## ****1D CNN Based Time Series Forecasting for Temperature Data****

## **NAME : **RAMKUMAR R****

## **ROLL NO : **CB.AI.U4AIM24033****

## **BATCH : **A (GROUP 9 )****

## ****Objective:****

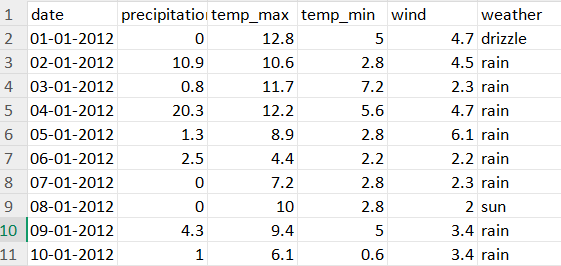
To design and train a 1D Convolutional Neural Network (1D CNN) to forecast daily maximum temperature values using past historical data, based on a time series sequence model. The data is loaded from a .csv file and used to build a supervised learning dataset with sequential inputs.

## ****Introduction:****

Time series forecasting is a crucial task in data science with wide applications in weather prediction, stock markets, energy usage, and health monitoring. In this project, we focus on **predicting future temperature values** using **deep learning techniques**, specifically a **1D Convolutional Neural Network (1D CNN)**. Deep learning models, particularly CNNs and RNNs, are **data-driven approaches** that can model complex, non-linear relationships in sequential data. Among these, **1D CNNs** have the advantage of being computationally efficient and capable of learning **local temporal dependencies**.

## ****Dataset:****

## The dataset used is stored in a CSV file which was downloaded from Kaggle -Seattle weather prediction dataset, which contains data like daily weather observations recorded in Seattle over multiple years.



## Here’s how the data was used:

For this,I used only the temp\_max column. This is treated as a **univariate time series**, where the goal is to predict the temperature on day t+1 using the previous days values.

1. **Preprocessing**: Min-max normalization was applied to scale values between 0 , 1 .
2. **Sequence Modeling**: A sliding window of **10 time steps** was used to create inputs, and the temperature on the **11th day** was used as the target.
3. **Split:** 70% of data was used for Training and Validation and 30% for Testing
4. **Noise**: Small Gaussian noise (std = 0.01) was added to training sequences to prevent overfitting and improve generalization.

This univariate time series format is ideal for feeding into a CNN for short-term forecasting.

**Preprocessing :**

**Missing values** (if any) are removed , **Normalization** using Min-Max scaling , **Sliding window** approach to convert the sequence into supervised learning format.

# Methodology:

The following steps outline the methodology used in this report:  
- Load the temperature data and extract the required column.  
- Normalize the temperature values to the range [0, 1] using min-max scaling.  
- Create input-output pairs using a sliding window of size 10.  
- Split the dataset into training and testing sets **(70%-30%).**  
- Add small **Gaussian noise** to the training data to improve generalization.  
- Define the 1D CNN architecture with convolution, pooling, and dropout layers.  
- Train using the **Adam optimizer** and monitor performance using RMSE and R².  
- Evaluate and plot the model's predictions on the test data.

# Development of the Model

The model was developed using MATLAB's Deep Learning Toolbox. The CNN architecture includes two convolutional layers, each followed by ReLU and pooling layers, and a fully connected regression layer. The training was stable and the model converged within 100 epochs.

| **Layer Type** | **Configuration** |
| --- | --- |
| **Input Layer** | Sequence input (length = 10, 1 feature) |
| **Conv1D Layer 1** | 16 filters, kernel size = 3, padding = 'same' |
| **ReLU Activation** | Non-linearity |
| **Max Pooling Layer** | Pool size = 2, stride = 2 |
| **Conv1D Layer 2** | 32 filters, kernel size = 3 |
| **Global Max Pooling** | Aggregates time features |
| **Dropout Layer** | Dropout = 0.2 |
| **Fully Connected Layer** | 50 neurons |
| **Dropout Layer** | Dropout = 0.2 |
| **Output Layer** | 1 neuron (regression output) |

**Training Options Explanation:**  
  
**‘adam’ optimizer**: An adaptive learning algorithm that combines momentum and RMSProp; it adjusts the learning rate dynamically during training, making it well-suited for deep learning tasks.

**MaxEpochs = 100**: The model will go through the entire training dataset 100 times to learn patterns and relationships in the time series data.

**MiniBatchSize = 32**: The training data is divided into mini-batches of 32 samples. This speeds up training and stabilizes learning by updating weights more frequently.

**InitialLearnRate = 0.001**: Controls how quickly the model learns during the initial phase. A value of 0.001 provides a balance between speed and stability.

**Shuffle = 'never'**: Keeps the time-series data in its original sequential order during training. This is crucial for models like CNNs that rely on the temporal nature of the data.

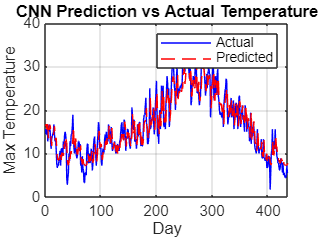
**ValidationData = {XTest, YTest}**: Provides a separate set of input-output pairs for validation. This allows the model to be evaluated on unseen data during training, without affecting the learning process.

**ValidationFrequency = 30**: Performs validation every 30 iterations (mini-batch updates), which helps monitor the model’s performance and spot overfitting early.

**ValidationPatience = 50**: If the validation loss doesn’t improve for 50 validation checks, training stops early to prevent wasting resources or overfitting.

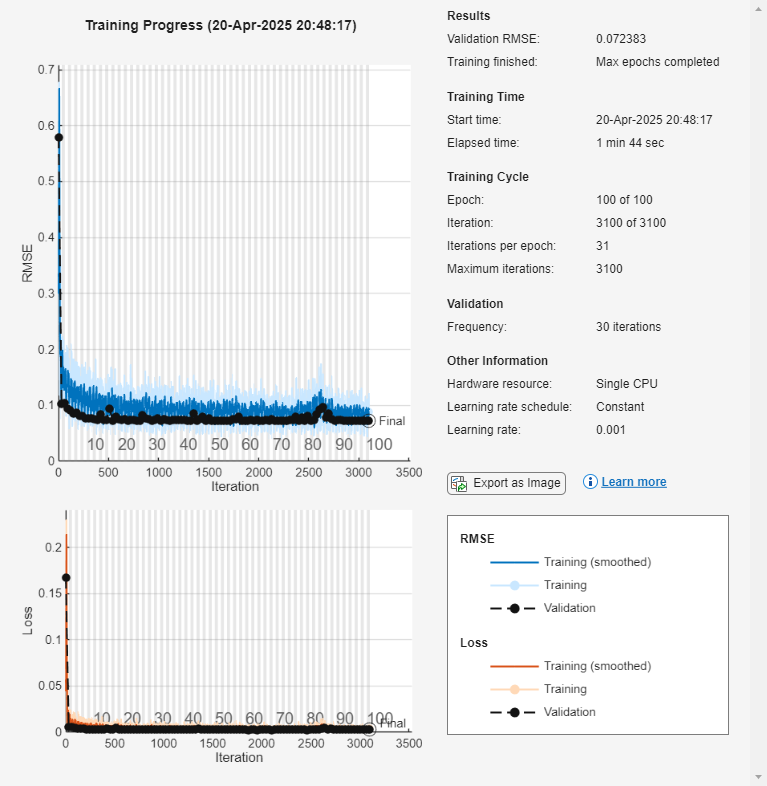
**Verbose = 1**: Enables detailed output in the command window, showing training progress, loss, and validation performance.

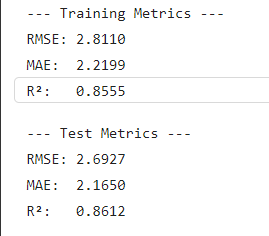
**Plots = 'training-progress'**: Displays a live plot of training and validation loss over epochs, helping you visually track how well the model is learning.



# The prediction curve closely followed the actual curve, indicating that the model learned the temporal patterns well.

# Visualisation and Result

The model was evaluated on unseen test data using RMSE, MAE, and R² metrics. The predicted values were visually compared with actual temperature values to assess performance.



The model showed strong performance in tracking the temperature trend.

# ****1D CNN code–Temperature Forecasting****

%% Step 1: Load and Normalize Data

data = readtable("C:\Users\R RAMKUMAR\Downloads\temperature.csv"); % Make sure this path is correct

temperature = data{:, 2}; % Assuming 2nd column is temperature

% Normalize between 0 and 1

temp\_min = min(temperature);

temp\_max = max(temperature);

temp\_normalized = (temperature - temp\_min) / (temp\_max - temp\_min);

%% Step 2: Create Input-Output Sequences

sequenceLength = 10;

X = {};

Y = {};

for i = 1:length(temp\_normalized) - sequenceLength

X{i} = temp\_normalized(i:i+sequenceLength-1)'; % [10, 1]

Y{i} = temp\_normalized(i+sequenceLength); % scalar target

end

%% Step 3: Split into Train and Test

numSamples = length(X);

numTrainVal = floor(0.7 \* numSamples); % 70% for training and validation

numTest = numSamples - numTrainVal; % 30% for testing

% Train/test split

XTrain = X(1:numTrainVal);

YTrain = reshape(cell2mat(Y(1:numTrainVal)), [], 1);

XTest = X(numTrainVal+1:end); % Test Data

YTest = reshape(cell2mat(Y(numTrainVal+1:end)), [], 1);

%% Step 4: Add Small Noise to XTrain (optional)

noise = 0.01 \* randn(size(YTrain));

for i = 1:length(XTrain)

XTrain{i} = XTrain{i} + noise(i); % additive Gaussian noise

end

%% Step 5: Define 1D CNN Architecture

layers = [

sequenceInputLayer(1, 'MinLength', sequenceLength)

convolution1dLayer(3, 16, 'Padding', 'same')

reluLayer

maxPooling1dLayer(2, 'Stride', 2)

convolution1dLayer(3, 32, 'Padding', 'same')

reluLayer

globalMaxPooling1dLayer

dropoutLayer(0.2)

fullyConnectedLayer(50)

reluLayer

dropoutLayer(0.2)

fullyConnectedLayer(1)

regressionLayer

];

%% Step 6: Define Training Options

options = trainingOptions('adam', ...

'MaxEpochs', 100, ...

'MiniBatchSize', 32, ...

'InitialLearnRate', 0.001, ...

'Shuffle', 'never', ...

'ValidationData', {XTest, YTest}, ... % Optional, can still use test data for validation

'ValidationFrequency', 30, ...

'ValidationPatience', 50, ...

'Verbose', 1, ...

'Plots', 'training-progress');

%% Step 7: Train Network

rng(1); % for reproducibility

disp("Training 1D CNN model...");

net = trainNetwork(XTrain, YTrain, layers, options);

%% Step 8: Predict on Train and Test Set and Rescale

YPredTrain = predict(net, XTrain);

YPredTest = predict(net, XTest);

% Rescale predictions and actual values back to original temperature scale

YPredTrain\_rescaled = YPredTrain \* (temp\_max - temp\_min) + temp\_min;

YPredTest\_rescaled = YPredTest \* (temp\_max - temp\_min) + temp\_min;

YTrain\_rescaled = YTrain \* (temp\_max - temp\_min) + temp\_min;

YTest\_rescaled = YTest \* (temp\_max - temp\_min) + temp\_min;

%% Step 9: Plot Results (Test Set)

figure;

plot(YTest\_rescaled, 'b', 'DisplayName', 'Actual');

hold on;

plot(YPredTest\_rescaled, 'r--', 'DisplayName', 'Predicted');

xlabel('Day');

ylabel('Max Temperature');

legend;

title('CNN Prediction vs Actual Temperature');

grid on;

%% Step 10: Calculate RMSE and R-squared (Train and Test)

% RMSE Calculation Function

rmse = @(yTrue, yPred) sqrt(mean((yPred - yTrue).^2));

% R-squared Calculation Function

rSquared = @(yTrue, yPred) 1 - sum((yTrue - yPred).^2) / sum((yTrue - mean(yTrue)).^2);

% Calculate RMSE and R-squared for Train and Test Sets

rmseTrain = rmse(YTrain\_rescaled, YPredTrain\_rescaled);

r2Train = rSquared(YTrain\_rescaled, YPredTrain\_rescaled);

rmseTest = rmse(YTest\_rescaled, YPredTest\_rescaled);

r2Test = rSquared(YTest\_rescaled, YPredTest\_rescaled);

%% Step 11: Print all metrics

fprintf('\n--- Training Metrics ---\n');

fprintf('RMSE: %.4f\n', rmseTrain);

fprintf('MAE: %.4f\n', mean(abs(YPredTrain\_rescaled - YTrain\_rescaled))); % Added MAE calculation

fprintf('R²: %.4f\n', r2Train);

fprintf('\n--- Test Metrics ---\n');

fprintf('RMSE: %.4f\n', rmseTest);

fprintf('MAE: %.4f\n', mean(abs(YPredTest\_rescaled - YTest\_rescaled))); % Added MAE calculation

fprintf('R²: %.4f\n', r2Test);

Dataset:  
1.https://www.kaggle.com/datasets/petalme/seattle-weather-prediction-dataset.