The Report is Generated by DrillBit AI Content Detection Software

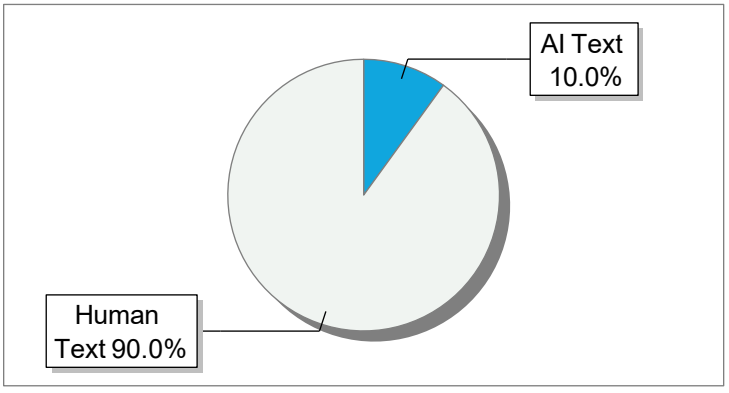
Submission Information

Author Name	Akhil Haldar, Anirban Halder, Subham Patra..
Title	Weather Forecasting using DNN
Paper/Submission ID	3851212
Submitted By	hod.cse@rcciit.org.in
Submission Date	2025-06-18 11:46:21
Total Pages	53
Document type	Thesis

Result Information

AI Text: **10 %**

Content Matched



Disclaimer:

* The content detection system employed here is powered by artificial intelligence (AI) technology.
* Its not always accurate and only help to author identify text that might be prepared by a AI tool.
* It is designed to assist in identifying & moderating content that may violate community guidelines/legal regulations, it may not be perfect.

DrillBit

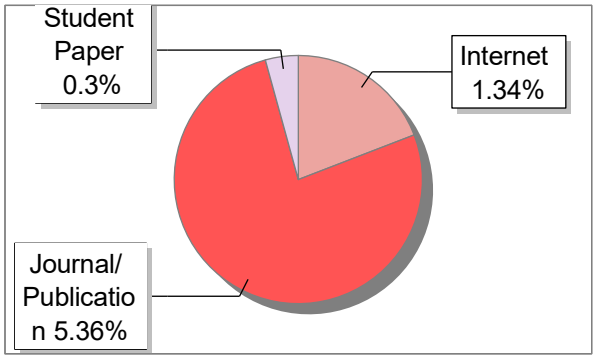
The Report is Generated by DrillBit Plagiarism Detection Software

Submission Information

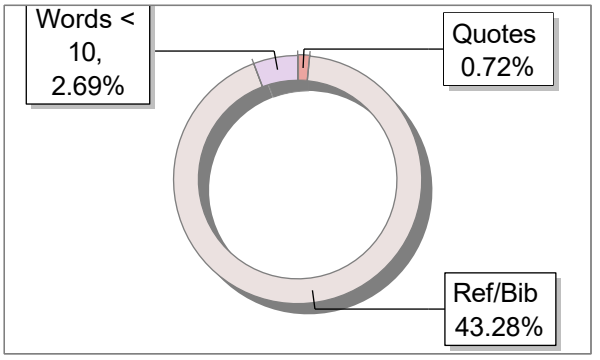
Author Name	Akhil Haldar, Anirban Halder, Subham Patra, Ritwick Ghosh
Title	Weather Forecasting using DNN
Paper/Submission ID	3851212
Submitted by	hod.cse@rcciit.org.in
Submission Date	2025-06-18 11:46:21
Total Pages, Total Words	53, 7742
Document type	Thesis

Result Information

Similarity **7 %**



Sources Type



Report Content

Exclude Information

Quotes	Excluded
References/Bibliography	Excluded
Source: Excluded < 10 Words	Excluded
Excluded Source	**0 %**
Excluded Phrases	Not Excluded

Database Selection

Language	English
Student Papers	Yes
Journals & publishers	Yes
Internet or Web	Yes
Institution Repository	Yes

ABSTRACT

Weather prediction plays a big role in supporting sustainable development by helping reduce the damage caused by extreme weather events. These events can otherwise slow down progress by many years. To make accurate predictions, it's important to understand both the patterns over time and how different weather factors affect each other across different locations. This project uses machine learning to forecast weather based on hourly weather data collected during three main seasons: summer, winter, and monsoon. The model is trained using this data and then used to predict future weather conditions by learning seasonal and hourly trends. Accurate forecasts are becoming more important for areas like farming, energy use, protecting the environment, and planning everyday activities. This study introduces a hybrid model that combines Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. CNNs are used to extract key patterns from spatial (location-related) data, while LSTMs are designed to understand and track how the data changes over time. Together, they improve the accuracy and stability of predictions. The model uses Mean Absolute Error (MAE) and RMSE to measure performance and is able to handle challenges like missing data and large, complex datasets effectively.

CONTENTS

LIST OF SYMBOLS

Symbol	Meaning	Page No.
€t	Error Time	16
σ	Transition Function	16
Ψ	Lag Polynomial	16
Φ	Autoregressive coefficient	16
γ	The kernel dimension	14

LIST OF ABBREVIATIONS

Abbreviation	Full Form / Meaning
CNN	Convolution Neural Network
DNN	Deep Neural Networks
LSTM	Long Short-Term Memory
RNN	Recurrent Neural Networks
ANN	Artificial Neural Networks
MSE	Mean Squared Error
MAE	Mean Absolute Error
RMSE	Root Mean Squared Error

LIST OF FIGURES

2.1 INTRODUCTION

Weather forecasting is a cornerstone of modern society, enabling the prediction of critical atmospheric conditions such as temperature, rainfall, humidity, and wind patterns. Accurate weather forecasting allows us to understand disaster management, right time of agriculture, future weather predictions, sudden weather fluctuations etc. Also through weather prediction we can get alert from any type of natural disaster and can ensure human safety. Farmers also dependent on weather forecast for better understanding of planting and harvesting.

[4]Machine learning and more notably deep learning-based approaches are emerging techniques in AI-based data analysis. This technique take the data analytical processes into another level and models built are data-driven rather than model-driven. For instance, a convolution-based neural network (CNNs) is preferable for problems such as image recognition, while RNNs are better suited for problems like time series data analysis.
[5]For extracting features, we get some variables that contain information from historical records. This dataset inserts basic variables as raw historical prices, technical indicators, or fluctuations of those variables. For input space and possible complexity of the feature space that needs a good prediction, like a CNN deep learning algorithm, which will be a favorable technique for feature extraction.

CNN is used for extracting features from the input data matrix by using filters, which helps in analyzing the weather features over the given location or region.

LSTM networks are designed and implemented in the model as they help in capturing the long-term dependencies between the data.

1.2 LITERATURE REVIEW

Sl No	Author Name	Paper Name	Contribution	Challenges
1	[2] Schizas et al.	Weather Forecasting using machine learning methods	Each neural network model is used to learn local geographical effects effectively and capture the non linearity of cloud behaviour. It also used Kick-out algorithm for speedup the training	Training large no of model is very expensive with respect to computation. ML models can't have the feature of explainability which is very essential in weather forecasting .this issue might sacrifice the predictive performance.
2.	[3]Ochiai et al.	Snowfall and weather forecasting by using ANN	build a 3-layer feedforward ANN model to feed the cloud dynamics and reduced the prediction error in test set by 20%. It also introduce the Kick-Out Algorithm for speedup the training process.	For Training thousand of data in parallel, it required large no of resources leads to to high computational cost.
3	[4] Tavakoli et al	The Performance of LSTM and BiLSTM in Forecasting Time Series	Conducts a comparison between unidirectional LSTM and Bidirectional LSTM, where BI-LSTM upto 30% in certain cases is proved to be more accurate in multivariate time series forecasting and multi input series forecasting.	Bi-LSTM converges more slowly compared to LSTM, and thus may require more training data compared to LSTM to reach the same level of loss. It updates its weight on more information compared to LSTM which may not be taken in account in all types of usages.
4	[5] Ehsan et al.	CNN-based stock market prediction using a diverse set of variables Author: Ehsan Hoseinzade, Saman Haratizadeh	Introduced one of the first 2D CNN prediction models, for financial forecasting and extracting relevant features from multiple input series.	Depending on how the variables are treated, a badly designed CNN pipeline leads to a worse result than a shallow ANN
5	[6]Sadeque et al.	A Deep Learning Approach to Predict Weather Data Using Cascaded LSTM Network	Cascading lightweight lstm model with less computation power required and better prediction accuracy compared than deep 1D CNN networks or regular lstm networks	Fluctuation of the windspeed variable compared to others, creates a gap in accuracy.
6	[7]Kreuzer et al.	Short-term temperature forecasts using a convolutional neural network — An application to different weather stations in Germany	Emphasizes the impact of the order of features in multivariate time series and generalizes the model better by shuffling the order of days in the time series. adding extra features like days and months to compensate the loss of seasonality	Simpler and more computationaly inexpensive models may outperformed the proposed cnn lstm hybrid model for smaller forecasting horizons.

2.1 Workflow for Weather Forecasting using DNN Techniques

2.1.1 Understanding the Problem

Weather forecasting is a complex endeavor, with numerous factors determining the weather, being numerous variations of weather depending on physical features like location and altitude. These predictions are crucial for various sectors, including agriculture, energy management, and disaster preparedness, where precise weather information ensures better planning and resource utilization. The disadvantages of complex statistical models are exacerbated with numerous features and the seasonal nature of the weather conditions. Here, DNNs can play a crucial role in discovering connections between factors that may not be visible in traditional statistical models and forecasting methods, leading to improved accuracy in forecasting.

Our proposed solution integrates CNNs and LSTMs. CNNs are utilized to extract spatial features from weather data, while LSTMs capture temporal dependencies, enabling the model to account for both geographical patterns and sequential weather variations.

2.1.2 The dataset

We train multivariate time series data of a city in Maharashtra called Nagpur. It's situated a landlocked city without presence of nearby rivers in Central India. We got 25 features and the features are given below. From the data pool, We remove features like totalSnow_cm, uvIndex, uvIndex.1, sunrise, moonrise, sunset, moonset which are rather dependent on the physical location of the city rather than statistical or dynamic features of weather. This is done in order to trim the dataset for easier computation before processing the rest of the dataset.

SL No.	**FACTOR**	**UNIT**	**DESCRIPTION**
1	maxtemp	Celsius	Maximum temperature recorded a day
2	mintemp	Celsius	Minimum temperature recorded in a day
3	Time of sun	Hour	The time of sun rising.
4	Moon illumination	Lux	The Intensity of the moonlight.
5	Dewpoint	Celsius	The temperature at which water vapor condenses into water.
6	Feels Like	Celsius	The temperature which human skin actually feels.
7	Heat Index	Celsius	The temperature which Combines air temperature and humidity.
8	Windchill	Celsius	The feels like temperature which human skin feels when wind is blowing
9	Wind gust	Kilometer per hour	The sudden increment of wind speed.
10	Cloud cover	percentage	The percentage of sky area covered by cloud.
11	Humidity	percentage	The amount of moisture in the air.
12	Precipitation	Millimeter	The amount of rain.
13	Pressure	Millibar	The weight of air on the earth’s surface.
14	Temperature	Celsius	How much hot and cool the air is.
15	Visibility	kilometer	The distance we can clearly see ahead.
16	Wind Direction	Degree	The direction from which air is blowing.

17	Windspeed	Kilometer per Hour	The speed of wind.

2.2. METHODOLOGIES OF IMPLEMENTATION

2.2.1. Metric Analysis

Mean Squared Error (MSE) measures the average of the squares of differences between predicted and actual values.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is the square root of the Mean Squared Error that occurs between the predicted and actual values.

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Mean Absolute Error (MAE) is the mean of the absolute value of the error that occurs between the predicted and actual values.

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

2.2.2. Neural Networks

The main factor underlying the success of a machine learning algorithm is to find relevant features and the appropriate model. Thereby, today, neural networks are a powerful solution in many applications when conventional machine learning models reach their limits. Especially CNNs introduced by [8] Lecun et al. (1998) are a solid alternative to classic approaches. Here, because we are dealing with time series, the network has to incorporate the temporal dependencies in the dataset. Hence, important temporal information should be carried along in time inside the neural networks. This can be achieved with RNNs. Thereby, LSTM networks, introduced by Hochreiter (1997), seem to be a promising approach, as LSTM networks solve the problem of exploding or vanishing gradients that RNNs often deal with Hochreiter, 1998).

2.2.3. Convolutional Neural Network Structures

We can use Neural networks for pattern recognition, for classification model, for prediction or regression models, or also for time series forecasting problems. CNN's had a big impact in machine learning as those networks have shown great success in many different visual and oral tasks, like image and video recognition as well as natural language processing. Therefore, many researchers explored the applications of the model to other domains.

Recently, CNNs have seen use in weather forecasting problems and it has outperformed LSTM networks. [7] In conventional machine learning techniques, features need to be defined and extracted manually based on knowledge of expert. This step is very computationally expensive, especially when considering correlations between input features of our data set. In contrast, raw data passes through CNN and extract important features during training process. The filters in convolutional layers learns the most relevant features from the input data. In general, CNNs are used for the detection of various image patterns, the input of a convolutional layer is assumed to be matrix pattern. Multi-variate time series data of length N with C channels with each channel representing one variable, therefore be either directly fed to the network as a one-dimensional vector of N elements with C network channels or as one image of size $C \times N$. Convolutional operations are applied to the time series data as well as across the channels, which leads to compute the feature maps based on multiple input channel data.

Thereby the image height represents a temporal factor, called window of the data and the image width represents the data channels. By applying a kernel $K \in \mathrm{R}\beta\times\gamma$ to a matrix, fresh data can be gathered, where the kernel dimension γ determines the incorporation between adjacent features. In the image I, $*$ operator defines convolution by t at position (a, b):

$$I_{filt}(a,b) = K * I(a,b) = \sum_{v=1}^{\beta} \sum_{v=1}^{\gamma} K(v,v) I(a-v, b-v).$$

The boundaries of the image must be treated separately, as neighbor pixels are missing, according to conventional filtering or convolution approaches, for example by using zero padding.

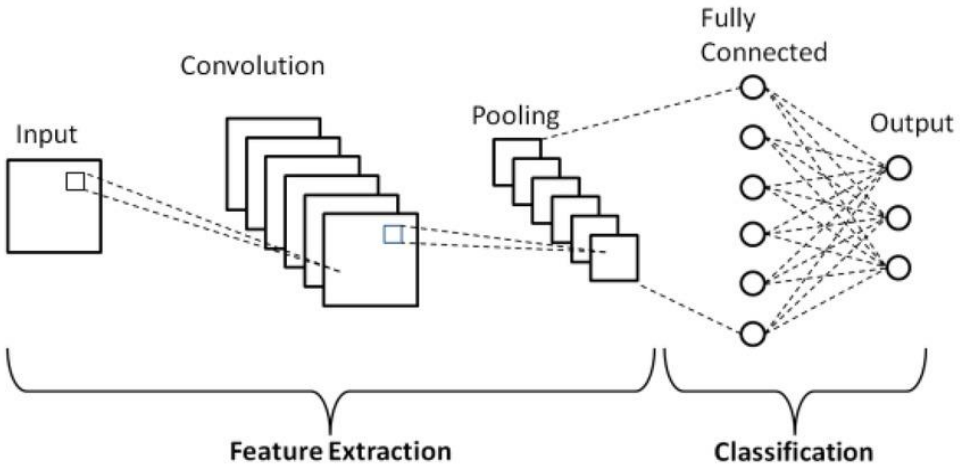


Figure – 1: Schematic Presentation of basic CNN Model[10]

2.2.3 LSTM Network Structures

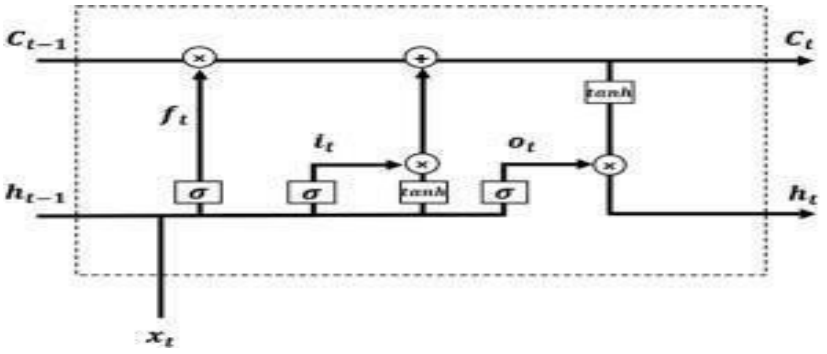


Figure – 2: Structure of a single LSTM cell, it includes an input gate (i) with activation vector $\dot{l}$t, an output gate o with activation vector $\boldsymbol{O}t$, and a forget gate f with activation vector .

LSTM networks provide the opportunity to deal with long time sequences. This is accomplished by using gates, which control the flow of data.

Fig.2 shows an LSTM cell includes an input gate i with its activation vector, an output gate o with its activation vector, and a forget gate f with activation vector ft. The input to the cell is Xt, which refers to a vector of observations at time t. In our case, since the LSTM is combined with the CNN, xt is the output vector from the CNN at time t. The variable Ct refers to the cell state and the output of the cell

Time represents t and ht is the related hidden state. For the transition function σ, the Rectified Linear Unit (ReLU)- function is used:

$\sigma(x) = \{0 \text{ for } x < 0, x \text{ for } x \geq 0$

$\sigma(x)$ refers to the activation function ReLU.

The gates name's already give an indication of their tasks. The work input gate works to add new data to the cell state according to:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i),$$
$$C_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C).$$

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f).$$

The LSTM's final output depends on both hidden state and the updated cell state.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o),$$
$$h_t = o_t \cdot tanh(C_t).$$

[9]Ordóñez & Roggen, 2016. In accordance with the usual notation, W and b are weights and biases belonging to the gates. For our new method, we combine a CNN and an LSTM network to form a ConvLSTM network, described in detail in Sections 2.2.1& 2.2.2

2.2 SOFTWARE & HARDWARE REQUIREMENTS

2.2.1 Software Requirements

The implementation of the weather forecasting model relies on various software tools and frameworks.

Here, Python is used as the primary programming language because it supports various important libraries and is also better for machine learning and other data manipulation work.

TensorFlow is the best framework for building, testing, and training DNN models. It also contains many supporting features.
Keras is a high-level API that streamlines the implementation process.

The project is being made and developed, and tested on Kaggle. It is a cloud-based, free platform where machine learning, data analysis, and data manipulation can be done. It allows a huge resource of data, provides notebooks for coding, and also gives tutorials. It provides a virtual GPU to run the model.
Libraries like NumPy, pandas, and Matplotlib are used for data processing, better visualization, manipulating, and cleaning the dataset.

2.1.1 Hardware Requirements

The model is trained using two NVIDIA T4 GPUs, each with 16GB VRAM. The computer requires a minimum of 16 GB of RAM, with 32 GB suggested for structured data controlling and processing.

At least 1 TB SSD is required for handling vast weather datasets and securing data-centric operations.

2.5 PROPOSED WORK

The data was first cleaned of irrelevant features to reduce the dimensions of the dataset. The yearly dataset is then split into multiple seasons, namely summer, winter, and monsoon. Summer is the months of March, April, May, and June. Monsoon corresponds to the months from July to October, and the rest of the year is winter.

After plotting the data, we see an outlier in the data on Sept 16, 2019, where all the features have the value 0. We replace the data of the whole day with the average of the last two days. Two extra features are added to the dataset, corresponding to the day and the month of the dataset. All the separate seasonal datasets are then split into a training and testing set, with an 80:20 ratio. We normalize the data using min-max normalization, mapping all the values into a domain [0,1]. This is important as features can have different scales of measurement, and certain features don't overpower each other, i.e., have more weight than others before the training.

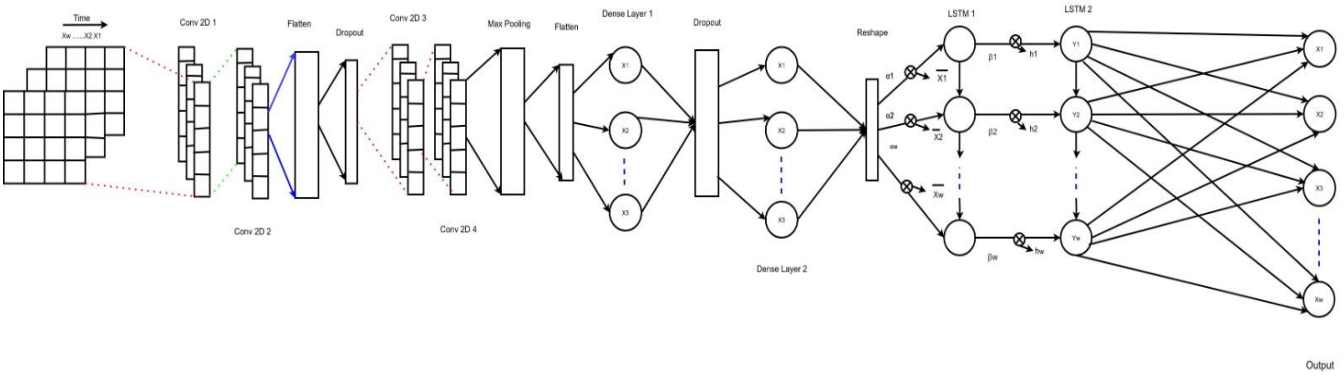
For ease of experimentation, we convert our datasets into samples of 24 timesteps and 15 features, with each timestep representing the reading of the sensors at the start of a distinct hour.
We seek to predict the hourly temperature of a full day, i.e., 00:00:00 to 23:00:00, i.e., 24 timesteps for one label while processing the measurements of the past 3 days. To achieve this, we create overlapping windows of 72 timesteps and another window of 24 timesteps, which follows 72 input timesteps. This is created for all of the data and is converted into a set of training inputs and training labels.

In the proposed model, we start with an input matrix of dimension 72 x 15, representing the 72 timesteps over three days and the 15 features we have taken in our dataset. First, we do a convolutional operation with eight filters of size 1 x 15, convoluting the 15 features into higher-level features and convoluting the

feature maps again over eight filters of size 3 x 1, trying to find a pattern over the features in sets of 3 hours. This is due to a pattern of 7 times of a day, i.e., midnight, morning, midday, afternoon, evening, dawn, dusk, but here we split into 8 times of a day instead of convoluting over 3 hours. We then carry out a max pooling of the features to downsize the feature maps. Adding a dropout of 20% to ensure better generalization and reduce overfitting.

We follow the output tensor and put it through another convolution over 8 filters of 3 x 1, based on the times of day. And then convolute another time over a standard kernel size of 3 x 3, which is heavily used in image processing, and then we pool again to down sample the feature maps.

The feature maps are then flattened. The flattened 1D vector is sent into a layer of dense neurons deep enough to hold all the elements of the feature maps, and the sigmoid activation is used to give negligible weight to irrelevant features. The output of dense neurons is then downsized, i.e., we pass it through a layer of 256 neurons to force the model to choose the relevant features. Then the output tensor is reshaped into a matrix of 4 x 64, to feed into an LSTM. Reshaping it this way reduces the forecasting horizon for the LSTM and experimentally produces better letters. We cascade two LSTM layers of 72 cells, which capture the dependencies over the 4 timesteps, and then have a final output of 24 neurons, each representing a timestep. Thus producing 24 labels as we intended



h Figure-2: Architecture of Proposed Model for training

SL No.	FEATURE NAME	UNIT	DESCRIPTION
1	tempC	Celsius	How much hot and cool the air is.
2	pressure	Millibar	The weight of air on the earth's surface.
3	Humidity	Percentage	The amount of moisture in the air.
4	maxtempC	Celsius	Maximum temperature recorded a day
5	mintempC	Celsius	Minimum temperature recorded in a day
6	FeelsLikeC	Celsius	The temperature which human skin actually feels.
7	HeatIndexC	Celsius	The amount of moisture in the air.
8	WindChillC	Celsius	The feels like temperature which human skin feels when wind is blowing
9	WindGustKmph	Kilometer per hour	The sudden increment of wind speed.
10	WinddirDegree	Degree	The direction from which air is blowing.
11	Windspeed	Kilometer per Hour	The speed of wind.
12	Day	Integer	Day of the month
13	Month	Integer	Integer value of the month

3.1 System Validation

MSE is used as a loss function during the training and validation phase. RMSE is also measured and depending on the RMSE improvements during the course of training, certain model weights are saved.

10% of the training set is used for validation, while training as the epochs, certain model weights of an epoch are stored in an array of model weights. The criteria for storing model weights is that it's at least 0.001 better than the model with minimum RMSE. At the end of training epochs, the weights of the models where the mean of RMSE i.e. training set RMSE and validation RMSE is the lowest, is restored to the layers of the model.

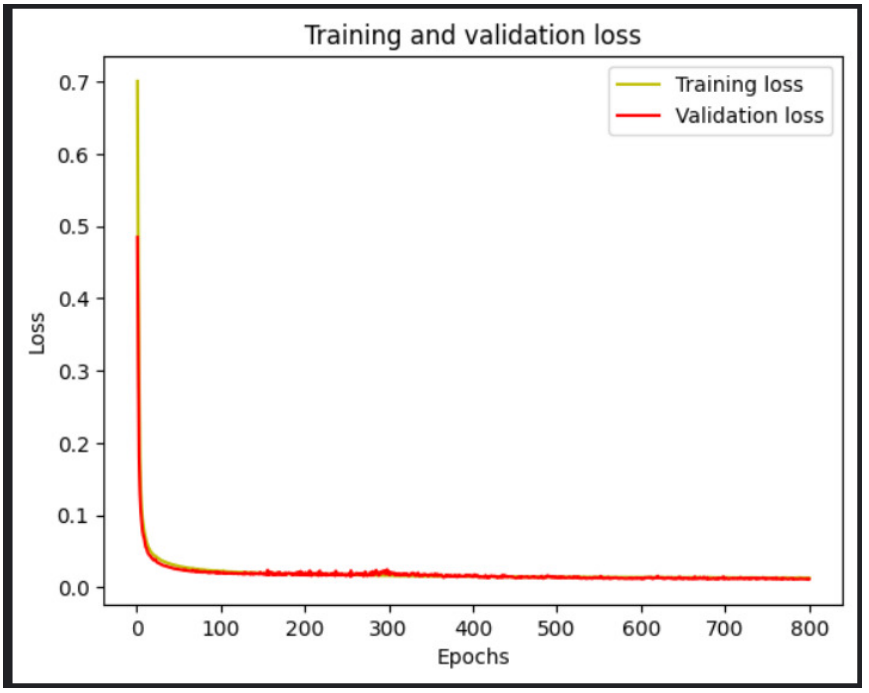


Figure - 4: Training and Validation Loss Curve of Summer model

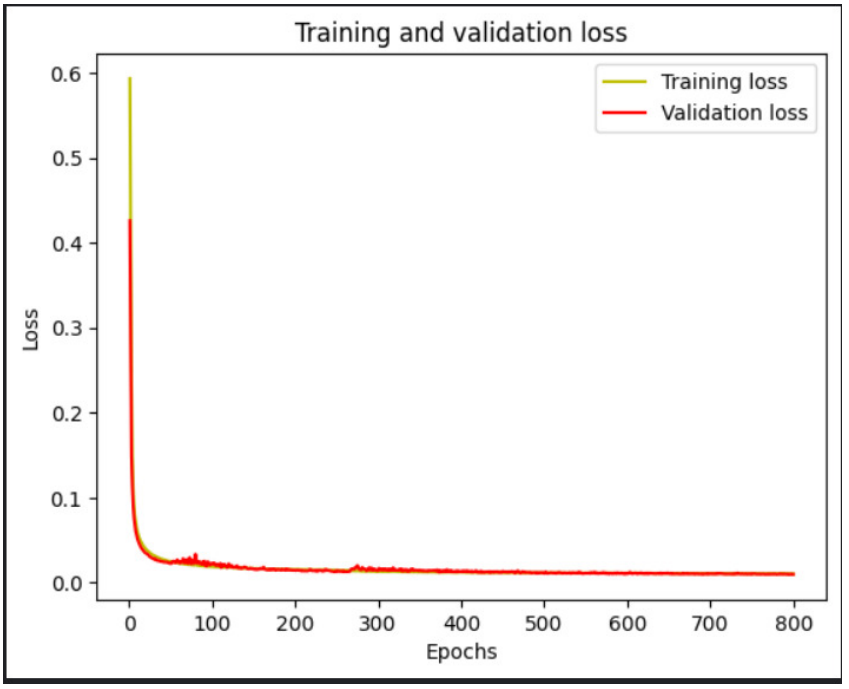


Figure - 5: Training and Validation Loss Curve of Monsoon model

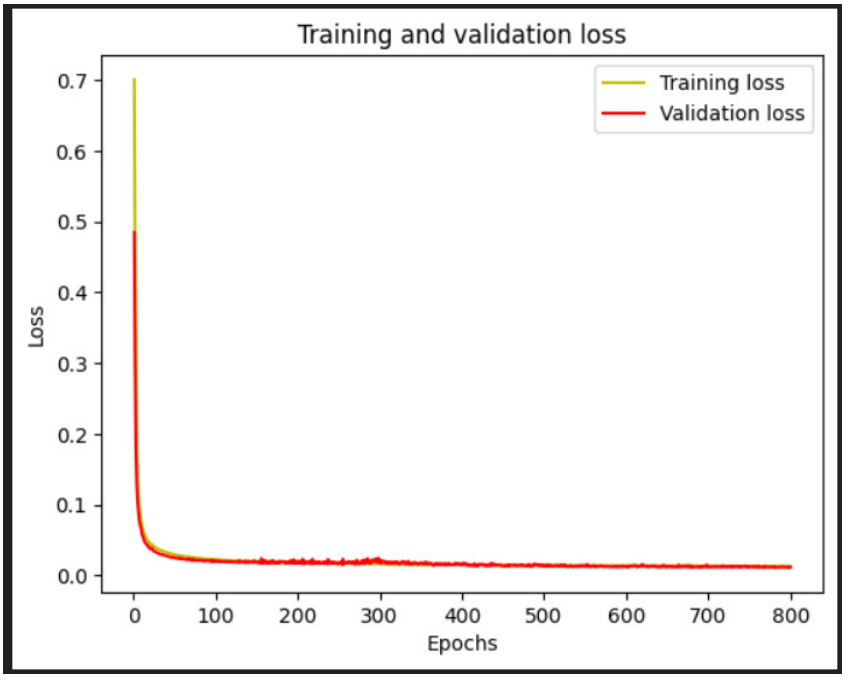


Figure - 6: Training and Validation Loss Curve of Summer model

3.2. Observed Output:

The output of the hybrid CNN model consists of predictions for 24 future timesteps with each time step signifying an hour of three critical meteorological features: and

temperature. These predictions provide a detailed forecast for each timestep, offering

insights into how these features are expected to evolve over time. The hybrid CNN leverages the strengths of convolutional layers to capture spatial and temporal dependencies in the data, resulting in accurate and robust predictions. This output can be utilized for weather forecasting, climate modeling, or other applications requiring precise short-term atmospheric condition predictions**.** The graphs below are the three different predictions for each season.

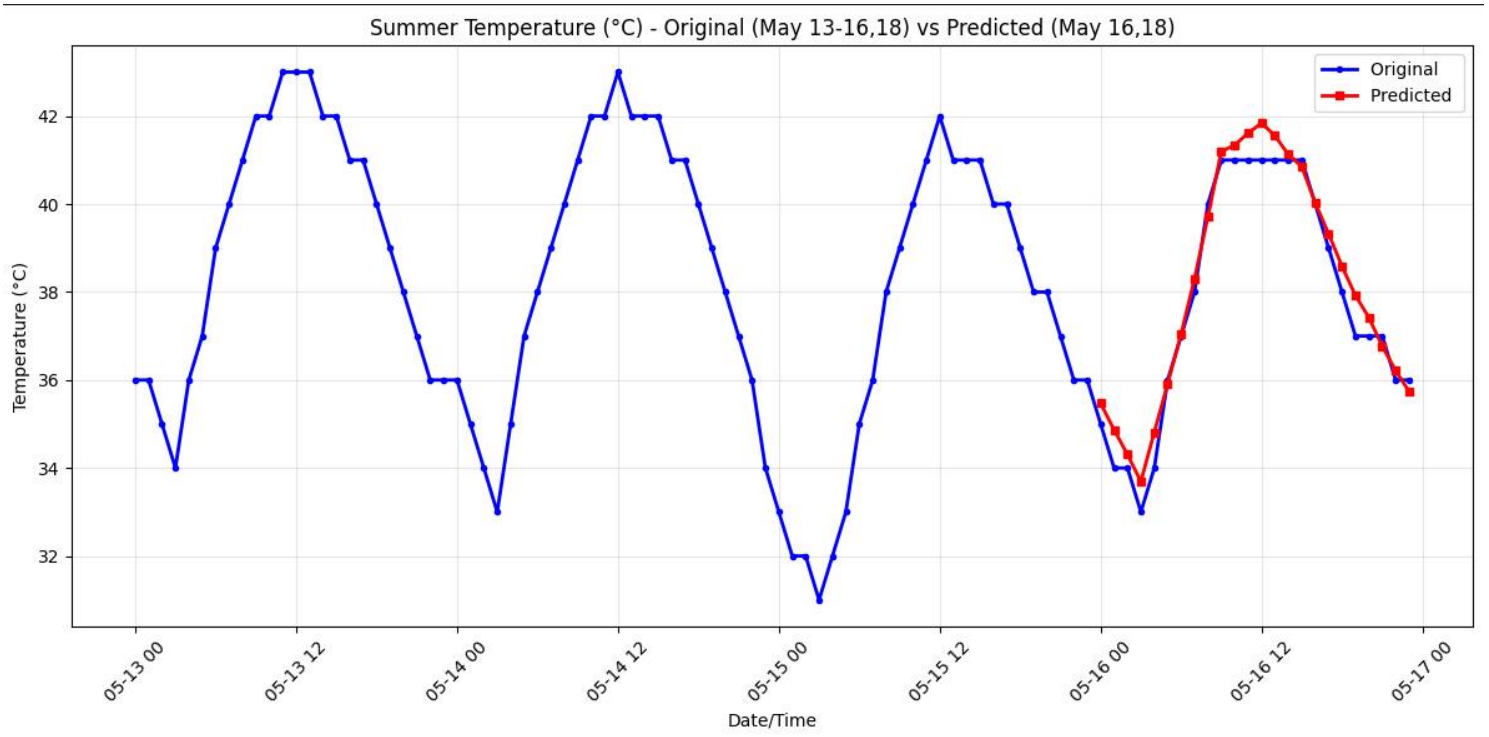


Figure-7: Comparison of Original and Predicted Temperature Of Summer

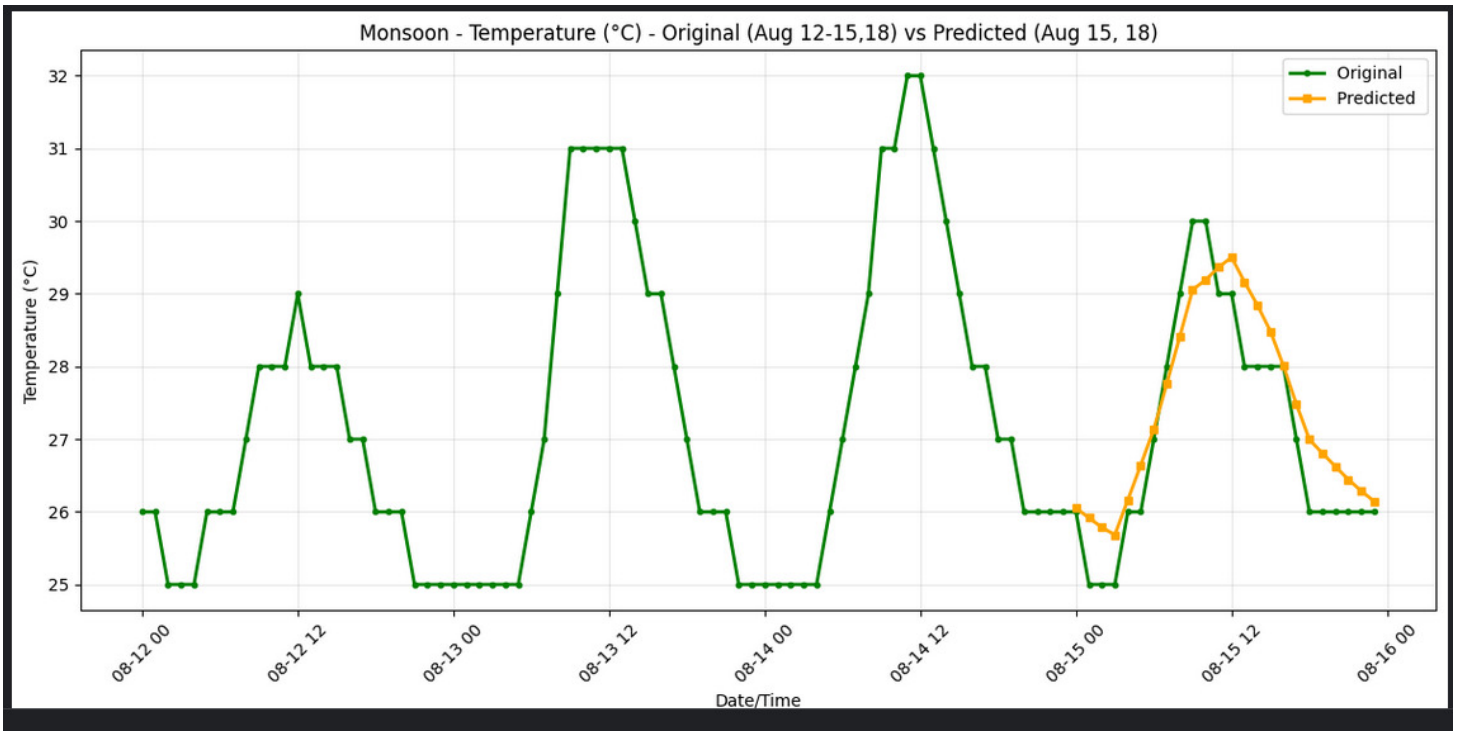


Figure – 8: Comparison of Original and Predicted Temperature Of Monsoon

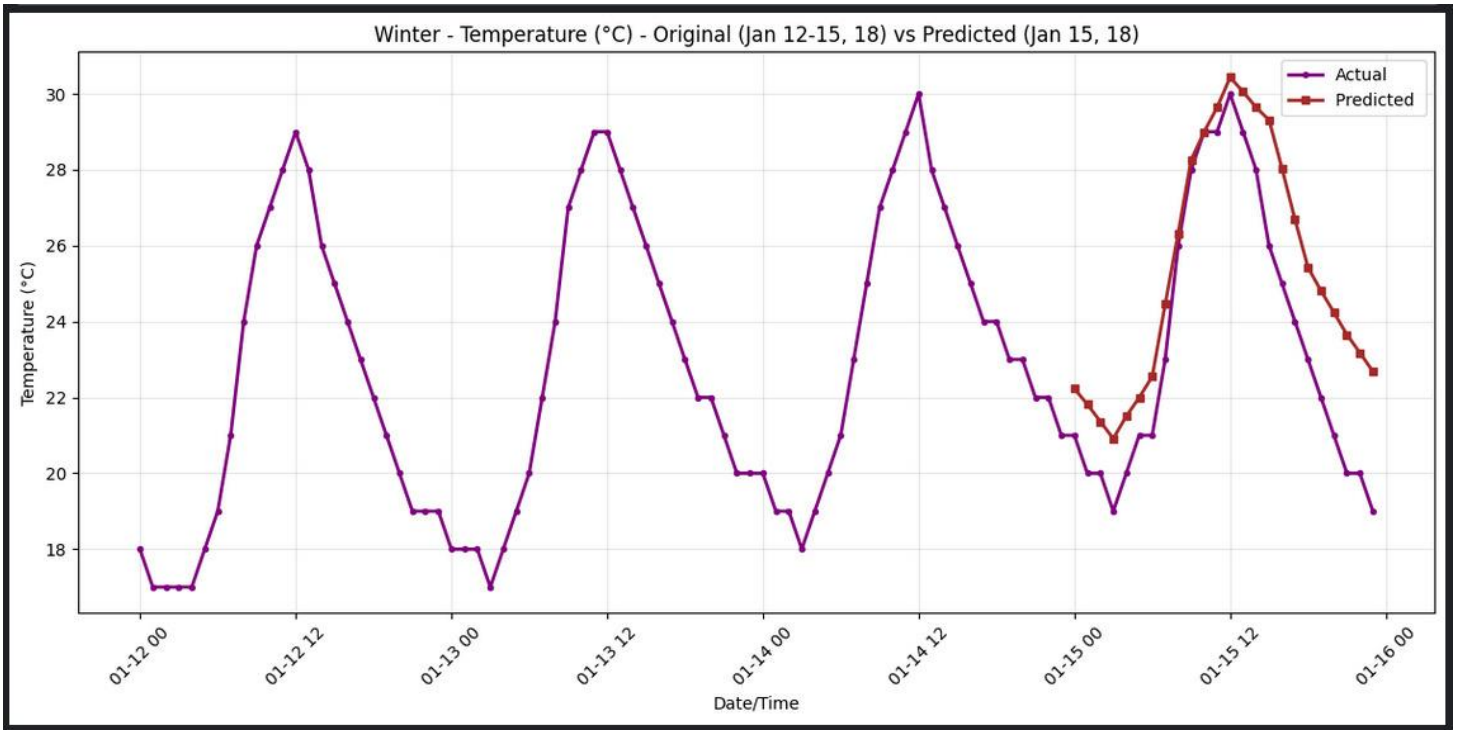


Figure – 9: Comparison of Original and Predicted Temperature Of Winter

3.3 PERFORMANCE ANALYSIS

The hybrid CNN-LSTM model demonstrated a remarkable ability to capture both spatial and temporal dependencies, leading to highly accurate weather predictions. This hybrid CNN-LSTM model is used to give the lower RMSE and MAE values across different seasons, which indicates the minimum deviation that takes place between the actual test set and predicted set.

In the training process, the model converged successfully and maintained the proper MAE values over 800 epochs.

After the prediction was done by a trained model, we plotted the graph for training and validation loss, which converged towards 0 as the epoch number increased.

In below figure-10,11,12 we show the deviation between actual and predicted values, which is minimum, and this is enough to justify the reliability and prediction capability of the model.

	SEASONS	RMSE	MAE
Proposed Model	Summer	2.20	1.69
	Winter	2.24	1.79
	Monsoon	1.68	1.28
Baseline Model	Summer	2.68	2.19
	Winter	2.27	1.70
	Monsoon	1.72	1.31

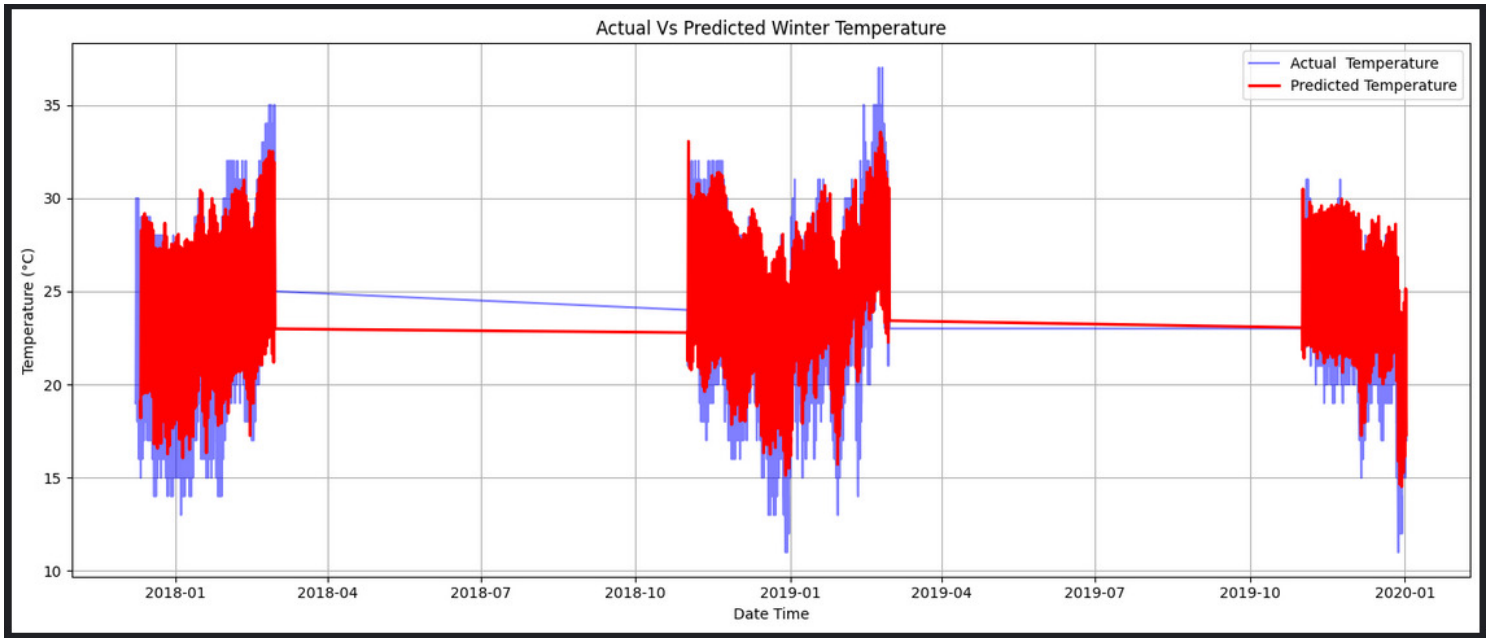


Figure – 10: Actual Vs Predicted Temperature for test set of Winter

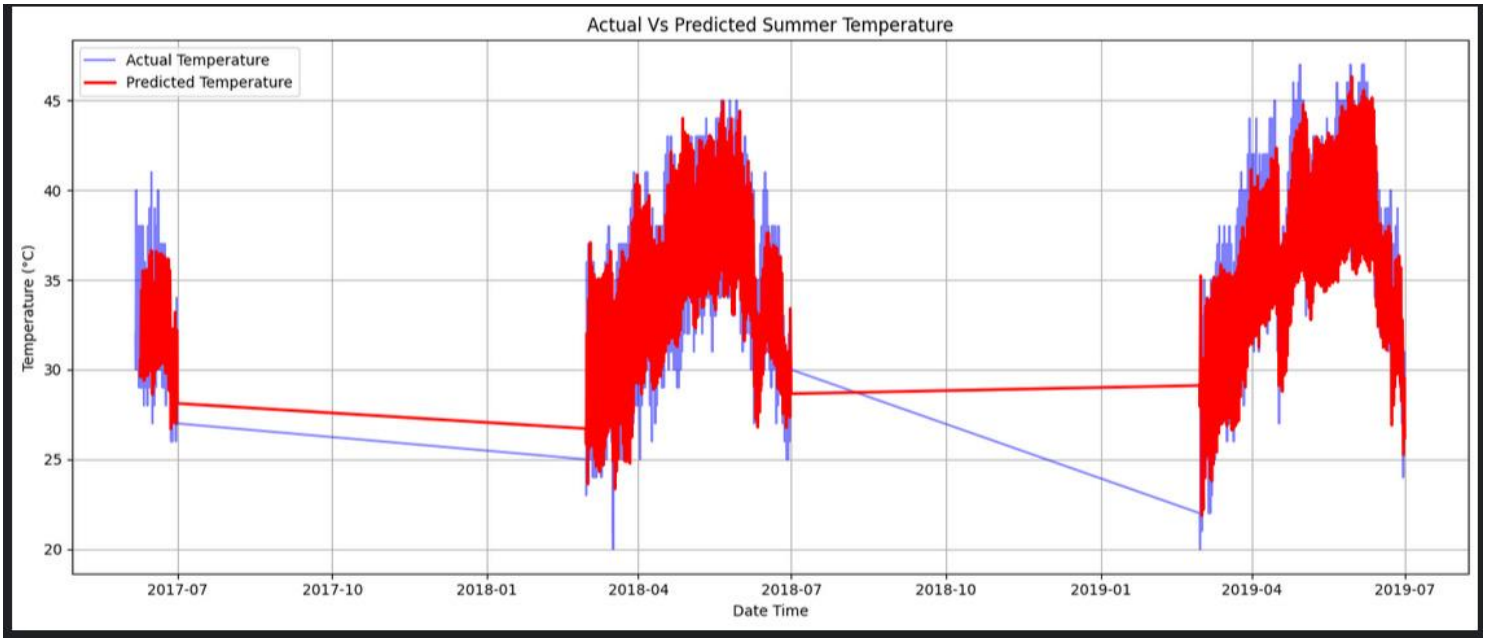


Figure – 11 : Actual Vs Predicted Temperature for test set of Summer.

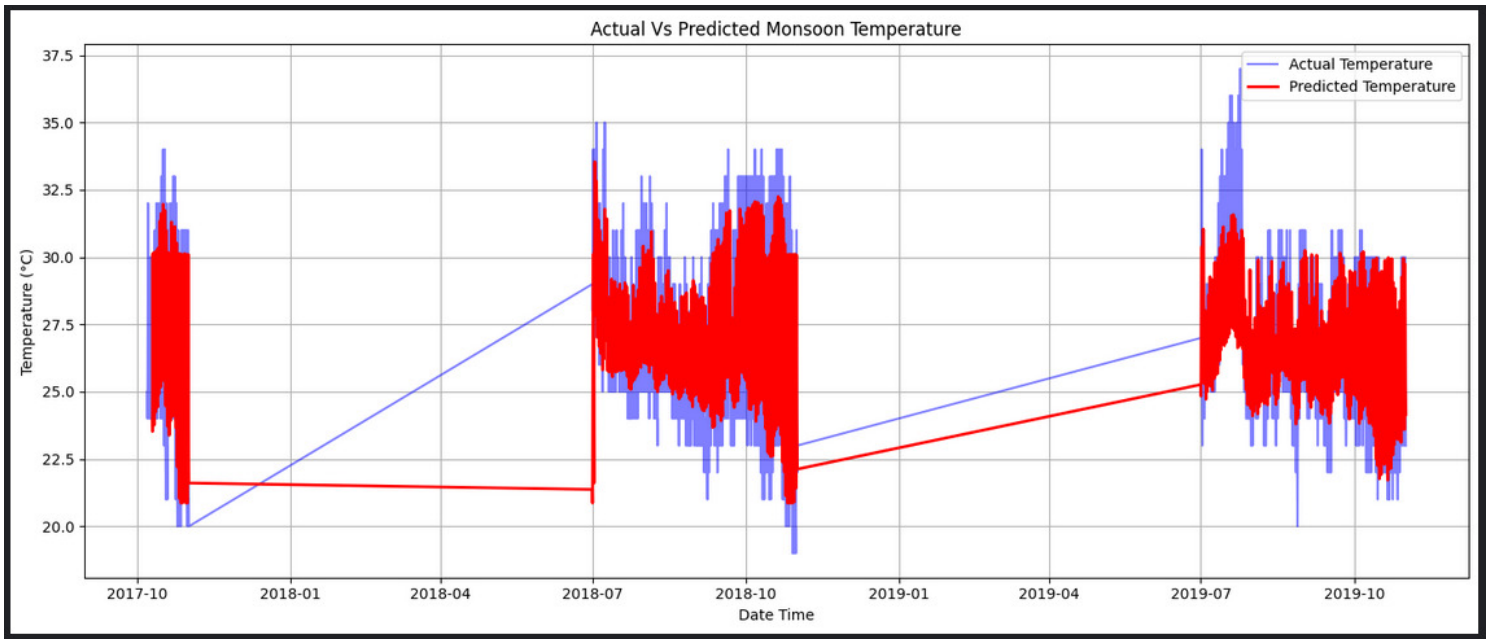


Figure -12: Actual Vs Predicted Temperature for the test set of Monsoon.

Conclusion

In conclusion, our project on weather forecasting using Deep Neural Network (DNN) techniques, particularly Long Short-Term Memory (LSTM) and Convolutional Neural Networks (CNN), has demonstrated the immense potential of AI-driven methodologies in advancing predictive accuracy and reliability. The LSTM models are dynamic and time-dependent in weather predictions. CNN is used for extracting spatial features, and it delivers a robust framework, pressure systems, and precipitation zones. Our project's main spotlight is better planning for the environment and overcoming problems that affect sectors like agriculture, aviation, and emergency management. Finally, from this Weather forecasting technique, we can reduce pivotal weather-related problems, and it contributes to environmental sustainability.

Future scope

The application of deep neural networks (DNNs) in weather forecasting represents a transformative advancement in meteorology. Exploring the integration of artificial intelligence and machine learning in weather forecasting for efficient prediction.

Develop explainable AI techniques that are lacking in deep learning techniques.
The inclusion of DNNs with the IoT advances enhances their utility. IoT devices, such as weather stations and mobile sensors, generate real-time data that DNNs can process to deliver hyper-localized forecasts. This real-time analysis in agriculture, aviation, and emergency management is beneficial, where timely and precise weather data is crucial.

Finally, the use of DNNs also extends to improving the visualization of weather data. By generating detailed and intuitive graphical representations, these systems make complex meteorological information accessible to both professionals and the general public, fostering better decision-making and awareness.

By leveraging their capabilities, meteorology can achieve unprecedented levels of accuracy and efficiency, addressing both immediate forecasting needs and long-term climate challenges.

References

1. Parry, M.; Canziani, O.; Palutikof, J.; van der Linden, P.; Hanson, C. Climate Change 2007: Impacts, Adaptation and Vulnerability. Available online: https://www.ipcc.ch/site/assets/uploads/2018/03/ar4_wg2_full_report.pdf (accessed on 30 October 2019).
2. Schizas, C., Michaelides, S., Pattichis, C., Livesay, R. (1991). In Proceedings of the Second International Conference on Artificial Neural Networks*, pp. 112–114. Springer, Heidelberg.
3. Ochiai K, Suzuki H, Shinozawa K, Fujii M, Sonehara N (1995) in Proceedings of ICNN'95 - International Conference on Neural Networks, vol. 2, pp. 1182–1187.
4. Reference: S. Siami-Namini, N. Tavakoli and A. S. Namin, "The Performance of LSTM and BiLSTM in Forecasting Time Series," 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 2019, pp. 3285-3292, doi: 10.1109/BigData47090.2019.9005997. keywords: {Biological system modeling;Training;Data models;Logic gates;Time series analysis;Predictive models;Recurrent neural networks}.
5. Ehsan Hoseinzade, Saman Haratizadeh, CNNpred: CNN-based stock market prediction using a diverse set of variables, Expert Systems with Applications, Volume 129, 2019, Pages 273-285, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2019.03.029.
6. Z. Al Sadeque and F. M. Bui, "A Deep Learning Approach to Predict Weather Data Using Cascaded LSTM Network," 2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), London, ON, Canada, 2020, pp. 1-5, doi: 10.1109/CCECE47787.2020.9255716.
7. Kreuzer, D., Munz, M., Schlüter, S. (2020). Short-term temperature forecasts using a convolutional neural network — An application to different weather stations in Germany. Machine Learning with Applications, 2, 100007. ISSN 2666-8270.
8. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998. doi: 10.1109/5.726791
9. F. J. Ordóñez and D. Roggen, "Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition," *Sensors*, vol. 16, no. 1, p. 115, Jan. 2016, 10.https://www.researchgate.net/figure/Schematic-diagram-of-a-basic-convolutional-neural-network- CNN-architecture-26_fig1_336805909

Appendix

Hyper-parameters of CNN Layers

Category	Parameter	Values	Descriptions
Network-conv2d(Layer 1)	Kernel Size	1 × 15	1x15 kernel
	Activation	ReLU	Rectified Linear Unit activation function is used.
	Strides	1 × 1	Sliding window of 1 in both dimensions.
	filters	8	Number of filters used in the convolution.
Network-conv2d(Layer2)	Kernel Size	3×1	3x1 kernel .
	activation	ReLU	Rectified Linear Unit activation function is used.
	Strides	1×1	Sliding window of 1 in both dimensions.
	filters	8	Number of filters used in the convolution.
Network conv2d(Layer3)	Kernel Size	3×1	3x1 kernel .
	activation	ReLU	Rectified Linear Unit activation function is used.
	Strides	1×1	Sliding window of 1 in both dimensions.
	filters	16	Number of filters used in the convolution.
Network conv2d(Layer4)	Kernel Size	3×3	3x3 kernel .
	activation	ReLU	Rectified Linear Unit activation function is used.
	Strides	1×1	Sliding window of 1 in both dimensions.
	filters	16	Number of filters used in the convolution.

Hyper-parameters of Neural Network

Category	Parameter	Values	Descriptions
Network	Learning Rate	0.001	Learning Rate
Network(Dense-1)	Neurons	1152	Amount Of Neurons
Network(Dense-2)	Neurons	256	Amount Of Neurons
Network(Dense-3)	Neurons	24	Amount Of Neurons
Training	Loss	MSE	Mean Squared Error
Training	Optimizer	AdamOptimizer	Method To Optimize
Training	Epochs	800	Number of training Epochs

Hyper-parameters of LSTM

Category	Parameter	Values	Description
LSTM(Layer 1)	Units	72	Number of units in the LSTM layer.
	Recurrent Activation	Sigmoid	Bias terms are added to the convolutional layer.
	Activation	Tanh	Activation function applied to the recurrent connections
LSTM(Layer2)	Units	72	Number of units in the LSTM layer.
	Recurrent Activation	Sigmoid	Bias terms are added to the convolutional layer.
	Activation	Tanh	Activation function applied to the recurrent connections

Project Code

```
import numpy as np
import pandas as pd
from keras.models import Sequential
import os
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import InputLayer
from keras.layers import LSTM,Reshape,Dropout
import keras
from keras import initializers, regularizers
from keras import layers
import tensorflow as tf
from sklearn import preprocessing
import matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.metrics import mean_absolute_error, mean_squared_error

data = pd.read_csv('/kaggle/input/historical-weather-data-for-indian-cities/nagpur.csv')
df = pd.DataFrame(data)
for x in
['totalSnow_cm','uvIndex','uvIndex.1','sunrise','moonrise','sunset','moonset','sunHour','moon_illumination','c
loudcover','visibility']:
        df.pop(x)

df['date_time'] = pd.to_datetime(df['date_time'])

df['date_time'] = pd.to_datetime(df['date_time'])
df.set_index('date_time', inplace=True)


col_order = ['tempC', 'pressure', 'humidity', 'maxtempC', 'mintempC', 'FeelsLikeC', 'HeatIndexC',
'WindChillC', 'WindGustKmph',
        'winddirDegree', 'windspeedKmph','precipMM','DewPointC']
df = df[col_order]

from datetime import datetime, timedelta
```

```
base_date = datetime(2019, 9, 16)

# Loop through each hour of the day
for h in range(6,28):
        time_stamp = base_date + timedelta(hours=h)  # this is a datetime object

        prev_date = time_stamp - timedelta(days=2)
        prev_date = prev_date + timedelta(hours=h)
        pp_date = time_stamp - timedelta(days=3)
        pp_date = pp_date + timedelta(hours=h)
        row = (df.loc[prev_date] + df.loc[pp_date]) // 2

        df.loc[time_stamp] = row

df['date_time'] = pd.to_datetime(df['date_time'])

 #Extract the month
df['month'] = df['date_time'].dt.month
df['day'] = df['date_time'].dt.day

df['season'] = df['month'].apply(
        lambda month: 'Summer' if month in [3, 4, 5, 6]
        else 'Monsoon' if month in [7,8, 9,10]
        else 'Winter'
)

# Optional: separate dataframes
summer_df = df[df['season'] == 'Summer']
monsoon_df = df[df['season'] == 'Monsoon']
winter_df = df[df['season'] == 'Winter']


summer_df = summer_df.set_index('date_time')
monsoon_df = monsoon_df.set_index('date_time')
winter_df = winter_df.set_index('date_time'


summer_df.pop('season')
monsoon_df.pop('season')
winter_df.pop('season')


nsteps_in = 3
nsteps_out = 1
features_out = 1
features_in = 15

n = len(summer_df)
train_end = int(0.8*n)
train_end = train_end - (train_end%24)
```

```
summer_train = summer_df[:train_end]
summer_test = summer_df[train_end:]

from sklearn.preprocessing import MinMaxScaler
summer_train_scaled = summer_train.copy()
summer_train_numeric = summer_train.select_dtypes(include=['number'])
summer_train_scaler = MinMaxScaler(feature_range=(0, 1))
summer_train_scaled = pd.DataFrame(
        summer_train_scaler.fit_transform(summer_train_numeric),
        columns=summer_train_numeric.columns,
        index=summer_train.index
)

from sklearn.preprocessing import MinMaxScaler
summer_test_scaled = summer_test.copy()
summer_test_numeric = summer_test.select_dtypes(include=['number'])
summer_test_scaler = MinMaxScaler(feature_range=(0, 1))
summer_test_scaled = pd.DataFrame(
        summer_test_scaler.fit_transform(summer_test_numeric),
        columns=summer_test_numeric.columns,
        index=summer_test.index
)

n = len(winter_df)
train_end = int(0.8*n)
train_end = train_end - (train_end%24)
winter_train = winter_df[:train_end]
winter_test = winter_df[train_end:]

from sklearn.preprocessing import MinMaxScaler
winter_train_scaled = winter_train.copy()
winter_train_numeric = winter_train.select_dtypes(include=['number'])
winter_train_scaler = MinMaxScaler(feature_range=(0, 1))
winter_train_scaled = pd.DataFrame(
        winter_train_scaler.fit_transform(winter_train_numeric),
        columns=winter_train_numeric.columns,
        index=winter_train.index
)

winter_test_scaled = winter_test.copy()
winter_test_numeric = winter_test.select_dtypes(include=['number'])
winter_test_scaler = MinMaxScaler(feature_range=(0, 1))
winter_test_scaled = pd.DataFrame(
        winter_test_scaler.fit_transform(winter_test_numeric),
        columns=winter_test_numeric.columns,
        index=winter_test.index
)

n = len(monsoon_df)
train_end = int(0.8*n)
train_end = train_end - (train_end%24)
```

```
monsoon_train = monsoon_df[:train_end]
monsoon_test = monsoon_df[train_end:]

from sklearn.preprocessing import MinMaxScaler
monsoon_train_scaled = monsoon_train.copy()
monsoon_train_numeric =  monsoon_train.select_dtypes(include=['number'])
monsoon_train_scaler = MinMaxScaler(feature_range=(0, 1))
monsoon_train_scaled = pd.DataFrame(
        monsoon_train_scaler.fit_transform(monsoon_train_numeric),
        columns=monsoon_train_numeric.columns,
        index=monsoon_train.index
)

monsoon_test_scaled = monsoon_test.copy()
monsoon_test_numeric =  monsoon_test.select_dtypes(include=['number'])
monsoon_test_scaler = MinMaxScaler(feature_range=(0, 1))
monsoon_test_scaled = pd.DataFrame(
        monsoon_test_scaler.fit_transform(monsoon_test_numeric),
        columns=monsoon_test_numeric.columns,
        index=monsoon_test.index
)

summer_train_datetime = summer_train_scaled.index
summer_train_datetime
summer_test_datetime = summer_test_scaled.index
summer_test_datetime
winter_train_datetime = winter_train_scaled.index
winter_train_datetime
winter_test_datetime = winter_test_scaled.index
winter_test_datetime
monsoon_train_datetime = monsoon_train_scaled.index
monsoon_train_datetime
monsoon_test_datetime = monsoon_test_scaled.index
Monsoon_test_datetime


def split_sequence(sequence,out_seq,n_steps_in, n_steps_out, stride=1):
        X, y = list(), list()
        sequence = sequence
        out_seq = out_seq

        # Use stride (x) to control how much the window shifts
        for i in range(0, len(sequence), stride):
        # find the end of this pattern
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out

        # check if we are beyond the sequence
        if out_end_ix > len(sequence):
        break
```

```
        seq_x, seq_y = sequence[i:end_ix], out_seq[end_ix:out_end_ix]
        seq_y = np.array(seq_y)
        X.append(seq_x)
        y.append(seq_y)

        return np.array(X), np.array(y)


lock_size = 24  # 24-hour block
summer_total_rows = len(summer_train_scaled)
summer_num_blocks = summer_total_rows // block_size

summer_blocks_X = []
dated_summer_blocks_X = []
for i in range(0, summer_num_blocks * block_size, block_size):
        summer_block = summer_train_scaled.iloc[i:i + block_size].values
        summer_transposed = summer_block
        summer_blocks_X.append(summer_transposed)
        summer_train_datetime = pd.Series(summer_train.index)
        dated_summer_block = summer_train_datetime.iloc[i:i + block_size]
        dated_summer_blocks_X.append(dated_summer_block)


summer_blocks_Y = []
dated_summer_blocks_Y = []
block_size = 24  # 24-hour block
summer_total_rows = len(summer_train_scaled)
summer_num_blocks = summer_total_rows // block_size
for i in range(0, summer_num_blocks * block_size, block_size):
        summer_block = summer_train_scaled.iloc[i:i + block_size, 0].values
        summer_blocks_Y.append(summer_block)
        summer_train_datetime = pd.Series(summer_train.index)
        dated_summer_block = summer_train_datetime.iloc[i:i + block_size]
        dated_summer_blocks_Y.append(dated_summer_block)



block_size = 24  # 24-hour block
summer_total_rows = len(summer_test_scaled)
summer_num_blocks = summer_total_rows // block_size

summer_testBlocks_X = []

for i in range(0, summer_num_blocks * block_size, block_size):
        summer_block = summer_test_scaled.iloc[i:i + block_size].values
        summer_transposed = summer_block
        summer_testBlocks_X.append(summer_transposed)

summer_testBlocks_Y = []
dated_summer_testBlocks_Y = []
block_size = 24  # 24-hour block
```

```
summer_total_rows = len(summer_train_scaled)
summer_num_blocks = summer_total_rows // block_size
for i in range(0, summer_num_blocks * block_size, block_size):
        summer_block = summer_test_scaled.iloc[i:i + block_size, 0].values
        summer_testBlocks_Y.append(summer_block)
        summer_test_datetime = pd.Series(summer_test.index)
        dated_summer_block = summer_test_datetime.iloc[i:i + block_size]
        dated_summer_testBlocks_Y.append(dated_summer_block)

block_size = 24  # 24-hour block
winter_total_rows = len(winter_train_scaled)
winter_num_blocks = winter_total_rows // block_size

winter_blocks_X = []
for i in range(0, winter_num_blocks * block_size, block_size):
        winter_block = winter_train_scaled.iloc[i:i + block_size].values
        winter_transposed = winter_block
        winter_blocks_X.append(winter_transposed)

winter_blocks_Y = []
dated_winter_blocks_Y = []
for i in range(0, winter_num_blocks * block_size, block_size):
        winter_block = winter_train_scaled.iloc[i:i + block_size, 0].values
        winter_blocks_Y.append(winter_block)
        winter_train_datetime = pd.Series(winter_train.index)
        dated_winter_block = winter_train_datetime.iloc[i:i + block_size]
        dated_winter_blocks_Y.append(dated_winter_block)



block_size = 24  # 24-hour block
winter_total_rows = len(winter_test_scaled)
winter_num_blocks = winter_total_rows // block_size

winter_testBlocks_X = []

for i in range(0, winter_num_blocks * block_size, block_size):
        winter_block = winter_test_scaled.iloc[i:i + block_size].values
        winter_transposed = winter_block
        winter_testBlocks_X.append(winter_transposed)

winter_testBlocks_Y = []
dated_winter_testBlocks_Y = []
block_size = 24  # 24-hour block
winter_total_rows = len(winter_train_scaled)
winter_num_blocks = winter_total_rows // block_size
for i in range(0, winter_num_blocks * block_size, block_size):
        winter_block = winter_test_scaled.iloc[i:i + block_size, 0].values
        winter_testBlocks_Y.append(winter_block)
        winter_test_datetime = pd.Series(winter_test.index)
        dated_winter_block = winter_test_datetime.iloc[i:i + block_size]
        dated_winter_testBlocks_Y.append(dated_winter_block)
```

```
block_size = 24  # 24-hour block
monsoon_total_rows = len(monsoon_train_scaled)
monsoon_num_blocks = monsoon_total_rows // block_size

monsoon_blocks_X = []
for i in range(0, monsoon_num_blocks * block_size, block_size):
        monsoon_block = monsoon_train_scaled.iloc[i:i + block_size].values
        monsoon_transposed = monsoon_block
        monsoon_blocks_X.append(monsoon_transposed)

monsoon_blocks_Y = []
dated_monsoon_blocks_Y = []
for i in range(0, monsoon_num_blocks * block_size, block_size):
        monsoon_block = monsoon_train_scaled.iloc[i:i + block_size, 0].values
        monsoon_blocks_Y.append(monsoon_block)
        monsoon_train_datetime = pd.Series(monsoon_train.index)
        dated_monsoon_block = monsoon_train_datetime.iloc[i:i + block_size]
        dated_monsoon_blocks_Y.append(dated_monsoon_block)



block_size = 24  # 24-hour block
monsoon_total_rows = len(monsoon_test_scaled)
monsoon_num_blocks = monsoon_total_rows // block_size

monsoon_testBlocks_X = []

for i in range(0, monsoon_num_blocks * block_size, block_size):
        monsoon_block = monsoon_test_scaled.iloc[i:i + block_size].values
        monsoon_transposed = monsoon_block
        monsoon_testBlocks_X.append(monsoon_transposed)

monsoon_testBlocks_Y = []
dated_monsoon_testBlocks_Y = []
block_size = 24  # 24-hour block
monsoon_total_rows = len(monsoon_test_scaled)
monsoon_num_blocks = monsoon_total_rows // block_size
for i in range(0, monsoon_num_blocks * block_size, block_size):
        monsoon_block = monsoon_test_scaled.iloc[i:i + block_size, 0].values
        monsoon_testBlocks_Y.append(monsoon_block)
        monsoon_test_datetime = pd.Series(monsoon_test.index)
        dated_monsoon_block = monsoon_test_datetime.iloc[i:i + block_size]
        dated_monsoon_testBlocks_Y.append(dated_monsoon_block)



summer_train_X, summer_train_Y = split_sequence(summer_blocks_X, summer_blocks_Y,
nsteps_in,nsteps_out,1)
summer_test_X, summer_test_Y = split_sequence(summer_testBlocks_X, summer_testBlocks_Y,
nsteps_in, nsteps_out,1)
```

```
_, dated_summer_train_Y = split_sequence(summer_blocks_X, dated_summer_blocks_Y,
nsteps_in,nsteps_out,1)
_, dated_summer_test_Y = split_sequence(summer_testBlocks_X, dated_summer_testBlocks_Y,
nsteps_in, nsteps_out,1)


winter_train_X, winter_train_Y = split_sequence(winter_blocks_X, winter_blocks_Y,
nsteps_in,nsteps_out,1)
winter_test_X, winter_test_Y = split_sequence(winter_testBlocks_X, winter_testBlocks_Y, nsteps_in,
nsteps_out,1)
_, dated_winter_train_Y = split_sequence(winter_blocks_X, dated_winter_blocks_Y,
nsteps_in,nsteps_out,1)
_, dated_winter_test_Y =
 split_sequence(winter_testBlocks_X, dated_winter_testBlocks_Y, nsteps_in, nsteps_out,1)


monsoon_train_X, monsoon_train_Y = split_sequence(monsoon_blocks_X, monsoon_blocks_Y,
nsteps_in,nsteps_out,1)
monsoon_test_X, monsoon_test_Y = split_sequence(monsoon_testBlocks_X, monsoon_testBlocks_Y,
nsteps_in, nsteps_out,1)
_, dated_monsoon_train_Y = split_sequence(monsoon_blocks_X, dated_monsoon_blocks_Y,
nsteps_in,nsteps_out,1)
_, dated_monsoon_test_Y = split_sequence(monsoon_testBlocks_X, dated_monsoon_testBlocks_Y,
nsteps_in, nsteps_out,1)

summer_train_Y  = summer_train_Y.reshape(summer_train_Y.shape[0], nsteps_out*24,features_out)
summer_test_Y = summer_test_Y.reshape(summer_test_Y.shape[0], nsteps_out*24, features_out)
dated_summer_train_Y  = dated_summer_train_Y.reshape(dated_summer_train_Y.shape[0],
nsteps_out*24,1)
dated_summer_test_Y  = dated_summer_test_Y.reshape(dated_summer_test_Y.shape[0],
nsteps_out*24,1)


winter_train_Y  = winter_train_Y.reshape(winter_train_Y.shape[0], nsteps_out*24,features_out)
winter_test_Y = winter_test_Y.reshape(winter_test_Y.shape[0], nsteps_out*24, features_out)
dated_winter_train_Y  = dated_winter_train_Y.reshape(dated_winter_train_Y.shape[0], nsteps_out*24,1)
dated_winter_test_Y  = dated_winter_test_Y.reshape(dated_winter_test_Y.shape[0], nsteps_out*24,1)


winter_train_Y  = winter_train_Y.reshape(winter_train_Y.shape[0], nsteps_out*24,features_out)
winter_test_Y = winter_test_Y.reshape(winter_test_Y.shape[0], nsteps_out*24, features_out)
dated_winter_train_Y  = dated_winter_train_Y.reshape(dated_winter_train_Y.shape[0], nsteps_out*24,1)
dated_winter_test_Y  = dated_winter_test_Y.reshape(dated_winter_test_Y.shape[0], nsteps_out*24,1)


summer_train_X = summer_train_X.reshape(summer_train_X.shape[0], nsteps_in*24,
summer_train_X.shape[3])
winter_train_X = winter_train_X.reshape(winter_train_X.shape[0], nsteps_in*24, winter_train_X.shape[3])
monsoon_train_X = monsoon_train_X.reshape(monsoon_train_X.shape[0], nsteps_in*24,
monsoon_train_X.shape[3])
```

```
summer_test_X = summer_test_X.reshape(summer_test_X.shape[0], nsteps_in*24,
summer_test_X.shape[3])
winter_test_X = winter_test_X.reshape(winter_test_X.shape[0], nsteps_in*24, winter_test_X.shape[3])
monsoon_test_X = monsoon_test_X.reshape(monsoon_test_X.shape[0], nsteps_in*24,
monsoon_test_X.shape[3])

date_list_summer_train_Y = []

for i in range(len(dated_summer_train_Y)):
        for j in range(len(dated_summer_train_Y[i])):
        date_list_summer_train_Y.append(dated_summer_train_Y[i][j][0])

date_list_summer_test_Y = []

for i in range(len(dated_summer_test_Y)):
        for j in range(len(dated_summer_test_Y[i])):
        date_list_summer_test_Y.append(dated_summer_test_Y[i][j][0])


date_list_monsoon_train_Y = []

for i in range(len(dated_monsoon_train_Y)):
        for j in range(len(dated_monsoon_train_Y[i])):
        date_list_monsoon_train_Y.append(dated_monsoon_train_Y[i][j][0])

date_list_monsoon_test_Y = []

for i in range(len(dated_monsoon_test_Y)):
        for j in range(len(dated_monsoon_test_Y[i])):
        date_list_monsoon_test_Y.append(dated_monsoon_test_Y[i][j][0])


date_list_winter_train_Y = []

for i in range(len(dated_winter_train_Y)):
        for j in range(len(dated_winter_train_Y[i])):
        date_list_winter_train_Y.append(dated_winter_train_Y[i][j][0])

date_list_winter_test_Y = []

for i in range(len(dated_winter_test_Y)):
        for j in range(len(dated_winter_test_Y[i])):
        date_list_winter_test_Y.append(dated_winter_test_Y[i][j][0])

saved_model_weights = []

class RMSECheckpoint(keras.callbacks.Callback):
        """Custom callback to save model weights when RMSE is at its lowest"""

        def __init__(self, weights_list=None, save_best_only=True, min_improvement=0.001):
```

```
super().__init__()
self.weights_list = weights_list if weights_list is not None else saved_model_weights
self.save_best_only = save_best_only
self.min_improvement = min_improvement
self.best_train_rmse = float('inf')
self.best_val_rmse = float('inf')
self.best_combined_rmse = float('inf')

def on_epoch_end(self, epoch, logs=None):
"""Check RMSE at end of each epoch and save weights if improved"""
if logs is None:
return

# Calculate RMSE from MSE loss
train_loss = logs.get('loss', float('inf'))
val_loss = logs.get('val_loss', float('inf'))

train_rmse = math.sqrt(train_loss) if train_loss > 0 else float('inf')
val_rmse = math.sqrt(val_loss) if val_loss > 0 else float('inf')
combined_rmse = (train_rmse + val_rmse) / 2

# Check if we should save (either best overall or significant improvement)
should_save = False
save_reason = ""

if self.save_best_only:
# Save only if combined RMSE is the best we've seen
if combined_rmse < self.best_combined_rmse - self.min_improvement:
        should_save = True
        save_reason = f"Best combined RMSE: {combined_rmse:.6f}"
else:
# Save if either train or val RMSE improved significantly
if (train_rmse < self.best_train_rmse - self.min_improvement or
        val_rmse < self.best_val_rmse - self.min_improvement):
        should_save = True
        save_reason = f"RMSE improved - Train: {train_rmse:.6f}, Val: {val_rmse:.6f}"

if should_save:
# Update best scores
self.best_train_rmse = min(self.best_train_rmse, train_rmse)
self.best_val_rmse = min(self.best_val_rmse, val_rmse)
self.best_combined_rmse = min(self.best_combined_rmse, combined_rmse)

# Create a deep copy of model weights
weights_copy = [layer.get_weights() for layer in self.model.layers if layer.get_weights()]

# Get additional metrics
train_mae = logs.get('mae', 0)
val_mae = logs.get('val_mae', 0)

# Store weights with metadata
```

```
            weight_info = {
                'epoch': epoch + 1,
                'weights': weights_copy,
                'train_rmse': float(train_rmse),
                'val_rmse': float(val_rmse),
                'combined_rmse': float(combined_rmse),
                'train_loss': float(train_loss),
                'val_loss': float(val_loss),
                'train_mae': float(train_mae),
                'val_mae': float(val_mae),
                'timestamp': datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
                'save_reason': save_reason
            }

            self.weights_list.append(weight_info)

            print(f"¥n✅ WEIGHTS SAVED!")
            print(f"📊 Epoch: {epoch + 1}")
            print(f"📉 Train RMSE: {train_rmse:.6f}")
            print(f"📉 Val RMSE: {val_rmse:.6f}")
            print(f"📈 Combined RMSE: {combined_rmse:.6f}")
            print(f"🎯 Reason: {save_reason}")
            print(f"💾 Total saved models: {len(self.weights_list)}")
            print("-" * 60)
            else:
            # Show progress without saving
            improvement_train = self.best_train_rmse - train_rmse
            improvement_val = self.best_val_rmse - val_rmse
            print(f"Epoch {epoch + 1}: Train RMSE: {train_rmse:.6f} (Δ{improvement_train:+.6f}), "
                  f"Val RMSE: {val_rmse:.6f} (Δ{improvement_val:+.6f}) - Not saved")



def summer_build(X_train, Y_train, save_weights=True,
                 save_best_only=True, min_improvement=0.0005):
    global saved_model_weights
    saved_model_weights.clear()
    model = Sequential()
    activity = regularizers.L2(0.00001)
    # Convolutional layers
#  model.add(input_layer)
    model.add(Conv2D(filters=8, kernel_size=(1,15), strides=(1, 1), padding="same",
                     activation='relu',input_shape=(72,15,1)))
    #model.add(MaxPooling2D(pool_size=(2, 1), padding="same"))
    #input_shape=(72,15,1)
    model.add(Conv2D(filters=8, kernel_size=(3,1), strides=(1, 1), padding="same",
                     activation='relu', activity_regularizer=activity))
    model.add(MaxPooling2D(pool_size=(2, 2), padding="same"))
    model.add(Dropout(0.2))
    model.add(Conv2D(filters=16, kernel_size=(3,1), strides=(1, 1), padding="same",
```

```
            activation='relu', activity_regularizer=activity))
        model.add(Conv2D(filters=16, kernel_size=(3,3), strides=(1, 1), padding="same",
            activation='relu', activity_regularizer=activity))
        model.add(MaxPooling2D(pool_size=( 3,3), padding="same"))

        initializer = initializers.GlorotUniform()

        # Reshape for LSTM
        conv_output_shape = model.output_shape
        print(f"Conv output shape: {conv_output_shape}")

        model.add(Flatten())
        model.add(Dense(1152, activation=activations.sigmoid, activity_regularizer=activity))
        model.add(Dense(256, activation=activations.leaky_relu, activity_regularizer=activity))
        model.add(Reshape((4,64)))
        model.add(LSTM(72,return_sequences=True,activity_regularizer=activity))
        model.add(LSTM(72,return_sequences=False,activity_regularizer=activity))

        model.add(Dense((nsteps_out*features_out*24),  activation=
activations.linear,activity_regularizer=activity, kernel_initializer=initializer))
        model.add(Reshape((nsteps_out*24, features_out)))

        # Compile model
        model.compile(loss='mse',
            optimizer=keras.optimizers.Adam(learning_rate=1e-3),
            metrics=['mae','accuracy'])

        # Prepare callbacks
        callbacks = []
        if save_weights:
        rmse_callback = RMSECheckpoint(save_best_only=save_best_only,
                        min_improvement=min_improvement)
        callbacks.append(rmse_callback)

        # Train model
        print(f"🚀 Starting training with RMSE-based weight saving")
        print(f"💾 Weight saving: {'Enabled' if save_weights else 'Disabled'}")
        print(f"🎯 Save strategy: {'Best only' if save_best_only else 'All improvements'}")
        print(f"📊 Min improvement: {min_improvement}")
        print("-" * 60)

        history = model.fit(X_train, Y_train,
            validation_split=0.10,
            epochs=800,
            batch_size=128,
            callbacks=callbacks,
            verbose=1)

        print(f"\n🎉 Training completed!")
        print(f"💾 Total models saved: {len(saved_model_weights)}")
```

```
    return model, history, saved_model_weights if save_weights else []


def restore_best_weights(model, weights_list=None, metric='combined_rmse'):
    """

    """
    if weights_list is None:
    weights_list = saved_model_weights

    if not weights_list:
    print("❌ No saved weights found!")
    return False

    # Find best weights based on specified metric (lowest RMSE)
    best_weights = min(weights_list, key=lambda x: x[metric])

    print(f"🔄 Restoring best weights (lowest {metric}):")
    print(f"📊 Epoch: {best_weights['epoch']}")
    print(f"📉 Train RMSE: {best_weights['train_rmse']:.6f}")
    print(f"📉 Val RMSE: {best_weights['val_rmse']:.6f}")
    print(f"📈 Combined RMSE: {best_weights['combined_rmse']:.6f}")
    print(f"🎯 Reason saved: {best_weights['save_reason']}")

    # Restore weights layer by layer
    layer_idx = 0
    for layer in model.layers:
    if layer.get_weights():  # Only layers with weights
    if layer_idx < len(best_weights['weights']):
        layer.set_weights(best_weights['weights'][layer_idx])
        layer_idx += 1

    print("✅ Weights restored successfully!")
    return True


summer_model,summer_history, model_weights = summer_build(summer_train_X, summer_train_Y)

loss = summer_history.history['loss']
val_loss = summer_history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

restore_best_weights(summer_model)
```

```
summer_model.save('summer_model_t.keras')


def winter_build(X_train, Y_train, save_weights=True,
                 save_best_only=True, min_improvement=0.0005):
        global saved_model_weights
        saved_model_weights.clear()
        model = Sequential()
        activity = regularizers.L2(0.00001)
        # Convolutional layers
        model.add(Conv2D(filters=8, kernel_size=(1,15), strides=(1, 1), padding="same",
                 activation='relu',input_shape=(72,15,1)))
        #model.add(MaxPooling2D(pool_size=(2, 1), padding="same"))
        #input_shape=(72,15,1)
        model.add(Conv2D(filters=8, kernel_size=(3,1), strides=(1, 1), padding="same",
                 activation='relu', activity_regularizer=activity))
        model.add(MaxPooling2D(pool_size=(2, 2), padding="same"))
        model.add(Conv2D(filters=16, kernel_size=(3,1), strides=(1, 1), padding="same",
                 activation='relu', activity_regularizer=activity))
        model.add(Conv2D(filters=16, kernel_size=(3,3), strides=(1, 1), padding="same",
                 activation='relu', activity_regularizer=activity))
        model.add(MaxPooling2D(pool_size=( 3,3), padding="same"))


        initializer = initializers.GlorotUniform()

        # Reshape for LSTM
        conv_output_shape = model.output_shape
        print(f"Conv output shape: {conv_output_shape}")
        #model.add(Reshape((-1, conv_output_shape[3])))
        model.add(Flatten())
        model.add(Dense(1024, activation=activations.sigmoid, activity_regularizer=activity))
        model.add(Dropout(0.2))
        model.add(Dense(256, activation=activations.leaky_relu, activity_regularizer=activity))
        model.add(Reshape((4,64)))
        model.add(LSTM(72,return_sequences=True,activity_regularizer=activity))
        model.add(LSTM(72,return_sequences=False,activity_regularizer=activity))
        model.add(Dense((nsteps_out*features_out*24),  activation=
activations.linear,activity_regularizer=activity, kernel_initializer=initializer))
        model.add(Reshape((nsteps_out*24, features_out)))          # Compile model
        model.compile(loss='mse',
                 optimizer=keras.optimizers.Adam(learning_rate=1e-3),
                 metrics=['mae','accuracy'])

        # Prepare callbacks
        callbacks = []
        if save_weights:
        rmse_callback = RMSECheckpoint(save_best_only=save_best_only,
                          min_improvement=min_improvement)
        callbacks.append(rmse_callback)
```

```
    # Train model
    print(f"🚀 Starting training with RMSE-based weight saving")
    print(f"💾 Weight saving: {'Enabled' if save_weights else 'Disabled'}")
    print(f"🎯 Save strategy: {'Best only' if save_best_only else 'All improvements'}")
    print(f"📊 Min improvement: {min_improvement}")
    print("-" * 60)

    history = model.fit(X_train, Y_train,
              validation_split=0.10,
              epochs=800,
              batch_size=128,
              callbacks=callbacks,
              verbose=1)

    print(f"\n🎉 Training completed!")
    print(f"💾 Total models saved: {len(saved_model_weights)}")

    return model, history, saved_model_weights if save_weights else []



saved_model_weights = []
winter_model,winter_history, winter_model_weights = winter_build(winter_train_X, winter_train_Y)


restore_best_weights(winter_model)

winter_model.save('winter_model_t.keras')

loss = winter_history.history['loss']
val_loss = winter_history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()



def monsoon_build(X_train, Y_train, save_weights=True,
              save_best_only=True, min_improvement=0.0005):
    saved_model_weights.clear()
    model = Sequential()
    activity = regularizers.L2(0.00001)
    # Convolutional layers
    model.add(Conv2D(filters=8, kernel_size=(1,15), strides=(1, 1), padding="same",
              activation='relu',input_shape=(72,15,1)))
```

```
        model.add(Conv2D(filters=8, kernel_size=(3,1), strides=(1, 1), padding="same",
                  activation='relu', activity_regularizer=activity))
        model.add(MaxPooling2D(pool_size=(2, 2), padding="same"))
        model.add(Dropout(0.2))
        model.add(Conv2D(filters=16, kernel_size=(3,1), strides=(1, 1), padding="same",
                  activation='relu', activity_regularizer=activity))
        model.add(Conv2D(filters=16, kernel_size=(3,3), strides=(1, 1), padding="same",
                  activation='relu', activity_regularizer=activity))
        model.add(MaxPooling2D(pool_size=( 3,3), padding="same"))


        initializer = initializers.GlorotUniform()

        # Reshape for LSTM
        conv_output_shape = model.output_shape
        print(f"Conv output shape: {conv_output_shape}")
        model.add(Flatten())
        model.add(Dense(1152, activation=activations.sigmoid, activity_regularizer=activity))
        model.add(Dropout(0.2))
        model.add(Dense(256, activation=activations.relu, activity_regularizer=activity))
        model.add(Reshape((4,64)))
        model.add(LSTM(72,return_sequences=True,activity_regularizer=activity))
        model.add(LSTM(72,return_sequences=False,activity_regularizer=activity))
        #model.add(Dropout(0.2))

        #model.add(Dense(256, activation=activations.leaky_relu, activity_regularizer=activity))
        model.add(Dense((nsteps_out*features_out*24),  activation=
activations.linear,activity_regularizer=activity, kernel_initializer=initializer))
        model.add(Reshape((nsteps_out*24, features_out)))


        # Compile model
        model.compile(loss='mse',
                  optimizer=keras.optimizers.Adam(learning_rate=1e-3),
                  metrics=['mae','accuracy'])

        # Prepare callbacks
        callbacks = []
        if save_weights:
        rmse_callback = RMSECheckpoint(save_best_only=save_best_only,
                           min_improvement=min_improvement)
        callbacks.append(rmse_callback)

        # Train model
        print(f"🚀 Starting training with RMSE-based weight saving")
        print(f"💾 Weight saving: {'Enabled' if save_weights else 'Disabled'}")
        print(f"🎯 Save strategy: {'Best only' if save_best_only else 'All improvements'}")
        print(f"📊 Min improvement: {min_improvement}")
        print("-" * 60)

        history = model.fit(X_train, Y_train,
```

```
                validation_split=0.10,
                epochs=800,
                batch_size=128,
                callbacks=callbacks,
                verbose=1)

        print(f"\n🎉 Training completed!")
        print(f"💾 Total models saved: {len(saved_model_weights)}")

        return model, history, saved_model_weights if save_weights else []


monsoon_model,monsoon_history, monsoon_model_weights = monsoon_build(monsoon_train_X,
monsoon_train_Y)

restore_best_weights(monsoon_model)

monsoon_model.save('monsoon_model_t.keras')


loss = monsoon_history.history['loss']
val_loss = monsoon_history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

import math
def denormalize(df,field,pred_arr):
        actual_field = df[field]
        actual_field = list(actual_field)

        mini = min(actual_field)
        maxi = max(actual_field)

        divisor = maxi-mini
        arr= np.array(pred_arr)
        col = 0
        if(field == 'pressure'):
        col = 1

        new_arr = []
        for i in range(len(arr)):
        for j in range(len(arr[i])):
        new_arr.append(arr[i][j][col])
        for i in range(len(new_arr)):
        new_arr[i] = (new_arr[i]*divisor + mini)
```

```
        return new_arr



from tensorflow.keras.models import load_model
summer_model = load_model("/kaggle/working/final_summer.keras")
monsoon_model = load_model("/kaggle/working/final_monsoon.keras")
winter_model = load_model("/kaggle/working/final_winter.keras")


summer_test_prediction = summer_model.predict(summer_test_X)
winter_test_prediction = winter_model.predict(winter_test_X)
monsoon_test_prediction = monsoon_model.predict(monsoon_test_X)

temp_summer_test = denormalize(summer_test,'tempC',summer_test_prediction)
temp_monsoon_test = denormalize(monsoon_test,'tempC',monsoon_test_prediction)
temp_winter_test = denormalize(winter_test,'tempC',winter_test_prediction)

pred_summer_test = {}
pred_summer_test = pd.DataFrame({
        'date_time': date_list_summer_test_Y,
        'tempC': temp_summer_test,

})
pred_summer_test['date_time'] = pd.to_datetime(pred_summer_test['date_time'])
pred_monsoon_test = {}
pred_monsoon_test = pd.DataFrame({
        'date_time': date_list_monsoon_test_Y,
        'tempC': temp_monsoon_test,

})
pred_monsoon_test['date_time'] = pd.to_datetime(pred_monsoon_test['date_time'])
pred_winter_test = {}
pred_winter_test = pd.DataFrame({
        'date_time': date_list_winter_test_Y,
        'tempC': temp_winter_test,
})

pred_winter_test['date_time'] = pd.to_datetime(pred_winter_test['date_time'])
import matplotlib.pyplot as plt

# Plot temperature
plt.figure(figsize=(14, 6))
plt.plot(winter_test.index, winter_test['tempC'], label='Actual  Temperature', color='blue', alpha=0.5)
plt.plot(pred_winter_test['date_time'], pred_winter_test['tempC'], label='Predicted Temperature', color='red',
linewidth=2)
plt.title('Actual Vs Predicted Winter Temperature')
plt.xlabel('Date Time')
plt.ylabel('Temperature (°C)')
plt.legend()
plt.grid(True)
```

```
plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt

# Plot temperature - Summer
plt.figure(figsize=(14, 6))
plt.plot(summer_test.index, summer_test['tempC'], label='Actual Temperature', color='blue', alpha=0.5)
plt.plot(pred_summer_test['date_time'], pred_summer_test['tempC'], label='Predicted Temperature',
color='red', linewidth=2)
plt.title('Actual Vs Predicted Summer Temperature')
plt.xlabel('Date Time')
plt.ylabel('Temperature (°C)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Plot temperature - Monsoon
plt.figure(figsize=(14, 6))
plt.plot(monsoon_test.index, monsoon_test['tempC'], label='Actual Temperature', color='blue', alpha=0.5)
plt.plot(pred_monsoon_test['date_time'], pred_monsoon_test['tempC'], label='Predicted Temperature',
color='red', linewidth=2)
plt.title('Actual Vs Predicted Monsoon Temperature')
plt.xlabel('Date Time')
plt.ylabel('Temperature (°C)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

pred_summer_test['date_time'] = pd.to_datetime(pred_summer_test['date_time'])
pred_summer_test.set_index('date_time', inplace=True)
pred_winter_test['date_time'] = pd.to_datetime(pred_winter_test['date_time'])
pred_winter_test.set_index('date_time', inplace=True)
pred_monsoon_test['date_time'] = pd.to_datetime(pred_monsoon_test['date_time'])
pred_monsoon_test.set_index('date_time', inplace=True)

import pandas as pd
import numpy as np

# Align the two DataFrames on index
common_index = pred_summer_test.index.intersection(summer_test.index)

# Get the values for comparison
actual = summer_test.loc[common_index, 'tempC']
predicted = pred_summer_test.loc[common_index, 'tempC']
mse = np.mean((actual - predicted) ** 2)
rmse = np.sqrt(mse)
mae = np.mean(np.abs(actual - predicted))
```

```
print("RMSE:", rmse)
print("MAE:", mae)

common_index = pred_monsoon_test.index.intersection(monsoon_test.index)

# Get the values for comparison
actual = monsoon_test.loc[common_index, 'tempC']
predicted = pred_monsoon_test.loc[common_index, 'tempC']

mse = np.mean((actual - predicted) ** 2)
rmse = np.sqrt(mse)
mae = np.mean(np.abs(actual - predicted))


print("RMSE:", rmse)
print("MAE:", mae)

common_index = pred_winter_test.index.intersection(winter_test.index)

# Get the values for comparison
actual = winter_test.loc[common_index, 'tempC']

predicted = pred_winter_test.loc[common_index, 'tempC']
mse = np.mean((actual - predicted) ** 2)
rmse = np.sqrt(mse)
mae = np.mean(np.abs(actual - predicted))


print("RMSE:", rmse)
print("MAE:", mae)

import matplotlib.pyplot as plt
import pandas as pd

# Filter original data for the date range (2018-05-13 to 2018-05-16)
original_data = summer_test.loc['2018-05-13':'2018-05-16']
# Filter predicted data for only the specific date (2018-05-16)
pred_data = pred_summer_test.loc['2018-05-16']

# Create the plot
plt.figure(figsize=(12, 6))
plt.plot(original_data.index, original_data['tempC'], linewidth=2, marker='o', markersize=3, label='Original ',
color='blue')
plt.plot(pred_data.index, pred_data['tempC'], linewidth=2, marker='s', markersize=4, label='Predicted ',
color='red')
plt.title('Summer Temperature (°C) - Original (May 13-16,18) vs Predicted (May 16,18)')
plt.xlabel('Date/Time')
plt.ylabel('Temperature (°C)')
plt.legend()
plt.grid(True, alpha=0.3)
```

```
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
import pandas as pd

# Monsoon plot - replace with your actual dates
# Example: plotting from 2018-07-10 to 2018-07-13 with predictions on 2018-07-13
original_monsoon = monsoon_test.loc['2018-08-12':'2018-08-15']
pred_monsoon = pred_monsoon_test.loc['2018-08-15']

# Create the plot
plt.figure(figsize=(12, 6))
plt.plot(original_monsoon.index, original_monsoon['tempC'], linewidth=2, marker='o', markersize=3,
label='Original ', color='green')
plt.plot(pred_monsoon.index, pred_monsoon['tempC'], linewidth=2, marker='s', markersize=4,
label='Predicted ', color='orange')
plt.title('Monsoon - Temperature (°C) - Original (Aug 12-15,18) vs Predicted (Aug 15, 18)')
plt.xlabel('Date/Time')
plt.ylabel('Temperature (°C)')
plt.legend()
plt.grid(True, alpha=0.3)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()


import matplotlib.pyplot as plt
import pandas as pd

# Winter plot - replace with your actual dates
# Example: plotting from 2018-12-20 to 2018-12-23 with predictions on 2018-12-23
original_winter = winter_test.loc['2018-01-12':'2018-01-15']
pred_winter = pred_winter_test.loc['2018-01-15']

# Create the plot
plt.figure(figsize=(12, 6))
plt.plot(original_winter.index, original_winter['tempC'], linewidth=2, marker='o', markersize=3,
label='Actual', color='purple')
plt.plot(pred_winter.index, pred_winter['tempC'], linewidth=2, marker='s', markersize=4, label='Predicted ',
color='brown')
plt.title('Winter - Temperature (°C) - Original (Jan 12-15, 18) vs Predicted (Jan 15, 18)')
plt.xlabel('Date/Time')
plt.ylabel('Temperature (°C)')
plt.legend()
plt.grid(True, alpha=0.3)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```