Introduction

Weather forecasting predicts future atmospheric conditions like temperature, rainfall, and wind using data from sources like satellites and sensors. It is essential for agriculture, disaster management, and daily planning.

[2] A Deep Neural Network (DNN) is a machine learning model with multiple layers of artificial neurons that learn hierarchical features from data. It captures complex patterns, making it suitable for tasks like image recognition, natural language processing, and forecasting.

[5] DNNs improve weather forecasting by capturing complex, non-linear patterns that traditional methods struggle with. Using CNNs for extraction of spatial features and LSTMs for long term dependencies between the data, DNNs offer more accurate predictions, handle real-time forecasting, and scale efficiently for various regions and variables.

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] Ehsan et al.	CNN pred: CNN-based stock market prediction using a diverse set of variables	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
4	[5]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.
5	[6]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.

DATA SET

We used time series data of a city in Maharashtra called Nagpur. It's situated a landlocked city without presence of nearby rivers in Central India. From the data pool, we get a set of 25 features, removing irrelevant features we get 15 features.

SL No.	FACTOR	UNIT
1	maxtemp	Celsius
2	mintemp	Celsius
3	sunHour	(kWh/m²)
4	Moon illumination	Lux
5	Dewpoint	Celsius
6	Feels Like	Celsius
7	HeatIndex	Celsius
8	WindChill	Celsius
9	WindGust	Kilometer per hour
10	Cloudcover	Percentage
11	Humidity	Percentage
12	Precipitation	Millimeter
13	Pressure	Millibar

SL No.	FACTOR	UNIT
14	tempC	Celsius
15	Visibility	Kilometer
16	Wind Direction	Degree
17	Windspeed	Kilometer per Hour
18	totalSnow	Centimeters (cm)
19	uvIndex	Integer
20	moonrise	Time
21	moonset	Time
22	sunrise	Time
23	sunset	Time
24	Day	Integer
25	Month	Integer

Step By Step Project Development

- **Step 1: Dataset Importing and Understanding**

We import the dataset “Historical Weather Data for Indian Cities”[1] which is publicly available on Kaggle.

date_time	maxtempC	mintempC	totalSnow_...	sunHour	uvIndex	moon_illu...	moonrise	moonset	sunrise	sunset	DewPointC	FeelsLikeC	Heatl
2009-01-01 00:00:00	30	14	0.0	8.7	5	31	09:57 AM	09:53 PM	06:51 AM	05:44 PM	6	17	17
2009-01-01 01:00:00	30	14	0.0	8.7	5	31	09:57 AM	09:53 PM	06:51 AM	05:44 PM	6	17	17
2009-01-01 02:00:00	30	14	0.0	8.7	5	31	09:57 AM	09:53 PM	06:51 AM	05:44 PM	5	16	16
2009-01-01 03:00:00	30	14	0.0	8.7	5	31	09:57 AM	09:53 PM	06:51 AM	05:44 PM	5	15	15

WindChillC	WindGust...	cloudcover	humidity	precipMM	pressure	tempC	visibility	winddirDe...	windspeed...
17	14	0	48	0.0	1013	14	10	15	6
17	16	0	48	0.0	1014	14	10	21	8
16	19	0	48	0.0	1014	14	10	27	9
15	21	0	49	0.0	1015	14	10	32	10

Step 2: Dataset Splitting

We split the data into 3 seasons: Summer, Winter, and Monsoon.

```
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']
```

Step By Step Project Development

Step 3: Training and Test Split

Divided the dataset into training and Testing sets (80% training, 20% testing).

```
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:]
```

Step 4: Scaling the Dataset

We used Min-max normalization to normalize all the features.

$$X' = \frac{x - min(x)}{max(x) - min(x)}$$

```
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
)
```

Step By Step Project Development

Step 5: Grouping Data into a 24-hour Block

We created the blocks containing 24 hours of data and appended them for each iteration.

We created a specific date and time block for the summer train dataset, which also has 24 Timestamps.

We also created this matrix for all three seasons.

```
block_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_blocks_X.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_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)
```

Step By Step Project Development

Step 6: Creating Overlapping Windows

We created the overlapping windows of size 3 days as an input sequence. For each input sequence, we created a window of the next 1 day as an output sequence.

```
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)
```

Step 7: Preparing Model Input

We reshape our samples into a 72 X 15 matrix, which is fed to our model.

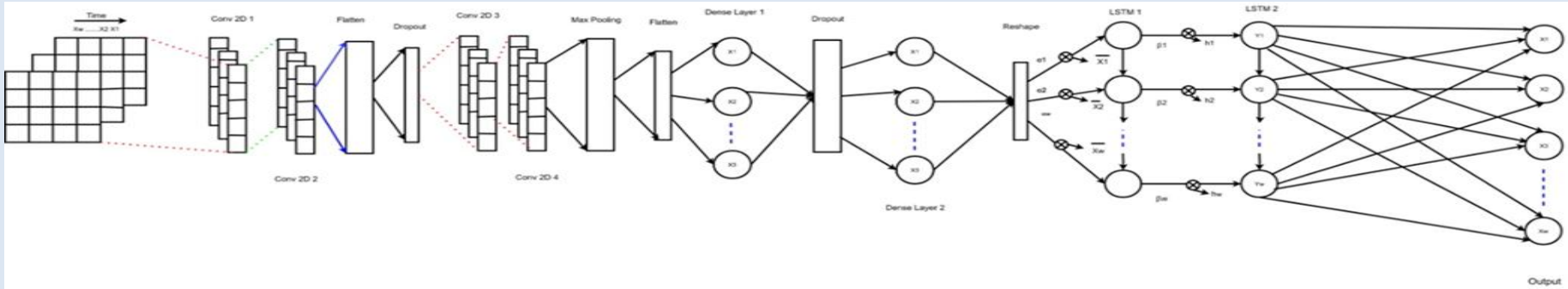
```
summer_train_X = summer_train_X.reshape(summer_train_X.shape[0], nsteps_in*24, summer_train_X.shape[3])
winter_train_X = winter_train_X.reshape(winter_train_X.shape[0], nsteps_in*24, winter_train_X.shape[3])
monsoon_train_X = monsoon_train_X.reshape(monsoon_train_X.shape[0], nsteps_in*24, monsoon_train_X.shape[3])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

Step By Step Project Development

Step 8: Model Building

We implemented a CNN-LSTM hybrid model where CNN was used for feature extraction and LSTM for capturing long-term dependencies between the data.

```
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)
    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(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)))
```



Step By Step Project Development

Step 9: Model Compilation & Training

We trained the model for 800 epochs with a learning rate of 0.001. We used 10% of the training data for validation, and also used the Adam optimizer to optimize the training process. The MSE (Mean Squared Error) loss function is Used here. We set model checkpoints based on the decreasing RMSE value. At the end of training, we restore the weights that resulted in the best combined RMSE value of the training and validation set.

Step 10: Denormalization & Prediction

We predict labels from the test set and denormalize them to produce the final prediction.

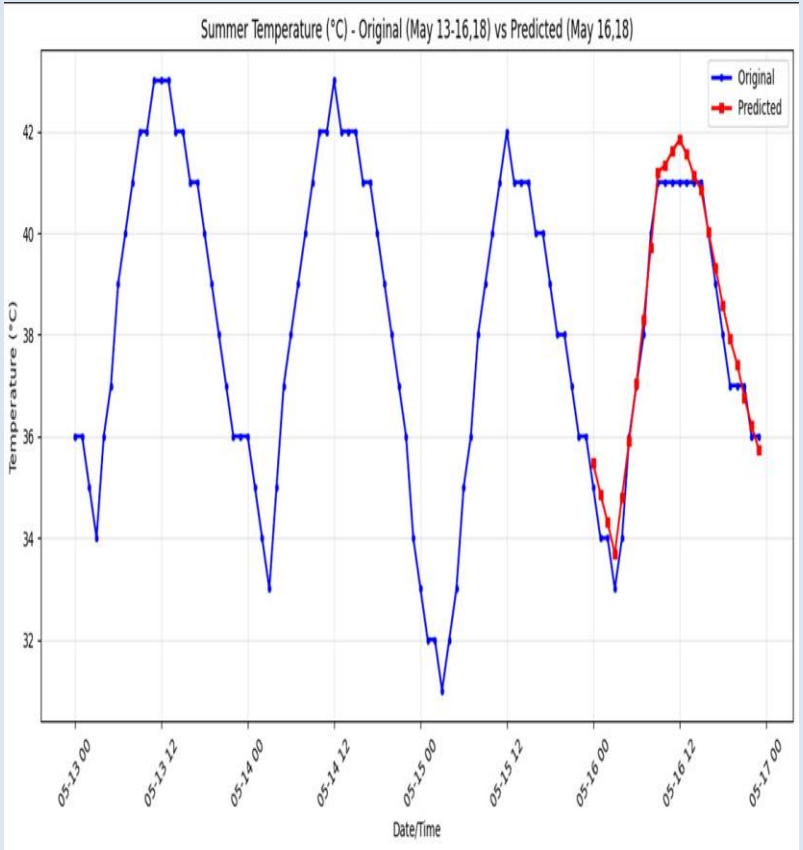
```
import math
def denormalize(df,field,pred_arr):
    actual_field = df[field]
    actual_field = list(actual_field)

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

    divisor = maxi-mini
    arr= np.array(pred_arr)
    col = 0
    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
```

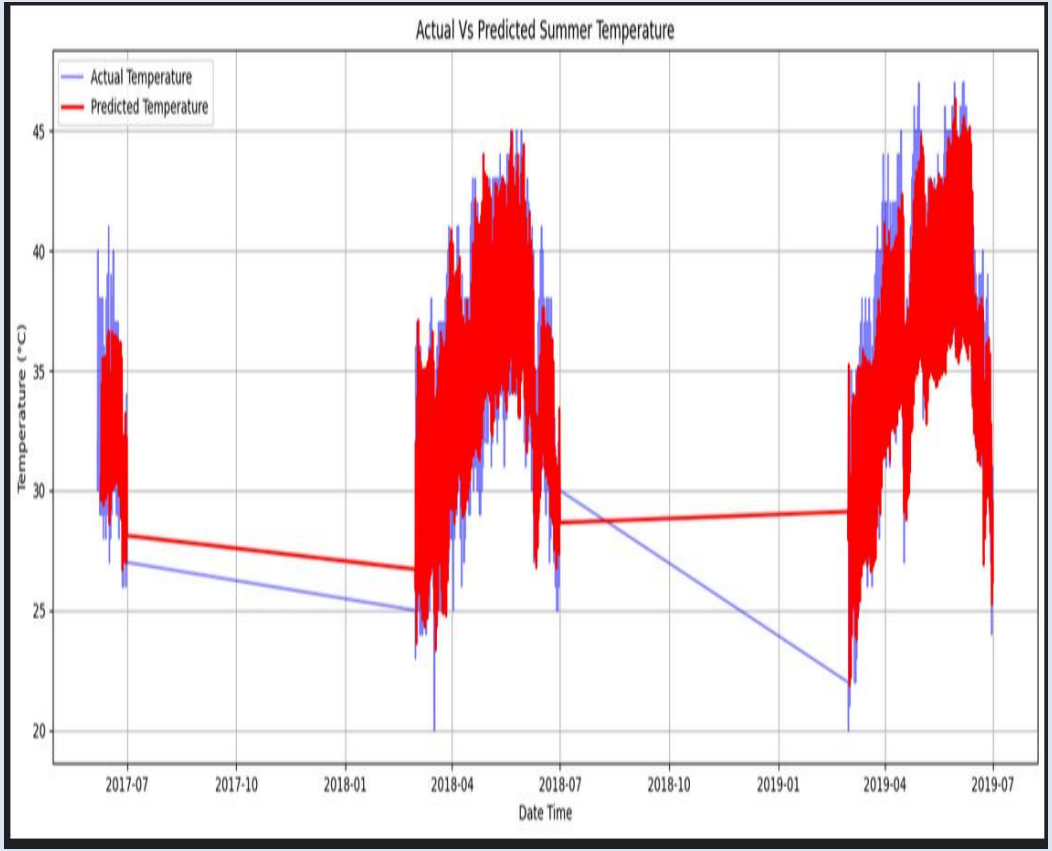
Observed Output

We compare our proposed model with a baseline model consisting of dense neurons only. On comparison, we get the following matrix below

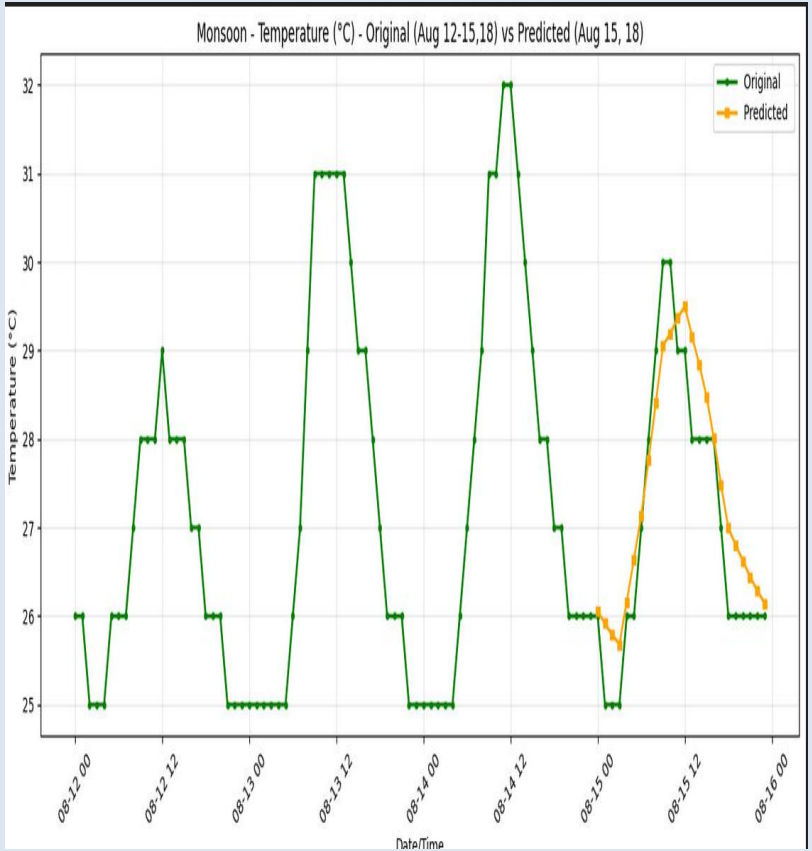
	SEASONS	RMSE	MAE
Proposed Model	Summer	2.20	1.69
	Winter	2.24	1.79
	Monsoon	1.68	1.28
Baseline Model	Summer	2.68	2.19
	Winter	2.27	1.80
	Monsoon	1.72	1.31

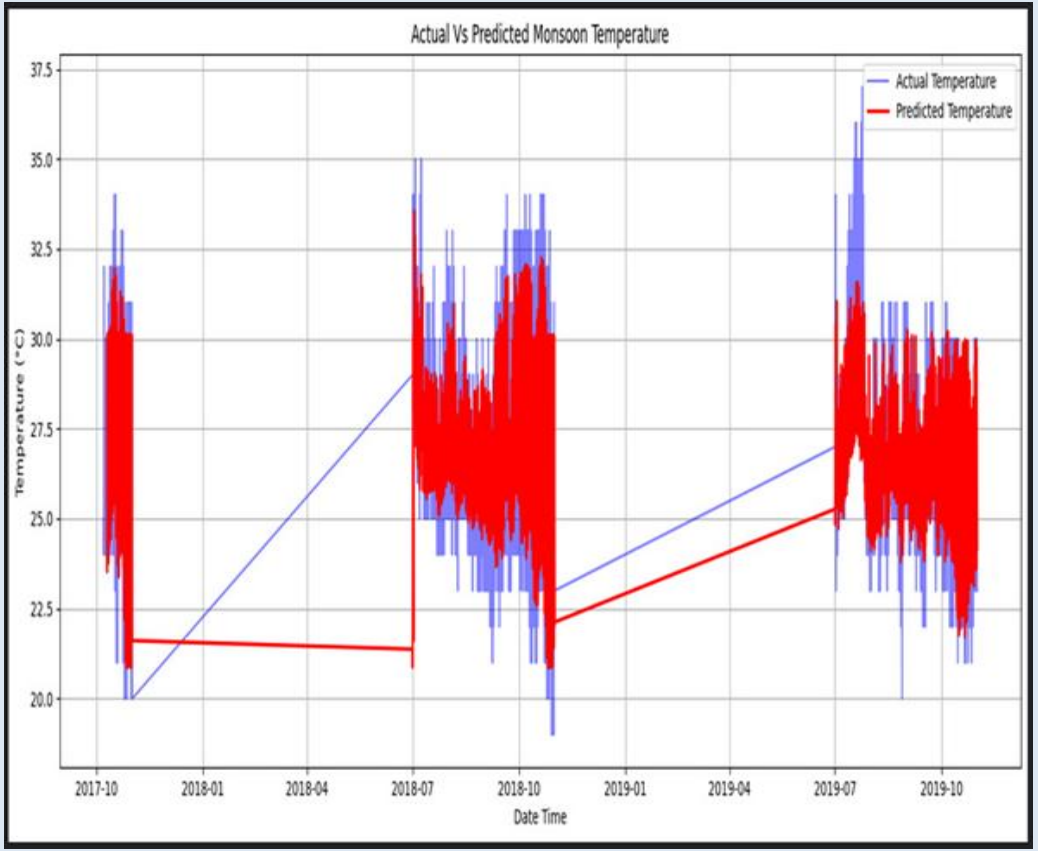
Graph Of Predicted vs Actual Temperature of Summer:



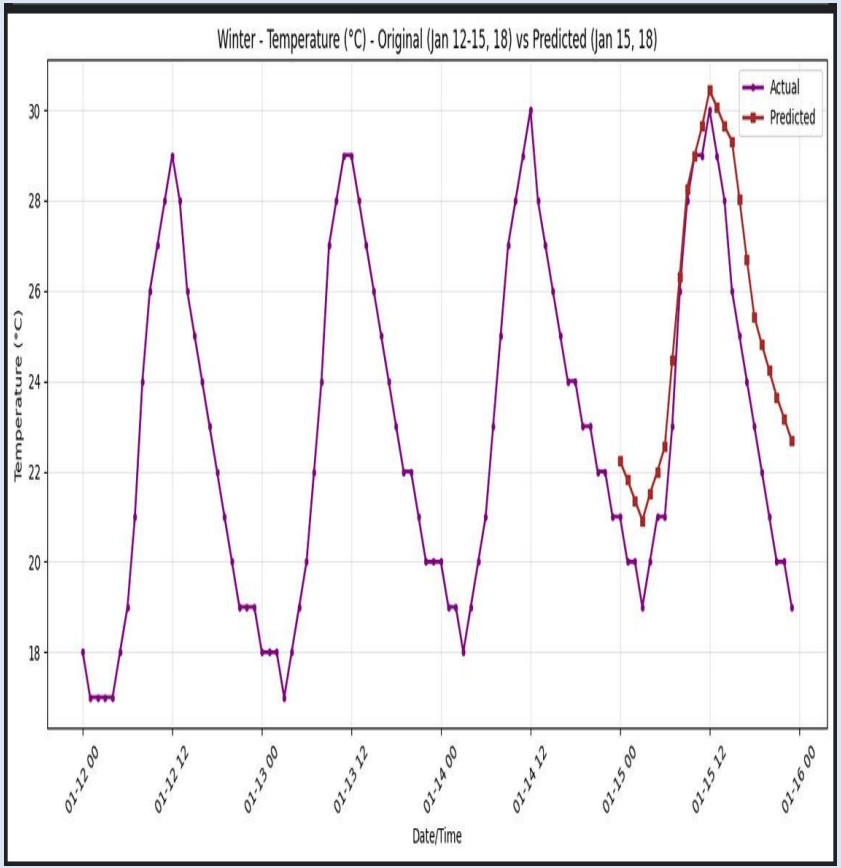


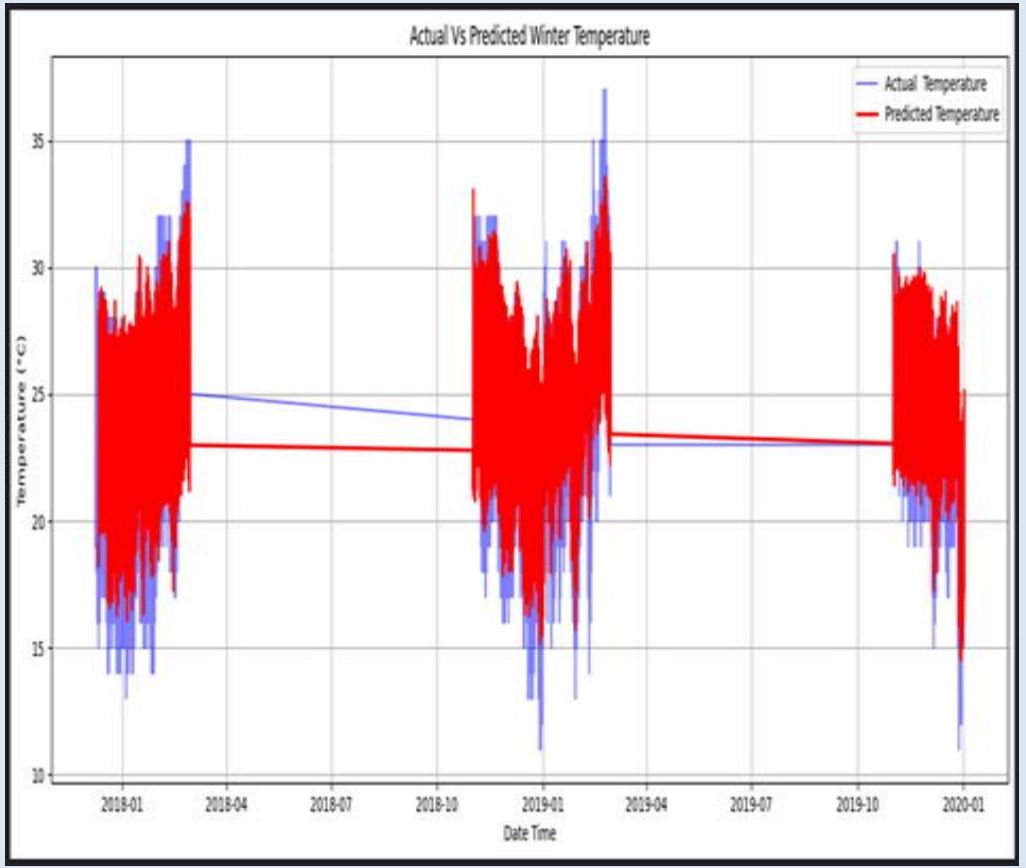
Graph Of Predicted vs Actual Temperature of Monsoon:





Graph Of Actual Temperature vs Predicted Temperature of Winter





Conclusion

The overall purpose of weather forecasting is to protect lives, agriculture, disaster management, and make daily life easier. In our project, we successfully developed a weather forecasting model using a deep neural network, which is combines a CNN and an LSTM network. In this, we used 72-hour data to predict the next 24-hour temperature of India based city of Nagpur. By developing and training the model, we are able to capture both spatial and temporal patterns. We enhanced performance through proper data preprocessing, normalization, seasonal splitting, and hyperparameter tuning. The results, measured using MAE and RMSE, confirm that deep learning techniques offer a powerful and scalable solution for precise short-term weather forecasting. Following this model, we can also predict other weather components like humidity, windspeed, air pressure, precipitation etc. This project highlights the potential of deep neural networks and opens the door for future modernization of technology. This approach can assist in better planning and decision-making for weather-dependent activities.

REFERENCES

1. www.kaggle.com/datasets/hiteshsoneji/historical-weather-data-for-indian-cities
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. 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.
5. 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.
6. 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.

Acknowledgement

We acknowledge our overwhelming gratitude & immense respect to our revered guide, Dr. Minakshi Banerjee (Professor of CSE, RCC Institute of Information Technology), under whose scholarly guidance, constant encouragement & untiring patience, we have the proud privilege to accomplish this entire project work. We feel enriched with the knowledge & sense of responsible approach we inherited from our guide & shall remain a treasure in our life.