to 228 and 114, respectively. The number of validation and test samples remained unchanged (114 and 17, respectively). We then proceeded to train the optimally configured models, as identified in Section 3.4, using the reduced training dataset sizes for both scenarios and datasets, alongside their leading contemporary counterparts. The outcomes are consolidated in Figure 8 for each dataset under scrutiny.



(**a**) PSNR (test set)—Crete. (**b**) PSNR (test set)—Italy.

(**c**) SSIM (test set)—Crete. (**d**) SSIM (test set)—Italy.

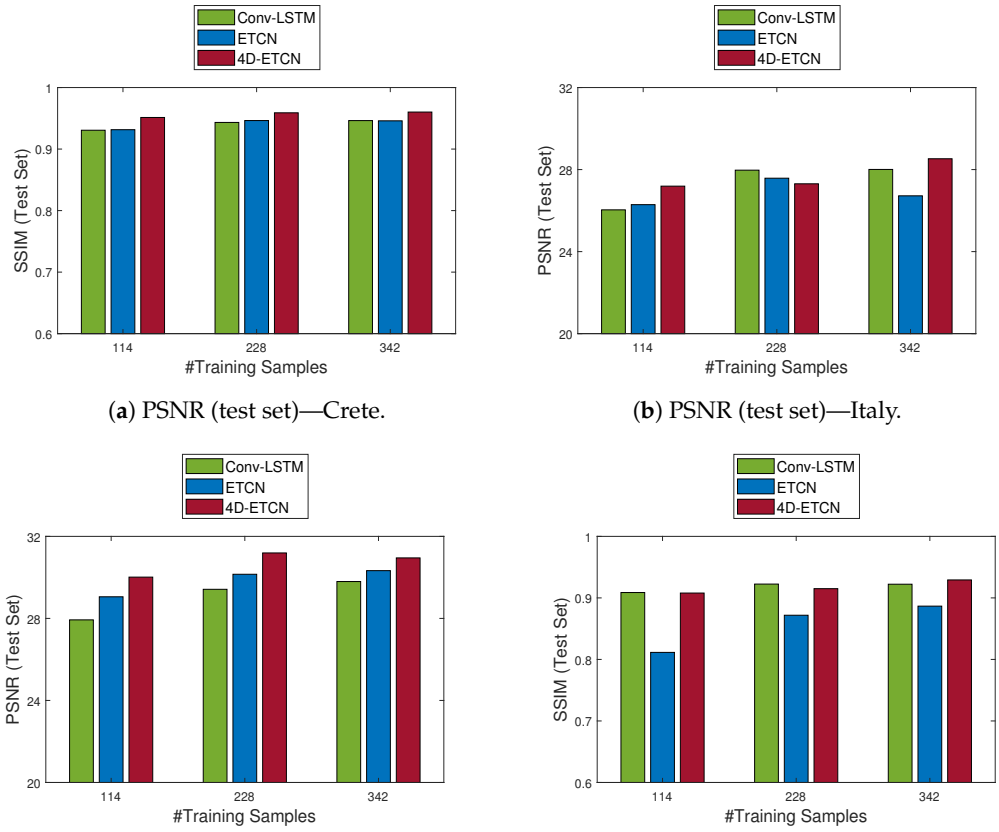
Figure 8. Regression metrics relative to the number of training samples indicate that while increasing the number of samples enhances the performance of comparative models, the proposed 4D architecture prevails in almost all instances.

We noted that the decrement in training samples marginally impacted the efficacy of the proposed 4D-ETCN model, which managed to maintain similar performance levels even when furnished with merely 33% of the total training samples. Conversely, the performance of the Conv-LSTM [9] and ETCN [9] models was noticeably enhanced across all visual perception metrics when provided with a larger pool of training samples. Overall,

the 4D-ETCN model consistently surpassed its rivals across all tested scenarios and datasets, thus underscoring the resilience and reliability of the proposed methodology.

4.2. Effect of Temporal Information on Forecasting Accuracy

In a final set of experiments, we aimed to quantify the effect of the temporal information on the performance of the forecasting models. Throughout the present study, we employed samples containing information at a half-year level (i.e., the temporal size of the data was equal to 6 months). We doubled this temporal window information to 1 year (i.e., the temporal data was now set to 12 months) in an attempt to explore how the models adapted to such an experimental scenario, while the training and validation sets remained the same (in terms of selected samples). On the contrary, as the test set was by assumption/construction focused on the last 2 available years 2019–2020, the number of test samples was reduced from 17 to 11 as a direct consequence of the whole year 2019 being used for forecasting the values of year 2020. With this setup, we performed exactly the same analysis that led to Section 4.1 and quantified the performance of all models in both under investigation dataset for all different numbers of training samples being available to them. The results are depicted in Figure 9, where we observe that once more the proposed 4D architecture outperformed its competing ones in nearly every case. We should notice though that although in the Crete dataset its dominance was clear in every experimental scenario, in the Italy dataset, when 66% of the whole training samples were available, the Conv-LSTM [9] model seemed to question its supremacy. However, its performance throughout all these examined cases indicates that it clearly consists of the best alternative among its competitors.



(**a**) PSNR (test set)—Crete.

(**b**) PSNR (test set)—Italy.

(**c**) SSIM (test set)—Crete.

(**d**) SSIM (test set)—Italy.

Figure 9. The regression metrics relative to the number of training samples for the proposed 4D-ETCN model and its counterparts under scenarios where an entire year is used for forecasting. The results reveal a notable trend where while augmenting the number of training samples enhances the performance of state-of-the-art models, this improvement does not suffice to surpass the performance of the proposed 4D model.

5. Conclusions

In this study, we introduced a novel DL architecture, the 4D-ETCN, tailored for simultaneous forecasting of multiple ECVs. This methodology expands on the foundations of cutting-edge ETCN models, thus transitioning them into their high-dimensional counterparts by adapting every aspect of their functionalities across all paths. Furthermore, we embarked on the task of creating, experimenting, and assessing diverse network configurations using actual data to pinpoint the most effective structure for the given challenge.

Our empirical investigations, conducted on two meticulously curated datasets, have substantiated that superior forecasting accuracy, relative to traditional and contemporary strategies, can be achieved by executing feature learning within the data's inherent domain. At the same time, its computational cost for processing data in higher dimensions is quite small during inference stage once the model is trained, while there is also room for further improvement both in terms of hardware and implementation aspects, as dictated in the respective sections. This advancement underscores the potential of our proposed model in enhancing the precision of ECVs' predictions.

Supplementary Materials: The Python scripts for constructing and previewing the proposed 4D-ETCN model will be available online at https://github.com/TITAN-Project-EU/Higher-Order-CNN (accessed on 29 May 2024), as part of the TITAN ERA Chair project which funded the present work.

Author Contributions: Conceptualization, M.G., G.T. and P.T.; methodology, M.G., G.T. and P.T.; formal analysis, M.G., G.T. and P.T.; writing—original draft preparation, M.G. and G.T.; writing—review and editing, P.T.; visualization, M.G. and G.T.; supervision, P.T.; project administration, P.T.; funding acquisition, P.T. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the TITAN ERA Chair project (contract no. 101086741) within the Horizon Europe Framework Program of the European Commission.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ECV	Essential Climate Variable
GCOS	Global Climate Observing System
ML	Machine Learning
DL	Deep Learning
RS	Remote Sensing
ETCN	Embedded Temporal Convolutional Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
TCN	Temporal Convolutional Network
GRU	Gated Recurrent Unit
CDS	Climate Data Store
ROI	Region Of Interest
MSE	Mean Square Error
MAE	Mean Absolute Error
RMSE	Root Mean Square Error
ubRMSE	Unbiased Root Mean Square Error
PSNR	Peak Signal-to-Noise Ratio
SSIM	Structural Similarity Index Measure
PLCC	Pearson Linear Correlation Coefficient

Appendix A. Forward Propagation in ND CNNs

Appendix A.1. Tucker Convolution Model Sanity Checks

In order to check the validity of our computations, we simulated two uniformly distributed random tensors $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$ of order $N = 1, 2, 3, 4$ and computed their convolution $\boldsymbol{\mathcal{Z}}$ via three different "convolution-models":

(i) Direct: The convolution is determined directly from sums, which are the definition of convolution.
(ii) Fourier: The (Fast) Fourier Transform (FFT) is used to perform the convolution.
(iii) Tucker: The convolution is computed via Tucker decomposition, as explained in the respective section of the paper.

The size of each input tensor across each of its modes is chosen uniformly at random to be in the interval $[2, 50]$ in order to be able to generate simulations that can fit into memory for all different tensor orders. Furthermore, the sizes of the core tensors $\boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{X}}}$ and $\boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{Y}}}$ across each of their modes are also selected uniformly at random to be in the intervals $[2, min(size(\boldsymbol{\mathcal{X}}))]$ and $[2, min(size(\boldsymbol{\mathcal{Y}}))]$, respectively. The bounds of these intervals are determined in such a way as follows:

- Perform N-D convolution at each simulation (i.e., by avoiding a selection of size equal to one across a specific mode, which indicates a trailing singleton dimension).
- Perform the maximum compression possible via $\boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{X}}}$ and $\boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{Y}}}$ (i.e., since in the Tucker decomposition format the sizes of $\boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{X}}}$ and $\boldsymbol{\mathcal{G}}_{\boldsymbol{\mathcal{Y}}}$ have to be at most equal to these of $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{Y}}$).

Moreover, the size of tensor $\boldsymbol{\mathcal{Y}}$ across each of its modes is at most equal to the respective one of $\boldsymbol{\mathcal{X}}$, with the intention of producing sensible convolution outputs at each subsection/padding scenario (e.g., full, same, valid).

With the above experimental setup, we performed 10 Monte Carlo simulations with the intention of measuring the discrepancy between the outputs of the three convolution models, as well as the time needed for the respective computations. The gap between the convolution models was quantified via their Normalized Mean Square Error (NMSE), which is defined as follows:

$$NMSE = \frac{\|\hat{\boldsymbol{\mathcal{Z}}} - \boldsymbol{\mathcal{Z}}\|_F}{\|\boldsymbol{\mathcal{Z}}\|_F} \tag{A1}$$

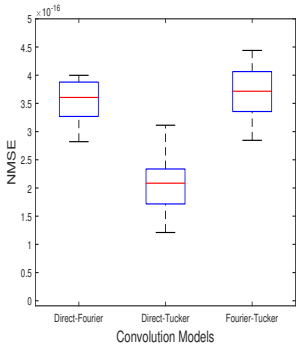
where $\boldsymbol{\mathcal{Z}}$ is the convolution output used each time as ground truth model (i.e., Direct and FFT), $\hat{\boldsymbol{\mathcal{Z}}}$ is the convolution output used each time as the approximation model (i.e., FFT and Tucker), and $\|\|_F$ stands for the Frobenius norm.

Figure A1 depicts the boxplot of the NMSE between the Tucker convolution model and the direct and Fourier ones for up to 4th order tensors and all available convolution subsections/paddings. On each box, the central mark indicates the median, and the bottom and top edges of the box indicate the 25th and 75th percentiles, respectively. The whiskers extend to the most extreme data points not considered outliers, and the outliers are plotted individually using the '+' marker symbol. By observing Figure A1, we can draw the following conclusions:

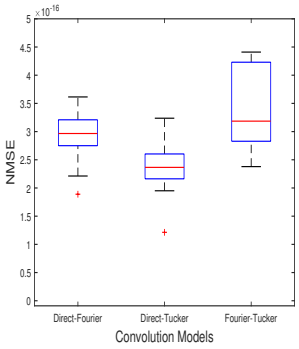
(i) The bottom and top of each box are the 25th and 75th percentiles of the sample, respectively. The distance between the bottom and top of each box is the interquartile range, which in the 1D and 2D cases is wider for the Fourier–Tucker models' comparison. On the contrary, for the 3D and 4D cases it is clearly narrower.
(ii) The red line in the middle of each box is the sample median. Even in the cases where the median is not centered in the box (i.e., the plot shows sample skewness), all median NMSE values are of order 10^{-16}, thus indicating the equivalence of computations.
(iii) The whiskers are lines extending above and below each box. Whiskers go from the end of the interquartile range to the furthest observation within the whisker length (the adjacent value). We observe that the whiskers for the Fourier–Tucker models' comparison are shorter in nearly every case, thus indicating tighter error distributions.

(iv) Observations beyond the whisker length are marked as outliers. By default, an outlier is a value that is more than 1.5 times the interquartile range away from the bottom or top of the box. Once more, we highlight the fact that the Fourier–Tucker models' comparison is the one with the least number of outliers, which are more to be found in the 4D convolution case.

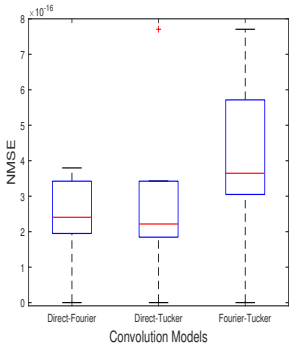
(v) In all cases, the relative approximation error is of order 10^{-16}, which indicates the validity of the convolution generalization via tensor decompositions.



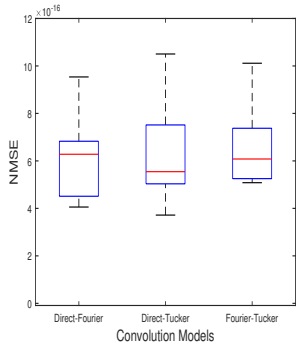
(**a**) Convolution—1D–Full.



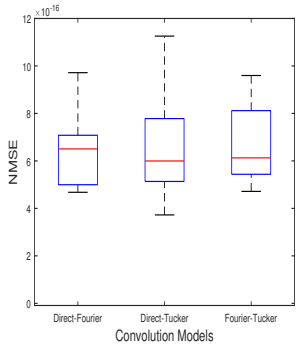
(**b**) Convolution—1D–Same.



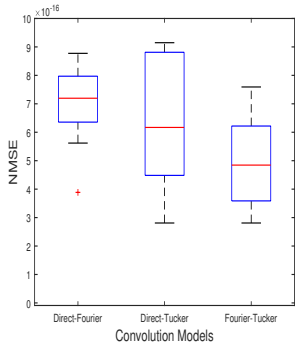
(**c**) Convolution—1D–Valid.



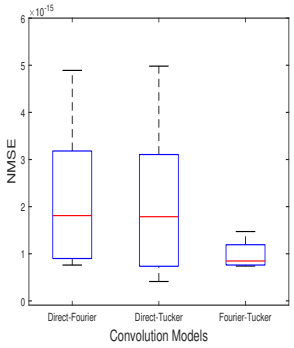
(**d**) Convolution—2D–Full.



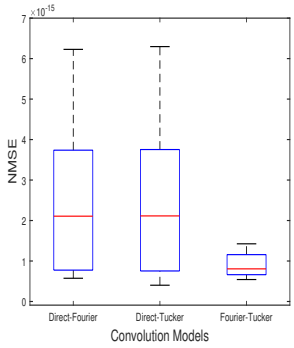
(**e**) Convolution—2D–Same.



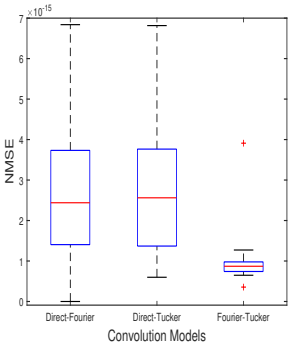
(**f**) Convolution—2D–Valid.



(**g**) Convolution—3D–Full.

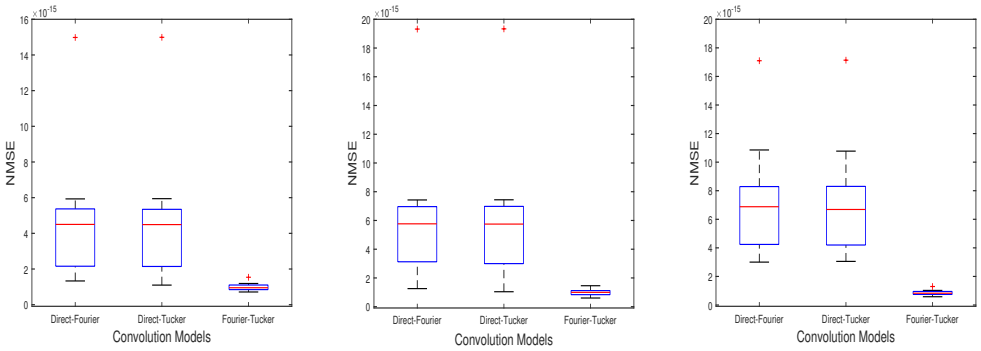


(**h**) Convolution—3D–Same.



(**i**) Convolution—3D–Valid.

Figure A1. *Cont.*



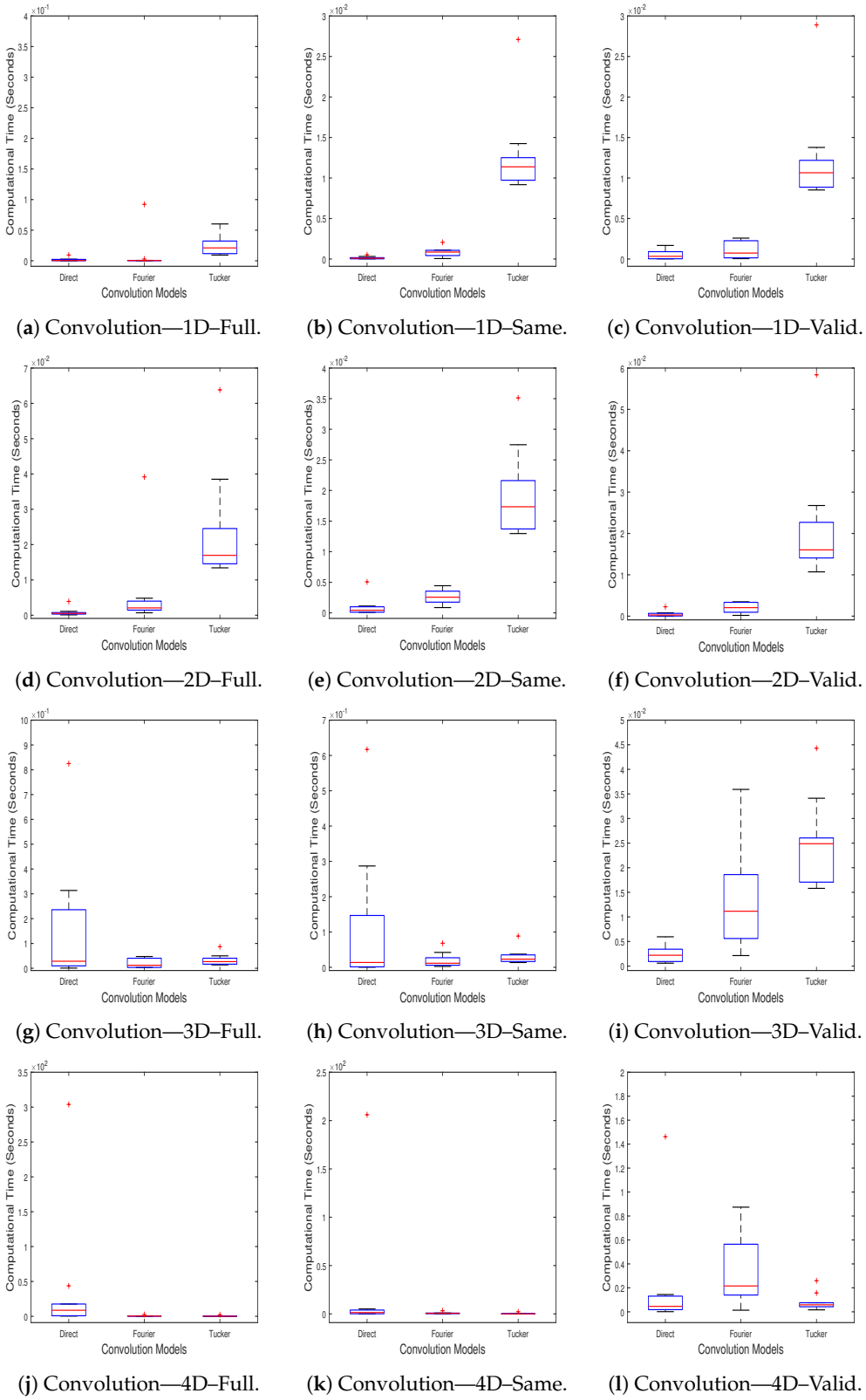
(**j**) Convolution—4D–Full. (**k**) Convolution—4D–Same. (**l**) Convolution—4D–Valid.

Figure A1. NMSE between the Tucker convolution model and the direct and Fourier ones for up to 4D data and all available convolution subsections/paddings. In all cases, the relative approximation error is of order 10^{-16}, thus indicating the validity of the convolution generalization via tensor decompositions.

In Figure A2 is depicted the boxplot of the computational times (in seconds) required by each convolution model to derive the desired output for up to 4th order tensors and all available convolution subsections/paddings. Based on the boxplot's interpretation explained earlier, we can highlight the following remarks:

(i) The interquartile range of the Tucker models is wider in the 1D and 2D cases and narrower for the 3D and 4D cases.
(ii) The median value of the Tucker model is slightly higher than these of the direct and Fourier ones in the 1D and 2D cases, whereas it is nearly the same or even lower in the 3D and 4D cases.
(iii) The whiskers for the Tucker model are shorter in the 1D and 2D cases and wider in the 3D and 4D cases.
(iv) The Tucker model seems quite robust to outliers in the 4D convolution case, which indicates faster computations in cases where the size across a specific mode of the convolution tensor is high.
(v) In all cases, the computational time required by each convolution model is low, except the 4D case, where the proposed convolution generalization via tensor decompositions scales better than its competitors.

Moreover, in Table A1, we report the mean computational times (in seconds) required by the Tucker convolution model and the direct and Fourier ones corresponding to the box plots shown in Figure A1. From the highlighted values, we notice that for up to three-dimensional data, the direct and Fourier convolution models slightly outperformed the Tucker one, but these differences in the mean values are nearly indifferent. On the contrary, when dealing with four-dimensional data, the Tucker convolution model clearly outperformed its competitors.



(**a**) Convolution—1D–Full. (**b**) Convolution—1D–Same. (**c**) Convolution—1D–Valid.

(**d**) Convolution—2D–Full. (**e**) Convolution—2D–Same. (**f**) Convolution—2D–Valid.

(**g**) Convolution—3D–Full. (**h**) Convolution—3D–Same. (**i**) Convolution—3D–Valid.

(**j**) Convolution—4D–Full. (**k**) Convolution—4D–Same. (**l**) Convolution—4D–Valid.

Figure A2. Computational time (in seconds) required by the Tucker convolution model and the direct and Fourier ones for up to 4D data and all available convolution subsections/paddings. The Tucker convolution model scales better than its competing ones in higher-dimensional cases.

Table A1. Mean computational time (in seconds) required by the Tucker convolution model and the direct and Fourier ones for up to 4D data and all available convolution subsections/paddings. For higher-dimensional data, the Tucker convolution model clearly scales better than its competitors.

Padding	Order	Direct	Fourier	Tucker
Full	1	**0.00164299**	0.00983335	0.06190979
	2	**0.00084248**	0.0059659	0.02388292
	3	0.16617763	**0.01885796**	0.03345958
	4	40.3023198	0.58974021	**0.2882506**
Same	1	**0.0001432**	0.00084148	0.01272597
	2	**0.00091602**	0.00261802	0.01915561
	3	0.12005669	**0.02093711**	0.02982094
	4	22.10111094	0.78723203	**0.27993217**
Valid	1	**0.00049343**	0.00107373	0.01228869
	2	**0.00052135**	0.00206695	0.02092461
	3	**0.00251685**	0.01359532	0.02471507
	4	0.19681736	0.32595058	**0.08041625**

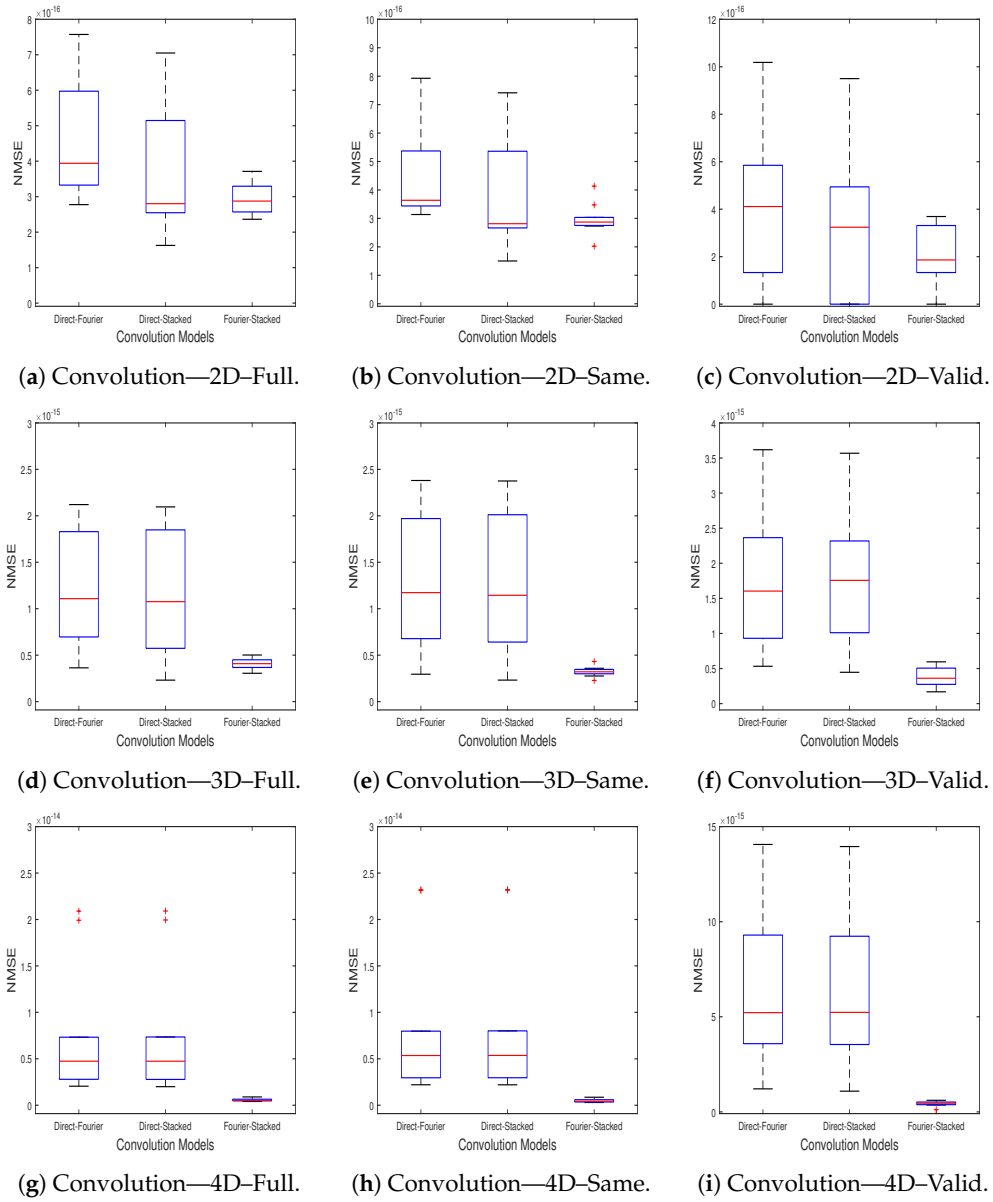
Based on the aforementioned analysis, the forward pass of the N-D CNNs can be computed via the Tucker convolution model, which on the one hand performs the correct computations, while at the same time scales quite well as the dimensionality of the problem increases.

Appendix A.2. Stacked Convolution Model Sanity Checks

Following the reasoning adopted for the Tucker convolution model, we accordingly performed 10 Monte Carlo simulations with the same assumptions, thus aiming to measure the gap between the proposed Stacked convolution model and the direct and Fourier ones. Once more, the discrepancy between them is quantified via their NMSE defined in Equation (A1).

Similarly to Figure A1, in Figure A3 is depicted the box plot of the NMSE between the proposed Stacked convolution model and the direct and Fourier ones for up to fourth order tensors and all available convolution subsections/paddings. By observing Figure A3, we can draw the following conclusions:

(i) The interquartile range is clearly narrower for the Fourier–Stacked models' comparison in every dimensionality and padding case.
(ii) All median NMSE values are of order 10^{-16}, thus indicating the equivalence of computations.
(iii) The whiskers for the Fourier–Stacked models' comparison are generally shorter, thus indicating tighter error distributions.
(iv) In nearly every case there are no outlier values, except the 4D one where once more the Fourier–Stacked models' comparison is the one with the least number of outliers.
(v) In all cases, the relative approximation error is of order 10^{-16}, which indicates the validity of the proposed convolution generalization via stacking lower-dimensional convolutions.



(**a**) Convolution—2D–Full. (**b**) Convolution—2D–Same. (**c**) Convolution—2D–Valid.

(**d**) Convolution—3D–Full. (**e**) Convolution—3D–Same. (**f**) Convolution—3D–Valid.

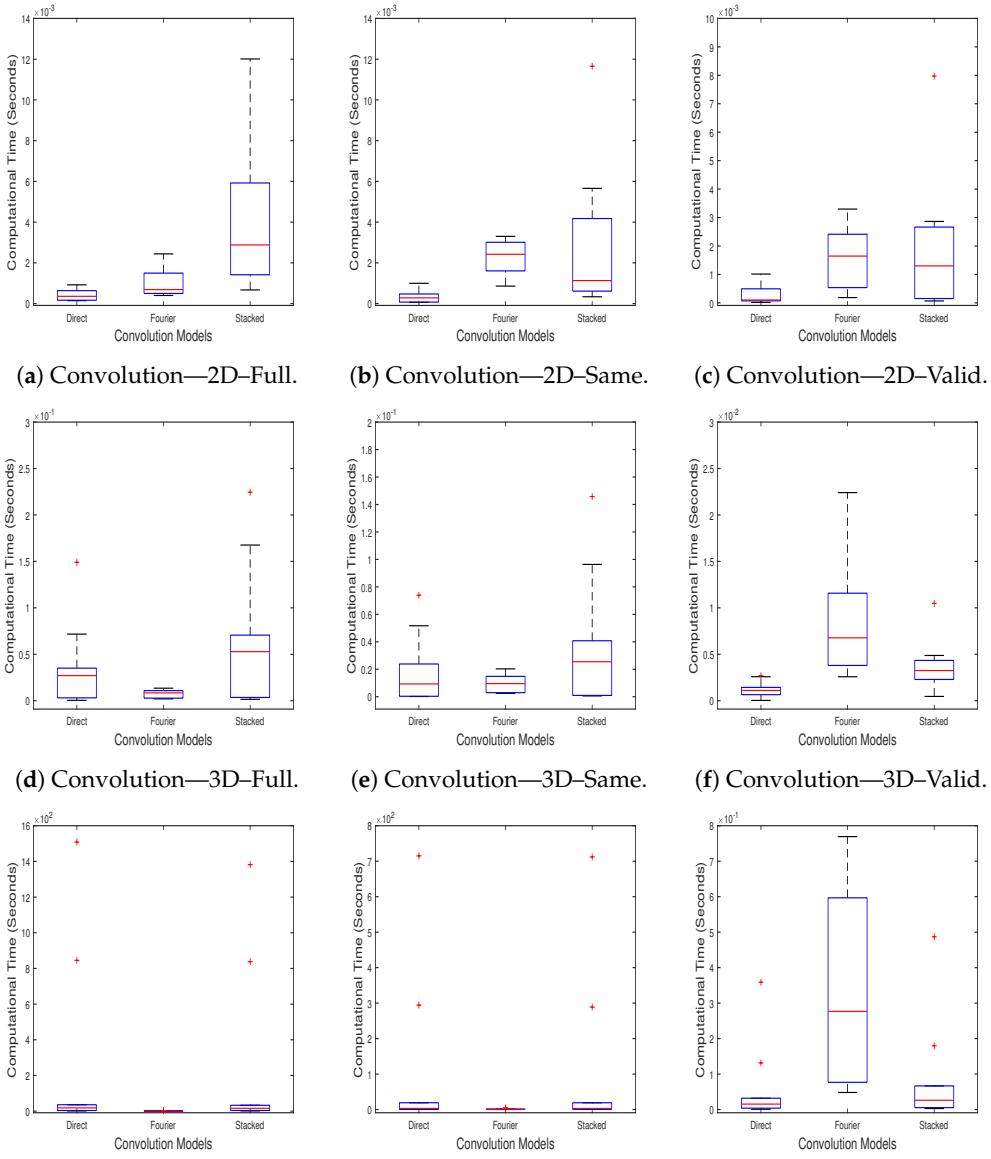
(**g**) Convolution—4D–Full. (**h**) Convolution—4D–Same. (**i**) Convolution—4D–Valid.

Figure A3. NMSE between the proposed Stacked convolution model and the direct and Fourier ones for up to 4D data and all available convolution paddings/subsections. In all cases, the relative approximation error is of order 10^{-16}, thus indicating the validity of the proposed convolution generalization via stacking lower-dimensional convolutions.

Similarly to Figure A2, in Figure A4 is depicted the box plot of the computational times (in seconds) required by each convolution model to derive the desired output for up to fourth order tensors and all available convolution subsections/paddings. Based on the box plot's interpretation explained earlier, we can highlight the following remarks:

(i) The interquartile range of the Stacked model is wider in the 2D and 3D cases and nearly equally narrow to its competitors for the 4D case.
(ii) The median value of the Stacked model is slightly higher than these of the direct and Fourier ones in the 2D and 3D cases, whereas it is nearly the same in the 4D case.
(iii) The whiskers for the Stacked model are greater in every convolution cases except the "valid" ones, where the Fourier ones are more significant.
(iv) The Stacked model seems as robust to outliers as the direct one, with both being slightly inferior to the Fourier convolution model.

(v) In all cases, the computational time required by each convolution model is low, thus indicating the speed and efficiency of the performed computations.



(**a**) Convolution—2D–Full. (**b**) Convolution—2D–Same. (**c**) Convolution—2D–Valid.

(**d**) Convolution—3D–Full. (**e**) Convolution—3D–Same. (**f**) Convolution—3D–Valid.

(**g**) Convolution—4D–Full. (**h**) Convolution—4D–Same. (**i**) Convolution—4D–Valid.

Figure A4. Computational time (in seconds) required by the proposed Stacked convolution model and the direct and Fourier ones for up to 4D data and all available convolution paddings/subsections. The proposed Stacked convolution model scales equally well to its competing ones in every case.

Stacked Convolution Algorithm

The Stacked Convolution Algorithm for up 4D tensors is outlines below.

Algorithm A1 Stacked Convolution Algorithm

procedure STACKED-CONVOLUTION($N, A, B, Padding$) ▷ Convolution between N-th order tensors A and B
  $w_a = size(A, N)$ ▷ Stacking dimension of tensor A
  $w_b = size(B, N)$ ▷ Stacking dimension of tensor B
  if $Padding == "Full"$ **then**
    $w_c = w_a + w_b - 1$ ▷ Stacking dimension of tensor C
  else if $Padding == "Same"$ **then**
    $w_c = w_a$ ▷ Stacking dimension of tensor C
  else if $Padding == "Valid"$ **then**
    $w_c = w_a - w_b + 1$ ▷ Stacking dimension of tensor C
  else
    return
  end if
  $F_R = cell(w_c, 1)$ ▷ Output tensors for each (N-1)-D frame
  for $i = 1 : w_b$ **do**
    for $i = 1 : w_a$ **do**
      $O_F = j + (i - floor(w_b/2)) - floor((w_a - w_c)/2) - 1$ ▷ Add results to this output frame
      if $O_F < 1$ OR $O_F \geq w_c + 1$ **then**
        continue
      end if
      if $N == 2$ **then**
        $F_{CNM1D} = conv(A(:, j), B(:, i), Padding)$ ▷ Stacked-convolution frame-results for 2-D tensors
      else if $N == 3$ **then**
        $F_{CNM1D} = conv2(A(:, :, j), B(:, :, i), Padding)$ ▷ Stacked-convolution frame-results for 3-D tensors
      else if $N == 4$ **then**
        $F_{CNM1D} = convn(A(:, :, :, j), B(:, :, :, i), Padding)$ ▷ Stacked-convolution frame-results for 4-D tensors
      else
        return
      end if
      if isempty($F_R\{O_F\}$) **then**
        $F_R\{O_F\} = F_{CNM1D}$
      else
        $F_R\{O_F\} = F_R\{O_F\} + F_{CNM1D}$
      end if
    end for
  end for
  if $Padding == "Full"$ **then**
    $C = zeros(size(A) + size(B) - 1)$
  else if $Padding == "Same"$ **then**
    $C = zeros(size(A))$
  else if $Padding == "Valid"$ **then**
    $C = zeros(size(A) - size(B) + 1)$
  else
    return
  end if
  if $N == 2$ **then**
    $C = horzcat(F_R\{:\})$
  else if $N == 3$ **then**
    for $k = 1 : w_c$ **do**
      $C(:, :, k) = F_R\{k\}$
    end for
  else if $N == 4$ **then**
    for $k = 1 : w_c$ **do**
      $C(:, :, :, k) = F_R\{k\}$
    end for
  else
    return
  end if
  return C ▷ Convolution between N-th order tensors A and B
end procedure

Appendix B. Backpropagation in ND CNNs

As stated earlier in Section 2.1.3, we can express the basic operation of an ND convolutional layer l in the "direct-sum" form as follows:

$$\begin{aligned}\mathcal{Z}_{i_1,i_2,\ldots,i_N}(l) &= \sum_{j_1}\sum_{j_2}\cdots\sum_{j_N}\mathcal{W}_{j_1,j_2,\ldots,j_N}(l)\mathcal{I}_{i_1-j_1,i_2-j_2,\ldots,i_N-j_N}(l-1)+\boldsymbol{b}(l)\\ &= \boldsymbol{\mathcal{W}}(l)\star\mathcal{I}_{i_1,i_2,\ldots,i_N}(l-1)+\boldsymbol{b}(l)\end{aligned} \tag{A2}$$

where $\{j_1, j_2, \ldots, j_N\}$ span the dimensions of the kernel, $\{i_1, i_2, \ldots, i_N\}$ span the dimensions of the input, and $\boldsymbol{b}$ is the bias term. As can be seen from Equation (A2), the input of the previous layer $(l-1)$ serves for the computation of the current output layer (l), and the input of the current layer l can be computed as follows:

$$\mathcal{I}_{i_1,i_2,\ldots,i_N}(l) = f(\mathcal{Z}_{i_1,i_2,\ldots,i_N}(l)) \tag{A3}$$

where f is a selected activation function (e.g., ReLU, tanh).

Based on the above, the gradients with respect to the input (i.e., $\frac{\partial\boldsymbol{\mathcal{L}}}{\partial\boldsymbol{\mathcal{I}}}$) and with respect to the filters (i.e., $\frac{\partial\boldsymbol{\mathcal{L}}}{\partial\boldsymbol{\mathcal{W}}}$) can be computed via the chain rule as follows:

$$\begin{cases}\frac{\partial\boldsymbol{\mathcal{L}}}{\partial\boldsymbol{\mathcal{I}}} = \frac{\partial\boldsymbol{\mathcal{L}}}{\partial\boldsymbol{\mathcal{Z}}} * \frac{\partial\boldsymbol{\mathcal{Z}}}{\partial\boldsymbol{\mathcal{I}}}\\ \frac{\partial\boldsymbol{\mathcal{L}}}{\partial\boldsymbol{\mathcal{W}}} = \frac{\partial\boldsymbol{\mathcal{L}}}{\partial\boldsymbol{\mathcal{Z}}} * \frac{\partial\boldsymbol{\mathcal{Z}}}{\partial\boldsymbol{\mathcal{W}}}\end{cases} \tag{A4}$$

In order to be able to provide Equation (A4) in the closed form, we need to compute $\frac{\partial\boldsymbol{\mathcal{L}}}{\partial\boldsymbol{\mathcal{Z}}}$, $\frac{\partial\boldsymbol{\mathcal{Z}}}{\partial\boldsymbol{\mathcal{I}}}$, and $\frac{\partial\boldsymbol{\mathcal{Z}}}{\partial\boldsymbol{\mathcal{W}}}$. $\frac{\partial\boldsymbol{\mathcal{L}}}{\partial\boldsymbol{\mathcal{Z}}}$ stands for the output error of our CNN with respect to each neuron in the network and is common in both cases, so we initially start with its analytical computation. More precisely, we have

$$\begin{aligned}\frac{\partial\boldsymbol{\mathcal{L}}}{\partial\boldsymbol{\mathcal{Z}}} &= \frac{\partial\mathcal{L}}{\partial\mathcal{Z}_{i_1,i_2,\ldots,i_N}(l)}\\ &= \sum_{u_1}\sum_{u_2}\cdots\sum_{u_N}\frac{\partial\mathcal{L}}{\partial\mathcal{Z}_{u_1,u_2,\ldots,u_N}(l+1)}\frac{\partial\mathcal{Z}_{u_1,u_2,\ldots,u_N}(l+1)}{\partial\mathcal{Z}_{i_1,i_2,\ldots,i_N}(l)}\\ &= \delta_{i_1,i_2,\ldots,i_N}(l)\end{aligned} \tag{A5}$$

where $\{u_1, u_2, \ldots, u_N\}$ are any N summation variables resulting by applying the chain rule over the range of possible $\boldsymbol{\mathcal{Z}}$ values. By using Equations (A2)–(A3)–(A5), we obtain the following:

$$\frac{\partial\boldsymbol{\mathcal{L}}}{\partial\boldsymbol{\mathcal{Z}}} = \sum_{u_1}\sum_{u_2}\cdots\sum_{u_N}\delta_{i_1,i_2,\ldots,i_N}(l+1)\frac{\partial\left[\sum_{j_1}\sum_{j_2}\cdots\sum_{j_N}\mathcal{W}_{j_1,j_2,\ldots,j_N}(l+1)f(\mathcal{Z}_{u_1-j_1,u_2-j_2,\ldots,u_N-j_N}(l))+\boldsymbol{b}(l+1)\right]}{\partial\mathcal{Z}_{i_1,i_2,\ldots,i_N}(l)} \tag{A6}$$

The derivative of the expression inside the brackets is zero unless $u_1 - j_1 = i_1$, $u_2 - j_2 = i_2, \ldots, u_N - j_N = i_N$, thus bearing in mind that in addition the derivative of $\boldsymbol{b}(l+1)$ with respect to $\mathcal{Z}_{i_1,i_2,\ldots,i_N}(l)$ is zero. In the nonzero regime where $u_1 - j_1 = i_1, u_2 - j_2 = i_2, \ldots, u_N - j_N = i_N$, we have $j_1 = u_1 - i_1 =, j_2 = u_2 - i_2, \ldots$, $j_N = u_N - i_N$, and taking the derivative of the expression in brackets, Equation (A6) becomes

$$\frac{\partial\boldsymbol{\mathcal{L}}}{\partial\boldsymbol{\mathcal{Z}}} = \sum_{u_1}\sum_{u_2}\cdots\sum_{u_N}\delta_{i_1,i_2,\ldots,i_N}(l+1)\left[\sum_{u_1-i_1}\sum_{u_2-i_2}\cdots\sum_{u_N-i_N}\mathcal{W}_{u_1-i_1,u_2-i_2,\ldots,u_N-i_N}(l+1)f'(\mathcal{Z}_{i_1,i_2,\ldots,i_N}(l))\right] \tag{A7}$$

The values of $\{i_1, i_2, \ldots, i_N\}$ and $\{u_1, u_2, \ldots, u_N\}$ are specified outside of the terms inside the brackets. Once the values of these variables are fixed, $\{u_1 - i_1, u_2 - i_2, \ldots, u_N - i_N\}$ inside the brackets are simply N constants. Therefore, the N summation evaluates to $\mathcal{W}_{u_1-i_1,u_2-i_2,\ldots,u_N-i_N}(l+1)f'(\mathcal{Z}_{i_1,i_2,\ldots,i_N}(l))$, and Equation (A7) becomes

$$\begin{aligned}\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{Z}}} &= \sum_{u_1}\sum_{u_2}\cdots\sum_{u_N}\delta_{i_1,i_2,\ldots,i_N}(l+1)\mathcal{W}_{u_1-i_1,u_2-i_2,\ldots,u_N-i_N}(l+1)f'(\mathcal{Z}_{i_1,i_2,\ldots,i_N}(l)) \\ &= f'(\mathcal{Z}_{i_1,i_2,\ldots,i_N}(l))\sum_{u_1}\sum_{u_2}\cdots\sum_{u_N}\delta_{i_1,i_2,\ldots,i_N}(l+1)\mathcal{W}_{u_1-i_1,u_2-i_2,\ldots,u_N-i_N}(l+1)\end{aligned} \tag{A8}$$

The N sum expression in the second line of this equation is in the form of a convolution, but the displacements are the negatives of those in Equation (A3). Therefore, we can write Equation (A8) as follows:

$$\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{Z}}} = f'(\mathcal{Z}_{i_1,i_2,\ldots,i_N}(l))\left[\sum_{u_1}\delta_{i_1,i_2,\ldots,i_N}(l+1) \star \mathcal{W}_{-i_1,-i_2,\ldots,-i_N}(l+1)\right] \tag{A9}$$

The negatives in the subscripts indicate that $\boldsymbol{\mathcal{W}}$ is reflected about all of its axes. This is the generalization of rotating a two-dimensional kernel $\boldsymbol{\mathcal{W}}$ by 180°. Using this fact, we finally arrive at an expression for the error at a layer l by writing Equation (A9) equivalently as follows:

$$\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{Z}}} = f'(\mathcal{Z}_{i_1,i_2,\ldots,i_N}(l))\left[\delta_{i_1,i_2,\ldots,i_N}(l+1) \star rot_N(\mathcal{W}_{i_1,i_2,\ldots,i_N}(l+1))\right] \tag{A10}$$

But, the kernels do not depend on $\{i_1, i_2, \ldots, i_N\}$, so we can write this equation as

$$\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{Z}}} = f'(\mathcal{Z}_{i_1,i_2,\ldots,i_N}(l))\left[\delta_{i_1,i_2,\ldots,i_N}(l+1) \star rot_N(\boldsymbol{\mathcal{W}}(l+1))\right] \tag{A11}$$

Having computed $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{Z}}}$, we have to compute the local gradients with respect to the layer's input ($\frac{\partial \boldsymbol{\mathcal{Z}}}{\partial \boldsymbol{\mathcal{I}}}$), the layer's weights ($\frac{\partial \boldsymbol{\mathcal{Z}}}{\partial \boldsymbol{\mathcal{W}}}$), and the layer's biases ($\frac{\partial \boldsymbol{\mathcal{Z}}}{\partial \boldsymbol{b}}$). These local gradients are going to be combined with the previously computed $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{Z}}}$ in order to derive the desired gradients needed (i.e., $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{I}}}$, $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{W}}}$ and $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{b}}$) for the backpropagation algorithm of the neural network.

Concerning the computation of the gradient with respect to the layer's input, $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{I}}}$, we have the following:

$$\begin{aligned}\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{I}}} &= \frac{\partial \mathcal{L}}{\partial \mathcal{I}_{i_1,i_2,\ldots,i_N}} \\ &= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N}\frac{\partial \mathcal{L}}{\partial \mathcal{Z}_{i_1,i_2,\ldots,i_N}(l)}\frac{\partial \mathcal{Z}_{i_1,i_2,\ldots,i_N}(l)}{\partial \mathcal{I}_{i_1,i_2,\ldots,i_N}} \\ &= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N}\delta_{i_1,i_2,\ldots,i_N}(l)\frac{\partial \mathcal{Z}_{i_1,i_2,\ldots,i_N}(l)}{\partial \mathcal{I}_{i_1,i_2,\ldots,i_N}} \\ &= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N}\delta_{i_1,i_2,\ldots,i_N}(l) \\ &\frac{\partial\left[\sum_{j_1}\sum_{j_2}\cdots\sum_{j_N}\mathcal{W}_{j_1,j_2,\ldots,j_N}(l)f(\mathcal{Z}_{i_1-j_1,i_2-j_2,\ldots,i_N-j_N}(l-1)) + \boldsymbol{b}(l)\right]}{\partial \mathcal{I}_{i_1,i_2,\ldots,i_N}}\end{aligned} \tag{A12}$$

Since the convolution operation is commutative, the last line of Equation (A12) can be written as follows:

$$
\begin{aligned}
\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{I}}} &= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N}\delta_{i_1,i_2,\ldots,i_N}(l) \\
&\frac{\partial\left[\sum_{j_1}\sum_{j_2}\cdots\sum_{j_N}\mathcal{W}_{j_1,j_2,\ldots,j_N}(l)f(\mathcal{Z}_{i_1-j_1,i_2-j_2,\ldots,i_N-j_N}(l-1))+\boldsymbol{b}(l)\right]}{\partial \mathcal{I}_{i_1,i_2,\ldots,i_N}} \\
&= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N}\delta_{i_1,i_2,\ldots,i_N}(l) \\
&\frac{\partial\left[\sum_{i_1}\sum_{i_2}\cdots\sum_{i_N}\mathcal{W}_{j_1-i_1,j_2-i_2,\ldots,j_N-i_N}(l)f(\mathcal{Z}_{i_1,i_2,\ldots,i_N}(l-1))+\boldsymbol{b}(l)\right]}{\partial \mathcal{I}_{i_1,i_2,\ldots,i_N}} \\
&= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N}\delta_{i_1,i_2,\ldots,i_N}(l)\mathcal{W}_{j_1-i_1,j_2-i_2,\ldots,j_N-i_N}(l) \\
&= \delta_{j_1,j_2,\ldots,j_N}(l)\star\mathcal{W}_{j_1,j_2,\ldots,j_N}(l) \\
&= conv\left(\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{Z}}},\boldsymbol{\mathcal{W}}(l)\right)
\end{aligned}
\tag{A13}
$$

Following a similar procedure as above for the gradient with respect to the layer's weights, $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{W}}}$, we obtain the following:

$$
\begin{aligned}
\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{W}}} &= \frac{\partial \mathcal{L}}{\partial \mathcal{W}_{j_1,j_2,\ldots,j_N}} \\
&= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N}\frac{\partial \mathcal{L}}{\partial \mathcal{Z}_{i_1,i_2,\ldots,i_N}(l)}\frac{\partial \mathcal{Z}_{i_1,i_2,\ldots,i_N}(l)}{\partial \mathcal{W}_{j_1,j_2,\ldots,j_N}} \\
&= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N}\delta_{i_1,i_2,\ldots,i_N}(l)\frac{\partial \mathcal{Z}_{i_1,i_2,\ldots,i_N}(l)}{\partial \mathcal{W}_{j_1,j_2,\ldots,j_N}} \\
&= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N}\delta_{i_1,i_2,\ldots,i_N}(l) \\
&\frac{\partial\left[\sum_{j_1}\sum_{j_2}\cdots\sum_{j_N}\mathcal{W}_{j_1,j_2,\ldots,j_N}(l)f(\mathcal{Z}_{i_1-j_1,i_2-j_2,\ldots,i_N-j_N}(l-1))+\boldsymbol{b}(l)\right]}{\partial \mathcal{W}_{j_1,j_2,\ldots,j_N}} \\
&= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N}\delta_{i_1,i_2,\ldots,i_N}(l)f(\mathcal{Z}_{i_1-j_1,i_2-j_2,\ldots,i_N-j_N}(l-1)) \\
&= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N}\delta_{i_1,i_2,\ldots,i_N}(l)\mathcal{I}_{i_1-j_1,i_2-j_2,\ldots,i_N-j_N}(l-1)
\end{aligned}
\tag{A14}
$$

Equation (A14) is in the form of a convolution, but upon comparing it to Equation (A3), we see there is a sign reversal between the summation variables and their corresponding subscripts. To put it in the form of a convolution, we write the last line of Equation (A14) as

$$
\begin{aligned}
\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{W}}} &= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N}\delta_{i_1,i_2,\ldots,i_N}(l)\mathcal{I}_{i_1-j_1,i_2-j_2,\ldots,i_N-j_N}(l-1) \\
&= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N}\delta_{i_1,i_2,\ldots,i_N}(l)\mathcal{I}_{-(j_1-i_1),-(j_2-i_2),\ldots,-(j_N-i_N)}(l-1) \\
&= \delta_{j_1,j_2,\ldots,j_N}(l)\star\mathcal{I}_{-j_1,-j_2,\ldots,-j_N}(l-1) \\
&= \delta_{j_1,j_2,\ldots,j_N}(l)\star rot_N(\boldsymbol{\mathcal{I}}(l-1)) \\
&= conv\left(\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{Z}}},rot_N(\boldsymbol{\mathcal{I}}(l-1))\right)
\end{aligned}
\tag{A15}
$$

Finally, as far as the gradient with respect to the layer's bias is concerned, $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{b}}$, we have

$$\begin{aligned}
\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{b}} &= \frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{b}(l)} \\
&= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N} \frac{\partial \mathcal{L}}{\partial \mathcal{Z}_{i_1,i_2,\ldots,i_N}(l)} \frac{\partial \mathcal{Z}_{i_1,i_2,\ldots,i_N}(l)}{\boldsymbol{b}(l)} \\
&= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N} \delta_{i_1,i_2,\ldots,i_N}(l) \frac{\partial \mathcal{Z}_{i_1,i_2,\ldots,i_N}(l)}{\boldsymbol{b}(l)} \\
&= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N} \delta_{i_1,i_2,\ldots,i_N}(l) \\
&\frac{\partial\left[\sum_{j_1}\sum_{j_2}\cdots\sum_{j_N} \mathcal{W}_{j_1,j_2,\ldots,j_N}(l) f(\mathcal{Z}_{i_1-j_1,i_2-j_2,\ldots,i_N-j_N}(l-1)) + \boldsymbol{b}(l)\right]}{\partial \boldsymbol{b}(l)} \\
&= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N} \delta_{i_1,i_2,\ldots,i_N}(l) \\
&= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N} \left(\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\mathcal{Z}}}\right)
\end{aligned} \tag{A16}$$

Equations (A13) and (A15) sketch the general methodology for computing the desired gradients, and they indicate that in reality both forward and backward passes of an ND convolutional layer are convolutions.

Appendix C. 4D-ETCN Architectures Parameter Tuning

The first set of experiments assesses the performance of the 4D-ETCN in relation to the filter size of its encoder. We trained three different 4D-ETCN models with filter size equal to $(2,2,2,2)$ $(3,3,3,3)$ and $(4,4,4,4)$, while the rest of the parameters remained to their default values as described above. In an attempt to investigate the impact of richer features' representations to the proposed model's performance, the number of filters on each of the three 4D convolutional layers of the encoder were also tuned at the same time by a starting tuple of 4–8–8 to a final one of 256–512–512, thus ending up with 21 different experimental scenarios in total. Table A2 shows that as we employed more filters, the obtained MSE loss decreased, thus indicating that the network's generalization capability improved. In addition, the kernel size of the 4D filters implies that wider receptive fields led to clearly ameliorated results for both datasets. We also observe that our model performed equally well in both datasets, thus indicating its general efficiency towards different input data sources. All in all, we can conclude that for the Crete dataset 4D filters of size $(4,4,4,4)$ should be employed, while for the toughest Italy one, 4D filters of size $(3,3,3,3)$ are enough.

Table A2. MSE loss for the trained 4D-ETCN models by tuning of its encoder. Employing more filters with wider receptive fields led to more complex and accurate models for both examined datasets.

Filter Size—Encoder	# Filters—Encoder	MSE—Crete	MSE—Italy
(2,2,2,2)	4–8–8	0.0203299	0.0351741
	8–16–16	0.00981018	0.0174846
	16–32–32	0.00279834	0.00610852
	32–64–64	0.00135296	0.00330851
	64–128–128	0.00104249	0.00214806
	128–256–256	0.000914299	0.00196186
	256–512–512	0.000865973	0.00182972

Table A2. *Cont.*

Filter Size—Encoder	# Filters—Encoder	MSE—Crete	MSE—Italy
(3,3,3,3)	4–8–8	0.0214919	0.0304642
	8–16–16	0.0049132	0.0108743
	16–32–32	0.00208164	0.00432188
	32–64–64	0.00111296	0.00281552
	64–128-128	0.000869734	0.0019207
	128–256–256	0.000776139	0.00155718
	256–512–512	0.00077451	**0.00149614**
(4,4,4,4)	4–8–8	0.0126214	0.0378467
	8–16–16	0.0055291	0.0112438
	16–32–32	0.00165563	0.00336393
	32–64–64	0.00133122	0.00256895
	64–128–128	0.000943455	0.00175497
	128–256–256	0.00079662	0.00157785
	256–512–512	**0.000751026**	0.00159778

The second set of experiments quantifies the impact of the kernel size of 1D convolutional layer of the TCN part of our proposed 4D model architecture. We again trained three different models with filter sizes equal to 2, 3, and 4, while the filters of the encoder part were tuned exactly as explained before. As before, we observe that more filters clearly led to better performing models, while the baseline value of the TCN 1D kernel needed to be increased to 3 for the Crete dataset and 4 for the more demanding Italy one.

The third hyperparameter examined in our ablation study is the 4D filter size at its decoder part, thus focusing on the receptive field of the upsampling/transpose convolution operation of the proposed 4D model architecture. We again trained three different models with filter size and number of filters of the encoder part analogous to the respective encoder case. Once more, making use of more filters boosted the models under examination, while larger receptive fields for upsampling ended up with more accurate predictions for both datasets.

The last hyperparameter tuned in our ablation study is the dropout rate, which was tuned in our quest to regularize its performance and avoid overfitting issues. For that cause, we tried three different dropout rates equal to 0.3, 0.4, and 0.5, and we trained the respective models with number of filters of the encoder part analogous to the respective encoder case. From this set of experiments, we observe that extreme dropout rates led to worse-performing models, thus indicating our architectures do not suffer from overfitting issues. In addition, employing more filters led to increased performance for both datasets, as in the previous set of experiments.

As can be seen from the results reported in Tables A2–A5, the two datasets required quite different hyperparameters' configuration for achieving optimal results. Based on these combinations highlighted in the aforementioned tables, we selected the the best-performing values for each investigated hyperparameter and performed a final set of optimization experiments by tuning only the number of filters in the encoder part of our 4D architecture. The results reported in Tables A6 and A7 imply that after the initial hyperparameters' tuning, adopting architectures with more filters in the encoder part of the network led to clear performance gains. All in all, the adopted model's architectures for the two different datasets derived from this ablation study are highlighted in Tables A6 and A7, and they were used throughout the rest of the present work.

Table A3. MSE loss for the trained 4D-ETCN models by tuning of its TCN. Increasing the 1D filter size and employing more filters led to better performing models for both examined datasets.

Filter Size—TCN	# Filters—Encoder	MSE—Crete	MSE—Italy
2	4–8–8	0.0203299	0.0351741
	8–16–16	0.00981018	0.0174846
	16–32–32	0.00279834	0.00610852
	32–64–64	0.00135296	0.00330851
	64–128–128	0.00104249	0.00214806
	128–256–256	0.000914299	0.00196186
	256–512–512	0.000865973	0.00182972
3	4–8–8	0.0154268	0.0381451
	8–16–16	0.00621275	0.0173167
	16–32–32	0.00239204	0.00518445
	32–64–64	0.00126476	0.00316894
	64–128–128	0.00103368	0.00240135
	128–256–256	0.000864882	0.00195037
	256–512–512	**0.000804169**	0.00179337
4	4–8–8	0.0168413	0.0364333
	8–16–16	0.00604127	0.0144646
	16–32–32	0.00195937	0.0041265
	32–64–64	0.00126467	0.00310254
	64–128-128	0.000947225	0.00225102
	128-256–256	0.000844021	0.0017829
	256–512–512	0.000811305	**0.00174023**

Table A4. MSE loss for the trained 4D-ETCN models by tuning of its decoder. Larger receptive fields for upsampling in conjunction with richer feature representations ended up with better performing models for both examined datasets.

Filter Size—Decoder	# Filters—Encoder	MSE—Crete	MSE—Italy
(2,2,2,2)	4–8–8	0.0203299	0.0351741
	8–16–16	0.00981018	0.0174846
	16–32–32	0.00279834	0.00610852
	32–64–64	0.00135296	0.00330851
	64–128–128	0.00104249	0.00214806
	128–256–256	0.000914299	0.00196186
	256–512–512	0.000865973	0.00182972

Table A4. *Cont.*

Filter Size—Decoder	# Filters—Encoder	MSE—Crete	MSE—Italy
(3,3,3,3)	4–8–8	0.00790696	0.0331473
	8–16–16	0.00295041	0.0137488
	16–32–32	0.00141169	0.00544591
	32–64–64	0.00105156	0.00291967
	64–128–128	0.000940168	0.00194803
	128–256–256	0.000834836	0.00181562
	256–512–512	0.0007741	**0.00165275**
(4,4,4,4)	4–8–8	0.00577045	0.022633
	8–16–16	0.00258578	0.00909528
	16–32–32	0.00133485	0.00490685
	32–64–64	0.000997249	0.00211714
	64–128–128	0.000929486	0.00184755
	128–256–256	0.00082758	0.00166246
	256–512–512	**0.000764315**	0.00169711

Table A5. MSE loss for the trained 4D-ETCN models by tuning of its dropout rate. Dropping half of the available data had a negative impact on the examined models, while more filters again contributed to enhanced performance for both examined datasets.

Dropout Rate	# Filters—Encoder	MSE—Crete	MSE—Italy
0.3	4–8–8	0.0203299	0.0351741
	8–16–16	0.00981018	0.0174846
	16–32–32	0.00279834	0.00610852
	32–64–64	0.00135296	0.00330851
	64–128–128	0.00104249	0.00214806
	128–256–256	0.000914299	0.00196186
	256–512–512	**0.000865973**	0.00182972
0.4	4–8–8	0.0341803	0.0498237
	8–16–16	0.0120504	0.0244045
	16–32–32	0.00388539	0.0093801
	32–64–64	0.00170305	0.00426584
	64–128–128	0.00112826	0.00252556
	128–256–256	0.000968859	0.00206769
	256–512–512	0.000870094	**0.00178803**
0.5	4–8–8	0.0477682	0.0616626
	8–16–16	0.0161133	0.0270213
	16–32–32	0.00581007	0.0120161
	32–64–64	0.00224298	0.00557922
	64–128–128	0.0012323	0.0027299
	128–256–256	0.000980545	0.00216101
	256–512–512	0.000881541	0.00191598

Table A6. Final tuning of the proposed 4D architecture for the Crete dataset. Adopting richer features' representations led to better performing model.

Filter Size—Encoder	Filter Size—ETCN	Filter Size—Decoder	Dropout Rate	# Filters—Encoder	MSE—Crete
(4,4,4,4)	3	(4,4,4,4)	0.3	4–8–8	0.00415853
				8–16–16	0.00247294
				16–32–32	0.000887057
				32–64–64	0.000797183
				64–128–128	0.000733693
				128–256–256	**0.000698685**
				256–512–512	0.000754672

Table A7. Final tuning of the proposed 4D architecture for the Italy dataset. Employing more filters had a clear positive effect on the proposed model's learning capacity.

Filter Size—Encoder	Filter Size—ETCN	Filter Size—Decoder	Dropout Rate	# Filters—Encoder	MSE—Italy
(3,3,3,3)	4	(3,3,3,3)	0.4	4–8–8	0.0299083
				8–16–16	0.00801586
				16–32–32	0.00326288
				32–64–64	0.00237769
				64–128–128	0.00163417
				128–256–256	0.0014781
				256–512–512	**0.00145329**

References

1. Bojinski, S.; Verstraete, M.; Peterson, T.C.; Richter, C.; Simmons, A.; Zemp, M. The concept of essential climate variables in support of climate research, applications, and policy. *Bull. Am. Meteorol. Soc.* **2014**, *95*, 1431–1443. [CrossRef]
2. Massonnet, F.; Bellprat, O.; Guemas, V.; Doblas-Reyes, F.J. Using climate models to estimate the quality of global observational data sets. *Science* **2016**, *354*, 452–455. [CrossRef] [PubMed]
3. Huntingford, C.; Jeffers, E.S.; Bonsall, M.B.; Christensen, H.M.; Lees, T.; Yang, H. Machine learning and artificial intelligence to aid climate change research and preparedness. *Environ. Res. Lett.* **2019**, *14*, 124007. [CrossRef]
4. Tsagkatakis, G.; Aidini, A.; Fotiadou, K.; Giannopoulos, M.; Pentari, A.; Tsakalides, P. Survey of deep-learning approaches for remote sensing observation enhancement. *Sensors* **2019**, *19*, 3929. [CrossRef] [PubMed]
5. Giannopoulos, M.; Aidini, A.; Pentari, A.; Fotiadou, K.; Tsakalides, P. Classification of compressed remote sensing multispectral images via convolutional neural networks. *J. Imaging* **2020**, *6*, 24. [CrossRef] [PubMed]
6. Chen, Y.; Lin, Z.; Zhao, X.; Wang, G.; Gu, Y. Deep learning-based classification of hyperspectral data. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2014**, *7*, 2094–2107. [CrossRef]
7. Giannopoulos, M.; Tsagkatakis, G.; Tsakalides, P. 4D U-Nets for Multi-Temporal Remote Sensing Data Classification. *Remote Sens.* **2022**, *14*, 634. [CrossRef]
8. Giannopoulos, M.; Tsagkatakis, G.; Tsakalides, P. 4D Convolutional Neural Networks for Multi-Spectral and Multi-Temporal Remote Sensing Data Classification. In Proceedings of the ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Singapore, 23–27 May 2022; pp. 1541–1545.
9. Villia, M.M.; Tsagkatakis, G.; Moghaddam, M.; Tsakalides, P. Embedded Temporal Convolutional Networks for Essential Climate Variables Forecasting. *Sensors* **2022**, *22*, 1851. [CrossRef] [PubMed]
10. Aspri, M.; Tsagkatakis, G.; Tsakalides, P. Distributed training and inference of deep learning models for multi-modal land cover classification. *Remote Sens.* **2020**, *12*, 2670. [CrossRef]
11. Bittner, K.; Cui, S.; Reinartz, P. Building extraction from remote sensing data using fully convolutional networks. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.-ISPRS Arch.* **2017**, *42*, 481–486. [CrossRef]
12. Stivaktakis, R.; Tsagkatakis, G.; Tsakalides, P. Deep learning for multilabel land cover scene categorization using data augmentation. *IEEE Geosci. Remote Sens. Lett.* **2019**, *16*, 1031–1035. [CrossRef]

13. Koehler, J.; Kuenzer, C. Forecasting spatio-temporal dynamics on the land surface using earth observation data—A review. *Remote Sens.* **2020**, *12*, 3513. [CrossRef]
14. Hansen, M.C.; Potapov, P.V.; Moore, R.; Hancher, M.; Turubanova, S.A.; Tyukavina, A.; Thau, D.; Stehman, S.V.; Goetz, S.J.; Loveland, T.R.; et al. High-resolution global maps of 21st-century forest cover change. *Science* **2013**, *342*, 850–853. [CrossRef]
15. Davis, K.F.; Koo, H.I.; Dell'Angelo, J.; D'Odorico, P.; Estes, L.; Kehoe, L.J.; Kharratzadeh, M.; Kuemmerle, T.; Machava, D.; Pais, A.d.J.R.; et al. Tropical forest loss enhanced by large-scale land acquisitions. *Nat. Geosci.* **2020**, *13*, 482–488. [CrossRef]
16. Lee, M.; Shevliakova, E.; Stock, C.A.; Malyshev, S.; Milly, P.C. Prominence of the tropics in the recent rise of global nitrogen pollution. *Nat. Commun.* **2019**, *10*, 1437. [CrossRef]
17. Shaddick, G.; Thomas, M.L.; Mudu, P.; Ruggeri, G.; Gumy, S. Half the world's population are exposed to increasing air pollution. *NPJ Clim. Atmos. Sci.* **2020**, *3*, 23. [CrossRef]
18. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
19. Shi, X.; Chen, Z.; Wang, H.; Yeung, D.Y.; Wong, W.K.; Woo, W.C. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In Proceedings of the Advances in Neural Information Processing Systems 28 (NIPS 2015), Montreal, QC, Canada, 7–12 December 2015.
20. Lea, C.; Flynn, M.D.; Vidal, R.; Reiter, A.; Hager, G.D. Temporal convolutional networks for action segmentation and detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 156–165.
21. Oord, A.v.d.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.; Kavukcuoglu, K. Wavenet: A generative model for raw audio. *arXiv* **2016**, arXiv:1609.03499.
22. Kalchbrenner, N.; Espeholt, L.; Simonyan, K.; Oord, A.v.d.; Graves, A.; Kavukcuoglu, K. Neural machine translation in linear time. *arXiv* **2016**, arXiv:1610.10099.
23. Myronenko, A.; Yang, D.; Buch, V.; Xu, D.; Ihsani, A.; Doyle, S.; Michalski, M.; Tenenholtz, N.; Roth, H. 4D CNN for semantic segmentation of cardiac volumetric sequences. In Proceedings of the International Workshop on Statistical Atlases and Computational Models of the Heart, Shenzhen, China, 13 October 2019; Springe: Berlin/Heidelberg, Germany, 2019; pp. 72–80.
24. Kim, S.; Hong, S.; Joh, M.; Song, S.k. Deeprain: Convlstm network for precipitation prediction using multichannel radar data. *arXiv* **2017**, arXiv:1711.02316.
25. Xiao, C.; Chen, N.; Hu, C.; Wang, K.; Xu, Z.; Cai, Y.; Xu, L.; Chen, Z.; Gong, J. A spatiotemporal deep learning model for sea surface temperature field prediction using time-series satellite data. *Environ. Model. Softw.* **2019**, *120*, 104502. [CrossRef]
26. Bai, S.; Kolter, J.Z.; Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv* **2018**, arXiv:1803.01271.
27. Vega-Márquez, B.; Rubio-Escudero, C.; Nepomuceno-Chamorro, I.A.; Arcos-Vargas, Á. Use of Deep Learning Architectures for Day-Ahead Electricity Price Forecasting over Different Time Periods in the Spanish Electricity Market. *Appl. Sci.* **2021**, *11*, 6097. [CrossRef]
28. Kolda, T.G.; Bader, B.W. Tensor decompositions and applications. *SIAM Rev.* **2009**, *51*, 455–500. [CrossRef]
29. Cichocki, A.; Mandic, D.; De Lathauwer, L.; Zhou, G.; Zhao, Q.; Caiafa, C.; Phan, H.A. Tensor decompositions for signal processing applications: From two-way to multiway component analysis. *IEEE Signal Process. Mag.* **2015**, *32*, 145–163. [CrossRef]
30. Papalexakis, E.E.; Faloutsos, C.; Sidiropoulos, N.D. Tensors for data mining and data fusion: Models, applications, and scalable algorithms. *ACM Trans. Intell. Syst. Technol. (TIST)* **2016**, *8*, 16. [CrossRef]
31. Panagakis, Y.; Kossaifi, J.; Chrysos, G.G.; Oldfield, J.; Nicolaou, M.A.; Anandkumar, A.; Zafeiriou, S. Tensor Methods in Computer Vision and Deep Learning. *Proc. IEEE* **2021**, *109*, 863–890. [CrossRef]
32. Tucker, L.R. Implications of factor analysis of three-way matrices for measurement of change. *Probl. Meas. Chang.* **1963**, *15*, 122–137.
33. Tucker, L.R. The extension of factor analysis to three-dimensional matrices. *Contrib. Math. Psychol.* **1964**, *110119*, 110–182.
34. Tucker, L.R. *Some Mathematical Notes on Three-Mode Factor Analysis* ; Department of Psychology, University of Illinois: Urbana, IL, USA, 1965.
35. De Lathauwer, L.; De Moor, B.; Vandewalle, J. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.* **2000**, *21*, 1253–1278. [CrossRef]
36. Cichocki, A.; Lee, N.; Oseledets, I.V.; Phan, A.H.; Zhao, Q.; Mandic, D. Low-rank tensor networks for dimensionality reduction and large-scale optimization problems: Perspectives and challenges part 1. *arXiv* **2016**, arXiv:1609.00893.
37. Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. Tensorflow: A system for large-scale machine learning. In Proceedings of the 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), Savannah, GA, USA, 2–4 November 2016; pp. 265–283.
38. Chetlur, S.; Woolley, C.; Vandermersch, P.; Cohen, J.; Tran, J.; Catanzaro, B.; Shelhamer, E. cudnn: Efficient primitives for deep learning. *arXiv* **2014**, arXiv:1410.0759.
39. Jorda, M.; Valero-Lara, P.; Pena, A.J. Performance evaluation of cudnn convolution algorithms on nvidia volta gpus. *IEEE Access* **2019**, *7*, 70461–70473. [CrossRef]
40. Psarras, C.; Karlsson, L.; Li, J.; Bientinesi, P. The landscape of software for tensor computations. *arXiv* **2021**, arXiv:2103.13756.
41. Kossaifi, J.; Panagakis, Y.; Anandkumar, A.; Pantic, M. Tensorly: Tensor learning in python. *arXiv* **2016**, arXiv:1610.09555.

42. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. Pytorch: An imperative style, high-performance deep learning library. In Proceedings of the Advances in Neural Information Processing Systems 32 (NeurIPS 2019), Vancouver, BC, Canada, 8–14 December 2019.
43. Harris, C.R.; Millman, K.J.; Van Der Walt, S.J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N.J.; et al. Array programming with NumPy. *Nature* **2020**, *585*, 357–362. [CrossRef] [PubMed]
44. Kolda, T.G.; Bader, B.W. *MATLAB Tensor Toolbox*; Technical Report; Sandia National Laboratories (SNL): Albuquerque, NM, USA; Livermore, CA, USA, 2006.
45. Zhang, S.; Guo, S.; Huang, W.; Scott, M.R.; Wang, L. V4d: 4d convolutional neural networks for video-level representation learning. *arXiv* **2020**, arXiv:2002.07442.
46. Salimans, T.; Kingma, D.P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In Proceedings of the Advances in Neural Information Processing Systems 29 (NIPS 2016), Barcelona, Spain, 5–10 December 2016.
47. Ronneberger, O.; Fischer, P.; Brox, T. U-net: Convolutional networks for biomedical image segmentation. In Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention, Munich, Germany, 5–9 October 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 234–241.
48. Çiçek, Ö.; Abdulkadir, A.; Lienkamp, S.S.; Brox, T.; Ronneberger, O. 3D U-Net: Learning dense volumetric segmentation from sparse annotation. In Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention, Athens, Greece, 17–21 October 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 424–432.
49. Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. JMLR Workshop and Conference Proceedings, Sardinia, Italy, 13–15 May 2010; pp. 249–256.
50. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
51. Muñoz Sabater, J. ERA5-Land Monthly Averaged Data from 1981 to Present, Copernicus Climate Change Service (C3S) Climate Data Store (CDS), 2019. Available online: https://cds.climate.copernicus.eu/cdsapp#!/dataset/10.24381/cds.68d2bb30?tab=overview (accessed on 1 February 2024).
52. Gorelick, N.; Hancher, M.; Dixon, M.; Ilyushchenko, S.; Thau, D.; Moore, R. Google Earth Engine: Planetary-scale geospatial analysis for everyone. *Remote Sens. Environ.* **2017**, *202*, 18–27. [CrossRef]
53. Chollet, F. Keras. Available online: https://keras.io (accessed on 1 February 2024).
54. Wang, Z.; Bovik, A.C.; Sheikh, H.R.; Simoncelli, E.P. Image quality assessment: From error visibility to structural similarity. *IEEE Trans. Image Process.* **2004**, *13*, 600–612. [CrossRef]