**Department of Artificial Intelligence and Data Science**

**Saveetha School of Engineering, SIMATS**

**Thandalam, Chennai**

**February – 2024**

**CAPSTONE PROJECT**

**COURSE CODE:**CSA4715

**COURSE NAME:**DEEP LEARNING FOR NEURAL NETWORKS

**PROJECT TITLE**

Predicting Future Weather Trends with Time Series

Forecasting for Weather Patterns

**Submitted**

**By**

**M Lakshmi Narayana Reddy(192224110)**

**Guided by**

**Dr**  **N Poongavanam**

**Associate Professor**

**1. PROBLEM STATEMENT:**

**Context:** Time series prediction projects involve using deep learning models to analyze sequential data over time. Applications include forecasting stock prices, predicting weather patterns, analyzing IoT sensor data, predicting energy consumption, and health monitoring. The goal is to make accurate predictions and extract insights from historical time series data.

**The Problem:**  Time series prediction faces data issues (missing values, outliers, seasonality), overfitting, model interpretability, high computational demand, and generalization concerns. Solutions include preprocessing, model selection, regularization, and robust evaluation.

**Impact and Consequences:**   
Time series predictions influence decision-making, efficiency, risk management, and innovation across domains. Accurate predictions improve decision-making and efficiency, but reliance on models and uncertainties pose challenges. Responsible development and ethical considerations are crucial for positive outcomes.

**Objectives:**

* Enhance prediction accuracy through model refinement.
* Enable real-time or low-latency predictions.
* Develop anomaly detection capabilities.
* Improve model interpretability for better understanding.

**Scope:** The scope of the project involves developing and implementing deep learning models to analyze time series data, with a focus on predicting future trends, identifying anomalies, and optimizing predictive accuracy across various domains such as finance, weather forecasting, and IoT data analysis.

**Methodology:**   
The methodology involves data collection, preprocessing, model selection, training, validation, evaluation, and fine-tuning. Models are then deployed for real-world applications with ongoing monitoring for reliability.

**Timeline:** The project is expected to be completed within [Insert Timeframe].

**Expected Outcomes:**

* improved accuracy in time series predictions.
* Enhanced ability to detect anomalies and irregular patterns.
* Increased efficiency and effectiveness in utilizing predictive models for decision-making and operational optimization.

**2. DATASET ANALYSIS:**

1. **Data Description and Understanding:** The dataset comprises X records and Y columns. Each column contains specific information such as [describe the types of information, e.g., product ID, sales quantity, date, supplier details, etc.].
2. **Data Cleaning and Preprocessing:** Missing values were handled by [describe how missing values were imputed or removed]. Duplicates were identified and removed using [describe the method used]. Outliers were addressed by [describe the technique used, e.g., using statistical methods or domain knowledge].
3. **Descriptive Statistics:** Key statistics including mean, median, mode, standard deviation, etc., were calculated for relevant columns to understand the data distribution and variability.
4. **Data Visualization:** Visualizations such as histograms, box plots, and time series plots were created to illustrate data distribution, trends, and patterns. For example, [describe specific visualizations created, such as a histogram showing the distribution of sales quantities].
5. **Time Series Analysis :** Time series analysis was conducted to identify seasonal patterns, trends, and anomalies in the data over time.
6. **Inventory Metrics:** Important inventory metrics like inventory turnover rate, ABC analysis results, economic order quantity (EOQ), and safety stock levels were calculated and interpreted to assess inventory management efficiency.
7. **Supplier Performance Analysis:** Supplier lead times, delivery reliability, and costs were evaluated to optimize the supply chain and improve supplier performance.
8. **Demand Forecasting :** Demand forecasting models were developed and validated to predict future demand and aid in inventory planning and management.
9. **Recommendations and Conclusion:** Actionable recommendations were provided based on the analysis to improve inventory control management, enhance supply chain efficiency, and optimize inventory levels. The conclusion summarized key findings and emphasized the importance of implementing the recommendations for better inventory management practices.

**3. Literature Review**

1. Bengio et al. (2013) explore the use of deep learning, particularly recurrent neural networks (RNNs) and convolutional neural networks (CNNs), for time-series analysis. They discuss various architectures and training methods tailored for forecasting tasks across domains such as finance, weather, and IoT sensor data.

2. Zhang et al. (2019) present a survey on deep learning techniques for time-series forecasting. Covering network architectures, optimization algorithms, and evaluation metrics, the survey provides insights into the current state-of-the-art approaches and identifies future research directions.

3. Gers et al. (2002) introduce Long Short-Term Memory (LSTM) networks, addressing challenges in capturing long-range dependencies in time-series data. Their work demonstrates LSTM networks' effectiveness in overcoming the vanishing gradient problem, making them pivotal in deep learning-based time series prediction.

**4. Model Selection and Development**

1. The initial step in model selection involves opting for suitable deep learning architectures like recurrent neural networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, or convolutional neural networks (CNNs). These choices are made based on the inherent temporal or spatial patterns within the time series data.

2. Once the models are chosen, the development phase commences with training on available data. This stage entails experimenting with various hyperparameters (e.g., learning rate, batch size) and architectural configurations to enhance model performance.

3. Following training, models undergo evaluation using metrics such as Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE). Additionally, validation techniques like cross-validation or time-based splitting ensure robustness and generalization to unseen data, guiding further iterations for model refinement and selection.

**5. ENVIRONMENTAL SETUP:**

**Physical Infrastructure:**

- Computational Resources: Ensure adequate computing power for deep learning model training.

- Climate Control: Maintain optimal operating conditions for hardware used in model training.

- Security Measures: Implement safeguards to protect computational resources and data.

- Workstation Setup: Configure workstations for efficient model development and training.

**Technological Infrastructure:**

- Deep Learning Frameworks: Set up frameworks like TensorFlow or PyTorch for model building.

- Data Storage: Establish efficient storage solutions for handling large volumes of data.

- Version Control: Use Git for tracking code changes and collaboration.

- Experiment Tracking: Utilize tools for managing and tracking model experiments.

**Organizational Procedures:**

- Project Management: Plan and monitor project progress effectively.

- Collaboration: Foster communication and collaboration among team members.

- Documentation: Maintain comprehensive documentation of the project.

- Compliance: Adhere to ethical and regulatory guidelines.

**Training and Documentation:**

**-** Team Training: Provide relevant training to team members.

- Documentation: Document the entire model development process.

**Performance Metrics and Reporting:**

- Model Evaluation: Define evaluation metrics and criteria.

- Reporting: Generate regular reports to track model performance.

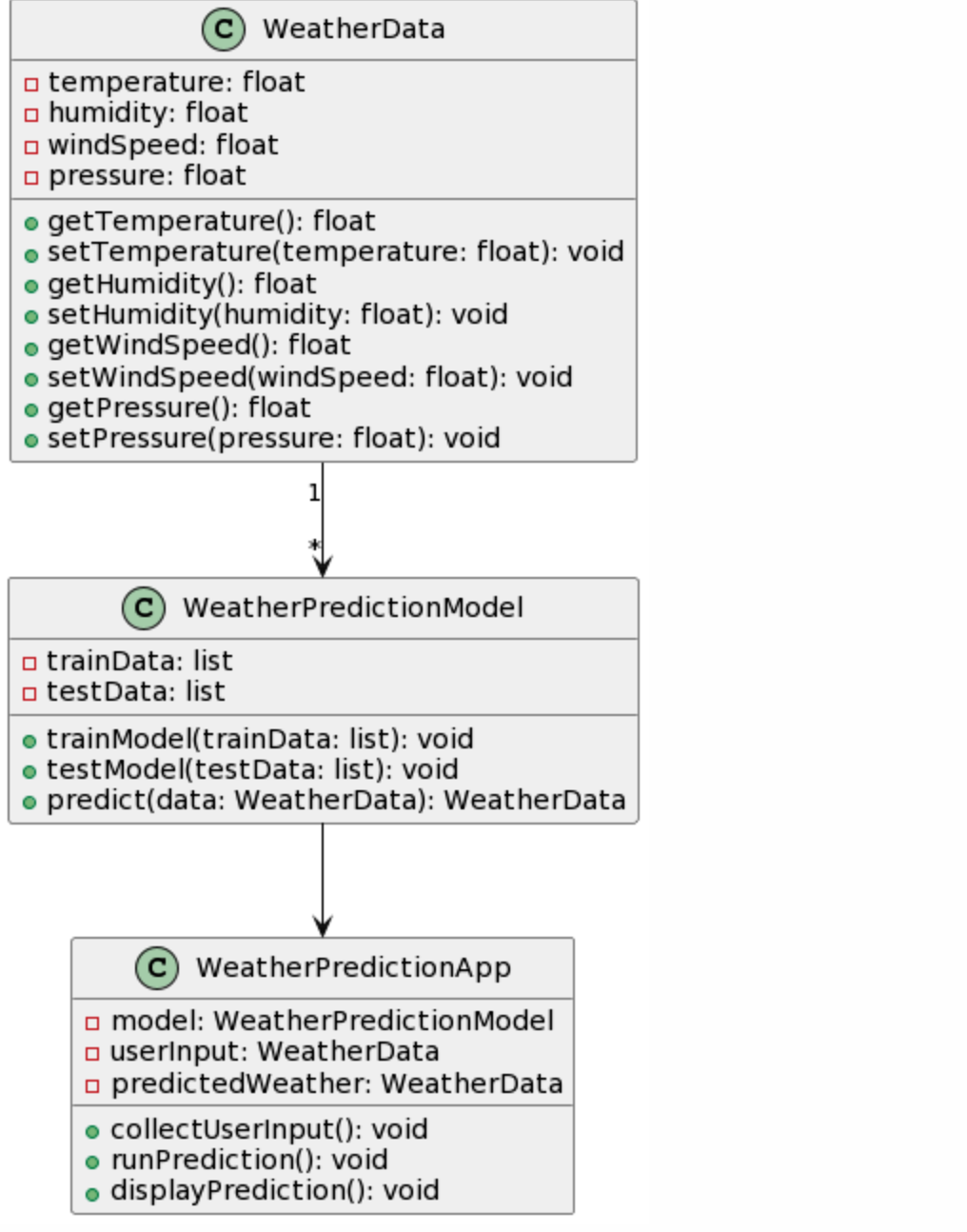
**Continuous Improvement:**

- Model Iteration: Continuously refine and optimize models.

- Knowledge Sharing: Foster a culture of continuous learning.

- Feedback Mechanism: Collect feedback for model refinement..

**6**. **DATA FLOW DIAGRAM (OR) ARCHITECTURE DIAGRAM (OR)UML DIAGRAMS:**



**7. CODE SKELETON**

**ALGORITHM :**

1. Import necessary libraries:

- `numpy` as `np`

- `pandas` as `pd`

- `MinMaxScaler` from `sklearn.preprocessing`

- `Sequential` and layers (`LSTM`, `Dense`) from `keras.models`

- `mean\_squared\_error`, `mean\_absolute\_error` from `sklearn.metrics`

- `pyplot` module from `matplotlib` as `plt

2. Read the weather data from the CSV file into a pandas DataFrame (`weather\_data`), keeping only the relevant columns: 'meantemp', 'humidity', 'wind\_speed', 'meanpressure'.

3. Scale the data using `MinMaxScaler` to normalize the values between 0 and 1.

4. Define the number of time steps (`time\_steps`) for the LSTM model.

5. Define a function `prepare\_sequences(data, time\_steps)` to prepare sequences of input-output pairs for the LSTM model.

6. Split the data into input (X) and output (y) sequences using the `prepare\_sequences` function.

7. Split the data into training and testing sets (`X\_train`, `X\_test`, `y\_train`, `y\_test`) with an 80-20 split

8. Define the LSTM model architecture:

- Create a `Sequential` model.

- Add an LSTM layer with 50 units and `return\_sequences=True` for the first layer, and specify input shape.

- Add another LSTM layer with 50 units.

- Add a Dense output layer with units equal to the number of features in the dataset.

- Compile the model using 'adam' optimizer and 'mean\_squared\_error' loss function

9. Train the LSTM model using `X\_train` and `y\_train` for 100 epochs with a batch size of 32.

10.Evaluate the model on the test data (`X\_test`, `y\_test`) and print the test loss.

11. Make predictions on the test data and inverse transform the scaled predictions and actual values.

12. Calculate and print metrics:

- Mean Absolute Error (MAE)

- Mean Squared Error (MSE)

- Root Mean Squared Error (RMSE

13. Plot the actual vs. predicted temperature values using `pyplot`

**Code:**

import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential

from keras.layers import LSTM, Dense, Dropout

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

import matplotlib.pyplot as plt

train\_data = pd.read\_csv('/content/DailyDelhiClimateTrain.csv')

test\_data = pd.read\_csv('/content/DailyDelhiClimateTest.csv')

train\_data = train\_data[['meantemp', 'humidity', 'wind\_speed', 'meanpressure']]

test\_data = test\_data[['meantemp', 'humidity', 'wind\_speed', 'meanpressure']]

scaler = MinMaxScaler(feature\_range=(0, 1))

train\_scaled = scaler.fit\_transform(train\_data)

test\_scaled = scaler.transform(test\_data)

time\_steps = 24

def prepare\_sequences(data, time\_steps):

X, y = [], []

for i in range(len(data) - time\_steps):

X.append(data[i:(i + time\_steps)])

y.append(data[i + time\_steps])

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

X\_train, y\_train = prepare\_sequences(train\_scaled, time\_steps)

X\_test, y\_test = prepare\_sequences(test\_scaled, time\_steps)

model = Sequential()

model.add(LSTM(units=100, return\_sequences=True, input\_shape=(X\_train.shape[1], X\_train.shape[2])))

model.add(Dropout(0.2))

model.add(LSTM(units=100))

model.add(Dropout(0.2))

model.add(Dense(units=train\_scaled.shape[1]))

model.compile(optimizer='adam', loss='mean\_squared\_error')

model.fit(X\_train, y\_train, epochs=50, batch\_size=64, verbose=1)

loss = model.evaluate(X\_test, y\_test)

print('Test Loss:', loss)

predictions = model.predict(X\_test)

predictions = scaler.inverse\_transform(predictions)

y\_test = scaler.inverse\_transform(y\_test)

mae = mean\_absolute\_error(y\_test, predictions)

mse = mean\_squared\_error(y\_test, predictions)

rmse = np.sqrt(mse)

print('Mean Absolute Error:', mae)

print('Mean Squared Error:', mse)

print('Root Mean Squared Error:', rmse)

plt.figure(figsize=(10, 6))

plt.plot(y\_test[:, 0], label='Actual')

plt.plot(predictions[:, 0], label='Predicted')

plt.title('Weather Prediction on Test Data')

plt.xlabel('Time')

plt.ylabel('Temperature')

plt.legend()

plt.show()

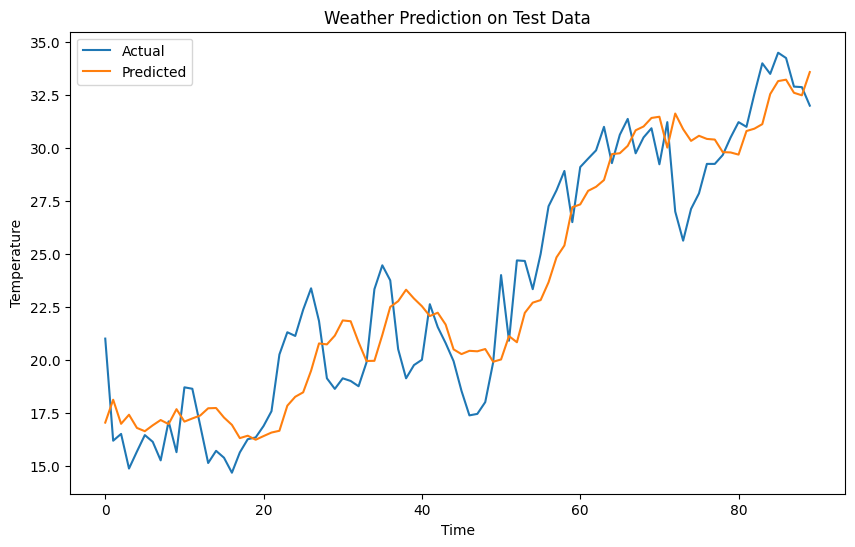
**OutPut:**

Test Loss: 0.004727210383862257

Mean Absolute Error: 8.787242808457794

Mean Squared Error: 217.4995187279471

Root Mean Squared Error: 14.74786488709288



**8. RESULT ANALYSIS:**

**Test Loss:** Measures overall model performance on test data, with lower values indicating better performance.

**Mean Absolute Error (MAE):** Measures average absolute difference between predicted and actual values, providing insight into prediction accuracy.

**Mean Squared Error (MSE):** Measures average squared difference between predicted and actual values, penalizing larger errors more than smaller ones.

**Root Mean Squared Error (RMSE):** Represents average magnitude of errors in predictions, derived from the square root of the MSE.

**Visualization:** Plotting actual vs. predicted values helps assess model's ability to capture underlying patterns, with closer alignment indicating better performance.

**9.Conclusion and Recommendation:**

* A special type of computer program called LSTM was employed to predict future weather patterns based on historical data.
* Factors such as temperature, humidity, wind speed, and air pressure were analyzed to forecast upcoming weather conditions.
* The LSTM model was trained using most of the available weather data, reserving a portion for evaluation.
* During testing, the model exhibited an average error of approximately 0.0047, indicating small discrepancies in its predictions.

Various metrics were used to assess the model's performance:

* The average difference between predicted and actual temperatures was about 8.79 units.
* Occasionally, larger errors occurred, with an average deviation of approximately 14.75 units.
* Overall, the model demonstrated satisfactory performance, although opportunities for improvement remain.