Assessment of LSTM, Conv2D and ConvLSTM2D Prediction Models for Long-Term Wind Speed and Direction Regression Analysis

Zaccheus Olaofe (zakky201@gmail.com)
University of Cape Town

Research Article

Keywords: LSTM Model, Convolutional Neural Network, Wind Speed Forecasts, Sectorwise Wind Direction, ConvLSTM2D Model, Keras Sequential Model

Posted Date: October 27th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-1011778/v1

Assessment of LSTM, Conv2D and ConvLSTM2D Prediction Models for Long-Term Wind Speed and Direction Regression Analysis

Abstract: This paper assessed the model performance accuracies of 3 forecast-based architectures (Long Short-Term Memory, LSTM; Convolutional Neural Network, Conv2D and hybrid ConvLSTM2D) for multivariate inputs to multi-steps wind speed and direction forecasts. These high-level neural network-based architectures were setup with the Keras sequential models trained to learn the historical patterns from the processed weather input datasets. To build these forecast models, the sampled time series weather observations at different station heights were obtained and reshaped for network layer compatibility, while the Adamax algorithm for the network optimization was considered. The trained and evaluated model performances with different input data sequences (normalized/un-normalized) were assessed while the forecast results were also compared with the Actual and Conv1D models. Upon optimal network training, the Conv2D model returned MSE, MAE and RMSE estimated values of 0.82, 4.48 and 0.91 %, respectively; the LSTM model returned 1.03, 4.75 and 1.01 %; while the ConvLSTM2D model returned 2.11, 10.13 and 1.45 %, respectively. Also, Conv2D validated model values of 3.16, 14.73 and 1.77 % were obtained %, respectively; 3.21, 14.98 and 1.82, for the LSTM-based; while ConvLSTM2D model returned 3.27, 15.92 and 1.91 %, respectively. Studied finding results show that better prediction and evaluation could be achieved for all the trained model architectures as compared to the untrained models. Also, from the predicted model results, the Keras sequential models were found to be useful for replicating the time-series historical wind speed and direction based on the well-tuned model hyperparameters as well as the input sequence structure

Keywords: LSTM Model; Convolutional Neural Network; Wind Speed Forecasts; Sectorwise Wind Direction; ConvLSTM2D Model; Keras Sequential Model

1.0 Introduction

In recent studies, the Long Short-Term Memory (LSTM), Recurrent Neural Network (RNN), Gated Recurrent Unit (GRU), Convolutional Neural Network (CNN) and hybrid model (LSTM-CNN) have been widely deployed in many machine-learning frameworks (short- to long-term application) [1–5]. Also, these forecast neural networks had shown different state-of-the-art results based on their selected network optimizers [6–11] for machine-learning algorithms and the model hyperparameters tuning techniques [12–16]. Within the deep-learning framework, both the network optimizer and tuning approach have played a critical role in the setup of an accurate forecast system as they do impact the model learning ability and forecasts skills.

Among these prediction models, the LSTM network have been preferred and utilized in complex frameworks for the processing of long-term non-linear input data sequence as compared to other state-of-the-art models. This choice was solely based on LSTM gating mechanism (memory cells) required for better feature extraction and modeling of long-term dependencies of the input sequence [17,49]. Although, it is well known in machine-learning that achieving a reliable state-of-the-art result with LSTM-network tuning is a difficult task as it increased the model complexity [18] (that is, it required the right selection and optimization of different model hyperparameters [11]) and computational time. Notwithstanding, this forecast-model have been found to be more efficient and reliable with optimally tuned hyperparameters [19]. Lastly, the sequential model learning adaptivity in extracting hierarchical local features [20] from a given input sequence had made the LSTM model competitive with existing machine-learning architectures [21] and implementable in various fields: petroleum production forecasts [22], Stock/financial markets forecast [23–24], workload predictions for cloud monitoring [25,26], text processing [27–29], speech recognitions [30–31], short-term photovoltaic and temperature forecasts [32–39], micro-pollutants forecasts in watershed [40], among others.

In comparison studies, the Convolutional networks have been also considered as a competitive forecast model with LSTM: - [1]due to fast/timely convergence, [2]easily trained forecasts model, and [3]ability to interpret the learnt features for a given input sequence. From the Keras sequential models, Baccouche et al [41] utilized the sequential Conv3D (3-D CNN) model for human action recognitions. Also, Keren et al [17] utilized the convolutional-RNN with the sequential input data within the audio classification frameworks. Yang et al [42] deployed the Sequential-CNN to extract effective spatial-temporal features for the given video frames. Ngoc Vu [18] considered the SCNN for slot filling task in spoken language understanding. Vinyals et al [43]and Ciresan et al [44] utilized the CNN models for image caption generation and traffic sign recognition, respectively. From the aforementioned studies, these forecast models had emerged and outperformed some existing forecasting networks (RNNs, ARIMA, self-organizing map, support vector machine, among others) in: [1]time series predictions, [2]regression and [3]classification analysis [32, 45]. Despite the wide applications of different forecasts model architectures in learning processes of the spatial-temporal features/patterns of a given input sequence, [1]the selection and optimal tuning of different model hyperparameters and

[2]model layers internal covariate shift in distributed input sequence are the main limitations of the deep-learning models [46]. To mitigate the internal shift effects within the network layers, Molina et al and Saxena [47–48] in their findings suggested the batch normalization technique as one useful method for correcting the internal distribution shifts between the model input and flatten layers. Nevertheless, the neural network architectures built with the Keras sequential models in TensorFlow had shown significant improvement and have been recently deployed in the deep-learning frameworks [11,17, 49] than the conventional forecast models [50–55]. With the Keras neural networks, different model hyperparameters could be optimally tuned with the model layers internal covariate shift being easily resolved by incorporating 'batch normalization layer' into the traditional LSTM and CNN model layers.

The Keras sequential model is a linear-stack of layers (model layer by layer) that can be concatenated for building a high-level neural network architecture [26]. As an important, emerging and promising tool in energy assessment and trading forecasts, different forecast models in the Keras Library [56] are available for designing the reliable forecast-based architectures [11, 17, 26, 57]. In few recent studies, Malakar et al [32] had designed the LSTM model for short-term solar energy predictions based on concatenated 4 sequentially arranged layers (1-input, 2-hidden and 1-output); Kong et al [58] utilized the LSTM network model for residential short-term load predictions within the deep-learning frameworks; Sutskever et al [57] utilized the LSTM neural network in machine translation; Reimers et al [11] adopted the LSTM model for sequence tagging. Xie et al [72] also utilized the multi-variable LSTM to predict the short-term wind speeds. Their studied findings revealed that the Keras sequential models were essential in short to long-term forecast projections based on its deep-learning ability [59–60]. However, for the multivariate inputs to multi-steps wind speed and direction forecasts, we couldn't find any extensive work as the Keras neural networks in long-term wind predictions had not been explored within an African context and we adopt the concept of batch normalization technique as one solution in addressing the ConvLSTM2D model covariate shifts as well as assessing the accuracy of the Keras neural networks in wind speed and direction projections.

For this present study, we proposed/assessed and evaluated 3 neural-network architectures (Conv2D, LSTM and ConvLSTM2D) built with Keras sequential models (TensorFlow 2.0) for projections of the time series wind speed and direction at some selected weather stations. Each Keras neural network is developed with linear stack of network layers complied together (Fig 1) as essential forecast model in wind studies. Due to [1]high-level model computations, [2]huge cost of ConvLSTM2D setup and [3]the model poor performance of interconnected layers if trained with long-term non-linear input datasets, the batch training approach is considered as a better option. Thus, batches of input data arrays from weather datasets into the Keras neural networks are fed and the hidden layer units are allowed to extract local features from an input sequence. In preparation of the forecast models for deep-learning frameworks, the historical time series variables of 3 stations are normalized (scaled-down) and utilized as training, validation and testing input data arrays, respectively; while model input dataset from 5 additional stations are obtained for network evaluations of trained and untrained (compiled layers model without training procedures). Also, the effect of overlearning of the developed neural networks with historical weather input dataset is considered with dropout regularization between 20 and 40% [37]. Furthermore, for model hyperparameters tuning and selection of reliable forecast system, the [1]model parameters, [2]learning-rate of network optimizer and [3]layers activation functions are rightly selected and initialized (Figs 1, S1–2); while the model training and validation losses are monitored. Lastly, the model prediction accuracy of Conv1D with unscaled inputs at 300 epochs (Fig S4) and scaled input arrays at 100 epochs (Fig S5 a–b) are compared with the LSTM performance at 100 epochs (Fig S9 a–b). Based on the model performances, the models forecast accuracy with the actual model is assessed ([1]time series wind speed and direction for 0-200 timesteps, [2]wind roses and the frequency occurrences in 12 wind sectors, [3]trained and untrained network models accuracy) for right selection and recommendation of a reliable forecast model in wind speed projection. Hence, the studied objectives are centered on the development of high-level Keras neural networks within the deep-learning framework that would be**:** – (1) utilized as stand-alone forecast model in energy assessment based on historical wind conditions of a given site; and lastly, to determine if the Keras neural networks for timely convergence could be deployed in deep-learning framework as the reliable forecast tools in replicating the historical wind speed and direction patterns at a given time horizon.

Following this introduction, the rest of the paper is structure: - section 2 describes the monitoring stations and data pre-processing; section 3 explains the methodology for: [1]model input data preparation, [2]Keras sequential model setup and development, [3]assessment of the model metrics (training and prediction accuracy). Lastly, the studied finding results and discussion, conclusion and direction for further studies relating to the developed forecast models are all presented in sections 4 and 5, respectively.

2.0 Stations/Data Description

The multivariate time series observations of different sampled rates (5–min/10–min/1–hour) and heights (2/10/20/60

m) AGL at 8 stations (Col 1 of Table 1) were continuously monitored by the weather sensors deployed at the South African Weather Stations (Paarl, Geelbek and Darling) and the South African Wind Atlas stations (WM01, WM02, WM03 and WM04). For the sampled periods, the historical time series of weather variables such as: wind directions at 10 (WD_10) and 20 m (WD_20); air temperature over 1–min at 2 m (T_{air}/Temp), 10 m (TS_10) and 60 m (TS_60); temperature gradient (T_{grad}); relative humidity (Hum/Rhum) over 1–min rate; wind speeds at 10 (WS_10) and 20 m (WS_20); as well as the gust (highest wind) at 10 m height (Gust_10) were collected and processed for model system architectures (Tables 2–4).

For each selected station, time series of 4 variables were obtained (Col. 4) and separated into 3–input variables (Col. 5) and 1–output variable datasets (Col. 6). Data cleansing were carried out by checking for missing data-point in each variable for each station height. However, in the collected dataset, there was no missing data point. Meanwhile, for the proposed forecast models (LSTM, Conv2D and ConvLSTM2D), the 3-input variables in 2-dimensional format (rows and columns) were used to prepare the model input data array (Cols. 8–13) while the 1-output variable (Col. 6) was used to prepare the model output array (Col. 14). Thus, the adopted technique for reshaping the time series station variables into: LSTM model input array (Col 11); Conv2D input array (Col 12); and ConvLSTM2D model input array (Col. 13) with its corresponding model output array (Col. 14) is discussed (section 3.1).

The prevailing wind flow(s) of the considered stations are: North–Easterly, South–Easterly, Westerly and South–Westerly to North–Westerly as shown (Fig. S3).

3.0 Methodology

The forecast accuracy of a developed sequential model is often based on the right selection and tuning of the network hyperparameter, model system architecture as well as the data-quality of input sequence. To make a reliable forecast of the local wind speed and sectorwise direction, the time series of a high-quality historical datasets (given sequence) is required. Thus, high-quality time series of past observations for the sampled periods were obtained, scaled–down and reshaped accordingly to the model specification of accepting input arrays into the network layer as discussed below: -

3.1 Input/Output Data Structure

In deep learning forecast model, a high-level neural network requires input data arrays that would be passed from the input nodes into the hidden layer so that the network model could process and learn the historical pattern, and makes a reasonable prediction. To ensure that the proposed sequential neural model trains well and converge quickly, the time-series of each selected station variable was normalized to similar range of values. Also, because the sampled time-series observations were noisy and stochastic in nature, the input data arrays at different sampled rates would negatively impacts the model training and validation performances. Hence, data normalization to improve the model generalization ability was also considered. As shown (Fig S4) in the Conv1D model training and validation losses, un–normalized datasets as model input arrays into 1st network layer didn't allow the gradient descents to converged and prevented the model deep learning at 300 iterations. Also, the plots of model performances with normalized input arrays (Fig. a–b of S5) show that the gradient descents converged quicker and generalized with model input data arrays at 100 iterations [61].

The time series of multivariate input variables for model training, validation and testing (100000x3) as well as for evaluation (41200x3) were obtained (Cols. 5 and 7 of Table 1) at the station heights (Col 1). For model input variables such as: - [1]wind direction (0–360°) [2]air temperature (0.3–46° C) and [3]relative humidity (9.7–93.7 %), these variables were scaled-down to similar values ranging from 0–1[26, 32]: –

$$X = \frac{(X_i - X_{min})}{(X_{max} - X_{min})} \quad (1)$$

where X_i, is the station variable value (> 0) at a given instance i; X_{min} and X_{max} are the minimum and maximum values per station variable, respectively; X is the normalized station datasets.

For Conv2D-based network (Fig 1a), this required a 4-dimensional (4-D) input data (sequence) with its corresponding 3-D input shape. The Conv2D input shape for the input-layer was obtained using an input_shape argument below: –

$$\text{Input\_Shape\_Conv2D} = (\text{batch}, \text{timesteps}, \text{features}) \quad (2)$$

where the batch =1, timesteps = number of row(s) per input variable and also denoted as $t_1, t_2, t_3, .., t_n$ for n = 100000; features = 3 (no of columns of input variables). From Eq. (2), the training, validation and testing data arrays had 1 batch, 100000 timesteps and 3 features; while the input shape of evaluation data array had 1 batch, 41200 timesteps and 3 features (Col. 9).

Note: For the Conv2D model, information about its input_shape structure (Input_Shape_Conv2D) was passed to the input (1st) layer while the subsequent model layers do automatic shape inference within the neural network (see block diagram of Fig 1a).

From Eq. (2), the 4–D model input data arrays into the Conv2D layer were obtained: –

$$\text{Input\_Data\_Conv2D} = \text{X.reshape(samples, Input\_Shape\_Conv2D)} \quad (3)$$

$$= \text{X.reshape(samples, batch, timesteps, features)} \quad (4)$$

where the samples =1 and X = station normalized datasets (Eq. 1). From Eq. (4), the model training, validation and testing input arrays were obtained as: x_train_mod = X_tr.reshape(1, 1, 100000, 3), x_val_mod = X_va.reshape(1, 1,100000,3) and x_test_mod = X_te.reshape(1,1,100000,3). For the evaluation input arrays (SM–ST1 to SM–ST5), these were obtained as X_ev.reshape(1,1,41200,3) as presented.

Meanwhile, a 4-D input shape to the 1st network layer of the ConvLSTM2D model was obtained:

$$\text{Input\_Shape\_ConvLSTM2D} = (\text{batch, timesteps, features, channels}) \quad (5)$$

where the channel set-value is 1 (channels =1). The summary of the 4-D input shape was presented (Col. 10).

From Eq. (5), the 5-D model input arrays into the ConvLSTM2D network layer were obtained: –

$$\text{Input\_Data\_ConvLSTM2D} = \text{X.reshape(samples, Input\_Shape\_ConvLSTM2D)} \quad (6)$$

$$= \text{X.reshape (samples, batch, timesteps, features, channels)} \quad (7)$$

Note: The 5–D input arrays of the ConvLSTM2D model were obtained as: x_train_mod = X_tr.reshape(1, 1, 100000, 3, 1), x_val_mod = X_va.reshape(1, 1, 100000, 3, 1) and x_test_mod = X_te.reshape(1,1,100000,3,1), respectively. The above 5–D input data array was interpreted as 1 sample size of 1 batch sequence of 100000 timesteps, 3 feature variables and passing through 1 channel (Col. 13). For the evaluation arrays, these were obtained as X_ev.reshape(1, 1, 41200, 3, 1).

Lastly, the 2-D input shape and 3-D model input arrays of the LSTM network were obtained [62, 32]: –

$$\text{Input\_Shape\_LSTM} = (\text{timesteps, features}) \quad (8)$$

$$\text{Input\_Data\_LSTM} = \text{X.reshape(samples, timesteps, features)} \quad (9)$$

Note: From Eq. 9, the 3-D input array of the LSTM model was interpreted as 1 sample of 100000 timesteps with 3 feature variables (Cols. 8 and 11). Also, the Conv1D model required a 3-D input data array (1,100000,3) with the corresponding 2-D model input_shape argument (100000,3).

Lastly, the model output vector of Conv2D, ConvLSTM2D and LSTM networks was obtained as: –

$$\text{Output\_Data} = \text{Y.reshape (samples, timesteps)} \quad (10)$$

where Y = expected output variable data. The model output data (arrays) of the training/validation/ evaluation were: calculated as- y_train_mod = Y_tr.reshape(1,100000), y_val_mod = Y_va.reshape(1,100000) and y_ev_mod = Y_ev.reshape (1, 41200), respectively (Col. 14 of Table 1).

Since batched training method is considered when reduced model computation complexity is required, we adopted the batch training method as a good option. For recalling the weighted sum of stored training model with the system architecture (Tables 2–4). this method is considered for: [1]5 sets of batch training dataset, [2]model evaluations and predictions of present and future input data sequence. The model input arrays (x_train_mod, x_val_mod, x_test_mod = 1,1,100000,3,1) were divided into 5 batches of model input datasets ([1:20000 = (1,1,20000,3,1)], [20001:40000 = (1,1,20000,3,1)], [40001:60000 = (1,1,20000,3,1)], [60001:80000 = (1,1,20000,3,1)] and [80001:100000 = (1,1,20000,3,1)]). The 1st batch data array (1:20000) was fed as 1st input sequence into the network input layer as presented in model system architectures (Tables 2–5) and thereafter, the procedures were repeated for subsequent batches.

Meanwhile, the transfer functions for initialization of input, hidden, dense and output layers of the neural networks were defined. For the layers (input, hidden and dense) of Conv2D and ConvLSTM2D models, the weighted sum of the input data arrays used the hyperbolic tangent transfer function with a zero-centered model output as determined [63]: -

$$\tanh = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})} \quad (11)$$

where e is the base of a natural logarithm; x is the vector of the outputs, taking scaled input and output values in the range of –1 to 1.

For the LSTM model layers (above), the weighted sum of the model input array was determined using rectified linear unit (relu) function while the sigmoid function of all the model output layer was also defined as [64]: -

$$\text{relu} = \max(0.0,\ x) \tag{12}$$

$$\text{sigmoid} = \frac{1}{(1+e^{-x})} \tag{13}$$

where sigmoid function takes any scale inputs and output value in the range of 0 to 1.

3.2 Conv2D Model Setup

The Conv1D model is a convolutional network that works best with very-short input data arrays for univariate time series forecasts. One main limitation of this model is its poor performance when utilized with long-term non-linear input sequence. To overcome this limitation, a robust model (2–dimensional CNN, Conv2D) that was suitable for the long-term multivariate inputs to multistep outputs was developed. Unlike the Conv1D model that utilized 1-D MaxPooling layer with 1-D kernel size (Table 4), the Conv2D sequential model in its architecture [42,18] utilized 2-D MaxPooling layer with 2-D kernel size (Table 2). Thus, the Conv2D model (Fig 1a) was built on the traditional Conv1D model architecture with replacement of '1-D MaxPooling layer and 1-D kernel size' with '2-D MaxPooling layer and 2-D kernel size'.

The developed Conv2D neural model architecture entails building a 2-D CNN model with normalized input/output data arrays. For this network setup, the hyperparameters that influence the model learning ability and performance from the input to output layers were identified, and initialized before the network layers setup was compiled (Fig 1a). **Firstly**, From Eq. (4), the Conv2D model normalized input arrays (0.0 – 1) were considered for: –[1]smoother gradient descent, [2]faster convergence [65, 50] of the training and validation losses (Figs a–b of 2 and 5), and [3]reduced number of epochs for learning the input sequence (Fig a–b of S4–5). **Next**, 7–layer Conv2D block diagram (Fig. 1a) and sequential model system architecture (Table 2) from Keras Library [56] were arranged/built with the following parameters: - '1 input layer and 1st 2-D MaxPooling layer', '1 convolutional hidden layer and 2nd 2-D MaxPooling layer', 1 flatten layer, 1st dense layer and 1 output (2nd connected dense) layer, respectively. The MaxPooling layer distill the output of previous layer by reducing the spatial dimensionality of the input sequence volume through down-sampling approach [66–68] and passed it to subsequent layer, while the flatten layer takes the processed input sequence from the previous layer and narrow the featured sequence by wrapping it as 1-D vector (from 4-D [1,1,20000,64] to 2-D output shape [1, 1280000]). The 1st connected dense layer extracts the wrapped input sequence from a flatten layer and interprets them for model predictions before passing through to the output nodes of the layer. **Lastly**, the following model hyperparameters were selected: 128, 64 and 72 convolutional filters (layer neurons) for the input, hidden and dense layers, respectively; the kernel size of 2-D convolution window (width=1, height=1) for input and hidden layers, respectively; strides value = 1 and the padding set to 'same' (for zero padded inputs with the input and output data arrays having similar spatial dimension), respectively; dropout regularization rates between 20 and 40 % (ensuring unused or randomly selected neurons in complex learning are ignored); tanh function to activate the input, hidden and dense layers but the sigmoid function for the output layer activation. **Other parameters** for Conv2D model build-up and learning were**:** epochs = 100 (iterations), batch_size =1, nb_classes = timesteps = 20000. **Thereafter**, step 6 (Appendix A.1) was used to compile and summarize the system architecture (output shape of each convolutional layer). For the network compilation (Fig 1a and step 6) that performs the regression analysis, the neural network was designed with the: [1]model optimizer of Adamax version of stochastic gradient descent (to achieve better model performance); [2]loss function (mean_square_error, MSE) and [3]model metrics (mean_absolute_error, MAE) for assessing the training and validation performances. Meanwhile the fit function (step 8) was used for training the compiled Conv2D network layers with model input data arrays. The procedures utilized for developing a Conv2D neural network (Fig 1a, Cols. 9 and 12 of Table 1, and Table 2) [32] were wrapped in 12 steps (Appendix A.1).

3.3 ConvLSTM2D Model Setup

Firstly, the 5-D ConvLSTM2D input arrays were obtained from Eq. (7). **Next**, 7–layer network of the ConvLSTM2D model block diagram (Fig. 1b) and system architecture (Table 3) were built with: - 1 input layer with 1st batch normalization layer (instead of 1st 2–D MaxPooling layer and dropout), 1 convolutional hidden layer with 2nd batch normalization layer (instead of 2nd 2–D MaxPooling layer and dropout rate), 1 flatten, 1 dense and 1 output layers, respectively. The batch normalization layer stabilized (regularized) the layer input arrays for faster deep learning process and also reduced the model learning time. Due to the complexity of this model, the batch normalization layer

was considered and takes an output data sequence from previous connected layer (input or hidden layer) and normalized it (resolved the internal covariate shift in a distributed sequence from previous layer) [48, 68–69] before passing the normalized sequence to subsequent layers network. The flatten layer accepts the processed input sequence from 2nd batch normalization layer and reduced/wrapped this feature sequence into a single 1-D vector (from 5-D [1,1,20000,3,60] to 2-D output shape [1,3600000]). The dense layer directly connected to the model flatten layer extracts the wrapped input sequence and interprets before passing it through to an output layer. **Lastly**, the following model hyperparameters were considered: 30, 60 and 25 filters (layer neurons) for input, hidden and connected dense layers, respectively; kernel sizes of (7,7) and (6,6) for the input and hidden layers, respectively; return sequence ='True' and padding = 'same', respectively; tanh activation function for the input, hidden and dense layers but sigmoid function for the model output layer. For the network compiler (Fig 1a), the Adamax was used as the network optimizer; the mean_square_error as loss function and mean_absolute_error as the model metrics for checking the training performance. **Other considered parameters** for the model build-up and learning process were**: -** epochs = 100, batch_size = 1, nb_classes = timesteps = 20000. The procedures for developing the ConvLSTM2D neural model (Fig 1b, Cols. 10 and 13 of Table 1, and Table 3) were wrapped into 12 steps (Appendix A.2).

3.4 LSTM Model Setup

The long short-term memory (LSTM) is a special form/extension of recurrent neural model 'RNN' that has a looped network layer(s) for deep learning and allows a given input sequence information to persist for longer time [26, 50]. To calculate the model predicted values for short-long term, the simple structure and operation of an LSTM cell has been presented in Fig. 4 of the literature [72], comprising of the following 4 stages [73]: -

$$Input\ gate\ (I_t) = \sigma_g(R_i h_{t-1} + W_i X_t + b_i) \quad (14a)$$
$$Memory\ Cell\ (C_t) = \sigma_c(R_c h_{t-1} + W_c X_t + b_c) \quad (14b)$$
$$Forget\ gate\ (f_t) = \sigma_g(R_f h_{t-1} + W_f X_t + b_f) \quad (14c)$$
$$Output\ gate\ (o_t) = \sigma_g(R_o h_{t-1} + W_o X_t + b_o) \quad (14d)$$

where W_i, W_c, W_f and W_o are the input neurons' shared weight matrices; h_t denotes the hidden state at a given time instance (t), X_t is the input of the network; σ is the gate activation function; h_{t-1} is the output at previous time (t-1); R_i, R_c, R_f and R_o are the recurrent weight matrices; b_i, b_c, b_f and b_o are the bias vectors.

Firstly, the 3-D LSTM model input arrays were obtained from Eq. (9). **Next**, 5–layer LSTM model block diagram (Fig. 1c) and system architecture (Table 5) were built with: 1 input layer with dropout rate of 20 %, 1 convolutional hidden layer with added dropout rate of 35 %, 1 flatten layer, 1 dense layer and 1 output layer. The flatten layer takes the input sequence from the hidden layer, and reduced this feature sequence into 1–D vector (from 3–D [1, 20000, 64] to 2–D output shape [1, 1280000]). Also, this network model utilized the relu activation function for all layers (input/hidden/dense) and the sigmoid function for the model output layer activation. **Lastly**, the following network hyperparameters were considered: 128-, 64- and 72-layer neurons (input, hidden and dense, respectively); return sequence ='True' for input and hidden layers, respectively. For the network optimizer (Fig1c), the Adamax version of stochastic gradient descent was also utilized; the mean_square_error as the loss function and mean_absolute_error as the metrics. **Other considered parameters** for the neural network build-up and learning were**:** epochs = 100, batch_size = 1, nb_classes = timesteps = 20000. The procedures for developing the LSTM neural model (Fig 1c, Cols. 8 and 11 of Table 1, and Table 5) were wrapped into 12 steps (Appendix A.3).

3.5 Model Performance Evaluations

The model performances (training, validation, evaluation and prediction) for each neural network were assessed using the model loss function and metrics as follows: –

$$\text{MSE (loss function)} = \sum_{i=1}^{N} \frac{(Y_{pred_i} - Y_{act_i})^2}{(N)} \quad (15)$$

where Y_{pred} denotes the i^{th} predicted wind speed and direction of the Conv2D, ConvLSTM2D and LSTM models; Y_{act} is the i^{th} actual model wind speed and direction; N is the input data-point per variable (Col. 7 of Table 1); MSE, RMSE and MAE are the mean_square_error, root_ mean_square_error and mean_absolute_error, respectively.

$$\text{MAE (metrics)} = \sum_{i=1}^{N} \frac{\left|(Y_{pred_i} - Y_{act_i})\right|}{(N)} \quad (16)$$

$$\text{RMSE} = \sqrt{\sum_{i=1}^{N} \frac{(Y_{pred_i} - Y_{act_i})^2}{(N)}} \quad (17)$$

The performance of each neural model in predictions of wind speed was assessed as:

$$\text{ME (m/s)} = \sum_{i=1}^{N} \frac{(Y_{pred_i} - Y_{act_i})}{(N)} \quad (18)$$

where estimated ME is the model mean error of wind speed (m/s)

4.0 Results and Discussion

4.1 Results

The studied finding results of the developed Conv2D, ConvLSTM2D and LSTM network models are depicted (Figs S1–10, supplementary file; Figs 2–10 and Table 6). Meanwhile, the training, validation, testing and prediction errors (1st and 2nd batched datasets) with evaluated stations result have been summarized (Table 6), in which the best performance is highlighted in bold.

4.2 Discussion

4.2.1 Model hyperparameters tuning

In model learning process of the historical wind speed and direction pattern, an appropriate allocation of different filter-size (layer-neurons) with the kernel size, network optimizer, as well as the right selection of the activation function of each network layer directly influence how the forecast model layers processed and learnt the input sequences at different station heights (Figs S1a and 2a, Figs S2a and 5a, and Figs S9a and 8a). The model performance comparisons of ConvLSTM2D and Conv2D with independent input data for model tuned hyperparameters (selected filters and kernel sizes) were presented (Figs a and d of S1–2). For the Conv2D setup, the kernel size (1,1) with small filters (12 (input) and 6 (hidden)) and dense layer filters =72 were considered (Fig S1a); before, a new network model with the same kernel size but with large filters (28 (input), 14 (hidden layer) and 72 (dense layer filters)) was built for comparisons (Fig. S1d). From the network performances (Figs. a and d of S1), it was clearly seen that the developed Conv2D model with 28 and 14 filter-sizes fairly learnt at 9 epochs with less model oscillation. Also, a fair wind speed forecast (Fig. b and e) with minima model errors (Fig c and f) were achieved as compared to developed model with '12 (input) and 6 (hidden) filters'. For further model tuning-process and better results with the Conv2D neural network, the forecast model with the kernel size of (1,1) but larger filter sizes of 128 and 64 (input and hidden layers) was adopted, and initialized for reliable time series wind predictions (Fig 1a and Table 2).

Meanwhile, for the ConvLSTM2D model with kernel size = (4,4) and selected filter-size (48 (input), 24 (hidden) and 25 (dense layer)), the network model learnt slowly with an input sequence (Fig S2a). Increasing the hyperparameters to kernels of (8,8) with larger filter sizes/neurons (64, 32 and 25, respectively), the forecast model didn't learn (Fig S2d) as the network training and validation quickly diverged at 9 epochs with over-predicted wind speeds (Fig S2e), and high forecasts mean errors (Fig. S2f). To achieve better performance and convergence with the ConvLSTM2D network, the new forecast model was built with 'filter-size = 30, kernel-size = (7,7) at the input layer', 'filter-size = 60, kernel-size = (6,6) at the hidden layer' and dense-layer filters =25 (Fig 1b and Table 3). Lastly, the LSTM network performances (training and validation) were also assessed (Fig S9a) with layer-neuron of '128 (input), 64 (hidden) and dense layer = 45'; and compared with newly built model of similar input and hidden neurons but different dense layer neurons=72 (Fig S9b). The LSTM network built with the dense neurons =72 showed significant improvement over the old forecast model with dense layer neurons = 45. For the reliable LSTM forecasts, the LSTM model with dense layer neurons =72 was retained for better model learning and predictions of time-series wind speed and directions (Fig 1c and Table 5).

From the comparisons of Conv2D and ConvLSTM2D architectures (Tables 2 with 3), the allocated filters size to the Conv2D input and hidden layers differ with the ConvLSTM2D allocated filter sizes. Based on the designed network architectures of Conv2D (Fig 1a and Table 2) and LSTM models (Fig 1c and Table 5), both model input layers were allocated higher filter-size/neurons =128 (step 2 of Appendix *A*.1 and *A*.3) for handling the input data sequence but was later decreased (layer filters =64 in step 3) as the sequence deepened into the hidden layer. Also, the 1st dense layer connected to the flatten layer had more filters (filters =72 in step 4b) than the hidden layer with filter-sizes = 64 (step 3). As the model learning process increases within the network layers, the filter sizes allocated to the dense layer also increased but with output spatial volume reduction at the flatten layer ((from 5-D to 2-D) in ConvLSTM2D model, (4-D to 2-D) in the Conv2D and (3-D to 2-D) in LSTM/Conv1D model). Meanwhile, for the ConvLSTM2D model of a hybrid network (Fig 1b and Table 3), the convolutional input layer was allocated few filter maps (filter-size = 30 in step 2 of *A.2*) but was increased to filter-size = 60 (step 3) at the hidden layer. As ConvLSTM2D model deepened for input sequence learning, the hidden and dense layers of the model usually required more filters than at an input layer. However, the 1st dense layer connected to the flatten layer had less filter-size =25 (instead of 72-filters as shown in step 4b) than the hidden layer filters = 60 (step 3 of *A.2*). This was due to computation complexity of ConvLSTM2D model if assigned a higher dense-layer filter. The reduction in the dense layer filters of ConvLSTM2D

only led to the fast-learning process and efficient computation with moderate memory needs.

4.2.2 Models result comparison and accuracy of forecast (training)

The model performances (validation and training losses) of the Conv2D (Figs 2a–b), ConvLSTM2D (Figs 5a–b) and LSTM (Figs 8a–b) networks have been assessed and compared (Table 6 in terms of the mean_square_error, MSE, and mean absolute_error, MAE). Also, the Conv1D neural model performances were assessed (Fig S5 a–b). For 1st batch input sequence, the Conv2D model training and validation losses converged at 6 epochs but with model validation loss oscillations in twisty pattern at 11–100 epochs (Fig 2a); for 2nd batch training, the model learnt better with lower gradient descent but with validation loss oscillations at 26–100 epochs (Fig 2b). For the ConvLSTM2D model performance, the training and validation losses converged at 11 epochs but with validation loss oscillations at 79–100 epochs (Fig 5a); while the model learnt better with loss convergence at 16 epochs for 2nd batch training data but with validation loss oscillation at 91–100 epochs (Fig 5b). Furthermore, the LSTM model training and validation losses were compared (Figs 8a–b): for the 1st batch training data, this network converged at 5 epochs with validation loss oscillations at 32–100 epochs (Fig 8a); for 2nd batch training, the model had higher gradient descent with losses convergence at 5 epochs but with the model validation loss oscillates at 24–100 epochs (Fig S10a). From an in-depth into the model performances (validation and training losses), Fig 5a–b clearly shows that the ConvLSTM2D would be a better choice for model training as compared to the Conv1D (Figs S5a–b) and LSTM (Figs 8a–b, S10a) networks. Also, the Conv2D model shows significant improvement over the LSTM and ConvLSTM2D model performances. The cause of the model validation loss oscillations could not be ascertained but may be attributed to historical input arrays (batched datasets) fed into the network input layer. Thus, passing either a recent input data sequence or more historical input arrays into network training/validation process may have greater impact on the model performances and oscillations stability [26].

Meanwhile, the Conv2D model predictions of time series wind speeds for 4 horizons (0–50, 50–100, 100–150, and 150–200 timesteps) were presented (Figs 3–4) and compared with the other model prediction results (ConvLSTM2D (Figs 6–7), LSTM2D (Figs 8d and S10d) and Conv1D (Fig S5d)). From these results, it was shown that all trained networks replicated the historical patterns of the wind speeds (Fig S6 a–f), however, LSTM model over-estimated the historical wind speeds but outperformed the Conv1D predictions. Also, the Conv2D model prediction results show significant improvements over the ConvLSTM2D results and other models; although both forecast models (Fig S8 a-b) accurately reproduced the actual wind direction patterns for the considered timesteps/ horizons. For the LSTM model predictions with dense layer-neurons = 45 (Fig S9c), the actual wind speeds were highly overestimated as compared to the model prediction with dense layer neurons = 72 (Fig 8d). The comparisons of LTSM models built with different dense layer neurons (Fig 8d vs S9c) revealed that the selected dense-layer neurons were highly sensitive to the model learning process and had greater impacts on the model predictions accuracy. With independent weather input data arrays, the model forecast ability in replicating the historical wind directions was assessed. The time series of wind direction forecasts from the LSTM trained models were compared (Fig. 8e). From the forecast results (Figs. 8e, S8a and S8b), the LSTM, Conv2D and ConvLSTM2D wind direction predictions were very close to the actual model results in most of the considered time instances. Also, the plots of Conv2D, LSTM and Actual models wind roses as well as their frequency occurrences in 12 wind sectors have been presented (Fig 10 a–f); while for the ConvLSTM2D model, this was depicted (Fig S7 a–b). From the model results, the Conv2D and LSTM network models replicated the sectorwise actual directions with their corresponding frequency occurrences (%) as compared to ConvLSTM2D model predictions in 12 wind sectors (0, 30, 60, ... 359^{o}).

The Conv2D model prediction errors (Fig 2 c–d) were compared with ConvLSTM2D (Fig 5 c–d) and LSTM errors for 200 timesteps (Fig c of 8 and S10). From the prediction errors comparison, Fig 2 c–d (Conv2D input layer with neurons/filters =128) shows significant improvement over Fig S1 c and f (Conv2D input layer filters of 12 and 28); Fig 5c–d (ConvLSTM2D filters=30 and kernel size=7) shows significant improvement over 'Fig S2c (ConvLSTM2D layer filters = 48 and kernel=4) and Fig S2f (ConvLSTM2D layer filters = 64 and kernel = 8)'; Fig. 8c (LSTM dense layer neurons = 72 (1st batch)) shows significant improvement over 'Fig S10 c (LSTM dense layer neurons = 72 (2nd batch)) and Fig S9d (LSTM dense layer neurons = 45 (1st batch))'. Overall, Conv2D forecast errors of wind speeds (Fig 2 c–d) were minima, followed by ConvLSTM2D (Fig 5 c–d) and LSTM prediction errors (Fig c of 8 and S10). Lastly, the accuracy of the Conv2D, ConvLSTM2D and LSTM models for both untrained (compiled layer models without training procedures) and trained networks (layer models with training procedures) have been assessed with evaluated input data arrays of 5 stations (SM–ST1 to SM–ST5). In terms of the MSEs and MAEs, the Conv2D model produced minima training and validation losses as compared to other model results (Table 6). Also, Conv2D model evaluated results were better with stations 1, 3 and 4 datasets as compared to other model evaluated results. LSTM model outperformed the Conv2D and ConvLSTM2D models at station 5; while ConvLSTM2D and LSTM models

outperformed the Conv2D at station 2. For the untrained models, the ConvLSTM2D showed slight improvement (at stations 1–2) as compared to Conv2D and LSTM model evaluations while the evaluated results for stations 3–5 were similar for all untrained network models (Fig. 9c). Using different datasets, the evaluated results (MSE/MAE/RMSE) of 4 developed prediction models by Qiu et al [62] were compared to our studied findings (Table 6). Authors in their studied findings reported 7.85,23.60 and 28.02 %, respectively for their LSTM model with DJIA data while estimated values of 18.39, 30.31 and 42.88 % were reported for the same model with HSI dataset. Comparing our evaluated results (Cols. 5, 8 and 11 of Table 6) of Keras LSTM model with the Authors LSTM model results, the Keras neural network results outperformed the Authors developed model. Also, the RMSE values of all evaluated Keras neural models (Table 6) outperformed Lee et al [70] reported RMSE values of 9.87–25.20 % and Gao et al [71] reported values of 4.62–17.3 %. For our developed hybrid model (ConvLSTM2D), the forecast RMSE results for 1st (5.92%) and 2nd (24.36%) batched dataset outperformed Wang et al (LSTM-CNN model) with estimated RMSE value = 62.1 % [38]. Lastly, our predicted/validated/evaluated model results in Table 6 outperformed Xie et al [72] summarized evaluation metric values (Table 5). Thus, prediction model errors from an optimally tuned model hyperparameters and high-quality datasets show that the forecast accuracy of Keras neural network architectures outperformed the traditional neural networks in existing literature.

5.0 Conclusion

From the studied results, it has been shown that the Keras neural network architectures in the wind speed and direction regression analysis required a well-structured input data, right selection and optimal tuned model hyperparameters before it could be utilized for reliable wind predictions. The developed network models with the weather input arrays for accurate wind speed and direction predictions were achieved. From finding results, the following conclusions are drawn: –

- The allocation of the filters (layer neurons) and kernel sizes should be based on input arrays dimension and the size of the interconnected layers within the network, as they had great impacts on the model learning ability (convergence or divergence) and directly influence the sequential network predictions accuracy.
- Independent datasets (high-quality) of different station heights should be normalized before used as the model input data array (sequence). The trained model with the scaled input arrays (0–1) was a better choice for reliable forecast time horizons.
- From the evaluated results, a better prediction result was obtained with the trained network models as compared to compiled layer models without training procedures.
- From the predicted results, the Keras sequential models were able to replicate the time-series of historical wind speed and direction based on the model hyperparameters tuning as well as the input sequence structure.
- From the trained and validated results, the Adamax optimizer of Conv2D model recorded the best performance with timely convergence for smaller training batch_size=1 as compared to other forecast models' configuration.
- From the model training and validation losses, the Conv2D neural network outperformed all other forecast models (ConvLSTM2D, LSTM and Conv1D) for wind speed and direction analysis. Hence, the Conv2D model would be a better choice for a stand-alone forecast model in the wind assessment based on the model learning ability with historical wind conditions of a given site.
- Lastly, the wind roses of Conv2D and LSTM with Actual models as well as their frequency occurrences show that the proposed neural networks within deep-learning framework was a reliable forecast tool in replicating the historical wind speed and direction patterns at the considered station heights.

ConvLSTM2D model performances (trainings/validations/predictions) were encouraging and calls for improvement through a hyperparameter tuning process based on allocation of higher dense-layer filters and lower kernel sizes. Also, since batched training method was adopted to reduce the computation complexity, the model framework could be refined with unbatched and seasonal datasets in further studies. It is worth noting that LSTM model is a good choice when model ability to memorize the long-learnt patterns through memory cells/gates is required but it's unsure if training with longer noisy (non-linear) input sequence would improve ConvLSTM2D and LSTM models performance. Future studies would consider bi-directional LSTM; incorporation of more hidden layers into the LSTM and ConvLSTM2D models, and the effects of batch normalization layer on the LSTM performances. Furthermore, the ConvLSTM2D model framework with other developed sequential models in classification task would be fully explored. Thus, higher machine learning platform to host the ConvLSTM2D model would be required in the future.

Acknowledgment

The model input arrays for setup/tuning of the developed sequential model were both sourced from the [1]South African Weather Service and [2]South African Wind Atlas Stations, and continuously appreciated. Station data descriptions are provided in Section 2 and may be assessed upon request to South African Weather Service representative(s). The author appreciates the financial supports from ZakkWea Energy {ZWE} to achieved the studied objectives. No financial grant was received for the study work.

Reference: –

[1] C. Feng, M. Cui, M. Lee, Zhang J, Hodge BM, Lu S, Hamann HF (2017) Short-term global horizontal irradiance forecasting based on sky imaging and pattern recognition. In: 2017 IEEE Power & Energy Society General Meeting, IEEE, pp 1–5. https:// doi org/10.1109/ PESGM.2017.82744 80

[2] M. Fliess, Join C, Voyant C (2018) Prediction bands for solar energy: New short-term time series forecasting techniques. Solar Energy 166:519–528. https://doi.org/10.1016/j.solener.2018.03.049

[3] A. Fouilloy, C. Voyant, Notton G, Motte F, Paoli C, Nivet ML, Guillot E, Duchaud JL (2018) Solar irradiation prediction with machine learning: forecasting models selection method depending on weather variability. Energy 165:620–629, https://doi.org/10.1016/j.energy.2018.09.116

[4] A. Gensler A, Henze J, Sick B, Raabe N (2016) Deep learning for solar power forecasting an approach using autoencoder and lstm neural networks. In: 2016 IEEE international conference on systems, man, and cybernetics (SMC), IEEE, pp 2858–2865, https://doi.org/10.1109/SMC.2016.7844673

[5] X. Ma and E.H. Hovy (2016) End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF, CoRR, abs/1603.01354.

[6] R. Pascanu, T. Mikolov, and Y. Bengio. 2013. On the Difficulty of Training Recurrent Neural Networks. In Proceedings of the 30th International Conference on Machine Learning, 28, ICML'13, 1310-1318. JMLR.org.

[7] M. D. Zeiler (2012). ADADELTA: an adaptive learning rate method. CoRR, abs/1212.5701.

[8] G. Hinton (2012). Neural Networks for Machine Learning - Lecture 6a - Overview of mini-batch gradient descent.

[9] D.P. Kingma and J. Ba. 2014. Adam: A Method for Stochastic Optimization. CoRR, abs/1412.6980.

[10] T. Dozat (2015). Incorporating Nesterov Momentum into Adam

[11] N. Reimers and I Gurevych (2017), Reporting Score Distributions Makes a Difference: Performance Study of LSTM-networks for Sequence Tagging. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP), Copenhagen, Denmark, September.

[12] A. Komninos and S. Manandhar (2016). Dependency based embeddings for sentence classification tasks. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 1490–1500, San Diego, California, June. Association for Computational Linguistics.

[13] Abdel-Nasser, M.; Mahmoud, K. (2019) Accurate photovoltaic power forecasting models using deep LSTM-RNN. Neural Comput. Appl., 31, 2727–2740.

[14] Y. Jung, J. Jung, B. Kim, S. Han; (2020) Long-short term memory recurrent neural network for modelling temporal patterns in long-term power forecasting for solar PV facilities: Case study of South Korea. J. Clean. Prod., 250, 119476.

[15] T.D.K Thara, Prema, P.S.; Xiong, F. (2019) Auto-detection of epileptic seizure events using deep neural network with different feature scaling techniques. Pattern Recognit. Letter, 128, 544–550.

[16] F Hutter, Holger Hoos, and Kevin Leyton-Brown (2014). An Efficient Approach for Assessing Hyperparameter Importance. In Proceedings of the 31st International Conference on International Conference on Machine Learning, 32, ICML'14, 754–762. JMLR.org.

[17] G. Keren and B. Schuller, (2016) "Convolutional RNN: An enhanced model for extracting features from sequential data," 2016 International Joint Conference on Neural Networks (IJCNN), 3412-3419, doi:10.1109/IJCNN.2016.7727636

[18] N.T. Vu (2016), Sequential Convolutional Neural Networks for Slot Filling in Spoken Language Understanding, DOI:10.2143/Interspeech.2016-395

[19] J. Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems, 25, 2951–2959. Curran Associates, Inc.

[20] Y. Zheng, Q. Liu, E. Chen, Y. Ge and J. Zhao, (2016), Exploiting multi–channels deep convolutional neural networks for multivariate time series classification. Front. Comput. Sci. 10, 96–112.

[21] Y. LeCun, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444. https:// doi. org/ 10. 1038/ nature14539

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

[23] P. Abinaya, V.S. Kumar, P. Balasubramanian and V.K Menon (2016), "Measuring stock price and trading volume causality among Nifty50 stocks: The Toda Yamamoto method." In Advances in Computing, Communications and Informatics (ICACCI) pp.1886–1890

[24] M. Hiransha, E.A. Gopalakrishnan, Vijay Krishna Menon, K.P. Soman, (2018), NSE Stock Market Prediction Using Deep–Learning Models, Procedia Computer Science, 132, 1351–1362, https://doi.org/10.1016/j.procs.2018.05.050.

[25] J. Kumar, A.K. Singh, (2016), Dynamic resource scaling in cloud using neural network and blackhole algorithm. In:2016 Fifth International Conference on Eco–friendly Computing and Communication Systems (ICECCS), 63–67.

[26] J. Kumar, R. Goomer and A.K. Singh; (2018) Long Short–Term Memory Recurrent Neural Network (LSTM–RNN) Based Workload Forecasting Model for Cloud Datacentres, 6th International Conference on Smart Computing and Communications, Procedia Computer Science 125, 676–682, DOI: 10.1016/j.procs.2017.12.087
[27] C.H Shih, Yan B C, Liu S H, et al. Investigating Siamese LSTM networks for text categorization[C]// Asia–pacific Signal & Information Processing Association Summit & Conference. 2017.
[28] F. Simistira, A. Ul–Hasan, V. Papavassiliou, and et al. (2015) Recognition of Historical Greek Polytonic Scripts Using LSTM Networks[C]// 13th International Conference on Document Analysis and Recognition.
[29] B. Jang, M. Kim, G. Harerimana, S. Kang, (2020), Bi-LSTM Model to Increase Accuracy in Text Classification: Combining Word2vec CNN and Attention Mechanism, Applied Sciences 10(17):5841
[30] Xu T, Zhang J, Ma Z, et al. (2017) Deep LSTM for Large Vocabulary Continuous Speech Recognition[J].
[31] J. Kim, El–Khamy M, Lee J. Residual LSTM: Design of a Deep Recurrent Architecture for Distant Speech Recognition[J]. 2017.
[32] S. Malakar, S. Goswami, B. Ganguli, et al. (2021), Designing a long short-term network for short-term forecasting of global horizontal irradiance. *SN Appl. Sci.* **3,** 477, https://doi.org/10.1007/s42452-021-04421-x
[33] C. Voyant, G. Notton, S. Kalogirou, Nivet ML, Paoli C, Motte F, Fouilloy A (2017) Machine learning methods for solar radiation forecasting: a review. Renew Energy 105:569–582. https:// doi.org/ 10. 1016/j. renene.2016.12.095
[34] P. Kumari, D. Toshniwal, (2021), long short–term memory–convolutional neural network based deep hybrid approach for solar irradiance forecasting, Applied Energy, 295, 117061, https://doi.org/10.1016/j.apenergy.2021.117061
[35] S. Sobri, S. Koohi–Kamali; Rahim, N.A. (2018), Solar photovoltaic generation forecasting methods: A review. Energy Convers. Manag., 156, 459–497.
[36] M. Konstantinou, S. Peratikou, Charalambides, A.G. (2021), Solar Photovoltaic Forecasting of Power Output Using LSTM Networks. Atmosphere, 12, 124. https://doi.org/10.3390/ atmos12010124
[37] N. Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and R Salakhutdinov. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. J. Mach. Learn. Res., 15(1):1929–1958, January.
[38] K. Wang, X Qi, H. Liu (2019) Photovoltaic power forecasting-based LSTM-Convolutional Network, Energy,189,116225.
[39] Gundu, V., Simon, S.P. (2021), Short Term Solar Power and Temperature Forecast Using Recurrent Neural Networks. Neural Process Letter, https://doi.org/10.1007/s11063-021-10606-7
[40] D. Yun, A. Abbas, J. Jeon, M. Ligaray, S. Baek, K.H. Cho, (2021), Developing a deep learning model for the simulation of micro-pollutants in a watershed, Journal of Cleaner Production, 300, 126858, https://doi.org/10.1016/j.jclepro.2021.126858.
[41] M. Baccouche, Mamalet F., Wolf C., Garcia C., Baskurt A. (2011) Sequential Deep Learning for Human Action Recognition. In: Salah A.A., Lepri B. (eds) Human Behavior Understanding. HBU 2011. Lecture Notes in Computer Science, vol 7065. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-25446-8_4
[42] H. Yang, C. Yuan, J. Xing, W. Hu (2017), "SCNN: Sequential convolutional neural network for human action recognition in videos", 2017 IEEE International Conference on Image Processing (ICIP), 355-359, doi: 10.1109/ICIP.2017.8296302.
[43] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, (2015), "Show and tell: A neural image caption generator," in Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, 3156–3164
[44] D Ciresan, U Meier, and J. Schmidhuber, (2012) "Multi-column deep neural networks for image classification," in Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Providence, RI, 2012, 3642–3649.
[45] A. Alzahrani, P. Shamsi, Dagli C, Ferdowsi M (2017) Solar irradiance forecasting using deep neural networks. Proc Comput Sci 114:304–313. https:// doi. org/ 10. 1016/j. procs. 2017. 09. 045
[46] Jian-Wei LIU , Hui-Dan ZHAO , 41ong-Lin LUO , Jun XU (2020) , Research Progress on Batch Normalization of Deep Learning and Its Related Algorithms, Acta Automatica Sinica, 46(6),_1090-1120, https://doi.org/10.16383/j.aas.c180564
[47] C.R.R Molina and O. P. Vila (2017), Solving internal covariate shift in deep learning with linked neurons, 7 Dec 2017, ar41v:1712.02609v1.
[48] S. Saxena; (2021) Introduction to Batch Normalization; https://www.analyticsvidhya.com/blog/2021/03/introduction–to–batch–normalization/ Accessed Date [31 march 2021]
[49] R. Chandra, S. Goyal, R. Gupta, (2016), Evaluation of deep learning models for multi–step ahead time series prediction, IEEE Access, 9, 1–20, https://doi.org/10.1109/ACCESS.2021.3085085
[50] A. Sagheer and M. Kotb. (2019), Unsupervised pre–training of a Deep LSTM–based Stacked Autoencoder for Multivariate time Series forecasting problems, Scientific Report, 9:19038, https://doi.org/10.1038/s41598–019–55320–6
[51] Li Q, Hao Qf, 41ao Lm, Li Zj, (2011), An Integrated Approach to Automatic Management of Virtualized Resources in Cloud Environments. Comput J.;54(6), 905–919.
[52] T. Vercauteren, Aggarwal P, Wang X, h Li T (2006), Hierarchical Forecasting of Web Server Workload Using Sequential Monte Carlo Training. In: 2006 40th Annual Conference on Information Sciences and Systems; 899–904.
[53] N. Roy, Dubey A, Gokhale A. (2011), Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting. In: Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing. CLOUD '11. Washington, DC, USA: IEEE Computer Society; 500–507.
[54] D. Ardagna, Casolari S, Colajanni M, Panicucci B (2012), Dual time-scale distributed capacity allocation and load redirect algorithms for cloud systems. Journal of Parallel and Distributed Computing;72(6):796 – 808.

[55] Y.S. Sun, Chen YF, Chen MC. (2013) A Workload Analysis of Live Event Broadcast Service in Cloud. Procedia Computer Science, 19:1028 – 1033. The 4th International Conference on Ambient Systems, Networks and Technologies (ANT 2013), the 3rd International Conference on Sustainable Energy Information Technology (SEIT–2013).
[56] A. Gulli, Pal S (2017), Deep learning with Keras. Packt Publishing Ltd, Birmingham
[57] I. Sutskever, O. Vinyals, and Q. V. Le, (2014) "Sequence to sequence learning with neural networks," in Proc. of Advances in neural information processing systems (NIPS), Montreal, Canada, 3104–3112.
[58] W. Kong, ZY Dong, F. Luo, K. Meng., (2017), Effects of automatic hyperparameter tuning for residential load forecasting via deep learning., 2017 Australasian Universities Power Engineering Conference (AUPEC), 1-6, doi:10.1109/AUPEC.2017.8282478
[59] S. Hochreiter, J. Schmidhuber, (1997), Long Short–Term Memory, Neural Computation, 9(8), 1735–1780, http://dx.doi.org/10.1162/neco.1997.9.8.1735
[60] J. Schmidhuber, (2015), Deep learning in neural networks: An overview, Neural Networks, 61, 85–117.
[61]https://medium.com/@urvashilluniya/why–data–normalization–is–necessary–for–machine–learning–models–681b65a05029/ Accessed Date [4/11/2019]
[62] J Qiu, B. Wang, C. Zhou (2020) Forecasting stock prices with long–short term memory neural network based on attention mechanism. PLoS ONE 15(1): e0227222. doi:10.1371/journal.pone.0227222
[63] Francois Chollet, (2017), Deep Leaning with Python, pp. 72, ISBN–13: 978–1617294433 [1st Edition].
[64] I. Goodfellow, Y. Bengio, and A. Courville; Deep Learning (Adaptive Computation and Machine Learning Series), ISBN–13: 978–0262035613; pp. 800, MIT Press; 18–11–2016.
[65] https://medium.com/@ashok.tankala/build–the–mnist–model–with–your–own–handwritten–digits–using–tensorflow–keras–and–python–f8ec9f871fd3/ Accessed Date [28/10/2019]
[66] https://machinelearningmastery.com/how–to–develop–convolutional–neural–network–models–for–time–series–forecasting/ Accessed Date [4/11/2019]
[67] https://missinglink.ai/guides/keras/keras–conv2d–working–cnn–2d–convolutions–keras/ Accessed Date [26/11/2019]
[68] Z. Xiang, P. Seeling; (2020) Machine learning for object detection, Computing in Communication Networks, Book Chapter 19, pp. 325–338, https://doi.org/10.1016/B978–0–12–820488–7.00034–7
[69] I. H. Witten, C. J. Pal; (2017) Deep learning in 'Data Mining, Practical Machine Learning Tools and Techniques' Book Chapter 10 (4th Edition), 417–466, https://doi.org/10.1016/B978–0–12–804291–5.00010–6
[70] W. Lee, K. Kim, J. Park, J. Kim, Y. Kim. (2018), Forecasting Solar Power Using Long–Short Term Memory and Convolutional Neural Networks, IEEE Access, 6, 73068–73080.
[71] M. Gao, Li, J.; Hong, F.; Long, D. (2019), Day-ahead power forecasting in a large-scale photovoltaic plant based on weather classification using LSTM. Energy, 187, 115838
[72] Xie, A.; Yang, H.; Chen, J.; Sheng, L.; Zhang, Q. A Short-Term Wind Speed Forecasting Model Based on a Multi-Variable Long Short-Term Memory Network. Atmosphere 2021, 12, 651. https://doi.org/10.3390/ atmos1205065
[73] R.L Abduljabbar, H. Dia, P.W. Tsai, S Liyanage, (2021) Short-Term Traffic Forecasting: An LSTM Network for Spatial-Temporal Speed Prediction. Future Transportation, 1, 21–37. https://doi.org/10.3390/ futuretransp1010003

Appendix A

A.1. Conv2D Model Set up (Steps 1-12)

Step 1: Create a Sequential Network Model

Step 2: Add an Input (1st network Layer)

```
model.add(Conv2D(filters=128, kernel_size=(1, 1), strides=1,padding='same', kernel_initializer='glorot_uniform', activation='tanh', input_shape= Input_Shape))

model.add(MaxPooling 2D(pool_size=(1, 1)))

model.add(Dropout(0.20))
```

Step 3: Add 1st Hıdden Layer

```
model.add(Conv2D(filters=64, kernel_size=(1, 1), strides=1, padding='same', activation='tanh'))
model.add(MaxPooling 2D(pool_size=(1, 1)))
model.add(Dropout(0.4))
```

Step 4a: Add a Flatten Layer

Step 4b: Add the 1st Dense Layer

```
model.add(Dense(72,activation='tanh'))
```

Step 5: Add an Output Layer (2nd Dense Layer)

```
model.add(Dense(nb_classes, activation='sigmoid'))
```

Step 6: Compıle and Print the Summary of Network Model

```
model.compile(loss=keras.losses.mse,optimizer=keras.optimizers.Adamax(), metrics=['mean_absolute_error'])
model.summary()
model.output.shape
```

Step 7a: Save the Model System Architecture before Traınıng

Step 7b: Create a Callback proved by Keras and Save the best Traınıng Model Weights

Step 8: Traın the Compiled Network Model

```
history = model.fit(x_train_mod, y_train_mod, batch_size=batch_size, epochs=epochs, validation_data =(x_val_mod, y_val_mod), verbose=1, callbacks=callbacks_list)
```

Step 9: Check the Network Model Performance with Testing Datasets

Step 10: Make Network Predıctıon based on Input Sequence

Step 11: Load/Recall the saved Untrained and Trained Model including the Weight and Optimizer

```
untrained_model = tf.keras.models.load_model('C:/Users/.spyder–py3/untrained_model_Conv2D_MyModel.hdf5')
trained_model = tf.keras.models.load_model('C:/ Users/.spyder–py3/trained_model_Conv2D_MyModel.hdf5'')
```

Step 12: Evaluate the Untrained and Trained Model with Evaluation Datasets

```
untrained_model_eval_te_1 = untrained_model.evaluate (x_te1_mod, y_te1_mod, batch_size=batch_size)
trained_model_eval_te_1 = trained_model.evaluate (x_te1_mod, y_te1_mod, batch_size=batch_size)
untrained_model_eval_te_2 = untrained_model.evaluate (x_te2_mod, y_te2_mod, batch_size=batch_size)
trained_model_eval_te_2 = trained_model.evaluate (x_te2_mod, y_te2_mod, batch_size=batch_size)
untrained_model_eval_te_3 = untrained_model.evaluate (x_te3_mod, y_te3_mod, batch_size=batch_size)
trained_model_eval_te_3 = trained_model.evaluate (x_te3_mod, y_te3_mod, batch_size=batch_size)
untrained_model_eval_te_4 = untrained_model.evaluate (x_te4_mod, y_te4_mod, batch_size=batch_size)
trained_model_eval_te_4 = trained_model.evaluate (x_te4_mod, y_te4_mod, batch_size=batch_size)
untrained_model_eval_te_5 = untrained_model.evaluate (x_te5_mod, y_te5_mod, batch_size=batch_size)
trained_model_eval_te_5 = trained_model.evaluate (x_te5_mod, y_te5_mod, batch_size=batch_size)
```

Step 13: Rescale the network forecast values (output data) from 0-1 back to the normal values (m/s)

A.2. ConvLSTM2D Model Set up (Steps 1-12)

Step 1: Create a Sequential Network Model

Step 2: Add an Input Layer

```
model.add(ConvLSTM2D(filters=30, kernel_size=(7,7), activation='tanh', return_sequences=True, padding='same', input_shape=Input_Shape, name='First_ConvLSTM2D'))
model.add(BatchNormalization())
```

Step 3: Add the 1st Hıdden Layer

```
model.add(ConvLSTM2D(filters=60, kernel_size=(6,6), activation='tanh', return_sequences=True, padding='same', name='Second_ConvLSTM2D'))

model.add(BatchNormalization())
```

Step 4b: Add a Flatten Layer

Step 4b: Add a Dense Layer

```
model.add(Dense(25, activation='tanh'))
```

Step 5: Add an Output Layer

```
model.add(Dense(nb_classes, activation='sigmoid'))
```

Step 6: to Step 10 were repeated

Step 11: Load the saved Untrained and Trained Model including the Weights and Optimizer

Step 12: Evaluate the Untrained and Trained Model with Evaluation Datasets

Step 13: Rescale the network forecast values (output data) from 0-1 back to the normal values (m/s)

A.3. LSTM Model Set up (Steps 1-12)

Step 1: Create a Sequential Network Model

Step 2: Add an Input Layer with 128 neurons

```
model.add(layers.LSTM(128, return_sequences=True, input_shape=(input_shape), activation='relu'))
model.add(layers.Dropout(0.20))
```

Step 3: Add the 1st Hıdden Layer with 64 neurons

```
model.add(layers.LSTM(64, return_sequences=True, activation='relu'))
model.add(layers.Dropout(0.35))
```

Step 4b: Add a Flatten Layer

Step 4b: Add a Dense Layer with 72 neurons

```
model.add(layers.Dense(72, activation='tanh'))
```

Step 5: Add an Output Layer

```
model.add(layers.Dense(nb_classes, activation='sigmoid'))
```

Step 6: to Step 10 were repeated

Step 11: Load the saved Untrained and Trained Model including the Weights and Optimizer

Step 12: Evaluate the Untrained and Trained Model with Evaluation Datasets

Step 13: Rescale the network forecast values (output data) from 0-1 back to the normal values (m/s)

Nomenclature

Temp/Train	Mean dry–bulb temperature (°C) over the last 1 minute at 2 m AGL
Hum/Rhum	Mean relative humidity (%) over the last 1 minute
Tgrad/TS_10/TS_60	Air temperature gradient and bulb measurements (°C) at 10 and 60 m AGL
WS_10	Mean scalar wind speed (m/s) over the last 5 minutes at 10 m AGL
WS_20	Mean scalar wind speed (m/s) over the last 5 minutes at 20 m AGL
WD_10	Mean wind direction (°) over the last 5/10 minutes/hourly – from True North (360) pole at 10 m AGL
WD_20	Mean wind direction (°) over the last 5/10 minutes/hourly – from True North (360) pole at 20 m AGL
Gust_10	Highest wind speed (m/s) that occurred in the last 5 minutes at 10 m AGL
SM	Station Monitor
X_tr	Input training data at SM1
X_va	Input validation data at SM2
X_te	Input testing data at SM3
y_tr	Output training data at SM1
y_va	Output validation data at SM2
y_te	Output testing data at SM3
SM1–Tr_B1, SM1–Tr_B2	Training data batches 1 and 2, respectively, at SM1.
SM2–Va_B1, SM2–Va_B2	Validation data batches 1 and 2, respectively, at SM2.
SM3–Te_B1, SM3–Te_B2	Testing data batches 1 and 2, respectively, at SM3.
SM–ST1, SM–ST2, SM–ST3, SM–ST4, SM–ST5	Evaluation datasets for 5 additional stations, respectively
1–D, 2–D, 3–D, 4–D, 5–D	1–dimension, 2–dimensional, 3–dimensional, 4–dimensional and 5–dimensional

Filters	Number of model layer neurons
Epochs	Number of training times to learn an input sequence
LSTM	Long Short Term Memory
CNN	Convolutional Neural Network
Conv1D	1–Dimensional CNN model
Conv2D	2–Dimensional CNN model
ConvLSTM2D	Hybrid network of ‘2–D CNN with LSTM’
MSE	mean_square_error
MAE	mean_absolute_error
ME	mean_error
RMSE	root_ mean_square_error

Figures

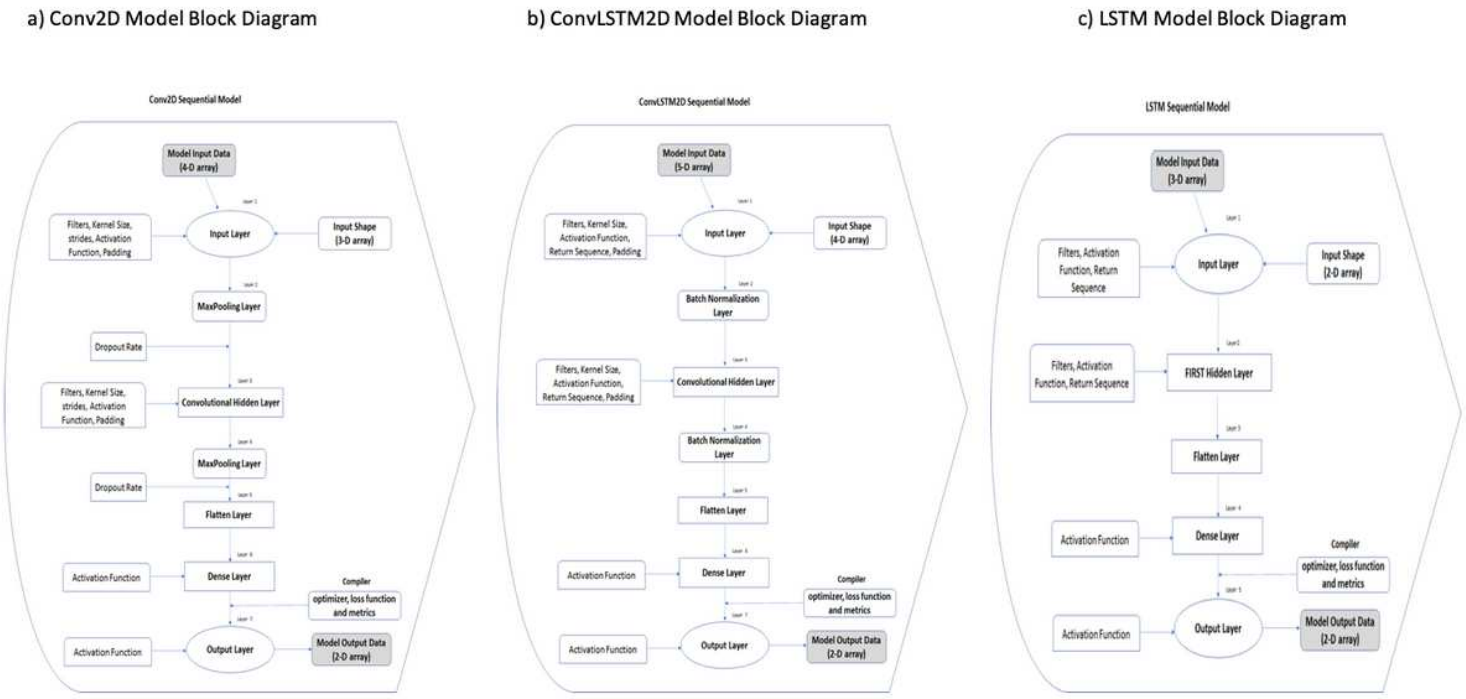

Figure 1

a-b: Model block diagrams of the Conv2D, ConvLSTM2D and LSTM sequential networks, respectively.

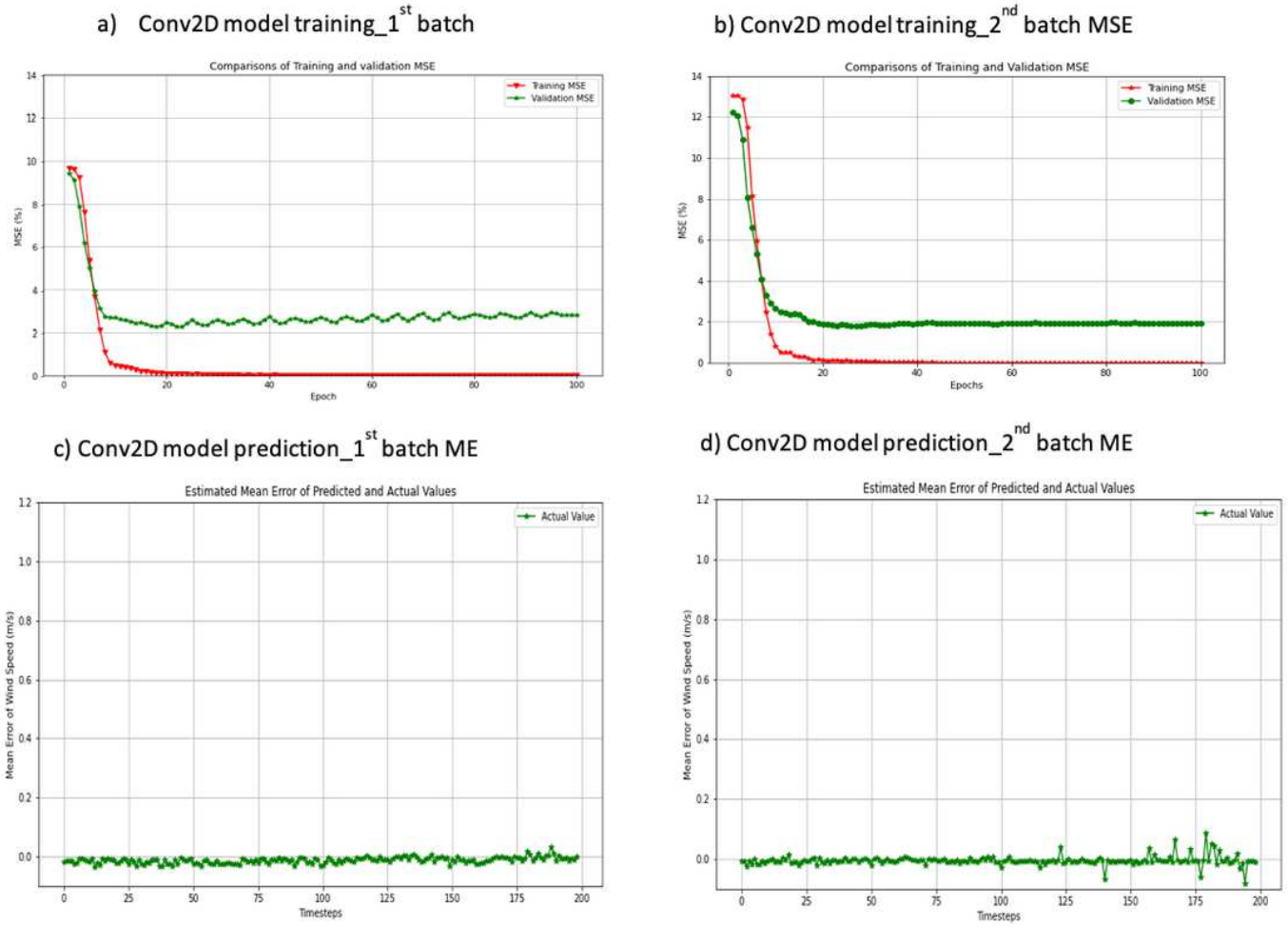

Figure 2

a-d: Performance comparisons (training and validation losses) and estimated ME values of Conv2D model prediction (200 timesteps).

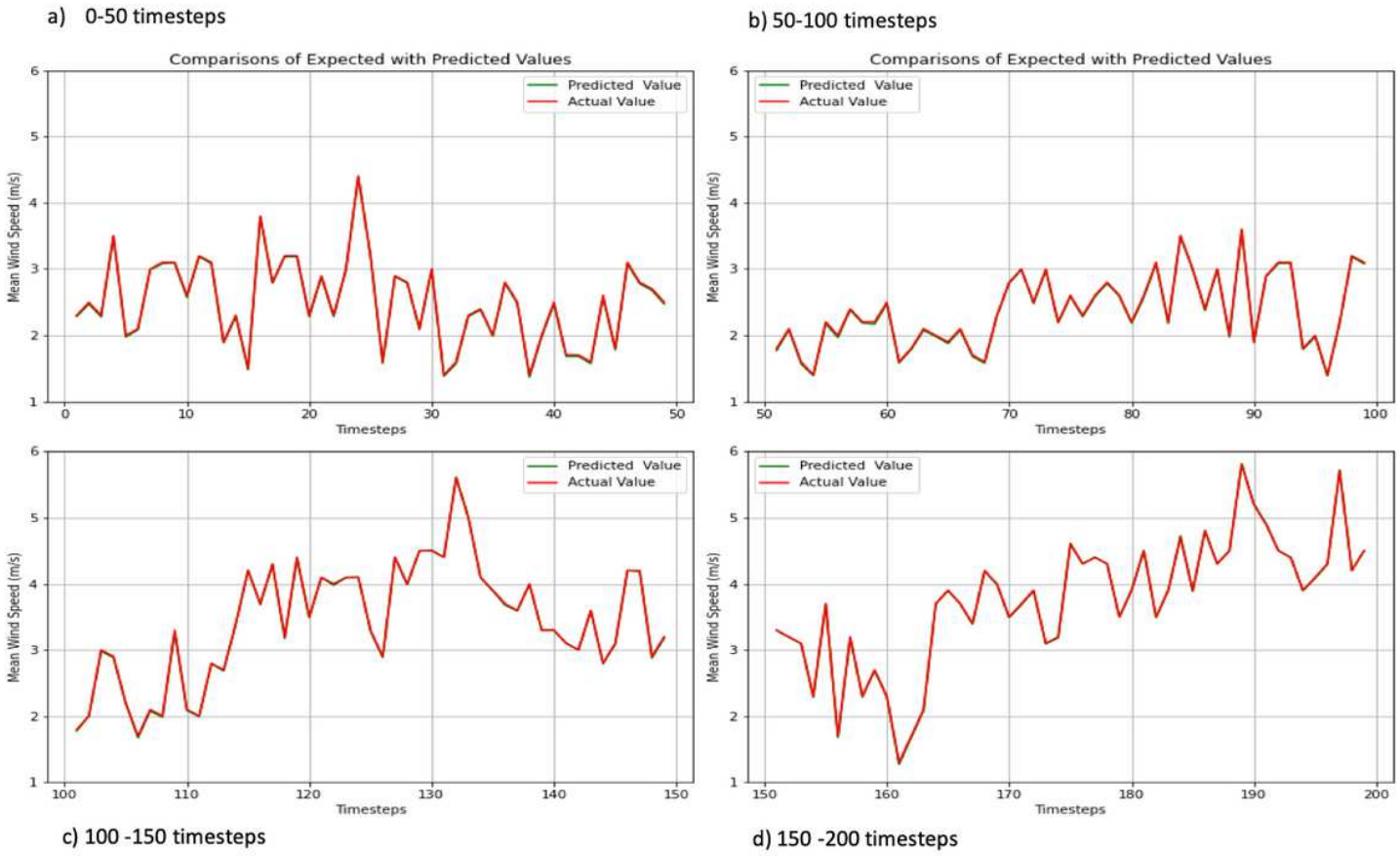

Figure 3

a-d: Conv2D model prediction for 1st batch dataset (200 timesteps)

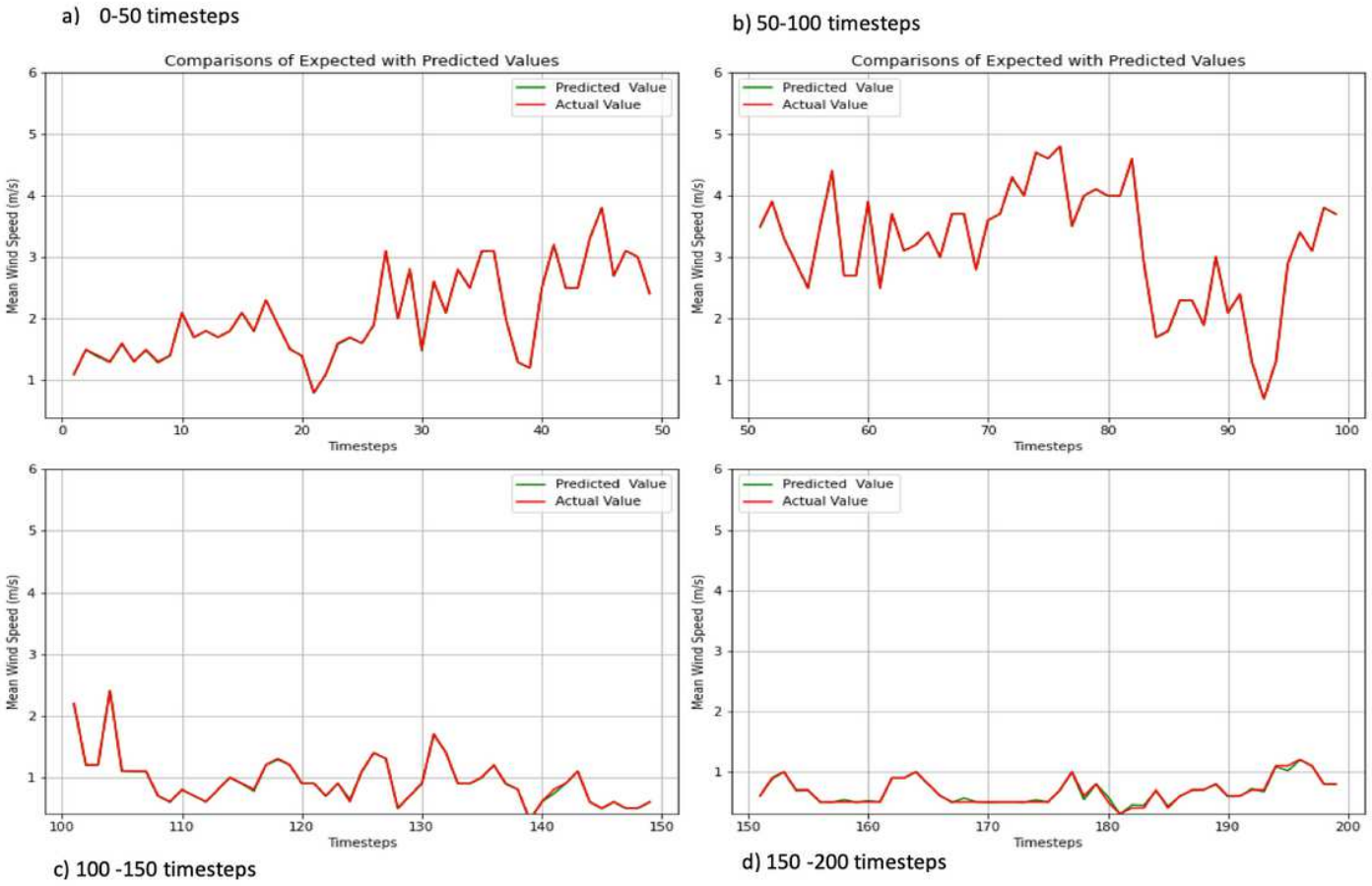

Figure 4

a-d: Conv2D model prediction for 2nd batch dataset (200 timesteps)

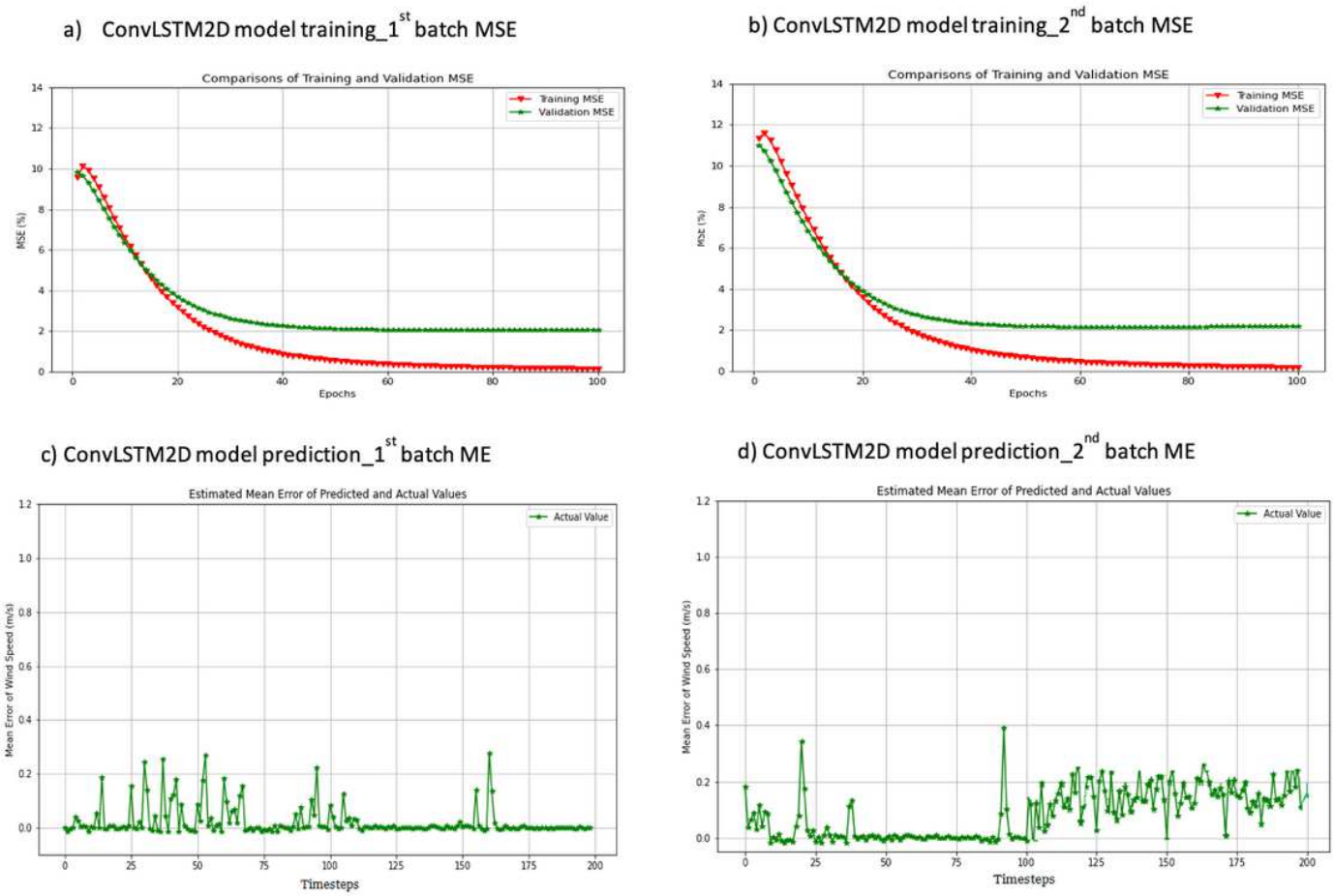

Figure 5

a-d: Performance comparisons (training and validation losses) and estimated ME values of ConvLSTM2D model prediction (200 timesteps).

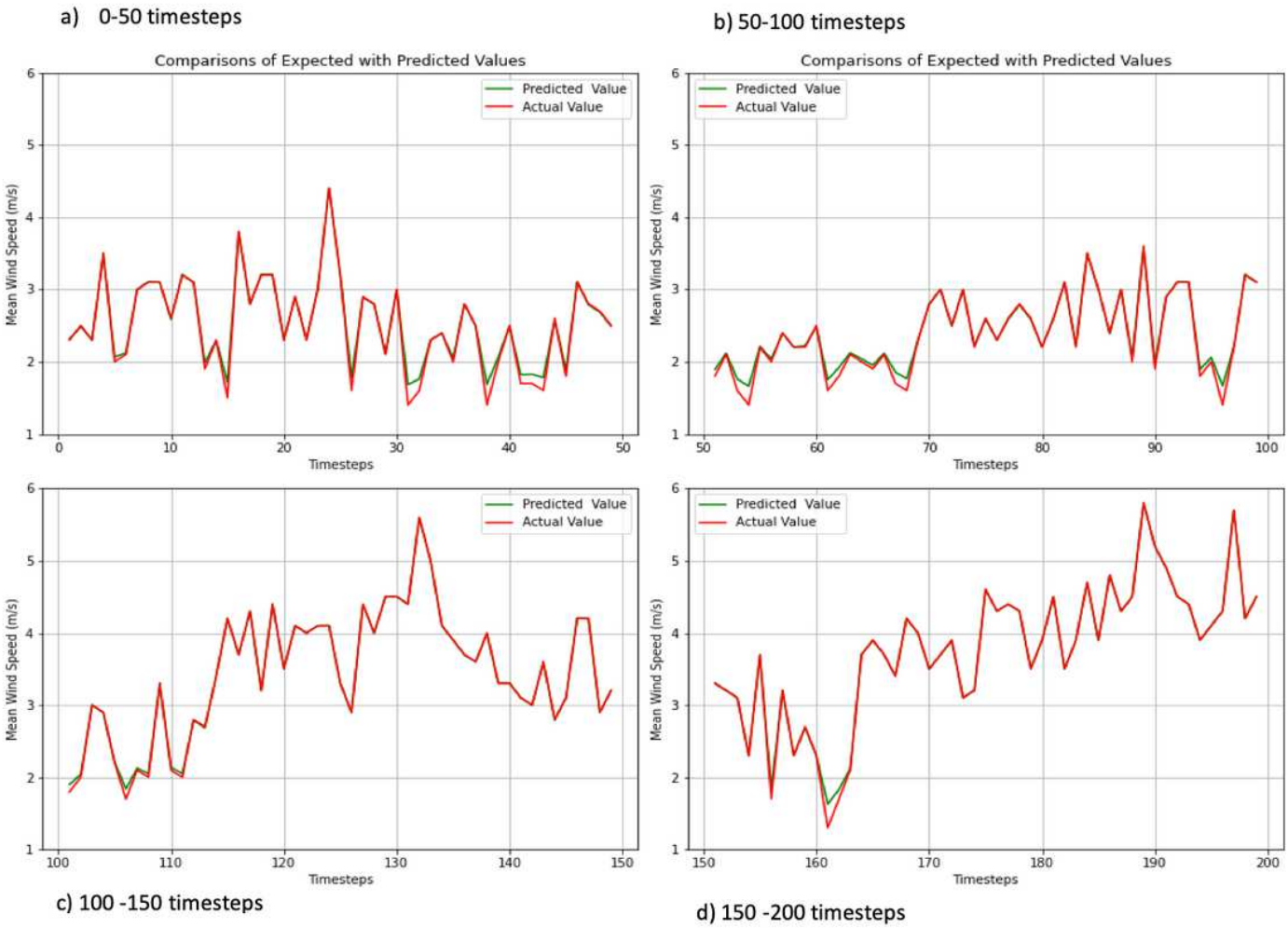

Figure 6

a-d: ConvLSTM2D model prediction for 1st batch dataset (200 timesteps)

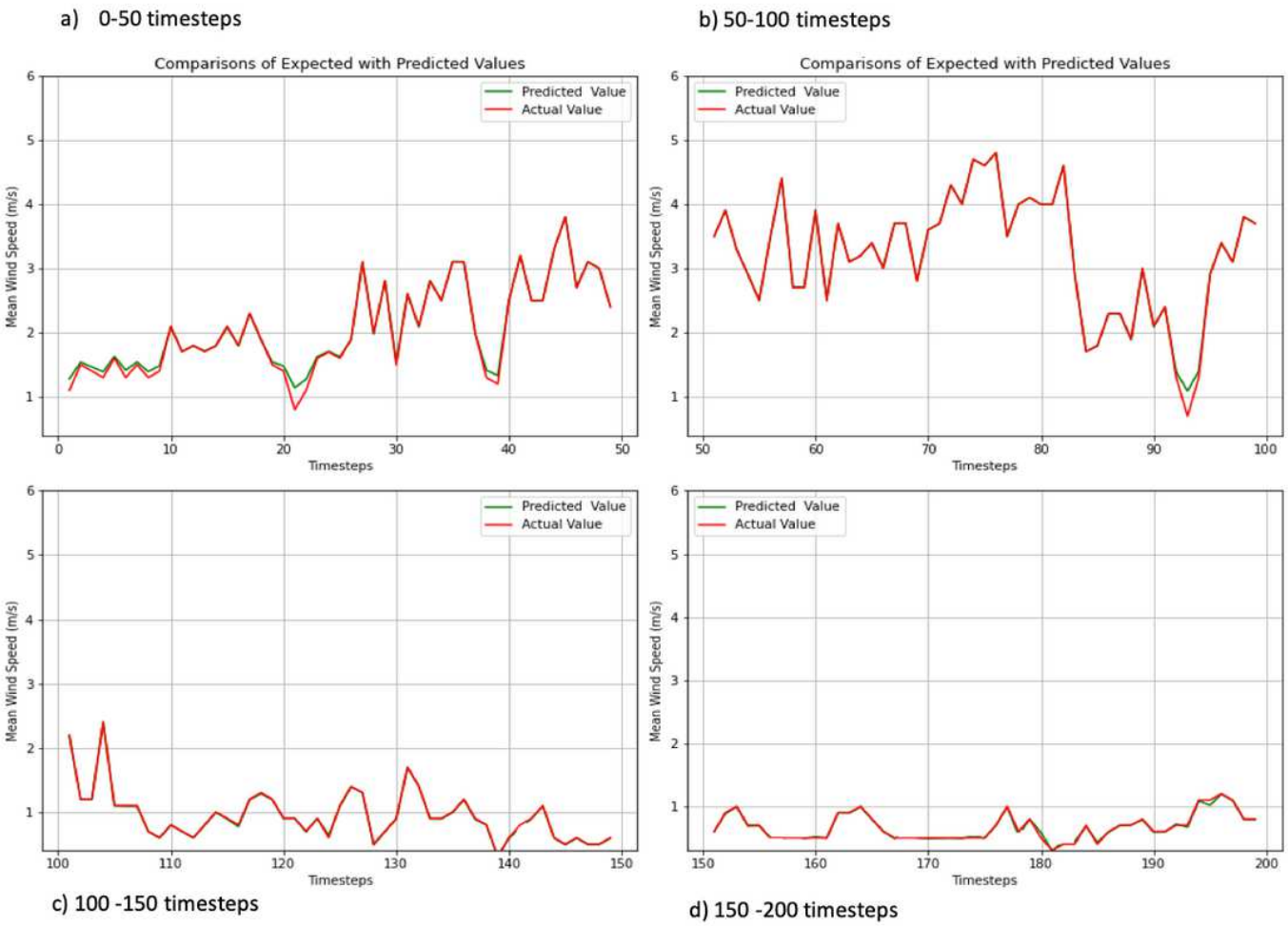

Figure 7

a-d: ConvLSTM2D model prediction for 2nd batch dataset (200 timesteps)

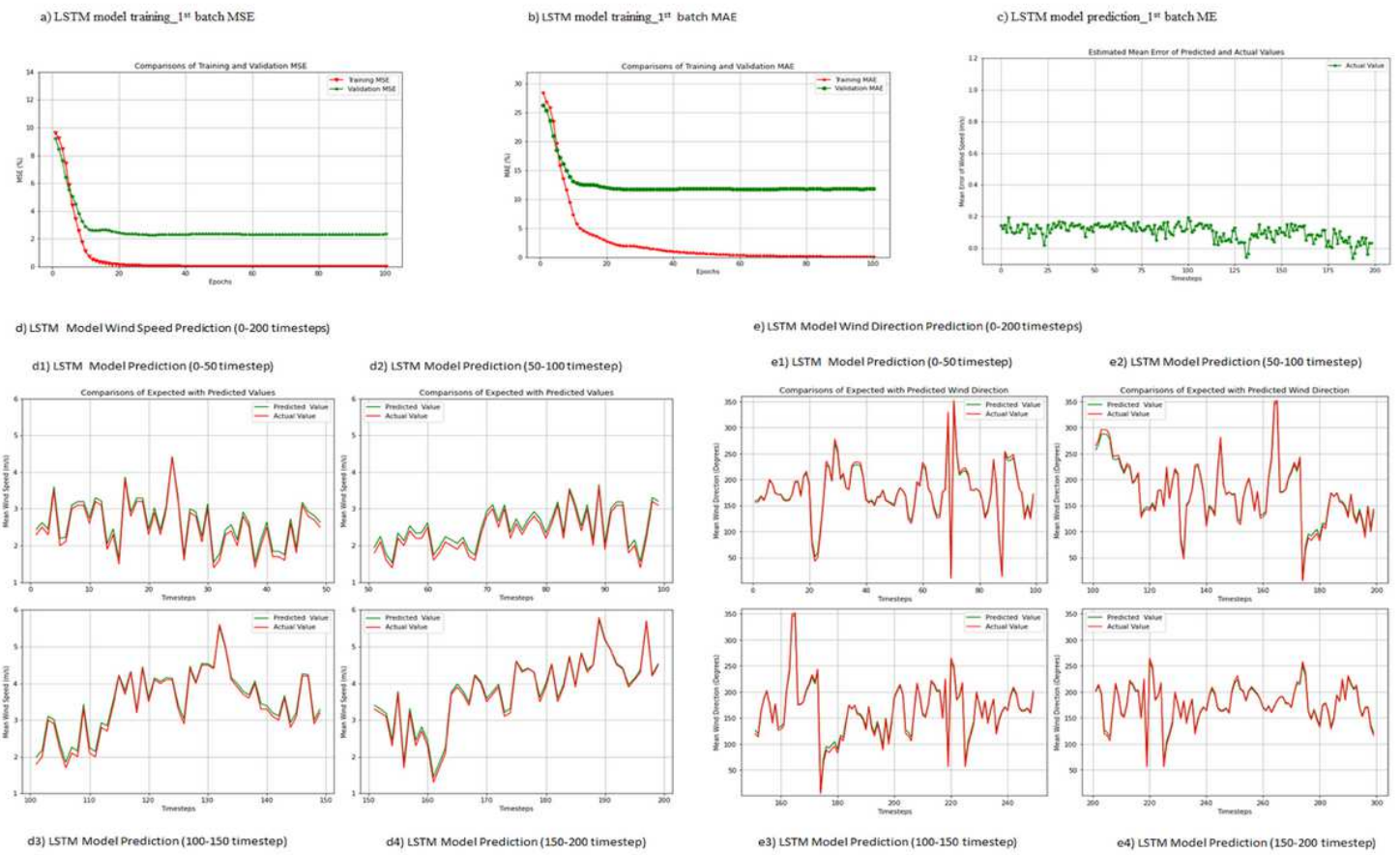

Figure 8

a-e: LSTM model performance comparisons for 1st batch dataset, model predictions ME values, and wind speed and direction predictions (200 timesteps)

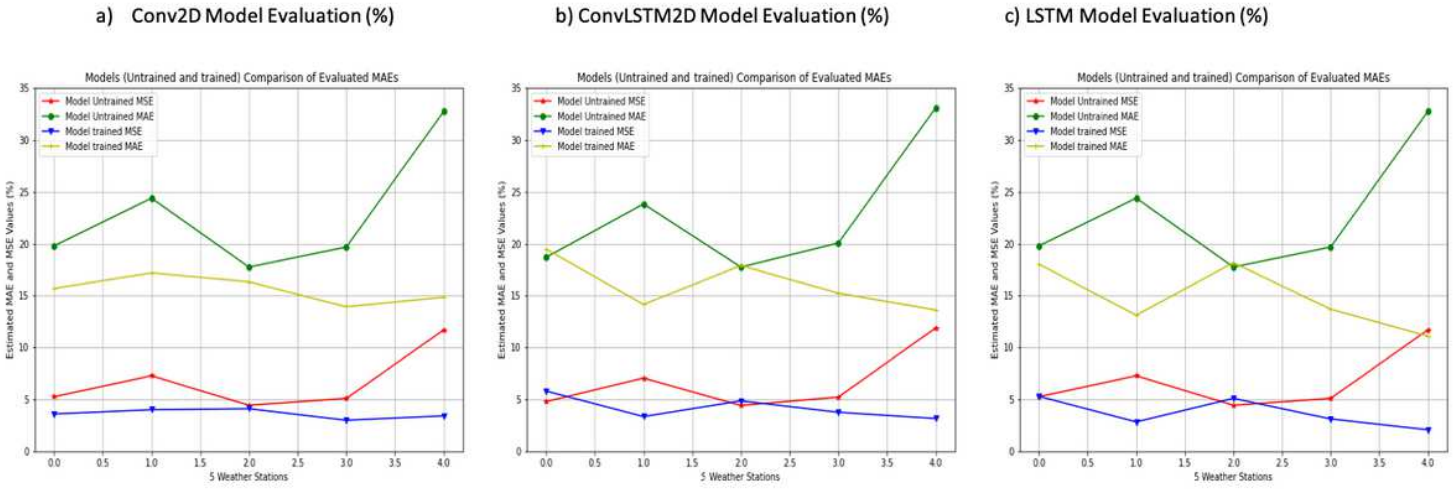

Figure 9

a-c: Evaluations of Conv2D, ConvLSTM2D and LSTM models (untrained and trained networks) in terms of MSE and MAE values, respectively.

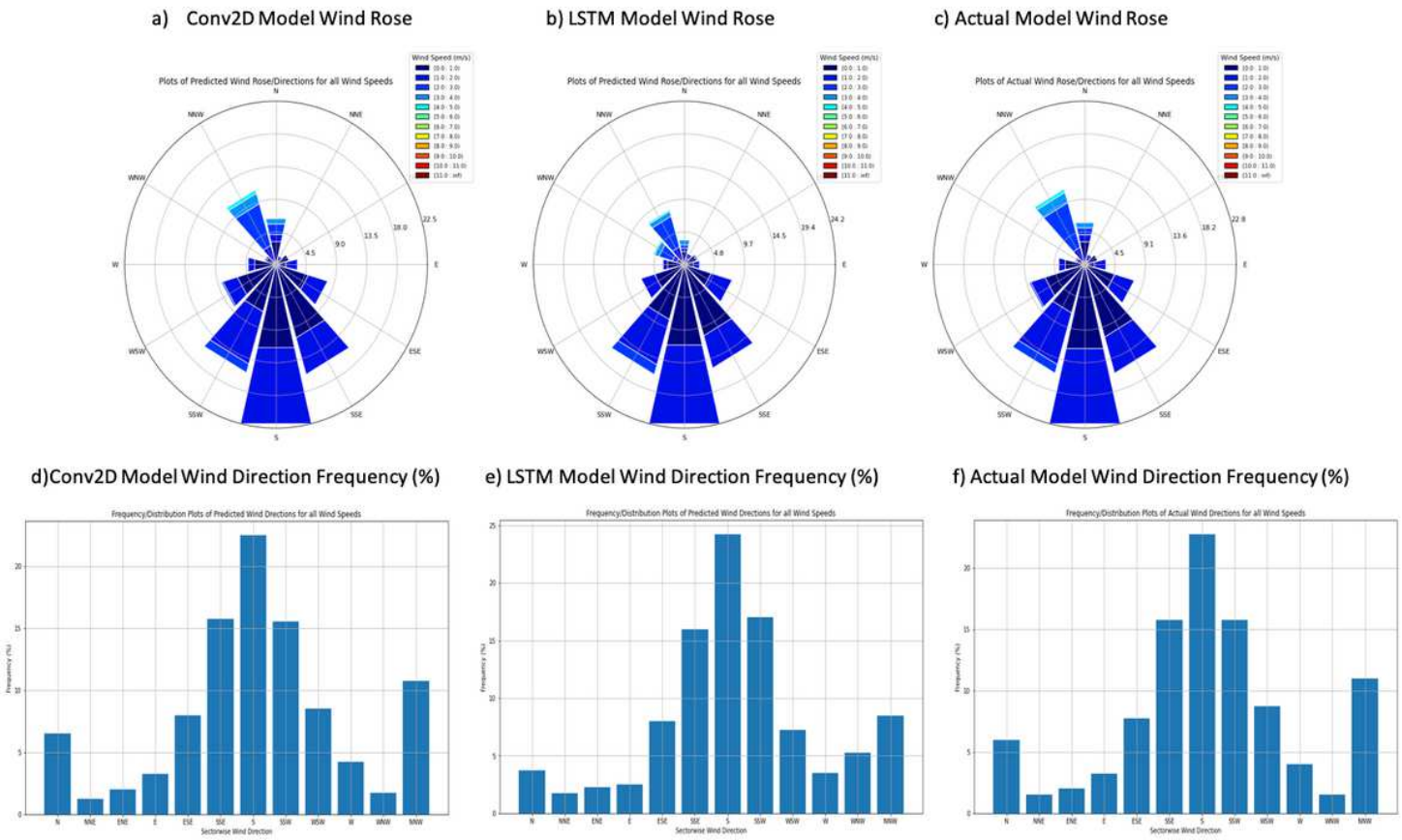

Figure 10

a-f: Plots of Conv2D, LSTM and Actual models' wind roses as well as their frequency occurrences of all wind directions, respectively.

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- SUPPLEMENTARY1.docx