Neural Networks 157 (2023) 39–53

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Image-based time series forecasting: A deep convolutional neural network approach

Artemios-Anargyros Semenoglou [*], Evangelos Spiliotis, Vassilios Assimakopoulos

Forecasting and Strategy Unit, School of Electrical and Computer Engineering, National Technical University of Athens, Athens, Greece

A R T I C L E I N F O

Article history:
Received 22 September 2021
Received in revised form 4 August 2022
Accepted 6 October 2022
Available online 15 October 2022

Keywords:
Time series
Forecasting
Images
Deep Learning
Convolutional Neural Networks
M competitions

A B S T R A C T

Inspired by the successful use of deep learning in computer vision, in this paper we introduce *ForCNN*, a novel deep learning method for univariate time series forecasting that mixes convolutional and dense layers in a single neural network. Instead of using conventional, numeric representations of time series data as input to the network, the proposed method considers visual representations of it in the form of images to directly produce point forecasts. Three variants of deep convolutional neural networks are examined to process the images, the first based on VGG-19, the second on ResNet-50, while the third on a self-designed architecture. The performance of the proposed approach is evaluated using time series of the M3 and M4 forecasting competitions. Our results suggest that image-based time series forecasting methods can outperform both standard and state-of-the-art forecasting models.

1. Introduction

Machine learning (ML) and more recently deep learning (DL) are at the heart of many modern scientific and technological advances, with computer vision (Feichtenhofer, Fan, Malik, & He, 2019; Ker, Wang, Rao, & Lim, 2018), natural language processing (Young, Hazarika, Poria, & Cambria, 2018), and anomaly detection (Naseer et al., 2018) being only a few of the areas where ML and DL algorithms have dominated over the years, influencing both academia and daily life (Makridakis, 2017).

In the field of computer vision, ML and DL have been successfully applied in several pattern recognition and content understanding tasks, including, but not limited to, image classification (Rawat & Wang, 2017), semantic segmentation (Garcia-Garcia, Orts-Escolano, Oprea, Villena-Martinez, Martinez-Gonzalez, & Garcia-Rodriguez, 2018), object detection (Zhao, Zheng, Xu, & Wu, 2019), image denoising (Tian, Fei, Zheng, Xu, Zuo, & Lin, 2020), and generation of artificial images (Frolov, Hinz, Raue, Hees, & Dengel, 2021; Karras, Laine, & Aila, 2019). AlexNet, developed by Krizhevsky, Sutskever, and Hinton (2012), was one of the first deep convolutional neural networks (CNNs) to be used for large-scale image recognition, winning the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) in 2012 (Russakovsky et al., 2015). Since then, many studies have focused on understanding CNNs (Zeiler & Fergus, 2014), expanding their architecture, and improving their performance (Sermanet, Eigen, Zhang, Mathieu, Fergus, & LeCun, 2013). For instance, Simonyan and Zisserman (2015) proposed the utilization of smaller filters along with the addition of more layers. This CNN architecture, often called VGG, outperformed AlexNet when tested in the ILSVRC data set. ResNet (He, Zhang, Ren, & Sun, 2016) was another critical development in the area of image recognition as it introduced shortcut connections and allowed for deeper models to be effectively trained. Other architectures, claiming even more accurate results, are those of the DenseNet (Huang, Liu, Van Der Maaten, & Weinberger, 2017), Xception (Chollet, 2017), MobileNet (Sandler, Howard, Zhu, Zhmoginov, & Chen, 2018), and EfficientNet (Tan & Le, 2019).

Despite their resounding success in the aforementioned applications, ML and DL algorithms have been adopted at a much slower pace in the field of time series forecasting (Hewamalage, Bergmeir, & Bandara, 2021), also focusing on particular forecasting applications. For instance, although ML and DL methods have been successfully used in energy (Chae, Horesh, Hwang, & Lee, 2016; Majumdar & Gupta, 2019), retail demand Huber and Stuckenschmidt (2020) and Spiliotis, Makridakis, Semenoglou, and Assimakopoulos (2020b), and stock market (Fischer & Krauss, 2018; Wang & Wang, 2017) forecasting applications, their results have been mixed in generic forecasting tasks and especially in those where data availability is limited (Makridakis, Spiliotis, & Assimakopoulos, 2018b) or series display heterogeneous characteristics (Spiliotis, Kouloumos, Assimakopoulos, & Makridakis, 2020a). In such settings, statistical time series forecasting methods, such as exponential smoothing and ARIMA models, have long been considered as the standard methods of choice since

[*] Corresponding author.
E-mail addresses: artemis@fsu.gr (A.-A. Semenoglou), spiliotis@fsu.gr (E. Spiliotis), vassim@fsu.gr (V. Assimakopoulos).

https://doi.org/10.1016/j.neunet.2022.10.006
0893-6080/© 2022 Elsevier Ltd. All rights reserved.

they are not "data-hungry" and are computational cheap. On the negative side, statistical methods are less generic in a sense that they prescribe the data generation process of the series, display limited learning capacity, and are typically trained in a series-by-series fashion, thus failing to capture valuable information that can be extracted from large data sets (Semenoglou, Spiliotis, Makridakis, & Assimakopoulos, 2021). The turning point for the perception towards ML methods in generic time series forecasting tasks was probably the M4 competition (Makridakis, Spiliotis, & Assimakopoulos, 2020) where the winning method mixed long short-term memory (LSTM) networks with standard exponential smoothing equations (Smyl, 2020), while the runner-up utilized XGBoost to optimally combine nine (mostly) statistical baseline forecasting models (Montero-Manso, Athanasopoulos, Hyndman, & Talagala, 2020). These results demonstrated the benefits of "cross-learning" and highlighted the potential value of ML methods.

Empirical evidence from forecasting competitions, along with greater accessibility to computer resources and ML libraries, also paved the way towards more advanced and accurate DL methods. Since the literature is vast, especially when application-specific methods are considered, below we focus on some generic DL forecasting methods that have become particularly popular in the recent years. Sen, Yu, and Dhillon (2019) proposed a DL method, based on temporal convolutions, that leverages both global learning from the entire data set and local learning from each time series separately. Salinas, Flunkert, Gasthaus, and Januschowski (2020) introduced DeepAR, a recurrent DL model that uses stacked LSTM layers to estimate the parameters of predefined distributions that represent the future observations of the time series being predicted, thus allowing the generation of both point and probabilistic forecasts. Lim, Arık, Loeff, and Pfister (2021) proposed Temporal Fusion Transformer (TFT), a deep neural network (NN) that utilizes LSTM encoders and a self-attention mechanism to generate multi-step forecasts. Finally, N-BEATS, introduced by Oreshkin, Carpov, Chapados, and Bengio (2019), involves deep NNs that build on fully-connected layers with backward and forward residual links. Apart from achieving state-of-the-art accuracy, N-BEATS results in interpretable forecasts.

It becomes evident that most state-of-the-art DL time series forecasting methods are based on recurrent neural networks (RNNs) that use numeric representations of time series data as input (Hewamalage et al., 2021), thus ignoring the advances reported for CNNs in computer vision. Indeed, although CNNs (Lai, Chang, Yang, & Liu, 2018; Shih, Sun, & Lee, 2019) and temporal CNNs (Van Den Oord et al., 2016) have recently become more popular for time series forecasting, they usually handle time series data as 1D numeric vectors instead of 2D images, which is surprising if we consider that CNNs have been originally designed for analyzing visual imagery.

Moreover, even when this is not the case, the NNs are typically used to develop classifiers or meta-learners that support the forecasting process and not to directly produce forecasts. For example, Cohen, Balch, and Veloso (2019) compared the performance of various classification models that used either numeric or visual representations of the S&P 500 index as input to predict whether the price will go up or down, concluding that the latter approach resulted in superior accuracy. In a similar classification task, Du and Barucca (2020) proposed a framework that converts the log return of financial time series to a gray-scale spectrum and uses a deep CNN to predict the future direction of prices. Cohen, Sood, Zeng, Balch, and Veloso (2020) introduced an image-to-image forecasting method where a convolutional autoencoder is used to process images of financial time series and provide the corresponding visual forecasts. Finally, Li, Kang, and Li (2020) explored the use of time series imaging for combining forecasts from different baseline forecasting methods based on the features that computer vision algorithms had extracted from each series.

CNNs have been considered alongside 2D inputs only in specific forecasting applications. For instance, Ma, Dai, He, Ma, Wang, and Wang (2017) leverage the temporal and spatial dependencies of traffic to create 2D images that are then introduced to a CNN-based model for predicting traffic speed. Zhang, Zheng, Qi, Li, Yi, and Li (2018) estimate crowd inflow and outflow in different regions of a city by employing a deep CNN, considering city grid maps of past inflows/outflows as input. Even in cases where no spatial dependencies exist in the data, 2D input vectors can still be constructed, provided that the available data set contains variables that are correlated with the target series. For example, Zhang and Guo (2020) encoded data from various variables that affect energy consumption as images in an attempt to predict more accurately the hourly electricity consumption at residential level. However, when tasked with forecasting sets of uncorrelated series without exogenous variables, the aforementioned approaches, used for constructing 2D inputs, become inapplicable. As a result, in univariate time series forecasting settings, like in our case, 1D CNNs become more relevant.

Despite the use of images in certain time series analysis tasks, to the best of our knowledge, no study has previously investigated the use of images as input to NNs to directly forecast a large set of uncorrelated time series, without additional explanatory variables. Given the potential benefits of such an approach, we believe that image-based time series forecasting requires more attention. The use of visual time series representations and deep 2D convolutions constitutes a novel approach for extrapolating patterns when compared to other types of NNs that solely rely on numeric input. Spatial structural information that is apparent in a visual representation of a time series offers a unique perspective, even if the same information is encoded in the original numeric data. An example of that comes from the way humans process information, being easier for most of us to recognize time series patterns like trends by looking at the plot of the series compared to reading through the subsequent observations. Another major advantage of treating time series as images comes from the breakthroughs in the field of computer vision. Research has provided key insights and deep architectures that enable the extraction of useful features from images that can be transferred in the forecasting domain with minimal adjustments and promising results. Finally, the image-based approach proposed in this paper is flexible and expandable depending on the application considered. For instance, the proposed methods can incorporate exogenous variables as additional features by including more channels as input. Motivated by the above, the contributions of this paper are threefold:

- We introduce an end-to-end, image-based DL approach for univariate time series forecasting, to be called *ForCNN*. We provide a detailed description of the overall framework that includes the necessary pre-processing steps and the architecture of the self-designed encoder used.
- We explore the potential value of well-established networks such as the ResNet-50 and VGG-19 as alternatives to the self-designed encoder. Leveraging such networks alleviates the burden of developing and optimizing new architectures by relying on already optimized and trained ones. The results suggest that it is possible to use such networks to generate accurate forecasts, thus further supporting the utilization of image-based forecasting approaches.
- We demonstrate the performance of the proposed approach through an empirical evaluation using two different sets of

series that consist of the monthly and yearly data of well-known forecasting competitions, namely the M3 (Makridakis & Hibon, 2000) and M4 (Makridakis et al., 2020). Both accuracy and computational cost metrics are considered to provide a comprehensive assessment, while various established methods, including state-of-the-art ML and DL models, are employed as benchmarks.

The rest of the paper is organized as follows: Section 2 introduces the methodological approach proposed for transforming time series data into images and the architecture of the models used for generating point forecasts. Section 3 describes the experimental design used to empirically evaluate the overall performance of the proposed approach. This includes the data sets used for training and testing *ForCNN*, the performance measures utilized, details on the implementation of *ForCNN*, and the benchmarks selected for assessing the relative performance of the proposed approach. Section 4 presents and discusses the results. Finally, Section 5 concludes the paper and provides directions of future research.

2. Methodological approach

A time series is defined as a set of data points listed in chronological order, with successive observations taken at regularly spaced intervals of time. A time series Y of length n can be expressed as:

$$Y_t = \{y_t \in \mathbb{R} \mid t = 1, 2, \dots, n\}, \tag{1}$$

where y_t represents the value of the time series at time t. Univariate time series forecasting aims at estimating the future values of the series based solely on its past observations. Solving this problem depends on discovering a function (or forecasting method) that approximates the underlying data generation process of the series, as follows:

$$\begin{gathered} f : \mathbb{R}^w \to \mathbb{R}^h \\ [y_{n+1}, \dots, y_{n+h}] = f(y_{n-w+1}, \dots, y_n) + [e_{n+1}, \dots, e_{n+h}] \end{gathered} \tag{2}$$

where $\{y_{n+1}, \dots, y_{n+h}\}$ are the future h observations of the series, f is the approximation function used to forecast these observations using w past values $(y_1, \dots, y_n)$ as input, and $\{e_{n+1}, \dots, e_{n+h}\}$ are the corresponding forecast errors. It becomes apparent that accurate forecasting requires methods that minimize forecast error.

The methodological approach proposed in this paper for forecasting time series through images consists of two phases. In the first phase, the time series data, originally provided as 1D numeric vectors, are pre-processed to be properly exported as 2D images. Each image corresponds to a particular window of the in-sample data of the series and serves as a substitute for the respective numeric input that would have been typically provided to standard ML or DL methods for training and forecasting purposes. In the second phase, the images created are used for training the deep NN, *ForCNN*, and producing out-of-sample forecasts.

2.1. Time series pre-processing

In order for all images to be extracted to depict time periods of equal length, the in-sample data of the time series are first divided into equal-sized windows of w observations each. Then, the min–max scaling is employed to adjust the window values in the target range of $[0, 1]$ and facilitate training across the numerous, diverse series of the data set (Shanker, Hu, & Hung, 1996). This means that, essentially, even different samples from the same series are treated as being independent from each other. Although the input values of each sample are always bound to the $[0, 1]$ range, the corresponding output values are not necessarily. Especially in tasks that involve forecasting heavily trended time series, this can be a useful attribute of the training set since it allows the NNs to learn that future values can be above or below any known value, across all samples. This approach has been successfully applied in order to create training data sets for similar forecasting problems in the past (Semenoglou et al., 2021; Smyl, 2020) and was therefore adopted in the proposed methodology.

Having scaled the original time series data, the next step of the pre-processing phase refers to the visualization, i.e. the transformation of the 1D numeric vectors into 2D images. This is done using simple line plots where the horizontal x-axis represents the time period each observation corresponds to and the vertical y-axis the scaled values of the observations. The plots are created and exported using the Matplotlib plotting library for Python.

Considering that the plotted line is the most important element in the created images, with its form and position containing the complete information required for generating forecasts, the width of the line is accordingly thickened to make the time series patterns more apparent. Other visual elements usually contained in plots, such as axes and legend, are not exported in the final images for the sake of simplicity. A monochromatic, black and white color scheme is chosen for creating the images, with the line representing the series being white and the background color being black. Thus, a single 8-bit integer value is used to represent each pixel instead of the three or more values typically required for each pixel in colored pictures. As a result, the redundant coloring information is eliminated, the input of the forecasting model is simplified, and the memory requirements for working with the images are significantly reduced. Finally, in order for all inputs to have the same dimensions, all images are resized to 64×64 pixels. Fig. 1 displays the final representations of 16 indicative time series that have been exported following the aforementioned pre-processing steps.

2.2. Model architecture

The images created in the pre-processing phase are provided as input to *ForCNN* in order for the DL network to be trained and then used to produce point forecasts for the time series of interest. The overall network consists of two modules, namely an encoder and a regressor, as shown in Fig. 2. Although these modules have distinct roles, they are trained concurrently as a single model.

Note that *ForCNN* serves as a generic framework for developing DL, image-based time series forecasting models. Thus, although in the following sections we will focus on a model that utilizes a self-designed encoder, to be called *ForCNN-SD*, we will later discuss how variants of this model can be constructed using other well-established deep CNNs. The architecture of *ForCNN-SD* is presented in Fig. 3.

2.2.1. Encoder

The objective of the first module of the NN, i.e. the encoder, is to transform each image X, provided as input to the network, into a vector W that contains a latent representation of X. This will allow the patterns of the series to be effectively filtered and the information that has to be learned by the network to be accordingly reduced in size to facilitate training without missing any significant knowledge. Given their successful use in numerous image related applications (Anwar, Majid, Qayyum, Awais, Alnowami, & Khan, 2018; Kamilaris & Prenafeta-Boldú, 2018; Rawat & Wang, 2017), this task is performed through layers that apply 2D convolutions to the input. Other advantages of 2D convolution layers that make them attractive for undertaking this

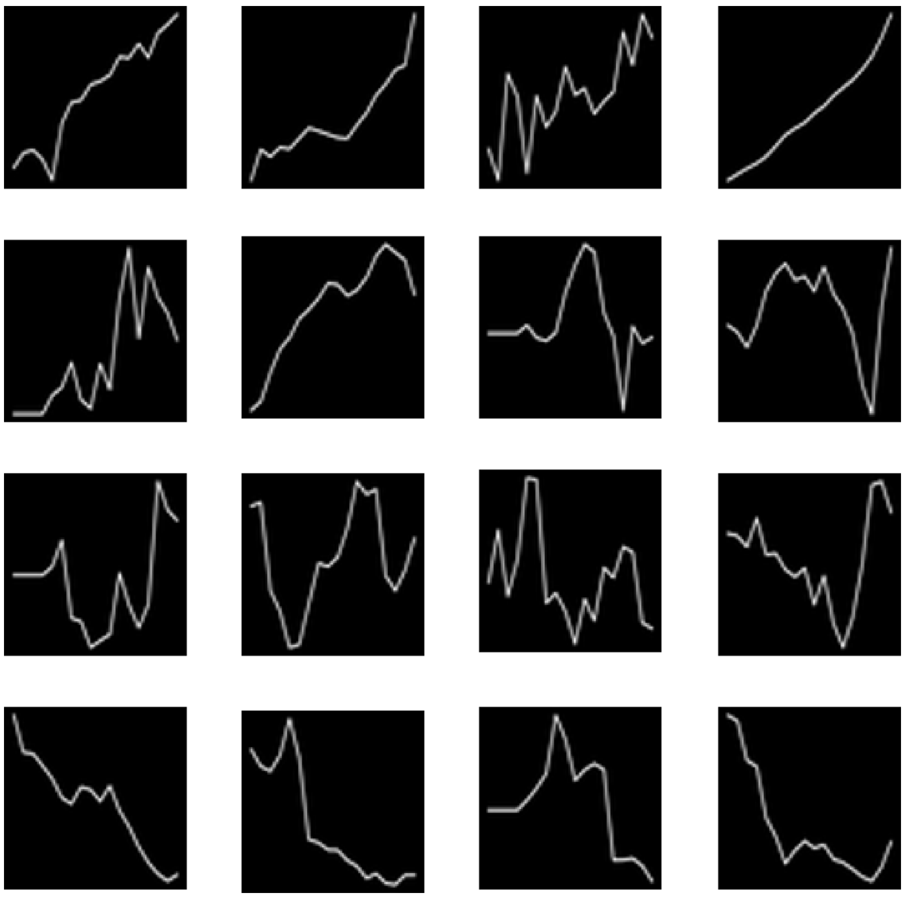

Fig. 1. Indicative visual representations of time series used by *ForCNN* as input, with w set to 18 observations.

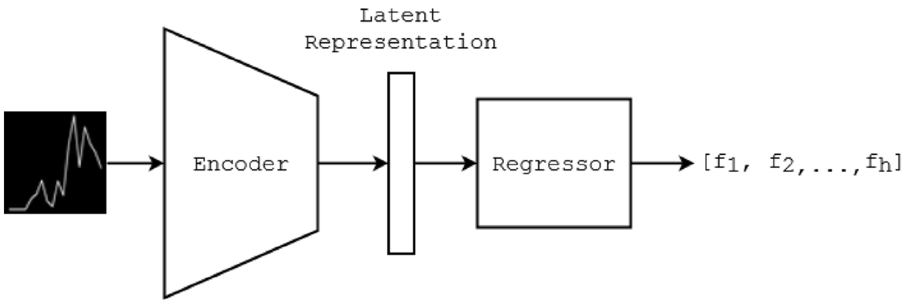

Fig. 2. Overview of the proposed image-based time series forecasting method, *ForCNN*, consisting of an encoder and a regressor module.

task include their ability to account for local dependencies around the pixels and their relatively few parameters that accelerate training without sacrificing learning capacity.

Specifically, in order to build the encoder of the network, we consider a deep convolutional architecture which is inspired by that of ResNet (He et al., 2016). Multiple studies and the overall trend in ML research favor deeper NN architectures when working with images (Chollet, 2017; Simonyan & Zisserman, 2015; Szegedy, Vanhoucke, Ioffe, Shlens, & Wojna, 2016) since the addition of more layers typically allows the model to progressively recognize and learn more specific patterns. At the same time, since deeper models are more difficult to train, they require additional components that allow them to handle issues related to vanishing gradients and degradation of training accuracy (He et al., 2016). To deal with these issues, ResNet utilizes "shortcut connections" between the layers that enable the input information of a block to be mapped directly to its output. Thus, when the identity mapping is considered optimal, the shortcut offers a direct way of achieving it rather than training a large number of parameters, used in non-linear transformations, to approximate the identity function.

Each convolutional layer of the encoder consists of a 2D convolution of its respective input (`Conv2D`) and uses 3 $\times$ 3 filters and zero padding to maintain the original input dimensions, followed by batch normalization (`Batch Norm`) and the application of the activation function, which is the Rectified Linear Unit (`ReLU`). The layers are then organized into blocks (`Block`), with each block having 3 convolutional layers and an identity shortcut from its input to its output. Note that the two information flows, i.e. from the main path of the block and the shortcut, are merged before the final application of the ReLU activation function. Finally, the residual convolutional blocks are organized into stacks (`Stack`). The number of convolutional filters used increases by a factor of 2 as more stacks are being added while the size of the feature maps

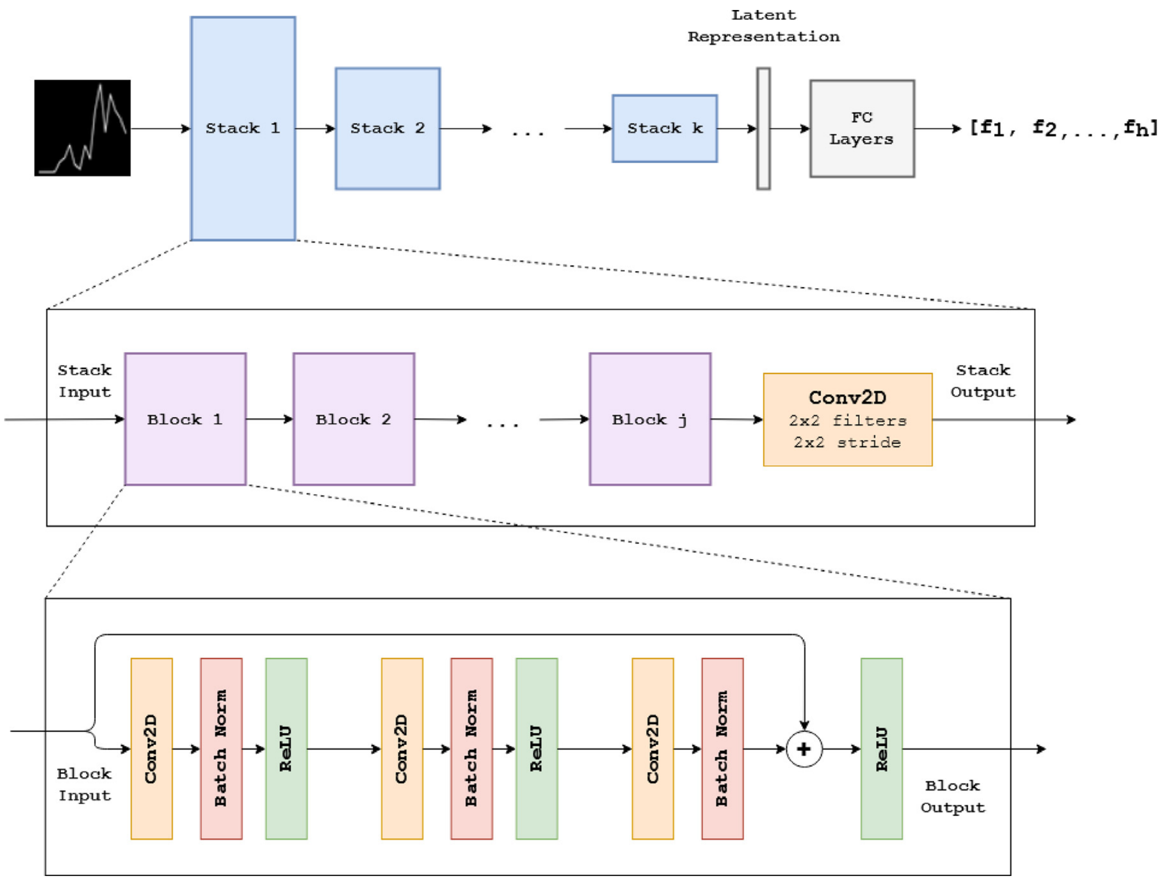

Fig. 3. The *ForCNN-SD* architecture. Top: The model consists of a series of `stacks` that take images as input and create latent representations of them. Fully-connected (`FC`) layers are then used to produce `h`-step-ahead forecasts based on the representations provided. Middle: Each `stack` consists of several convolutional `blocks` and a final convolution layer (`Conv2D`) for reducing the size of the exported feature maps. Bottom: Each block consists of three sets of convolution layers of the following transformations: 2D convolution (`Conv2D`), batch normalization (`Batch Norm`), and application of the `ReLU` activation function. A shortcut connection is used at the block level.

decreases by the same factor. Note also that instead of employing traditional pooling layers with fixed functions (e.g. max pooling or average pooling), 2D convolutions are used with 2×2 filters and a stride of 2, essentially decreasing the spatial size by the required factor. After the final stack, the resulting feature maps are concatenated and flattened to form the embedding vector W.

2.2.2. Regressor

The objective of the second module of the NN, i.e. the regressor, is to produce the requested forecasts F given the embedding vector W that has been created by the encoder. The regressor is implemented as a simple NN with fully-connected (FC) non-linear hidden layers (the ReLU function is used for transforming the input of the nodes) and a linear output layer. The forecasts are produced for all the forecasting horizons considered simultaneously, meaning that the nodes of the output layer are equal in size with the forecasting horizon examined.

Note that the regressor can be implemented using recurrent layers instead of the fully-connected layers proposed in *ForCNN-SD*. Recently, combinations of convolutional and LSTM-based architectures have been successfully introduced for time series forecasting (Livieris, Pintelas, & Pintelas, 2020; Xue, Triguero, Figueredo, & Landa-Silva, 2019) and a similar approach can possibly be exploited for use at an image-based model. However, selecting between using fully-connected or recurrent layers for the regressor is not a trivial process. Depending on the application, NNs that are based on dense layers can be as accurate as models based on recurrent layers, if not more. N-BEATS (Oreshkin et al., 2019), which is based on fully-connected layers, is an indicative example of such architectures, demonstrating that deep MLPs can achieve state-of-the-art performance in forecasting problems similar to ours. An additional point to consider is that LSTM-based architectures are inherently more complex to deploy effectively and require more time and computational resources as they have more parameters that need optimizing. Thus, in the present study, fully-connected layers were used in order to develop a regressor that is easy to implement, efficient in terms of computational time and resources needed for training purposes, and relatively accurate when used in batch time series forecasting tasks (Semenoglou et al., 2021).

2.2.3. Generalizing ForCNN

As noted earlier, *ForCNN-SD* adopts a specific, self-designed architecture. This is done in order for the proposed method to be tailored to the particular characteristics of the images used as input at the forecasting task at hand (e.g. single channel input of relatively simple images). However, the main framework can be adjusted according to the preferences of the user and the requirements of the forecasting task. For instance, although gray-scale line plots provide a clear and intuitive representation of the series, requiring also less memory to store and process, they may be replaced by recurrence or colored figures (Li et al., 2020). Similarly, the layers of the regressor can be extended or reduced in size depending on the complexity of the forecasting task and the length of the forecasting horizon. More importantly, the proposed encoder can be replaced by any other NN capable of extracting meaningful features from images.

In order to demonstrate the flexibility of the proposed method and further explore the potential value of image-based time series forecasting models, we proceed by considering two variants of the *ForCNN-SD* model, constructed by replacing the suggested encoder with well-established CNNs in the field of image recognition. To that end, we consider the ResNet-50 and VGG-19 DL models, to be called *ForCNN-ResNet* and *ForCNN-VGG*, respectively. Note that since ResNet-50 and VGG-19 are used just for extracting the embedding vector W from the images, the top FC layers of their architectures are dropped before being introduced to the *ForCNN* framework. Moreover, given that both models were originally developed for handling colored images, consisting of three channels, we simply repeat the constructed gray-scale images three times to match their input format. The pre-trained weights of ResNet-50 and VGG-19 on the ImageNet data set are used to initialize the encoder modules of *ForCNN-ResNet* and *ForCNN-VGG*, while random weights to initialize their regressor modules.

3. Experimental setup

3.1. Data set

In order to evaluate the forecasting performance of the proposed method, we consider two sets of series: the 23,000 yearly time series of the M4 competition data set (Makridakis et al., 2020) and the 1428 monthly series of the M3 competition data set (Makridakis & Hibon, 2000). Both data sets are comprised of multiple series from diverse domains (finance, micro, macro, industry, and demographics) that facilitate training and provide better generalization of our findings (Spiliotis et al., 2020a). Moreover, they are publicly available and well-established as benchmark sets, enabling the replication of our results (Makridakis, Assimakopoulos, & Spiliotis, 2018a) and their direct comparison to those of other studies claiming accuracy improvements using either standard or state-of-the-art forecasting methods (Makridakis, Spiliotis, & Assimakopoulos, 2018c).

In the case of the yearly series, following the original set-up of the M4 competition, the forecasting horizon, h, is set equal to 6 years and the number of the output nodes of the regressor module of the NNs is accordingly defined. Consequently, the last 6 observations of the series (out-of-sample) are hidden for the final evaluation of the models, while the rest of the observations (in-sample) are used for hyper-parameter optimization, training, and validation purposes. Also, a sliding window strategy is applied on the in-sample part of the series, where possible, in order to extract multiple windows from each series and maximize the number of available training and validation samples. The size of the windows used for creating the images, w, is set equal to $3 \times h = 18$ so that a reasonable number of observations is available for analyzing the patterns of the data and producing the corresponding forecasts. As a result, the final train set consists of approximately 235,000 samples, extracted from the in-sample part of the initial 23,000 yearly time series.

Similarly, in the case of the monthly series, the set-up of the M3 competition was followed for training and evaluating the examined models. The forecasting horizon, h, is set equal to 18 months. Thus, the last 18 observations of the series are hidden until the evaluation phase, while the rest of the observations are used for training and validating the models. Note that monthly series often exhibit strong seasonality that may be difficult for NNs to capture, especially when relatively short series are involved and no exogenous date-related information is provided (Barker, 2020; Zhang & Qi, 2005). Therefore, the classical multiplicative decomposition is applied for seasonally adjusting the series before further processing them, if needed, as determined by an autocorrelation-based seasonality test (Makridakis et al., 2020). The inverse transformation is then applied to re-seasonalize the forecasts of the series that were identified as seasonal. The sliding window strategy, as described above, is used on the in-sample part of the monthly series to create the train set of the models. The size of the input windows, w, is set equal to 36 observations (3 seasonal periods). As a result, the final train set consists of approximately 67,000 samples, extracted from the in-sample part of the initial 1428 monthly time series.

Note that the minimum and maximum values used in the pre-processing phase of the method for scaling the data are defined for each window separately using only the in-sample part of the series, i.e. the 18 and 36 observations of each window in the case of the yearly and monthly series, respectively. These values are then used for re-scaling the forecasts and measuring forecasting performance, either for training or testing purposes. Also note that, in the testing phase, if a window has less observations than what is required to produce forecasts, then additional data points are artificially added at its start using a naive backcasting approach, i.e. the first available data point of the window is used to fill the missing ones backwards. On the contrary, incomplete windows are dropped in the training phase.

3.2. ForCNN implementation

ForCNN-SD is implemented using Python 3.7.6, Tensorflow v.2.2.0, and the Keras API built on top of Tensorflow. Given the complex nature of the model and the large number hyper-parameters available, optimizing the architecture of the network and the training process is essential. In this respect, a hyper-parameter search is performed to determine the optimal values for the most critical hyper-parameters, based on the yearly series of the M4 competition. The Tree-of-Parzen-Estimators (TPE) algorithm is employed for performing the search, as implemented in the *HyperOpt* library for Python (Bergstra, Komer, Eliasmith, Yamins, & Cox, 2015). The optimal values, as determined by the search, are then used for building and training the final model. Appendix A contains a summary of the optimized hyper-parameters, the search range for each of them, and their final optimal values. Furthermore, for the two ForCNN variants, i.e. *ForCNN-ResNet* and *ForCNN-VGG*, the self-designed encoder was replaced by the pre-trained ResNet50 and VGG19 networks respectively, as implemented in Tensorflow's Keras API.

The optimal hyper-parameters values, as determined by the optimization process on the yearly series of the M4 competition, are also transferred to the models used for forecasting the monthly series of the M3 competition. As a result, it is possible to investigate whether the proposed DL architecture of *ForCNN-SD* and its variants (*ForCNN-ResNet* and *ForCNN-VGG*) has the ability to generalize well and adapt to data of different features.

Finally, to reduce the effect that random initial weights may have on the final forecasting performance of the proposed method, improve its accuracy and robustness, and draw more objective conclusions (Barrow, Crone, & Kourentzes, 2010; Kourentzes, Barrow, & Crone, 2014), we consider an ensemble of 50 different *ForCNN-SD* models and use the median operator to obtain the final forecasts, i.e. train *ForCNN-SD* 50 times and combine the outputs of the individual models for each forecasting horizon separately. In this regard, although all models used within the ensemble will have the same architecture and hyper-parameters, each model will display different initial weights and use different training and validation splits from the initial data set. The same ensembling strategy is utilized for *ForCNN-ResNet* and *ForCNN-VGG*.

Ensembling has long been considered as a useful practice in the forecasting literature (Bates & Granger, 1969) and numerous forecast combination schemes, both simple and sophisticated ones, have been proposed to exploit its full potential (for an encyclopedic review on forecast combinations, please refer to section 2.6 of Petropoulos et al., 2022). Simply put, since no forecasting model can consistently outperform all models available, combining the forecasts of multiple models, each making different assumptions about the data and having a different structure, can effectively improve overall forecast accuracy. By ensembling forecasts the errors of the participating models cancel each other, while model and parameter uncertainty is reduced (Petropoulos, Hyndman, & Bergmeir, 2018). As a result, the practice of combining has become particularly popular among the ML forecasting community as well, where bagging, boosting, and heavy ensembling are widely used to improve the performance of single models (Bojer & Meldgaard, 2021). These findings motivated the utilization of ensembling in the present study.

3.3. Benchmarks

We consider five different benchmarks to demonstrate the relative forecasting performance of the proposed approach, as described below.

- **Theta:** Theta (Assimakopoulos & Nikolopoulos, 2000) is a well-established and highly accurate statistical method that became popular due to its superior performance in the M3 competition (Makridakis & Hibon, 2000), being also the most accurate benchmark in the M4. Therefore, it is often considered a standard of comparison. The method was implemented as in the M4 competition.[1]
- **ES-RNN:** This method, proposed by Smyl (2020), was the winning submission of the M4 competition and the most accurate one for the yearly series of the respective data set, also used in the present study. The method is essentially a hybrid, mixing traditional exponential smoothing models with an LSTM-based architecture. Although methods that outperform the winning submission of a competition after the out-of-sample data have been released cannot claim any victory, such comparisons are useful for understanding their potential value over existing state-of-the-art methods. Note that, for the purposes of the present study, *ES-RNN* was not replicated. Instead, the original forecasts submitted to the M4 competition were used.[2]
- **MLP:** This method is an ensemble of 50, simple feed-forward NNs with fully-connected layers and serves as a standard "cross-learning" ML benchmark (Semenoglou et al., 2021). Similar to the proposed method, *MLP* is implemented using Python 3.7.6, Tensorflow v.2.2.0, and the Keras API. Following the steps outlined in Section 2, the numeric vectors, before being exported as images, are used as input to the *MLP*. This means that, for the case of the 1428 monthly series of the M3 competition, data were seasonally adjusted and then scaled. Accordingly, samples were simply scaled for the case of the 23,000 yearly series of the M4 competition. As for the output, the required multi-step forecasts are produced simultaneously, exactly as done by the regressor module of the *ForCNN* method. Although *MLP* is relatively simple in its nature, its architecture and training hyper-parameters were optimized using the TPE algorithm in order for the comparisons performed with the optimized *ForCNN* methods to be fair (for more details on the selected values please see Appendix A).
- **CNN-1D:** This method is an ensemble of 50 simple CNNs that use 1D vectors as input, similar to the *MLP*. Comparing the proposed image-based methods to *CNN-1D* is important, since it expands the discussion by considering an approach that uses convolutions on traditional numeric representations of the series. *CNN-1D* is implemented using Python 3.7.6, Tensorflow v.2.2.0, and the Keras API. As is the case with the *MLP*, the 1D numeric vectors are constructed by slicing the available series and then scaling the resulting windows based on their in-sample part. Also, for the case of the 1428 monthly series of the M3 competition, series were seasonally adjusted before being sliced and scaled. The multi-step forecasts required are produced simultaneously from the last dense layer of *CNN-1D*. The architecture and training process of *CNN-1D* were optimized using the TPE algori Appendix A provides a more detailed summary of *CNN-1D*'s architecture and training.
- **N-BEATS:** This method was proposed by Oreshkin et al. (2019) and is essentially an ensemble of 180 deep neural networks of fully-connected layers with backward and forward residual links. Currently, it is considered as the state-of-the-art approach in time series forecasting, and has been successfully used in various applications (Putz, Gumhalter, & Auer, 2021; Stevenson, Rodriguez-Fernandez, Minisci, & Camacho, 2022), including the second best submission of the most recent M5 competition (Anderer & Li, 2022). In the original paper, two N-BEATS configurations were presented based on the interpretability of the underlying models, N-BEATS-Generic and N-BEATS-Interpretable. For the purposes of this study we employ the N-BEATS-Generic variant (to be simply called *N-BEATS*) since it provides marginally more accurate forecasts, according to the original paper. Note that *N-BEATS* was configured and trained as proposed by the authors in the original study. The method was replicated using the official release provided.[3].
- **DeepAR:** This method was proposed by Salinas et al. (2020) and is an auto-regressive recurrent neural network model based on LSTM cells. It was initially developed as an architecture for probabilistic forecasting but it is also capable of providing very accurate point forecasts. DeepAR has been employed in several studies and applications (Mashlakov, Kuronen, Lensu, Kaarna, & Honkapuro, 2021; Park, Park, & Hwang, 2020), including the third best submission of the most recent M5 competition (Jeon & Seong, 2021), and is now considered a well-established standard of comparison. In order for the comparisons performed with the rest of the methods to be fair, *DeepAR*'s hyper-parameters were optimized using TPE algorithm in a fashion similar to that of the proposed imaged-based models (for more details on the selected values, please see Appendix A). Furthermore, 50 DeepAR models were trained and their forecasts were combined using the median operator. For the purposes of this study, the DeepAR models were implemented in Python, using the GluonTS modeling package, developed by Amazon (Alexandrov et al., 2019).

3.4. Forecasting accuracy measures

Enabling direct comparisons of our results with those of other relevant studies is of major importance. Hence, we choose to evaluate the forecasting accuracy of the presented models using the official measures of the M3 and M4 forecasting competitions. Specifically, the official measure used in the M3 competition was

[1] https://github.com/Mcompetitions/M4-methods/blob/master/Benchmarks%20and%20Evaluation.R.

[2] Forecasts retrieved from the GitHub repository of the M4 competition (https://github.com/Mcompetitions/M4-methods/tree/master/Point%20Forecasts).

[3] github.com/ElementAI/N-BEATS

the symmetric mean absolute percentage error (sMAPE; Makridakis, 1993), while in M4 the overall weighted average (OWA) of the sMAPE and the mean absolute scaled error (MASE; Hyndman & Koehler, 2006). The sMAPE, MASE, and OWA measures are defined as:

$$sMAPE = \frac{2}{h}\sum_{t=n+1}^{n+h}\frac{|y_t - f_t|}{|y_t| + |f_t|} \times 100\%, \tag{3}$$

$$MASE = \frac{\frac{1}{h}\sum_{t=n+1}^{n+h}|y_t - f_t|}{\frac{1}{n-1}\sum_{t=2}^{n}|t_t - t_{t-1}|}, \tag{4}$$

$$OWA_{Method} = \frac{\frac{sMAPE_{Method}}{sMAPE_{Naive2}} + \frac{MASE_{Method}}{MASE_{Naive2}}}{2}, \tag{5}$$

where f_t is the forecast of the examined method at point t, y_t the actual value of the series at the same point, and n the number of available historical observations. $sMAPE_{Method}$ and $MASE_{Method}$ refer to the overall forecasting accuracy of the examined method, as measured across the complete set of predicted series by sMAPE and MASE, respectively, with Naive2 being the benchmark used for scaling these scores and calculating OWA, as originally done in the M4 competition (Makridakis et al., 2020).

3.5. Computational cost measures

Apart from forecasting accuracy, another critical aspect for assessing the utility of sophisticated DL models is their computational cost. Depending on the application and the time or resources available, minor accuracy improvements may not always justify the increased cost. Thus, in order to provide a more comprehensive comparison of the proposed forecasting methods, we also report (in minutes) the time required for fitting the methods ($CT_{Training}$) and for extracting the forecasts given the fitted methods ($CT_{Inference}$), as well as the maximum memory usage in GB (Memory Usage). The measures are also reported for all the benchmarks considered.

Note that for the case of the *Theta* method, $CT_{Inference}$ is reported as the total time required for generating the forecasts since, when working with statistical approaches, it is difficult to distinguish between training and inference time. Also, for the case of the *ES-RNN* method, it was not possible to report computational demand, since the method was not replicated by us, as described in Section 3.3. When multiple models are trained as components of an ensemble, $CT_{Training}$ refers to the time required for training all participating models sequentially. Similarly, $CT_{Inference}$ refers to the time needed for calculating and combining the forecasts of all the models involved in the ensemble.

All forecasting methods were trained and evaluated using an Ubuntu 18.04.6 LTS server with the following characteristics: 2 Intel Xeon Gold 5118 CPUs @ 2.30 GHz, x64 based processor, 24 cores, 48 logical processors, 64.00 GB RAM, one Quadro RTX 5000 GPU with 16 GB RAM.

4. Results and discussion

The overall forecasting accuracy and computational cost of the proposed methods along with that of the considered benchmarks is summarized in Table 1 for the 23,000 yearly series of the M4 competition and in Table 2 for the 1428 monthly series of the M3 competition. The second, third, and fourth columns correspond to the scores reported according to the sMAPE, MASE, and OWA measures, respectively. The fifth, sixth, and seventh columns correspond to the training time, inference time, and maximum memory usage, respectively. Note that, as described in Section 2.2.3, three variants of the proposed method are examined, each one utilizing a different encoder for analyzing the images provided as input.

Table 1
Overall forecasting accuracy and computational cost of the proposed method and the considered benchmarks across the 23,000 yearly series of the M4 competition. Three variants of the proposed method are examined.

Model	sMAPE	MASE	OWA	$CT_{Training}$	$CT_{Inference}$	Memory usage
Benchmarks						
Theta	14.593	3.382	0.872	–	2.3	–
ES-RNN	13.176	2.980	0.778	–	–	–
MLP	13.089	2.954	0.772	351	1	3.0
CNN-1D	13.100	2.958	0.773	214	1	2.8
N-BEATS	13.114	2.936	0.771	4766	12	3.1
DeepAR	13.562	3.146	0.811	3317	1397	0.6
Examined models						
ForCNN-SD	**13.024**	**2.935**	**0.768**	3349	14	5.1
ForCNN-VGG	13.119	2.956	0.773	3715	11	5.8
ForCNN-ResNet	13.342	3.010	0.787	11038	18	6.0

The first, major finding of Table 1 is that using time series images for producing forecasts instead of conventional, numeric vectors, leads to notable accuracy improvements regardless the measure used for assessing forecasting performance. Specifically, *ForCNN-SD* provides 0.50%, 0.64%, and 0.52% more accurate forecasts than the *MLP* benchmark in terms of sMAPE, MASE, and OWA accordingly. Moreover, in comparison to *CNN-1D* that also uses convolutions on traditional time series representations, *ForCNN-SD* is 0.58%, 0.78%, and 0.65% more accurate, as measured by sMAPE, MASE, and OWA. When compared to *N-BEATS*, the top performing DL benchmark considered, the forecasts provided by *ForCNN-SD* are more accurate by 0.69%, 0.03%, and 0.38% according to sMAPE, MASE, and OWA, respectively. When compared against *ES-RNN*, the winning submission of the M4 competition, the accuracy improvements are even more significant, with *ForCNN-SD* being 1.15%, 1.51%, and 1.29% more accurate according to sMAPE, MASE, and OWA, respectively. Furthermore, *ForCNN-SD* provides 4.13%, 7.19%, and 5.60% more accurate forecasts than *DeepAR* in terms of sMAPE, MASE, and OWA. Finally, when compared to *Theta*, a strong statistical benchmark, *ForCNN-SD* displays even better results, as it is 10.75%, 13.22%, and 11.93% more accurate according to sMAPE, MASE, and OWA. At this point, it is important to note that *DeepAR*, *ES-RNN*, *MLP*, *CNN-1D* and *N-BEATS* are highly competitive benchmarks and significantly more accurate than any standard time series forecasting method. Therefore, any reported improvement over these benchmarks, even if small in absolute values, is of utmost importance. Thus, it is evident that the proposed method is a promising alternative for batch time series forecasting.

A second point of discussion concerns the performance of *ForCNN-VGG* and *ForCNN-ResNet*. *ForCNN-VGG* is less accurate than *ForCNN-SD* by 0.73%, 0.72%, and 0.65% as measured by sMAPE, MASE, and OWA. By the same measures, *ForCNN-ResNet* is also less accurate than *ForCNN-SD* by 2.44%, 2.56%, and 2.47%, respectively. Despite the fact that neither *ForCNN-VGG* nor *ForCNN-ResNet* outperform our proposed architecture, their accuracy is still noteworthy. *ForCNN-VGG* provides more accurate forecasts than both *ES-RNN*, *DeepAR* and *Theta*, while *ForCNN-ResNet* outperforms *DeepAR* and *Theta*. An explanation for the relatively worse performance of the two variants of *ForCNN* may lie in the fact that VGG and ResNet were originally designed to handle larger, colored images and for performing a different computer vision task (image recognition vs. pattern analysis). As a result, using such pre-trained NNs would probably require further adjustments to their architectures and the overall framework of the methodology to achieve more accurate forecasts. On the positive side, the results indicate that relatively simple, self-designed encoders can be effectively used to analyze time series images and improve forecasting performance.

Table 2
Overall forecasting accuracy and computational cost of the proposed method and the considered benchmarks across all 1428 monthly series of the M3 competition. Three variants of the proposed method are examined.

Model	sMAPE	MASE	OWA	$CT_{Training}$	$CT_{Inference}$	Memory usage
Benchmarks						
Theta	13.888	0.863	0.830	–	0.2	–
MLP	13.738	0.862	0.825	71	0.2	4.0
CNN-1D	13.757	0.866	0.827	106	0.2	4.0
N-BEATS	13.363	**0.813**	**0.790**	3032	9.0	2.5
DeepAR	13.572	0.868	0.823	9686	136	0.7
Examined models						
ForCNN-SD	13.359	0.833	0.800	3747	3.1	8.3
ForCNN-VGG	**13.228**	0.833	0.796	4087	3.0	8.3
ForCNN-ResNet	13.669	0.847	0.816	4264	3.9	8.6

The reported forecasting performances are the result of following the steps outlined in Sections 2 and 3 and are indicative of a forecasting pipeline that includes both a hyper-parameter optimization process for determining the "optimal" architecture and an ensemble of 50 randomly initialized models for eliminating the uncertainty from randomly initializing the NNs. Despite that, it is interesting to explore the sensitivity of the method's accuracy to different network hyper-parameter values and initializations regardless, especially for the case of the proposed *ForCNN-SD*. First, in order to assess the effect the different hyper-parameter values on the forecasting accuracy of *ForCNN-SD*, 10 additional *ForCNN-SD*'s variants were trained and evaluated based on the final configurations tested by the TPE algorithm. These configurations include encoders and regressors of different sizes, encoders with and without shortcut connections, and trainers with varying learning rates, patience values, and batch sizes. To make the comparisons fair, ensembles of 50 models were used when evaluating each *ForCNN-SD* configuration. The average forecasting accuracy of the different *ForCNN-SD* configurations was 13.043%, according to sMAPE, with a standard deviation of 0.040%. Based on the out-of-sample evaluation, the best configuration has an accuracy of 13.000%, while the worse configuration has an accuracy of 13.138%. It is interesting to note that the configuration suggested by the optimization algorithm A is not the best performing, with 4 out of the 10 configurations tested providing more accurate results. In addition to that, we examine the sensitivity of *ForCNN-SD* to random initializations. For that purpose, we train and evaluate 10 additional ensembles, consisting of 50 *ForCNN-SD* networks each, with different initial random seeds. All models share the same hyper-parameters, as determined by the original optimization process, making them comparable to the *ForCNN-SD* method reported in Table 1. The average accuracy of the forecasts across all tested ensembles was 13.025% in terms of sMAPE, with a standard deviation of 0.010%. The best performing *ForCNN-SD* ensemble has an accuracy of 13.010%, while the worse configuration has an accuracy of 13.043%. Overall, these findings confirm that the reported *ForCNN-SD* results are indicative and relatively insensitive to hyper-parameter selection or initialization.

The results of Table 2 indicate that, similar to the yearly data, *ForCNN-SD* can generate highly accurate forecasts when used for predicting monthly time series. However, in this case, the benefits of the proposed image-based approach are more limited. When compared to *N-BEATS*, the most accurate DL benchmark, *ForCNN-SD*'s forecasts are 0.03% more accurate according to sMAPE, but 2.40% and 1.19% less accurate according to MASE and OWA respectively. Yet, *ForCNN-SD* is still more accurate than the rest of the benchmarks considered. Specifically, it provides 1.59%, 4.20%, and 2.90% more accurate forecasts than the *DeepAR* benchmark in terms of sMAPE, MASE, and OWA, respectively. In comparison with *MLP*, *ForCNN-SD* generates 2.84%, 3.48%, and 3.16% more accurate forecasts, as measured by sMAPE, MASE, and OWA. Furthermore, *ForCNN-SD* outperforms the simpler convolutional benchmark, *CNN-1D*, by 2.89%, 3.81%, and 3.26% in terms of sMAPE, MASE, and OWA respectively. Finally, when compared to *Theta*, the winning submission of the M3 competition, *ForCNN-SD* exhibits even better results, as it is 3.96%, 3.63%, and 3.80% more accurate according to sMAPE, MASE, and OWA.

Another interesting finding from the evaluation on the 1428 monthly series is the increased accuracy of *ForCNN-VGG* compared to those of *ForCNN-SD*, as opposed to the case of the yearly series, where both *ForCNN-VGG* and *ForCNN-ResNet* variants were found to be less accurate than the self-designed architecture. Specifically, *ForCNN-VGG* is 0.99%, 0.00%, and 0.49% more accurate than *ForCNN-SD* according to MAPE, MASE, and OWA. When compared to *N-BEATS*, *ForCNN-VGG* is 1.02% more accurate in terms of sMAPE but 2.40% and 0.70% less accurate in terms of MASE and OWA. On the other hand, *ForCNN-ResNet* remains less accurate than *ForCNN-SD* by 2.32%, 1.68%, and 2.00%, as measured by sMAPE, MASE, and OWA, but outperforms the *MLP*, *CNN-1D* and *Theta* benchmarks.

To identify cases where *ForCNN-SD* may result in significantly more accurate forecasts compared to *N-BEATS*, the most accurate benchmark considered, and vice-versa, Fig. 4 presents a set of indicative examples of time series drawn from the M3 and M4 data sets. As seen, both *N-BEATS* and *ForCNN-SD* are highly capable of generating accurate forecasts, although in some cases they may either over- or underestimate trends.

In order to further investigate the differences reported between the proposed imaged-based DL methods and the benchmarks, we employ the multiple comparisons with the best (MCB) test (Koning, Franses, Hibon, & Stekler, 2005). The test computes the average ranks of the forecasting methods according to sMAPE or MASE across the complete data set of the study and concludes whether or not these are statistically different. Fig. 5 presents the results of the analysis. If the intervals of two methods do not overlap, this indicates a statistically different performance. Thus, methods that do not overlap with the gray interval of the figures are considered significantly worse than the best, and vice versa.

We find that, according to MCB, *ForCNN-SD* is significantly more accurate than the rest of the methods considered when used for forecasting the yearly series of the M4 competition. Moreover, we observe that *ForCNN-VGG* is similarly accurate to two state-of-the-art forecasting methods (*N-BEATS* and *ES-RNN*) and that *ForCNN-ResNet* manages to provide better forecasts than *DeepAR* and the statistical benchmark, *Theta*. The results are slightly different for the case of the M3 monthly series where *ForCNN-SD*, *ForCNN-VGG*, and *N-BEATS* are identified as similarly accurate, with the latter model displaying the lowest average rank.

In addition to MCB, we apply the model confidence test (Hansen, Lunde, & Nason, 2011), as implemented in the *MCS* package for R (Catania & Bernardi, 2017), that is better suited to understand whether it is possible to identify a superior set of methods. This bootstrap-based approach stops when one or more methods are identified as "superior" to all others given a certain level of confidence. The results of the test are summarized in Table 3. As seen, the findings of the model confidence test are in general agreement with those of the MCB test, identifying *ForCNN-SD*, *MLP*, *CNN-1D*, *N-BEATS*, and *ForCNN-VGG* as superior methods in the M4 data set, while *ForCNN-VGG*, *N-BEATS*, and *ForCNN-SD* as superior models in the M3 data set. Based on the above, we conclude that the results of Tables 1 and 2 are aligned with those of the statistical tests employed and that, although the benchmarks considered were competitive, including state-of-the-art implementations of ML and DL models, image-based time

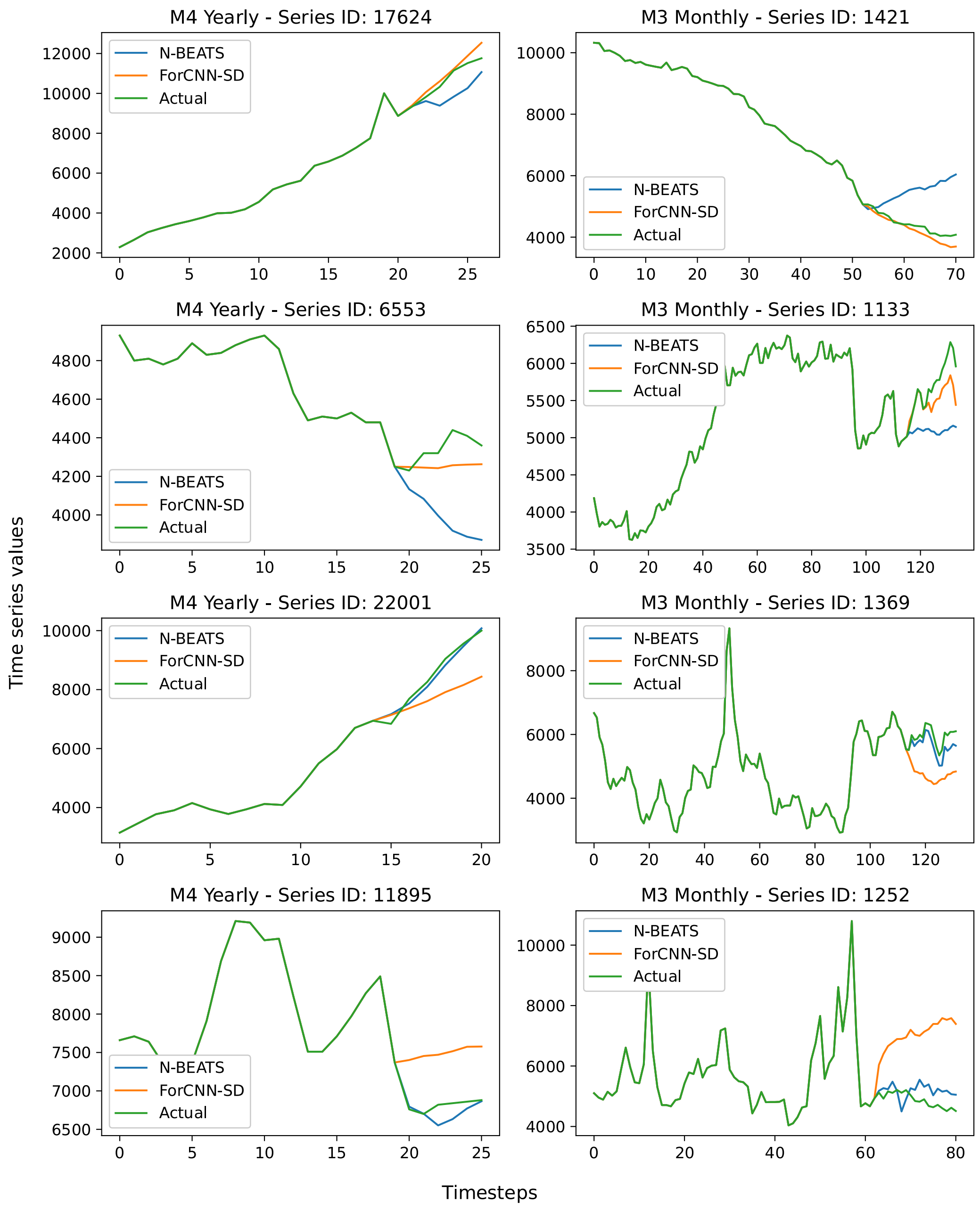

Fig. 4. Indicative plots of time series and the corresponding predictions provided by *ForCNN-SD* and *N-BEATS*. The green line represents the actual historical and future observations. The orange line represents the forecasts generated by *ForCNN-SD*, while the blue line represents the forecasts generated by *N-BEATS*. The top two rows show indicative cases where *ForCNN-SD* significantly outperforms *N-BEATS*. The bottom two rows show cases where *N-BEATS* significantly outperforms *ForCNN-SD*. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 3
Superior forecasting methods based on the model confidence test.

Measure	Superior set of methods
M4 Yearly	*ForCNN-SD*; *MLP* ; *CNN-1D* ; *N-BEATS*; *ForCNN-VGG*
M3 Monthly	*ForCNN-VGG*; *N-BEATS*; *ForCNN-SD*

series forecasting managed to provide similarly accurate, if not better results.

Finally, it is important to comment of the computational requirements of the examined methods, as reported in Tables 1 and 2. As expected, relatively more sophisticated DL methods require more time for training and generating forecasts compared to simpler ML and statistical methods, such as *MLP*, *CNN-1D* and *Theta*. Overall, *ForCNN-SD* is faster than *N-BEATS* and *DeepAR* by approximately 24 and 23 h, respectively, for the case of the 23,000 yearly series. When the 1428 monthly series are considered, *ForCNN-SD* is 9 h slower than *N-BEATS* but 105 h faster than *DeepAR*. Between the three *ForCNN* variants, *ForCNN-SD* has the lowest training time but *ForCNN-VGG* has the lowest inference time. Observe that the differences in training time are much greater (*ForCNN-SD* is faster than *ForCNN-VGG* by 6 to 9 h) than those in inference time (*ForCNN-VGG* is faster than *ForCNN-SD* by 4 min) and, as a result, *ForCNN-SD* is faster over the entire forecasting process. In contrast, *ForCNN-ResNet* requires significantly more time compared to the rest of the methods. Note, however, that computational time can be improved by parallelizing the training process. Furthermore, in terms of the memory usage, the three *ForCNN* variants have similar memory requirements and, overall,

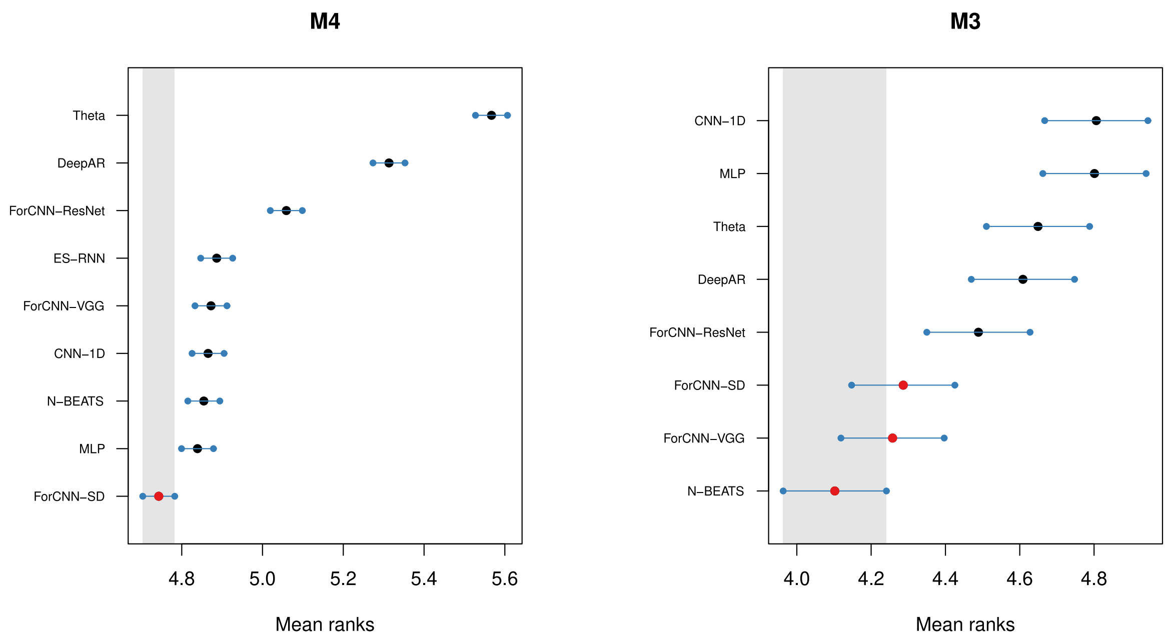

Fig. 5. Average ranks and 95% confidence intervals of the three *ForCNN* variants and benchmarks considered over the 23,000 M4 yearly (left) and 1428 M3 monthly (right) series. The multiple comparisons with the best test, as proposed by Koning et al. (2005), is applied using sMAPE for ranking the methods.

Table A.4
The architecture and the training hyper-parameters of the *ForCNN-SD* model proposed in this study, along with the considered search space for each hyper-parameter and the optimal values as determined by the optimization algorithm.

Hyper-parameter	Search space	Optimal value
Number of stacks	[2, 3, 4, 5]	5
Number of blocks	[2, 3, 4, 5]	3
Number of top FC layers	[1, 2, 3]	2
Shortcut connections	[True, False]	True
Optimizer	[Adam]	Adam
Learning rate	[0.0001, 0.0005, 0.001, 0.005, 0.01]	0.001
Max training epochs	[1000]	1000
Early stopping patience	[2, 5, 10]	10
Batch size	[64, 128, 256, 512]	64
Loss function	[MAE, MSE, sMAPE]	MAE

Table A.5
The architecture and the training hyper-parameters of the *MLP* used as benchmark in this study, along with the considered search space for each hyper-parameter and the optimal values as determined by the optimization algorithm.

Hyper-parameter	Search space	Optimal value
Number of hidden layers	[1, 2, 3]	3
Size of hidden layers	[0.5, 1.0, 1.5, 2.0] $\times$IS	1.5 $\times$ IS
Optimizer	[Adam]	Adam
Learning rate	[0.0001, 0.0005, 0.001, 0.005, 0.01]	0.001
Max training epochs	[1000]	1000
Early stopping patience	[2, 5, 10]	10
Batch size	[64, 128, 256, 512]	64
Loss function	[MAE, MSE, sMAPE]	MAE

Table A.6
The architecture and the training hyper-parameters of the *CNN-1D* used as benchmark in this study, along with the considered search space for each hyper-parameter and the optimal values as determined by the optimization algorithm.

Hyper-parameter	Search space	Optimal value
Number of convolutional layers	[1, 2, 3, 5]	3
Number of convolutional filters	[8, 16, 32, 64, 128]	16
Kernel size	[1, 2, 3]	3
Dilation rate	[1, 2, 3]	2
Optimizer	[Adam]	Adam
Learning rate	[0.0001, 0.0005, 0.001, 0.005, 0.01]	0.005
Max training epochs	[1000]	1000
Early stopping patience	[2, 5, 10]	10
Batch size	[64, 128, 256, 512]	128
Loss function	[MAE, MSE]	MAE

they demand more memory compared to traditional statistical, ML, and DL methods. These requirements can be attributed to the fact that images, being large 2D matrices, take up more space than the corresponding 1D numeric vectors.

5. Conclusions

This study investigated whether the use of visual time series representations and architectures inspired by the area of computer vision can lead to superior forecasting accuracy when compared to existing state-of-the-art time series forecasting methods. The proposed DL method, called *ForCNN*, which mixes convolutional and dense layers in a single neural network, constitutes an end-to-end solution for univariate time series forecasting that leverages the recent advances reported for the case of deep CNN when used for analyzing images and recognizing patterns.

Our results suggest that image-based time series forecasting can produce highly accurate forecasts and outperform both standard and advanced forecasting methods of either statistical or ML nature. When yearly series, typically dominated by trend patterns, are considered, the proposed *ForCNN-SD* outperforms all other benchmarks in terms of forecasting accuracy and computational time. Moreover, image-based approaches are able to generate highly accurate predictions when tasked with forecasting monthly series, outperforming most benchmarks and being on par with the top state-of-the-art DL method. More importantly, the results indicate that DL models may be able to handle more effectively visual time series representations than conventional, numeric ones to generate forecasts.

Pre-trained, well-established deep CNNs, like ResNet-50 and VGG-19, were also considered for analyzing the images provided

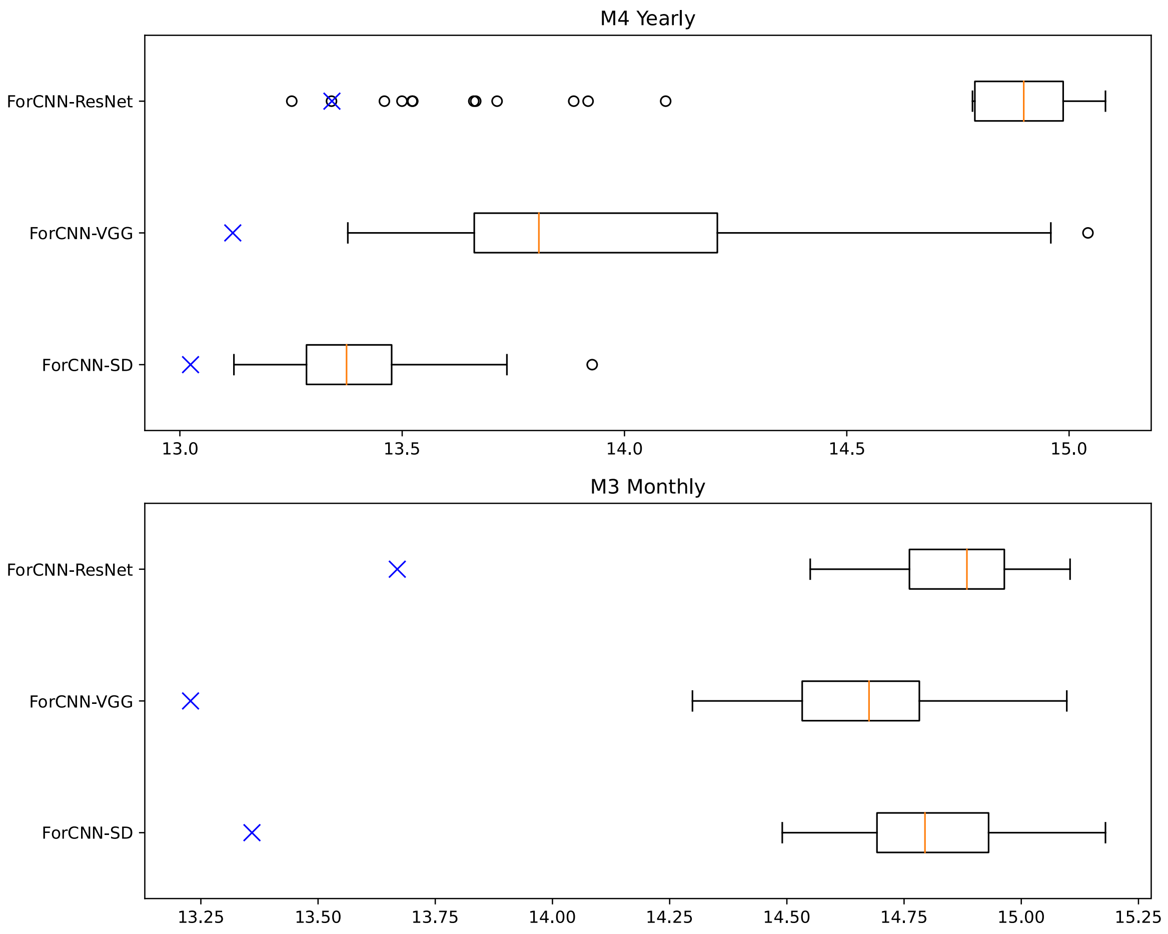

Fig. B.6. Distribution of sMAPE forecasting errors of the 50 networks participating in each of the ensembles. "X" marks the forecasting accuracy of the respective ensemble. The top figure corresponds to the results based on the 23,000 yearly series of the M4 competition. The bottom figure corresponds to the results based on the 1428 monthly series of the M3 competition.

Table A.7
The architecture and the training hyper-parameters of the *DeepAR* used as benchmark in this study, along with the considered search space for each hyper-parameter and the optimal values as determined by the optimization algorithm.

Hyper-parameter	Search space	Optimal value
Cell type	[LSTM, GRU]	LSTM
Number of recurrent layers	[1, 2, 3, 5 10]	3
Number of cells	[16, 32, 64, 128, 256, 512]	512
Dropout rate	[7%, 12%, 20%]	7%
Training epochs	[64, 128, 256, 512, 1024, 2048]	128
Number of batches per epoch	[32, 64, 128, 256, 512, 1024]	128
Batch size	[32, 64, 128, 256, 512]	128
Learning rate	[0.0001, 0.0005, 0.001, 0.005, 0.01]	0.005
Patience	[8, 16, 32, 64]	8

as input to the proposed method and eliminate the need for designing and optimizing new convolutional architecture from scratch. Between the two, the accuracy and computational cost of *ForCNN-VGG* was comparable to that of the self-designed encoder, while *ForCNN-ResNet*'s performance was less impressive, irrespective of the evaluation data set. Overall, our results suggest that using VGG-19 as a replacement for the encoder is indeed a viable option.

Our findings greatly motivate future research in the area of image-based time series forecasting, that could focus on, but not limited to, the following aspects:

- The present study considered two large, diverse sets of 23,000 yearly and 1428 monthly time series respectively, that originated from six particular domains (micro, macro, industry, finance, demographic and other). Although monthly series differ significantly from yearly ones, both data sets consist of continuous, relatively short series of low frequency. It would be interesting to examine how the proposed method performs for the case of high-frequency data (e.g. daily and hourly series) or intermittent and lumpy data, as well as, for cases of special forecasting applications (e.g. stock market, retail, and energy forecasting).
- Recurrence and colored plots could be used instead of simple, gray-scale line figures to extract more information from the series and provide more accurate forecasts.
- Additional work should be done to understand how existing, advanced DL models used in computer vision can be exploited for identifying time series patterns, extracting knowledge from large data sets, and using such knowledge to improve forecasting accuracy, while reducing computational cost.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Appendix A. ForCNN-SD, MLP, CNN-1D & DeepAR hyper-parameters

This appendix provides details on the hyper-parameter optimization of the proposed *ForCNN-SD* model, as well as, of the *MLP*, *CNN-1D* and *DeepAR* benchmarks. The Tree-of-Parzen-Estimators (TPE) algorithm is used on a validation set to determine the optimal set of values after 100 iterations for all models considered. Note that, in all cases, the optimal hyper-parameter values are determined based on the yearly series of the M4 competition and they are also used for forecasting the monthly series of the M3 competition. Table A.4 contains the hyper-parameters that are being optimized, along with the search space and their final, optimal values for *ForCNN-SD*. Tables A.5–A.7 contain similar information for the *MLP*, *CNN-1D* and *DeepAR* respectively.

Appendix B. Forecasting errors distribution

In this appendix we provide more details on the distribution of sMAPE errors of the individual networks that contribute forecasts to the ensemble. Note that the final forecasts of each *ForCNN* variant are calculated by combining the output of 50 networks of that architecture, as described in Section 3.2. Fig. B.6 shows the distributions of errors of the individual networks of the three *ForCNN* variants, for the cases of the yearly and the monthly data respectively. The forecasting accuracy of each variant's ensemble is also shown, as the corresponding "X" sign.

The first major conclusion from the analysis relates to the benefit of employing ensembles of networks, as opposed to using a single network. Even though this approach requires more computational time, the forecasting accuracy improvements are significant across all architectures and sets of series. Our results are aligned with existing literature in forecasting that highlights the importance of combining predictions from different models in order to make the final forecasts more robust. That is also the case for *N-BEATS* (Oreshkin et al., 2019) that is essentially an ensemble of 180 individual networks.

When considering the M3 monthly series, the forecasting errors of the networks are similarly distributed across the three *ForCNN* variants, with the respective ensembles being far more accurate in the comparison. In the case of the M4 yearly data, each variant's networks exhibit different patterns. According to the distribution of errors, the individual *ForCNN-SD* networks are more robust and their accuracy is closer to that of the final ensemble. On the other hand, *ForCNN-VGG* networks exhibit a wide range of forecasting errors, with the majority of the models being considerably less accurate than the respective ensemble. Finally, the large majority of *ForCNN-ResNet* networks have similar performance, being significantly less accurate than the ensemble. However, 2 out of the 50 networks generate forecasts that are at least as accurate as the forecasts of the final ensemble.

References

Alexandrov, A., Benidis, K., Bohlke-Schneider, M., Flunkert, V., Gasthaus, J., Januschowski, T., et al. (2019). GluonTS: Probabilistic time series models in python. arXiv preprint arXiv:1906.05264.

Anderer, M., & Li, F. (2022). Hierarchical forecasting with a top-down alignment of independent-level forecasts. *International Journal of Forecasting*, http://dx.doi.org/10.1016/j.ijforecast.2021.12.015.

Anwar, S. M., Majid, M., Qayyum, A., Awais, M., Alnowami, M., & Khan, M. K. (2018). Medical image analysis using convolutional neural networks: a review. *Journal of Medical Systems*, *42*(11), 226. http://dx.doi.org/10.1007/s10916-018-1088-1.

Assimakopoulos, V., & Nikolopoulos, K. (2000). The theta model: A decomposition approach to forecasting. *International Journal of Forecasting*, *16*, 521–530. http://dx.doi.org/10.1016/S0169-2070(00)00066-2.

Barker, J. (2020). Machine learning in M4: What makes a good unstructured model? *International Journal of Forecasting*, *36*(1), 150–155. http://dx.doi.org/10.1016/j.ijforecast.2019.06.001.

Barrow, D. K., Crone, S. F., & Kourentzes, N. (2010). An evaluation of neural network ensembles and model selection for time series prediction. In *The 2010 international joint conference on neural networks* (pp. 1–8). http://dx.doi.org/10.1109/IJCNN.2010.5596686.

Bates, J. M., & Granger, C. W. J. (1969). The combination of forecasts. *Journal of the Operational Research Society*, *20*(4), 451–468.

Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., & Cox, D. D. (2015). Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, *8*(1), Article 014008. http://dx.doi.org/10.1088/1749-4699/8/1/014008.

Bojer, C. S., & Meldgaard, J. P. (2021). Kaggle forecasting competitions: An overlooked learning opportunity. *International Journal of Forecasting*, *37*(2), 587–603.

Catania, L., & Bernardi, M. (2017). MCS: Model confidence set procedure. R package version 0.1.3.

Chae, Y. T., Horesh, R., Hwang, Y., & Lee, Y. M. (2016). Artificial neural network model for forecasting sub-hourly electricity usage in commercial buildings. *Energy and Buildings*, *111*, 184–194. http://dx.doi.org/10.1016/j.enbuild.2015.11.045.

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE conference on computer vision and pattern recognition* (pp. 1800–1807). http://dx.doi.org/10.1109/CVPR.2017.195.

Cohen, N., Balch, T., & Veloso, M. (2019). The effect of visual design in image classification. arXiv preprint arXiv:1907.09567.

Cohen, N., Sood, S., Zeng, Z., Balch, T., & Veloso, M. (2020). Visual forecasting of time series with image-to-image regression. arXiv preprint arXiv:2011.09052.

Du, B., & Barucca, P. (2020). Image processing tools for financial time series classification. arXiv preprint arXiv:2008.06042.

Feichtenhofer, C., Fan, H., Malik, J., & He, K. (2019). SlowFast networks for video recognition. In *2019 IEEE/CVF international conference on computer vision* (pp. 6201–6210). http://dx.doi.org/10.1109/ICCV.2019.00630.

Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, *270*(2), 654–669. http://dx.doi.org/10.1016/j.ejor.2017.11.054.

Frolov, S., Hinz, T., Raue, F., Hees, J., & Dengel, A. (2021). Adversarial text-to-image synthesis: A review. *Neural Networks*, *144*, 187–209. http://dx.doi.org/10.1016/j.neunet.2021.07.019.

Garcia-Garcia, A., Orts-Escolano, S., Oprea, S., Villena-Martinez, V., Martinez-Gonzalez, P., & Garcia-Rodriguez, J. (2018). A survey on deep learning techniques for image and video semantic segmentation. *Applied Soft Computing*, *70*, 41–65. http://dx.doi.org/10.1016/j.asoc.2018.05.018.

Hansen, P. R., Lunde, A., & Nason, J. M. (2011). The model confidence set. *Econometrica*, *79*(2), 453–497.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE conference on computer vision and pattern recognition* (pp. 770–778). http://dx.doi.org/10.1109/CVPR.2016.90.

Hewamalage, H., Bergmeir, C., & Bandara, K. (2021). Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, *37*(1), 388–427. http://dx.doi.org/10.1016/j.ijforecast.2020.06.008.

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *2017 IEEE conference on computer vision and pattern recognition* (pp. 2261–2269). http://dx.doi.org/10.1109/CVPR.2017.243.

Huber, J., & Stuckenschmidt, H. (2020). Daily retail demand forecasting using machine learning with emphasis on calendric special days. *International Journal of Forecasting*, *36*(4), 1420–1438. http://dx.doi.org/10.1016/j.ijforecast.2020.02.005.

Hyndman, R. J., & Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting*, *22*(4), 679–688. http://dx.doi.org/10.1016/j.ijforecast.2006.03.001.

Jeon, Y., & Seong, S. (2021). Robust recurrent network model for intermittent time-series forecasting. *International Journal of Forecasting*, http://dx.doi.org/10.1016/j.ijforecast.2021.07.004.

Kamilaris, A., & Prenafeta-Boldú, F. X. (2018). A review of the use of convolutional neural networks in agriculture. *The Journal of Agricultural Science*, *156*(3), 312–322. http://dx.doi.org/10.1017/S0021859618000436.

Karras, T., Laine, S., & Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *2019 IEEE/CVF conference on computer vision and pattern recognition* (pp. 4396–4405). http://dx.doi.org/10.1109/CVPR.2019.00453.

Ker, J., Wang, L., Rao, J., & Lim, T. (2018). Deep learning applications in medical image analysis. *IEEE Access*, *6*, 9375–9389. http://dx.doi.org/10.1109/ACCESS.2017.2788044.

Koning, A. J., Franses, P. H., Hibon, M., & Stekler, H. (2005). The M3 competition: Statistical tests of the results. *International Journal of Forecasting*, *21*(3), 397–409. http://dx.doi.org/10.1016/j.ijforecast.2004.10.003.

Kourentzes, N., Barrow, D. K., & Crone, S. F. (2014). Neural network ensemble operators for time series forecasting. *Expert Systems with Applications*, *41*(9), 4235–4244. http://dx.doi.org/10.1016/j.eswa.2013.12.011.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Vol. 25*, *Advances in neural information processing systems* (pp. 1097–1105). Curran Associates, Inc..

Lai, G., Chang, W.-C., Yang, Y., & Liu, H. (2018). Modeling long- and short-term temporal patterns with deep neural networks. In *The 41st international ACM SIGIR conference on research and development in information retrieval SIGIR '18* (pp. 95–104). Association for Computing Machinery, http://dx.doi.org/10.1145/3209978.3210006.

Li, X., Kang, Y., & Li, F. (2020). Forecasting with time series imaging. *Expert Systems with Applications*, *160*, Article 113680. http://dx.doi.org/10.1016/j.eswa.2020.113680.

Lim, B., Arık, S. O., Loeff, N., & Pfister, T. (2021). Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, *37*(4), 1748–1764. http://dx.doi.org/10.1016/j.ijforecast.2021.03.012.

Livieris, I. E., Pintelas, E., & Pintelas, P. (2020). A CNN–LSTM model for gold price time-series forecasting. *Neural Computing and Applications*, *32*(23), 17351–17360. http://dx.doi.org/10.1007/s00521-020-04867-x.

Ma, X., Dai, Z., He, Z., Ma, J., Wang, Y., & Wang, Y. (2017). Learning traffic as images: A deep convolutional neural network for large-scale transportation network speed prediction. *Sensors*, *17*(4), http://dx.doi.org/10.3390/s17040818.

Majumdar, A., & Gupta, M. (2019). Recurrent transform learning. *Neural Networks*, *118*, 271–279. http://dx.doi.org/10.1016/j.neunet.2019.07.003.

Makridakis, S. (1993). Accuracy measures: theoretical and practical concerns. *International Journal of Forecasting*, *9*(4), 527–529. http://dx.doi.org/10.1016/0169-2070(93)90079-3.

Makridakis, S. (2017). The forthcoming artificial intelligence (AI) revolution: Its impact on society and firms. *Futures*, *90*, 46–60. http://dx.doi.org/10.1016/j.futures.2017.03.006.

Makridakis, S., Assimakopoulos, V., & Spiliotis, E. (2018a). Objectivity, reproducibility and replicability in forecasting research. *International Journal of Forecasting*, *34*(4), 835–838. http://dx.doi.org/10.1016/j.ijforecast.2018.05.001.

Makridakis, S., & Hibon, M. (2000). The M3-competition: results, conclusions and implications. *International Journal of Forecasting*, *16*(4), 451–476. http://dx.doi.org/10.1016/S0169-2070(00)00057-1, The M3- Competition.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2018b). Statistical and machine learning forecasting methods: Concerns and ways forward. *PLOS ONE*, *13*(3), 1–26. http://dx.doi.org/10.1371/journal.pone.0194889.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2018c). The M4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, *34*, http://dx.doi.org/10.1016/j.ijforecast.2018.06.001.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2020). The M4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, *36*(1), 54–74. http://dx.doi.org/10.1016/j.ijforecast.2019.04.014.

Mashlakov, A., Kuronen, T., Lensu, L., Kaarna, A., & Honkapuro, S. (2021). Assessing the performance of deep learning models for multivariate probabilistic energy forecasting. *Applied Energy*, *285*, Article 116405. http://dx.doi.org/10.1016/j.apenergy.2020.116405.

Montero-Manso, P., Athanasopoulos, G., Hyndman, R. J., & Talagala, T. S. (2020). FFORMA: Feature-based forecast model averaging. *International Journal of Forecasting*, *36*(1), 86–92. http://dx.doi.org/10.1016/j.ijforecast.2019.02.011.

Naseer, S., Saleem, Y., Khalid, S., Bashir, M. K., Han, J., Iqbal, M. M., et al. (2018). Enhanced network anomaly detection based on deep neural networks. *IEEE Access*, *6*, 48231–48246. http://dx.doi.org/10.1109/ACCESS.2018.2863036.

Oreshkin, B. N., Carpov, D., Chapados, N., & Bengio, Y. (2019). N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. arXiv preprint arXiv:1905.10437.

Park, S., Park, S., & Hwang, E. (2020). Normalized residue analysis for deep learning based probabilistic forecasting of photovoltaic generations. In *2020 IEEE international conference on big data and smart computing (BigComp)* (pp. 483–486). http://dx.doi.org/10.1109/BigComp48618.2020.00-20.

Petropoulos, F., Apiletti, D., Assimakopoulos, V., Babai, M. Z., Barrow, D. K., Ben Taieb, S., et al. (2022). Forecasting: theory and practice. *International Journal of Forecasting*, *38*(3), 705–871.

Petropoulos, F., Hyndman, R. J., & Bergmeir, C. (2018). Exploring the sources of uncertainty: Why does bagging for time series forecasting work? *European Journal of Operational Research*, *268*(2), 545–554.

Putz, D., Gumhalter, M., & Auer, H. (2021). A novel approach to multi-horizon wind power forecasting based on deep neural architecture. *Renewable Energy*, *178*, 494–505. http://dx.doi.org/10.1016/j.renene.2021.06.099.

Rawat, W., & Wang, Z. (2017). Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, *29*(9), 2352–2449. http://dx.doi.org/10.1162/neco_a_00990.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., et al. (2015). ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, *115*(3), 211–252. http://dx.doi.org/10.1007/s11263-015-0816-y.

Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2020). Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, *36*(3), 1181–1191. http://dx.doi.org/10.1016/j.ijforecast.2019.07.001.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Semenoglou, A.-A., Spiliotis, E., Makridakis, S., & Assimakopoulos, V. (2021). Investigating the accuracy of cross-learning time series forecasting methods. *International Journal of Forecasting*, *37*(3), 1072–1084. http://dx.doi.org/10.1016/j.ijforecast.2020.11.009.

Sen, R., Yu, H.-F., & Dhillon, I. S. (2019). Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Vol. 32*, *Advances in neural information processing systems*. Curran Associates, Inc..

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & LeCun, Y. (2013). OverFeat: Integrated recognition, localization and detection using convolutional networks. arXiv preprint arXiv:1312.6229.

Shanker, M., Hu, M., & Hung, M. (1996). Effect of data standardization on neural network training. *Omega*, *24*(4), 385–397. http://dx.doi.org/10.1016/0305-0483(96)00010-2.

Shih, S.-Y., Sun, F.-K., & Lee, H.-Y. (2019). Temporal pattern attention for multivariate time series forecasting. *Machine Learning*, *108*(8–9), 1421–1441. http://dx.doi.org/10.1007/s10994-019-05815-0.

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International conference on learning representations*.

Smyl, S. (2020). A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, *36*(1), 75–85. http://dx.doi.org/10.1016/j.ijforecast.2019.03.017.

Spiliotis, E., Kouloumos, A., Assimakopoulos, V., & Makridakis, S. (2020a). Are forecasting competitions data representative of the reality? *International Journal of Forecasting*, *36*(1), 37–53. http://dx.doi.org/10.1016/j.ijforecast.2018.12.007.

Spiliotis, E., Makridakis, S., Semenoglou, A.-A., & Assimakopoulos, V. (2020b). Comparison of statistical and machine learning methods for daily SKU demand forecasting. *Operational Research*, 1–25.

Stevenson, E., Rodriguez-Fernandez, V., Minisci, E., & Camacho, D. (2022). A deep learning approach to solar radio flux forecasting. *Acta Astronautica*, *193*, 595–606. http://dx.doi.org/10.1016/j.actaastro.2021.08.004.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Tan, M., & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In K. Chaudhuri, & R. Salakhutdinov (Eds.), *proceedings of machine learning research*: *Vol. 97*, *Proceedings of the 36th international conference on machine learning*. PMLR.

Tian, C., Fei, L., Zheng, W., Xu, Y., Zuo, W., & Lin, C.-W. (2020). Deep learning on image denoising: An overview. *Neural Networks*, *131*, 251–275. http://dx.doi.org/10.1016/j.neunet.2020.07.025.

Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., et al. (2016). WaveNet: A generative model for raw audio. In *The 9th ISCA speech synthesis workshop* (p. 125).

Wang, J., & Wang, J. (2017). Forecasting stochastic neural network based on financial empirical mode decomposition. *Neural Networks*, *90*, 8–20. http://dx.doi.org/10.1016/j.neunet.2017.03.004.

Xue, N., Triguero, I., Figueredo, G. P., & Landa-Silva, D. (2019). Evolving deep CNN-LSTMs for inventory time series prediction. In *2019 IEEE congress on evolutionary computation* (pp. 1517–1524). http://dx.doi.org/10.1109/CEC.2019.8789957.

Young, T., Hazarika, D., Poria, S., & Cambria, E. (2018). Recent trends in deep learning based natural language processing [review article]. *IEEE Computational Intelligence Magazine*, *13*(3), 55–75. http://dx.doi.org/10.1109/MCI.2018.2840738.

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In D. Fleet, T. Pajdla, B. Schiele, & T. Tuytelaars (Eds.), *Computer vision – ECCV 2014* (pp. 818–833). Cham: Springer International Publishing.

Zhang, G., & Guo, J. (2020). A novel ensemble method for hourly residential electricity consumption forecasting by imaging time series. *Energy*, *203*, Article 117858. http://dx.doi.org/10.1016/j.energy.2020.117858.

Zhang, G., & Qi, M. (2005). Neural network forecasting for seasonal and trend time series. *European Journal of Operational Research*, *160*(2), 501–514. http://dx.doi.org/10.1016/j.ejor.2003.08.037.

Zhang, J., Zheng, Y., Qi, D., Li, R., Yi, X., & Li, T. (2018). Predicting citywide crowd flows using deep spatio-temporal residual networks. *Artificial Intelligence*, *259*, 147–166. http://dx.doi.org/10.1016/j.artint.2018.03.002.

Zhao, Z., Zheng, P., Xu, S., & Wu, X. (2019). Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, *30*(11), 3212–3232. http://dx.doi.org/10.1109/TNNLS.2018.2876865.