DOI: https://doi.org/10.62985/j.huit_ojs.vol26.no2E.391

IMPACT OF HYPERPARAMETERS ON CNN PERFORMANCE FOR SHORT-TERM ELECTRICITY LOAD FORECASTING

Tuan Anh Nguyen[1], Thanh Ngoc Tran[2], Thanh Thuan Nguyen[3]

Faculty of Electrical Engineering Technology,

Industrial University of Ho Chi Minh City (IUH), Vietnam

Email: *[1]nguyenanhtuan@iuh.edu.vn, [2]tranthanhngoc@iuh.edu.vn,*

[3]nguyenthanhthuan@iuh.edu.vn

Received: 18 January 2026; Revised: 3 February 2026; Accepted: 21 April 2026

ABSTRACT

This paper investigates the effects of key hyperparameters on the forecasting accuracy of a one-dimensional convolutional neural network for short-term electricity load forecasting. A univariate load time series is first transformed into supervised learning samples using a 24-step sliding window. From these generated samples, the most recent 840 samples are retained to reflect recent operating patterns. These 840 samples are then split chronologically into 672 training samples (80%) and 168 testing samples (20%). On the training set, three-fold Time-Series cross-validation is employed to preserve temporal order. A total of 160 configurations are examined by varying the learning rate, number of convolutional filters, kernel size, and dropout rate, while keeping the number of dense units, batch size, and training epochs fixed. Forecasting performance is measured using mean absolute percentage error (MAPE). The results indicate that hyperparameter choices significantly affect both accuracy and fold-to-fold stability; the best configuration achieves a mean cross-validated MAPE of approximately 2.13% with a standard deviation of about 0.44%. These findings underscore the importance of meticulous hyperparameter selection in enhancing the reliability of compact 1D convolutional models for short-term load forecasting.

Keywords: Short-term electricity load forecasting, 1D convolutional neural network, Hyperparameter tuning, Time series forecasting.

1. INTRODUCTION

Short-term load forecasting (STLF) is a crucial task in modern power system operation and planning. Accurate forecasts enhance generation scheduling, optimize resource allocation, reduce reserve costs, support frequency, voltage control, and improve reliability, especially as renewable integration increases uncertainty and variability. As electricity demand is increasingly influenced by consumer behavior, operating conditions, and stochastic factors, both forecasting accuracy and model stability have become more critical, particularly for applications requiring fast decision-making and real-world deployment.

Over the past decades, STLF has been addressed using classical statistical and time-series forecasting methods, such as regression-based approaches [1], moving averages [2], exponential smoothing [3], ARIMA [4], SARIMA [5], and SARIMAX [6]. These methods offer interpretability and low computational cost, but often struggle to capture nonlinear relationships and complex temporal patterns. Subsequently, machine learning techniques-including SVR [7], decision trees [8], random forests [9], and boosting models-were introduced to better learn nonlinear mappings from data, such as XGBoost [10], CatBoost [11],

and LightGBM [12]. More recently, deep learning has emerged as a dominant trend, with architectures such as MLP [13], RNN [14], LSTM [15], GRU [16], 1D-CNN [17-18], Transformer [19], and hybrid models [20] that combine signal decomposition or multiple forecasting components. Owing to their ability to automatically extract features and model nonlinear dynamics, deep learning approaches frequently achieve superior performance in many STLF scenarios.

However, the performance of deep learning models depends not only on the data but also firmly on the choice of hyperparameters [21]. Architectural hyperparameters (number of filters, kernel size, and dense units), regularization hyperparameters (dropout), and optimization/training hyperparameters (learning rate, batch size, and number of epochs) directly impact representational capacity, convergence behavior, and generalization. In practice, changing hyperparameters within the same architecture can substantially alter forecasting errors and sensitivity across different training and validation splits. Therefore, identifying which hyperparameters have the most significant influence and in what direction is essential for building models that are both accurate and stable. Many STLF studies propose new architectures or report the best scores, but rarely explain the underlying mechanisms. Few systematically quantify how key hyperparameters influence error distributions and performance stability under time-aware evaluation. In univariate STLF, the 1D-CNN serves as a strong baseline: compact, fast, and deployable, and capable of learning local temporal patterns. We investigate the effects of its hyperparameters.

Accordingly, this paper establishes a time-consistent evaluation framework. The original daily load table, which contains 24 hourly load values for each day, is first converted into a univariate load sequence by arranging the hourly observations in chronological order. A sliding window with length $H = 24$is then applied to generate supervised input-output pairs, where each input sample contains the previous 24 hourly values and the target is the next load value. From the generated supervised samples, only the most recent 840 are retained to emphasize recent operating behavior. These retained samples are then divided chronologically into 672 training samples and 168 testing samples. A 3-fold TimeSeriesSplit is applied only to the training set to ensure temporal validity and avoid information leakage.

2. PROPOSED METHODOLOGY

2.1. CNN Model Architecture

In this study, a one-dimensional Convolutional Neural Network (1D-CNN) is adopted for univariate short-term load forecasting. The original dataset is organized as a daily table with 24 hourly load values per day. To construct a univariate forecasting series, the hourly load values are arranged in chronological order to form a single continuous sequence. A sliding window of length 24 is then used to generate supervised samples. For each sample, the input consists of 24 consecutive hourly load values, and the target is the immediately following load value. Therefore, each input sample is reshaped to (24, 1) before being fed into the 1D-CNN model. The original load series is transformed into supervised samples using a sliding window of length $H = 24$. Hence, each input sample is represented as:

$$\mathbf{x} \in \mathbb{R}^{H\times 1} = \mathbb{R}^{24\times 1} \tag{1}$$

Where "1" denotes a single input channel. The proposed CNN architecture consists of the following main components:

Conv1D layer: the network employs one Conv1D layer with Ffilters and kernel size K, followed by the ReLU activation. The temporal convolution can be expressed as:

$$z_f(t) = \text{ReLU}\left(\sum_{i=0}^{K-1} w_{f,i}\ x(t+i) + b_f\right), f = 1, \ldots, F \tag{2}$$

In the provided implementation, Conv1D uses the default "valid" padding, leading to an output length:

$$L_1 = H - K + 1 \tag{3}$$

MaxPooling1D layer: a MaxPooling1D layer with pool_size = 2 is applied to reduce the temporal resolution and retain salient features:

$$L_2 = \left\lfloor \frac{L_1}{2} \right\rfloor \tag{4}$$

Flatten + Dense layer: the pooled feature maps are flattened and fed into a fully connected layer with Uhidden units (dense_units) and ReLU activation to capture higher-level nonlinear relationships:

$$\text{h} = \text{ReLU}(\text{Wv} + \text{c}) \tag{5}$$

Where **v** denotes the flattened feature vector, a Dropout layer with rate p is included to regularize and mitigate overfitting. Output layer, the final Dense layer has one neuron to predict the next-step load value: $\hat{y} \in \mathbb{R}$

The model is trained using the Mean Squared Error (MSE) loss function and the Adam optimizer, with the learning rate initialized at the start of theof the η:

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N} (y_n - \hat{y}_n)^2 \tag{6}$$

2.2. CNN Model Hyperparameters

In machine learning, model parameters are learned from data during training to minimize the loss. In contrast, hyperparameters are settings that the user designs and controls to influence the model structure and training strategy. As a result, hyperparameter choices directly influence representational capacity, convergence, and generalization. In 1D-CNN time-series forecasting, hyperparameters are commonly grouped into architecture, regularization, and training/optimization. Architecture includes the number of filters, kernel size, and the number of dense layer units (dense_units), which govern temporal feature extraction and model complexity. Regularization is typically represented by dropout to reduce overfitting. Training/optimization includes learning rate, batch size, and epochs, which affect learning dynamics and update stability.

In this study, we focus on hyperparameters that most directly impact feature learning and training stability: filters, kernel_size, dropout, and the Adam learning rate. To ensure a fair comparison, dense_units, batch_size, and epochs are fixed, allowing performance differences to be attributed primarily to the selected hyperparameters.

2.3. Flowchart for Evaluating the Impact of Hyperparameters on the CNN Model

Figure 1 presents the complete framework adopted in this study to investigate how different hyperparameter settings affect the predictive performance of the 1D-CNN model. The workflow begins with the original input dataset, denoted as close paren, which contains historical load observations. In its raw form, the data are organized as daily records, with each row containing 24 hourly electricity load values. To prepare the data for time-series forecasting, these hourly values are first rearranged in chronological order to form a

continuous univariate load sequence. This step ensures that the temporal dependency of the load demand is preserved and that the sequential nature of the forecasting problem is properly represented before model construction.

After the chronological load sequence is obtained, a data processing stage is performed to transform the raw series into a supervised learning format suitable for the 1D-CNN model. Specifically, a sliding-window strategy with a window length of 24 is employed. In this mechanism, each input sample is formed from 24 consecutive hourly load values, while the corresponding output is the target value to be predicted in the next time step. By moving the window one step forward throughout the entire sequence, a large number of input-output pairs $(X, Y)$are generated. This transformation allows the 1D-CNN to learn meaningful temporal patterns from neighboring hourly observations and to capture the short-term dependence structure embedded in the load series.

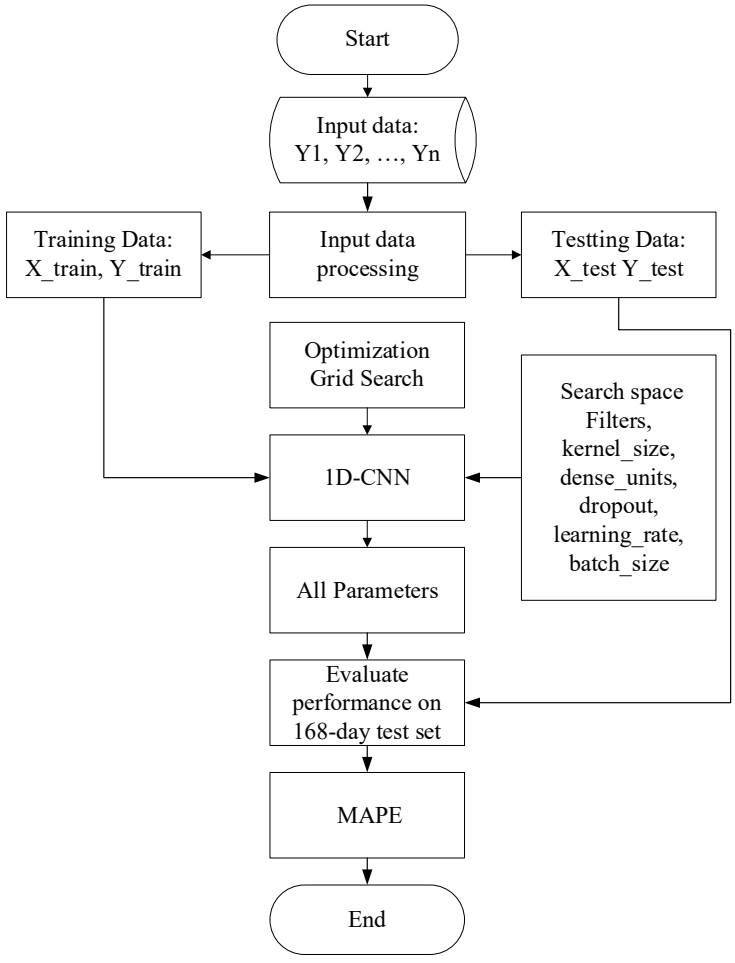

Fig. 1. Workflow for assessing hyperparameter effects on 1D-CNN performance

From all generated samples, the most recent 840 observations are retained for the experiments. These samples are then divided chronologically into a training set of 672 samples $(X_{train}, Y_{train})$and an independent test set of 168 samples (X_{test}, Y_{test}). This chronological split is used to preserve the temporal order of the data and to avoid information leakage from future observations into the training process.

As illustrated in Figure 1, hyperparameter optimization is performed only on the training set using a grid search. The search space includes several key hyperparameters of the 1D-CNN model, namely the number of filters, kernel size, number of dense units, dropout rate, learning

rate, and batch size. To ensure a reliable selection process for time-series forecasting, each candidate configuration is evaluated using a 3-fold TimeSeriesSplit.

After the optimization stage, the best hyperparameter combination is selected as the final parameter set for the 1D-CNN model. The optimized model is then tested on the held-out 168-day test set to evaluate its generalization capability on unseen data. Finally, the forecasting accuracy is measured using the Mean Absolute Percentage Error (MAPE), which is adopted as the main performance metric in this study.

3. EXPERIMENTAL SETUP AND RESULTS

3.1. Data Analysis

The dataset used in this study was collected from the HCMPC P24 system, recording electricity load data in Ho Chi Minh City from October 2010 to December 2012. It contains 3,004 records, each corresponding to a day, represented by 25 columns. The first column is the Date, and the remaining columns (from 00:00 to 23:00) represent hourly electricity load values in MW. The dataset is complete, with no missing values, ensuring its integrity for analysis and model training. Significant fluctuations in electricity demand are observed, with peak hours (e.g., late afternoon) showing higher demand compared to off-peak hours (e.g., early morning). The load peaks during high-demand hours and decreases at night. The difference between minimum and maximum load reflects the substantial variation in electricity consumption throughout the day, influenced by factors such as consumer behavior, weather, and operational aspects in the city's power system. To further understand the dataset and its implications for load forecasting, Figure 2 provides a visual representation of the entire load time series, along with the train-test split.

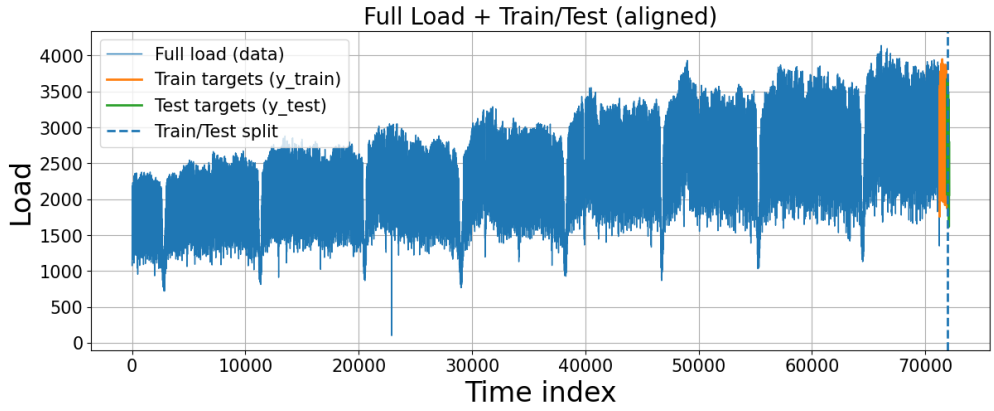

Fig. 2. Full load time series with aligned train/test split

This figure displays the entire dataset, with the train and test subsets clearly distinguished, enabling an assessment of the model's ability to predict future load patterns. The alignment of these sets ensures a direct comparison of the model's performance on unseen data, evaluating its generalization ability. The data shows significant fluctuations, especially between peak and off-peak hours, highlighting the complexity of electricity demand. These fluctuations are crucial for accurate forecasting. Figure 3 further details the split of the training and test sets for the target load series (y) used for model evaluation.

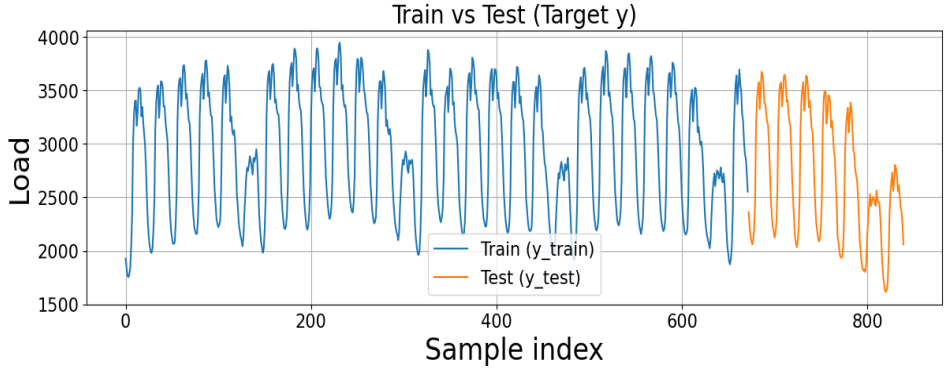

Fig. 3. Train vs. test split of the target load series (y)

The data is divided into two sets: 80% allocated to the training set and 20% to the testing set. After converting the original hourly load data into supervised samples using a 24-step sliding window, the most recent 840 generated samples are retained for the experiments. These retained samples are then divided chronologically into 672 training samples and 168 testing samples. Thus, the independent test set corresponds to the final 168 samples, while the remaining 672 samples are used for model development and TimeSeriesSplit cross-validation.

3.2. Hyperparameter Search Space Definition

Table 1 presents the hyperparameter search space for the 1D-CNN model, listing the values considered for key hyperparameters during optimization.

Table 1. Hyperparameter search space for the 1d-cnn model

Hyperparameter	Values used	Note
Learning Rate	[4×10^{-4}, 5×10^{-4}, 6×10^{-4}, 8×10^{-4}, 1×10^{-3}]	Tuned
Filters	[32, 64]	Tuned
Kernel Size	[3, 4, 5, 7]	Tuned
Dropout Rate	[0.05, 0.10, 0.15, 0.20]	Tuned
Dense Units	32	Fixed
Batch Size	32	Fixed
Epochs	500	Fixed

In this study, several hyperparameters were tuned to optimize the model's performance. The learning rate was varied across five values: $[4\times10^{-4}, 5\times10^{-4}, 6\times10^{-4}, 8\times10^{-4}, 1\times10^{-3}]$, as it plays a crucial role in model convergence. A learning rate that is too large can lead to overshooting, while one that is too small can cause slow convergence or getting stuck in local minima. By testing different values, the goal was to find an optimal learning rate that ensures fast yet stable convergence without compromising the model's generalization.

Filters were tested with values 32 and 64 to balance model complexity and generalization. A larger number of filters captures more complex features, but it also increases the risk of overfitting. Kernel sizes were tested at 3, 4, 5, and 7 to balance localized patterns and broader trends. The dropout rates were set to 0.05, 0.10, 0.15, and 0.20 to prevent overfitting and to ensure robust feature learning and improved generalization. Dense units, batch size, and epochs were fixed at 32, 32, and 500, respectively, to ensure model stability and sufficient training time without overfitting. These hyperparameters ensure the model captures temporal patterns in electricity load data while maintaining efficiency and preventing overfitting.

3.3. Experimental Setup and Environment

All experiments were conducted in Google Colab using the Python 3 runtime. To ensure compatibility among the optimization and wrapper libraries, the software environment was prepared by upgrading pip and reinstalling the core scientific stack, including NumPy, SciPy, scikit-learn, and SciKeras (via pip uninstall followed by a clean pip install). The forecasting model was implemented in TensorFlow/Keras, while hyperparameter tuning was executed using SciKeras (KerasRegressor) integrated with scikit-learn search utilities (e.g., GridSearchCV and RandomizedSearchCV). Data loading and preprocessing were performed with pandas and NumPy, and visualization was performed with Matplotlib. The input dataset was read from a CSV file stored on Google Drive and then converted into supervised learning samples for the 1D-CNN pipeline.

Regarding hardware, the Colab runtime was configured with an NVIDIA GPU accelerator (A100), and the High-RAM option was enabled to support repeated model training during the hyperparameter search process. This configuration provides sufficient computational resources for convolutional model training and allows efficient execution of multiple experimental runs under a consistent cloud-based environment. In the current study, performance stability was primarily assessed by variation across the three TimeSeriesSplit folds. Repeated runs with different random seeds were not included in the present experiments and will be considered in future work.

3.4. Experimental Results

In this experiment, the effect of dropout on 1D-CNN forecasting accuracy is evaluated by examining the distribution of MAPE (%) across hyperparameter configurations. Fig. 4 compares four dropout settings (0.05, 0.10, 0.15, and 0.20) via boxplots, where lower MAPE indicates better performance. Fig. 4 shows that the median MAPE increases as dropout rises, implying reduced accuracy under stronger regularization. This is confirmed by Table 2: median MAPE = 3.1435% (0.05), 3.3594% (0.10), 3.6827% (0.15), and 4.2980% (0.20).

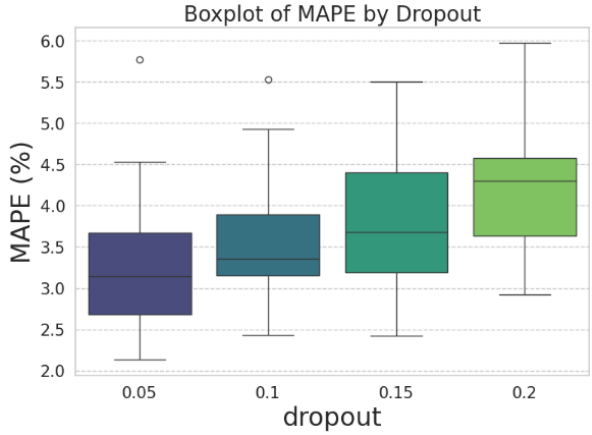

Fig. 4. Effect of dropout rate on forecasting accuracy measured by MAPE (%)

Moreover, Table 2 reports the descriptive statistics corresponding to the components of the boxplots: Q1 and Q3 represent the lower and upper edges of each box in Fig. 4. At the same time, IQR = Q3 − Q1 determines the box height and reflects the variability of MAPE at each dropout level.

Table 2. Statistics of mape for each dropout value

Dropout	Count	Mean	STD	Min	Q1	Median	Q3	IQR	Max
0.05	40	3.250	0.726	2.131	2.6783	3.1435	3.673	0.995	5.77
0.10	40	3.604	0.689	2.431	3.1516	3.3594	3.891	0.739	5.53
0.15	40	3.835	0.760	2.421	3.1869	3.6827	4.403	1.216	5.50
0.20	40	4.215	0.692	2.918	3.6380	4.2980	4.577	0.939	5.97

Regarding dispersion, dropout 0.15 yields the largest IQR (1.216), indicating a wider spread of forecasting errors, whereas dropout 0.10 yields a smaller IQR (0.7389), implying a more concentrated error distribution. In addition, the min and max values in Table 2 support the interpretation of the tails (whiskers/outliers) observed in Fig. 4, with the maximum error reaching 5.97% at dropout 0.20. These findings suggest that an excessively high dropout rate may reduce the model's adequate representational capacity in the considered 1D-CNN configuration, thereby increasing forecasting error.

Figure 5 presents a boxplot of MAPE (%) for two filter settings (32 and 64), evaluating the effect of the number of convolutional filters in the Conv1D layer on the 1D-CNN model's forecasting accuracy (lower MAPE indicates better performance).

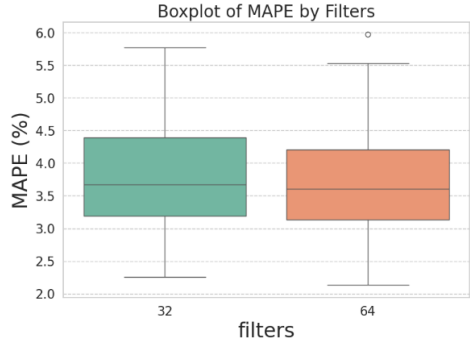

Fig. 5. Effect of filter rate on forecasting accuracy measured by MAPE (%)

From Fig. 5, the MAPE distributions under the two filter configurations appear highly similar; however, the box for filters = 64 is slightly shifted downward and has a lower median than the box for filters = 32, indicating a slight reduction in the typical forecasting error with more filters. In addition, the box for filters = 64 is marginally narrower, suggesting slightly lower MAPE variability. Both configurations still show upper-side outliers, reflecting a few cases with relatively large errors under certain hyperparameter combinations. Table 3 provides quantitative support for these visual observations.

Table 3. Statistics of mape for each filter value

Filters	Count	Mean	Std	Min	Q1	Median	Q3	Iqr	Max
32	80	3.766	0.768	2.257	3.186	3.668	4.393	1.206	5.773
64	80	3.685	0.819	2.131	3.131	3.605	4.204	1.073	5.968

Median MAPE drops from 3.6682% (filters=32) to 3.6052% (filters=64), and mean MAPE declines from 3.7666% to 3.6852%, suggesting lower average error with 64 filters. Dispersion also tightens: IQR (Q3−Q1) decreases from 1.2061 (3.1869–4.3930) to 1.0735 (3.1313–4.2048), indicating a more concentrated error distribution. Box edges represent Q1 and Q3, aligning with Table 3 and confirming reduced spread. Tail behavior matches Fig. 5:

the minimum MAPE is 2.1313% at 64 filters, while the maximum 5.9689% also occurs there, producing upper outliers. Overall, 64 filters yield modest accuracy gains and slightly tighter variability. Figure 6 presents a boxplot of MAPE (%) across four kernel sizes (3, 4, 5, and 7) to evaluate how the convolutional kernel size in the Conv1D layer affects the 1D-CNN model's forecasting accuracy (lower MAPE indicates better performance).

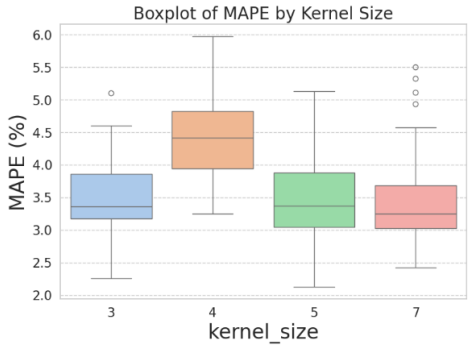

Fig. 6. Effect of kernel_size on forecasting accuracy measured by MAPE (%)

In Fig. 6, the kernel size = 4 configuration is visibly shifted upward, with a higher median and a generally higher distribution than the other settings, indicating worse typical forecasting accuracy. In contrast, kernel sizes 3, 5, and 7 yield noticeably lower medians, with kernel size = 7 showing the lowest median and a relatively tighter box, suggesting both improved accuracy and more stable performance. Upper-side outliers are present across all kernel sizes, indicating that a small number of hyperparameter combinations still produce relatively large errors; however, kernel size = 4 exhibits the most unfavorable upper tail. These visual observations are quantitatively supported by Table 4.

Table 4. Statistics of mape for each kernel size value

Kernel Size	Count	Mean	Std	Min	Q1	Median	Q3	IQR	Max
3	40	3.48	0.663	2.257	3.172	3.355	3.860	0.687	5.097
4	40	4.46	0.628	3.252	3.943	4.415	4.827	0.883	5.968
5	40	3.49	0.648	2.131	3.045	3.370	3.881	0.835	5.132
7	40	3.48	0.760	2.425	3.023	3.244	3.688	0.664	5.501

The median MAPE is highest for kernel_size = 4 (4.4157%), decreases to 3.3552% for kernel_size = 3, 3.3707% for kernel_size = 5, and reaches a minimum of 3.2442% for kernel_size = 7, indicating that kernel size 7 provides the best typical performance. The mean MAPE follows a similar trend, with kernel_size = 4 yielding the highest average error (4.4568%), while kernel_size = 7 yields the best (3.4824%). In terms of dispersion, kernel_size = 7 has the smallest IQR (0.6649), showing the most concentrated error distribution, while kernel_size = 4 exhibits the largest IQR (0.8833), reflecting the highest variability. Kernel_size = 4 also has the highest maximum MAPE (5.9689%) and a relatively high minimum MAPE (3.2520%). Overall, kernel_size = 7 yields the lowest median MAPE and the tightest error distribution, making it the best choice, while kernel_size = 4 results in higher errors and greater variability.

Figure 7 presents a boxplot of MAPE (%) across five learning rates (0.0004, 0.0005, 0.0006, 0.0008, and 0.0010) to evaluate the effect of the learning rate on the 1D-CNN model's forecasting accuracy (lower MAPE indicates better performance).

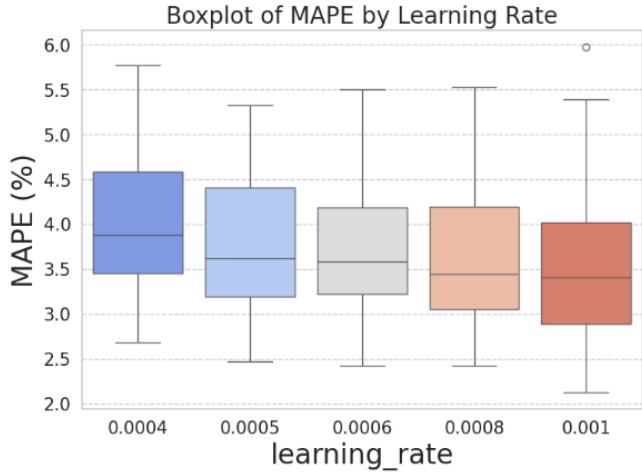

Fig. 7. Effect of learning_rate on forecasting accuracy measured by MAPE (%)

From Fig. 7, it can be observed that the MAPE distribution gradually shifts downward as the learning rate increases from 0.0004 to 0.0010, indicating an overall improvement in forecasting accuracy within the investigated learning-rate range. The boxes corresponding to higher learning rates, especially 0.0008 and 0.0010, are positioned lower than those of 0.0004, reflecting reduced typical errors. At the same time, upper-side outliers are present across all learning-rate settings, indicating that some hyperparameter combinations still yield relatively large errors. These visual observations are quantitatively supported by Table 5.

Table 5. Statistics of mape for each learning rate value

Learning rate	Count	Mean	Std	Min	Q1	Median	Q3	IQR	Max
0.0004	32	4.003	0.778	2.682	3.4533	3.8778	4.5870	1.1337	5.7737
0.0005	32	3.809	0.783	2.474	3.1907	3.6181	4.4113	1.2206	5.3285
0.0006	32	3.695	0.753	2.425	3.2188	3.5848	4.1859	0.9671	5.5015
0.0008	32	3.589	0.713	2.421	3.0508	3.4426	4.1898	1.1391	5.5317
0.0010	32	3.531	0.887	2.131	2.8864	3.4050	4.0221	1.1357	5.9689

As the learning rate increases, both the median and the mean MAPE decrease, with the median decreasing from 3.8778% at 0.0004 to 3.4050% at 0.0010, and the mean declining from 4.0029% to 3.5318%. This indicates that higher learning rates generally lead to lower typical prediction errors. Regarding dispersion, the learning rate of 0.0006 produces the smallest IQR (0.9671), indicating the most stable error distribution, whereas 0.0005 yields the largest IQR (1.2206), suggesting greater variability. Although higher learning rates, such as 0.0008 and 0.0010, continue to improve the central tendency, they do not reduce dispersion further compared with 0.0006. The minimum MAPE also improves as the learning rate increases, reaching 2.1313% at 0.0010. However, the maximum MAPE remains relatively high, particularly at 0.0010 (5.9689%), suggesting that large prediction errors may still occur in some cases. Overall, a learning rate of 0.0010 provides the best typical performance, as shown by the lowest median and mean MAPE, while 0.0006 offers the most stable central error distribution.

4. CONCLUSION AND FUTURE WORK

In this study, we investigated the impact of key hyperparameters on the forecasting accuracy of a 1D Convolutional Neural Network (CNN) model for short-term electricity load forecasting. A univariate load time series was converted into a supervised learning format using a 24-step sliding window. The dataset was split into training (80%) and testing (20%) sets, with a 3-fold TimeSeriesSplit applied to ensure temporal validity. Across 160 configurations, the results showed that hyperparameter tuning significantly impacts forecasting accuracy and stability. The best configuration achieved a cross-validation MAPE of 2.13% with a standard deviation of 0.44%. Key findings include that increasing dropout led to higher median MAPE, indicating over-regularization reduced representational capacity. Increasing the number of filters from 32 to 64 slightly improved accuracy, while a kernel size of 7 produced lower typical errors. Larger learning rates improved accuracy by reducing the MAPE.

Future work includes validating the model on multiple datasets under various seasonal or regional conditions, expanding to a multivariate framework with exogenous variables, and comparing it with other models such as LSTM, GRU, and Transformer. Efficient hyperparameter optimization strategies are necessary to reduce computational costs, and further analysis of training and inference times should inform real-world deployment. Addressing these areas will improve the model's adaptability and efficiency for practical energy management systems.

REFERENCES

[1] N. Shabbir, R. Ahmadiahangar, A. Rosin, M. Jawad, J. Kilter, and J. Martins, "XGBoost-based short-term electrical load forecasting considering trends and periodicity in historical data," in 2023 *IEEE Int. Conf. Energy Technol. Future Grids (ETFG)*, 2023, pp. 1–6, doi: https://doi.org/10.1109/ETFG55873.2023.10407926

[2] U. I. Lezama Lope, A. Benavides-Vázquez, G. Santamaría-Bonfil, and R. Z. Ríos-Mercado, "Fast and efficient very short-term load forecasting using analogue and moving average tools," *IEEE Lat. Am. Trans.*, vol. 21, no. 9, pp. 1015–1021, 2023, doi: https://doi.org/10.1109/TLA.2023.10251808.

[3] M. Pleños, "Time series forecasting using Holt-Winters exponential smoothing: Application to abaca fiber data," *Zesz. Nauk. SGGW Warszawie - Probl. Rol. Światowego.*, vol. 22, no. 2, pp. 17–29, 2022, doi: https://doi.org/10.22630/PRS.2022.22.2.6.

[4] U. Samal and A. Kumar, "Enhancing software reliability forecasting through a hybrid ARIMA-ANN model," *Arab. J. Sci. Eng.,* vol. 49, pp. 7571–7584, 2024, doi: https://doi.org/10.1007/s13369-023-08486-1.

[5] N. T. N. Anh, N. N. Anh, T. N. Thang, V. K. Solanki, R. G. Crespo, and N. Q. Dat, "Online SARIMA applied for short-term electricity load forecasting," *Appl. Intell.,* vol. 54, no. 1, pp. 1003–1019, 2024, doi: https://doi.org/10.1007/s10489-023-05230-y.

[6] V. Gayathry, D. Kaliyaperumal, and S. R. Salkuti, "Seasonal solar irradiance forecasting using artificial intelligence techniques with uncertainty analysis," *Sci. Rep.,* vol. 14, no. 1, pp. 1–19, 2024, doi: https://doi.org/10.1038/s41598-024-68531-3.

[7] M. Safari, A. H. Rabiee, and J. Joudaki, "Developing a support vector regression model for prediction of main and lateral bending angles in laser tube bending process," *Materials.*, vol. 16, no. 8, 2023, doi: https://doi.org/10.3390/ma16083251.

[8] A. Ajder, H. A. A. Hamza, and R. Ayaz, "Wavelet-enhanced hybrid LSTM-XGBoost model for predicting time series containing unpredictable events," *IEEE Access.*, vol. 13, no. April, pp. 58671–58679, 2025, doi: https://doi.org/10.1109/ACCESS.2025.3556540.

[9] M. Goyal and M. Pandey, "A systematic analysis for energy performance predictions in residential buildings using ensemble learning," *Arab. J. Sci. Eng.*, 2020, doi: https://doi.org/10.1007/s13369-020-05069-2.

[10] L. Semmelmann, S. Henni, and C. Weinhardt, "Load forecasting for energy communities: A novel LSTM-XGBoost hybrid model based on smart meter data," *Energy Inform.*, vol. 5, no. 1, pp. 1–21, 2022, doi: https://doi.org/10.1186/s42162-022-00212-9.

[11] L. Zhang and D. Jánošík, "Enhanced short-term load forecasting with hybrid machine learning models: CatBoost and XGBoost approaches," *Expert Syst. Appl.*, vol. 241, 2024, doi: https://doi.org/10.1016/j.eswa.2023.122686.

[12] X. Yao, X. Fu, and C. Zong, "Short-term load forecasting method based on feature preference strategy and LightGBM-XGBoost," *IEEE Access.*, vol. 10, no. June, pp. 75257–75268, 2022, doi: https://doi.org/10.1109/ACCESS.2022.3192011.

[13] A. Faustine, N. J. Nunes, and L. Pereira, "Efficiency through simplicity: MLP-based approach for net-load forecasting with uncertainty estimates in low-voltage distribution networks," *IEEE Trans. Power Syst.,* vol. 40, no. 1, pp. 46–56, 2024, doi: https://doi.org/10.1109/TPWRS.2024.3400123.

[14] A. O. Aseeri, "Effective RNN-based forecasting methodology design for improving short-term power load forecasts: Application to large-scale power-grid time series," *J. Comput. Sci.*, vol. 68, no. February, p. 101984, 2023, doi: https://doi.org/10.1016/j.jocs.2023.101984.

[15] K. Zhu, Y. Li, W. Mao, F. Li, and J. Yan, "LSTM enhanced by dual-attention-based encoder-decoder for daily peak load forecasting," *Electr. Power Syst. Res.,* vol. 208, no. February, p. 107860, 2022, doi: https://doi.org/10.1016/j.epsr.2022.107860.

[16] Z. Liang, R. Mieth, and Y. Dvorkin, "Operation-adversarial scenario generation," *Electr. Power Syst. Res.*, vol. 212, no. October 2021, p. 108451, 2022, doi: https://doi.org/10.1016/j.epsr.2022.108451.

[17] C. Wang, X. Li, Y. Shi, W. Jiang, Q. Song, and X. Li, "Load forecasting method based on CNN and extended LSTM," *Energy Rep.,* vol. 12, no. August, pp. 2452–2461, 2024, doi: https://doi.org/10.1016/j.egyr.2024.07.030.

[18] T. H. Bao Huy, D. N. Vo, K. P. Nguyen, V. Q. Huynh, M. Q. Huynh, and K. H. Truong, "Short-term load forecasting in power system using CNN-LSTM neural network," in 2023 *IEEE Asia Meet. Environ. Electr. Eng. (EEE-AM)*, 2023, pp. 1–6, doi: https://doi.org/10.1109/EEE-AM58328.2023.10395221.

[19] L. Li, X. Su, X. Bi, Y. Lu, and X. Sun, "A novel Transformer-based network forecasting method for building cooling loads," *Energy Build.,* vol. 296, no. July, p. 113409, 2023, doi: https://doi.org/10.1016/j.enbuild.2023.113409.

[20] Z. Tian, W. Liu, W. Jiang, and C. Wu, "CNNs-Transformer based day-ahead probabilistic load forecasting for weekends with limited data availability," *Energy.*, vol. 293, no. February, p. 130666, 2024, doi: https://doi.org/10.1016/j.energy.2024.130666.

[21] T. T. Ngoc, L. Van Dai, and C. M. Thuyen, "Support vector regression based on grid search method of hyperparameters for load forecasting," *Acta Polytech. Hung.*, vol. 18, no. 2, pp. 143–158, 2021, doi: https://doi.org/10.12700/APH.18.2.2021.2.8.

[22] T. N. Tran, "A new grid search algorithm based on median values for SVR model in case of load forecasting," *Period. Polytech. Electr. Eng. Comput. Sci.*, vol. 67, no. 1, pp. 51–60, 2023, doi: https://doi.org/10.3311/PPee.20887.