World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(REVIEW ARTICLE)

How machine learning can revolutionize building comfort: Accessing the impact of occupancy prediction models on HVAC control system

Sheriff Adefolarin Adepoju *

Department of Computer Science, College of Engineering, Prairie View A and M University, Texas, United States.

World Journal of Advanced Research and Reviews, 2025, 25(01), 2315-2327

Publication history: Received on 07 December 2024; revised on 19 January 2025; accepted on 22 january 2025

Article DOI: https://doi.org/10.30574/wjarr.2025.25.1.0161

Abstract

Ventilation control systems globally constitute among the most significant energy demands in the building industry. Optimizing the energy usage of such systems is imperative for constructing sustainable buildings and is essential for achieving environmental sustainability. Occupancy factors of these buildings, particularly Heating, Ventilation and Air Conditioning (HVAC) optimization, are pivotal for energy optimization. In this work, we apply machine learning approaches to improve the efficiency of the HVAC systems of the Engineering Classroom and Research Building (ENCARB) at Prairie View A&M University. We focus on supporting real-time automated HVAC control through the accurate estimation of HVAC temperature based on occupancy patterns. Therefore, we introduce AIRFLO, an AI-powered Robust Framework for Learning to Optimize HVAC energy consumption. Our framework integrates several occupancy factors obtained from the class schedule to estimate ideal HVAC temperatures. Specifically, we incorporate user nonspecific behavior vis-a-vis energy HVAC service period within the ENCARB Building to gather useful matrices as the basis for the model design. To learn the complex patterns in data, we trained our framework using supervised machine learning. Specifically, we initially trained our framework using an ensemble of multilayer neural networks using our training data and observed the estimation performance on an independent validation set. To learn deeper representations and perform systematic comparative model analysis, we proposed our plan to incorporate both Convolutional Neural Networks (CNNs) and Graph Neural Networks (GNNs). Overall, our proposed approach will offer a comprehensive and scalable framework for optimizing HVAC energy optimization, leading to improved sustainability.

Keywords: Time series analysis; Machine learning; Adaptive HVAC control; Occupancy sensing; Supervised learning

1. Introduction

To solve complex problems, neural networks are systems composed of neurons and basic processing units with multi-input and single-output structures that have demonstrated promising effective use (G. Wang et al., 2024). These processing units are controlled by mathematical procedures that manage the inputs and outputs of the system to optimize the network performance by adjusting data transmission strategies to enhance training rate and network efficiency (Su et al., 2023). Neural network processing efficiently solves complex problems across domains such as pattern recognition, image processing, and computer vision (Mo & Xiang, 2024). However, it is with the increased scale and complexity of neural network models, including higher resource consumption for training more extensive networks. Techniques like model compression through pruning and quantization are solutions to reduce resource requests while keeping good performance.

This project explores the development of neural network models for predicting temperature settings, specifically for an HVAC system. The system seeks to continuously transform historical data to a format suitable for neural network processing. There are different types of neural networks available to be used. Some are Feedforward, perceptron,

* Corresponding author: Sheriff Adefolarin Adepoju

multilayer, recurrent, Ling short-term memory (LSTM), Convolutional neural network, sequence-to-sequence models, modular neural network, graph neural network, and many more (Gou et al., 2024). However, for this research, we shall be focused on convolutional and graph neural networks. ENCARB can save energy by integrating predictive modeling with the present HVAC system. By autonomously adjusting the temperature of a room to suit the inhabitants, thermal comfort and user satisfaction can be achieved. This project substantiates the possibilities of fusing advancements in machine learning and IoT devices to improve building management.

The aim is to offer a comprehensive and scalable framework for optimizing HVAC energy optimization, leading to improved sustainability. The primary goal was to develop a CNN of GNN model capable of predicting temperature based on various inputs to build a robust model for accurate temperature prediction in a building, contributing valuable insights into the applicability of CNN or GNN in time-series forecasting. The sophistication and intricacy of the neural network model's architecture, the algorithms used for training, and the overall system setup required to support the model's operation are significant aspects to consider. Neural networks opens up the likelihood of complex models often seen as "black boxes" because understanding the relation-ships they learn can be challenging, leading to issues with trust and acceptance (Rubio et al., 2024).

Quality of data encompasses balance and accuracy. Poor data quality can lead to biased predictions, perpetuating and amplifying existing prejudices in sensitive applications such as hiring, lending, and law enforcement. On the other hand, high-quality but nontypical data can lead to overfitting, making the model perform well on training data but could be better on unseen data. Privacy concerns can prevent data sharing across departments of the same school, reducing the ability of models to learn from diverse datasets. In mitigating this challenge, synthesized data can be employed but may not capture the complexities present in real datasets. Testing neural networks, especially deep learning models, requires significant computational power, leading to the need for expensive hardware. Proper testing, like hyperparameter tuning and cross-validation, can take a lot of time. Using machine learning models to predict temperature regulation in a building offers many advantages over manual computations or other methods, significantly in-creasing adaptability, efficiency, and effectiveness (Esra-filian-Najafabadi & Haghighat, 2022). An exponential in-crease of parameters will take longer to compile, leading to errors and inefficiencies manually.

Machine learning can automate temperature adjustments based on diverse parameters such as time, occupancy, and external weather, thus reducing manual oversight and optimizing operational efficiency. These models dynamically adapt to changes, improving comfort and energy efficiency by learning from historical data to predict optimal settings that balance comfort with minimal energy use. Additionally, machine learning can detect anomalies suggesting necessary maintenance before issues escalate, which minimizes downtime and maintenance costs. Moreover, these models personalize climate conditions within buildings, catering to individual occupants' preferences and increasing satisfaction and productivity. They are also scalable, capable of managing multiple buildings simultaneously, and work seamlessly with IoT devices, which collect real-time data to inform temperature adjustments. Initially, setting up a machine-learning system may require some investment, but the long-term savings on energy and maintenance costs make it cost-effective. Inte-grating machine learning into building temperature control systems marks a significant step towards more responsive, intelligent, and sustainable building management.

2. Material and methods

A detailed methodological framework is necessary for a study using machine learning (ML) to evaluate how occupancy levels affect room temperature and the cooling system's ability to regulate these effects. This framework will encompass data collection, preprocessing, feature engineering, model selection, training, validation, and analysis. The method adopted in this project to develop a Convolutional Neural Network (CNN) model for temperature pre-diction encompasses several key stages, from initial data preprocessing through to model evaluation and refinement. This systematic approach ensures the model is both accurate and robust, capable of generalizing well to new, unseen data.

2.1. Dataset

We collected data from the ENCARB build over a period of one month. The data we collected was not sufficient then we employed a data synthesis approach to compile more dataset for our study of temperature settings across different parameters. Traditional data collection methods using heat sensors were utilized and due to the limited availability of comprehensive student behavior data spans multiple months and classes could not be collected. More so, privacy concerns about using accurate student data necessitated an approach that could be generated from the limited data we had to generate realistic yet non-identifiable datasets.

2.2. Algorithm

We used a statistical model to synthesize and generate data based on the historical measurements we had gathered. Statistical synthesis is particularly suited for this task because it generates high-quality, realistic data from limited input samples. Using a statistical model is based on its successful application in similar studies where data authenticity and variability are critical.

2.2.1. Feature Generation

We try to understand the available raw data. This generation includes identifying relevant data, understanding the data types (numerical, categorical, timestamp), and cognizing the potential relationships between various data elements.

2.2.2. Temporal Features

Since CNNs are typically used for spatial data, incorporating time can be challenging. Our approach uses temporal features such as the time slot of the day, day of the week, and month as categorical inputs, which are one-hot encoded to preserve their discrete nature.

2.2.3. Windowing

We use a sliding window approach to capture temporal dependencies in temperature changes. For instance, creating input features from temperature data over the past time slot, day, or week. This method involves generating a series of overlapping windows across the data that allow a model to learn from sequences of inputs to predict future values.

- Steps
  - Features (Window of Inputs) - The model places a fixed number of consecutive data points as input features and utilizes them to predict the output at a future point.
  - Lagged Features - The model uses past values (lags) as input to predict the value at a future time step. This is akin to shifting the dataset to several steps and using these past values to predict future outcomes.
  - Window and Lag Sizes - The window size determines the number of past observations that predict the following observation. The lag size specifies how far ahead the model should predict. Window Size – 29 and Lag Size – 30.
  - We implemented the window approach using a function (series_to_supervised). We designed the function to re-structure the dataset into a format suitable for supervised learning, where each row contains the input features (past observations) and the output (future observation) that the model needs to predict.
  - The model used a lag size of 30 to predict the outcome of 30 steps ahead based on the input sequence of the past 29 steps. This configuration of using a window of 29 to predict 30 steps into the future allows the model to capture sufficient historical context to make an informed prediction about the future state, which is essential in many time-series fore-casting applications like temperature prediction in the HVAC system.

2.2.4. Environmental Conditions

Normalize external weather conditions like temperature and humidity to ensure they are on a scale like other features. The time of number of students registered for a class or an event and date information is transformed into a cyclical feature to get patterns like day, seasonality, and time that affect the classroom temperature.

Table 1 List of features

S/N	Feature Description	Number of Features
1	Day - Numeric code representing different days of the week	5
2	Time Slot - Numeric code representing different time slots in a day	10
3	Season	3
4	Week	16
5	Total Students - Average number of students present during the specified class and time slot. (min-max normalization)	1
6	Classroom capacity – Total number of students a class can take. (min-max normalization)	1
	Total Features	36

Total Number of features: 36

Strategies such as imputation or removal of rows with missing values were employed to ensure the dataset's completeness, enhancing model reliability. Continuous variables like 'Total_Students' and 'Season_Temp' were normalized to have similar scales, facilitating more efficient learning by the CNN model. The dataset was enriched by calculating new features, such as deriving exact dates from week numbers and transforming these temporal variables into more informative features for the model.

Table 2 List of features

Training	Validation	Testing
65,488	16,372	81,860

We rigorously tested the model's ability to generalize to new data, as the test dataset is separate and of significant size, so we can better understand the bias-variance tradeoff in the model. A significant difference in performance between the validation and test sets may show us overfitting. The dataset test provided a more reliable estimate of the models' performance metrics, giving us a clearer picture of how the model might perform in real-world scenarios. The test dataset is representative of the same distribution as the training and validation sets to avoid biased performance estimates.

A critical step for a CNN model while dealing with time series data is the transforming of the dataset into a supervised learning format with defined input-output pairs. Past observations (lagged features) were constructed to serve as inputs for predicting future temperature values, effectively turning the dataset into a sequence prediction problem.

Average temperature as an additional aggregated metric was computed and merged back into the dataset to provide the model with a richer set of features for learning.

2.3. AIRFLO Model training

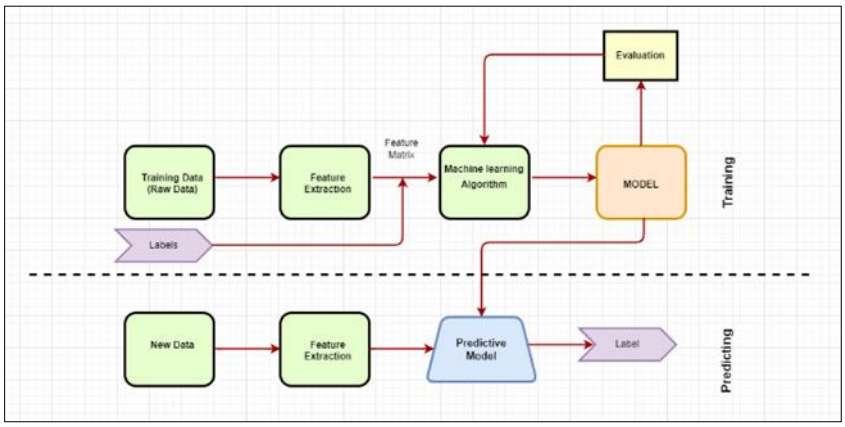

Figure 1 Data Flow Chart

2.3.1. Model A

One convolutional layer is used in the neural network model, using 64 filters. These filters help the model in learning various features from the data. The kernel size is set to 8, meaning that the convolution operation uses a window of 8 consecutive data points to calculate the output of the layer, which helps extract localized feature patterns in the input data sequence. The activation function used is 'relu' (Rectified Linear Unit), which introduces non-linearity into the model, allowing it to learn more complex patterns. The neural network model utilizes two dense layers. The first layer has 50 neurons and uses the ReLU (Rectified Linear Unit) activation function. It is a fully connected layer that can learn non-linear combinations of the features extracted by the previous layers, including the convolutional and max pooling layers. The network's second layer (output layer) has a single neuron, and since it does not specify an activation function, it defaults to a linear activation. This layer is typical for regression tasks where the model predicts a continuous value. A dropout layer is positioned between the two dense layers in the neural network model with a dropout rate of 20%, placed right after the first dense layer with 50 neurons and before the output dense layer with one neuron. This setup helps reduce the chance of overfitting by randomly turning off 20% of the neurons in the layer during training, which forces the network to develop more robust features that do not rely too heavily on any individual element of the

previous layer's output. The batch size is 512, the epoch is 900, and the Adam optimizer's learning rate (lr) is 0.0003. Mean squared error (MSE) is employed as the loss function. Implemented early stopping to halt training when the validation loss ceased to decrease, thereby preventing overtraining.

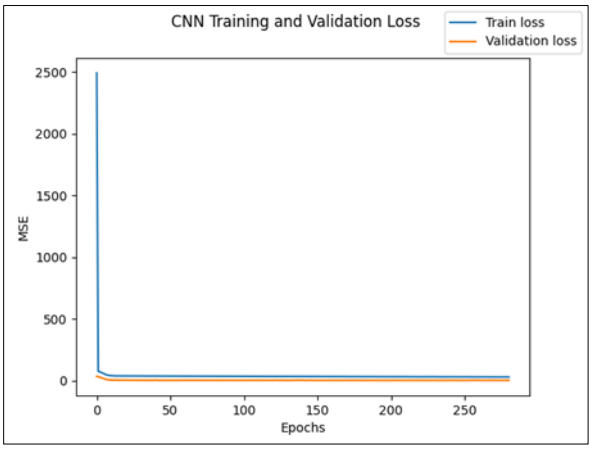

Figure 2 Training and Validation loss Graph for Model A

2.3.2. Model B

This model involves refining the architecture to capture the temporal dependencies better and applying more sophisticated techniques like regularization and learning rate schedules. The idea is to capture more complex patterns but at the risk of overfitting—a gradual decrease in learning rate during training to help the model fine-tune its weights more effectively. We retain L2 regularization to prevent overfitting by penalizing large weights. The dropout schedule may further regularize the model during training. The batch and epoch numbers remain the same. The optimizer is Adam, with a learning rate of 0.0001.

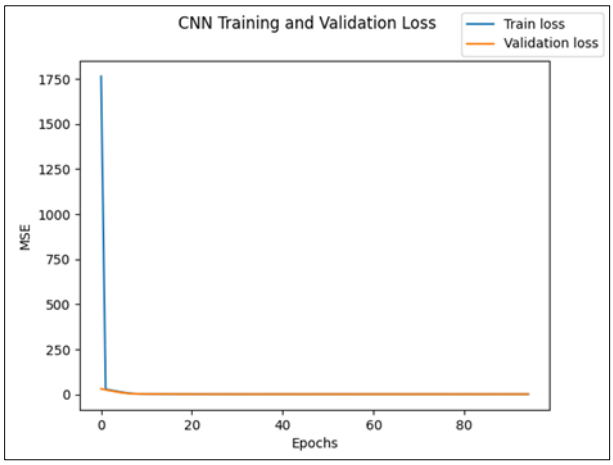

Figure 3 Training and Validation loss Graph for Model B

2.3.3. Model C

This model applies some changes and configurations to the neural network model. Two convolutional layers are used in the neural network model, using 64 filters and a kernel size 8. In contrast, the second layer uses 128 filters and a kernel size of 8, and both use ReLU activation. Following each convolutional layer are batch normalization and max pooling, which help normalize the input layer by re-centering, re-scaling, and reducing the spatial size of the representation, respectively. We use two dropout layers to reduce overfitting by randomly setting a fraction of the input units to zero during training. We position the first dropout (0.3) after the dense layer of 100 neurons and the second dropout (0.3) after the second dense layer of 50 neurons. The network consists of 3 dense layers of 100, 50, and 1, respectively. The output dense layer with a single neuron is for regression tasks. It has no activation specified, implying a linear activation. We have a batch size of 512, the number of Epoch remains 900, and the Adam Optimizer remains 0.0003.

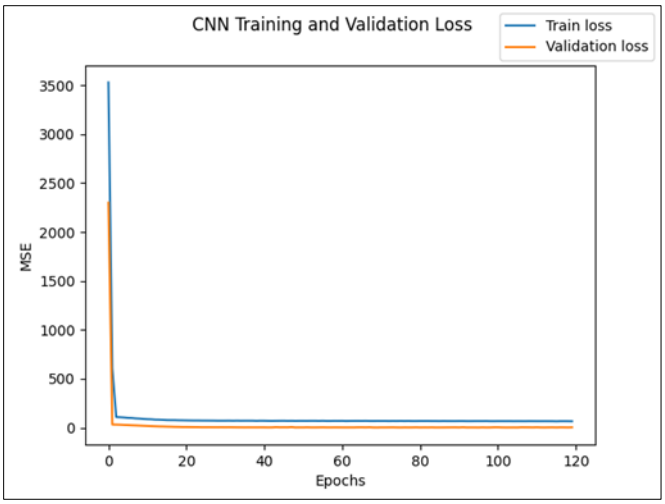

Figure 4 Training and Validation loss Graph for Model C

2.3.4. Model D

This model uses a neural net-work model combining convolutional and Long Short-Term Memory (LSTM) layers, which are defined and trained for a regression task. The model includes one convolutional layer using 64 filters with a kernel size of 3 and applies the ReLU activation function. The model contains one dropout layer (0.3) positioned after the first dense layer that randomly sets 30% of the input units to zero during training. This technique helps prevent overfitting by encouraging the network to learn more robust features that do not rely on a few neurons.

The model uses two dense layers (100, activation='relu'), with the first dense layer with 100 neurons, utilizing the ReLU activation function to introduce non-linearity into the network. The second dense layer, the output layer with a single neuron, is appropriate for regression tasks where the model predicts a continuous value. The layer does not specify an activation function, implying a linear activation that directly outputs the prediction. The dropout layer is placed between the dense layers, specifically between the first and output layers, providing regularization only after the complex non-linear transformation by the first dense layer. The batch size is 512, the number of epochs is 900, and the learning rate for the Adam optimizer is 0.0003. An early stopping mechanism is employed to monitor the validation loss (val_loss), stopping the training if the loss does not improve by at least 0.0001 for 50 consecutive epochs (patience=50).

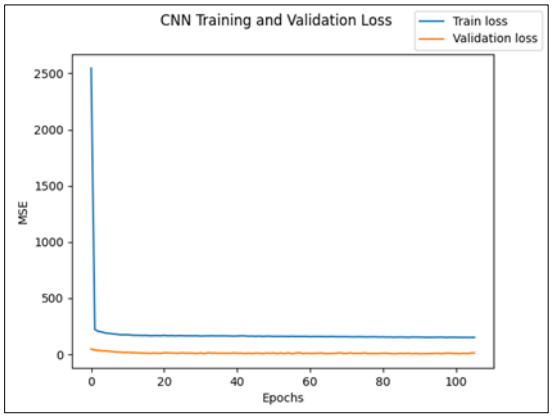

Figure 5 Training and Validation loss Graph for Model D

2.3.5. Model E

This model uses a neural network model that Utilizes 64 filters with a kernel size of 8, applied to the input shape derived from X_train_series. The First MaxPooling1D Layer reduces the spatial dimensions (i.e., the length of the time series data) by a factor of 2, which helps reduce computational complexity and controls overfitting. Like the first, the second conventional layer uses 64 filters with a kernel size of 8, further processing the data to extract more refined features. The Second MaxPooling1D Layer further reduces the data dimensionality, making the network invariant to small shifts and distortions in the input data. The Third Conv1D Layer in-creases the complexity of the network by using 128 filters.

The Third MaxPooling1D Layer continues to decrease the data dimensionality. The Fourth Conv1D Layer Continues with 128 filters. Fourth MaxPooling1D Layer Reduces the output of the previous convolutional layer to a minimal size, preparing the data for the final classification or regression step. The Flatten Converts the multi-dimensional output of the last pooling layer into a single long vector. First Dense Layer Comprising 50 neurons, this layer is used for further processing the features extracted by the convolutions. Dropout Layer: With a dropout rate of 0.2, this layer randomly sets input units to 0 at each step during training, which helps prevent overfitting. The Output Dense Layer Contains a single neuron. Since no activation function is specified, it defaults to a linear activation, which is typical for regression tasks. The batch size is 512, the number of epochs is 900, and the learning rate for the Adam optimizer is 0.0003. An early stopping mechanism is employed to monitor the validation loss (val_loss), stopping the training if the loss does not improve by at least 0.0001 for 50 consecutive epochs (patience=50).

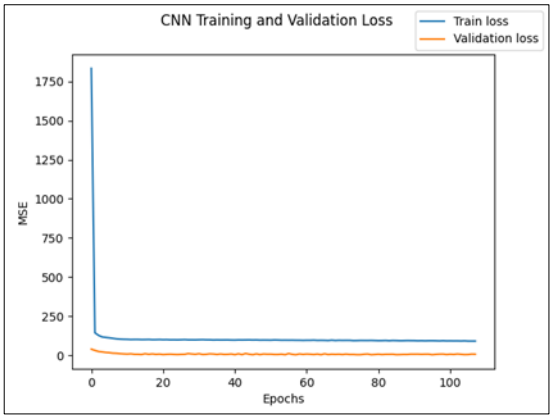

Figure 6 Training and Validation loss Graph for Model E

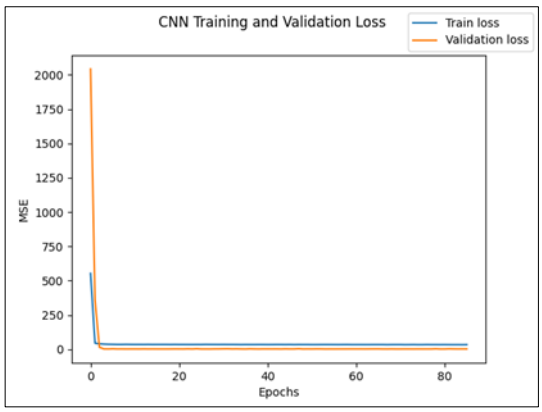

Figure 7 Training and Validation loss Graph for Model F

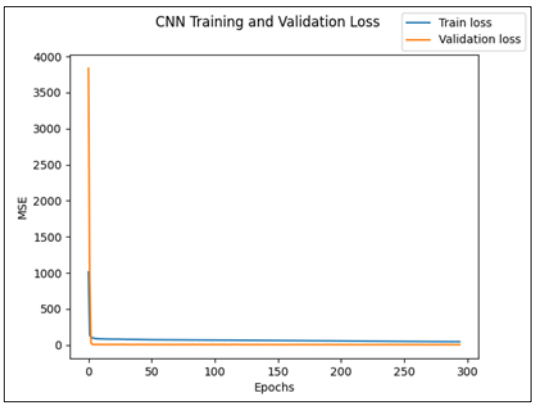

Figure 8 Training and Validation loss Graph, Model G

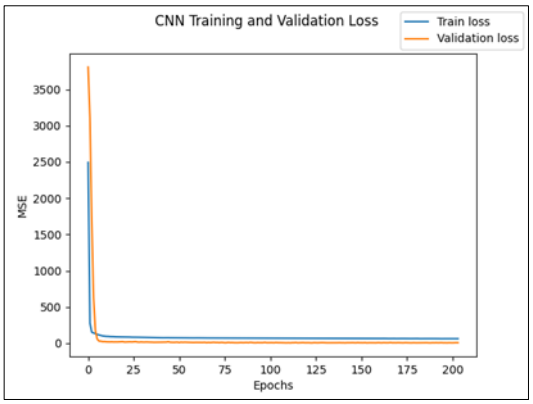

Figure 9 Training and Validation loss Graph for Model E

Table 3 Models and their layers

Model Name	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Layer 6	Layer 7	Layer 8	Layer 9	Layer 10
Model A	Conv2D(filters=64, kernel_size=(2, 1), activation='relu', input_shape=(num_features, 1, 1))	Conv2D(filters=32, kernel_size=(2, 1), activation='relu')	MaxPooling2D(pool_size=(2, 1))	Flatten()	Dense(50, activation='relu')	Dropout(0.2)	Dense(1)			

Model B	Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(38, 1))	Conv1D(filters=32, kernel_size=2, activation='relu')	Flatten()	Dense(64, activation='relu')	Dense(32, activation='relu')	Dense(1)				
Model C	Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(features.shape[1], 1))	Conv1D(filters=32, kernel_size=2, activation='relu')	MaxPooling1D(pool_size=2),	Flatten()	Dense(50, activation='relu')	Dropout(0.2)	Dense(1)			
Model D	Conv2D(filters=64, kernel_size=(2, 1), activation='relu', input_shape=(num_features, 1, 1))	Conv2D(filters=32, kernel_size=(2, 1), activation='relu')	MaxPooling2D(pool_size=(2, 1))	Flatten()	Dense(50, activation='relu')	Dropout(0.5)	Dense(1)			
Model E	Conv2D(filters=128, kernel_size=(3, 1), activation='relu', input_shape=(num_features, 1, 1))	Conv2D(filters=64, kernel_size=(3, 1), activation='relu')	MaxPooling2D(pool_size=(2, 1))	Conv2D(filters=32, kernel_size=(3, 1), activation='relu')	MaxPooling2D(pool_size=(2, 1))	Conv2D(filters=16, kernel_size=(3, 1), activation='relu')	Flatten()	Dense(units=100, activation='relu')	Dropout(rate=0.5)	Dense(units=1)
Model F	Conv1D(filters=128, kernel_size=3, activation='relu', input_shape=(features.shape[1], 1))	BatchNormalization()	Conv1D(filters=64, kernel_size=3, activation='relu')	MaxPooling1D(pool_size=2)	Flatten()	Dense(100, activation='relu')	Dropout (0.3)	Dense (1)		
Model G	Conv1D(filters=128, kernel_size=3, activation='relu', input_shape=(features.shape[1], 1))	BatchNormalization()	MaxPooling1D(pool_size=2)	Conv1D(filters=64, kernel_size=3, activation='relu')	BatchNormalization()	MaxPooling1D(pool_size=2)	Flatten()	Dense(100, activation='relu', kernel_regularizer='l2')	Dropout(0.3)	Dense(50, activation='relu', kernel_regularizer='l2')
Model H	Conv2D(filters=32, kernel_size=(		MaxPooling2D(pool	Dropout(0.1)	Conv2D(filters=64,		MaxPooling2D(pool		Flatten(),	Dense(32, activati

	2, 1), activation='relu', input_shape=(num_features, 1, 1))	BatchNormalization()	_size=(2, 1))		kernel_size=(2, 1), activation='relu')	BatchNormalization()	_size=(2, 1))	Dropout(0.1)		on='relu'),
										Dropout(0.2),
									Dense(64, activation='relu'),	
									Dropout(0.2)	Dense(1

Table 4 Models and their RMSE Score

	Total Params	**Trainable Params**	**Non-trainable Params**	**Train (RMSE)**	**Validate (RMSE)**
Model A	33,221	33,221	0	1.5392241477966309	1.5359832048416138
Model B	80,225	80,225	0	1.500288486480713	1.499264121055603
Model C	21,661	21,661	0	1.573879361152649	1.5726734399795532
Model D	33,221	33,221	0	1.7749559879302979	1.7689486742019653
Model E	41,081	41,081	0	1.8417853116989136	1.854941487312317
Model F	134,665	134,409	256	1.5205743312835693	1.5218079090118408
Model G	82,321	81,937	384	1.548300862312317	1.5427883863449097
Model H	67,505	67,313	192	2.377387762069702	2.3820128440856934

Models F through H were also tested with varying degrees of settings for each of their layers as shown in the table above. We assessed the models' accuracy and generalization capability by calculating the root mean squared error (RMSE) and mean absolute error (MAE) on both the training and validation sets. We generated training and validation loss plots over epochs to gain insights into the learning process and highlight any overfitting or underfitting issues. We will compare the root mean square error (RMSE) values for training and validation sets to determine which model has the best result. Recall that the RMSE is the difference between values predicted by a model and observed values. A lower RMSE value indicates a better fit of the model to the data.

Based on these metrics, Model B emerges as the most effective, exhibiting the lowest RMSE values for training (1.50) and validation (1.49). The narrow gap between these scores indicates that Model B generalizes exceptionally well, achieving a balanced performance with minimal evidence of overfitting or underfitting. Model F mirrors the performance of Model B, with a training RMSE of 1.50 and a validation RMSE of 1.521, suggesting it also generalizes well, albeit with a marginal increase in the training error compared to Model B.

Model A and Model G, with RMSE values of 1.5392 and 1.543 for training and 1.5395 1.5428 for validation respectively, show less accuracy in its predictions than Model B and F. However, the consistent RMSE between training and validation suggests that while not as accurate, Model A and `G generalizes as well as it learns.

Model C presents a training RMSE of 1.573 and a slightly lower validation RMSE of 1.572. Although these values are better than those of Model E, they indicate a decrease in prediction accuracy and a slight potential for overfitting compared to Models A and G.

Model D and Model H, with a training RMSE of 1.84 and 2.37 with validation RMSE of 1.85 and 2.38, exhibits the highest discrepancy in fitting the training data versus generalizing to the validation data among the models. The relatively high RMSE values suggest that Model D and H, still needs to perform more effectively regarding overall prediction accuracy.

Hence, based on the evaluation results, Model B is the best choice based on the provided RMSE scores, demonstrating the most accurate and generalizable performance.

Table 5 Models tested with Unseen Data and it's RMSE Score

	Total Params	**Trainable Params**	**Non-trainable Params**	**Train (RMSE)**	**Validate (RMSE)**
Model B	80,225	80,225	0		
Model F	134,665	134,409	256	1.7374496459960938	1.7520452737808228
Model H	67,505	67,313	192	2.562861204147339	2.589235782623291

The table above presents a comparison of three machine learning models (Model B, Model F, and Model H) for temperature prediction based on their Root Mean Squared Error (RMSE) scores when tested with unseen data. Model B maintains the best overall performance, having the lowest RMSE scores for both training and validation, despite having fewer parameters than Model F.

Model F performs moderately, with slightly higher RMSE scores compared to Model B but lower than Model H. It balances complexity with performance as shown during the initial training and validation tests.

Model H shows the highest RMSE scores for both training and validation, indicating it may be the least accurate. Its simpler structure may contribute to its lower performance. Model F can be considered if a slightly more complex model is preferred, but its performance gain over Model B might not justify the increase in complexity.

3. Results and discussion

The output received from the model, represented as {'predictions': [[50.320196151733334]]}, indicates that the model has processed the input data and generated a prediction. The result is provided in a JSON-like dictionary format with a key 'predictions,' pointing to a list containing another list. This nesting indicates that the model could potentially output multiple predictions at once, or it is formatted to handle batch predictions where each inner list represents the output for a separate input instance. The model is a regression model predicting continuous values; this output value (50.320196151733334) represents the predicted value for the given input.

3.1. Proposed Sample Architecture for CNN on an IoT Device

The materials and methods should be typed in Cambria with font size 10 and justify alignment. Author can select Normal style setting from Styles of this template. The simplest way is to replace (copy-paste) the content with your own material. Method and analysis which is performed in your research work should be written in this section. A simple strategy to follow is to use keywords from your title in first few sentences.

3.1.1. Processor

Choose an IoT device with a capable processor, such as an ARM Cortex-M series for simpler models or Cortex-A for more complex models. Raspberry Pi is well-suited for such a task.

The actual authors can be referred to, but the reference number(s) must always be given. e.g.: 'Barnaby and Jones [8] obtained a different....'

3.2. Hardware Considerations

- *Processor:* Choose an IoT device with a capable processor, such as an ARM Cortex-M series for simpler models or Cortex-A for more complex models. Raspberry Pi is well-suited for such a task.
- *Memory*: Adequate RAM and storage to handle the model and the data. Devices with at least 1GB of RAM are preferable for this model.
- *Power Source*: The system will be connected to the mains for an electrical source and have a battery as backup.

3.3. Software and Tools

- *Operating System:* Using a lightweight OS that can run Python, such as Raspbian for Raspberry Pi or a custom Linux distribution.
- *Frameworks:* TensorFlow Lite optimizes running machine learning models with limited mobile and embedded device resources.
- *Model Conversion Tools* - Use tools like TensorFlow Lite Converter to convert a TensorFlow model into a Tensor-Flow Lite model, which is more suitable for mobile and embedded devices.
- *CNN Model Design*: Design a simpler CNN with fewer layers and parameters that can still achieve the necessary accuracy for the task.
- *Model Training and Conversion:* Train the model on a more powerful machine/cloud infrastructure. Convert the model to TensorFlow Lite format using the TensorFlow Lite Converter. This step optimizes the model for inference on low-power and re-source-constrained devices.
- *Deployment, Inference and Processing:* Deploy the TensorFlow Lite model onto the IoT device. Utilize the TensorFlow Lite interpreter to run the model. Ensure efficient data handling from sensors to the model. Implement the model to process data in real-time, ensuring it can handle the input data rate.
- *Monitoring and Updates*: Performance Monitoring: Continuously monitor the performance and stability of the model on the device. Updates: Allow for remote model updates to improve performance or fix issues.

4. Conclusion

The methodical approach adopted in this project, from initial data processing to model evaluation and refinement, embodies the iterative nature of machine learning projects. By carefully preparing the data, designing a suitable model architecture, and rigorously evaluating the model's performance, this project demonstrates the potential of CNNs in predicting temperatures from time-series data, extending their application beyond traditional image processing tasks. This personal observation serves as an anecdote and a re-al-world example of the need for effective thermal management in architectural and HVAC design. It highlights an everyday application of thermodynamic principles and human biology, reinforcing the importance of interdisciplinary approaches in creating comfortable, productive, and sustainable indoor environments.

GNN can be leveraged in related environmental modeling tasked to develop graph representations that accurately reflect the real-world relationships between variables influencing temperature and experimenting with different GNN architectures (such as Graph Convolutional Networks and graph Attention Networks) to identify the most effective models for specific types of prediction tasks. The exploration involving hybrid models that combine GNNs with other neural network architectures, such as RNNs or CNNs, to capture spatial, temporal, and graph-structured data attributes simultaneously can be employed to improve the output.

IoT devices can provide real-time, granular data on indoor temperatures, humidity levels, occupancy patterns, and HVAC system performance. This wealth of data can be used to train more accurate and responsive temperature prediction models. These devices can also be deployed through-out the ENCARB building to monitor various environ-mental factors, enabling models to understand and predict microclimate variations within different spaces.

Compliance with ethical standards

Disclosure of conflict of interest

There was no conflict of interest in the progression of this research.

References

[1] Babalhavaeji, A., Radmanesh, M., Jalili, M., & Gonzalez, S. A. (2023). Photovoltaic generation forecasting using convolutional and recurrent neural networks. *Energy Reports*, *9*, 119–123. https://doi.org/10.1016/j.egyr.2023.09.149

[2] Chen, J., Xi, X., Sheng, V. S., & Cui, Z. (2023). Randomly Wired Graph Neural Network for Chinese NER. *Expert Systems with Applications*, *227*, 120245. https://doi.org/10.1016/j.eswa.2023.120245

[3] Correa-Jullian, C., Cardemil, J. M., López Droguett, E., & Behzad, M. (2020). Assessment of Deep Learning techniques for Prognosis of solar thermal systems. *Renewable Energy*, *145*, 2178–2191. https://doi.org/10.1016/j.renene.2019.07.100

[4] Esrafilian-Najafabadi, M., & Haghighat, F. (2022). Impact of occupancy prediction models on building HVAC control system performance: Application of machine learning techniques. *Energy and Buildings*, *257*, 111808. https://doi.org/10.1016/j.enbuild.2021.111808

[5] Ghadimi, A., & Beigy, H. (2023). SGCSumm: An extractive multi-document summarization method based on pre-trained language model, submodularity, and graph convolutional neural networks. *Expert Systems with Applications*, *215*, 119308. https://doi.org/10.1016/j.eswa.2022.119308``

[6] Gou, Y., Fu, X., Zhao, S., He, P., Zhao, C., & Li, G. (2024). Improved convolutional neural network-assisted laser-induced breakdown spectroscopy for identification of soil contamination types. *Spectrochimica Acta Part B: Atomic Spectroscopy*, *215*, 106910. https://doi.org/10.1016/j.sab.2024.106910

[7] Haider, S. A., Sajid, M., Sajid, H., Uddin, E., & Ayaz, Y. (2022). Deep learning and statistical methods for short- and long-term solar irradiance forecasting for Islamabad. *Renewable Energy*, *198*, 51–60. https://doi.org/10.1016/j.renene.2022.07.136

[8] Jahedsaravani, A., Massinaei, M., & Zarie, M. (2023). Measurement of bubble size and froth velocity using convolutional neural networks. *Minerals Engineering*, *204*, 108400. https://doi.org/10.1016/j.mineng.2023.108400

[9] K. He, X. Zhang, S. Ren and J. Sun. (n.d.). *Deep Residual Learning for Image Recognition*.

[10] Lang, L., Tavadze, P., Prusinowski, M., Andrews, Z., Neumann, C., Trejos, T., & Romero, A. H. (2023). Using convolutional neural networks to support examiners in duct tape physical fit comparisons. *Forensic Science International*, *353*, 111884. https://doi.org/10.1016/j.forsciint.2023.111884

[11] Mo, Z., & Xiang, W. (2024). Maximum output discrepancy computation for convolutional neural network compression. *Information Sciences*, *665*, 120367. https://doi.org/10.1016/j.ins.2024.120367

[12] Nehzati, M. (2024). Integrating Convolutional Neural Networks for Improved Software Engineering: A Collaborative and Unbalanced Data Perspective. *Memories - Materials, Devices, Circuits and Systems*, 100106. https://doi.org/10.1016/j.memori.2024.100106

[13] Peng, Y., Rysanek, A., Nagy, Z., & Schlüter, A. (2018). Using machine learning techniques for occupancy-prediction-based cooling control in office buildings. *Applied Energy*, *211*, 1343–1358. https://doi.org/10.1016/j.apenergy.2017.12.002

[14] Rubio, J. D. J., Garcia, D., Rosas, F. J., Hernandez, M. A., Pacheco, J., & Zacarias, A. (2024). Stable convolutional neural network for economy applications. *Engineering Applications of Artificial Intelligence*, *132*, 107998. https://doi.org/10.1016/j.engappai.2024.107998

[15] Şengül, I., Candar, E., Demirçubuk, I., & Şengül, G. (2023). NEUROCHEMICAL PROPERTIES OF HUMAN SPINAL SOMATIC MOTOR NEURONS. *IBRO Neuroscience Reports*, *15*, S715. https://doi.org/10.1016/j.ibneur.2023.08.1452

[16] Shboul, B., AL-Arfi, I., Michailos, S., Ingham, D., Ma, L., Hughes, K. J., & Pourkashanian, M. (2021). A new ANN model for hourly solar radiation and wind speed prediction: A case study over the north & south of the Arabian Peninsula. *Sustainable Energy Technologies and Assessments*, *46*, 101248. https://doi.org/10.1016/j.seta.2021.101248

[17] Shoaei, M., Noorollahi, Y., Hajinezhad, A., & Moosavian, S. F. (2024). A review of the applications of artificial intelligence in renewable energy systems: An approach-based study. *Energy Conversion and Management*, *306*, 118207. https://doi.org/10.1016/j.enconman.2024.118207

[18] Su, D., Chen, L., Du, X., Liu, M., & Jin, L. (2023). Constructing convolutional neural network by utilizing nematode connectome: A brain-inspired method. *Applied Soft Computing*, *149*, 110992. https://doi.org/10.1016/j.asoc.2023.110992

[19] Wang, G., Hu, J., Zhang, Y., Xiao, Z., Huang, M., He, Z., Chen, J., & Bai, Z. (2024). A modified U-Net convolutional neural network for segmenting periprostatic adipose tissue based on contour feature learning. *Heliyon*, *10*(3), e25030. https://doi.org/10.1016/j.heliyon.2024.e25030

[20] Wang, X., Luo, Z., He, R., & Shao, Y. (2023). Novel medical question and answer system: Graph convolutional neural network based with knowledge graph optimization. *Expert Systems with Applications*, *227*, 120211. https://doi.org/10.1016/j.eswa.2023.120211

[21] Xu, X., Shi, X., & Shang, M. (2023). Graph neural networks via contrast between separation and aggregation for self and neighborhood. *Expert Systems with Applications*, *224*, 119994. https://doi.org/10.1016/j.eswa.2023.119994

[22] Zompola, A., Korfiati, A., Theofilatos, K., & Mavroudi, S. (2023). Omics-CNN: A comprehensive pipeline for predictive analytics in quantitative omics using one-dimensional convolutional neural networks. *Heliyon*, *9*(11), e21165. https://doi.org/10.1016/j.heliyon.2023.e21165

[23] Zunair, H., & Ben Hamza, A. (2021). Sharp U-Net: Depthwise convolutional network for biomedical image segmentation. *Computers in Biology and Medicine*, *136*, 104699. https://doi.org/10.1016/j.compbiomed.2021.104699