Two deep learning approaches to forecasting disaggregated freight flows: convolutional and encoder–decoder recurrent

Isidro Lloret[1] · José A. Troyano[2] · Fernando Enríquez[2] · Juan-José González-de-la-Rosa[3]

Abstract
Time series forecasting of disaggregated freight flow is a key issue in decision-making by port authorities. For this purpose and to test new deep learning techniques we have selected seven time series of imported goods from Morocco to Spain through the port of Algeciras, and we have tested two forecasting deep neural networks models: dilated causal convolutional and encoder–decoder recurrent. We have experimented with four different granularities for each series: quarterly, monthly, weekly and daily. The results show that our neural network models can manage these raw series without first removing seasonality or trend. We also highlight the ability of neural models to work with a fixed input size of one year, being able to make good predictions using the same input size for all granularities. The two deep learning models have globally improved the benchmarks of the M4 Competition of forecasting. Each neural network model obtains its best results under different circumstances: the recurrent one with daily granularity and intermittent series, and the convolutional one with weekly and monthly granularities.

Keywords Disaggregated freight transport · Time-series forecasting · Machine learning · Deep learning · Dilated convolutional neural network · Encoder–decoder recurrent neural network

1 Introduction

Seaports are major infrastructure elements of transportation networks in supply chains. Port authorities are responsible for making decisions regarding these infrastructures and their facilities in light of freight transport demand forecasts (Systematics 1997; Chopra 2019). Disaggregated freight flow allows for a segmented analysis of future freight transport demand in order to help prioritise transport investments, transport policy development and the growth of the industry of logistics (Havenga 2013).

The aim of this work is to evaluate the usefulness of machine learning techniques for the prediction of the flow of goods through seaports. Machine learning techniques base their predictions on inferred models from data. There are multiple machine learning applications in many domains, as healthcare, education and sports, but has only recently been applied to commercial transactions, so its application to the study of international trade trends is limited (Batarseh et al. 2019).

We are especially interested in deep learning models. Deep learning is a specific kind of machine learning that allows a machine to learn complicated concepts by building them from simpler concepts. The different components of this type of model are organized into a deep and layered structure (Goodfellow et al. 2016).

✉ Isidro Lloret
isidro.lloret@uca.es

José A. Troyano
troyano@us.es

Fernando Enríquez
fenros@us.es

Juan-José González-de-la-Rosa
juanjose.delarosa@uca.es

1 Department of Computer Engineering, Engineering School of Algeciras, University of Cádiz, S/N Ramón Puyol Av., 11202 Algeciras, Spain

2 Department of Languages and Computer Systems, Computer Engineering School, University of Seville, S/N Reina Mercedes Av., 41012 Sevilla, Spain

3 Research Unit in Computational Instrumentation and Industrial Electronics PAIDI-TIC-168, Area of Electronics, Engineering School of Algeciras, University of Cádiz, S/N Ramón Puyol Av., 11202 Algeciras, Spain

Our experiments seek to give insights into the applicability of the new deep learning techniques to the forecasting of the flow of goods. Specifically, we have used two deep learning models based on the following techniques:

- *Convolutional neural networks* A type of artificial neural network in which neurons are organized into hierarchical blocks, each of which specializes in enhancing certain features of the input data.
- *Recurrent neural networks* Another type of neural network composed of nodes that, in addition to receiving information from the previous layer, can also obtain it from themselves, thus allowing the sequential processing of data.

The data we have selected for the experimentation consists of seven disaggregated time series of goods importations from Morocco to Spain through the port of Algeciras in the period 2000–2016. The series present a great variety of seasonality and trend, which will allow us to evaluate the behavior of our forecasting techniques under very different conditions. We have also resampled the original series into four granularities (quarterly, monthly, weekly and daily), each associated with a specific forecast horizon (8 quarters, 18 months, 13 weeks, and 14 days), thus covering short, medium and long term forecasting. With this variety we wanted to avoid one of the most common shortcomings of many time series forecasting works, in which conclusions are based on a few or a single time series, as well as on one-step or short-term predictions (Makridakis et al. 2018). We consider this wide range of experiments to be an additional guarantee for the conclusions we draw from our results.

Our experiments show that both deep learning models globally beat the overall weighted average (OWA) accuracy measure of all baselines. We use as baselines all the methods included in the M4 Competition benchmarks. By levels of granularity, the recurrent model obtain the best results for daily data and the convolutional model for weekly data.

This paper is organized in six sections. The second section deals with defining the specific details of our forecasting task. Section 3 describes the two deep learning models we have used in our experiments. In Sect. 4 we present our experimental methodology. We discuss the results in Sect. 5 and extract conclusions in Sect. 6.

2 The task

In this section we describe the time series forecasting task. First, we present the dataset used in our experiments, and then we explain the machine learning approach to forecasting.

2.1 Data description and forecast horizons

Our dataset consists of seven disaggregated time series of goods importations from Morocco to Spain through the port of Algeciras in the period 2000–2016, corresponding to the European tariff chapters shown in Table 1.

The seven series represent a great variety of seasonality and trend, which will allow us to evaluate the behavior of our forecasting techniques under very different conditions.

Maritime traffic time series are updated in the databases with the current timestamp for every port operation (loading or unloading). From the original series, which have a one-minute granularity, we have resampled them into four levels of granularity (quarterly, monthly, weekly and daily), and for each we have considered a multi-step forecast horizon as defined in the M4 Competition (Makridakis et al. 2020). This way we obtain a set of 28 series (available at, Xxxxx 2020) for short, medium and long term forecasting. And therefore we can support very different types of decisions, like strategic, planning and operational decisions.

Table 2 shows for each granularity level, the forecast horizon, the seasonality period and the time interval of the series. Daily series are the ones presenting two types of seasonality (annual and weekly). For these type of series we have shortened the time interval to the last two and a half years, which is long enough for our prediction purposes.

Figure 1 shows the seven series with monthly granularity. Validation and test periods are plotted in green and orange, respectively. We describe how we have done the validation and test splitting in Sect. 2.2. As we can see, there are highly seasonal series (like chapter 7), intermittent series (like chapter 10) and series with a mix of trend, seasonality and noise (like chapter 3).

Series with small or zero values can cause problems in the calculation of error measurements (Sect. 4.2). To avoid this, we have treated all the series in the same way, adding a constant value of 1 to them.

Figure 2 shows the autocorrelation plots of the daily series in period 2000–2016. These plots shed light on which series have weekly, annual or both seasonalities. As we can observe, the seven series show different behavioural patterns. The weekly seasonality seems to be present in all the series, even in chapters 7 and 10 where it appears in an overlapping way to the annual seasonality. Chapter 7, shows a well defined annual seasonality. And chapter 10 seems to integrate an annual seasonality with a semiannual one.

2.2 Machine learning approach to forecasting

A time series is a sequence of observations s_t taken from a variable at successive equally spaced points t in time. For the purpose of multi-step forecasting, we represent a series by

Table 1 European tariff chapter of selected series

Chapter	Description
3	Fish and crustaceans, molluscs and other aquatic invertebrates
7	Edible vegetables and certain roots and tubers
10	Cereals
16	Preparations of meat, of fish or of crustaceans, molluscs or other aquatic invertebrates
62	Articles of apparel and clothing accessories, not knitted or crocheted
69	Ceramic products
85	Electrical machinery and equipment and parts thereof; sound recorders and reproducers, television image and sound recorders and reproducers, and parts and accessories of such articles

Table 2 Time series characteristics by granularity

Granularity level	Seasonality period	Forecast horizon	Time interval
Quarterly	Year	8	2000–2016
Monthly	Year	18	2000–2016
Weekly	Year	13	2000–2016
Daily	Year/Week	14	2014–07 to 2016–12

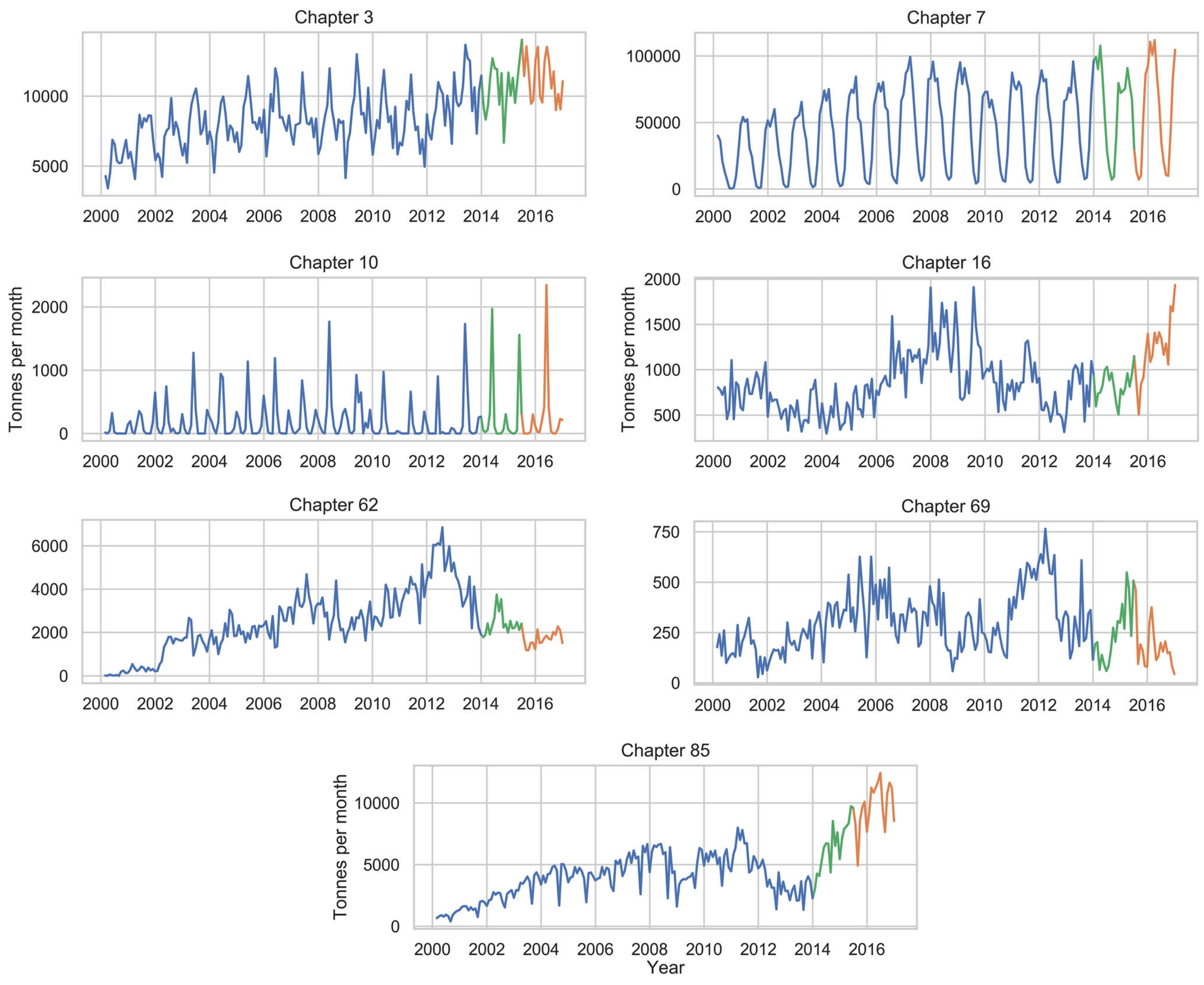

Fig. 1 Monthly series. Green represents the validation data and orange the test data

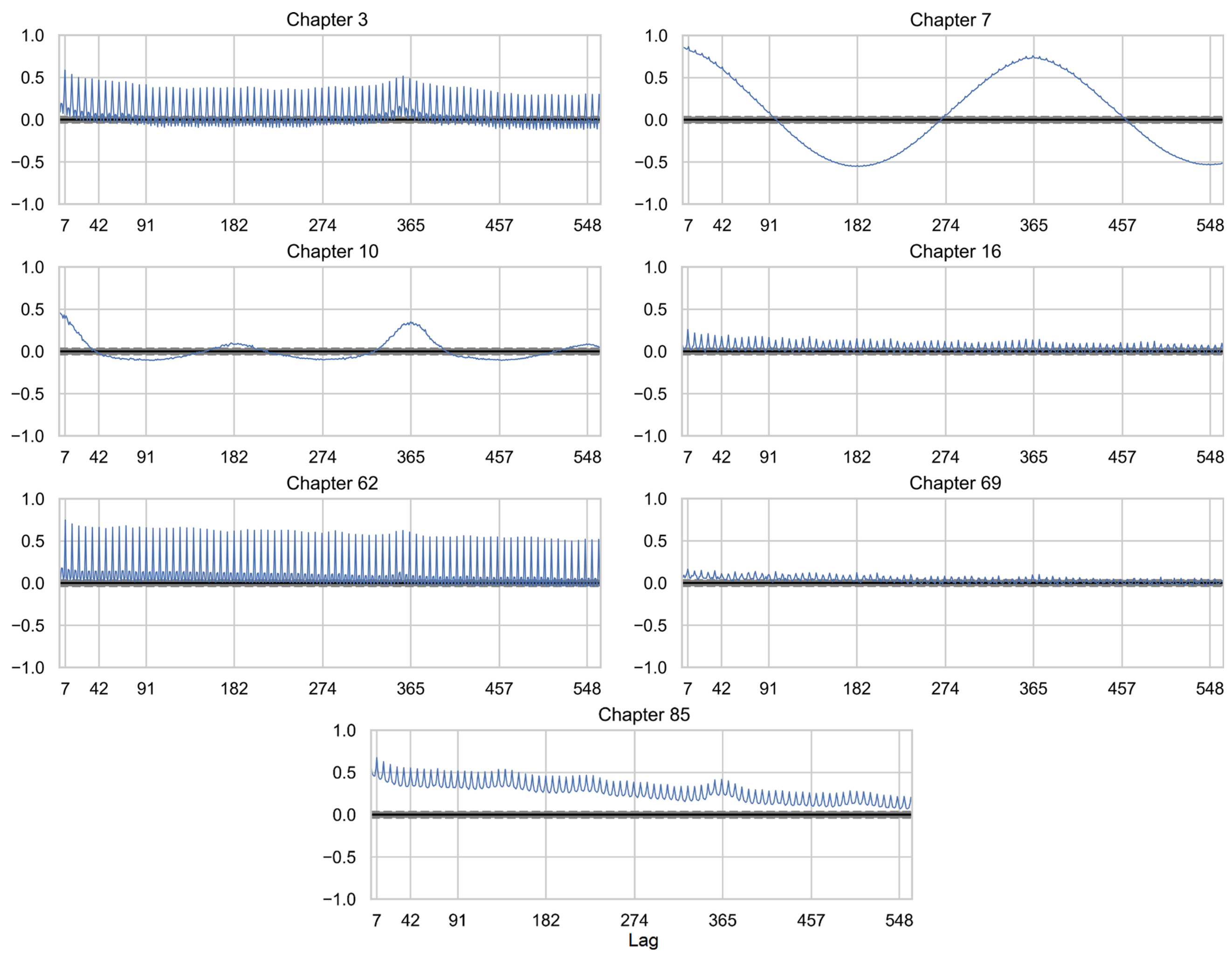

Fig. 2 Autocorrelation plot of daily series in period 2000–2016

$$S = \{s_1, s_2, \ldots, s_T, s_{T+1}, \ldots, s_{T+h}, \ldots\}, \tag{1}$$

where s_T is the last observation previous to the prediction of the observed values $\{s_{T+1}, \ldots, s_{T+h}\}$, being h the forecast horizon.

A machine learning algorithm is able to learn from experience to perform a task by combining an optimization procedure, a cost function, a model and a dataset. Time series forecasting can be performed as a regression task, where a machine learning algorithm is asked to make a prediction **y** for the next h time steps, conditioned by the n preceding observations **x**. To address this, we have considered two forecasting strategies (Fig. 3), namely the multiple output and the recursive strategies (Bontempi et al. 2012). In the multiple output strategy, our model defines a mapping from $\mathbb{R}^n \rightarrow \mathbb{R}^h$ to get **y** directly from **x**, however, in the recursive strategy our model defines a mapping from $\mathbb{R}^n \rightarrow \mathbb{R}$ to obtain a one-step prediction y from **x**. For the next prediction, we modify the vector **x** by removing the oldest observation and adding y so that it is temporarily ordered. We iterate in the same way until we obtain the h one-step predictions for the vector **y**.

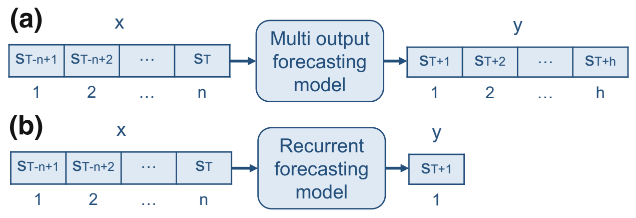

Fig. 3 Machine learning vector-to-vector models for time series forecasting. The input vector **x** contains the n values previous to the prediction. The output vector **y** contains the predicted values. **a** The multiple output model obtains the complete h (forecast horizon) values of the forecast. **b** The recurrent model obtains a one-step prediction value that has to be used iteratively to obtain the h values

Once the mappings of our model are defined, we need a dataset of examples so that the machine learning algorithm can experience through a supervised learning scheme. The

Fig. 4 Training sets for multiple output and recursive forecasting strategies

$$X = \begin{bmatrix} s_1 & s_2 & \cdots & s_n \\ s_2 & s_3 & \cdots & s_{n+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{T-h-n+1} & s_{T-h-n+2} & \cdots & s_{T-h} \end{bmatrix} \quad Y = \begin{bmatrix} s_{n+1} & s_{n+2} & \cdots & s_{n+h} \\ s_{n+2} & s_{n+3} & \cdots & s_{n+1+h} \\ \vdots & \vdots & \ddots & \vdots \\ s_{T-h+1} & s_{T-h+2} & \cdots & s_T \end{bmatrix}$$

(a) Multiple output

$$X = \begin{bmatrix} s_1 & s_2 & \cdots & s_n \\ s_2 & s_3 & \cdots & s_{n+1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{T-n} & s_{T-n+1} & \cdots & s_{T-1} \end{bmatrix} \quad Y = \begin{bmatrix} s_{n+1} \\ s_{n+2} \\ \vdots \\ s_T \end{bmatrix}$$

(b) Recursive

training sets for our two forecasting strategies are created from the observed values $\{s_1, \ldots, s_T\}$, and consists of matrices $\mathbf{X}$ and $\mathbf{Y}$ (Fig. 4), where each training pair is provided by the ith row of $\mathbf{X}$ and $\mathbf{Y}$ respectively. The goal for our model is to learn from the example pairs how to predict $\mathbf{y}$ from $\mathbf{x}$ when using the multiple output strategy, and how to predict the single output y from $\mathbf{x}$ in the recursive case. The strength of the multiple output strategy is its ability to capture the dependence between the predicted values. In the case of the recursive strategy, every predicted value depends on the accuracy of the previous ones, accumulating errors, so it may deteriorate. The advantage of the recursive strategy is that it results in a training set larger that the direct one, increased by $h - 1$ pairs. In addition, it uses data closer to the prediction as input for the training. In our experiments we have reached better results using the direct strategy for daily and weekly granularities, and using the recursive one for monthly and quarterly granularities. This makes sense because, for monthly and quarterly granularities we have less data points for training.

During the learning process, the optimization procedure aims to minimize a cost function, that provides a measure of the error in the model forecasts. This is achieved by adjusting certain internal parameters of the model, represented by the vector $\boldsymbol{\theta}$, that define the mapping $\mathbf{y} = f(\mathbf{x}, \boldsymbol{\theta})$ of the model; from now, when the forecasting strategy is recursive, we will consider $\mathbf{y}$ a one-dimensional vector. Among the possible cost functions, we have chosen the mean absolute error (MAE), defined over the complete dataset of m training examples $(\mathbf{x}, \mathbf{y})$ by:

$$J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \|\mathbf{y} - f(\mathbf{x}, \boldsymbol{\theta})\|_1, \tag{2}$$

where $\|.\|_1$ denotes L_1 norm. Minimizing the MEA cost function leads to a function $f(\mathbf{x}, \boldsymbol{\theta})$ that estimates the median value of $\mathbf{y}$ for each $\mathbf{x}$; assuming that the training pairs are a subset of the infinitely many samples from the true data generating distribution (Goodfellow et al. 2016). We have used other regression-specific cost functions and obtained worse results. This is consistent with the recent work of Chai et al. (2019), where they concluded that the MAE cost function can achieve a better robustness and generalization capability than the conventional mean square error (MSE); and the investigation of Qi et al. (2020) about the advantages of the MAE in our context of deep neural network-based vector-to-vector regression.

Due to the complexity of neural network models, they are usually optimized using an iterative method, such as gradient descent, that allows controlling the overfitting during optimization. To do this, a percentage of training pairs is separated, so that they do not influence the optimization, although the value of the cost function $J(\boldsymbol{\theta})$ is calculated for these pairs, saving the parameters $\boldsymbol{\theta}$ each time a new minimum is reached for them. Once the optimization is complete, the saved values are used to define the model.

It is a common practice to split the time series into training and test portions (Hyndman and Athanasopoulos 2018). The test data are used to evaluate the accuracy of a model trained with previous data (Fig. 5a). It is necessary to adjust the model's hyperparameters before the test forecasting are made. To achieve this, the training data are split into a validation and a new training part (Fig. 5b). Then, the model is trained repeatedly until the hyperparameters are tuned to get the best prediction regarding the validation data. A length for the test and validation data equal to the forecast horizon is just the right size for our task.

3 Neural network architectures

The capability of machine learning models in the effective determination of seasonal effect and trend in time series is a controversial issue (Makridakis et al. 2018). In our deep learning models, we have found that it is possible, achieving the best results with an input data size of one year, agreeing with the conclusions of Hamzaçebi (2008), who proposes

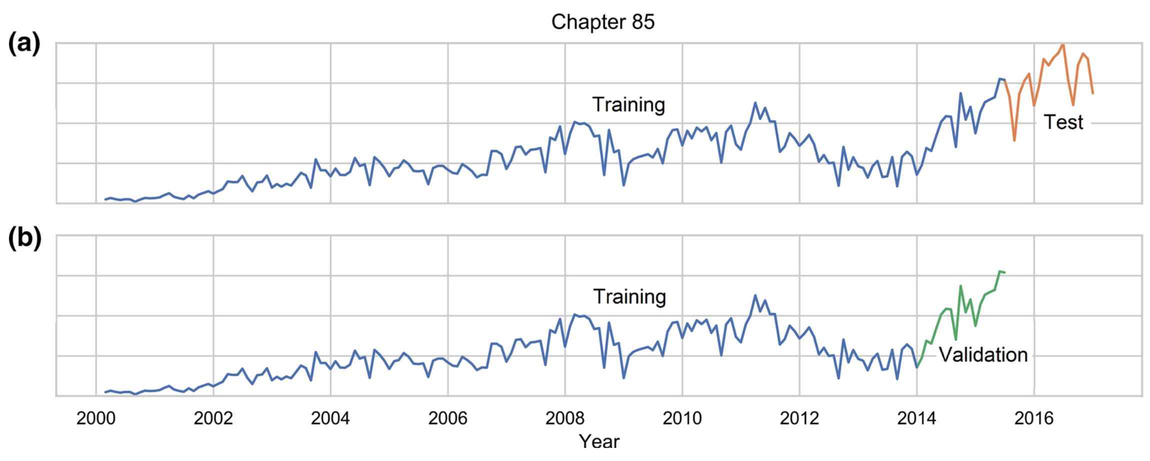

Fig. 5 Model training, validation and test datasets during: **a** the test process, **b** the validation process

an input size equal to the length of the seasonal period for improving forecasts with seasonal series.

3.1 Convolutional networks

A convolutional neural network (Aloysius and Geetha 2017) (CNN) is a type of artificial neural network in which neurons are organized in blocks, resulting in a neuron specialization similar to that present in the neurons of the visual cortex of a biological brain. Although the original concepts of the CNNs are inspired by image recognition, the model is applied very successfully to many other tasks. The hierarchical structure of CNNs makes them less dense, and less prone to overfitting, than the general (fully connected) neural network model. CNNs usually include two types of layers: convolutional and pooling layers. Convolutional layers apply filters by means of operations that involve a region of the input data. In image processing, for example, these filters allow certain aspects of the images to be highlighted, such as borders, vertical lines, etc. Pooling layers are usually applied after a convolution and they summarize, by reducing the dimensionality, the features extracted by the convolutional filters. It is usual to complete the architecture of a CNN with a final fully connected layer that performs the high-level reasoning of the network, using as input the features extracted by the previous convolutional and pooling layers. CNNs designed for image processing work with 2D data, but they can be defined for any other dimension. 1D CNNs, for example, can process sequential data such as natural language texts or time series (Lim et al. 2017).

3.1.1 The dilated causal convolutional layer

A 1D-convolutional layer is composed of a number of units that perform the convolution operation. Usually in practice, each unit computes the cross-correlation operation of an n-dimensional input vector $\mathbf{x}$ and a learnable vector $\mathbf{k}$, called kernel or filter. A one-unit convolutional layer outputs a vector $\mathbf{y}$ given by

$$y_t = (\mathbf{x} \star \mathbf{k})_t = \sum_i x_{t+i-1} k_i, \tag{3}$$

where $\star$ denotes a cross-correlation operator. Each y_t contains a similarity measure, obtained through the dot product of a region of the input $\mathbf{x}$ (starting at x_t) and the filter $\mathbf{k}$. The output $\mathbf{y}$ is sometimes referred to as the feature map. When forecasting time series, the value of y_t cannot depend on future values of $\mathbf{x}$, so the causal convolution is used to address it:

$$y_t = \sum_i x_{t-i+1} k_i, \tag{4}$$

where an implicit zero padding of $\mathbf{x}$ is assumed in the implementation, so that both $\mathbf{x}$ and $\mathbf{y}$ are n-dimensional. Figure 6 depicts this operation. A causal convolutional layer with m units would provide a $n \times m$ feature map. It is very common to work with a stack of convolutional layers, so they must be able to process a $n \times m$ matrix $\mathbf{X}$ to get a $n \times m$ matrix $\mathbf{Y}$, in which each feature $Y_{t,f}$ is calculated using the 2D-slice $\mathsf{K}_{f,:,:}$ of the filter tensor and a region of the matrix $\mathbf{X}$ ending at row t:

$$Y_{t,f} = \sum_{i,j} X_{t-i+1,j} \mathsf{K}_{f,i,j} \tag{5}$$

To deal with long-range time dependencies in the time series, one solution is to increase the input region affecting the features (receptive field). The dilated causal convolution applies the filter over a region larger than its size by skipping input values with a certain rate r:

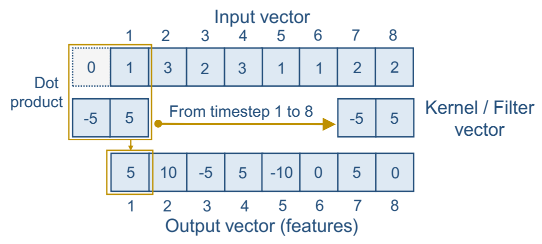

Fig. 6 Causal convolutional unit example. The dot product of the kernel vector and a slice of the input is calculated for each time step (the stride value is 1), padding with zeros at the beginning; and the features are stored in the output vector. Causal convolution means that features are not based on future time steps

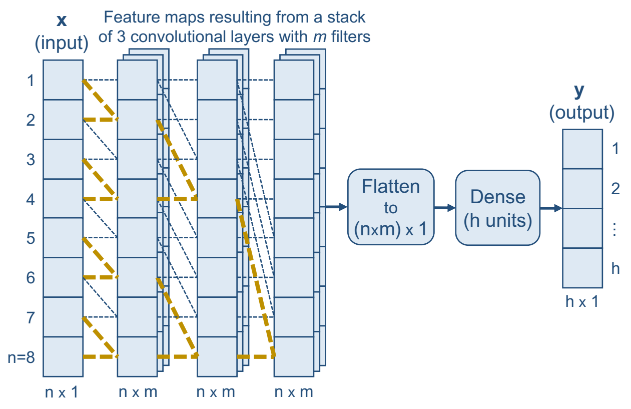

Fig. 7 DCCNN forecasting model. Without loss of generality, we have assumed an input vector **x** of length $n = 8$. Each convolutional layer (omitted for clarity) contains m filters of 2 rows that get a new feature map. The yellow lines show how an increasing dilation rate in powers of 2 enlarge the receptive field affecting the features. Finally, the last feature map is resized so that the dense layer can predict the output **y**

$$Y_{t,f} = \sum_{i,j} X_{t-ir+r,j} \mathsf{K}_{f,i,j} \tag{6}$$

Figure 7 shows the feature maps obtained by a stack of dilated causal convolutional layers, and how a dilation rate that increases exponentially in each layer, results in an exponential growth of the receptive field. To complete the convolutional layer, a learnable bias b_f is added per filter, and also an activation function:

$$Y_{t,f} = \phi\left(\sum_{i,j} X_{t-ir+r,j} \mathsf{K}_{f,i,j} + b_f\right) \tag{7}$$

where ϕ is usually the rectified linear activation function (ReLU).

3.1.2 Dilated causal convolutional neural network (DCCNN)

In this deep learning approach, we have leveraged new architectures from the work of Oord et al. (2016), based on the use of a stack of dilated causal convolutional layers to deal with long-term dependencies.

Our DCCNN model defines a vector-to-vector mapping $\mathbf{y} = f(\mathbf{x})$, from the n previous values of a time series to the next h forecasted values, and is trained to estimate the median value of **y** conditioned to **x**, as explained in Sect. 2.2. The model is implemented using a stack of dilated causal convolutional layers as defined in Eq. (7). The stack has l layers, each consisting of m units (filters), and a dilation rate r starting at 1 and doubling with each successive layer. The first layer maps the input vector **x** to a $n \times m$ feature map. Successive layers take as input the feature map of the previous layer to produce a new one, in which each feature is influenced by a receptive field that is consequently duplicated each time, starting from a length of 2 (Fig. 7). We have determined the number of layers l so that the size of the receptive field does not exceed the dimensionality of the input vector **x**:

$$l = \lfloor \log_2 n \rfloor, \tag{8}$$

where $\lfloor . \rfloor$ denotes the integer part.

Once the input vector **x** is processed by the learned stack of convolutional layers, the resulting meaningful feature map needs to be reshaped to a vector. Then, a learned dense layer with h units maps the features vector to the forecasting output vector **y**. Dilation significantly reduce the number of convolutional layers required, not being necessary the addition of pooling layers. The number of filters m of the convolutional layers is a hyperparameter tuned during model validation.

3.2 Recurrent networks

Recurrent neural networks (Mandic and Chambers 2001) is another major architecture in the current catalogue of deep neural network models.

They are basically characterized by having layers that, in addition to receiving information from the previous layer of the network, also receive it from the state of the layer itself in the previous iteration, i.e. they have cycles and therefore state memory. This makes them dependent on time and causes the order in which the data are provided to directly influence the results obtained, something appropriate when working with time series.

3.2.1 From the dense layer to the GRU layer

A typical dense (fully connected) layer contains m neural units that map from a d-dimensional input vector **x** to an m-dimensional output vector **y**, and is defined by

$$\mathbf{y} = \phi(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{9}$$

where ϕ is the activation function that makes nonlinearity possible, $\mathbf{W}$ is a learnable $m \times d$ weight matrix, and $\mathbf{b}$ a learnable m-dimensional bias vector. If we feedback the output to the input, we get a recurrent neural network (RNN) layer, that can operate on a sequence of inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_\tau$ into a sequence of "hidden" outputs $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_\tau$, with each output $\mathbf{h}_t$ depending on the input $\mathbf{x}_t$ (at the same time step) and the output $\mathbf{h}_{t-1}$(at the previous time step):

$$\mathbf{h}_t = \phi(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}), \tag{10}$$

where ϕ is usually the tanh activation function and $\mathbf{U}$ is a learnable $m \times m$ weight matrix. Considering the RNN layer as a dynamical system, $\mathbf{h}_t$ will represent the state at time step t. The RNN layer can use the state to store information about activations triggered by previously processed data. Learning about long sequences, using a gradient descent method, presents the main problem that the flow of the back-propagation error tends to disappear. Hochreiter and Schmidhuber (1997) provided a remedy to this problem with the Long Short-Term Memory (LSTM) architecture, in which they achieve a constant error flow through the use of gating mechanisms to control the flow of information. Later, Cho et al. (2014) proposed the Gated Recurrent Unit (GRU) architecture, based on this same concept and structurally similar, but simpler and usually faster than an LSTM, yielding similar results (Chung et al. 2014).

In a GRU layer, the output state $\mathbf{h}_t$ is a mixture of the previous state $\mathbf{h}_{t-1}$ and the proposed estate $\tilde{\mathbf{h}}_t$, governed in each dimension by an update gate $\mathbf{u}_t$,

$$\mathbf{h}_t = \mathbf{u}_t \odot \tilde{\mathbf{h}}_t + (1 - \mathbf{u}_t) \odot \mathbf{h}_{t-1}, \tag{11}$$

where $\odot$ denotes the Hadamard product, allowing that a value close to 1 in a dimension of the gate $\mathbf{u}_t$ causes the update of this dimension in the output state $\mathbf{h}_t$ to the value for this dimension in the proposed state $\tilde{\mathbf{h}}_t$. Similarly, a value close to 0 causes an update in the output state from the previous state, allowing to remember useful information for the future. The update gate $\mathbf{u}_t$ is a mapping from the current input $\mathbf{x}_t$ and the previous output estate $\mathbf{h}_{t-1}$ to the range 0 to 1,

$$\mathbf{u}_t = \sigma(\mathbf{W}_u\mathbf{x}_t + \mathbf{U}_u\mathbf{h}_{t-1} + \mathbf{b}_u), \tag{12}$$

where σ is the logistic sigmoid function, $\mathbf{W}_u$ and $\mathbf{U}_u$ are learnable weight matrices, and $\mathbf{b}_u$ is a learnable bias vector. The output candidate $\tilde{\mathbf{h}}_t$ is calculated similarly to the output state in an RNN layer, but using a reset gate $\mathbf{r}_t$ to selectively erase part of the information contained in the previous state $\mathbf{h}_{t-1}$:

$$\tilde{\mathbf{h}}_t = \phi(\mathbf{W}\mathbf{x}_t + \mathbf{U}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}). \tag{13}$$

The reset gate $\mathbf{r}_t$ is computed in a similar way as the update gate:

$$\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{b}_r). \tag{14}$$

As a whole, the GRU layer is capable of preserving relevant information in future, forgetting what is no longer of interest.

3.2.2 Encoder–decoder recurrent neural network (EDRNN)

This deep learning approach is based on the encoder–decoder architecture proposed by Cho et al. (2014) and Sutskever et al. (2014), to map sequences to sequences.

Our EDRNN model defines a vector-to-vector mapping $\mathbf{y} = f(\mathbf{x})$, from the n previous values of a time series to the next h forecasted values, which is trained to estimate the median value of $\mathbf{y}$ conditioned to $\mathbf{x}$, as explained in Sect. 2.2. The model is implemented using GRU layers as defined in Eqs. (11–14) for our particular case of scalar sequence of inputs x_t, and consists of two separate neural networks (encoder and decoder) jointly trained. The encoder–decoder architecture (Fig. 8) allows processing input and output sequences of different lengths. Its design and operation are described below:

(a) *Encoder* A learned GRU layer with m units processes the input vector $\mathbf{x} = (x_1, \ldots, x_n)$ in temporal order, updating the m-dimensional output state $\mathbf{h}_t$ for every input value so the final state $\mathbf{c}$ codifies the whole history of the input. The number of units m is a hyperparameter tuned during model validation.

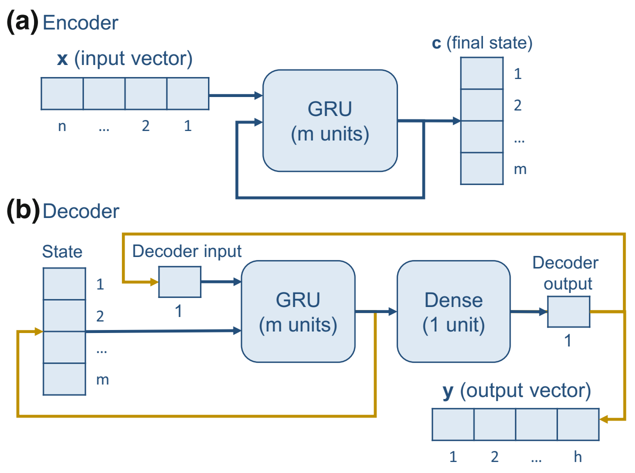

Fig. 8 EDRNN forecasting model. **a** Encoder. The input vector $\mathbf{x}$ is summarized in a state vector $\mathbf{c}$. **b** Decoder. The state of the GRU layer is initialized to the final state $\mathbf{c}$ of the encoder, and the decoder input to x_n. In orange we highlight the prediction loop where, for each iteration, the new predicted value is stored in the next component of the output vector $\mathbf{y}$, and the GRU and dense layer outputs are fed back

Table 3 Benchmarks of the M4 Competition used as baseline

Statistical benchmarks without removing the seasonality or trend		
1	Naïve 1	All forecasts are equal to the last observation prior to the prediction
2	Seasonal Naïve (Naïve S)	All forecasts for the same time step within the seasonal period are equal to the last known observation of the same time step. This means that the last observed seasonal period is repeated consecutively in the prediction
Statistical benchmarks removing the seasonality		
3	Naïve 2	Like Naïve 1
4	SES	Simple exponential smoothing
5	Holt	Holt's exponential smoothing
6	Damped	Damped exponential smoothing
7	Combination (Comb)	The arithmetic average of methods SES, Holt and Damped
8	Theta	It was the winner method of the M3 Competition. It decomposes a series into two or more Theta lines that are extrapolated separately and combined to form a forecast (Assimakopoulos and Nikolopoulos 2000)
Machine learning benchmarks removing the seasonality and linear trend		
9	MLP	A multilayer perceptron
10	RNN	A basic recurrent neural network

(b) *Decoder* It consists of a GRU layer with m units and a dense layer with 1 unit. At each time step t the dense layer maps the state $\mathbf{h}_t$ of the GRU layer to the decoder output y_t, and the GRU layer maps the previous state $\mathbf{h}_{t-1}$ and the previous decoder output y_{t-1} to the current state $\mathbf{h}_t$. Thus, the decoder model can be defined by

$$y_t = g(\mathbf{h}_{t-1}, y_{t-1}). \tag{15}$$

The decoder is trained according to this mapping, with the initial $\mathbf{h}_{t-1}$ set to the final state $\mathbf{c}$ of the encoder, and the initial y_{t-1} set to the last encoder input x_n. To generate the first prediction y_1 of the decoder, we start by initializing it in the same way as before. Then, we iterate by feeding back each decoder output as a new decoder input, and each output state of the GRU layer as the new input state, until we complete h consecutive predictions.

4 Methodology

In this section, we present our experimental methodology. First, we briefly describe the baseline methods. In the second subsection, we present the different evaluation metrics we have used. Finally, in the third subsection we specify the main aspects of our experimental scenarios.

4.1 Baseline

We have taken as baseline the ten benchmarks of the M4 Competition (Makridakis et al. 2020) listed in Table 3 and whose code is available online.[1]

In some of these methods, the seasonality of the series is removed when passing a 90% autocorrelation test, and in others the linear trend may also be removed. In these cases, once the forecasts are made and before the results are evaluated, both are restored.

4.2 Evaluation metrics

Among the forecast accuracy metrics typically used, we have adopted those of the M4 Competition.

These metrics are expressed in terms of: the training data $\{s_1, s_2, \ldots, s_T\}$; the test data $\{s_{T+1}, s_{T+2}, \ldots, s_{T+h}\}$, where h is the forecast horizon; and the forecast data $\{\widehat{s_{T+1}}, \widehat{s_{T+2}}, \ldots, \widehat{s_{T+h}}\}$.

Detailed information on these metrics is given below:

– The symmetric mean absolute percentage error (sMAPE) metric was originally defined by Armstrong (1985) and later modified:

$$sMAPE = \frac{2}{h} \sum_{t=T+1}^{T+h} \frac{|s_t - \widehat{s_t}|}{|s_t| + |\widehat{s_t}|} \cdot 100(\%) \tag{16}$$

This percentage metric provides a result between 0 and 200%. It has the disadvantage of being undefined for intermittent series when any observed value s_t and its prediction $\widehat{s_t}$ are equal to zero. Also problematic are small values of s_t (Hyndman and Koehler 2006). Moreira-Matias et al. (2013) encountered these problems and added a constant to the denominator to produce more accurate statistics, based on the work of Jaynes (2003). We have adopted a similar solution, adding 1 to all the series before making the forecasts, and subtracting it after

[1] https://github.com/M4Competition/M4-methods.

calculating the error metrics. In this way we also avoid problems related to the seasonal adjustments of the baseline methods when zero values are present.

– The mean absolute scaled error (MASE) was proposed by Hyndman and Koehler (2006) to make it possible to compare the accuracy of forecasts for all types of series. The seasonal MASE is a scaled metric that divides each forecast absolute error by the mean absolute error (MAE) of the Naïve S forecast method on the training data, and is defined as:

$$\begin{aligned} MASE &= \frac{1}{h}\frac{\sum_{t=T+1}^{T+h}|s_t - \widehat{s_t}|}{MAE(NaïveS_{train})} \\ &= \frac{1}{h}\frac{\sum_{t=T+1}^{T+h}|s_t - \widehat{s_t}|}{\frac{1}{T-m}\sum_{t=m+1}^{T}|s_t - s_{t-m}|} \end{aligned} \tag{17}$$

where m is the length of the seasonal period considered. Note that a one-step forecasting is done for the seasonal naïve method, which means that every forecast $\widehat{s_t}$ is equal to s_{t-m}. The m values considered for the quarterly and monthly granularity are 4 and 12, respectively. For others granularities, we have selected the values giving the best results of the baseline methods, which are 52 for weekly and 7 for the daily. In any case, for those time series that fail the seasonality test of the baseline methods, m is set to 1.

– The overall weighted average (OWA) summarizes the two previous metrics in a single value that facilitates the ranking of the different forecasting methods. An OWA value of less than 1 means that the forecast error of the evaluated method is less than that of the Naïve 2 method. The OWA is calculated for each forecasting method as follows:

1. sMAPE and MASE are computed for each series individually.
2. sMAPE mean and MASE mean are obtained across all series.
3. sMAPE mean and MASE mean are made relative to its corresponding sMAPE mean and MASE mean of the forecast Naïve 2 method.
4. Finally, the mean of the relative sMAPE and the relative MASE is obtained.

$$OWA = \left(\frac{\overline{sMAPE}}{\overline{sMAPE}_{Naïve2}} + \frac{\overline{MASE}}{\overline{MASE}_{Naïve2)}}\right)/2 \tag{18}$$

4.3 Implementation framework and setup

The proposed models, convolutional (DCCNN) and recurrent (EDRNN), have been implemented using Keras library (Chollet et al. 2015) with TensorFlow in the backend and executed on an Intel(R) Core(TM) i7-6700 CPU @3.4GHz processor having four cores, with 32GB and a NVIDIA GeForce GTX 1070. The pseudorandom number generator has been initialized so that the results can be reproduced.

Relying on the capabilities of deep learning we have considered the input time series without making any transformation or eliminating the seasonality; only the data was scaled to the [0, 1] range before being input to the models, and then restored in the output data.

We have selected the models hyperparameters exclusively on the basis of the validation results, keeping the test data separate to evaluate the models once they have been optimized. Due to the large number of combinations in choosing the best model configuration and the training time spent in each case, we have manually tuned almost all the hyperparameters, as they are: optimizer is Adam, activation function for the dense output layer is linear, batch size is the training data size, number of epochs is 125 and 525 for DCCNN and EDRNN, respectively, and loss function is the MAE. Also a 33% of training data is reserved to control overfitting (model weights are saved for the best result obtained with this data). Concerning the input length of the models, our selection has been the corresponding to one year, which confirms the proposal of (Hamzaçebi 2008) for seasonal time series.

Only one hyperparameter has been selected automatically for the two models depending on the granularity of the data. These hyperparameters are the number of filters of the convolutional layers in model DCCNN and the number of units of the recurrent layers in model EDRNN. To accomplish this for a specific model and granularity, we have first obtained the forecasts within the range [0, 1] for all series, and averaged their MAE. This is calculated for each value of the hyperparameter, varying it from 1 to 250 for the daily series and from 1 to 200 for the others. The results are stored in a vector. Then a moving average of order 7 is calculated from this vector, and also a moving standard deviation of order 7 so to have information about the variability of the averaged values. We have stored these results in two vectors $ma7$ and $std7$ respectively. Finally, the selected value for the hyperparameter is the one associated to the minimum of $ma7 + std7/2$ (Fig. 9). Table 4 shows the hyperparameters values for all the models and granularities.

We have chosen a recursive prediction strategy for granularities with small training data, such as quarterly and monthly. This way we make the best use of the limited data available. For weekly and daily granularities, however, the models have taken full advantage of the direct strategy.

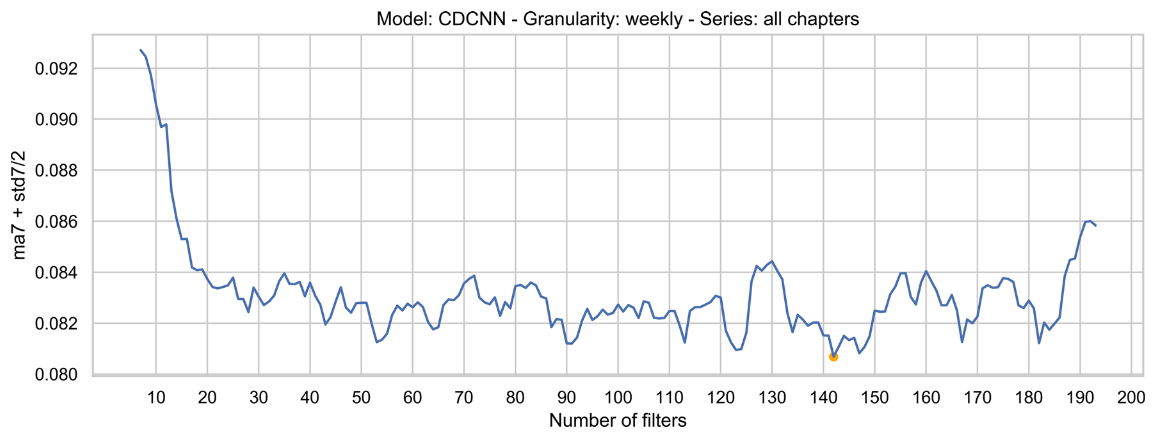

Fig. 9 Automatic selection of hyperparameters. The number of filters of convolutional layers in model DCCNN with weekly series is set to 142

Table 4 Hyperparameters values automatically selected for the two models

Model DCCNN				Model EDRNN			
Number of filters in convolutional layers				Number of units in recurrent layers			
Quarterly	Monthly	Weekly	Daily	Quarterly	Monthly	Weekly	Daily
24	52	142	39	20	137	29	211

5 Experimental results

In this section we present the main results of our experiments. First, we include a comparison to show that our two network architectures improve the results of the baselines.

In the second subsection we show, through several graphs, forecast examples generated by the two network architectures for the four levels of granularity used in the experiments.

5.1 Test and validation results

Table 5 shows the overall performance of our two models, taking all series and granularity levels together. We outperform the best OWA of the benchmarks with the two models using both validation and test data. The convolutional model DCCNN improves by 0.74–6.93% and the recurrent model EDRNN improves by 4.96–8.93%.

Tables 6 and 7 show the validation and test results, respectively, detailed for each granularity level. We mainly highlight the performance of the recurrent model with daily data, and the convolutional model with the weekly one. Both used a direct forecasting strategy that significantly improves the best OWA of the benchmarks by more than 12%. Also outstanding is the convolutional model with monthly data and a recursive forecasting strategy that improves the best OWA of the benchmarks. In any case, except for the quarterly granularity, at least one of our two models outperforms all the benchmarks.

Seasonality is perfectly captured by the two models. Evidence of this is that the convolutional model with weekly data widely exceeds the best benchmarks, Naïve 2 and Naïve S, on validation and test data respectively. These two benchmarkts make their forecasts based exclusively in the seasonality (Naïve 2) or the last observed seasonal period (Naïve S). Also, the recurrent model succeeds in capturing the seasonality as it outperforms all the benchmarks for daily data, where all the series pass the weekly seasonality test. In Sect. 5.2 we will appreciate this phenomenon through charts that show examples of several forecasts with both models.

As for the intermittent data, which are found in all granularities but are more present in the daily series, the results lead us to highlight one of the two models, the recurrent one.

The quarterly granularity is the only one for which our models cannot improve the benchmarks. We believe that this is due to the lower availability of data for this granularity, which penalizes deep network models that need a larger amount of data to be properly trained.

In order to reduce the variability of results, each forecast has been repeated 7 times and averaged before calculating the error metrics.

5.2 Examples of forecasts

In Figs. 10, 11, 12 and 13 we show forecasts on test data for different series using the convolutional architecture DCCNN and the recurrent architecture EDRNN. We have chosen a series for each granularity, depending on the type of seasonality:

Table 5 Validation and test global results

	Validation				Test			
	sMAPE	MASE	OWA	%	sMAPE	MASE	OWA	%
Naïve 1	64.999	1.958	1.226	−41.51	62.331	2.109	1.211	−40.51
Naïve S	50.910	1.505	0.951	−9.75	46.554	1.434	**0.862**	**0.00**
Naïve 2	56.182	1.512	1.000	−15.42	54.360	1.654	1.000	−16.06
SES	51.056	1.248	0.867	−0.11	52.401	1.327	0.883	−2.48
Holt	52.706	1.363	0.920	−6.16	55.279	1.361	0.920	−6.73
Damped	51.502	1.274	0.880	−1.54	52.388	1.320	0.881	−2.22
Theta	50.916	1.249	**0.866**	**0.00**	52.372	1.312	0.878	−1.93
Comb	51.483	1.290	0.885	−2.12	52.616	1.322	0.883	−2.53
MLP	55.795	1.526	1.001	−15.58	61.680	1.728	1.090	−26.46
RNN	50.061	1.279	0.868	−0.23	56.112	1.451	0.955	−10.79
DCCNN	48.811	1.287	**0.860**	**0.74**	46.279	1.245	**0.802**	**6.93**
EDRNN	41.860	1.273	**0.794**	**8.39**	44.607	1.352	**0.819**	**4.96**

Error reduction (%) on the best benchmark is shown

Table 6 Validation results for granularity level

	Quarterly			Monthly			Weekly			Daily		
	sMAPE	MASE	OWA	sMAPE	MASE	OWA	sMAPE	MASE	OWA	sMAPE	MASE	OWA
Naïve 1	56.319	2.322	1.162	58.706	2.287	1.356	59.454	1.202	1.406	85.519	2.019	1.087
Naïve S	51.429	2.169	1.074	45.763	1.432	0.945	50.687	1.136	1.264	55.760	1.283	**0.699**
Naïve 2	52.278	1.862	1.000	46.727	1.571	1.000	42.438	0.853	**1.000**	83.285	1.760	1.000
SES	45.257	1.787	0.912	40.165	1.284	0.839	47.635	0.838	1.053	71.169	1.085	0.736
Holt	52.541	2.214	1.097	40.660	1.358	0.867	47.789	0.845	1.058	69.834	1.034	0.713
Damped	47.622	1.924	0.972	39.611	1.262	**0.825**	47.678	0.842	1.055	71.098	1.069	0.731
Theta	44.598	1.771	0.902	40.300	1.301	0.845	47.630	0.839	1.053	71.134	1.087	0.736
Comb	47.529	1.963	0.982	40.044	1.295	0.840	47.701	0.841	1.055	70.659	1.059	0.725
MLP	42.913	1.893	0.919	45.762	1.608	1.001	57.495	1.175	1.366	77.010	1.429	0.868
RNN	40.265	1.642	**0.826**	44.627	1.531	0.965	45.805	0.872	1.051	69.545	1.068	0.721
DCCNN	51.970	2.168	**1.079**	35.950	1.202	**0.767**	39.194	0.698	**0.871**	68.130	1.079	**0.716**
			−30.6%			**7.1%**			**12.9%**			**−2.3%**
EDRNN	48.718	2.070	**1.022**	34.882	1.248	**0.771**	41.763	0.779	**0.949**	42.076	0.996	**0.536**
			−23.7%			**6.6%**			**5.1%**			**23.4%**

Error reduction (%) on the best benchmark is shown

- *Quarterly* we show chapter 7, which is the one with the purest annual seasonality.
- *Monthly* we show chapter 10, that has annual and semi-annual seasonality.
- *Weekly* we show chapter 85 that presents an annual seasonality and different trends over time.
- *Daily* We use chapter 3, that has a weekly seasonal and a fairly consistent trend over time.

As we can see from the comparative graphs of the forecasts on test data, both architectures behave quite well. Figure 13 shows that DCCNN behaves especially better for the weekly series, as it does on chapter 85. EDRNN, on the other hand, makes its best forecasts for the daily granularity. We can observe this phenomenon by the quality of the daily forecast of chapter 3 (Fig. 13).

6 Conclusions

In this paper, we have evaluated the usefulness of two deep learning models for predicting the flow of goods through seaports, in order to give insights to port authorities in their decision-making process. We have experimented with seven time series of imported goods from Morocco to Spain through the port of Algeciras. The initial series had a granularity of one minute, and they were resampled into four granularities (quarterly, monthly, weekly and daily), each associated with

Table 7 Test results for granularity level

	Quarterly			Monthly			Weekly			Daily		
	sMAPE	MASE	OWA	sMAPE	MASE	OWA	sMAPE	MASE	OWA	sMAPE	MASE	OWA
Naïve 1	48.339	2.628	1.290	58.402	2.410	1.545	66.643	1.865	0.979	75.940	1.534	1.157
Naïve S	39.012	2.032	1.018	41.322	1.446	1.000	44.657	0.939	**0.581**	61.224	1.319	0.967
Naïve 2	39.361	1.943	1.000	43.536	1.378	1.000	62.988	2.073	1.000	71.556	1.224	1.000
SES	35.838	1.725	0.899	42.589	1.269	0.950	51.281	1.297	0.720	79.893	1.017	0.974
Holt	48.496	1.800	1.079	42.132	1.212	**0.924**	51.037	1.289	0.716	79.449	1.141	1.021
Damped	35.706	1.671	**0.884**	41.963	1.225	0.927	51.253	1.296	0.719	80.630	1.087	1.007
Theta	35.945	1.680	0.889	42.819	1.264	0.950	51.083	1.289	0.716	79.643	1.016	0.972
Comb	37.098	1.687	0.905	42.095	1.226	0.928	51.191	1.294	0.718	80.078	1.081	1.001
MLP	44.774	2.381	1.181	52.707	1.738	1.236	61.403	1.562	0.864	87.835	1.232	1.117
RNN	44.596	2.236	1.142	47.765	1.467	1.081	50.908	1.128	0.676	81.179	0.973	**0.965**
DCCNN	37.467	1.985	**0.987**	38.072	1.314	**0.914**	40.644	0.779	**0.510**	68.932	0.902	**0.850**
			−11.7%			**1.1%**			**12.1%**			**11.9%**
EDRNN	41.651	2.289	**1.118**	41.690	1.407	**0.989**	51.410	1.009	**0.652**	43.679	0.703	**0.592**
			−26.5%			**−7.1%**			**−12.1%**			**38.6%**

Error reduction (%) on the best benchmark is shown

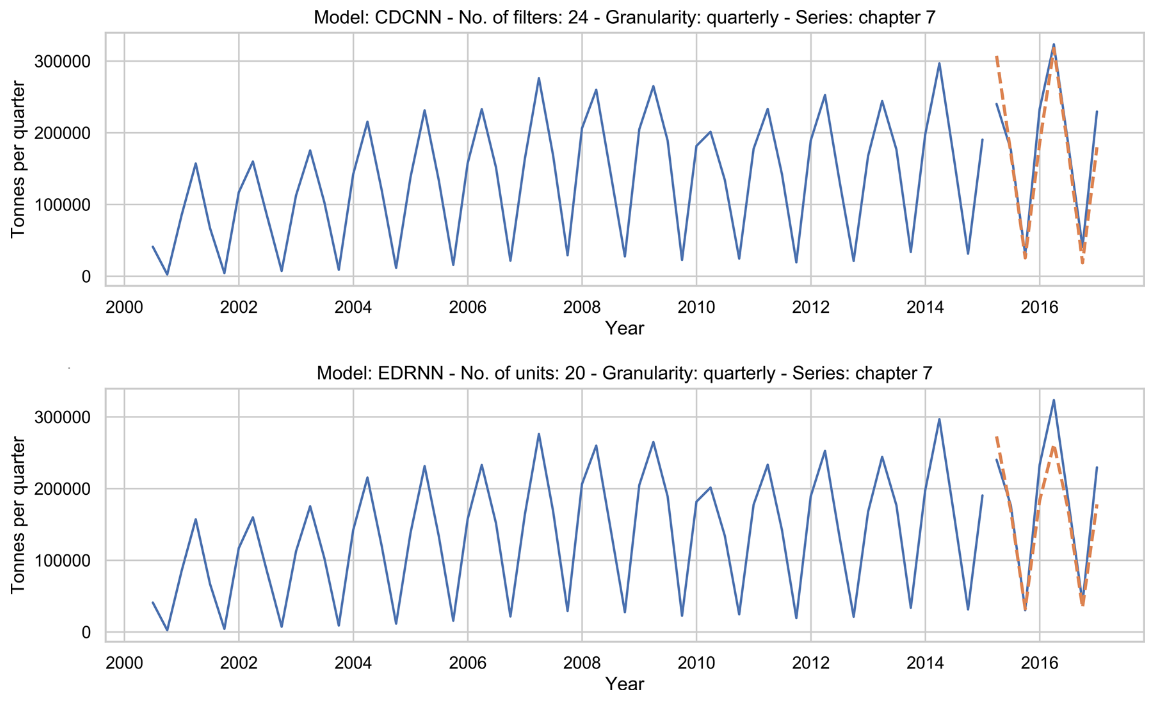

Fig. 10 Examples of quarterly forecasts on test data

a specific forecast horizon (8 quarters, 18 months, 13 weeks, and 14 days). The experiments show that our two models, dilated causal convolutional neural network and encoder–decoder recurrent neural network, can manage these raw series without first removing seasonality or trend.

Both models have globally beaten the OWA accuracy measure of all M4 Competition benchmarks. By levels of granularity, we highlight the recurrent model with daily data, which has improved the OWA of the best benchmark by 38.6% on the test set. Also remarkable is the convolutional model with weekly data, improving the best OWA by 12.1% on the test set. With quarterly data the results have been poor due to the few data available for a proper training of the models. The recurrent model achieve better results than the convolutional one for more intermittent series such as the daily ones.

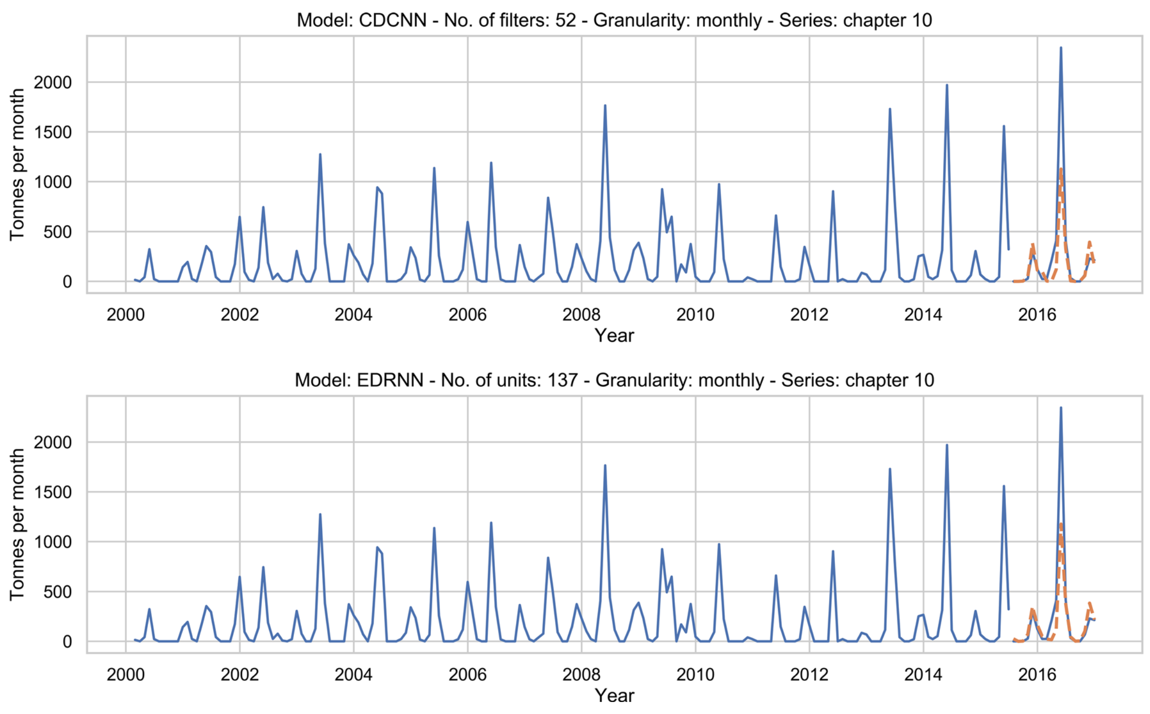

Fig. 11 Examples of monthly forecasts on test data

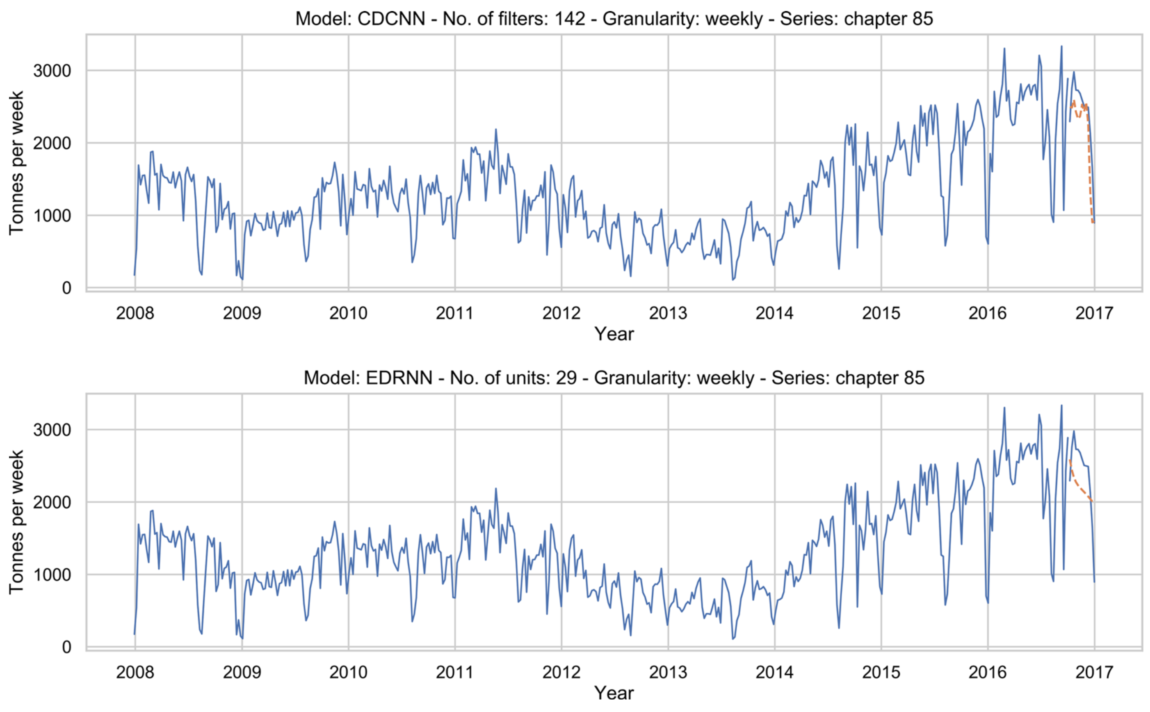

Fig. 12 Examples of weekly forecasts on test data

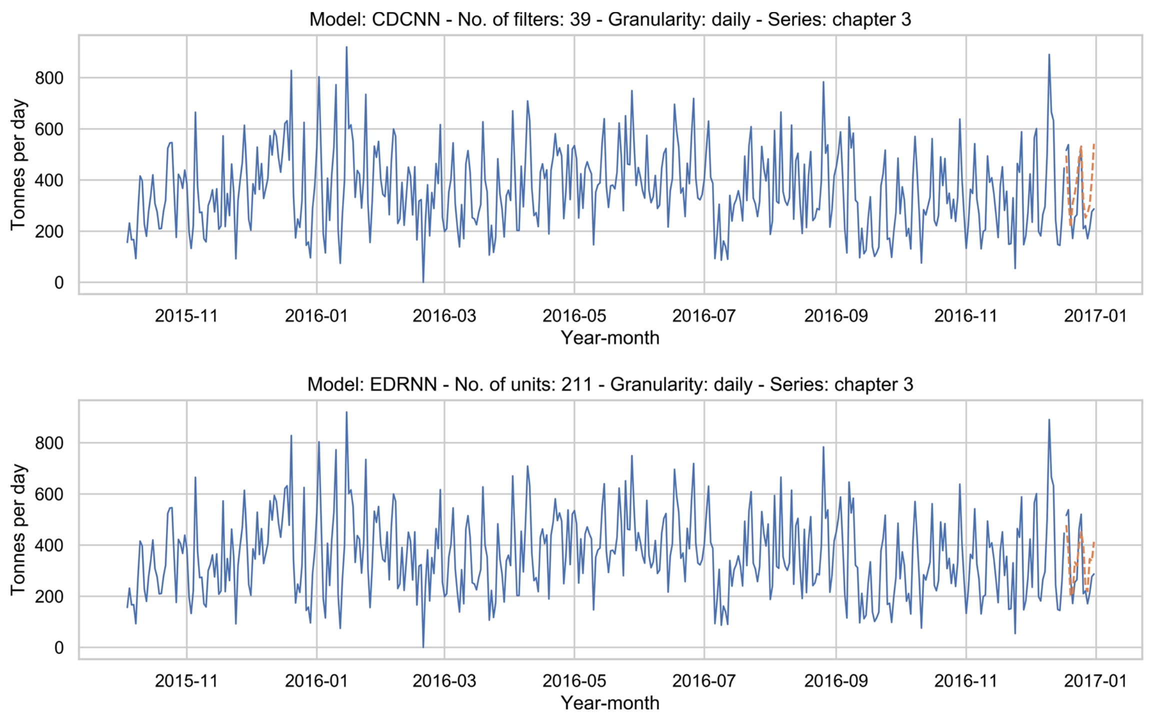

Fig. 13 Examples of daily forecasts on test data

We also wish to highlight the success in choosing a year for the length of the input data to the models, endorsed by the work of Hamzaçebi (2008), who proposes an input size equal to the length of the seasonal period for improving forecasts with seasonal series. The results and the visualization of the forecasts show us that the two models capture the seasonality and trend correctly, even for intermittent series, without applying any previous transformation on the data and simply scaling it to the [0, 1] range.

In our particular scenario of multi-output models, the direct forecasting strategy yields successful results on weekly and daily series, capturing the dependence between the predicted values. However, for monthly and quarterly series, where less training data is available, we have obtained better results with the recursive strategy, which increases the size of the training data using input data closer to the forecast.

Acknowledgements The authors are very grateful to the Algeciras Bay Port Authority for kindly providing the dataset used in this work.

Funding This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest/competing interests, that could have appeared to influence the work reported in this paper.

References

Aloysius N, Geetha M (2017) A review on deep convolutional neural networks. In: 2017 international conference on communication and signal processing (ICCSP), IEEE, pp 0588–0592

Armstrong JS (1985) Long-range forecasting. Wiley, New York ETC

Assimakopoulos V, Nikolopoulos K (2000) The theta model: a decomposition approach to forecasting. Int J Forecast 16(4):521–530

Batarseh F, Gopinath M, Nalluru G, Beckman J (2019) Application of machine learning in forecasting international trade trends. arXiv preprint arXiv:1910.03112

Bontempi G, Taieb SB, Le Borgne YA (2012) Machine learning strategies for time series forecasting. In: European business intelligence summer school, Springer, pp 62–77

Chai L, Du J, Liu QF, Lee CH (2019) Using generalized gaussian distributions to improve regression error modeling for deep learning-based speech enhancement. IEEE/ACM Trans Audio Speech Lang Process 27(12):1919–1931

Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using RNN encoder–decoder for statistical machine translation. arXiv preprint arXiv:1406.1078

Chollet F, et al. (2015) Keras. https://keras.io. Accessed 26 Apr 2020

Chopra S (1960) Supply chain management: strategy, planning and operation / Chopra S (2019) Kellogg School of Management. Seventh edition. Pearson Education, New York, NY

Chung J, Gulcehre C, Cho K, Bengio Y (2014) Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555

Goodfellow I, Bengio Y, Courville A (2016) Deep Learning. MIT Press. http://www.deeplearningbook.org. Accessed 16 Jan 2021

Hamzaçebi C (2008) Improving artificial neural networks' performance in seasonal time series forecasting. Inf Sci 178(23):4550–4559

Havenga JH (2013) The importance of disaggregated freight flow forecasts to inform transport infrastructure investments. J Transp Supply Chain Manag 7(1):1–7

Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780

Hyndman RJ, Athanasopoulos G (2018) Forecasting: principles and practice. OTexts

Hyndman RJ, Koehler AB (2006) Another look at measures of forecast accuracy. Int J Forecast 22(4):679–688

Jaynes, ET (2003) In: Bretthorst GL (ed) Probability theory: the logic of science. Cambridge University Press, Cambridge. https://doi.org/10.1017/CBO9780511790423

Lim H, Park J, Han Y (2017) Rare sound event detection using 1d convolutional recurrent neural networks. In: Proceedings of the detection and classification of acoustic scenes and events 2017 workshop, pp 80–84

Lloret I (2020) Imported goods from morocco to Spain through the port of algeciras (2000–2016). Mendeley Data, V1. https://doi.org/10.17632/gnh24w3mrg.1

Makridakis S, Spiliotis E, Assimakopoulos V (2018) Statistical and Machine Learning forecasting methods: concerns and ways forward. PLoS ONE 13(3):e0194889. https://doi.org/10.1371/journal.pone.0194889

Makridakis S, Spiliotis E, Assimakopoulos V (2020) The m4 competition: 100,000 time series and 61 forecasting methods. Int J Forecast 36(1):54–74

Mandic DP, Chambers J (2001) Recurrent neural networks for prediction: learning algorithms, architectures and stability. Wiley, Hoboken

Moreira-Matias L, Gama J, Ferreira M, Mendes-Moreira J, Damas L (2013) Predicting taxi-passenger demand using streaming data. IEEE Trans Intell Transp Syst 14(3):1393–1402

Oord Avd, Dieleman S, Zen H, Simonyan K, Vinyals O, Graves A, Kalchbrenner N, Senior A, Kavukcuoglu K (2016) Wavenet: a generative model for raw audio. arXiv preprint arXiv:1609.03499

Qi J, Du J, Siniscalchi SM, Ma X, Lee CH (2020) On mean absolute error for deep neural network based vector-to-vector regression. IEEE Signal Process Lett 27:1485–1489

Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: Advances in neural information processing systems, pp 3104–3112

Systematics C (1997) Inc. NCHRP report 388: a guidebook for forecasting freight transportation demand. Transportation Research Board, Washington DC