Advancing Predictive Modeling in Conventional Solar Stills: A Deep Learning Approach Leveraging Data Augmentation and Convolutional Neural Networks.

Hashim H. Migaybil [a,b], Bhushan Gopaluni [b]

[a] Department of Chemical and Materials Engineering, Faculty of Engineering, King Abdulaziz University, Jeddah 21589, Saudi Arabia

[b] Department of Chemical and Biological Engineering, Faculty of Applied Science, The University of British Columbia, Vancouver, BC V6T1Z3, Canada

[a] Corresponding author, Hashim H. Migaybil (e-mail: hmigaybil@kau.edu.sa).

Supplementary Materials: Foundational Principles of 1D Convolutional Neural Networks and Support Vector Regression for Sequential Numerical Data Analysis in Regression Tasks

1. Introduction

This supplementary document presents a clear and comprehensive overview of the fundamental principles underlying one-dimensional Convolutional Neural Networks (CNN-1D) and Support Vector Regression (SVR). The chosen input variables—daily solar radiation (Is), ambient temperature (T_{amb}), wind speed (V_w), relative humidity (RH), and saline water feed flow rate (P_{st}), have been identified as critical factors influencing the performance of solar stills. This highlights their importance in predictive modeling for a continuous numerical target variable, specifically daily freshwater productivity (P_{std}, L/day). In addition to discussing CNN-1D architectures, the document introduces both baseline and augmented SVR models, highlighting their methodological construction and theoretical foundations. The aim is to equip readers with a robust conceptual and mathematical understanding of these models, covering essential formulations, kernel functions, loss objectives, and optimization procedures. Special emphasis is placed on applying these models to sequential or vector-based numerical datasets derived from operational measurements of solar stills. This framework is intended to inform model selection, feature engineering, and data-driven performance tuning in regression-based forecasting applications.

2. CNN-1D Conceptual Framework for Regression

One-dimensional Convolutional Neural Networks (CNN-1D) are a specialized subset of deep neural networks derived from traditional multi-dimensional Convolutional Neural Network (CNN) architectures. They are specifically designed for the automated extraction and hierarchical learning of patterns from one-dimensional input data, such as time series and sequential datasets, to enable continuous numerical predictions in regression tasks. A defining characteristic of CNN-1D is its utilization of learnable convolutional filters (or kernels) that traverse the temporal dimension of the input. Each convolutional layer performs a linear transformation followed by a non-linear activation function, which increases the model's expressiveness. This operation enables the detection of localized features and recurring patterns across different temporal scales, which are essential for effectively modeling and forecasting the associated continuous target variables [1] [2].

2.1 Core Operations and Mathematical Formulation

In the context of time-series regression utilizing CNN-1D models, each sample in the input dataset is structured as a three-dimensional matrix with the dimensions $[n, k, f]$, where n represents the number of sequences, k denotes the look-back window (number of time steps), and f indicates the number of input features. As shown in Figure 1, the architecture of the CNN-1D model for regression tasks consists of an input layer, several convolutional layers, a flattening operation, and one or more fully connected (dense) layers, often enhanced with dropout for regularization, followed by a final output layer. This model is designed to automatically identify local temporal patterns in sequential numerical inputs by utilizing learnable filters across time steps. The convolutional layers convert raw input features into hierarchical feature maps, which are subsequently flattened and fed into dense layers to capture global dependencies. This architecture enables effective learning from time-dependent data, such as multivariate sensor signals or environmental measurements, allowing for efficient learning from complex data sets. The diagram elucidates the foundational computational workflow and structural design of the CNN-1D models utilized in this study. It is included in the supplementary materials for conceptual clarity and was not derived from the empirical results presented in the main manuscript.

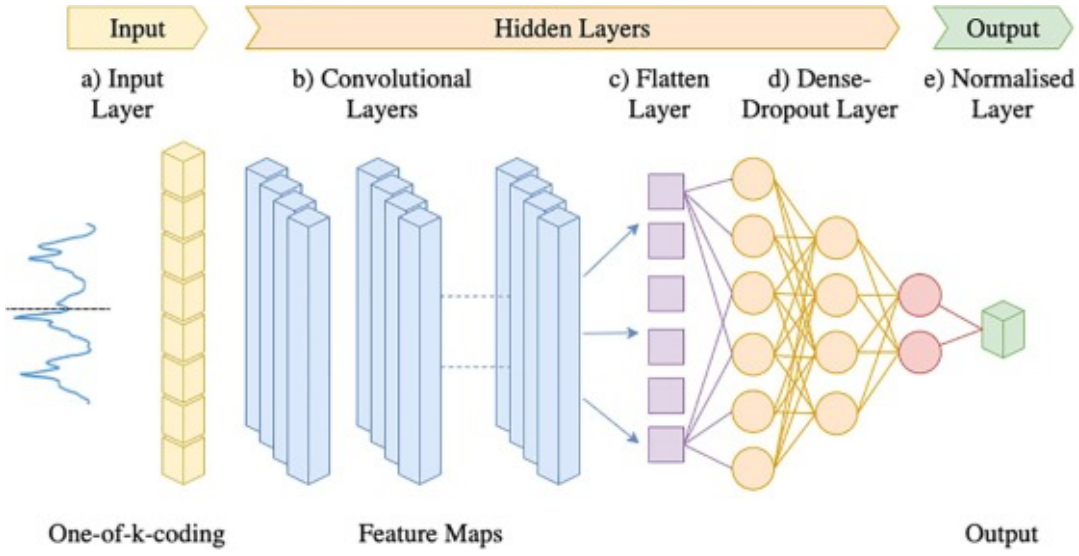

Figure 1: CNN-1D architecture. Structural layout of a one-dimensional convolutional neural network adapted for time-series regression, adopted from [3]. The pipeline includes input, convolutional, flattening, dense, and output layers, highlighting its ability to extract temporal patterns while limiting overfitting.

The primary computational operation involved is 1D convolution, which can be mathematically defined as:

$$Z^{(l)}[i,j] = \sum_{k=0}^{K-1} w^{(l)}[k,j] \cdot A^{(l-1)}[i+k] + b^{(l)}[j] \quad (1)$$

Where $w^{(l)}$ and $b^{(l)}$ denotes the weights and bias for the l^{th} layer, and K is the kernel size. With a stride $S = 1$ and padding set to 'same', the output retains the input length $L_{out} = L_{in}$, achieved by

applying $P = \left[\frac{K-1}{2}\right]$ Padding on each side. The Rectified Linear Unit (ReLU) activation function, defined as $f(x) = \max(0, x)$, is used to introduce non-linearity [4] [5]. Mathematically, the input to the first layer is represented as $A^{(0)} = x$, where x denotes the input sequence matrix. The linear output of the l^{th} convolutional layer is expressed as $Z^{(l)}[i, j]$. This output is then passed through the ReLU activation function to create the activation map $A^{(l)}[i, j]$ defined by:

$$A^{(l)}[i, j] = \max\left(0\ ,\ Z^{(l)}[i, j]\right) \text{ and } A^{(0)} = x \quad (2)$$

This piecewise non-linearity enhances the network's capacity to recognize intricate patterns by selectively activating relevant features while preserving the flow of gradients. The ReLU function is widely favored due to its computational efficiency and its effectiveness in addressing vanishing gradient issues, making it a vital component in the hierarchical representation learning process of CNN-based regression models. Following the final convolutional layer, a flatten operation is applied to transform the multi-dimensional output tensor $\left(A^{(l)}\right) \epsilon\ \mathbb{R}^M$ into a one-dimensional vector $f = Flatten\left(A^{(l)}\right) \epsilon\ \mathbb{R}^M\ , where\ M = T\ \times\ m$, and T denotes the temporal dimension. This flattened representation is then passed to a fully connected dense layer that performs the non-linear transformation using the ReLU activation function, defined as [6]:

$$h_m = \max\left(0\ , \sum_{i=1}^{M} W_{mn}\ .\ f_n +\ b_m\right),\ \ for\ \ m = 1, \ldots, M \quad (3)$$

Where:

- h_m: Output from the m-th neuron in the dense hidden layer after applying the ReLU activation function.
- $\max(0, .)$: ReLU activation function, which returns the input if positive, and zero otherwise.
- W_{mn}: Weight connecting the n-th feature from the flattened layer to the m -th neuron in the dense hidden layer.
- f_n: n-th input value from the flattened layer (i.e., output from the previous convolutional layer).
- b_m: Bias term associated with the m-th neuron in the dense hidden layer.
- M: Total number of neurons in the dense hidden layer (e.g., 128).

This operation enables hierarchical feature abstraction and interaction across all time steps and channels. Finally, the model outputs the scalar prediction $\hat{y} \epsilon\ \mathbb{R}^M$, corresponding to each input sequence, through a linear output layer defined as [7]:

$$\hat{y} = \sum_{i=1}^{M} w_m^{(0)}\ .\ h_m + b^{(0)} \quad (4)$$

- $\hat{y}$: Predicted scalar output value from the network (i.e., estimated water productivity).
- $\mathrm{w}_\mathrm{m}^{(0)}$: Weight connecting the m-th neuron in the dense hidden layer to the output layer.
- $\mathrm{b}^{(0)}$: Bias term in the output layer.

The linear output layer is optimized during training to regress the continuous target variable associated with solar still freshwater productivity. The model's output is a single continuous variable per sequence, optimized to minimize a regression loss function, typically Mean Squared Error (MSE).

2.2 Parameter Optimization for Regression

Hyperparameter optimization for the CNN-1D regression models was conducted using the Hyperopt library, employing the Tree-structured Parzen Estimator (TPE) as the search algorithm. The tuned hyperparameters included the number of convolutional layers, filter size, kernel size, batch size, learning rate, batch size, number of dense units, regularization strength (L2), and number of training epochs. These parameters were sampled from well-defined discrete or continuous search spaces, depending on their characteristics (e.g., the learning rate is continuous, and the number of layers is discrete). The optimization process focused on minimizing the validation loss, typically measured by the Mean Squared Error (MSE), to enhance the model's generalization capabilities. The batch size determines the number of training samples used to estimate the gradient in each iteration, playing a crucial role in the stability and speed of the training process. The number of epochs indicates how many times the entire training dataset is processed by the network, which directly influences the model's convergence.

To mitigate overfitting, an early stopping criterion with a patience parameter was implemented, which halted training when the validation loss failed to improve over a predetermined number of consecutive epochs. The trainable parameters of the CNN-1D model, including convolutional kernels, biases, and dense layer weights, were updated iteratively using gradient-based optimization algorithms, such as the Adam optimizer, which is an adaptive variant of stochastic gradient descent (SGD). The fundamental update rule for the model parameters θ (weights and biases) follows the form:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t) \quad (5)$$

Where:

- θ_t denotes the parameter vector at iteration t
- η is the learning rate
- $\nabla_\theta \mathcal{L}$ is the gradient loss function $\mathcal{L}$ with respect to the parameters.

This process is applied to minimize a predefined loss function, such as the Mean Squared Error (MSE) or Mean Absolute Error (MAE), calculated over the training set, while also evaluating the same loss on the validation set to assess generalization performance [8] [9]. This comprehensive framework facilitated a systematic exploration of architecture-performance trade-offs (space), resulting in reproducible and robust model configurations [10] [11].

3. Support Vector Regression (SVR) Conceptual Framework

Support Vector Regression (SVR) is a robust machine learning algorithm that adapts the principles of Support Vector Machines (SVM) for regression tasks involving continuous numerical outputs. Unlike traditional regression models that aim to minimize prediction errors across all data points, SVR introduces the concept of an ε-insensitive margin, within which errors are not penalized. The

algorithm seeks to identify a regression function that fits the data while maximizing the margin around it. Data points that lie outside this margin are referred to as support vectors, which play a crucial role in defining the regression solution. This design allows SVR to be resilient to outliers and effective in situations with limited data availability. SVR is particularly well-suited for non-linear regression problems, utilizing a kernel trick to implicitly transform the input space into a higher-dimensional feature space without requiring the explicit computation of the transformation. In this context, the Radial Basis Function (RBF) kernel was chosen for its ability to effectively model nonlinear relationships in time series and environmental datasets [12].

3.1 Core Operations and Mathematical Formulation

The primary goal of SVR is to estimate a continuous-valued function $f(x)$ such that the predicted output does not deviate from the actual target y_i by more than a threshold ε for each training data point x_i, while simultaneously maintaining the function as flat as possible. This formulation is optimized by solving a convex quadratic programming (QP) problem known as the dual optimization problem, with the prediction function defined as:

$$\hat{y} = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*)\, K(x_i, x) + b \quad (6)$$

Where:

- $\hat{y} \in \mathbb{R}$ is the predicted continuous value,
- α_i , α_i^* are the dual Lagrange multipliers,
- $K(x_i, x)$ is the kernel function that measures similarity between input vectors,
- b is the model bias, and
- N is the total number of training instances.

To capture non-linear relationships, SVR employs the RBF kernel, which maps input vectors into a high-dimensional feature space without explicit transformation:

$$K(x_i, x) = exp(-\gamma \, \|x_i - x\|^2) \quad (7)$$

Here γ is the kernel coefficient that defines the influence radius of each data point in the transformed feature space. This "kernel trick" enables SVR to operate in a high-dimensional space without explicitly computing the mapping $\phi(x)$, allowing for efficient modeling of nonlinear regression surfaces.

The learning objective of SVR loss combines model complexity minimization, which is achieved through L2 regularization with prediction tolerance using an ε insensitive loss function, into a unified cost function. This dual objective ensures both flatness of the regression function and tolerance for small deviations within an ε margin:

$$\mathcal{L}_{\mathrm{SVR}} = \frac{1}{2} \, \|\mathrm{w}\|^2 + \mathrm{C} \sum_{\mathrm{i}=1}^{\mathrm{N}} \max(0, \left|y_{\mathrm{i}} - \hat{y}_{\mathrm{i}}\right| - \varepsilon \quad (8)$$

Where:

- w is the weight vector in the induced feature space,
- $\mathrm{C} > 0$ is the regularization parameter controlling the trade-off between model flatness and tolerance for deviations beyond ε.

- ε defines the width of the margin (tube) around the regression function within which errors are not penalized.
- y_i and $\hat{y}_i$ are the actual and predicted values, respectively.

Figure 2 visually represents the theoretical foundation of SVR. The left plot depicts a non-linear regression scenario in the original input space, where the model seeks to fit a function within a defined ε-insensitive margin. This margin allows for some tolerable deviations (ξ) from the actual values. The right plot shows the same problem transformed into a high-dimensional feature space using a kernel function $\phi(x)$, where the regression function becomes linear. In this high-dimensional space, SVR aims to identify a hyperplane that minimizes both prediction error and model complexity through regularization. This illustration clarifies how SVR effectively handles non-linear regression by utilizing kernel methods and an ε-insensitive loss function. Like Figure 1, this diagram is included in the supplementary material for explanatory purposes only and was not derived from the dataset or results used in the core analysis of the manuscript.

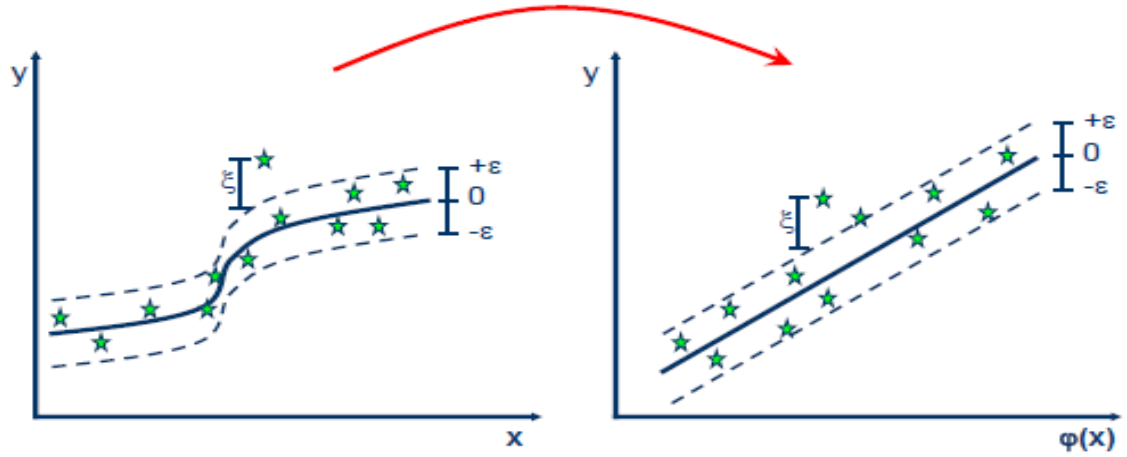

Figure 2: SVR framework. Schematic of Support Vector Regression with ε-insensitive loss, illustrating input space, kernel mapping (e.g., RBF), and regression function in feature space adopted from [13]. The ε-tube and slack variables are shown alongside corresponding equations (prediction, kernel, loss).

Slack variables that account for violations occurring outside the ϵ-insensitive tube are implicitly managed through the dual Lagrange multipliers α_i α_i^* in the SVR formulation, which streamlines the optimization process. Alternatively, the primal form explicitly captures these violations using slack variables ξ_i, ξ_i^* penalizes them in the objective function to ensure a balance between model complexity and prediction tolerance. Unlike neural networks, SVR does not rely on gradient descent or backpropagation for training. Instead, the solution is achieved through convex optimization solvers that guarantee global optimality within a well-defined dual quadratic programming (QP) framework. This characteristic, along with the sparsity of solution, where only support vectors contribute to predictions, makes SVR particularly robust and efficient in dealing with high-dimensional, noisy, or small-sample datasets. Equations (6) – (8) and the dual-based training mechanism are fundamental to the SVR algorithm and are widely cited in the literature [14] [15] [12] [13].

3.2 Key Hyperparameters

In Support Vector Regression (SVR), the effectiveness of the model is significantly influenced by the thoughtful selection and tuning of key hyperparameters, specifically the regularization parameter C, the kernel coefficient γ for non-linear kernels (such as the RBF), and the width of the ε-insensitive tube. The parameter C regulates the trade-off between minimizing training error and maintaining a smooth model structure. Higher values of C result in models that tightly fit the training data, which can lead to reduced generalizability, while lower values allow for a wider margin with increased tolerance for error. The γ parameter in the RBF kernel function determines the influence each training example has. A high γ value leads to a model that fits the training data closely, thereby increasing the risk of overfitting, whereas a low γ value yields a smoother function.

Meanwhile, the ε parameter establishes the width of the margin of tolerance, within which no penalty is applied for prediction errors. Proper tuning of this parameter enables the model to disregard minor deviations, thereby improving its robustness in the presence of noise. Hyperparameter optimization for SVR typically employs systematic search strategies, including grid search, random search, or Bayesian optimization, often guided by cross-validation performance. In this study, a TPE approach was utilized through the Hyperopt library to explore a defined search space and minimize validation loss (MSE). This approach ensured an efficient and reproducible convergence to optimal configurations. It is essential to note that these hyperparameters are problem-specific and should be optimized in conjunction with the dataset's characteristics and the regression objectives [16].

4. Gaussian Noise Injection for Data Augmentation

In this study, a controlled Gaussian noise injection strategy was employed, commonly referred to as jittering, as a data augmentation technique specifically for the training datasets of both the CNN-1D and SVR models. This approach involves perturbing each input feature vector $\mathcal{X} \epsilon \mathbb{R}^5$ by adding noise sampled from a zero-mean Gaussian distribution $\mathcal{N}(0, \sigma^2)$, while preserving the original target output intact. By artificially introducing statistically consistent variability, jittering enhances the effective training distribution, enabling the model to better generalize to unseen patterns and mitigate the risk of overfitting. The rationale behind this technique lies in its ability to simulate plausible variations within the physical limits of the system being modeled, specifically, solar still freshwater productivity, without violating domain realism. Notably, the augmentation was limited to the input space and applied solely to the training portion (70%) of the dataset, resulting in six perturbed replicas for each original instance. This process significantly enriched the training set while preserving the integrity of the validation and test distributions. Existing empirical evidence highlights the effectiveness of Gaussian noise augmentation in enhancing the robustness and predictive reliability of models, particularly in regression scenarios involving sparse data with continuous numerical inputs [17].

5. CNN-1D Model Augmentation Factor and Look-back Window Optimization

An optimization study was conducted to evaluate augmentation factors (3–10) and look-back windows (3–10) using the augmented CNN-1D framework. The results, ranked by the Overall Index of Model Performance (OIMP) in Table 1, revealed that an augmentation factor of 6 and a 7-day look-back window achieved the best balance of predictive accuracy, residual stability, and computational efficiency (Test RMSE = 0.045, MAE = 0.035, R^2 = 0.971, OIMP = 0.970). Longer windows (e.g., 9–10 days) yielded marginally higher R^2 values, albeit at the expense of overfitting

and increased computational cost. In contrast, smaller windows and lower augmentation factors underperformed due to weaker temporal representation. For methodological consistency and fairness, this optimized configuration was applied across all models: augmentation factor 6 was used for the augmented CNN-1D and the augmented SVR, while the 7-day look-back window was applied uniformly to both baseline CNN-1D and baseline SVR. The 7-day window also aligns with the natural weekly variability observed in meteorological conditions and solar still productivity, making it the most effective balance between capturing temporal dependencies and maintaining computational efficiency. This ensures that the comparative analysis accurately reflects differences in model architecture rather than variations in data preparation. Final reporting relies on test results, as they provide the most accurate and unbiased measure of generalization performance.

Table 1: CNN-1D augmentation–look-back optimization. Ranked results (augmentation factor 3–10; look-back 3–10) using the augmented CNN-1D model, with the optimal configuration (6,7) achieving the best balance of accuracy, generalization, and efficiency.

Rank	Aug. Factor	Look-back Window	Test RMSE	Test MAE	Test R^2	Test OIMP	Remarks
1	**6**	**7**	**0.04**	**0.03**	**0.97**	**0.97**	Optimal balance of accuracy and generalization; adopted for all models.
2	10	7	0.05	0.03	0.96	0.96	Competitive performance; slightly higher RMSE and computational cost.
3	10	9	0.04	0.03	0.97	0.96	Very strong R^2 but more prone to overfitting due to longer temporal memory.
4	9	8	0.04	0.037	0.96	0.96	Stable, but marginally weaker generalization compared to Rank 1.
5	8	7	0.05	0.04	0.95	0.95	Acceptable performance and clear degradation compared to top-ranked setups.
6	7	7	0.07	0.05	0.93	0.93	Generalization weakened; higher errors show underfitting.
7	7	6	0.07	0.05	0.93	0.93	Same issue as Rank 6; short window limited feature extraction.
8	5	7	0.07	0.05	0.93	0.93	Lower robustness; less effective temporal pattern learning.
9	3	7	0.07	0.05	0.92	0.92	Weak feature representation due to insufficient augmentation.
10	6	5	0.07	0.06	0.92	0.92	Smaller window lengthened errors; failed to capture dependencies.

11	3	5	0.08	0.05	0.91	0.92	Clear underperformance; augmentation too low.
12	4	7	0.08	0.06	0.91	0.91	Relatively weak; poor residual control.
13	5	6	0.08	0.06	0.88	0.90	High RMSE; suggests inadequate feature utilization.
14	5	4	0.10	0.09	0.85	0.87	Significant decline; short look-back ineffective.
15	6	4	0.10	0.08	0.86	0.88	Same as Rank 14; temporal dependencies lost.
16	3	3	0.11	0.09	0.84	0.86	Worst-case scenario; both augmentation and temporal depth too shallow.

Figure 3 provides visual evidence of the optimization process undertaken for the augmentation factor and the look-back window size. The scatter and heatmap plots clearly identify the configuration of six augmented samples per training instance and a seven-day look-back window as optimal, achieving the best balance between accuracy, stability, and computational efficiency. While higher augmentation factors and longer windows occasionally produced marginal gains in R^2, they were prone to overfitting and increased training cost. Conversely, smaller augmentation levels and shorter windows underperformed due to insufficient temporal representation. These results demonstrate that the adopted configuration is not arbitrary but grounded in systematic evaluation, ensuring fairness and integrity across all models. Collectively, these analyses underscore that augmentation-enhanced deep learning, particularly CNN-1D, provides a robust and scalable framework for forecasting freshwater productivity in conventional solar stills.

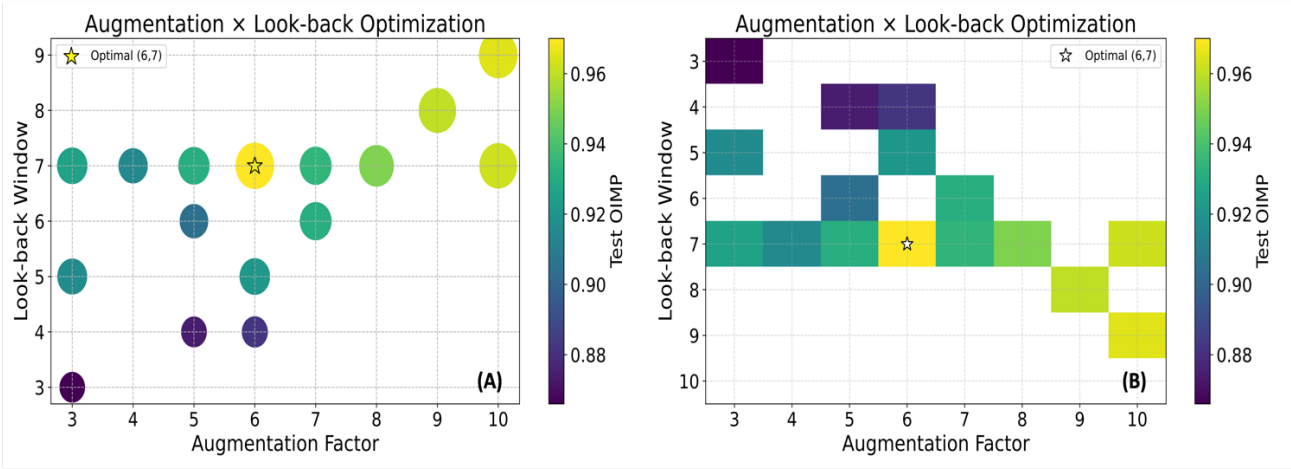

Figure 3: Optimization of augmentation and look-back settings. (A) Scatter plot of augmentation factor versus look-back window, with marker size inversely proportional to RMSE and color representing Test OIMP. (B) Heatmap of Test OIMP values, confirming the optimal setting (6,7).

6. Benchmarking Performance of Augmented Deep Learning and Regression Models

Before finalizing CNN-1D and SVR, extensive experiments were conducted with multiple architectures, including ANN, LSTM, GRU, transformers, and linear/ensemble models. While recurrent and transformer-based models demonstrated the capacity to capture temporal dependencies, their predictive improvements were limited given the small dataset size. They were accompanied by substantially higher computational costs and an increased risk of overfitting. CNN-1D, by contrast, offered superior predictive accuracy, strong generalization, and efficient training under the augmentation strategy. SVR was selected as a complementary benchmark because of its widespread use in regression with small datasets and its kernel-based non-linear modeling capability. To maintain focus, only CNN-1D and SVR were retained in the main manuscript, while the comparative outcomes of other tested models are reported in the Supplementary Materials for completeness. Table 2 presents the benchmarking results of eight models—three augmented deep learning models (ANN, LSTM, GRU), three augmented linear ML models (Ridge, Multiple Linear Regression, Elastic Net), and the two optimized models (CNN-1D and SVR). All models were trained and evaluated under the same data pipeline. The results confirm that augmented CNN-1D outperformed all alternatives, with the lowest test RMSE (0.04), highest R^2 (0.97), and OIMP (0.97). Augmented SVR, while competitive among linear methods, lagged behind CNN-1D in capturing complex temporal–meteorological interactions. LSTM and GRU showed promise but were computationally more expensive, while ANN underperformed significantly. Ridge, MLR, and Elastic Net demonstrated stable but limited predictive power.

Table 2: Model benchmarking results. Comparative test-stage metrics (RMSE, MAE, R^2, OIMP) for augmented deep learning and regression models, with best-performing results highlighted for CNN-1D and SVR.

Model	Stage	RMSE	MAE	R^2	OIMP	Remarks
ANN	Test	0.11	0.90	0.80	0.83	Underfitting; weak nonlinear capture.
	Train	0.13	0.10	0.78	0.82	High bias; poor generalization.
	Val	0.09	0.08	0.85	0.85	Better validation, still unstable.
LSTM	Test	0.06	0.04	0.93	0.94	Strong, but higher complexity than CNN-1D.
	Train	0.03	0.02	0.98	0.98	Very strong training fit.
	Val	0.05	0.04	0.96	0.96	Good validation, slight risk of overfitting.
GRU	Test	0.09	0.07	0.88	0.89	Balanced but weaker than LSTM.
	Train	0.07	0.06	0.92	0.92	Solid, but less accurate than CNN/LSTM.
	Val	0.08	0.07	0.89	0.89	Stable generalization.
CNN-1D	**Test**	**0.04**	**0.03**	**0.97**	**0.97**	Best overall; captures spatial–temporal features efficiently.
	Train	0.03	0.02	0.98	0.98	Excellent training fit without overfitting.
	Val	0.04	0.03	0.97	0.97	Robust validation.
Ridge	Test	0.11	0.09	0.81	0.84	Strongest linear model but limited nonlinear capture.
	Train	0.09	0.07	0.86	0.87	Stable, consistent performance.
	Val	0.13	0.11	0.74	0.78	Poor validation fit.
MLR	Test	0.11	0.09	0.81	0.85	Simpler linear baseline, outperformed by Ridge.
	Train	0.09	0.07	0.87	0.88	Reasonable training fit.

	Val	0.13	0.11	0.74	0.79	Weak validation performance.
Elastic Net	Test	0.12	0.10	0.78	0.83	Penalized model, still underperforms Ridge.
	Train	0.11	0.09	0.83	0.86	Moderately stable.
	Val	0.13	0.11	0.75	0.80	Over-penalization evident.
SVR	**Test**	**0.10**	**0.08**	**0.85**	**0.88**	Competitive, best among linear models, but weaker than CNN.
	Train	0.09	0.08	0.88	0.89	Stable training fit.
	Val	0.11	0.10	0.81	0.84	Moderate validation fit.

The comparative results presented in Figures 4 (A–D) highlight the relative performance of deep learning versus regression-based models. Deep learning architectures (CNN-1D, LSTM, GRU, and ANN) consistently achieved higher predictive accuracy (Test R^2) and lower error dispersion (Test RMSE) than regression-based models (SVR, Ridge, Elastic Net, and MLR). Among them, the augmented CNN-1D model delivered the strongest generalization (Test $R^2 = 0.97$, Test RMSE = 0.04, OIMP = 0.97), followed by LSTM, which also demonstrated robust performance but with higher computational demands. In contrast, regression models exhibited weaker alignment with nonlinear meteorological-productivity dynamics, with OIMP values generally below 0.85, confirming their limited suitability for high-variance time-series predictions.

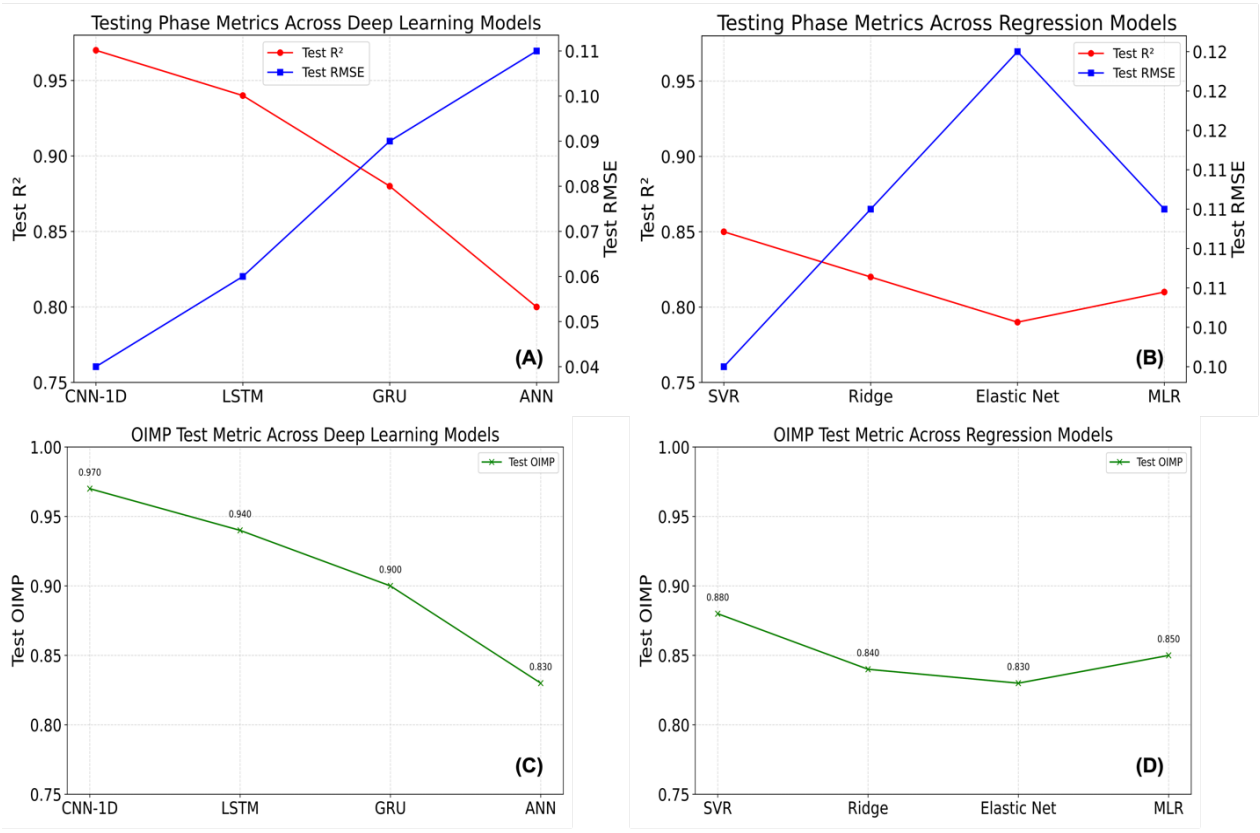

Figure 4: Benchmarking performance plots. (A) R^2 and RMSE of deep learning models (CNN-1D, LSTM, GRU, ANN). (B) R^2 and RMSE of regression models (SVR, Ridge, Elastic Net, MLR). (C) OIMP for deep learning models. (D) OIMP for regression models. The augmented CNN-1D shows the highest performance, with SVR as the strongest regression benchmark.

7. Classification Analysis of Model Performance

In addition to regression forecasting, a binary classification analysis was conducted as a complementary evaluation of model performance. This sub-task was designed to assess the capacity of different models to categorize freshwater productivity into “normal” and “high-output” ranges, reflecting a scenario of potential operational interest. The classification results in Figure 5 highlight the effectiveness of data augmentation in enhancing model performance, particularly within CNN-based architectures. Table 3 shows that the augmented CNN-1D model achieved the highest accuracy (88%) and F1-score (0.82), demonstrating improved generalization and a reduced false positive rate, which is crucial for the reliable classification of water output. Furthermore, its recall (0.88) demonstrates strong sensitivity in identifying high-output instances while maintaining better precision than other models.

On the other hand, the baseline CNN-1D model, although it achieved a slightly higher recall (0.92), exhibited lower precision (0.61) and accuracy (78%), resulting in a greater number of misclassifications. This underscores the role of augmentation in refining decision boundaries and enhancing classification reliability. Both SVR models (Augmented and Baseline) produced identical outcomes (accuracy: 78%, F1-score: 0.71, recall: 0.83, precision: 0.62), indicating that Gaussian noise had a negligible effect on their performance. This suggests that SVR's dependence on support vectors and kernel transformations renders it less sensitive to minor variations introduced by Gaussian noise. Unlike CNNs, which adaptively refine feature extraction using augmented data, SVR is built around support vectors and margin maximization, making it inherently less responsive to small perturbations. These findings reinforce the advantages of CNNs in processing augmented time-series data, demonstrating their potential to enhance predictive accuracy in solar still performance analysis.

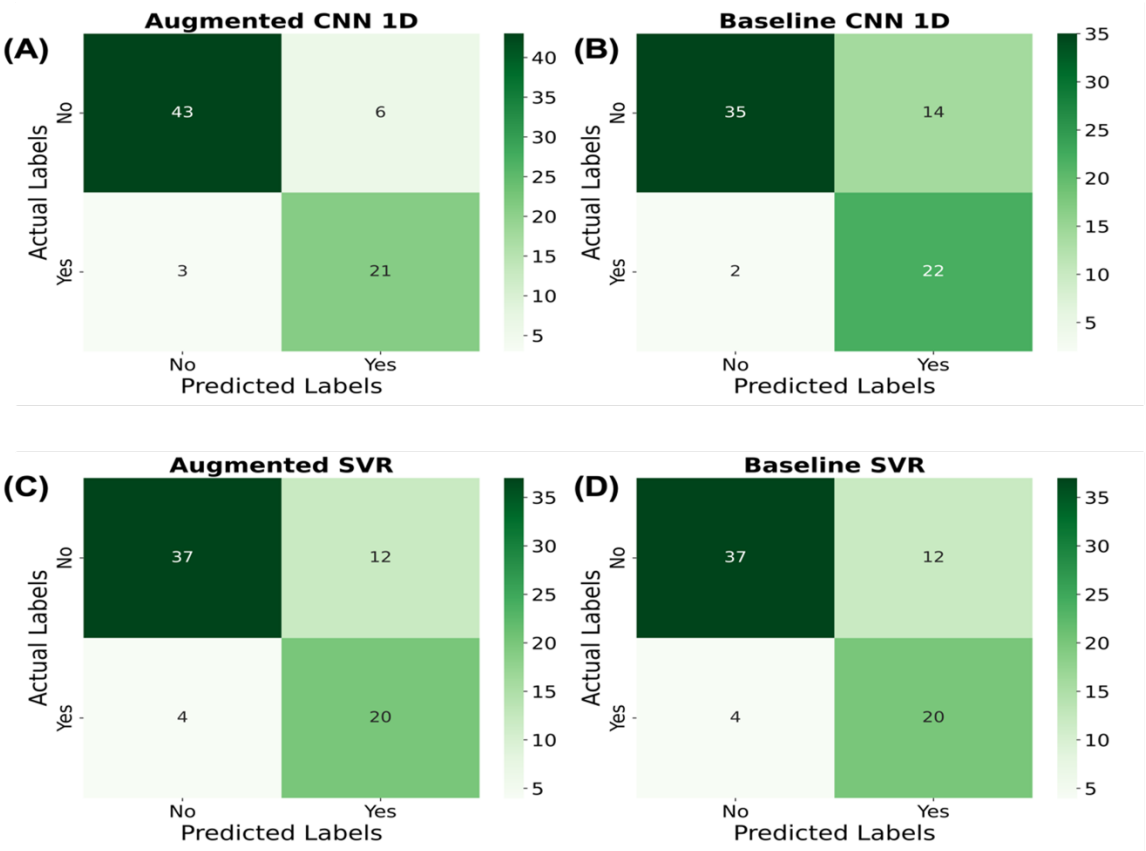

Figure 5: Classification confusion matrices. Matrices for CNN-1D and SVR models under baseline and augmented settings using a threshold-based classification of freshwater output (>1.25× mean). Augmentation notably reduced false positives for CNN-1D, improving classification stability.

Table 3: Classification performance metrics. Accuracy, precision, recall, and F1-score for CNN-1D and SVR models under baseline and augmented configurations. The augmented CNN-1D achieved an accuracy of 88% and an F1-score of 0.82, whereas SVR showed limited benefit from augmentation.

Model/ Metric	Accuracy	Precision	Recall	F1-Score
Augmented CNN 1D	**88%**	**0.78**	**0.88**	**0.82**
Baseline CNN 1D	78%	0.61	0.92	0.73
Augmented SVR	78%	0.62	0.83	0.71
Baseline SVR	78%	0.62	0.83	0.71

8. Conclusion

This supplementary document provides a comprehensive conceptual, methodological, and mathematical analysis of one-dimensional Convolutional Neural Networks (CNN-1D) and Support Vector Regression (SVR), designed for forecasting continuous numerical outputs in solar desalination systems. The CNN-1D model demonstrated strong capability in extracting localized temporal features from sequential meteorological and operational inputs, while the SVR model provided a kernel-driven framework for capturing nonlinear relationships in high-dimensional feature spaces. Both baseline and augmented SVR configurations were introduced, with detailed attention to their construction, learning objectives, and hyperparameter tuning. In addition to

providing theoretical insights, implementation factors were systematically addressed, including time-series preprocessing, data augmentation through Gaussian noise (jittering), and model optimization using the TPE algorithm.

A key methodological contribution was the systematic optimization of the augmentation factor and the look-back window size in the CNN-1D model. By jointly varying these parameters (factors 3–10 and windows 3–10), an optimal configuration of an augmentation factor of 6 and a look-back window of 7 was identified, based on the Overall Index of Model Performance (OIMP) and supported by RMSE, MAE, and R^2 metrics. This configuration balanced predictive accuracy, residual stability, and computational efficiency, providing the foundation for fair benchmarking across all models.

The benchmarking analysis compared eight augmented models spanning deep learning and linear regression families. Among deep learning methods, the augmented CNN-1D achieved the best performance ($R^2 = 0.97$, RMSE = 0.04, OIMP = 0.97), followed by LSTM and GRU, while ANN showed moderate results. Within the regression family, SVR (RBF kernel) emerged as the most competitive ($R^2 = 0.85$, OIMP = 0.88), outperforming ridge, elastic net, and multiple linear regression models, although it still lags the deep learning counterparts. These findings confirm that CNN-1D's hierarchical feature extraction is better suited to capturing temporal dependencies and noise-induced variability than regression-based methods. They also demonstrate the relative robustness of SVR compared to other linear models. Moreover, the introduction of a threshold-based classification approach provided improved interpretability and resilience against outliers, further extending the framework's practical applicability.

Collectively, these analyses underline the reliability, reproducibility, and practical applicability of the proposed framework. By integrating augmentation × look-back optimization with systematic benchmarking, the study establishes a rigorous foundation for model comparison, ensuring that performance outcomes reflect architectural strengths rather than inconsistencies in data representation. This not only strengthens confidence in the CNN-1D as the optimal predictive model but also clarifies the role and limitations of regression-based alternatives in environmental time-series forecasting.

Nomenclature

P_{std}	—	Freshwater productivity, L/day
Is	—	Solar Radiation, MJ/day
T_{amb}	—	Ambient temperature, ^oC
V_w	—	Wind speed, m/s
RH	—	Relative humidity, %
P_{st}	—	Saline water feed flowrate, L/day

List of Abbreviations

ML	—	Machine Learning
AI	—	Artificial Intelligence
CNN	—	Convolutional Neural Network
1D	—	One Dimensional

SVR	—	Support Vector Regression
ReLU	—	Rectified Linear Unit
L2	—	CNN-1D Regularization Strength
TPE	—	Tree Parzen Estimator
MSE	—	Mean Squared Error
MAE	—	Mean Absolute Error
SGD	—	stochastic Gradient Descent
RBF	—	Radial Basis Function
R^2	—	Coefficient of determination
RMSE	—	Root mean square error
OIMP	—	Overall index of model Performance

Appendix

Mathematical Symbols Definition

Symbol Name		Definition/ Meaning
$\tilde{X}_{train}[i,j]$		Augmented input sample using Gaussian noise
$\tilde{\mathcal{Y}}_{train}[i,j]$		Ground truth label (unchanged during augmentation)
$X_{scaled}, \mathcal{Y}_{scaled}$		Normalized features and labels using Min-Max scaling
$Z^{(l)}[i,j]$		Output of convolution before activation
$A^{(l)}[i,j]$		Activation output at time step [i, j] from the current layer
$A^{(l-1)}[i+k]$		Activation output at time step [i, j] from the previous layer
$A^{(0)} = x$		Original input time series to the first convolutional layer
$w^{(l)}[k,j]$		Weight of kernel position k for filter j
$b^{(l)}[j]$		Bias term for filter j
P		Padding size
L_{in}		Input sequence length
L_{out}		Output sequence length
S		Stride (1)
K		Kernel size (3)
k		Kernel index from 0 to $K-1$ ($K=3$)
i		Time index (temporal position)
j		Filter index (1 to 128)
l		Layer index (l = 1,2,3)
$f \in \mathbb{R}^M$		Flattened feature vector, M = T × filters
h_m		Output from the m-th neuron in the dense hidden layer
$\max(0,.)$		ReLU activation function.
W_{mn}		Weight connecting n-th feature to m -th neuron in the dense hidden layer.
b_m		Bias term associated with the m-th neuron in the dense hidden layer.
f_n		n-th input value from the flattened layer $f \in \mathbb{R}^M$
M		Total number of neurons in the dense hidden layer (128).

$\hat{y}$		Predicted scalar output value for a generic input sample x
$w_m^{(0)}$		Weight connecting m-th neuron in the dense hidden layer to output layer.
$b^{(0)}$		Bias term in the output layer.
η		Learning rate ∈ [10^{-4}, 10^{-2}]
λ		Regularization strength ∈ [10^{-4}, 10^{-2}]
θ_t		Learnable parameter (weights and biases) vector at iteration t
θ_{t+1}		Updated set of parameters for the next iteration
$\nabla_\theta \mathcal{L}$		Gradient loss function $\mathcal{L}$ with respect to the parameters θ at iteration t
x_i		Support vectors from training
x		Current input for which we are predicting $\hat{y}$
α_i , α_i^*		Dual Lagrange multipliers
$K(x_i, x)$		RBF kernel function that measures similarity between input vectors
b		Model bias
N		Total number of training instances
γ		Kernel coefficient
$\mathcal{L}_{\mathrm{SVR}}$		SVR loss function/ learning objective
w		Weight vector in the induced feature space
C		SVR Regularization parameter
ε		Width of the margin (tube) around the regression function
y_i		Actual continuous output value
$\hat{y}_\mathrm{i}$		Predicted continuous output for the i-th training sample x_i
$\mathcal{N}(0, \sigma^2)$		Gaussian noise with mean 0 and variance σ²

References

[1] A. Jernelv, I. L., Hjelme, D. R., Matsuura, Y., & Aksnes, "Convolutional neural networks for classification and regression analysis of one-dimensional spectral data," *arXiv Prepr. arXiv2005.07530*, 2020.

[2] J. Brownlee, *Deep learning for time series forecasting: predict the future with MLPs, CNNs and LSTMs in Python. Machine Learning Mastery*. 2018.

[3] A. Sánchez-Reolid, R., de la Rosa, F. L., López, M. T., & Fernández-Caballero, "One-dimensional convolutional neural networks for low/high arousal classification from electrodermal activity," *Biomed. Signal Process. Control*, vol. 71, p. 103203, 2022.

[4] Q. J. Zhou, Y., Wu, W., Nathan, R., & Wang, "Deep learning-based rapid flood inundation modeling for flat floodplains with complex flow paths," *Water Resour. Res.*, vol. 58(12), p. e2022WR033214, 2022.

[5] A. Cacciari, I., & Ranfagni, "Hands-On Fundamentals of 1D Convolutional Neural Networks—A Tutorial for Beginner Users," *Appl. Sci.*, vol. 14(18), pp. 2076–3417, 2024.

[6] I. Goodfellow, *Deep learning*. MIT Press, 2016.

[7] F. Chollet, F., & Chollet, *Deep learning with Python*. Simon and Schuster, 2021.

[8] A. Shekhar, S., Bansode, A., & Salim, "A comparative study of hyper-parameter optimization tools," in *In 2021 IEEE Asia-Pacific Conference on Computer Science and*

Data Engineering (CSDE), pp. 1-6 IEEE.
[9] J. Kingma, D. P., & Ba, "Adam: A method for stochastic optimization," *arXiv Prepr. arXiv1412.6980*, 2014.
[10] D. Bergstra, J., Yamins, D., & Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *In International conference on machine learning, PMLR*, pp. 115–123.
[11] D. D. Bergstra, J., Yamins, D., & Cox, "Hyperopt: Distributed asynchronous hyper-parameter optimization. Astrophysics Source Code Library, ascl-2205." 2022.
[12] N. Verma, "An Introduction to Support Vector Regression (SVR) in Machine Learning," *Medium*, 2023. https://medium.com/@nandiniverma78988/an-introduction-to-support-vector-regression-svr-in-machine-learning-681d541a829a.
[13] I. Bhattacharyya, "Support vector regression or svr," *Coinmonks*, 2018. .
[14] B. Smola, A. J., & Schölkopf, "A tutorial on support vector regression," *Stat. Comput.*, vol. 14, pp. 199–222, 2004.
[15] V. N. Vapnik, *The Nature of Statistical Learning Theory*. Springer, 1995.
[16] N. Van Otten, "Support Vector Regression (SVR) Simplified & How To Tutorial In Python," Data Science," *Machine Learning*, 2024. https://spotintelligence.com/2024/05/08/support-vector-regression-svr/.
[17] S. Iwana, B. K., & Uchida, "An empirical survey of data augmentation for time series classification with neural networks," *PLoS One*, vol. 16(7), p. e0254841, 2021, [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0254841.