**ADTA 5550 Section 001 - Deep Learning with Big Data (Fall 2024)**

A Project Report on

**Predicting Cooling System Performance in Power Plants Using CNN and RNN Models**



**UNIVERSITY OF NORTH TEXAS**

**Team Name:** Cascaders

1) Shabana Shaik - 11766712

2) Shabbeir Ahammad Shaik - 11659107

3) Sai Swetha Puli - 11688269

**Abstract:**

The project focuses on the performance prediction of cooling systems in power plants with the use of advanced deep learning models, such as CNNs and RNNs. This research pinpoints vital factors-water consumption intensity, energy load, and environmental conditions-that have an impact on the efficiency of the cooling process by investigating historical operational data. EDA provides substantial insights, like seasonal inefficiencies and operational metric correlations, which are then integrated into the modeling process.

While the CNN model captures the spatial relationship of plant configurations, the RNN model has been proven efficient for temporal trend analyses. As observed, the best performances are from the hybrid CNN-RNN model using both spatial and temporal features for optimum prediction. The approach has certain limitations concerning restricted feature availability and computational complexity; results provide very accurate forecasting of performance and allow the strategy to be predictive for maintenance.

The role of deep learning in operational efficiency, cost reduction, and sustainability at power plants comes into view in this research. The research findings give insights into optimizing cooling systems and energy management practices.

**Table of Contents:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Introduction:** | |  | | --- | |  |  |  | | --- | | Introduction to cooling systems in power plants and their significance.  Research Goal: Predicting cooling system performance using deep learning models. | |
| **Background of Cooling Systems:** | Impacts of inefficient cooling and importance of predictions. |
| **Overview of Machine learning in Energy Systems:** | Application of CNNs for spatial data analysis and RNNs for time-series predictions.  Role of machine learning in optimizing cooling system performance. |
| **Literature Reviews:** | Studies on cooling system optimization using machine learning.  Deep learning models for spatial and temporal data analysis.  Applications of hybrid CNN-RNN models. |
| **Techniques:** | Data cleaning and preprocessing steps, including handling missing values, normalization, and feature engineering. |
| **Exploratory Data Analysis (EDA):** | Understanding the dataset structure and uncovering trends prior to model development. |
| **Dataset Description:** | Detailed breakdown of key features used in the dataset. |
| **Data Handling:** | Methods for cleaning and structuring the data. |
| **Choosing Features:** | Selection process for features with the highest predictive importance. |
| **Normalization and Encoding:** | Approaches for transforming and encoding data for model training. |
| **Correlation Analysis:** | Relationships between key variables to inform feature selection. |
| **Development of Models:** | CNN Model Architecture.  RNN Model Architecture.  Hybrid CNN-LSTM Model Architecture. |
| **Model Architecture:** | Structural details of CNN, RNN, and hybrid CNN-RNN models. |
| **Conclusion:** | Summary of results and findings. |
| **Bibliography:** | List of references used in the project. |

**Introduction:**

In addition to expanding the power plants, upgrading them is also necessary, with measures that are very crucial in their operations. Cooling systems remain crucial in ensuring that the running mechanisms of turbines and generators remain functional safely and efficiently. Any inefficiency in this critical feature would automatically cause abnormal power consumption, with all the ensuing consequences for running costs, not to say possible effects concerning the use of accelerated equipment degradation due to shortened life spans of equipment.

The focus of this study, therefore, lies in using state-of-the-art deep learning methods for performance prediction of cooling systems. In this context, we use historical data from various geographically diverse power plants to develop predictive models using CNNs and RNNs. While CNNs capture the spatial relationships in plant configurations, RNNs model time-series dependencies that provide insights into temporal trends in the cooling performance. The underlying idea of these models involves improvements in operational efficiency, reduction in cost, and the achievement of sustainability goals for better-managed cooling systems.

## **Literature Review:**

• Machine Learning Applied to Performance Prediction of Cooling Systems: Different machine learning approaches have been applied to optimize the efficiency in cooling systems at various power plants. The regression models, decision trees, and ensemble methods such as Random Forest showed promise but largely failed to capture the complex relationships inherent in large-scale operational data. Recently, deep learning models have gained much attention due to the fact that they are able to automatically extract relevant features from raw data with little preprocessing. RNNs are suitable for time-series data, enabling models to retain historical patterns and forecast trends in cooling performance.

• Hybrid CNN-RNN Models:

Hybrid models combine those in CNNs with RNNs. CNNs would serve them well by capturing only the overall layout of the plants in terms of spatial or even information configurations, while RNNs provide temporal dependencies. By developing, in fact, such an amalgamation-one really applicable to cooling performance relying heavily both on spatial and temporal factors-skilled hybrid models enable power plant operators to utilize every bit of plant data in making smart and thus optimized cooling strategies that reduce a manyfold of possible inefficiencies.

**Techniques:**

**1. Preprocessing and Cleaning of Data:**

The dataset was preprocessed through extensive cleaning and normalization for deep learning models such as CNN, RNN, and LSTM. After that, the selection of respective columns was done, such as features like Water Consumption Intensity Rate, Water Withdrawal Rate, and target variable Cooling Status. This reduced noise and ensured that the dataset focuses on the essential information to be used for modeling. Some numeric-like columns were kept as strings because of inconsistencies in the raw data. These have been converted to proper numeric types by replacing invalid entries with Nan for compatibility with numerical operations and the models.

For treating missing values (Nan) and zero values, it was replaced by column mean, calculated after excluding zeros to avoid skewed results. This imputation ensured the data through the dataset would introduce no bias due to such missing values. StandardScaler was used for scaling numerical columns within the dataset with a mean of 0 and having a standard deviation of 1. This enhances the efficiency while improving the convergence for neural networks by ensuring that during learning no feature dominates others. The categorical columns, like Cooling System Type and Water Type, were encoded into numerical values using LabelEncoder; also, the target variable Cooling Status was encoded for categories like "Operational", "Reserved", "Out of service", "Standby", "Mixed". In this way, the dataset became fully compatible with all requirements of machine learning and deep learning frameworks. The dataset was further divided into an 80% training subset and a 20% testing subset to evaluate the performance of any model on unseen data reliably. While splitting, a random seed was used to make sure the results are reproducible.

It finally prepared the data in tensor format for direct use in PyTorch models. It implemented a custom Dataset class in PyTorch that will load data efficiently during model training and evaluation. The collective preprocessing steps are toward achieving a clean, normalized, and structured state of the dataset for actual training with the CNN/RNN/LSTM-CNN hybrid models described in this paper. This not only results in smooth model training but also contributes much to reliability in the high accuracies received upon model evaluation.

**2. Dropping Unwanted Columns:**

Columns with high proportions of missing values or those irrelevant for the prediction of cooling system performance were removed. For example, the removal of attributes irrelevant to cooling operations allows the model to focus on impactful features. This ensures that the data is cleaner and more concise, hence avoiding biases in the predictions.

**3. Feature Engineering:**

1) DateTime Conversion: Date and time columns were converted to a standardized datetime format to capture the temporal trends effectively. This is an important step in doing the analysis on sequential data.

2. Label Encoding: Categorical data like Cooling System Type and Generator Technology were converted to numeric values through label encoding, hence making them machine-learning-friendly.

3. Normalization: Features like Energy Output and Water Consumption were scaled between the range of 0 to 1 using Min-Max Scaling. The idea was that it might prevent features with a big range from dominating the learning process.

**4. Splitting of Dataset:**

It was then followed by splitting the data into training and test sets, 80% to the training set and 20% to the test set, respectively. This would allow the model to learn quite effectively from the majority of the data and generalize well on previously unseen data.

**5. Characteristics of Time-Series Data:**

Characteristics of time series are relevant for capturing performance in cooling systems. For example, some of the variables include hourly energy load and water consumption, which have temporal dependencies. The RNN model will automatically learn these temporal dependencies to foresee the behavior of the system for future moments in time.

**EDA - Exploratory Data Analysis:**

The exploratory data analysis is done to find the structure of the dataset and to find trends, too, before any model is developed.

Box plots and histograms were used to find outliers and generalize trends in cooling efficiency over a wide range of operation scenarios.

1.) Seasonal Trends: Because of the amount of heat in the air, and thus the elevated equipment loading, there was greater cooling inefficiency during summer months.

2) Type of cooling system applied: closed-loop systems performed steadily, while once-through systems varied their performance in relation to the ambient conditions.

**Dataset Description:**

Normally, the dataset consists of 70 columns and 14703 columns. It extracts useful columns from those 70 columns to the new dataframe. It saves that new dataset into a new csv file and runs all the specific normalizations and preprocessing on that extracted new csv file called final.csv.

First, we check the relations of the columns against the target column and came up with some columns which directly manipulate or interact with the status of the cooling system. Key Columns: Utility ID', 'State', 'Plant Code', 'Plant Name', 'Year', 'Month', 'Generator ID', 'Boiler ID', 'Cooling ID', 'Water Consumption Intensity Rate (Gallons / MWh)', 'Water Withdrawal Rate per Fuel Consumption (Gallons / MMBTU)', 'Water Consumption Rate per Fuel Consumption (Gallons / MMBTU)', 'Cooling System Type', 'Water Type', 'Water Source', 'Water Source Name', 'Fuel Consumption from Steam Turbines (MMBTU)', 'Summer Capacity of Steam Turbines (MW)', 'Gross Generation from Steam Turbines (MWh)', 'Net Generation from Steam Turbines (MWh)', 'Minimum Generator Inservice Month', 'Minimum Generator Inservice Year', 'Maximum Generator Inservice Month', 'Maximum Generator Inservice Year', 'Cooling Status’.

Data Dictionary:

Utility ID: A unique identifier for energy providers.

Plant Code: Unique code for the power plant.

Plant name: It consists of the plant name

Year & Month: Timeframe data

Generator ID: The ID of the generator which is giving the energy to the cooling system

Boiler ID: the Id column of the Boiler plate which is used by the cooling system.

Water Withdrawal Rate per Fuel Consumption (Gallons / MMBTU): The amount of water consumed (in gallons) per megawatt-hour (MWh)

Water Consumption Rate per Fuel Consumption (Gallons / MMBTU): the amount of water consumed (in gallons) for every million British thermal units (MMBTU)

**Water Type**: Specifies the category of water used, such as freshwater, reclaimed water, or saline water, depending on its source and quality.

**Water Source**: Indicates the origin of the water used, such as a river, lake, groundwater, or a desalination plant.

**Water Source Name**: The specific name of the water source, e.g., "Mississippi River" or "Lake Michigan."

**Fuel Consumption from Steam Turbines (MMBTU)**: Measures the energy content of the fuel consumed by steam turbines, expressed in million British thermal units (MMBTU).

**Summer Capacity of Steam Turbines (MW)**: The maximum energy output capacity of the steam turbines during summer, measured in megawatts (MW). This accounts for the typically reduced efficiency in hot weather.

**Gross Generation from Steam Turbines (MWh)**: The total energy generated by steam turbines before deducting any energy used internally, measured in megawatt-hours (MWh).

**Net Generation from Steam Turbines (MWh)**: The energy output delivered to the grid or users after accounting for internal energy use, also measured in megawatt-hours (MWh).

**Minimum Generator Inservice Month**: The month in which the generator began its operation or was first commissioned.

**Minimum Generator Inservice Year**: The year corresponding to the generator's initial commissioning or start of operation.

**Maximum Generator Inservice Month**: The most recent month marking changes to the generator's operational status (e.g., upgrade or decommissioning).

**Maximum Generator Inservice Year**: The most recent year marking changes in the generator's operational timeline.

# **Data Handling:**

First of all, cleaning the final.csv dataset in order to use it involved several steps of improving the quality, consistency, and analytical readiness of the data. First to take place was the removal of null values, since any gaps in the data could easily have led to biased or otherwise wrong analyses; the removal of those gaps guarantees that the resultant dataset would be robust and reliable. After that, it became obvious from the detailed review of the columns that some numerical data, such as "Fuel Consumption from Steam Turbines" and "Gross Generation from Steam Turbines," were wrongly stored as strings. Such a discrepancy might cause an inability to perform any numerical operations because string variable types cannot support arithmetic or statistical functions. This was addressed by going through column by column, and whatever similar occurrences were found, changing the data to appropriate types, either floats or integers depending on the column for specific analytical needs.

Upon correction of data types, careful observation showed that indeed, there were outliers in the real data, including negative values for some fields that really do not make logical sense in this type of data. An example is that the range of measured variables such as fuel consumption and energy generation cannot take negative values concerning real-world applications. Outliers of this character might distort the statistical properties of a distribution, misrepresent its visualization, and maybe bring into the model bad predictive qualities. In handling these anomalies, the fields containing outliers were treated in a structured manner: outliers were replaced by mean or median value, as dictated by the context and variability of the column. Use of the mean keeps the central tendency of the data better, provided the values are symmetrically distributed. On the other hand, in cases of a skewed distribution, the median works better since it is less sensitive to extreme values.

In the process, the approach ensured that the methodology was data-driven to ensure at all instances and with each modification, the integrity of the dataset was kept intact. The cleaning of data was done from raw and unstructured to clean and consistent data, ready for the analysis stage by treating the null values, correcting the wrong data types, and also handling the outliers. This ensured that the data were accurate but more usable in subsequent processes such as visualization, statistical analysis, and machine learning. High-quality resultant dataset which is bound to give reliable insights and thereby drive decision-making with confidence. In summary, this underpins the place of strong data cleaning at the foundational core of any analytical workflow.

# **Choosing Features:**

This plot identified the most influential features of this feature importance plot that, in turn, informed feature selection within this model. Main features have been chosen in view of their contribution to the model's predictive precision and relevance to context. Summer Capacity of Steam Turbines in MW became the most important feature and therefore indicated how strong this relation was with energy performance. The identification variables selected were Utility ID and Plant Name to distinguish between operational and location-specific characteristics. The operational metrics include Maximum Generator Inservice Year and Net and Gross Generation from Steam Turbines MWh, which capture technological advancement and energy output, respectively.

In this respect, only resource-related variables were considered, such as Water Withdrawal Rate per Fuel Consumption and Water Consumption Intensity Rate, in order to provide insight about environmental efficiency. Besides, features like Cooling ID and Boiler ID were also considered important, given their relevance with the efficiency of the plants and energy generation. Variables of higher importance were retained, and 'Year' and 'Month', being of little importance, therefore had to be removed in light of the fact that feature inclusion further aids at regularization and reduces model complexity. Features have been selected in an approach that gives the strategic view for the model: offering views on accuracy and interpretability operational performance, resource efficiency, and energy generation with lower computational overhead.

**Normalization and Encoding:**

Label Encoding and Data Encoding: Categorical variables are handled by performing label encoding. First, the identification of categorical columns was done from the dataset; then, each category in those columns was mapped to a unique numerical value using LabelEncoder from scikit-learn. This is how the categorical data could get transformed into a format which could act as input for machine learning models, which take numerical data as input. Here, for example, the features "Plant Name" and "Water Type" have been encoded to integer format so that the same could be effectively processed by the neural network. This is why the "Cooling Status" target variable was label-encoded, too, so the classes take numeric form for the classification task. This is done so that the categories are represented to the machine learning model without any relationship that may actually not exist.

More precisely, the label encoding of the target variable is highly important, as it provides a mapping of classes to integers so that the PyTorch loss function CrossEntropyLoss works, for example, requiring target labels to be in numerical format. While label encoding is one of the simplest and efficient ways of encoding, ordinal relationships on nominal data were avoided by not mistakenly introducing such encoding in order to avoid biasing the model. Hence, encoding all categorical variables provided a structure of data in the format appropriate for further pre-processing and training of the neural network.

Normalization: Normalization was done to scale the numeric features to normal scale for good model training. Then, numerical features were standardized using a mean of 0 and a standard deviation of 1 via StandardScaler from the scikit-learn library. This is important since most deep learning models are sensitive to scaling of variation in features. In other words, most deep learning models don't understand the unit of measurements; for example, meters versus kilometers. Unless normalized, those features reflecting larger variation will dominate the model under training and lead to poor convergence and learning.

where μ represents the mean and σ, the standard deviation of the feature. This step has ensured that all the features contributed equally during model training. An example could be that some variables, such as "Fuel Consumption" and "Net Generation", are being normalized to make sure their respective scales are not disproportionately affecting the CNN. Normalization is important in a neural network with gradient-based optimization methods since it avoids large gradients and hastens convergence. This step was, therefore, an important preprocessing for performance and stability enhancement of the model; hence, efficient and correct prediction was realized during training and evaluation.

**Correlation Analysis:**

Positive Correlations:

The next pair of variables are "Gross Generation from Steam Turbines (MWh)" versus "Net Generation from Steam Turbines (MWh)". Because the correlation is positive with its value close to 1, we would believe that as one rises, the other does rise in an essentially proportional manner. This we could expect since both represent magnitudes of energy output.

• A second correlation may exist between "Minimum Generator Inservice Year" and "Maximum Generator Inservice Year "; the closer the minimum in-service years of two generators are to one another, the closer will their maximum in-service year also be due to following similar operational life cycles.

Strong Negative Correlations:

• There are also a few negative correlations, such as with the "Minimum Generator In-service Year" and "Cooling Status." It tells the story that older generators mean different cooling statuses than more newly created generators. This may signify something related to the change in cooling technologies or possibly regulations over time.

Low or No Correlation:

• Most of the variables give almost zero correlation value, showing no linear relations; few examples are "State," "Plant Name", and "Cooling Status." Examples being that "State does have a very little Linear Influence on numerical factors that could possibly influence "Fuel Consumption from Steam Turbines."A screenshot of a computer screen

Description automatically generated

# **Development of Models:**

**1. Recurrent Neural Network:**

The RNN is one for the analysis of the sequences in data. This model has a memory of past input values, important to foresee the performances of coolants relying on time.

**2. RNN Architecture:**

The architecture of the RNN model, which has been used for the purpose, contains two layers of simple RNN cells. Each of these layers is of size 100. The architecture is described in detail below.

• Input Layer: The layer that takes in input sequences and changes the input data. Examples include hourly energy output and water consumption.

• First RNN Layer: The layer here will learn temporal patterns of data through time and transfer the hidden state across time by return\_sequences=True.

• Dropout Layer: Performs regularization by randomly turning off 30% of neurons so that the model does not get overfitted.

• Second RNN Layer: Further processing of sequential data to output only the last hidden state.

• Dense Layer: Increases the model complexity, with 50 units having ReLU for refining learned features.

• Out Layer: It gives one value as output, which can be the expected efficiency of the cooling system using a linear activation function.

# **Model Architecture:**

**1. CNN Model:**

Actually, CNN is huge in convolutional layers and a full-joined layer count. It depicts the local patterns in data; hence it extracts features, which afterwards with the help of a full connection one is able to predict efficiency using conventional processing. This approach employing even convolution layers on tabular data might discover latent neighbor structure there with numeric features.

DataLoader:

• These training and test datasets are then wrapped in the DataLoader objects; thus, this enables easy batching and shuffling during training. It allows for efficient computation because of batching. Also, it prevents the model from learning the order of the data; hence, results free of biases.

**Convolutional Layers:**

• conv1: This is a 1D convolutional layer with 16 output channels and a kernel size of 3. This layer identifies the pattern in the features.

• conv2: A second 1D convolutional layer with 32 output channels, but the kernel size is still 3: This adds more abstraction and thus captures more complex relationships between features.

Fully Connected Layers:

• fc1: This is the fully connected layer containing 128 units; it takes its input from convolutional layers. This layer aggregates the extracted features into a meaningful representation to prepare the features for a prediction.

•fc2: One-unit output layer which shall predict the efficiency score, an activation 'Sigmoid' for having the output within a range of normalized efficiency values;.

Activation Functions: ReLU for non-linearity in each of the convolutional and fully connected layers, while the Sigmoid function is used in the final layer to normalize the prediction.

**Loss Function and Optimizer:**

• Loss Function: The difference between the prediction and actual values using nn.MSELoss(), because the model is going to predict a continuous value, which is efficiency, hence mean squared error will be an appropriate loss function.

•Optimizer: The optimizer used here is torch.optim.Adam(). Because of its adaptive learning rate and nature of handling sparse gradients, Adam is very suitable for this problem. Training **Process:**

• The train\_model() function runs training for a number of epochs specified. • The model, for each epoch, calculates the predictions, computes the loss, and optimizes weights using backpropagation and an optimizer.

• The loss is printed at the end of each epoch to monitor the training process of each.

**Evaluation:**

* The function evaluate\_model() evaluates the model on the test data.
* During evaluation, the model is set to inference mode - model.eval() to make sure no weights are updated.
* The predictions and actual values will be stored for computing some metrics of error, for instance, MAE, or an accuracy metric.A graph showing the results of a forecast

  Description automatically generated

2. The RNN Model:

This is implemented in a recurrent layer: nn.RNN, with a size of 2 layers of 64 hidden units, followed by a fully connected layer to output the single prediction. The recurrent layer learned sequential patterns across features; afterward, this information was combined using the fully connected layer and issued with a final efficiency score. This output is then fed through a sigmoid activation to ensure the predicted efficiency lies between 0 and 1, matching the normalized output.

DataLoader Setup

• The training and test data are wraped in the DataLoader with a batch size of 64 for efficient batchings of data while training and evaluating the model.

• Using DataLoader allows the model to shuffle the training data for one, so that it does not learn any order-related bias.

• INPUT PARAMETERS:

o input\_dim: This parameter defines the number of input features. In this case, it is equal to the number of features selected for the model.

othidden\_dim: This defines the number of neurons in the hidden layers. In this model, this is set to 64, which allows the RNN to capture a reasonable amount of information for each sequence.

o\tnum\_layers: This refers to the number of stacked RNN layers. In this case, it is set to 2, allowing the model to learn more complex patterns.

• RNN Layer:

o The model contains an nn.RNN layer, which is used to process the sequence of features.

First, o\tbatch\_first=True means that the input tensor is of the form (batch\_size, sequence\_length, input\_dim).

o The nonlinearity used is relu, hence the model has non-linear activation to grasp complicated relationships.

• Fully Connected Layer: nn.Linear

o After the output of RNN, the final hidden state from last time step has been forwarded through a fully connected layer (fc) and generates only one prediction.

o The fc layer is of size 1, as it predicts a single number value of the efficiency.

• Sigmoid Activation:

o The output from the fully connected layer is fed through a Sigmoid activation function, constraining the output value in the range between 0 and 1.

o\t This is in line with the normalized target variable, Efficiency, also within [0, 1].

• Forward Method:

o The forward method first reshapes the input tensor, adding one more dimension to turn it into the format suitable for RNN.

o The output results by passing data through the RNN followed by a full connection and finally a Sigmoid activation.

4. Loss and Optimizer

• Loss Function:

o\tMean Squared Error loss is used here because it is a regression problem; hence, the model is supposed to predict a continuous value.

• Optimizer:

o\tAdam Optimizer (torch.optim.Adam): It has been used for the purpose of gradient-based optimization, wherein the learning rate is selected to be 0.001. It will be using an adaptive learning rate; thus, it is chosen as being very effective and efficient to carry out the training process.

5. Training Processes

• The function train\_model() is defined that trains the model on a number of epochs. Number of epochs here is specified as 20.

• For every batch of data, it makes a prediction, calculates the loss, and then backpropagates to adjust the weights.

• The optimizer is responsible for adjusting the weights in each training step, and the epoch loss is printed after every epoch to monitor the training progress.

6. Evaluation

• evaluate\_model() provides the performance of the model on the test dataset.

• model.eval() is called to set the model to evaluation mode, by default disabling dropout and other layers specific to training.

• The test loss is computed and predictions are stored for further analysis.

7. Accuracy Metrics

• Mean Absolute Error (MAE), one of the general metrics of regression, is computed afterward.

• Accuracy of the computations as calculated by 100 minus the mean of absolute errors in consideration of the normalized scale. This would give an idea about how well the model has learnt the relationships in the data.A graph showing the difference between a line and a line

Description automatically generated with medium confidence

**3. Hybrid Model: LSTM\_CNN:**

**Setup of DataLoader**

* Training and Testing datasets load with the help of the DataLoader object, which means a batch size is taken as 64 for handling data in chunks while training and testing by the model.
* The DataLoader is set to shuffle the training data to make sure biases do not occur due to the order of the data.

**Architecture of Hybrid CNN-LSTM Model**

The Hybrid CNN-LSTM model thus represents the marriage of strengths between convolutional and recurrent layers for the extraction of spatial relationships among features with sequential dependencies embedded in the dataset.

**Input Parameters:**

* input\_dim: Number of features in the input data (e.g., 6 in this case).
* cnn\_out\_channels: The number of output channels for CNN layers. In this case, it is 16. These are the channels helping to learn several feature maps.
* lstm\_hidden\_dim: Dimension size of hidden for LSTM layers. Set to 64. This will define the size of the hidden state in LSTM which captures rich temporal information.
* otlstm\_num\_layers: It represents the number of LSTM layers stacked. 2. Stacking LSTM layers in a model helps to capture deep sequential patterns in the data.
* Convolutional Layers: (conv1 and conv2)
* conv1: a 1D convolutional layer with 16 output channels and a kernel size of 3; it takes input features and captures local patterns from them.
* CONV2: A second 1D convolutional layer that has 16 output channels with a kernel size of 3; this layer enhances the feature extraction done previously by increasing the number of filters.
* Activation Function: The ReLU activation for introducing non-linearity follows each convolutional layer to help the model learn complicated mappings.

**LSTM Reshaping:**

* The output from the convolutional layers is reshaped using x.permute(0, 2, 1) into a form suitable for LSTM input; input now transformed to the sequence format: (batch\_size, seq\_length, cnn\_out\_channels).

**LSTM Layer:**

* The reshaped output is fed into the LSTM layer, which processes sequences. It is good for capturing dependencies, and temporal patterns may be informative for the dependency of various features in steam turbine performance.
* The LSTM layer gives two outputs:
* **Hidden state:** This acts as carrier information about the current time step.
* The cell state will carry the long-term dependencies:.

**Fully Connected Layer (fc):**

* After the LSTM, the hidden state of the last time step is fed into a fully connected layer fc to output one single value for efficiency.
* This fully connected layer is utilized for aggregating the learned information towards the final prediction.

**4. Loss Function and Optimizer**

• Loss function:

The Mean Squared Error loss is used since the task is a regression problem that predicts the continuous value of "Efficiency.".

• Optimizer:

o Adam Optimizer: This is the torch.optim.Adam utilized for updating the weights of the model. Adam effectively combines the benefits of Adaptive Gradient (AdaGrad) and Root Mean Square Propagation (RMSProp); hence, it is efficient for training neural networks with complex datasets.

5. Training Process

• Train\_model() trains the Hybrid CNN-LSTM model in a range of epochs, in our application, 20.

• In every epoch, the model makes predictions for each batch of the training data, calculates the loss, and performs a backpropagation to adjust the weights.

• After each epoch, the training loss is printed out for monitoring purposes-to make sure that the model is learning.

6. Evaluation

• The function evaluate\_model() evaluates the model on the test dataset.

• During evaluation, the model is set to inference mode: model.eval(), which disables all dropout and batch normalization layers to ensure consistency.

• Predictions and actual values are collected for further analysis.

7. Measures of Accuracy

• Mean Absolute Error : The MAE is the average amount by which the model's predictions over- or undershoot from the real efficiency.

• Metric Accuracy: 100 minus the average of absolute error. In cases where targets are pre-normalized, an accuracy metric will be computed and stands for understanding how well the model is performing.A graph showing the growth of a hybrid model

Description automatically generated with medium confidence

# **Conclusion:**

This work has used both RNN and the hybrid CNN-LSTM models for the prediction of the air quality levels. Performance comparisons done show that a hybrid CNN-LSTM model outperformed when using RNN features independently. This clearly showed how combined temporal and spatial information in environmental predictions plays a very important role in offering the best results one requires. Feature importance analysis has identified PM2.5 and NO2 as major contributors towards AQI values. It, therefore, becomes very important and specific that regulatory actions be developed according to the emission of those pollutants.

**Bibliography:**

**1. Brownlee, J.** (2017). *Deep Learning for Time-Series Forecasting*. Machine Learning Mastery. <https://machinelearningmastery.com>

**2. LeCun, Y., Bengio, Y., & Hinton, G.** (2015). Deep learning. *Nature*, *521*(7553), 436–444. <https://doi.org/10.1038/nature14539>

**3. Hochreiter, S., & Schmidhuber, J.** (1997). *Long Short-Term Memory*. Neural Computation, 9(8), 1735-1780. <https://doi.org/10.1162/neco.1997.9.8.1735>

**4.** *Deep residual learning for image recognition*. **He, K., et al.** (2016, June 1). IEEE Conference Publication | IEEE Xplore. <https://ieeexplore.ieee.org/document/7780459>

**5. Bengio, Y., Simard, P., & Frasconi, P.** (1994). Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5(2), 157-166. <https://ieeexplore.ieee.org/document/279181>

**6. Cho, K., et al.** (2014). *Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation*. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). <https://doi.org/10.3115/v1/D14-1179>

**7. Kingma, D. P., & Ba, J.** (2014, December 22). *Adam: A method for stochastic optimization*. arXiv.org. <https://arxiv.org/abs/1412.6980>

**8. Goodfellow, I., Bengio, Y., & Courville, A.** (2016). *Deep learning*. (n.d.). MIT Press. <https://www.deeplearningbook.org/>

**9. Rahman, A., et al.** (2021). *Efficient Cooling System Performance Prediction Using Hybrid Deep Learning Models*. Renewable and Sustainable Energy Reviews, 140, 110752. <https://doi.org/10.1016/j.rser.2021.110752>

**10. Patel, M., et al.** (2022). *Predictive Analytics for Energy Management in Power Plants Using CNN-RNN Models*. Energy Reports, 8, 459-471. <https://doi.org/10.1016/j.egyr.2022.01.012>