Arabian Journal for Science and Engineering (2021) 46:3423–3442
https://doi.org/10.1007/s13369-020-05109-x

RESEARCH ARTICLE-COMPUTER ENGINEERING AND COMPUTER SCIENCE

Multivariate Time Series Forecasting with Dilated Residual Convolutional Neural Networks for Urban Air Quality Prediction

Meriem Benhaddi[1] · **Jamal Ouarzazi**[2]

Received: 15 March 2020 / Accepted: 2 November 2020 / Published online: 3 January 2021
© King Fahd University of Petroleum & Minerals 2021

Abstract
Multivariate time series (MTS) forecasting is a research field that is gaining more and more importance as time series data generators proliferate in the growing era of Internet of Things. Deep learning architecture for MTS data has been and still a very active research area as there is no comprehensive comparative study of the different architectures, let alone a perfect architecture that can solve all types of time series modeling problem. In this paper, we start by highlighting time series analysis requirements, to propose a new model for conditional multivariate time series forecasting based on the lately introduced WaveNet architecture. Our model is composed of stacked residual causal dilated convolutions that provide large scope in time series history and foster learning of long-term dependencies. Parameterized skip connections allow catching early trends and properties while conditioning enables modeling the associations that exist between multivariate data. We used group normalization for its stability and independence from the batch size. We present a deep comparison of the structure, flexibility, flow control, memory consumption, robustness and stability capacities of our new model against those of recurrent neural networks state-of-the-art approaches—long-short-term-memory (LSTM) and gated recurrent unit (GRU). To assess the performances of our model, we conduct extensive experimental testing on the task of urban air quality prediction in Marrakesh city using six real-world multi-sensor and multivariate time series datasets. We compare the results of our model against the results of LSTM and GRU. Our experiments revealed that our proposed WaveNet-temporal-CNN outperform recurrent models ability to learn long-term dependencies in a time-efficient way.

Keywords Deep learning · Temporal convolutional neural network · WaveNet · Causal and dilated convolutions · Residual connections · Skip connections · Conditional multivariate time series · Group normalization · Recurrent neural network · LSTM · GRU · Forecasting · Air quality prediction

1 Introduction

Prediction activities are attached to human existence since the dawn time, and the reason why forecasting is so important is that prediction of future events is a crucial information for many planning or decision-making activities ranging from manufacturing operations and inventory control in retail stores to public sectors activities such as demography control and city management [1]. Forecasting is equally much needed in pollution as the latter has a detrimental effect on human health. Particulate matter (PM) is invisible material released via vehicle exhaust emissions and ends up suspended in the atmosphere. A recent study led by College University of London showed a tight correlation between air quality and suicide and depression rates [2].

Quantitative forecasting methods predict future data based on historical data and make use of different types of algorithms that can have different optimization strategies and whose quality is measured by the number of errors or by the total distance between the predicted and the observed/real values [3–5]. Neural networks (NN) or deep neural networks are machine learning techniques that have lately surpassed other classical methods as they allow the super calculator to learn how to learn the correlations and dependencies between different types of data. Deep neural networks (DNN) do not

✉ Meriem Benhaddi
m.benhaddi@uca.ac.ma

Jamal Ouarzazi
ouarzazi@uca.ac.ma

[1] Faculty of Sciences and Techniques, Cadi Ayyad University of Marrakesh, Marrakesh, Morocco

[2] Faculty of Sciences Semlalia, Cadi Ayyad University of Marrakech, Marrakesh, Morocco

need humans to do data features engineering, as this is necessary with classical methods, thus alleviating humans' tasks and making the learning process quicker and more flexible. DNN have been successfully used for different tasks where pattern recognition is needed, ranging from images classification or object localization with convolutional neural networks (CNN), to generating new data instances based on phenomenons studying in order to predict and forecast future data or to generate pieces of art [6]. DNN architectures and topologies development are a very active area of research as new ones are constantly composed to solve different problems [7]. Van Veen have highlighted the diversity of NN by listing twenty nine different architectures in 2016 [8], while Brunton and Kutz [9] have enumerated eighteen of the most considered architectures in the literature. Many researchers have explored and experimented different types of DNN models and architectures trying to discover what best suits a given type of task in a given application domain; some attempts to consider a unified model for multi-task resolution have been proposed [10].

In this paper, we are interested in multivariate time series data modeling DNN architectures, particularly generative auto-encoder models such as recurrent neural networks (RNN) and convolutional neural networks (CNN) [11]. A time series denotes for a chronologically ordered set of values of one studied variable [12]. Time series data are involved in most forecasting problems as they offer a simple and efficient technique to predict the future. Starting with the autoregressive integrated moving average models (ARIMA) [13] and getting to new regression models that made an important advance, many of the models used in time series forecasting have been explored for many decades but lately had to elapse in favor of machine learning techniques. RNN showed to be promising for capturing time dependencies over sequence of variable-sized vector input. Long short-term memory (LSTM) and gated recurrent unit (GRU) (variant of LSTM introduced in 2014) are the two improved versions of RNN, the most used and encountered in the literature. LSTM and GRU are called gated RNN as they both rely on different types of gates to regulate the data flow inside their units. Multiple empirical research studies have been conducted in the last few years trying to compare the performances of these two architectures regarding machine translation, speech recognition, traffic flow prediction, health monitoring, video segmentation, emotion classification, etc. The most prominent empirical evaluation was led by Greff et al. on several LSTM variants and concluded that the result performances were very comparable [7, 14–16]. While CNN have being applied to multi-dimensional data for video segmentation and images classification, one-dimensional CNN are suitable for time series data analysis; however, one-dimensional CNN cannot capture long-term time-dependencies with non-fixed length data as RNN do. Temporal convolutional neural networks (TCNN) are a new family of CNN architecture that can hold past data along the network as RNN do [17]. Recent research demonstrates that TCNN and RNN deliver competitive results performances in diverse applications including audio processing [18], text classification [19] and time series forecasting [11], (mainly with one-dimensional temporal data), while TCNN surpass RNN's performances in translation tasks [20].

A multivariate time series (MTS) consists of a collection of interdependent time series variables, where each variable's future value rely on its own past values as well as the past values of other variables; hence, the difficulty of multivariate time series analysis which surpasses that of univariate time series [21]. The need for MTS analysis is exhorted by big data and Internet of Things (IoT) technologies which enable mixing data from various and disparate sources, MTS collection and processing potentially being realized in real-time using parallel computing. This presents challenges such as handling multiple parameters within high dimensionality [22]. Many models were proposed in the literature to tackle this issue: multi-channel deep CNN [23, 24], time-CNN [25], residual network or ResNets [26] and encoder [27], to name just a few. However, none of these architectures could outperform the others even though ResNet stands out for its flexibility compared to the rest of the models [11]. In 2016, Google search team introduced WaveNet architecture, a stack of dilated convolutions with residual and skip connections for text-to-speech generation [28]. The authors demonstrated that WaveNet architecture is very promising for sequence-based tasks with long-term dependencies. WaveNet indeed surpasses other CNN by its capacity to build deep networks while gradually expanding the receptive field and adding conditioning to better 'see' the long-term dependencies, while preserving network complexity.

Through a literature review, we noticed that WaveNet for MTS forecasting has not been explored as it should be. We believe that WaveNet is a revolutionary step and a unique type of CNN architectures as it brings together many interesting concepts. Being aware of the complexity of MTS forecasting task and having considered all the requirements and challenges, we propose in this paper a new temporal CNN architecture for MTS forecasting through an adaptation of WaveNet model. Our model is unique and draws its strength from a pre-study of MTS requirements as well as a comparison of LSTM, GRU and WaveNet models' theoretical basis. These pre-studies drive our choices and ensure our model's supremacy. We assess the performance of the proposed model using urban air quality multi-dimensional time series datasets collected by multi-sensors in Marrakesh city. Our work is innovative since it presents a new model coupled with a solid methodology that performs theoretical and experimental studies which both confirm that WaveNet is the future of MTS forecasting.

The main contributions of this paper are as follows:

1. Listing time series analysis requirements and comparatively showing how each of the studied models (LSTM, GRU and temporal CNN) can meet these requirements.
2. Introducing a new TCNN architecture for MTS forecasting, which is an adapted WaveNet architecture that best suits MTS analysis requirements addressed in point 1. Our model is a dilated residual temporal CNN that enjoys highly parameterized and well situated skip connections, variables' conditioning, as well as the recently proposed group normalization.
3. Presenting a deep theoretical comparison of LSTM, GRU and our model by their structures, flexibility, flow control, memory consumption, robustness and stability capacities, and highlighting the reasons of our model's supremacy over LSTM and GRU, through an in-depth analysis of the internal structure and mathematical logic of the three models.
4. Delivering empirical evaluation results using real datasets, which show that our model outperforms LSTM and GRU in accuracy and speed. Our model is able to learn long-term dependencies and complex correlations among MTS in real-world datasets.

This paper is organized as follows: Sect. 2.1 formalizes the multivariate sequence problem description. Section 2.2 discusses related work and what distinguishes our paper compared to previous work. Section 2.3 presents LSTM and GRU, and Sect. 2.4 lists the optimization techniques used in this paper. Section 3 explains our new WaveNet-based Temporal CNN (WTCNN) model and its main concepts. Section 4 provides an in-depth comparison of the structure, flexibility, flow control, memory consumption, robustness and stability capacities of LSTM, GRU and WTCNN, based on their type of internal structure and logic. Section 5 is devoted to experiments and assessment of early assumptions. Discussions are included in Sect. 6. Finally, Sect. 7 concludes this paper and provides limitations and future work.

2 Background and Related Work

In this section, the MTS problem is formulated. We then present the current state of research in deep learning for MTS forecasting. The deep learning architectures used in our comparative study are introduced afterward: LSTM and GRU; finally, are presented the used optimization techniques (hyperparameter tuning, walk-forward cross-validation and error metrics).

2.1 Multivariate Sequence Problem Description

The air quality prediction we tackle in this paper is multivariate and multi-output forecasting problem. From a set of features that describe the current state of the study environment at a moment t, we want to predict the future values of a set of five air pollutants based on their own previous values and previous values of the other features.

In a more formal way, this is a sequence to sequence time series prediction problem, where given an m-dimensional input at a time t, $X = (x_1^t, x_2^t, \ldots, x_m^t)$ with $x_j^t \in \mathbb{R}^n$, we calculate the output vector $Y = (x_1^{t+1}, x_2^{t+1}, \ldots, x_p^{t+1})$ at a time $t+1$, where $p < m$, and each x_j^{t+1} depends on its own previous value as well as on the previous values of other features. The aim of this sequence data modeling is to find a mapping function f that produces a prediction vector Y from a historical vector X:

$$(x_1^{t+1}, x_2^{t+1}, \ldots, x_p^{t+1}) = f(x_1^t, x_2^t, \ldots, x_m^t) \tag{1}$$

2.2 Related Work

Various approaches have been proposed to solve time series forecasting problem; however, less studies have ventured into the field of MTS classification due to the curse of dimensionality [29]. Yet, over the past decades, a great deal of attention has been devoted to MTS analysis. Deep learning models have particularly stood out as they are not based on handcrafted features, as well as for their ability to handle high degree of compositional and hierarchical functions, representing input–output mapping through multiple composed transformations [30].

RNN, specifically LSTM and GRU, have been considered as the default choice for sequence-based task modeling in general and time series analysis in particular. Che et al. developed a GRU-based model for MTS classification with missing observations [31]. Filonov et al. used a LSTM-based approach to detect industrial faults in MTS data [32]. However, back propagation through time (BPTT) is computationally costly and RNN's training is reported to be difficult [33].

Inspired by convolutional models success in image processing tasks, researchers started recently using CNN-based models to classify MTS data. Multi-channel deep CNN (MCD-CNN) approach [23, 24] was originally proposed for MTS classification; MCD-CNN is a traditional deep CNN which applies filters separately (in parallel) to each individual dimension. The authors used two convolutional layers where all dimensions' outputs are concatenated, followed by a fully connected (FC) layer and a softmax classifier. The model is assessed on two MTS datasets, but no comparative analysis with similar models was drawn up. Time-CNN [25]

was also originally proposed for MTS classification; similar to MCD-CNN, Time CNN is a traditional deep CNN that consists of two convolutional layers followed by a FC layer with sigmoid function; the FC being the output layer. Time-CNN differs, however, by the use of mean squared error as an alternative of the traditional categorical cross-entropy loss function, as well as the use of average pooling instead of max pooling. Hybrid deep learning models combining RNN and CNN have equally surfaced; Fazle et al.'s work explored a LSTM-fully CNN combination-based solution with a squeeze-and-excitation block [34].

However, deep CNN tend to overfit easily due to big set of parameters to compute. Residual Networks or ResNets [26] add residual connections which allow building very deep networks. Residual connections are shortcuts that lock the network's complexity while stacking additional layers. According to Ismail Fawaz et al. [11], ResNets have outperformed all the time series classification models due to their ability to build powerful deep layers without suffering from low performances caused by vanishing or exploding gradient. Particularly, Wang et al. have joined a fully convolutional neural network to a residual network (ResNet) to build a three block model, ended with a global average pooling layer and a softmax classifier [26]. On the other hand, Yazdanbakhsh and Dick have used dilated convolutions for MTS classification in human activity recognition task [35]; thanks to the receptive field expansion, the network is able to access a broad range of history for better classification.

Bringing together the benefits from residual connections and dilated convolutions, WaveNet model was originally proposed by Google search team [28] for generating raw audio; it has equally been able to provide promising results for time series classification through capturing long-term dependencies in a simpler and more efficient way than other types of networks [36]. Inspired by WaveNet, Borovykh et al. presented in 2017 an adapted model for financial time series forecasting using MTS datasets [37]. Authors used dilated convolutions as core block and applied conditioning to the multivariate dataset. Additionally, they augmented the original architecture with new types of parameterized connections and used Relu function instead of tanh and sigmoid functions. Authors reported facing overfitting problem while trying to increase the number of layers and filters, which prevent the network from learning nonlinearities in variables' relationships. In addition, parameterized skip connections have been placed solely in the first layer on the input and conditions, and the use of Relu function intensifies the exploding gradient problem as it does not limit values. As the authors have concluded their paper, there is scope for improvement, which we try to explore in this paper.

Since we believe that WaveNet architecture holds unexplored potential for MTS forecasting, we present in this paper a new WaveNet-based adapted architecture that applies structural changes to the original WaveNet model. More elements of comparison between our proposed architecture and the original one as well as that of Borovikh et al. are presented in Sect. 3.2.

2.3 Background of Deep Learning Architectures for Time Series Forecasting

2.3.1 Long Short-Term Memory

Long Short-Term Memory neural network is an artificial recurrent neural network type of architecture that focuses on modeling the sequential relationship between past, present and future data. LSTM architecture was initially proposed in 1997 by Hochreiter and Schmidhuber [38] as a remedy to the vanishing or exploding gradient problem encountered in classical old RNN architecture. LSTM introduces new elements called gates. The LSTM initial version did not include forget gate, but only the input and output gates. In 1999, Gers et al. introduced the forget gate allowing the LSTM to decide whether to forget or to remember data [39], thus enabling it to manage the data life time duration by holding its value as long as it is needed and forgetting it once it is no longer required. A typical LSTM unit is composed of three gates: input gate, forget gate and output gate. Figure 1 shows LSTM architecture with forget gate and peephole connections. LSTM tries to mimic the human brain by learning when to forget and when to remember. In 2000, Gers and Schmidhuber added peephole connections into LSTM architecture [40]. Peephole connections link the cell to the gates in order to allow the gates to see the cell state even when the output gate is closed; this allows more sophisticated data flow control and greater accuracy in time dependencies retrieving. Self-learned weights are established during the training phase by means of specific equations (Eqs. 2 to 7 below).

LSTM showed their great results on language modeling and phoneme classification [41], handwriting recognition [42], speech recognition [43], reinforcement learning [44], sentiment classification [45], human movement trajectories [46], music improvisation [47] and others.

Let us define x^t as the input at time t. The LSTM layer parameters are:

- W_z, W_i, W_f and W_o as the input weights
- R_z, R_i, R_f and R_o as the recurrent weights
- p_i, p_f and p_o as the peephole weights
- b_z, b_i, b_f and b_o as the bias weights

The data flow is controlled by the following equations:
LSTM block input:

$$z^t = \tanh(W_z x^t + R_z y^{t-1} + b_z) \tag{2}$$

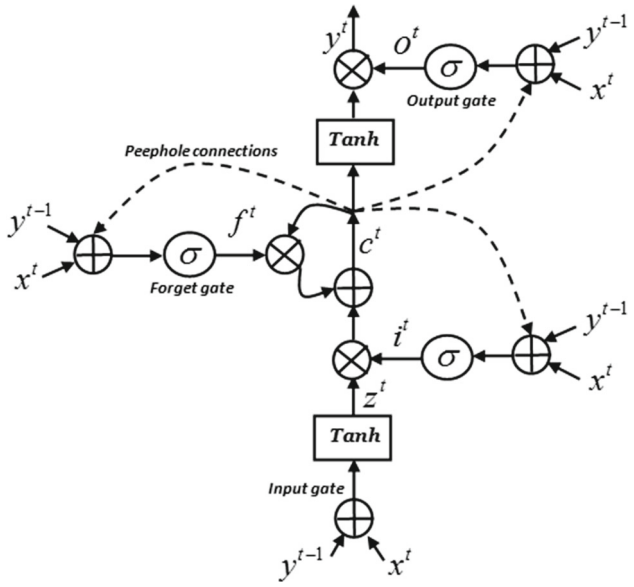


Fig. 1 LSTM with forget gate and peephole connections [14]

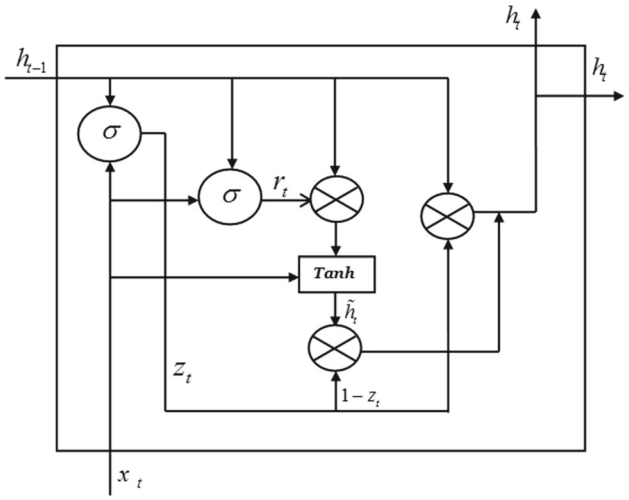


Fig. 2 A gated recurrent unit [48]

input gate:

$$i^t = \sigma(W_i x^t + R_i y^{t-1} + p_i \odot c^{t-1} + b_i) \tag{3}$$

forget gate:

$$f^t = \sigma(W_f x^t + R_f y^{t-1} + p_f \odot c^{t-1} + b_f) \tag{4}$$

cell:

$$c^t = i^t \odot z^t + f^t \odot c^{t-1}) \tag{5}$$

output gate:

$$o^t = \sigma(W_o x^t + R_o y^{t-1} + p_o \odot c^{t-1} + b_o) \tag{6}$$

LSTM block output:

$$y^t = o^t \odot \tanh(c^t) \tag{7}$$

where sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ and $\tanh(x) = \frac{2}{1+e^{-2x}} - 1$ are the network activation functions, $\odot$ is the point-wise multiplication function.

2.3.2 Gated Recurrent Unit

2014 saw the appearance of the gated recurrent unit (GRU) [48] which is a simplified LSTM variant. In fact, LSTM have shown to be very effective with the vanishing or exploding gradient problem; however, it is highly demanding in computational memory due to the multiple memory cells in its architecture. GRU architecture is similar to LSTM as they both have gates that control the flow of data within the unit; they are dissimilar as GRU has fewer cells allowing less computation and easier implementation. The Gated Recurrent unit, presented in Fig. 2, is comprised of an update gate and a reset gate; the update gate allows the model to determine which information from the past needs to be passed along to the future, whereas the rest gate specifies the information to forget.

The GRU activation function is modeled as:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t \tag{8}$$

where $z_t = \sigma(W_r x_t + U_r h_{t-1})$ value is updated as:

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \tag{9}$$

where W and U are the network weights. The candidate activation is computed as:

$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1})) \tag{10}$$

where $\odot$ represents the element-wise multiplication. r_t is computed similarly to z_t:

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \tag{11}$$

Like LSTM, GRU are being explored and experimented by the community of researchers working on sequence modelling on various problems such as emotion classification [49], time series data analysis [50], image analysis [51], speech recognition and synthesis [52, 53], to name but a few. Some elements of comparison between GRU and LSTM are presented in Sect. 3 below.

2.4 Neural Network Optimization

2.4.1 Hyperparameter Tuning

Neural network optimization is an important issue in deep learning as the hyperparameters choice greatly impacts the training and prediction abilities of the NN model [54]. For hyperparameters optimization (number of layers and neurons, filter size, dropout rate, gradient clipping, input length, …), we use grid search techniques which creates a network search space equal to the Cartesian product of all the hyperparameters's search space. For each hyperparameter, the learning process is launched in a certain search space to choose the best configuration.

2.4.2 Walk-forward Cross-validation

Cross-validation techniques are needed to assess how well the results (weights values) obtained from the training phase will generalize during the test phase. k-fold cross-validation is the most used technique and consists of arbitrarily splitting the training dataset into a number of k slices, each slice being used exactly one time as a validation set, while the other ($k - 1$) slices are used as training set; the total dataset is hence trained k times. However, k-fold technique does not guarantee that the order of data records will be conserved when splitting it, which makes it not suitable for time series data validation. Walk-forward cross-validation is used instead and consists of the following steps:

Walk-forward procedure:

1. *Data are split into history h and future data. The model learns the best parameters from h*
2. *The model proceeds to the prediction of the next time step value*
3. *The corresponding observed value is added to the history h for predicting the next value*
4. *The difference between the predicted value and the observed one is stored for the final assessment of the model*
5. *Repeat from step 1*

2.4.3 Evaluation Metrics

To assess the three models, we used root mean squared error (RMSE), root relative squared error (RRSE) and mean absolute percentage error (MAPE), defined as:

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{j=1}^{n} e_j^2}{n}} \tag{12}$$

$$\mathrm{RRSE} = \sqrt{\frac{\sum_{j=1}^{n} e_j^2}{\sum_{j=1}^{n} (A_j - \overline{A})^2}} \tag{13}$$

$$\mathrm{MAPE} = \frac{100}{n} \sum_{j}^{n} \frac{|e_j|}{|A_j|} \tag{14}$$

where e is the error or difference between the predicted value and the observed or real one, A is the real value, and $\overline{A}$ is the mean of the real value.

3 Proposal of a New WaveNet-Based Architecture

In this section, the proposed new WaveNet-based Temporal CNN model is presented through its theoretical basis and mathematical formulas. We start by presenting the blocks of WaveNet architecture, which are firstly causal and dilated convolutions, secondly skip and residual connections, and thirdly local conditioning. Each block description is put in the context of MTS forecasting requirements, thus explaining the reason for using each block by our model. Section 3.2 describes, through a list of upgrades, how and why we made changes and arrange the aforementioned blocks to build our new Wavenet-TCNN for MTS forecasting; our model's architecture, its mathematical equations as well as a description of how our method performs are all revealed in this section.

3.1 Multivariate Temporal Dilated Residual Convolutional Neural Network

CNN are a subclass of neural networks that have been widely successful with image and video analysis. The main concept of CNN is to apply a convolution function to an input and a kernel, which are two matrices, in order to generate a feature map; this feature map is a transformation that reduces the input dimension and creates new matrices that describe the object features, ultimately achieving classification [55]. Unlike CNN, TCNN have the ability to handle sequences of data of non-fixed length and deliver sequence with the same length. In order to process long-term time-dependencies within data, TCNN are provided with two mechanisms called causal (dilated) convolutions and residual connections whose formalisms are presented in the following subsections. Local conditioning enables capturing the correlations between multivariate variables and is presented in Sect. 3.1.3. Section 3.2 describes our new Wavenet-TCNN (WTCNN) architecture and the analogy/dissimilarity with WaveNet structure.

3.1.1 Causal and Dilated Convolutions

Time series forecasting means that predicting future states at time t should only rely on past values $x^{(1)}, \ldots, x^{(t-1)}, x^{(t)}$; this defines what is called the "causal" characteristic that structures RNN. In the other hand, in order for CNN to capture long-term time dependencies, a dilation operation is applied allowing to expand the receptive field of the convolution and consisting of enlarging the kernel by introducing holes (trous hence the name of atrous convolutions) or blank spaces between the kernel elements [56, 57]. The dilation extent is determined by a hyperparameter d called the dilation rate. Figure 3 depicts a convolution with different dilation rates (3.a), as well as the gradual transformation of the receptive field through layers on a multi-layer one-dimensional dilated CNN, with a dilation rate equal 2 to the power of layer index (3.b). Dilation differs from zero padding in that the kernel size is augmented by the dilation, while in zero padding the kernel is identical, but elements with zero values are added around the input.

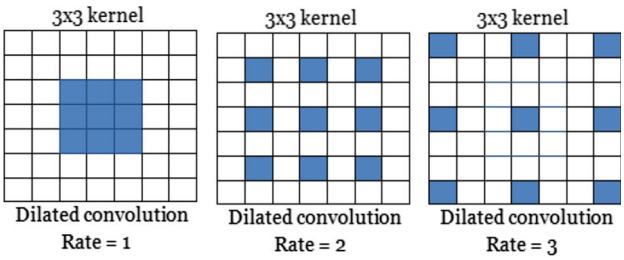
The dilated convolution function noted as $*_d$ is defined as follow:

$$f(i) = (x *_d w)(i) = \sum_{j=0}^{k-1} x(i - d.j)w(j) \tag{15}$$

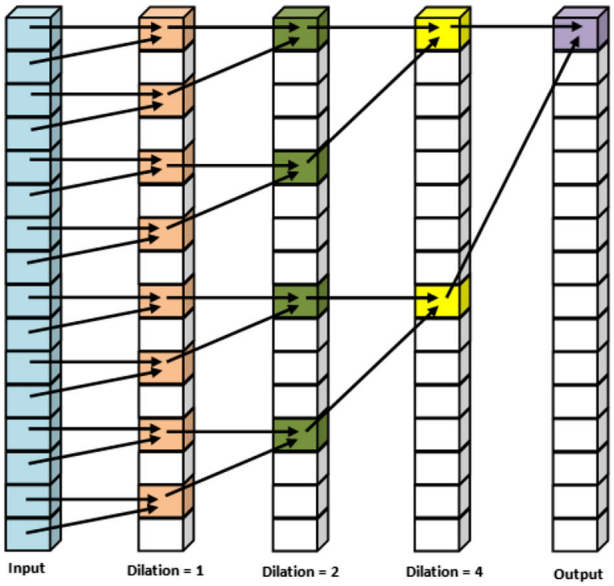
where d is the dilation rate, k is the kernel size, w is the kernel $\in \mathbb{R}^k$, and x the input $\in \mathbb{R}^n$.

3.1.2 Residual and Skip Connections

In order to learn long-term dependencies in time series data, CNN need to stack a high number of layers. However, with deeper networks, training becomes uneasy or even a hard task; this is when the network performance starts to degrade. This problem has been first highlighted by He et al. [58] where the authors introduced the residual block that allows the network to learn the identity function through what is called the residual connection or identity shortcut shown in Fig. 4. In fact, the authors demonstrated that while the network goes deeper, the training error reaches a minimum for a given number of layers n, then increases with the rest of the pile. Allowing the network to take a shortcut equal to the identity function enables it to keep the performance of n layers or to enhance it. In other words, with the extra stacked layers, the network will learn something or will do nothing. Learning the identity function can be hard to achieve for neural networks as nonlinear transformations are involved. While a plain network tries to learn the output function $H(x)$, the layers in a residual block aim to learn the residual one: $F(x) = H(x) - x$. Thereby, the network learns how to switch the layers weights to zero when it is needed, avoiding the vanishing/exploding problem.



(a) Convolution with different dilation rate [55]



(b) Multi-layer one-dimensional dilated CNN. Dilation rate equal 2 to the power of layer index.

Fig. 3 **a** Convolution with different dilation rate [56]. **b** Multi-layer one-dimensional dilated CNN. Dilation rate equal 2 to the power of layer index

Skip connections are similar to residual connections as they are both shortcuts added into the model architecture. With skip connections, each residual block can also push its output directly to the model's last node, where it is combined with the outputs from all the other levels as well as with the single final output that passed through all the blocks transformations. Using skip connections with time series data allows bringing earlier layers features to be considered with and without the model nonlinear deformation, making the model aware of all the data trends and seasonality, thus enhancing the network convergence.

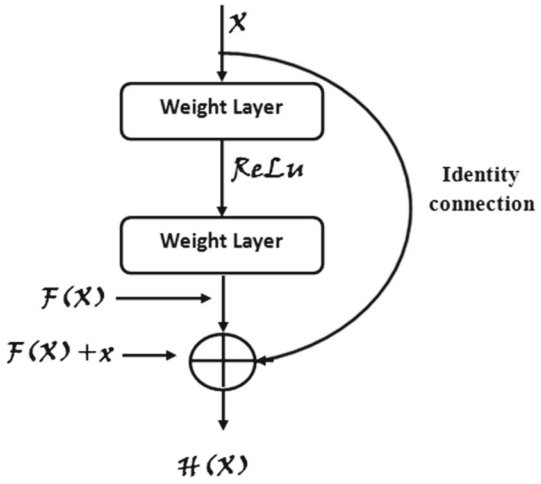


Fig. 4 Residual connection [58]

3.1.3 Local Conditioning for Multivariate Variables

Local conditioning allows predicting the value of variable x_i^{t+1} conditioned on the rest of dataset variables; that is to say that the model uses x_i^{t+1}'s own previous values as well as the previous values of all the other variables noted y_i^t, as a MTS variable (example: if $i = 1$ then $y_i = \{x_2, \ldots, x_m\}$). In our WTCNN architecture, detailed in Fig. 5, we split the variable y into m variables; however, for sake of simplicity in Eqs. (13) and (14), we keep it non-split. Equation (1) in Sect. 2.1 becomes:

$$f^{t+1}(x_i|y_i) = f(x_i^{t+1}|x_i^0, \ldots, x_i^t, y_i^0, \ldots, y_i^t) \tag{16}$$

$$f(x_i|y_i) = \prod_{t=0}^{T-1} f^t(x_i|y_i) \tag{17}$$

with $i \in \{1, \ldots, p\}$.

3.2 Our New WTCNN Model's Structure

The complete structure of our folded and unfolded WTCNN architecture is represented in Fig. 5. Our new model is distinguished by the following improvements:

Upgrade 1: The gated activation unit in the initial WaveNet architecture acts as the RNN's gates and allows the network to learn nonlinear dependencies. The drawback of the *ReLu* function used in [37] is the exploding output as it does not limit values, leading to the exploding gradient problem. Using tanh and sigmoid help preventing forecast deviation, while tanh calculate a transformed input and sigmoid let the unit remember or forget the data. In our architecture, we use a gated activation unit for the input and conditions. The conditioning operates via Eq. (15) below.

Upgrade 2: As opposed to [37], we added skip connections that link the output of each layer directly to the last node. Moreover, we parameterized these skip connections as opposed to plain ones in the original Wavenet architecture [28], by adding a sigmoid function in each one of them, thus increasing our model's capability to know if an early feature is useful as it is or it is more appropriate after nonlinear transformation.

Upgrade 3: As mentioned above, the main goal of skip connections is to push the input and conditions directly to the network output, thus capturing early data properties; in order to avoid further transformations induced by 1×1 convolution weights updates, we placed the skip connection before the 1×1 convolution, leaving the latter only the role of feature map size adjustment [see Eq. (19)].

Upgrade 4: Input and conditions' normalization is an important step as it helps speeding up the training. We use Group normalization (GN) instead of batch normalization used by [37]. GN was first introduced by Wu and He in 2018 [59], having being distinguished by its results stability and its independence from the batch size.

Upgrade 5: We added a new filter component to clear seasonality from time series which can affect the quality of predictions.

1×1 convolutions are used in order to match the residual block input feature map to the output one as well as to reduce the output dimensionality before passing it through a dense layer with one final output node to compute a single forecasted value.

To predict x_i^{t+1}, causality is equally applied to x and y so that the receptive field contains only $x_i^0, \ldots, x_i^t$ and $y_i^0, \ldots, y_i^t$. For the conditions dilated convolutions, and as in [37], we used $1 \times k$ convolution instead of 1×1 convolution used in the original WaveNet architecture. In fact, the receptive field $\boldsymbol{r}$ is correlated with the filter size $\boldsymbol{k}$ such as: $r = 2^{L-1}k$, where $\boldsymbol{L}$ is the number of layers. Therefore, by using $1 \times k$ convolution instead of 1×1 convolution, we help the model learn dependencies faster and more efficiently.

The mathematics behind our model is explained by the equations below. Table 1 lists the symbols used in our model's equations.

To simplify the working process of our model, we would say that in the first layer, the input and conditions are normalized with GN before going through dilated convolutions and nonlinear functions with tanh and sigmoid. The sum (element-wise multiplication) of these two transformations consists of the layer early property that is directly pushed to the output through a skipped connection. The same value continue its journey within the residual layer; its size is adjusted with a 1×1 convolution before meeting the input and conditions through residual connections, which constitutes the layer output. These operations are repeated for the other layers, until the final layer provide its output. This output is

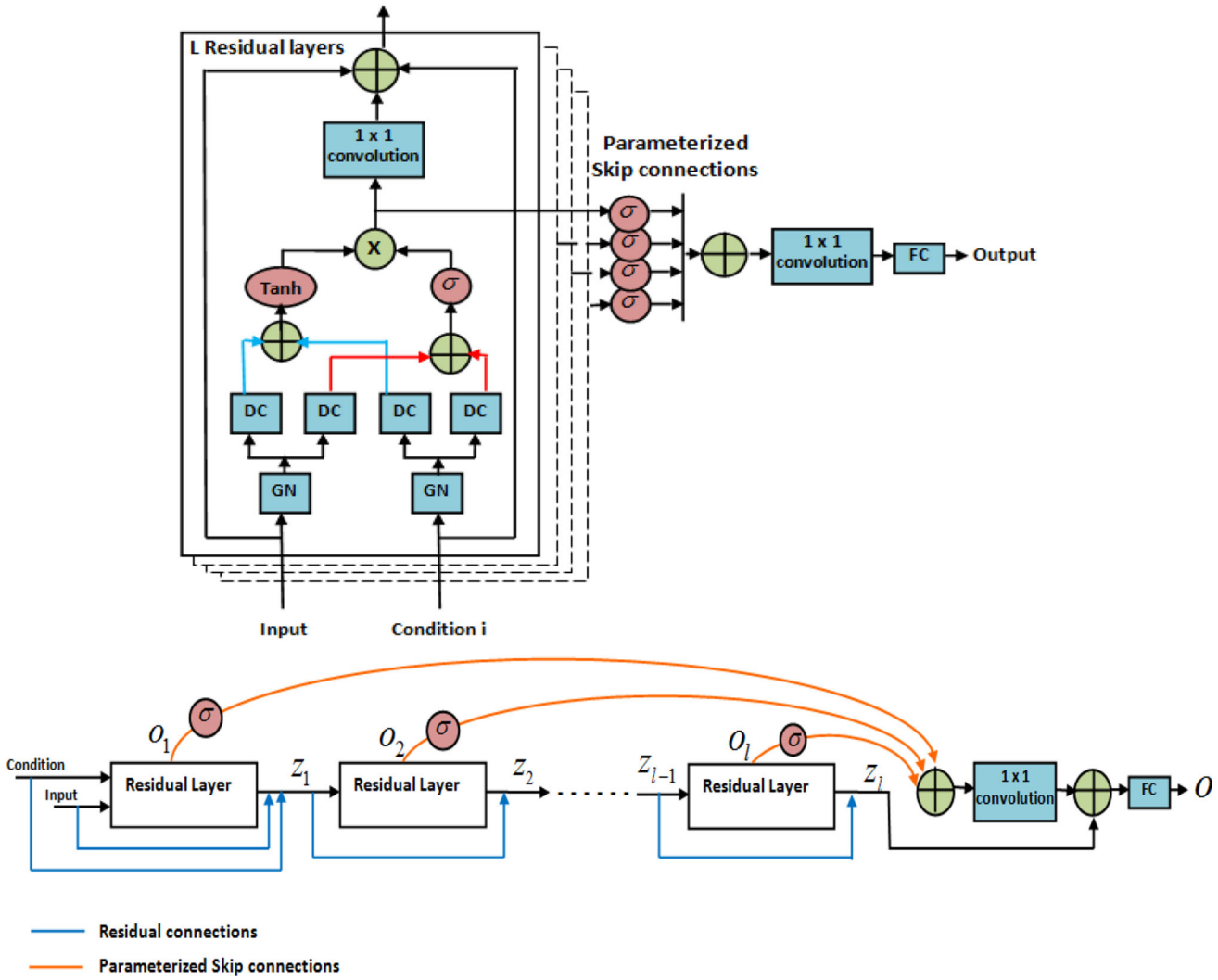


Fig. 5 Our folded (T) and unfolded (L) WTCNN architecture

summed with the sigmoid transformation of all the layers properties obtained earlier (after size adjustment to ensure that they have the same number of channels). The result passes through a dense layer to provide the final network output: the forecasted time series variable.

Each residual layer has one input (except for the first layer which has the input and conditions) which it uses twice—before and after nonlinear transformations—and two outputs, one towards the next residual layer and the other to the final output.

The number of dilated convolutions varies according to the layer order. In the first layer, $(2\times(M+1))$ dilated convolutions are executed, M being the number of conditions. In the rest of stacked layer, only two dilated convolutions are placed before tanh and sigmoid function. In every layer, all dilated convolutions are performed in parallel, which contribute to speed up training.

At training time, the inputs/outputs of the multivariate training dataset are fed into the model to set the weights' values that minimize the cost function. At testing time, a one-hour ahead forecasting $x^{(t)}$ is performed using its own past values $x^{(t-r)}, \ldots, x^{(t-1)}$, and past values of the conditions $c_i^{(t-r)}, \ldots, c_i^{(t-1)}$, with r being the dilated convolution receptive field. The output of the network is a vector of predicted values $X_p = \left[x^1, \ldots, x^n\right]$ where n is the size of testing dataset.

Finally, it is essential to note that nonlinear dependencies in MTS can only be learned if both the filter width and the number of layers is higher than 1. It is recommended to set the number of filters low ~ 1 in order to avoid an explosion in the number of parameters [37]. The number of layers is correlated with the receptive field as specified earlier.

For the first layer:

Table 1 Symbols used in our model's equations

$*$	Convolution operator
$\odot$	Element-wise multiplication operator
σ	Sigmoid function
DC	Dilated convolution
GN	Group normalization
X	Input
c_j	j thcondition
o_l	l thoutput of the gated activation unit of layer l
z_l	l thlayer output that goes as input of the next layer
C_{1x1}	1×1 convolution
FC	Fully connected or dense layer
$w_{f,l}$	Dilated convolution weights of the lth layer's input, preceding tanh activation
$w_{g,l}$	Dilated convolution weights of the $l$thlayer's input, preceding sigmoid activation
v_{f_j}	Dilated convolution weights of the first layer's $j$thcondition, preceding tanh activation function
v_{g_j}	Dilated convolution weights of the first layer's $j$thcondition, preceding sigmoid activation function

$$o_1 = \tanh(w_{f,1} * GN(X) + \sum_j v_{f_j} * GN(c_j)) \odot \sigma(w_{g,1} * GN(X) + \sum_j v_{g_j} * GN(c_j)) \tag{18}$$

$$z_1 = C_{1x1} o_1 + X + \sum_j c_j \tag{19}$$

For the other layers (l from 2 to L):

$$o_l = \tanh(w_{f,l} * z_{l-1}) \odot \sigma(w_{g,l} * z_{l-1}) \tag{20}$$

$$z_l = C_{1x1} o_l + z_{l-1} \tag{21}$$

For the network output:

$$o = FC(C_{1x1}(\sum_{l=1}^{L} \sigma(o_l)) + z_L) \tag{22}$$

4 First Elements of Comparison

In this section, we present elements of our first analysis of similarities and dissimilarities between the experimented and proposed NN architectures based on their internal structure and type of model. Table 2 lists the time series analysis requirements and shows how each type of the studied NN model addresses these requirements.

The similarity between LSTM and GRU is very notable as the main role of their different gates is to help the model determine which past information is to be remembered or to be forgotten. This common feature allows keeping past data values through time and thereby solving the vanishing or exploding gradient problem. However, they differ in their internal topology and the number of cells, and thus in the data flow control:

Table 2 Time series analysis requirements

Time series requirement/model	LSTM/GRU	WTCNN
Using only past data to predict future data	Unidirectional LSTM/GRU	Causal convolutions
Capturing long term-dependencies	Gating mechanism + Peephole connections	Dilated convolutions + gated activation unit
Catching time series trends and seasonality	–	Skip connections (parameterized in our model)
Dealing with vanishing/exploding gradient problem	Gating mechanism	Residual blocks + output limiting activation function such as tanh

- Memory state and hidden state: to transfer information, LSTM is provided with separate types of cells: memory or cell state $\boldsymbol{c}$ and hidden state $\boldsymbol{h}$. The first allows remembering information for a long time, while the latter is a transformation of $\boldsymbol{c}$ that is combined with the next time iteration input. In GRU, these two states are merged into the hidden state. In total, each LSTM unit has three inputs (x_t, c_{t-1} and h_{t-1}) and two outputs (c_t and h_t), while GRU has only two inputs (x_t and h_{t-1}) and one output (h_t).
- The state exposure: while LSTM unit defines how much output data to expose through the output gate, GRU unit pushes the whole state toward its gates without any filter.

From these first analysis elements, we conclude that GRU networks are simpler and faster to train than LSTM, as they use less cells, less training parameters and thus less computation effort and time. However, GRU are not capable of controlling data flow efficiently as LSTM do due to the elimination of cells that keep track of intermediate data values. Hence, state-of-the-art research comparison has not concluded which one would deliver better results [7, 14–16].

With regard to WTCNN, they have the following advantages:

- WTCNN require less memory as they do not need to store intermediate results of cells as LSTM and GRU do with their numerous gates. Moreover, as a variant of CNN,

Table 3 LSTM-GRU-WTCNN comparison

	LSTM	GRU	WTCNN
Structure	Gated RNN (with 3 gates)	Gated RNN (with 2 gates)	Dilated causal CNN with residual and skip connections
Flexibility	Not flexible	Not flexible	Very flexible: Different dilation factors and filter sizes, parameterized residual and skip connections
Memory and computation requirement	High	Medium (less gates than LSTM)	Low (no intermediate results storing and parameter sharing)
Training process	Sequential	Sequential	Parallel
Training and convergence speed	Low	Medium	High (thanks to parallelism)
Data flow control	High (through gates)	Medium (through gates)	High (through gated activation unit, residual and skip connections)
Robustness to input transformation	Low	Low	High
Stability	Acceptable	Acceptable	High
Depth	Limited	Limited	Not limited

WTCNN operate on the principle of parameters sharing through one kernel sliding.

- WTCNN operations are processed in parallel by taking advantage of the kernel sliding and parameter sharing characteristics and are therefore faster than RNN.
- Thanks to the dilation mechanism, WTCNN have variable output size depending on the dilation rate, thus adding more flexibility to this type of architecture. Moreover, the adjustable filter size allows better control of the model's structure.
- Residual connections allow constructing very deep networks without suffering from low performances (vanishing/exploding gradient problem), making WTCNN more stable and richer than RNN [60]. Skip connections add more flexibility and increase the network convergence.
- WTCNN are robust to input distortion/transformation, which is inherited from the CNN convolutional principal.

From these first elements of LSTM-GRU-WTCNN comparison, summarized in Table 3, we can state that WTCNN are more promising for time series prediction problems as they show more signs of success than LSTM or GRU. The following fifth section aims at validating this assumption by concrete experimental results.

5 Experiments

This section is dedicated to experimental results. Prior to the empirical evaluation, a description of the used datasets is presented hereafter.

5.1 Dataset

The goal of the study carried by this paper is to assess the performances of our proposed WTCNN model leading to the best air quality prediction capabilities, in order to later on develop an air quality management system for the city of Marrakesh. Marrakech–Tensift–El Haouz region is located in the center of Morocco and covers an area of 31,160 km^2. Marrakech is a non-industrial area; however, vehicle exhaust emissions produce excessive quantities of invisible polluting gases.

In this paper, we use six multivariate datasets that contains the air quality data of three stations located in Marrakesh city (Jamaa El Fna (JMF), Mhamid and Dawdiat; Table 4 details the stations' description), which were hourly collected from June 1, 2009 to November 28, 2010 (JMF_1, Mhamid_1 and Dawdiat_1), and from June 18, 2015 to September 26, 2016 (JMF_2, Mhamid_2 and Dawdiat_2). The six datasets contain nine hourly recorded dependent variables: five pollutants (carbon monoxide (CO), nitrogen dioxide (NO2), ozone (O3), particulate matter (PM10) and sulfur dioxide (SO2)), humidity ratio (HR), temperature Celsius (TC), wind speed in m/s (WS) and solar radiation in W/m^2 (SR). Our goal is to predict the five pollutants' (CO, NO_2, O_3, PM_{10} and SO_2) value of a time t based on their own precedent values and the precedent values of the other 8 variables.

5.2 Experiments

5.2.1 Data Preparation and Experimental Choices

All the dataset has been split into training (75%) and testing (25%) sets to perform the 1-h ahead forecasting using walk-forward procedure (see Sect. 2.4.2). To impute missing data,

Table 4 Stations description

Station	Type	Latitude	Longitude
JMF	Urban	31.620254	7.988897
Mhamid	Urban	31.596289	8.043691
Dawdiat	Urban	31.653676	7.995688

Part of the used data is available in the figshare repository, https://figshare.com/s/6a8706f5f46adda9f5b5

values recorded 24 h ago have been used in order to preserve the causality of data.

Neural networks in general learn data dependencies using nonlinear activation functions, and therefore do not require heavy transformations to make time series data stationary like classic machine learning methods do. However, we did start by normalizing our datasets by scaling the features to the [0, 1] range; we also applied seasonal adjustment with differencing by 24. Next, in order to make our time series data fit a supervised learning use case, a transformation was needed to create a list of tuples $(x_1^{t-1}, x_2^{t-1}, \ldots, x_m^{t-1}, x_k^t, \ldots, x_l^t)$, where m is 9 (for 9 variables), k is 1 and l is 5 (for 5 outputs to predict). The tuples are then split into input $(x_1^{t-1}, x_2^{t-1}, \ldots, x_m^{t-1})$ and output $(x_k^t, \ldots, x_l^t)$ data. In addition, the models expect three-dimensional input data as (samples, timesteps, features), where samples are the number of tuples from the previous step.

For hyperparameters setting, we used grid search (see Sect. 2.4.1) for some of the parameters, adopted the tuning used in the previous literature work for some other parameters and calculated some others using formulas. Section 3.2 explains the working process of our WTCNN model and describes the correlation between some significant hyperparameters, which is an important introduction to this section of hyperparameters setting.

The batch size value is 100; Adam [61] is the optimizer used by our three models, with an initial learning rate equal to 0.001 and a reduction factor of 0.25 each 10 epochs.

For LSTM/GRU models, the number of layers L is chosen from [1, 3] range with neuron set N chosen from [50, 100, 150, 200] range via grid search. The final output layer provides a vector of five elements that predicts the next hour values of five pollutants (CO, NO_2, O_3, PM10 and SO_2). Dropout rate value is set from [0.1, 0.6] range. Gradient clipping [62] technique is used as well to push training convergence, with a clip norm value set from [0.2, 1] range. Glorot's uniform is chosen as the initialization method. As shown in Table 5, the parameters are set to $L = 3$, $N = [50, 100, 150]$, dropout rate $= 0.2$, clip value $= 0.5$.

For WTCNN model, the number of residual layers RL is chosen from [6, 8] range and the kernel size K is set from $[1 \times 2, \ldots, 1 \times 7]$ range via grid search. We use dilation factor d related to the layer index i as d equal 2 to the power of i or 2^i

Table 5 Models' parameters values

Model	LSTM/GRU	WTCNN
No. of layers	3	
No. of neurons	[50, 100, 150]	
No. of Residual layer RL		6
kernel size		2 or 3
No. of filter/DC		1
Dilation factor		2^i, i in [1..6]
No. DC/RL		18 for RL = 1, 2 for RL in [2..6]
Optimizer	Adam	
Loss function	mse	
Epochs	50	
Batch	100	
Learning rate	0.01	
Decay/10 epochs	0.25	
Dropout	0.2	N/A
Clip value	0.5	N/A

as used in [37] and [63]. As shown in Table 5, the parameters are set to RL = 6, $K = 1 \times 2$ or $K = 1 \times 3$ (or simply 2 or 3), Number of filter in the dilated convolutions $f^l = 1$ for $l =$ [1,…, RL] (as mentioned earlier in Sect. 3.2, the number of filter is kept low in order to avoid an explosion in the number of parameters). N/A means not applied.

We used a training time series of length 24 to 250 (using discrete values) while making sure that the WTCNN receptive field and LSTM and GRU timesteps are the same. This is important to give all the models the same past visibility.

For an input length (receptive filed or timesteps) $\boldsymbol{r}$ equal to 24, and according to the formula in Sect. 3.2 ($r = 2^{L-1}k$), the number of layers in each dilated convolution can be calculated as:

if $K = 2$ then $2^L = r$, L is $\approx$ to 5.
if $K = 3$ then $2^{L-1} = r/3$, L is $=$ to 4.

With both K and L higher than 1 (a prerequisite mentioned in Sect. 3.2), our model is therefore able to learn nonlinear dependencies in MTS.

For the first WTCNN residual layer, the number of parallel dilated convolutions No. DC is equal to 18 ($(2 \times (M + 1))$, M being the number of conditions as specified in Sect. 3.2, which is 8 in our use case); No. DC is equal to 2 for the rest of the layers.

We conducted extensive experiments to test the impacts of kernel size and timesteps on WTCNN performances. The results can be found in Sect. 5.2.2.1.

Table 6 Performances results (RMSE, RRSE and MAPE) of each model with six datasets

Dataset/model	LSTM			GRU			WTCNN		
	RMSE	RRSE	MAPE	RMSE	RRSE	MAPE	RMSE	RRSE	MAPE
JMF_1	0.036	24.554	16.552	0.031	21.281	8.356	0.009	15.854	0.991
JMF_2	0.035	24.551	16.471	0.030	21.265	8.229	0.008	15.845	0.985
Mhamid_1	0.037	24.592	16.605	0.032	21.293	8.378	0.011	15.891	0.994
Mhamid_2	0.036	24.552	16.654	0.031	21.284	8.421	0.009	15.867	1.023
Dawdiate_1	0.036	24.549	16.542	0.031	21.289	8.312	0.009	15.866	0.986
Dawdiate_2	0.035	24.553	16.489	0.030	21.275	8.293	0.008	15.839	0.975

Table 7 Performances results summary (RMSE, RRSE and MAPE) of each model

Model	RMSE	RRSE	MAPE	Accuracy
LSTM	0.036	24.554	16.552	83.5
GRU	0.031	21.281	8.356	91.7
WTCNN	0.009	15.854	0.991	99.0

We use Keras library running on Tensorflow as LSTM, GRU and causal dilated convolutions are implemented in this language. The experiments have been carried out on a computer PC with intel® Core™ i7-6800k-6 cores-3.4 Ghz with 16 GB RAM and GPU of NVIDIA GeForce GTX 1080. The algorithms are implemented in Python 3. We recorded and compared the best performances for each model.

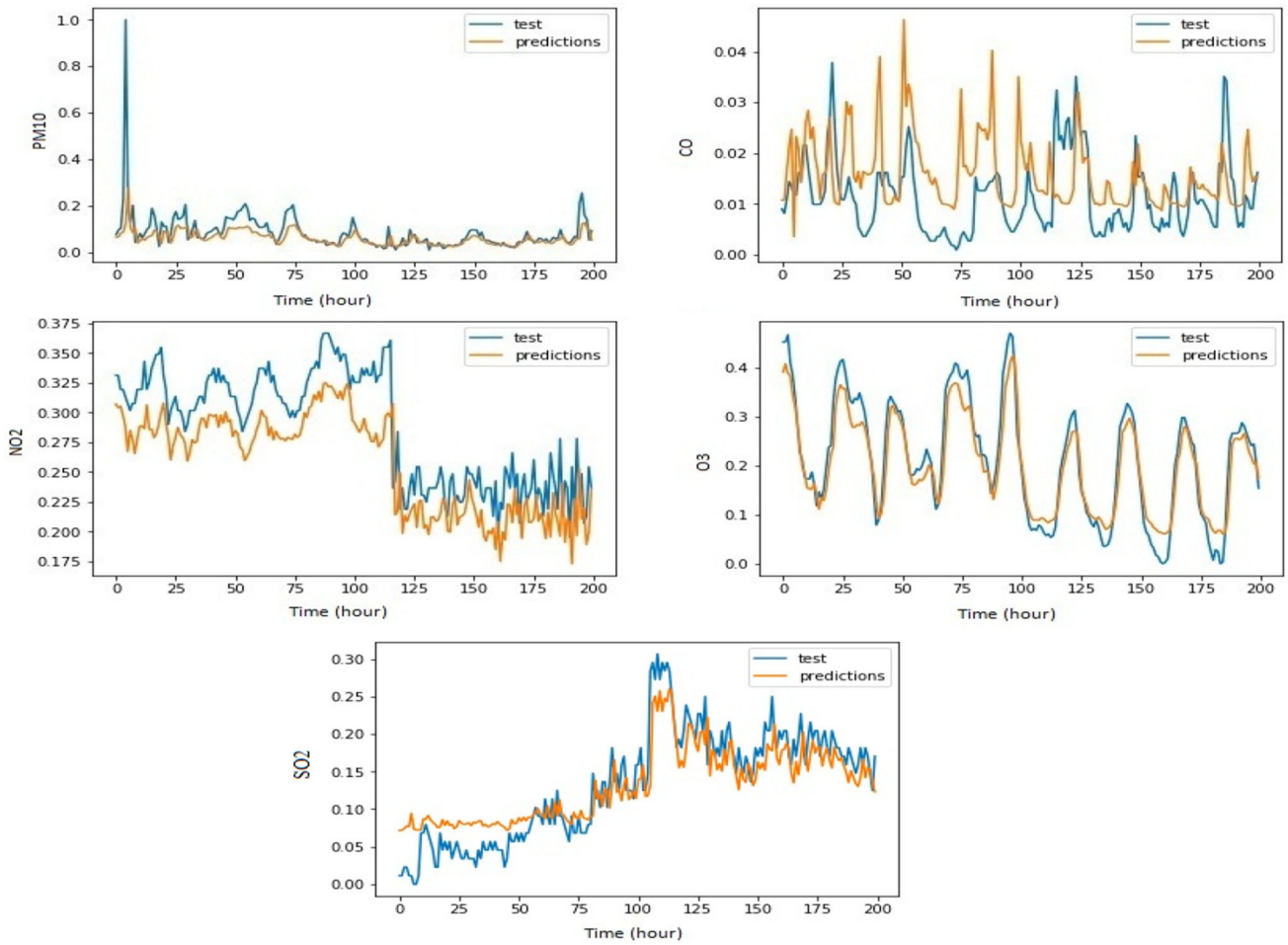


Fig. 6 The forecasting results of LSTM

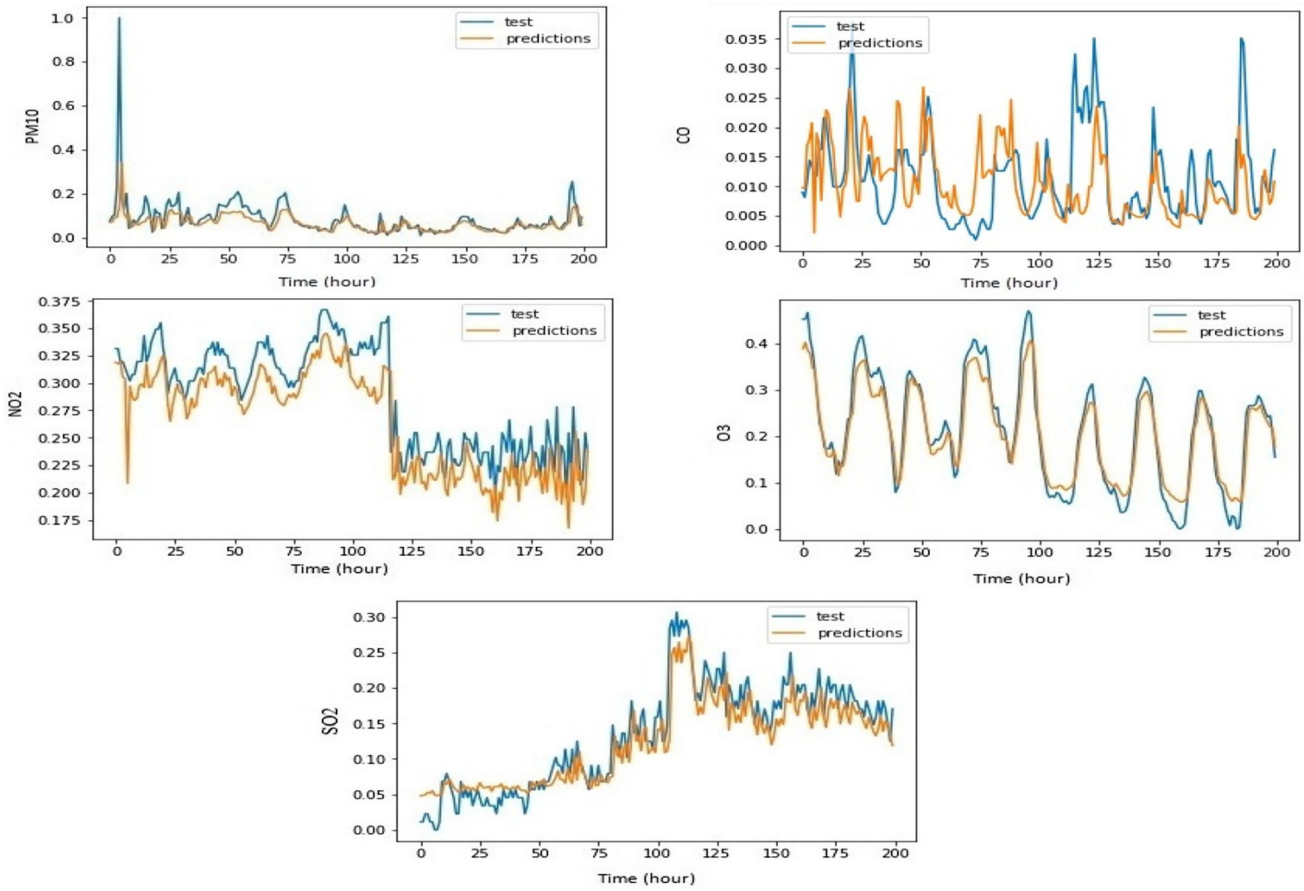


Fig. 7 The forecasting results of GRU

5.2.2 Experimental Results

This section is dedicated to experimental results that relate error metrics values, accuracy, forecasting results, convergence, performances for different timesteps and different filter size for WTCNN, and finally training speed. It is noted that results and optimal values do not change much from one dataset to another as the number of variables is the same and sizes are approximately similar in all the datasets.

Tables 6 and 7 summarize the performance results on RMSE, RRSE and MAPE with the six datasets and in summary, respectively—the lower value is better. For all datasets, the three metrics recorded their best values for WTCNN, followed by GRU. Our model lowers RMSE to 0.008 and MAPE to 0.975. It is clearly noticeable that WTCNN outperform LSTM and GRU by about 8–16%. GRU get better results than LSTM. In fact, while WTCNN reaches 99% accuracy, GRU and LSTM are capped at 91.7% and 83.5, respectively.

Figures 6, 7 and 8 show the forecasting curves of LSTM, GRU and WTCNN, respectively, for the five pollutants: CO, NO2, O3, PM10 and SO2, where WTCNN shows better generalization ability than LSTM and GRU. LSTM and GRU both got their best results for a history of length 24, while WTCNN performances were independent of history length (see Sect. 5.2.2.2 below).

Moreover, Fig. 9 depicts the convergence curve of LSTM, GRU and WTCNN showing that our model converges better and faster than the two RNN descendants. While TCNN quickly stabilizes around 175 epochs with a very low loss value—close to 0, GRU and LSTM curves seem to be slowly varying until they stabilize around 300 and 400 epochs, respectively, with higher loss value—0.004.

These experimental results confirm the analysis presented earlier in Sect. 4 proving that WTCNN is greater at capturing time series long-term dependencies and catching trends and seasonality.

Impacts of Timesteps and Kernel (or Filter) Size on WTCNN

Timesteps or input length The choice of past observations size can highly impact forecasting performances. Across the curves of accuracy regarding different timesteps plotted in Fig. 10, we note a constant convergence of WTCNN for different timesteps, while LSTM and GRU significantly lose precision as timesteps grow. LSTM accuracy reaches 5% after 50 timesteps, while GRU accuracy reaches 15% after 100 timesteps.

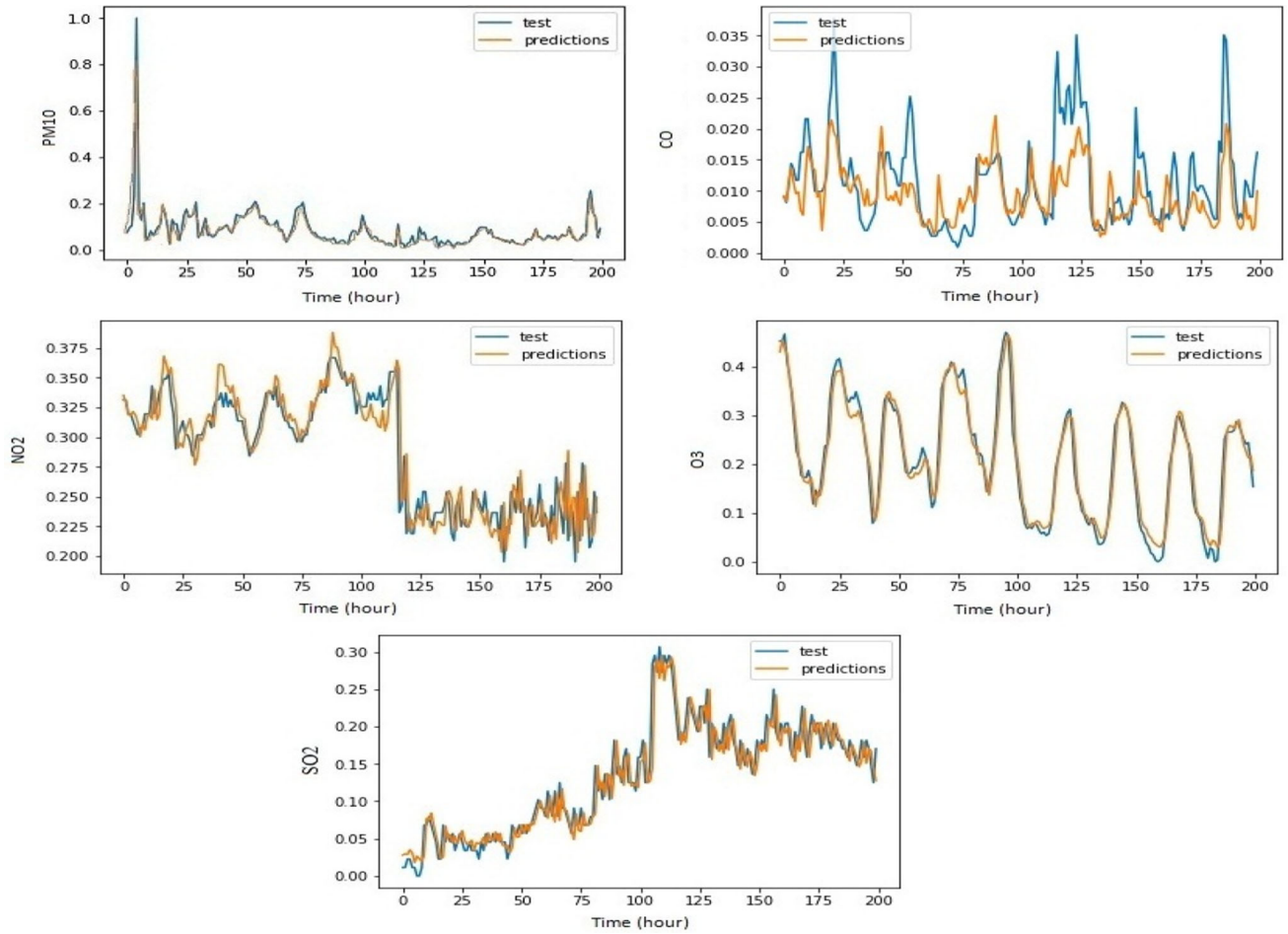


Fig. 8 The forecasting results of WTCNN. WTCNN outperform LSTM and GRU

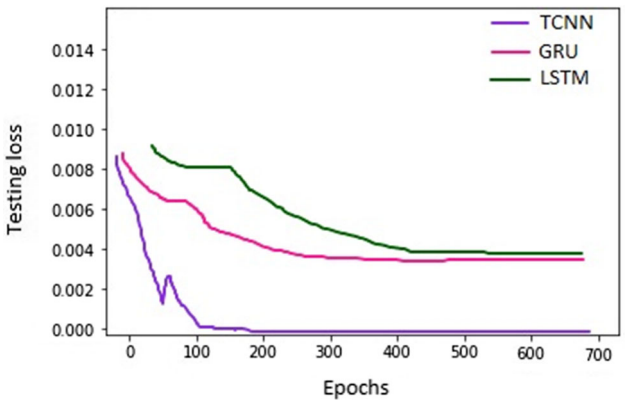


Fig. 9 Convergence of LSTM, GRU and WTCNN. WTCNN outperform LSTM and GRU

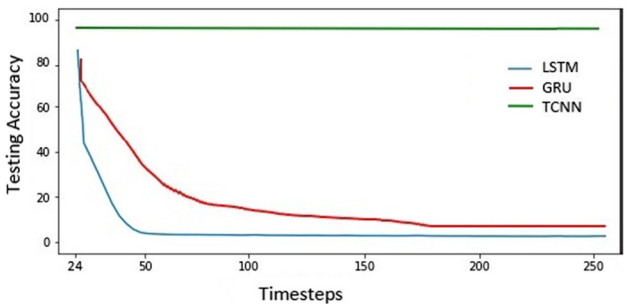


Fig. 10 Accuracy for different timesteps

These results denote WTCNN's ability to successfully consider a longer history than RNN models and thus indicate WTCNN capacity of handling big number of parameters without overfitting. This can be useful for phenomena whose understanding depends on very long history, such as some specific medical studies.

Filter size The considered task highly impacts the filter size choice; in fact, for image convolutions for example, large context matters and makes larger filter size more efficient; however, local context is more significant in language modeling, which makes smaller filter size more effective [63]. For MTS analysis, we found that small to medium filter sizes work well and pushes fast convergence. Figure 11 depicts

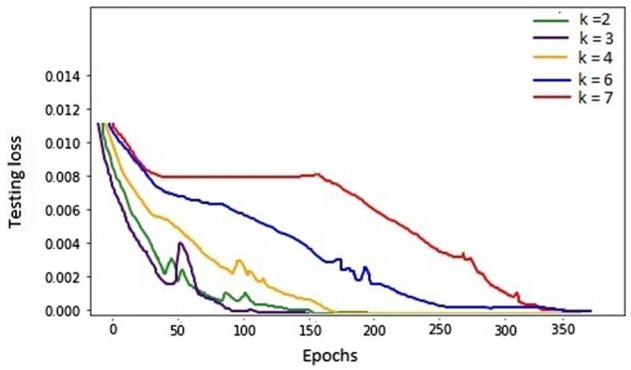


Fig. 11 WTCNN performances for different kernel sizes k. $k = [2, 3, 4, 6, 7]$

Table 8 Models speed comparison—s/epoch for each model

	LSTM	GRU	WTCNN
First three datasets	19.6	12.4	5.3
Last three datasets	15.5	9.8	4.13

different WTCNN performances for different kernel sizes $k = [2, 3, 4, 6, 7]$. A kernel size of 2 and 3 converges at 100 epochs, while a kernel size of 4, 6 and 7 converges at 170, 250 and 320, respectively.

Models Speed Table 8 shows the models speed comparison, which is represented by the time (in seconds) taken by each model to complete an epoch training (denoted s/epoch). "First three datasets" refers to the three stations hourly records from June 1, 2009 to November 28, 2010, while "Last three datasets" refers to the three stations hourly records from June 18, 2015 to September 26, 2016. We grouped datasets in two units as the number of variables and the size of time series is the same in the datasets of the same group. For both datasets groups, WTCNN shows the best results with [5.3, 4.13] s/epochs, GRU takes second place with [12.4, 9.8] s/epochs, while LSTM is found to be the slowest with [19.6, 15.5] s/epochs, thus confirming the analysis presented earlier in this paper. These values are obtained with timesteps values, kernel size and number of epochs that speed up convergence as detailed in previous sections.

In summary, for air quality multivariate time series forecasting with six datasets and different empirical conditions, our model delivered the best results. To validate these results, Table 9 indicates the p values of Wilcoxon signed-rank test between our model and RNN models. It shows that our model is significantly different from both RNN models. With a p value of 0.00025 for GRU and 0.00019 for LSTM, we can reject the null-hypothesis that our model is from the same distribution as GRU and LSTM. Moreover, we notice that the difference between GRU and LSTM is not significant ($p = 0.075$).

Table 9 The p values of Wilcoxon signed-rank test between our model and RNN models

	LSTM	GRU	WTCNN
LSTM	/	0.075	0.00019
GRU	0.075	/	0.00025
WTCNN	0.00019	0.00025	/

6 Discussions

6.1 Overfitting, Generalization and Convergence Speed

Overfitting is a common problem in neural networks that use small datasets or/and large set of hyperparameters. In our experiments, we used six datasets with more than nine thousands records in each dataset, which allows for a good generalization level. Through experiments, we noticed good accuracy for WTCNN with almost 100%; far behind, the RNN models record a significant deviation. In fact, LSTM and GRU suffer from overfitting due to vanishing gradient issue. As RNN models depth is very limited, hope is put on regularization techniques to improve accuracy. Unfortunately, the use of dropout and gradient clipping did not allow RNN models to outperform WTCNN. In WTCNN, skip connections, gated activation and inherited parallel processing nature helped pushing the network toward rapid convergence (see Fig. 9) and better generalization (see Figs. 6,7,8). As noted in Sect. 3.2, tanh and sigmoid functions helped preventing forecast deviation. Another important feature of our model is we added parameterized skip connections from each layer's output directly to the network last node, which we placed before the 1×1 convolution, thus capturing early data properties and greatly increasing the model efficiency. The parameterization allowed each skip connection to decide the level of importance of the property which has flowed through it, thereby providing the model with more processing intelligence. In other words, if a particular property is not of help for the forecast, the model can simply assign it a low weight. Furthermore, while expanding the receptive field of each layer, dilated convolutions fostered generalization with less need for parameters, computation and memory consumption.

WTCNN best accuracy was also due to its robustness to input transformation—as a CNN descendant. In fact, different starting time can be considered as a translation of time series; hence, the WTCNN best results when proceeding from training to testing. Besides, we encourage more examination of WTCNN for time series frequency variation.

Moreover, high data flow control in WTCNN with gated activation, parameterized skip and residual connections, as well as low computation and memory consumption, allowed

handling long data history, called time steps (see Fig. 10), and thus more accuracy in catching long-term dependencies.

6.2 How Deep Are Deep Neural Networks?

Since their first appearance, DNN have aroused a lot of interest and challenges as deeper means more capacity of features' discovery. However, DNN should not only refer to the number of hidden layers but also to their capacity of capturing abstract knowledge in more complex context and data correlation [64], at the risk of making them shallow deep DNN.

For TS analysis, ResNet is the deepest architecture with 11 layers [26]. For MTS forecasting, our 6 residual layers model enjoys shortcut connections—skip and residual connections—that makes the gradient directly reach the model's last node, thus minimizing the network loss induced by deep stacked layers' complexity. Moreover, as we placed parameterized skip connections before 1×1 convolution, the model's last node was delivered with plain early data properties without further transformations, which alleviates the model from weights updates' complexity.

In the other hand, dilated convolutions are not to be outdone when improving accuracy of DNN. In fact, while the network deepens, non-dilated convolutions coupled with pooling tend to reduce information representation in feature maps. Dilated convolutions are able to expand the receptive field without information loss, thus preserving a large temporal range as much as possible. In our experiments, we used $1 \times k$ convolutions which helped further enlarge the receptive field (see Sect. 3.2).

Furthermore, deep networks are faced with overfitting, demanding more regularization. We provide an element of solution to this problem by normalizing the input and conditions. Our model draws its strength from incorporating normalization into the architecture while using batch size-independent Group Normalization, which increases the model's stability.

It is worth noting that our datasets holds numerical values from not so many disparate sources. For more complex datasets, we shall use deeper networks with more regularization techniques.

7 Conclusion

Multivariate time series (MTS) data modeling has known great progress in recent years thanks to neural network use explosion. Highly accurate MTS forecasting can make decision-making process quicker and more effective for industrial and scientific fields. Consumers' habits analysis using MTS from multiples information sources in a sales department, real-time and multidimensional data processing for predictive maintenance service in a complex environment such as factory 4.0, or strongly correlated financial time series data forecasting are examples where MTS analysis is needed as part of a global business intelligence system. New challenges arise as more studies attempt to improve results delivered by univariate models.

In this paper, we have proposed a new multivariate temporal dilated residual convolutional neural networks structure adapted from the recent WaveNet architecture, to best suit MTS analysis requirements addressed as well in this paper. Our model aims to be a tailor-made solution to TS challenges addressed in Table 2, which are restricting analysis to past chronological data, capturing long-term dependencies, catching time series trends and seasonality, and finally dealing with vanishing/exploding gradient problem. Our model—WTCNN—consists of stacked residual causal dilated convolutions augmented with parameterized skip connections and group normalization that allow learning long-term dependencies and catching time series trends and seasonality; we have added conditioning to the model's first layer in order to retrieve multivariate data associations. We then have presented an in-depth comparison of three deep learning neural networks for MTS analysis: LSTM and GRU which are recurrent neural network model, along with WTCNN. First elements of comparison have been supported by the experimentation we have conducted on six real multi-sensor and MTS datasets for the task of urban air quality prediction in Marrakesh city. Results demonstrated WTCNN's higher performances and better capacities in handling long-term dependencies. Sections 4 and 5 have put forward elements to answer the following raised question: Why do WTCNN work better than LSTM/GRU? In fact, we can emphasize that:

- WTCNN and convolutional networks in general offer massive parallelism, which allow them to be faster than the sequential LSTM/GRU processes.
- WTCNN needs for memory is lower than LSTM/GRU needs as the former do not store intermediate results; this allows for fast training as less data storing means less time consumption
- Residual networks in WTCNN make it possible to develop and deepen the network without impairing its performances due to easy learning of the identity function. This ability allows for fast convergence and thus easy and efficient training
- Residual connections coupled with skip connection mechanism used by WTCNN in order to avoid vanishing/exploding gradient problem, compared with gating mechanism used by LSTM/GRU, is much more efficient with memory saving as it does not consume any additional cells. Moreover, skip connections allow constructing very deep neural networks and accessing broad range of history without having to deal with the problem of vanishing/exploding gradient, while LSTM/GRU architecture

capacity to handle very long sequences is limited, which make WTCNN more efficient
- TCNN offer more flexible structure; dilation mechanism factor and filter size are keys to long-term time dependencies learning. In fact, while LSTM/GRU units are rigid boxes, WTCNN can use different dilation factors and filter sizes allowing for more flexibility and adaptation of the model, as well as increasing its performances.

From the study led in this paper, we can conclude that parallel processing of residual dilation mechanism coupled with parameterized skip convolutions is more efficient that gating mechanism for multivariate time series forecasting. However, more issues need to be addressed in future work; first, different multivariate datasets should be experimented in order to study how our proposed architecture interacts with different dataset's characteristics. In fact, the problem's domain, sample size, time series length, training size, variables' varying frequency and existence of missing values can greatly affect a neural network's performances [11]. In a more complex big data applications context, the model's scalability will be put to the test; multivariate datasets having a large number of observed variables require dimension reduction, selection optimization or variables' hierarchization in order to exclude the less significant variables and minimize the problem complexity [65].

Second, the model's training and testing time becomes its Achilles' heel in a dynamic approach with great data velocity requiring high frequency modeling. To speed up a network processing, Mathieu et al. used fast Fourier transform-based convolutions [66], Ramachandran et al. proposed a fast generation method for WaveNet model which cache internal network states [67], while Winograd algorithm was explored by Lavin and Gray [68]. In our future work, more attention will be dedicated to this challenge by exploring these methods.

Third, as normalization is known to highly impact DNNs' learning convergence, especially to solve the problem of internal covariate shift [69], we used Group Normalization to normalize the network inputs. Other types of normalization exist such as z-normalization, layer normalization or instance normalization, which should be carefully explored.

Finally, multi-step time series forecasting, which involves predicting multiple future values, can pose additional challenges and is more difficult than single-step forecasting. In our future work, we aim to study and respond to the aforementioned limitations by performing more in-depth experiments.

References

1. Makridakis, S.; Wheel Wright, S.; Hyndman, R.J.: Forecasting: Methods and Applications, 3rd edn Wiley, New York (1998)
2. Braithwaite, I.; Zhang, S.; Kirkbride, J.B.; Osborn, D.P.J.; Hayes, J.F.: Air pollution (particulate matter) exposure and associations with depression, anxiety, bipolar, psychosis and suicide risk: a systematic review and meta-analysis. *Environ. Health Perspect. J.*, 2019. https://doi.org/10.1289/EHP4595
3. Vovk, V.G.: Universal forcasting algorithms. *Inf. Comput.* **96**, 245–277, 1992
4. Ma, J.; Ma, X.: A review of forecasting algorithms and energy management strategies for microgrids. *Syst. Sci. Control Eng. Open Access J.* **6**, 237–248, 2018
5. Makridakis, S.; Spiliotis, E.; Assimakopoulos, V.: Statistical and machine learning forecasting methods: concerns and ways forward. *PLoS ONE* **13**(3), 1–26, 2018
6. Gatys, L.; Ecker, A.; Bethge, M.: A neural algorithm of artistic style. arXiv:1508.06576. Accessed 10 Mar 2020 (2015)
7. Alom, M.Z.; Taha, T.M.; Yakopcic, C.; Westberg, S.; Sidike, P.; Nasrin, M.S.; Hasan, M.; Van Essen, B.C.; Awwal, A.A.; Asari, V.K.: A state-of-the-art survey on deep learning theory and architectures. *Electronics* **8**(3), 292, 2019
8. Van Veen, F.: The neural network zoo. https://www.asimovinstitute.org/neural-network-zoo/. Accessed 27 Oct 2019 (2016)
9. Brunton, S.; Kutz, J.: Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control. Cambridge University Press, Cambridge (2019)
10. Kaiser, L.; Gomez, A.N.; Shazeer, N.; Vaswani, A.; Parmar, N.; Jones, L.; Uszkoreit, J.: One model to learn them all. arXiv:1706.05137. Accessed 10 Mar 2020 (2017)
11. Ismail Fawaz, H.; Forestier, G.; Weber, J.; Idoumghar, L.; Muller, P.-A.: Deep learning for time series classification: a review. arXiv:1809.04356. Accessed 10 Mar 2020 (2018)
12. Montgomery, D.C.; Jennings, C.L.; Kulahc, M.: Introduction to Time Series Analysis and Forecasting. Wiley Series in Probability and Statistics. Wiley, Hoboken (2015)
13. Box, G.E.P.; Jenkins, G.I.: Series Analysis: Forecasting and Control, 5th edn Holden-Day, San Francisco (1976)
14. Greff, K.; Srivastava, R.K.; Koutnık, J.; Steunebrink, B.R.; Schmidhuber, J.: LSTM: a search space odyssey. CoRR. arXiv:1503.04069. Accessed 10 Mar 2020 (2015)
15. Chung, J.; Gulcehre, C.; Cho, K.; Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv:1412.3555. Accessed 10 Mar 2020 (2014)
16. Salehinejad, H.; Baarbe, J.; Sankar, S.; Barfett, J.; Colak, E.; Valaee, S.: Recent advances in recurrent neural networks. CoRR. arXiv:1801.01078 (2018)
17. Lea, C.; Flynn, M.D.; Vidal, R.; Reiter, A.; Hager, G.D.: Temporal convolutional networks for action segmentation and detection. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1003–1012, Hawai, USA, June (2017)
18. Chang, S.Y.; Li, B.; Simko, G.; Sainath, T.N.; Tripathi, A.; van den Oord, A.; Vinyals, O.: Temporal modeling using dilated convolution and gating for voice-activity-detection. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 5549–5553, Calgary, Canada, April (2018)
19. Kalchbrenner, N.; Grefenstette, E.; Blunsom, P.: A convolutional neural network for modelling sentences. arXiv:1404.2188. Accessed 10 Mar 2020 (2014)
20. Kalchbrenner, N.; Espeholt, L.; Simonyan, K.; van den Oord, A.; Graves, A.; Kavukcuoglu, K.: Neural machine translation in linear time. arXiv:1610.10099v2. Accessed 10 Mar 2020 (2017)
21. Tsay, R.S.: Multivariate Time Series Analysis: With R and Financial Applications. Wiley, Chichester (2014)
22. Zhou, P.-Y.; Chan, K.C.: A feature extraction method for multivariate time series classification using temporal patterns. In: Advances in Knowledge Discovery and Data Mining, vol. 9078, pp. 409–421. Springer, Berlin (2015)

23. Zheng, Y.; Liu, Q.; Chen, E.; Ge, Y.; Zhao, J.L.: Time series classification using multi-channels deep convolutional neural networks. *Web Age Inf. Manag.* **8485**, 298–310, 2014
24. Zheng, Y.; Liu, Q.; Chen, E.; Ge, Y.; Zhao, J.L.: Exploiting multi-channels deep convolutional neural networks for multivariate time series classification. *Front. Comput. Sci.* **10**(1), 96–112, 2016
25. Zhao, R.; Wang, D.; Yan, R.; Mao, K.; Shen, F.; Wang, J.: Machine health monitoring using local feature-based gated recurrent unit networks. *IEEE Trans. Ind. Electron.* **65**(2), 1539–1548, 2017
26. Wang, Z.; Yan, W.; Oates, T.: Time series classification from scratch with deep neural networks: a strong baseline. In: International Joint Conference on Neural Networks, pp. 1578–1585, Alaska, USA, May (2017)
27. Serrà, J.; Pascual, S.; Karatzoglou, A.: Towards a universal neural network encoder for time series. arXiv:1805.0390. Accessed 10 Mar 2020 (2018)
28. van den Oord, A.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.; Kavukcuoglu, K.: WaveNet: A Generative Model for Raw Audio. arXiv:1609.03499. Accessed 10 Mar 2020 (2016)
29. Keogh, E.; Mueen, A.: Curse of dimensionality. In: Encyclopedia of Machine Learning and Data Mining, pp. 314–315. Springer, Boston (2017)
30. Poggio, T.; Mhaskar, H.; Rosasco, L.; Miranda, B.; Liao, Q.: Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. *Int. J. Autom. Comput.* **14**(5), 503–519, 2017
31. Che, Z.; Purushotham, S.; Cho, K.; Sontag, D.; Liu, Y.: Recurrent neural networks for multivariate time series with missing values. *Sci. Rep.* **8**(1), 6085, 2018
32. Filonov, P.; Lavrentyev, A.; Vorontsov, A.: Multivariate industrial time series with cyber-attack simulation: fault detection using an LSTM-based predictive data model. In: NIPS Time Series Workshop. arXiv:1612.06676. Accessed 1 Jul 2020 (2016)
33. Pascanu, R.; Mikolov, T.; Bengio, Y.: On the difficulty of training recurrent neural networks. IN: International Conference on Machine Learning, pp. 1310–1318, Atlanta, USA, June (2013)
34. Fazle, K.; Somshubra, M.; Houshang, D.; Samuel, H.: Multivariate LSTM-FCNs for time series classification. arXiv:1801.04503. Accessed 1 Jul 2020 (2018)
35. Yazdanbakhsh, O.; Dick, S.: Multivariate time series classification using dilated convolutional neural network. arXiv:1905.01697. Accessed 1 Jul 2020 (2019)
36. Kechyn, G.; Yu, L.; Zang, Y.; Kechyn, S.: Sales forecasting using WaveNet within the framework of the Kaggle competition. arXiv:1803.04037. Accessed 1 Jul 2020 (2018)
37. Borovykh, A.; Bohte, S.; Oosterlee, C.W.: Dilated convolutional neural networks for time series forecasting. *J. Comput. Finance* **22**(4), 73–101, 2017
38. Hochreiter, S.; Schmidhuber, J.: Long short-term memory. *Neural Comput.* **9**(8), 1735–1780, 1997
39. Gers, F.; Schmidhuber, J.; Cummins, F.: Learning to forget: continual prediction with LSTM. *Neural Comput.* **12**(10), 2451–2471, 2000
40. Gers, F.A.; Schmidhuber, J.: Recurrent nets that time and count. In: Neural Networks. IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN), pp. 189–194, July (2000)
41. Sundermeyer, M.; Schluter, R.; Ney, H.: LSTM neural networks for language modeling. In: INTERSPEECH, pp. 194–197, Portland, USA, September (2012)
42. Doetsch, P.; Kozielski, M.; Ney, H.: Fast and robust training of recurrent neural networks for offline handwriting recognition. In: 14th International Conference on Frontiers in Handwriting Recognition (ICFHR), pp. 279–284, Crete, Greece, September (2014)
43. Graves, A.; Jaitly, N.; Mohamed, A.-R.: Hybrid speech recognition with deep bidirectional LSTM. In: IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), pp. 273–278 (2013)
44. Bakker, B.: Reinforcement learning with long short-term memory. *Adv. Neural. Inf. Process. Syst.* **14**, 1475–1482, 2002
45. Wang, Y.; Huang, M.; Zhu, X.; Zhao, L.: Attention-based LSTM for aspect-level sentiment classification. In: Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 606–615, Austin, Texas, USA, September (2016)
46. Alahi, A.; Goel, K.; Ramanathan, V.; Robicquet, A.; Fei-Fei, L.; Savarese, S.: Social LSTM: human trajectory prediction in crowded spaces. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 961–971, Nevada, USA, June (2016)
47. Eck, D.; Schmidhuber, J.: Finding temporal structure in music: blues improvisation with LSTM recurrent networks. In: IEEE Workshop on Neural Networks for Signal Processing, pp. 747–756, New York, USA, September (2002)
48. Cho, K.; van Merrienboer, B.; Bahdanau, D.; Bengio, Y.: On the properties of neural machine translation: encoder–decoder approaches. arXiv:1409.1259. Accessed 10 Mar 2020 (2014)
49. Rana, R.: Gated recurrent unit (GRU) for emotion classification from noisy speech. arXiv:1612.07778. Accessed 10 Mar 2020 (2016)
50. Zhao, B.; Lu, H.; Chen, S.; Liu, J.; Wu, D.: Convolutional neural networks for time series classification. *Syst. Eng. Electron.* **28**(1), 162–169, 2017
51. Andermatt, S.; Pezold, S.; Cattin, P.: Multi-dimensional gated recurrent units for the segmentation of biomedical 3D-data. In: International Workshop on Deep Learning in Medical Image Analysis (DLMIA), pp. 142–151, Athens, Greece, October (2016)
52. Ravanelli, M.; Brakel, P.; Omologo, M.; Bengio, Y.: Light gated recurrent units for speech recognition. arXiv:1803.10225 (2018)
53. Wu, Z.; King, S.: Investigating gated recurrent neural networks for speech synthesis. In: IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), pp. 5140–5144, Shanghai, China, March (2016)
54. Nielsen, M.: Neural networks and deep learning. Free book. http://neuralnetworksanddeeplearning.com. Accessed 10 Mar 2020 (2018)
55. Dumoulin, V.; Visin, F.: A guide to convolution arithmetic for deep learning. arXiv:1603.07285. Accessed 10 Mar 2020 (2016)
56. Yu, F.; Koltun, V.: Multi-scale context aggregation by dilated convolutions. arXiv:1511.07122 (2015)
57. Chen, L.C.; Papandreou, G.; Kokkinos, I.; Murphy, K.; Yuille, A.L.: Semantic image segmentation with deep convolutional nets and fully connected CRFs. arXiv:1412.7062 (2015)
58. He, K.; Zhang, X.; Ren, S.; Sun, J.: Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR, Las Vegas, NV, USA, pp. 770–778, 27–30 June (2016)
59. Wu, Y.; He, K.: Group normalization. arXiv:1803.08494 (2018)
60. Gasparin, A.; Lukovic, S.; Alippi, C.: Deep learning for time series forecasting: the electric load case. arXiv:1907.09207 (2019)
61. Kingma, D.; Adam, J.B.: A method for stochastic optimization. arXiv:1412.6980 (2014)
62. Pascanu, R.; Mikolov, T.; Bengio, T.: On the difficulty of training recurrent neural networks. In: 30th International Conference on Machine Learning, ICML, Atlanta, GA, USA, pp. 1310–1318, 16–21 June (2013)
63. Bai, S.; Kolter, J.; Koltun, V.: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv:1803.01271. Accessed 10 Mar 2020 (2018)
64. Marcus, G.: Deep learning: a critical appraisal. arXiv:1801.00631. Accessed 1 Jul 2020 (2018)
65. Hmamouche, Y.; Przymus, P.M.; Alouaoui, H.; Casali, A.; Lakhal, L.: Large multivariate time series forecasting: survey on methods

and scalability. Utilizing big data paradigms for business intelligence, pp. 170–197. IGI Global (2019)
66. Mathieu, M.; Henaff, M.; LeCun, Y.: Fast training of convolutional networks through ffts. arXiv:1312.5851. Accessed 1 Jul 2020 (2013)
67. Ramachandran, P.; Le Paine, T.; Khorrami, P.; Babaeizadeh, M.; Chang, S.; Zhang, Y.; Hasegawa-Johnson, M.A.; Campbell, R.; Huang, T.: Fast generation for convolutional autoregressive models. arXiv:1704.06001. Accessed 1 Jul 2020 (2017)
68. Lavin, A.; Gray, S.: Fast algorithms for convolutional neural networks. arXiv:1509.09308. Accessed 1 Jul 2020 (2015)
69. Ioffe, S.; Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning, vol. 37, pp. 448–456, Lille, France, July (2015)