Chapter 1

Introduction

Wind energy has become crucial to renewable energy and significantly contributes to sustainable power generation worldwide. It is essential to accurately predict wind speed to make the most of wind energy resources, as it significantly impacts the efficiency and reliability of wind power systems. Short-term wind speed forecasting is essential for optimizing energy production, integrating with the power grid, and making operational decisions in wind farms.

In the past, traditional methods of predicting wind speed relied on physical modeling techniques, which, although helpful, sometimes needed help to capture the complex dynamics of wind behavior over different periods. Recent advances in Machine Learning (ML) and Deep Learning (DL) have significantly transformed wind speed prediction, presenting promising opportunities for enhancing accuracy and efficiency.

Numerous studies have explored the use of ML techniques for wind speed forecasting. Drawing from a spectrum of recent research contributions, including works by Ibrahim et al. (2020), Duan et al. (2021), Ren et al. (2024), Liu and Liu (2023), Liu and Zhang (2024), Shen et al. (2022), Fantini et al. (2024), Liu et al. (2021), and Kim et al. (2023), this paper embarks on an exploration of various advanced models for wind speed prediction.

Ibrahim et al. (2020) proposed artificial learning-based algorithms for short-term wind speed forecasting, emphasizing the importance of computational intelligence in improving prediction accuracy. Duan et al. (2021) propose a hybrid forecasting system integrating data decomposition techniques and recurrent neural networks, showcasing its superiority in wind speed prediction accuracy. Ren et al. (2024) present a deep convolutional interval type 2 fuzzy system with adaptive feature selection for ultra-short-term wind speed prediction, underscoring advancements in interpretability and accuracy. Additionally, Liu and Liu (2023) propose a novel hybrid model incorporating variational mode decomposition, sample entropy, and bidirectional LSTM for wind speed prediction, demonstrating superior performance in prediction accuracy.

Furthermore, Liu and Zhang (2024) evaluate the integration of wavelet transform with recurrent neural networks for wind speed prediction, shedding light on the efficacy of pre-processing techniques in enhancing prediction accuracy. Shen et al. (2022) introduce a CNN and LSTM hybrid neural network for wind speed prediction in uncrewed sailboats, offering insights into accurate and stable wind speed forecasting. Fantini et al. (2024) explore the application of wavelet transform as a pre-processing technique for recurrent neural networks, aiming to enhance the accuracy of hourly wind forecasting. Moreover, Liu et al. (2021) conducted a comparison of SARIMA, GRU, and LSTM models for short-term offshore wind speed forecasting. Their findings indicate that SARIMA outperformed GRU and LSTM models in accurately forecasting future wind speeds.

Finally, Kim et al. (2023) provide a comprehensive tutorial on time series prediction using 1D-CNN and BiLSTM models, offering guidance for researchers seeking to leverage deep learning techniques in various applications, including wind speed prediction. In light of these advancements, this paper explores the methodologies, challenges, and potential applications of advanced hybrid models in wind speed prediction, aiming to contribute to the optimization and sustainability of wind energy generation on a global scale.

1.1 Motivation

1. **Significance of Renewable Energy**
    - Wind energy is a crucial part of renewable energy sources.
    - Effective wind speed prediction is vital for optimizing wind power generation and grid integration.

2. **Limitations of Conventional Methods**

   - Physical modeling techniques may struggle to capture the complex dynamics of wind behavior.
   - Traditional machine learning approaches may lack the capacity to handle nonlinear characteristics present in wind speed data.

3. **Success of Deep Learning Models**

   - Advanced deep learning models, such as LSTM, CNN, GRU, and Bi-LSTM, offer promising avenues for overcoming these limitations.
   - These models have demonstrated proficiency in capturing temporal dependencies and nonlinear patterns in time series data.

1.2 Objectives

1. **Evaluate Advanced Deep Learning Models**
   Evaluate the efficacy of advanced deep learning models, such as Long Short-Term Memory (LSTM), Convolutional Neural Networks (CNN), Gated Recurrent Unit (GRU), and Bidirectional LSTM (Bi-LSTM), for wind speed prediction.

2. **Validation of Performance**
   Validate the performance of the proposed models using real-world wind speed data from diverse geographical locations and climatic conditions. Evaluate the models' ability to generalize across various datasets to ensure their resilience and relevance in real-world scenarios.

Chapter 2

Related Work

2.1 Models Employed

2.1.1 LSTM

Long Short-Term Memory (LSTM) is a recurrent neural network architecture devised by Sepp Hochreiter and Jürgen Schmidhuber in 1997. LSTM is a potent form of Recurrent Neural Network engineered to address the issue of retaining information across extended sequences.Unlike normal RNNs, which suffer from problems such as vanishing gradients, LSTMs manage long-term dependencies significantly better.

1. **LSTM Unit**
   At its core, an LSTM network comprises individual memory units known as LSTM units. These units are responsible for processing sequential data and retaining information over time. LSTMs actually incorporate elements of both recurrent neural networks (RNNs) and Feedforward Neural Networks.

2. **Feed Forward Neural Networks**
   Within each LSTM Unit, there are four distinct feed-forward neural networks: the input gate, the output gate, the forget gate, and a core cell state. These gates determine which information from the previous state is retained or discarded, and which new information is incorporated.

3. **Vector Inputs**

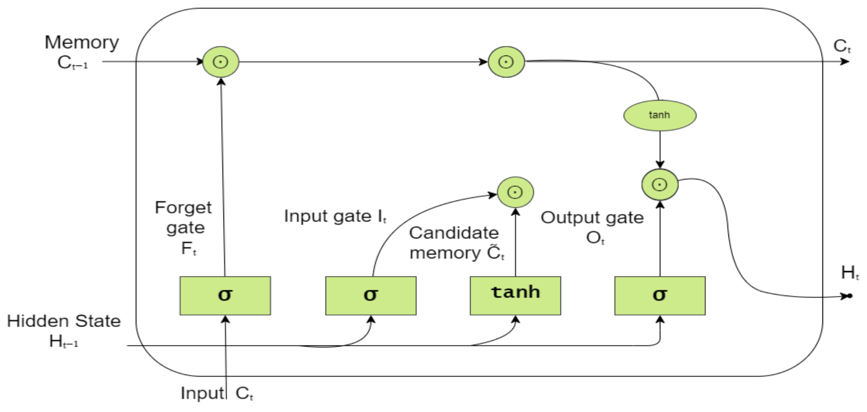

Figure 1: **LSTM Architecture (Shen 2022)**

There are three vector inputs(Fig 1) to the LSTM unit.

- Hidden State (H): A vector containing short-term memory from the previous time step, generated by the LSTM itself.
- Cell State (C): A vector representing the long-term memory of the unit, carrying information potentially from many past time steps.
- External Input (X): New information fed to the unit from outside the network at the current time step.

4. **Gate Functions**

Three Gate functions namely, Forget gate, Input Gate and Output Gate. In brief,

- The forget gate regulates what information to discard from the previous cell state
- The input gate creates a candidate vector for new information.
- It accepts the hidden state and the current cell state as inputs and generates a vector ranging between 0 and 1, controlling the information flow from the cell state to the hidden state.

2.1.2 GRU

Gated Recurrent Unit (GRU) is a type of recurrent neural network created to be proficient at processing sequential data. Similar to LSTM, a GRU network comprises individual units known as GRU cells. These cells serve as the fundamental processing units for sequential data, capable of retaining and updating information over time.

1. **Gates in GRU**

   GRU units mainly incorporate two gates(Fig 2) in its architecture

   - Update Gate: It considers both the current input (X(t)) and the previous hidden state (H(t-1)) to determine the balance between the forgotten information from the reset gate and the potential new information. A higher value emphasizes the new input, while a lower value prioritizes retention of past information.
   - Reset Gate: The reset gate governs the continuity of information from the previous hidden state (H(t-1)) depending on the current input (X(t)). A value closer to 1 signifies a higher degree of retention, while a value near 0 indicates a substantial reset, prioritizing the new input.

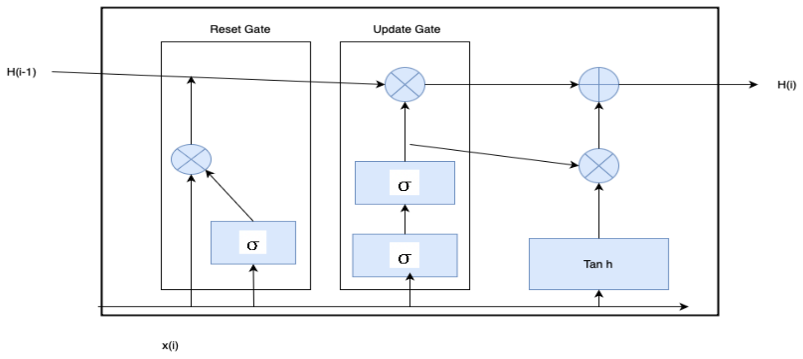

Figure 2: **GRU Architecture (Fantini 2024)**

2. **Vector Input**

   In contrast to LSTM, GRU integrates a single vector input. This single input encompasses both the current input and the previous hidden state.

3. **Memory Management**
   GRU units dynamically update their internal state by considering both the input and the previous state, without explicitly distinguishing between long-term and short-term memory components. This streamlined approach enables efficient memory management and computational efficiency.

2.1.3 CNN

CNN stands for Convolutional Neural Network. It is a sort of manufactured neural network commonly utilized in picture acknowledgment and computer vision assignments. CNNs take motivation particularly from the human visual cortex.

1. **Convulational layers**
   These layers comprise of channels (called parts) that slide over the input image, performing operations like convolution. Each channel extricates certain features from the picture, such as edges, surfaces, or shapes. Different filters are connected to produce a set of maps, capturing distinctive perspectives of the input image.

2. **Pooling Layers**
   After convolution, pooling layers(Fig 3) downsample the include maps, lessening their dimensionality. Common pooling operations incorporate max pooling and average pooling, which extricate the most noteworthy highlights from each locale of the feature maps.

3. **Activation Function**
   Non-linear enactment capacities like ReLU (Rectified Linear Unit) are applied to the yield of convolutional and pooling layers. These capacities introduce non-linearity into the organize, permitting it to learn complex designs and connections in the input data.

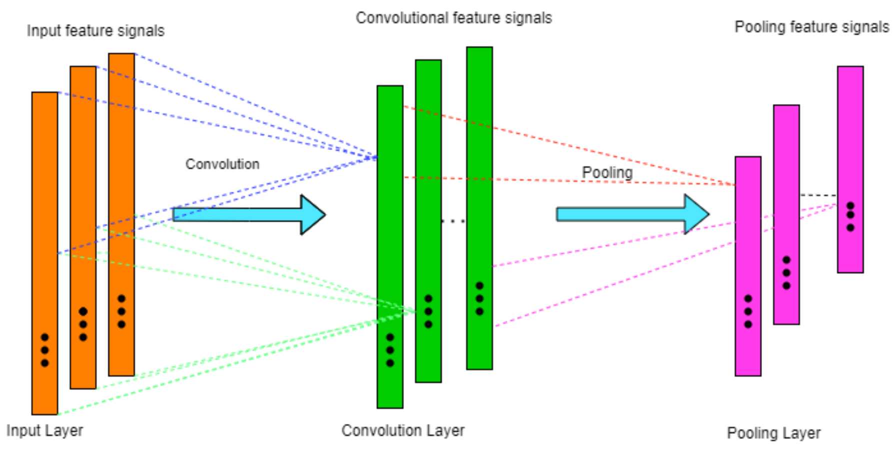

Figure 3: **CNN Architecture (Kim 2023)**

4. **Fully Connected Layers**
   The yield from the convolutional and pooling layers is smoothed and passed through one or more completely associated layers. These layers perform classification by learning to outline the extricated highlights to particular yield classes.

2.1.4 BiLSTM

Bidirectional Long Short-Term Memory (BiLSTM) represents a significant advancement in recurrent neural network (RNN) architecture, building upon the foundation of LSTM. This section provides a brief overview of BiLSTM and its significance in sequence modeling tasks.

- **Bidirectional Processing in BiLSTM**
  BiLSTM comprises two LSTM layers(Fig 4): a forward LSTM and a backward LSTM. The forward LSTM forms the input arrangement in the standard forward heading, whereas the backward LSTM forms it in switch. It highlights the significance of bidirectional handling in capturing both past and future setting data for each time step.
- **Leveraging Bidirectional Context**
  BiLSTM leverages bidirectional setting to improve grouping modeling. By

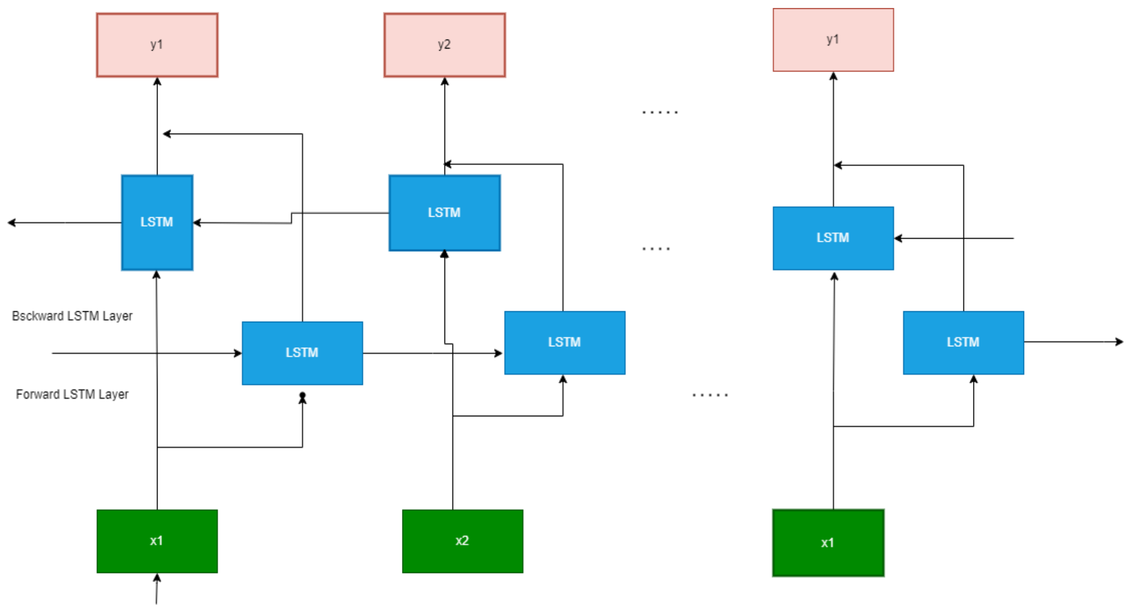

Figure 4: **BiLSTM Architecture (Liu 2023)**

considering data from both preceding (t-1) and succeeding (t+1) time steps, BiLSTM picks up a more comprehensive understanding of the input grouping. This bidirectional setting empowers BiLSTM to capture long-range conditions and complicated real life relationships, leading to improved prescient performance.

- **Advantages Over Unidirectional LSTM**
  BiLSTM systems process input sequences in both forward and backward directions simultaneously. This allows them to capture pertinent information from both past and future contexts, resulting in a more comprehensive understanding of the input sequence. Moreover, it encourages superior data flow throughout the arrange. This can offer relief from the vanishing slope issue and improve the model's capacity to capture long-range conditions in sequential data.

Chapter 3

Proposed Methodology

3.1 Block diagram of the proposed methodology

Our study encompasses multiple datasets(Fig 6) spanning various regions, each documenting wind speed measurements at hourly intervals from 1980 to 2018. To ensure consistency across datasets, we standardized the wind speed values, scaling them to a uniform range from 0 to 1.

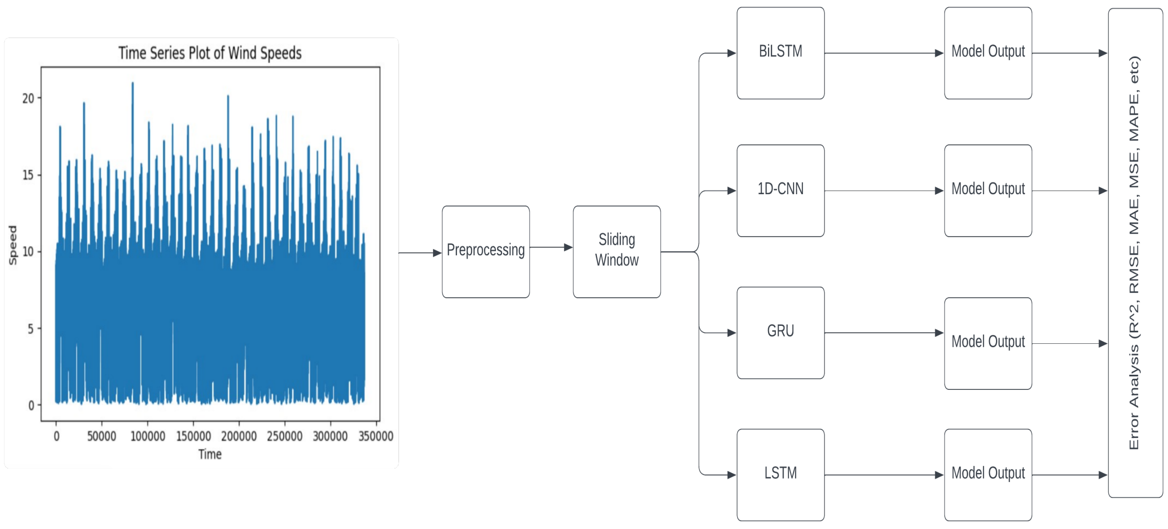

Figure 5: Block diagram of the proposed methodology

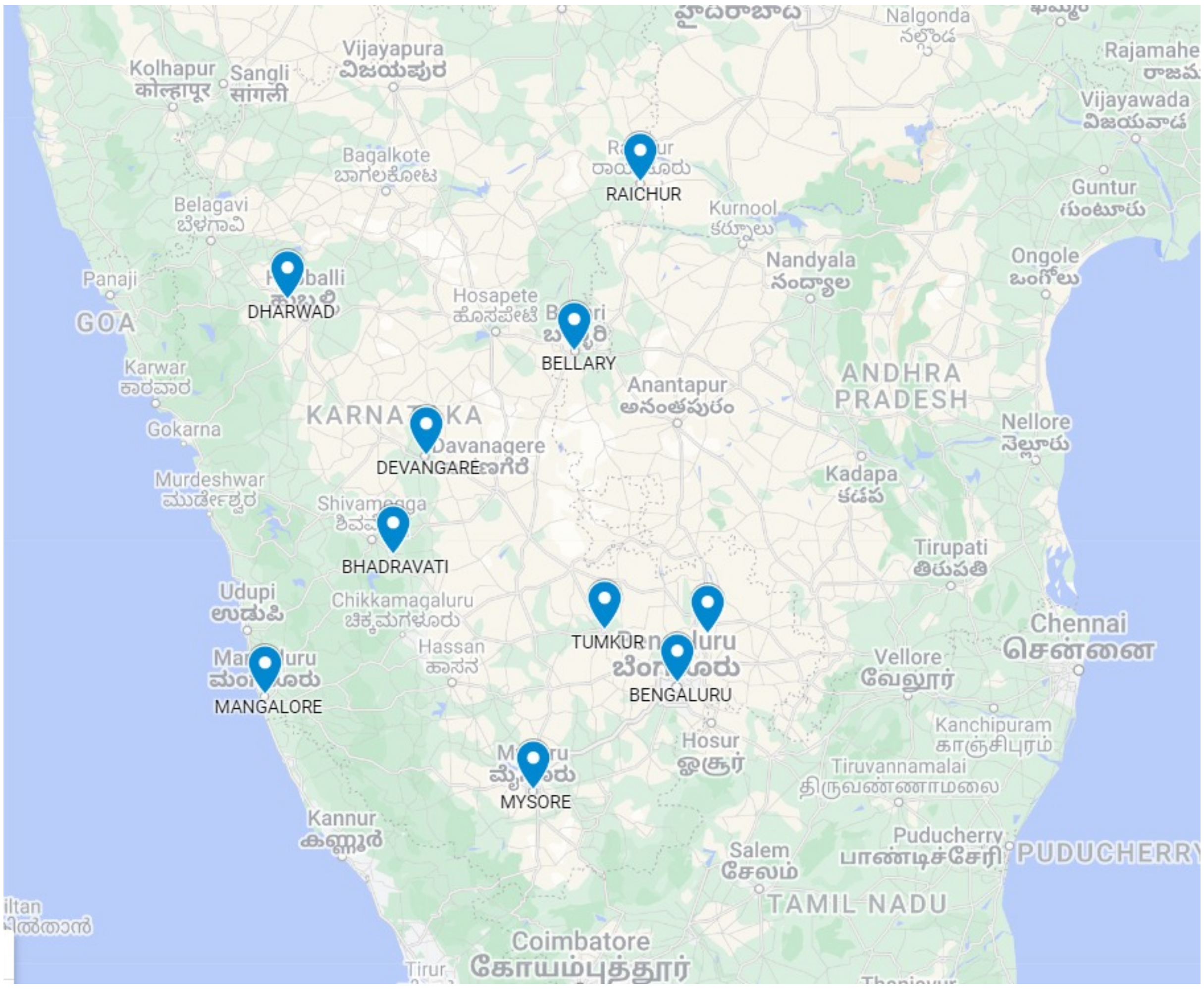

Figure 6: Map of Karnataka showing the chosen stations

- **Data Distribution**

  An exploratory investigation of the wind speed information was conducted by plotting the wind speed estimations for the starting 100,000 columns. The examination revealed discernible designs suggestive of seasonality inside the dataset. Notably, the plotted information shown(Fig 7) repetitive spikes happening at reliable intervals, indicating the nearness of occasional fluctuations.

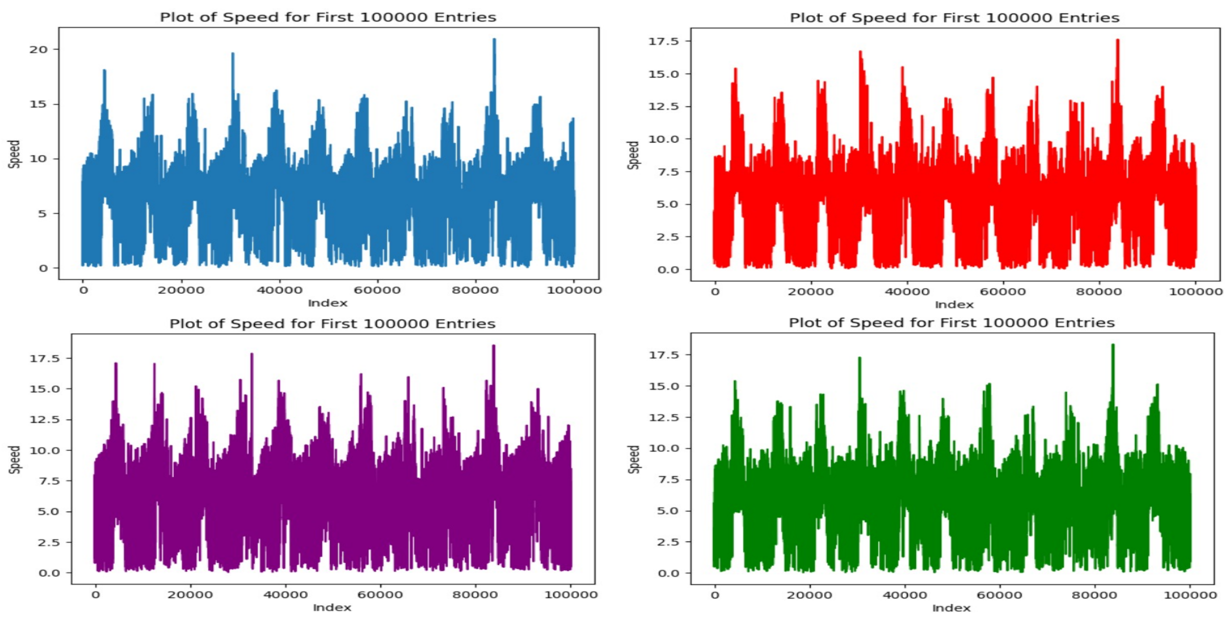

Figure 7: Dataset visualization for stations Bellary (Blue), Bhadravati (Red), Raichur (Violet) and Tumkur (Green).

- **Autocorrelation Analysis**
  In pursuit of identifying the significant lag values for predicting current-day wind speed, we conducted autocorrelation analysis. This process involved assessing the correlation between wind speed measurements at different time lags. By plotting autocorrelation graphs, we visually inspected the correlation strength at various lag values.

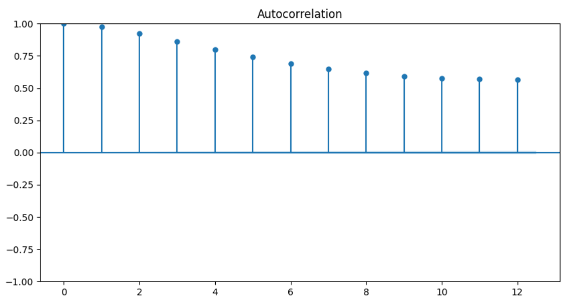

Figure 8: Autocorrelation plot with number of lags set to 12

The autocorrelation analysis revealed distinct patterns(Fig 8), indicating the influence of previous-day wind speed measurements on the current-day prediction. By scrutinizing the autocorrelation graphs, we discerned lag values with higher correlation coefficients, signifying their greater importance in forecasting wind speed.

- **Data preprocessing**

  1. **Handling Missing Values**
     In this step we have filled 0 where value is missing column, for this we have used fillna function of pandas library. It is important to handle missing values in some machine learning models which don't handle missing values well.

  2. **Feature Scaling**
     The machine learning approach of feature scaling is used to standardise the variety of features or variables in the dataset. By having a wider range of values than others, the intention is to ensure that no particular characteristic dominates the learning process. For feature scaling, we employed min-max scaling(1) in our project. In machine learning, the min-max feature scaling technique is often employed to rescale numerical features to a specific range, commonly between 0 and 1. The min-max scaling formula is expressed as:

$$x' = \frac{x - \min(X)}{\max(X) - \min(X)} \tag{1}$$

     In this formula, x represents the original feature value, min(x) denotes the minimum value of the feature in the dataset, and max(x) represents the maximum value of the feature in the dataset. The resulting scaled values fall within the range of 0 to 1, with the minimum value being scaled to 0 and the maximum value scaled to 1. This technique is particularly beneficial for algorithms relying on distance calculations, like k-nearest neighbors and support vector machines, as it guarantees that all features are on a consistent scale.

- **Dividing dataset by a sliding window technique**

  The dataset experienced preprocessing through the usage of a sliding window method, which encouraged the creation of input-output pairs for consequent prescient modeling. Particularly, each occurrence in the dataset was divided into consecutive windows, where the wind speed measurements from the current hour and the going before four hours were utilized as input features. Consequently, the target variable, speaking to the wind speed for the subsequent hour, was decided based on the information for the 6th hour within each window.

  This was decided by analysing the auto-correlation plot of dataset signifying the weight-age of each month's value over time (Fig 8). Since almost all months gave similar auto-correlation values, above window was presumed. This approach guaranteed that the demonstrate was prepared to predict the wind speed of the 6th hour utilizing the information from the preceding 5 hours. This handle was iteratively rehashed over the dataset, increasing the window by one hour at a time. As a result, the X dataset was constructed, comprising arrangements of wind speed estimations crossing five-hour windows. Concurrently, the comparing y dataset was shaped, comprising of the wind speed values for the 6th hour in each window, serving as the target variable for prediction. In outline, the sliding window procedure empowered the precise creation of input-output sets, subsequently encouraging the advancement of prescient models for wind speed forecasting.

- **Model Training**

  Various Models Used

  Many models such as LSTM, GRU, 1D CNN, and BiLSTM have been employed in wind speed prediction studies. These models were utilized to assess their predictive capabilities across diverse datasets, aiming to discern the most effective performer among them. The comparative analysis was conducted to discern the model that exhibits superior performance across varied wind speed datasets of different regions of Karnataka.

1. **LSTM (Long Short-Term Memory)**
   LSTM (Long Short-Term Memory) is a recurrent neural network (RNN) specifically engineered to address the vanishing gradient problem. It effectively preserves long-term dependencies in sequential data through memory cells, rendering it well-suited for tasks such as time series prediction and natural language processing.
2. **GRU (Gated Recurrent Unit)**
   GRU (Gated Recurrent Unit) is an RNN architecture similar to LSTM, crafted to capture dependencies over extended periods within sequential data. It simplifies the LSTM structure by consolidating the forget and input gates into a single update gate, thereby reducing computational complexity while retaining effectiveness in tasks like language modeling and speech recognition
3. **1D CNN (One-Dimensional Convolutional Neural Network)**
   Unlike traditional CNNs used for image processing, 1D CNNs are tailored for sequential data analysis. They apply one-dimensional convolutional filters to extract spatial features from sequential inputs, making them effective for tasks such as sensor data analysis and time series forecasting.
4. **BiLSTM (Bidirectional LSTM)**
   BiLSTM expands on the LSTM architecture by handling input sequences in both forward and backward directions. This bidirectional approach captures contextual information from past and future time steps, thereby enriching the model's comprehension of temporal dependencies and enhancing performance in tasks like sentiment analysis and named entity recognition..

- **Hyperparameter Tuning**

The models underwent an extensive process of hyperparameter tuning, employing the Keras Tuner library to optimize their performance. This involved meticulous adjustments to various parameters such as the number of units, layers, and activation functions. Through systematic exploration, the optimal

hyperparameters for each model were discerned. Subsequently, these best-performing hyperparameters were incorporated into the model configuration, culminating in the construction of the most effective model variant.
This optimization process was facilitated through the use of the Keras Tuner library.

- **Learning Rate**
  This parameter dictates the magnitude of updates made to the model's weights during training. It's a critical factor in ensuring the model converges efficiently and remains stable throughout the training process.
- **Number of Layers**
  Referring to the depth of a neural network, this parameter encompasses the count of hidden layers in deep learning architectures. While increasing layer count can boost the model's ability to discern intricate patterns, it also escalates the risk of overfitting.
- **Number of Units/Neurons**
  Number of Units/Neurons: This parameter denotes the quantity of neurons or units within each layer of the neural network. Adjusting this count influences the model's capacity to represent complex features and incurs computational overhead.
- **Activation Functions**
  Activation functions, including ReLU, sigmoid, and tanh, introduce non-linearity to the output of neurons, allowing the model to capture complex data relationships. Choosing the right activation functions is crucial for the effectiveness of the model.
- **Model Architecture**
  This parameter encapsulates the overall structure and connectivity of the neural network, including the arrangement and type of layers (e.g., convolutional, recurrent, dense). The choice of architecture profoundly impacts the model's performance and adaptability to various tasks.
- **Batch Size**

This parameter defines the number of training examples processed in each iteration of gradient descent. Optimizing batch size influences training efficiency and can affect the model's convergence dynamics.

Hyperparameter	**Value**
Number of epochs	100
Learning rate	0.0001
Optimizer	Adam
Training batch size	64
Train test split	70:30
Activation function	ReLU

Table 1: Hyperparameters and Their Values

Model Generated after tuning

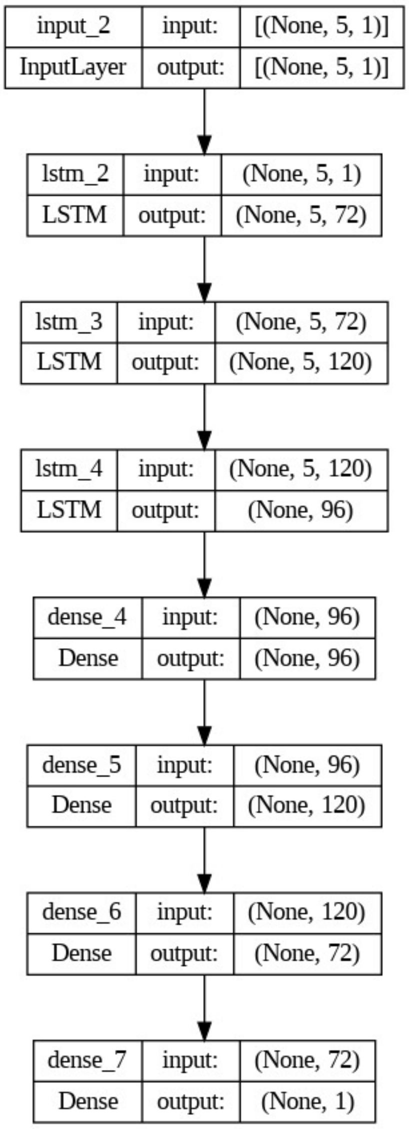
input_2 | input: | [(None, 5, 1)]
InputLayer | output: | [(None, 5, 1)]
lstm_2 | input: | (None, 5, 1)
LSTM | output: | (None, 5, 72)
lstm_3 | input: | (None, 5, 72)
LSTM | output: | (None, 5, 120)
lstm_4 | input: | (None, 5, 120)
LSTM | output: | (None, 96)
dense_4 | input: | (None, 96)
Dense | output: | (None, 96)
dense_5 | input: | (None, 96)
Dense | output: | (None, 120)
dense_6 | input: | (None, 120)
Dense | output: | (None, 72)
dense_7 | input: | (None, 72)
Dense | output: | (None, 1)

Figure 9: Bellary Dataset's LSTM Model after Tuning

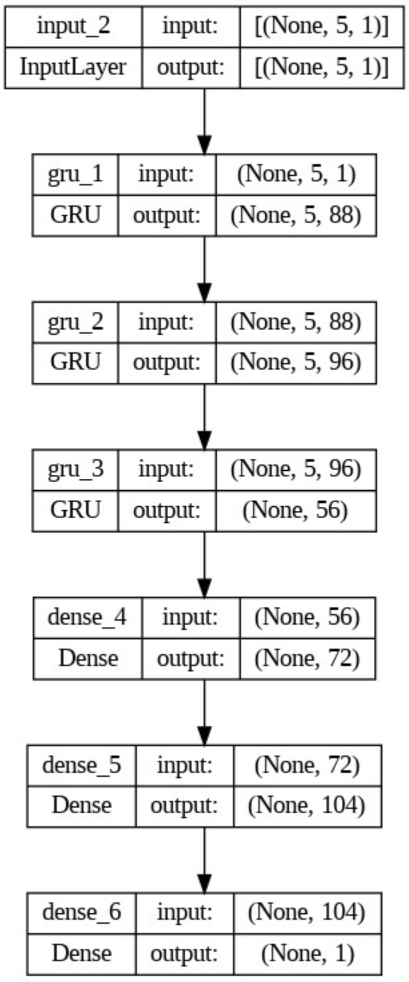
input_2 | input: | [(None, 5, 1)]
InputLayer | output: | [(None, 5, 1)]
gru_1 | input: | (None, 5, 1)
GRU | output: | (None, 5, 88)
gru_2 | input: | (None, 5, 88)
GRU | output: | (None, 5, 96)
gru_3 | input: | (None, 5, 96)
GRU | output: | (None, 56)
dense_4 | input: | (None, 56)
Dense | output: | (None, 72)
dense_5 | input: | (None, 72)
Dense | output: | (None, 104)
dense_6 | input: | (None, 104)
Dense | output: | (None, 1)

Figure 10: Bellary Dataset's GRU Model after Tuning

- **Model fitting**
  The model was trained on the designated training set, and subsequent predictions were generated using the independent test set. Various graphical representations were employed to visualize the progression of the validation loss over training. Additionally, graphs were generated to compare and contrast the actual and predicted values, providing insights into the model's performance and predictive accuracy.

  The graph illustrating the history of loss over the number of epochs (Figure 11) demonstrates a consistent decrease in the validation loss throughout the

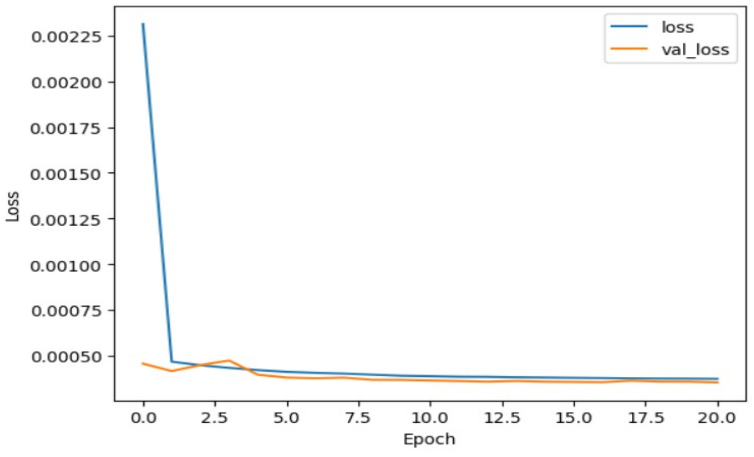

Figure 11: Loss History Visualization for Bengaluru dataset LSTM model

training process. Towards the end of the training, the validation loss reaches a minimal level. This trend signifies the model's ability to progressively improve its performance by effectively minimizing the discrepancy between predicted and actual values, ultimately resulting in a highly optimized validation loss.

Chapter 4

Experimental Setup

4.1 Technologies Used

4.1.1 Platform used: Google Colab

Google Colab is a cloud-based platform tailored for executing Python code efficiently. It grants access to GPU and TPU resources, significantly speeding up computations for deep learning tasks. Notably, it comes equipped with pre-installed libraries and packages commonly employed in data science and machine learning, sparing users the hassle of manual installation and configuration. Moreover, Google Colab facilitates real-time collaboration, enabling multiple users to work on the same notebook concurrently. This feature fosters teamwork and facilitates knowledge sharing among collaborators.

4.1.2 Libraries Used

- *Tensorflow*

Tensorflow is the machine learning framework that provides tools that ensure the construction and deployment of models and libraries. Its scalability and distributed computing capabilities are instrumental in handling large-scale datasets and accelerating training times, while its extensibility allows for the integration of custom components and algorithms, further expanding its functionality. TensorFlow provides a comprehensive framework for implementing and training CNNs, to leverage deep learning techniques for a wide range of computer vision tasks.

▮ *Keras*

Keras is a high-level neural networks API that simplifies development of deep learning networks with fast model prototyping. Keras gives an incredibly user-friendly deep learning framework to build and train machine learning models. It offers a diverse range of pre-trained models through its applications module, enabling neural network architectures like VGG, ResNet, and Inception for transfer learning and feature extraction.

▮ *Scikit-learn*

Scikit-learn is a multi-tasking machine learning library that is backed by a powerful set of algorithms for classification, regression, clustering, dimensionality reduction, and model assessment. It consists of embedded tasks, including feature selection, model training, hyperparameter tuning, cross-validation, as well as performance measures.

▮ *NumPy*

NumPy stands as a cornerstone package for scientific computing in Python, offering robust tools tailored for working with arrays and matrices. Its efficient implementation of mathematical functions and operations makes it essential for tasks ranging from data manipulation to numerical analysis and machine learning.

▮ *Pandas*

Pandas simplifies data handling with its structures such as DataFrames and Series, which are specifically designed for structured data management. Its extensive toolkit includes functions for data cleaning, transformation, and aggregation to preprocess raw data effectively. Pandas facilitate tasks like data manipulation, visualization, and trend analysis.

▮ *Matplotlib*

Matplotlib is a tool used for graph plotting enabling the creation of static, animated, and interactive graphics, offering a diverse palette of data charts and plots. It helps to visualize distributions, correlation matrices, and other pertinent information crucial for analyzing and interpreting data, thereby useful for timely detection and intervention strategies.

Chapter 5

Results Analysis

Root Mean Square Error (RMSE):

It quantifies the average magnitude of the residuals, which are the differences between predicted and observed values, by calculating the square root of the average squared differences. This provides a comprehensive assessment of prediction accuracy in regression models.

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

where:

- n is the number of observations,
- y_i is the observed value,
- $\hat{y}_i$ is the predicted value.

Mean Squared Error (MSE)
It is a measure used in regression analysis to evaluate the average squared difference between predicted and observed values(3), providing insight into the overall model performance.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3}$$

where:

- n is the number of observations,
- y_i is the observed value,
- $\hat{y}_i$ is the predicted value.

Mean Absolute Error (MAE)
It's a metric utilized in regression analysis to quantify the average absolute difference between predicted and observed values. This offers a straightforward measure of model accuracy that's less influenced by outliers compared to MSE or RMSE.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4}$$

where:

- n is the number of observations,
- y_i is the observed value,
- $\hat{y}_i$ is the predicted value.

R Squared:
The coefficient of determination (R-squared) assesses the extent to which the variance in the dependent variable can be explained by the independent variable(s), reflecting the goodness of fit of the regression model. It ranges between 0 and 1, with higher values indicating a stronger fit.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{5}$$

where:

- n is the number of observations,
- y_i is the observed value,
- $\hat{y}_i$ is the predicted value,
- $\bar{y}$ is the mean of the observed values.

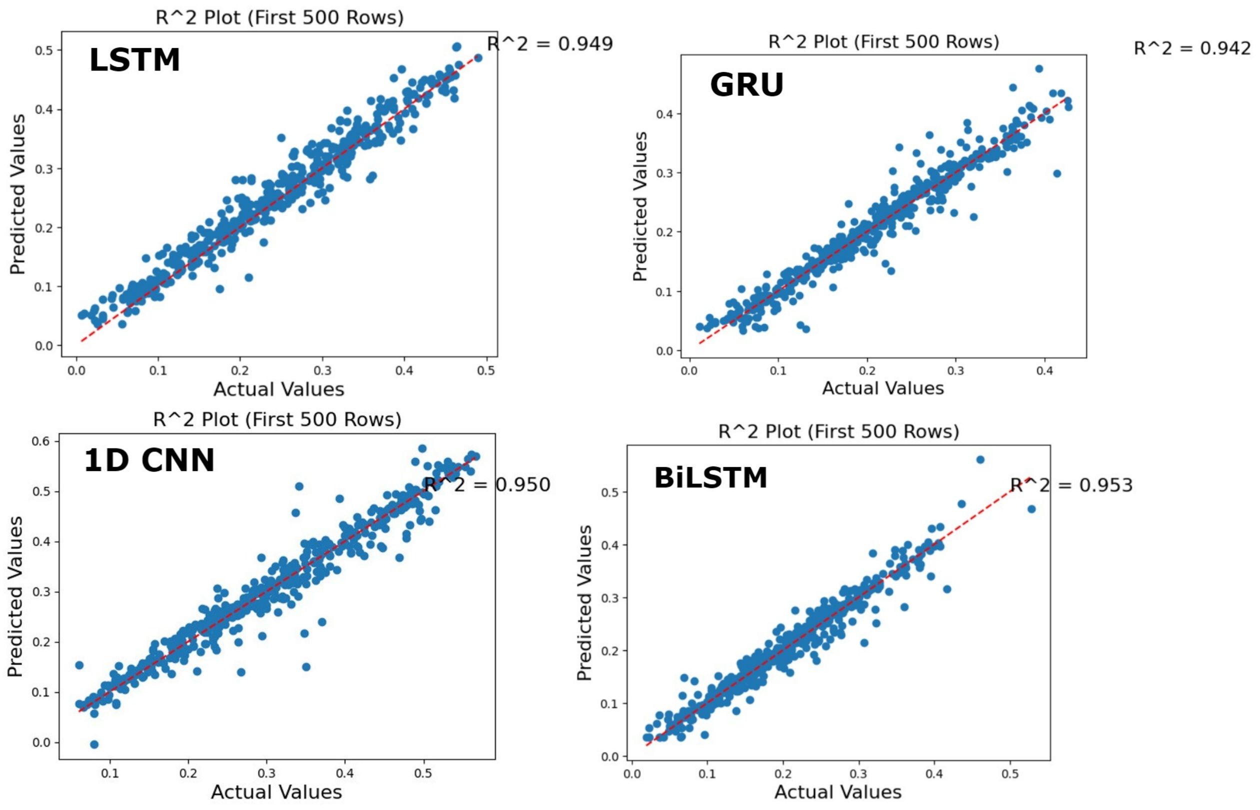

Figure 12: R squared plots for all models in Bengaluru dataset.

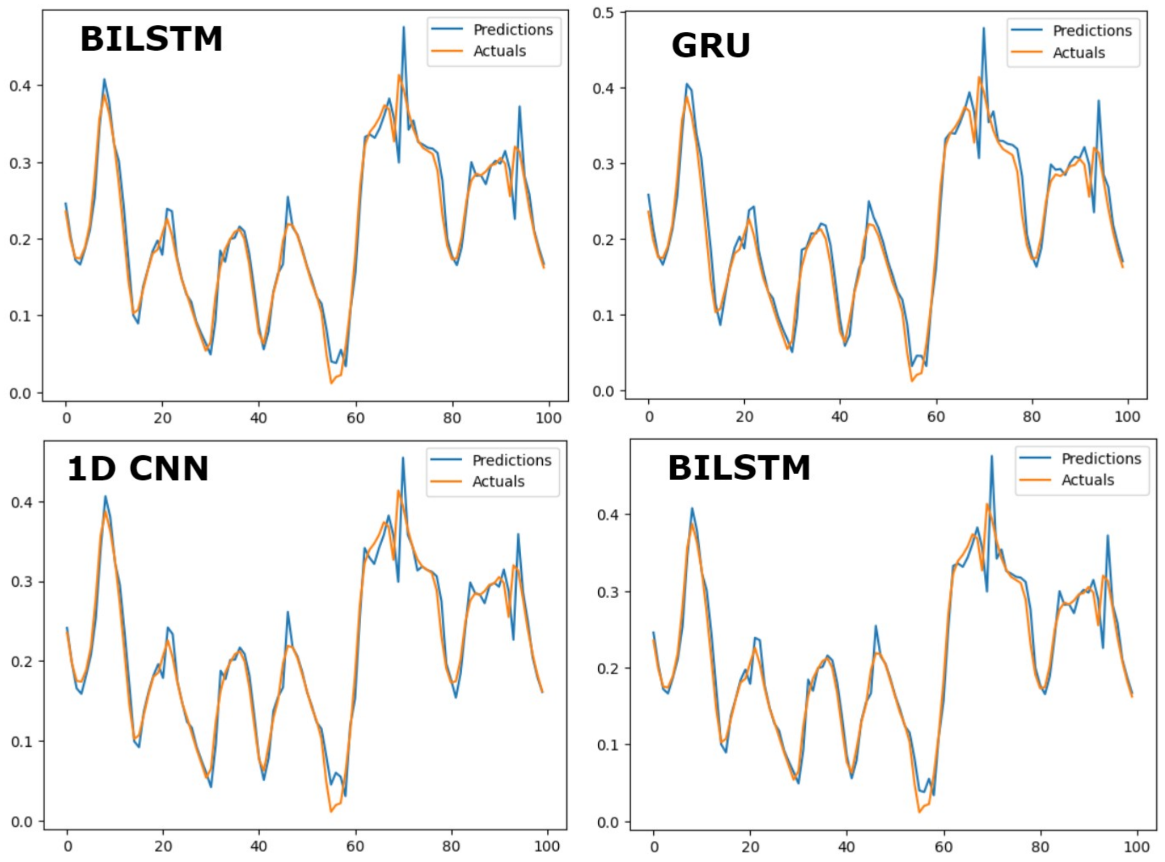

Figure 13: Predicted vs Actual wind speed values for all models in Bengaluru dataset.

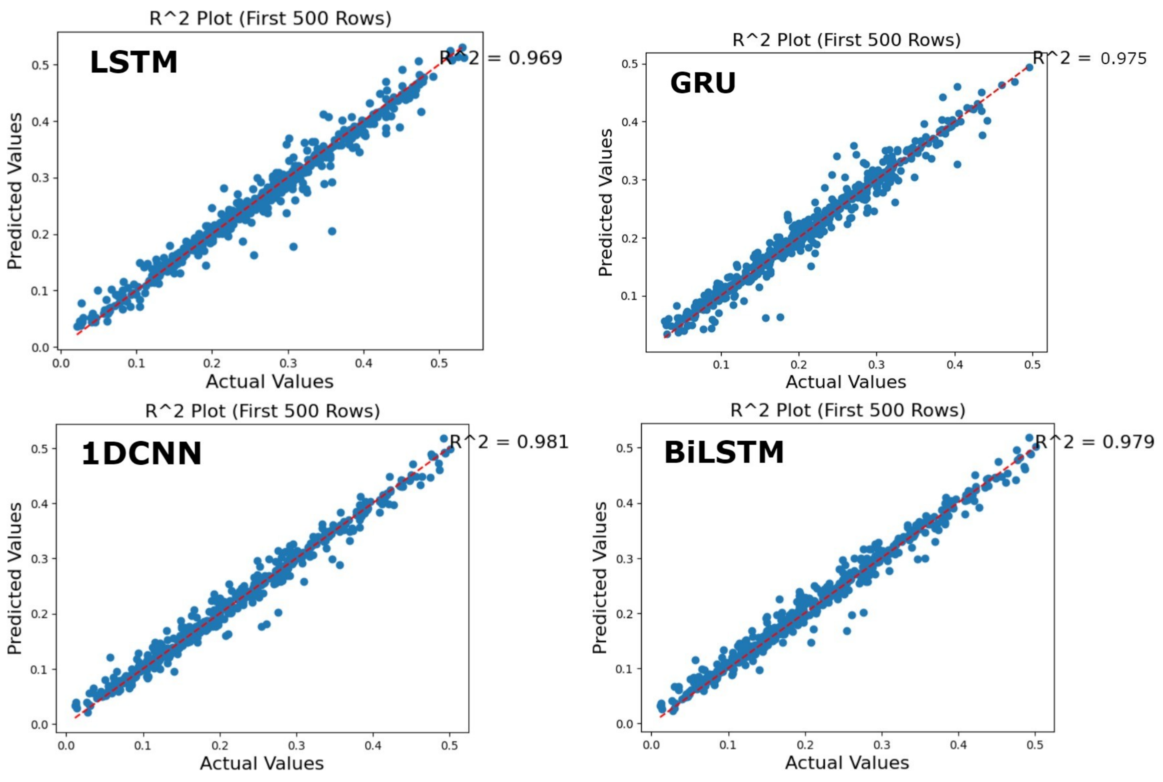

Figure 14: R squared plots for all models in Raichur dataset.

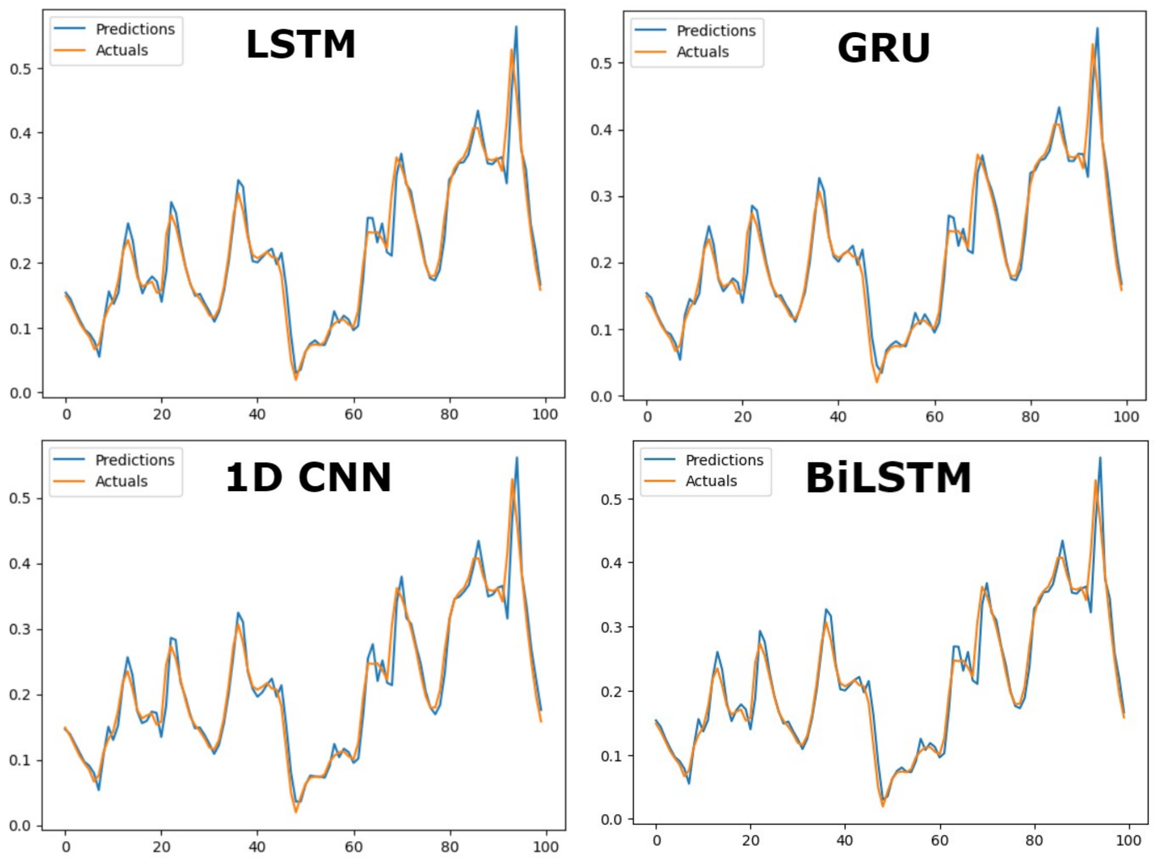

Figure 15: Predicted vs Actual wind speed values for all models in Raichur dataset.

From the above graphs(Figure 14,15), it can be inferred that the model is providing accurate predictions relative to the actual values. The inference is based on the close alignment observed between the predicted and actual values, as evidenced by the similarity between the two plotted curves. This concurrence indicates the model's effectiveness in capturing the underlying patterns within the data.

Table 2: Different Errors of the models of different regions

	R^2	**RMSE**	**MAE**	**MSE**
MANGALORE				
LSTM	0.97711	0.01964	0.01446	0.00039
GRU	**0.98301**	**0.01692**	**0.01224**	**0.00029**
ID CNN	0.98309	0.21964	0.15830	0.04824
BILSTM	0.98167	0.01758	0.01270	0.00031
DHARWAD				
LSTM	0.96976	0.02344	0.01669	0.00055
GRU	**0.97981**	**0.01915**	**0.01235**	**0.00037**
1D CNN	0.97714	0.02038	0.01314	0.00042
BILSTM	0.97000	0.01978	0.01234	0.00042
BENGALURU				
LSTM	0.96500	0.02519	0.01840	0.00063
GRU	0.96627	0.02473	0.01773	0.00061
1D CNN	**0.96875**	**0.02380**	**0.01744**	**0.00057**
BILSTM	0.96604	0.02360	0.17839	0.00059
BELLARY				
LSTM	0.97128	0.02184	0.01566	0.00048
GRU	**0.97663**	**0.01970**	**0.01276**	**0.00039**
1D CNN	0.97611	0.01992	0.01337	0.00040
BILSTM	0.96136	0.02533	0.01884	0.00064
BHADRAVATI				
LSTM	0.97686	0.02145	0.01443	0.00046
GRU	**0.98089**	**0.01950**	**0.01269**	**0.00038**
1D CNN	0.97589	0.02190	0.01562	0.00048
BILSTM	0.98033	0.01978	0.01297	0.00039

Table 3: Different Errors of the models of different regions

	Rˆ2	RMSE	MAE	MSE
RAICHUR				
LSTM	0.96347	0.02360	0.01807	0.00056
GRU	0.97556	0.01931	0.01277	0.00037
1D CNN	**0.97590**	**0.01917**	**0.01235**	**0.00037**
BILSTM	0.97552	0.01932	0.01273	0.00037
TUMKUR				
LSTM	0.96700	0.02275	0.01487	0.00052
GRU	**0.97421**	**0.02011**	**0.01223**	**0.00040**
1D CNN	0.97140	0.02117	0.01308	0.00045
BILSTM	0.97308	0.02054	0.01266	0.00042
DEVANGRE				
LSTM	0.98097	0.01912	0.01218	0.00037
GRU	0.98060	0.01930	0.01219	0.00037
1D CNN	**0.98171**	**0.01874**	**0.01191**	**0.00035**
BILSTM	0.98022	0.01949	0.01234	0.00038
VIJAYPUR				
LSTM	0.98012	0.01720	0.01060	0.00030
GRU	**0.98137**	**0.01665**	**0.01042**	**0.00028**
1D CNN	0.97944	0.01749	0.01082	0.00031
BILSTM	0.98112	0.01676	0.01038	0.00028

Table 4: Ranking models with respect to their performance and calculating their weighted average.

Location	**LSTM**	**GRU**	**1D-CNN**	**BiLSTM**
MANGALORE	4	1	2	3
DHARWAD	4	1	2	3
BENGALURU	4	2	1	3
BELLARY	3	1	2	4
BHADRAVATI	3	1	4	2
RAICHUR	4	2	1	3
TUMKUR	4	1	3	2
DEVANGRE	2	3	1	4
VIJAYPUR	3	1	4	2
WEIGHTED AVERAGE	3.4	1.4	2.2	2.8

Chapter 6

Conclusion and Future Work

In the investigation of wind speed prediction utilizing diverse deep learning architectures, including LSTM, BiLSTM, 1D CNN, and GRU, experimentation was conducted across 11 distinct datasets sourced from various geographical regions. Following comprehensive evaluation, it was determined that the GRU model consistently surpassed all other models across all datasets and geographic locales. This empirical evidence suggests the superior aptitude of the GRU model in effectively capturing the intricate temporal dependencies inherent in wind speed data. As such, the GRU model emerges as the preferred choice for wind speed prediction tasks, exhibiting heightened efficacy in capturing and leveraging temporal correlations. These findings underscore the potential of deep learning methodologies, particularly the GRU model, to enhance the accuracy and reliability of wind speed forecasting, thereby contributing to advancements in renewable energy forecasting technologies.

In anticipation of future research and development, several directions are envisaged. Firstly, we plan to dive into diverse approaches for signal decomposition, aiming to refine input preprocessing techniques. This endeavor seeks to enhance the model's ability to discern underlying patterns,remove noise from data thereby augmenting prediction performance. Furthermore, the research endeavors to extend its reach to include multivariate forecasting techniques integrating other environmental variables like temperature, humidity, and air pressure. This expansion aims to provide a more holistic understanding of the factors influencing wind speed dynamics. Moreover, there's also an aspiration to explore advanced hybrid forecasting models like VMD-LSTM Hybrid, complementing the deep learning approaches, and potentially capturing additional facets of underlying data patterns.