SHORT-TERM LOAD FORECASTING IN POWER SYSTEMS USING DEEP LEARNING ALGORITHMS

A PROJECT REPORT

submitted by

ASISH JOHNSON

TKM21EEPS06

the APJ Abdul Kalam Technological University

in partial fulfillment of the requirements of the award of the

Degree

of

Master of Technology

in

Power Systems

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

T.K.M COLLEGE OF ENGINEERING, KOLLAM

APRIL 2023

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

T.K.M COLLEGE OF ENGINEERING, KOLLAM

CERTIFICATE

This is to certify that the report entitled '**Short-Term Load Forecasting in Power Systems Using Deep Learning Algorithms '** submitted by **Asish Johnson** to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the Degree of Master of Technology in Electrical and Electronics Engineering is a bonafide record of the project work carried out by him under our guidance and supervision. This project report in any form has not been submitted to any other University or Institute for any purpose.

Dr. Mohammed Mansoor O
Project guide
Asst. Professor
Dept of EEE, TKMCE

(External Examiner)

Prof. Jibi P Mathew
Project Coordinator
Assistant Professor
Dept of EEE, TKMCE

Dr. Sabeena Beevi K
Head of Department
Professor
Dept of EEE, TKMCE

ACKNOWLEDGEMENT

First of all, I am indebted to the **God Almighty** for giving me an opportunity to excel in my effort to complete this project on time.

I am extremely grateful to **Dr. T. Shahul Hameed**, Principal, TKM College of Engineering, and **Dr. Sabeena Beevi. K**, Head of the Department, Department of Electrical and Electronics Engineering, for providing all required resources for successful completion of my project.

I am greatly obliged to **Dr. Mohammed Mansoor O**, Asst. Professor, Project Guide, Department of Electrical and Electronics Engineering, for his encouragement and support.

My heartfelt gratitude to **Prof. Shanavas T. N**, Associate Professor, PG co-ordinator, Department of Electrical and Electronics Engineering, for his valuable suggestions and guidance in the preparation of the project report.

I express my thanks to **Prof. Jibi P Mathew**, Asst. Professor, Project co-ordinator, Department of Electrical and Electronics Engineering, and all staff members and friends for all help and co-ordination extended in bringing out this project success fully in time.

I will be failing in duty if I do not acknowledge with grateful thanks to the authors of the references and other literatures referred to in this project.

Last but not the least, I am very much thankful to my parents and my wife who guided me in every step which I took.

Place: Kollam
Date: APRIL 2023

ASISH JOHNSON
TKM21EEPS06

Abstract

Short-Term Load Forecasting (STLF) plays a crucial role in power system planning and operation, as it helps utilities to efficiently allocate their resources and ensure reliable service to customer. In this project the performance of different forecasting algorithms such as Long Short-Term Memory (LSTM), Particle Swarm Optimization-Gated Recurrent Unit (PSO-GRU), Multivariate LSTM, and 1-Dimensional Convolution Neural Network-Long Short-Term Memory (1-D CNN LSTM) are evaluated. A widely used benchmark dataset namely Global Energy Forecasting Competition (GEFCOM) dataset is used in this work for training and performance validation. The performance of different load forecasting models is compared using performance indices like accuracy and Mean Absolute Percentage Error (MAPE). Among the different models used for short-term load forecasting, Multivariate LSTM model is found to be more accurate than other models. The results indicate that Multivariate LSTM is a promising approach for STLF, and its superior performance is attributed to its ability to handle multiple input variables. The study highlights the importance of model selection in accurate load forecasting and demonstrates the potential of Multivariate LSTM for STLF. The findings can help power system planners and operators to choose an appropriate STLF algorithm based on their specific needs and requirements.

Keywords: **short-term load forecasting; long short-term memory; convolutional neural network.**

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The amount of electricity required to meet the demand is foreseen through load forecasting. It aids in meeting demand and improves growth management. An essential part of power system power management is load forecasting [1]. The electric utility benefits from accurate load forecasting when making key choices about the generation and purchase of electricity, unit commitment, the reduction of spinning reserve capacity, the scheduling of maintenance plans, network planning, and infrastructure development. It is crucial for the dependability of a power system in addition to playing a significant part in lowering the cost of generation. The system operator prepares for load shedding, power purchases, hydropower scheduling, hydro thermal coordination, and turning on and off peaking units using the results of the load forecasts. For energy providers, autonomous system operators, financial firms, and other participants in the production, transmission, distribution, and energy markets, load estimates are crucial.

1.1 Background of the Project

Forecasts for load show when load growth will occur. Power firms use it to provide demand. Detrimental inaccuracy forecasting may have a negative impact on consumer production levels. The operational costs are decreased and plant and unit daily operations run smoothly with accurate load forecasts. The management of plants can save millions of rupees with a 0.5% increase in accuracy. The accuracy of the forecast is improved by taking into account the weather state, which plays a significant impact. The recent usage of clever strategies and the integration of a few technologies has contributed to minimising error. The network takes longer to converge because of the vast amount of previous data required for hourly load forecasts. Because of this, achieving minimal error and requiring little training time, the forecasting approach is accurate and useful [2]. The choice of superior training techniques aids in forecast accuracy. In order to estimate the load on the electrical system, forecast accuracy is essential. It has been challenging to predict future load using historical data, especially when it comes to holidays as well as days with harsh weather. Undoubtedly, it is difficult for energy companies and consumers to estimate their individual loads with accuracy. This problem has existed for many years. Because of this, numerous loads

forecasting strategies, spanning from conventional to intelligent systems, have been created to date and emphasised in numerous studies. Forecast accuracy can be used to determine the final difference between different approaches.

1.2 Load Forecast Horizons

Three distinct time frames are available for load forecasts: extremely short-term, lasting from a few seconds to hours; short-term, lasting from an hour over a week; medium-term, lasting from a week to a year; and long-term, lasting longer than a year. For capacity development plans, capital investments, and corporate budgeting, long-term forecasts are necessary. Due to upcoming uncertainties including political variables, the state of the economy, and per capita growth, among others, these estimates are frequently complicated. Medium-term projections cover plant and network maintenance and planned outages. Every day, the power system must schedule the power generation for the following day, which is done using short-term load forecasting (STLF). With developed models, STLF is discussed in this study. Figure 1.1 provides examples of several load forecasting techniques.

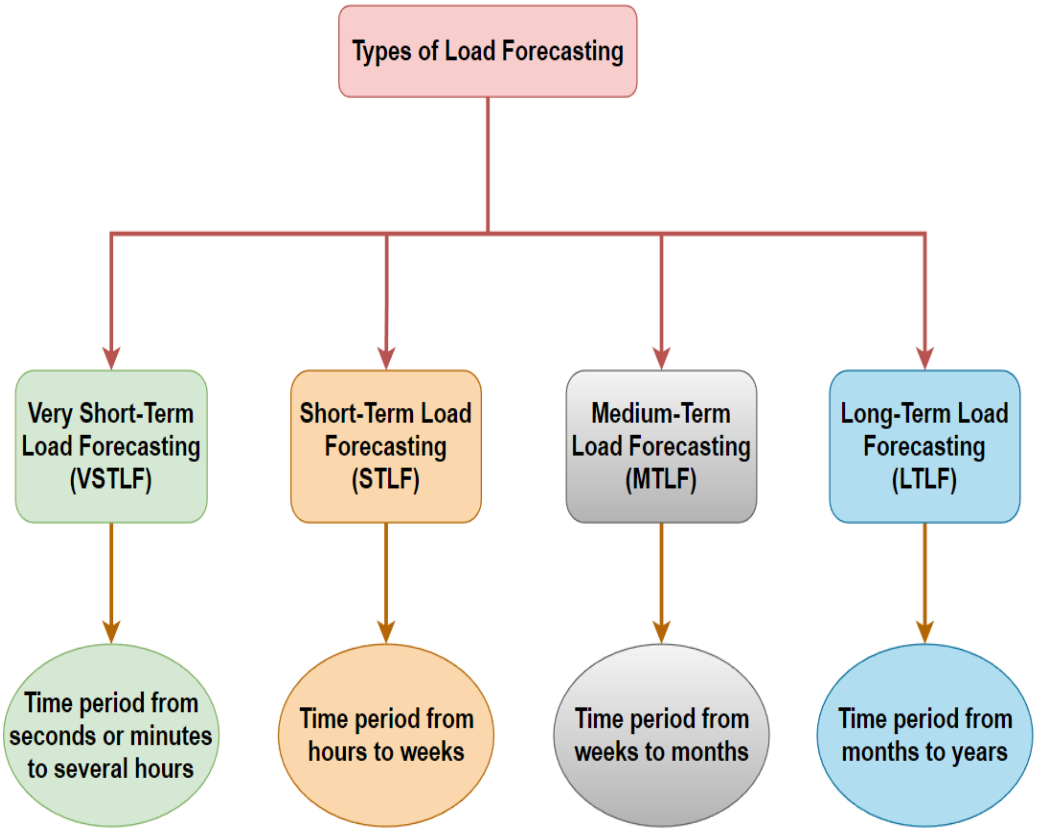

Fig. 1. 1 Types of Load Forecasting

For load dispatch, STLF is a daily work that is essential, and the correctness of this task has a significant impact on the system's efficiency and dependability. Underprediction of STLF results in insufficient reserve capacity preparation, which raises operating costs by requiring the usage of peak units. Overprediction of STLF results in too large reserve capacity, which is likewise associated to high operating costs. The most widely used methods for predicting load include time series-based models, similar-day approach, and models based on intelligent systems. Major disadvantages of several traditional forecasting techniques include their inadequacy to map the load's non-linear feature. Therefore, it is imperative to substantially replace traditional methodologies with intelligent system-based models. Statistical methods and artificial intelligence algorithms like neural network models, fuzzy logic, and expert systems are used in the majority of forecasting models [3].

Artificial neural networks (ANN) are widely used in STLF, compared to all other intelligent techniques. The majority of current load forecasting studies are ANN-based, and most of these studies have produced accurate forecasts. Since ANNs can generalise and learn non-linear correlations between variables, these methods are frequently chosen for STLF issues [4]. The capacity of ANNs to repeatedly change the synoptic weights between layers is one of its other key characteristics. On the other hand, conventional procedures demand static, difficult mathematical equations but nevertheless perform badly when compared to methods based on intelligence.

It may be possible to improve the predicting outcome with the recent advancement of new mathematical models, such as data mining as well as artificial intelligence techniques. The deregulation of the electricity market in recent years has increased the importance of STLF and raised its challenges. As a result, accurate forecasting aids in the trade of electrical energy and the establishment of spot prices for the system to obtain least purchasing costs from the electricity market.

1.3 Requirements of Load Forecasting

In most Energy Management Systems (EMS) and load dispatch centres, there is an STLF module. A good STLF system should fulfil the following requirements [5]:

i. Accuracy
ii. Speed
iii. Detection of bad data

iv. User friendly
v. Automatic forecasting

1.3.1 Accuracy

The accuracy of the STLF process' predictions is its most crucial requirement. Economic dispatch, system dependability, and trading in the power markets all depend on accurate data. Making the forecasting outcome as accurate as feasible is the primary objective of the majority of STLF literature, as well as of this thesis.

1.3.2 Speed

Accuracy is increased by using the most recent historical data as well as weather forecasts. The STLF application uses historical data with weather forecast data to shorten computer runtime and produce the predicted result as soon as possible. As a result, a fundamental element of the forecasting programme is forecasting speed. Programs with excessively long training times should be discontinued, and new methods for cutting training times should be used. Accuracy shouldn't be sacrificed while using new approaches that require less training time.

1.3.3 Detection of bad data

In the advanced power systems of today, measurement equipment is spread out across the entirety of the system, and communication connections are used to send the data that is gathered by that equipment to a centralised point that is responsible for command and control. The load statistics that reach the dispatch centre occasionally may be inaccurate as a result of intermittent measurement or communication failure, but they are nevertheless kept in the history database. This is the case even though the statistics may be inaccurate. This is the case despite the fact that the data might not be completely correct. At first, the STLF systems relied on power system administrators to manually find incorrect data and remove it from the system. The most recent development in this area is to have the system take care of it rather than the operators. This decreases the amount of manpower required of the operators and boosts the detection rates.

1.3.4 User friendly

The load forecasting interface should be simple, practical, and intuitive to use. The users can quickly specify what they would like to forecast as well as whether they want to do

so using tables or images. In order for users to readily access the output and save it for later use, it should also be presented in a numerical and visual style.

1.3.5 Automatic forecasting

Multiple models are frequently incorporated into a single STLF system. This is done with the intention of minimising the possibility of an individual making an imprecise prediction. In the prior, such a mechanism would always require the intervention of the operator, who would then choose the appropriate amount of weight to assign to each model in order to arrive at the combinative result. It would be helpful if the system could compute its final forecasting result based on the way previous days' forecasts played out. This would make using the system significantly more convenient.

1.4 Aims and Objectives of the Project

The aim of the project is to develop accurate and reliable models for short-term load forecasting in power systems. The project proposes the use of four different deep learning models: LSTM, PSO based GRU, Multivariate LSTM, and 1 Dimensional CNN based LSTM to predict the load demand of a power system for a short-term period, typically up to 24 hours in advance. The work also address the limitations of traditional statistical models used in load forecasting, which are often limited by their assumptions and inability to capture complex nonlinear relationships between the load demand and weather conditions, time, and other factors that affect electricity consumption. The proposed deep learning models leverage the power of neural networks to automatically learn these relationships and patterns from the historical data and make accurate predictions of future load demand.

To achieve this aim, the project will collect and pre-process historical load and weather data, and train and evaluate the performance of the proposed deep learning models using various performance matrix such as mean absolute percentage error (MAPE). The project will also compare the performance of the proposed models against each other and against existing state-of-the-art methods in the literature. The goal of the project is to develop a robust and accurate short-term load forecasting model that can help power system operators to make informed decisions in real-time, such as scheduling the generation and distribution of electricity, managing the grid stability, and reducing the cost of energy.

The objectives of the project are listed as follows.

- To develop an LSTM model for load forecasting. The LSTM model is a type of recurrent neural network (RNN) that can learn long-term dependencies in time series data. The LSTM model is trained on the pre-processed dataset and used to forecast the load for the next time step.
- To develop a PSO-based GRU model for load forecasting. The PSO algorithm is used to optimize the parameters of the GRU model, which is another type of RNN. The PSO-based GRU model is trained and evaluated on the same dataset as the LSTM model.
- To develop a Multivariate LSTM model for load forecasting. The Multivariate LSTM model can take multiple input features into account while forecasting the load. This model is also trained and evaluated on the same dataset as the other models.
- To develop a 1D CNN model for load forecasting. The 1D CNN model is a type of feedforward neural network that can learn spatial features from one-dimensional data.
- The final objective is to compare the performance of the different deep learning models developed in the project. The performance of the model is compared with accuracy and Mean Absolute Percentage Error (MAPE) which is the evaluation matrix to identify the best performing model.

1.5 Outline of the Thesis

The project starts w with the general background of load forecasting horizons and the requirements of load forecasting. Furthermore, the chapter also discusses about the project aim as well as objectives.

A thorough assessment of the literature is done in Chapter 2. The topics, load forecasting, integrated forecasting with short-term load forecasting, and machine learning and deep learning methodologies used for STLF, including both conventional and cutting-edge techniques are discussed in Chapter 2.

The emphasis in Chapter 3 discusses about the methodology and data set for the proposed system.

Chapter 4 discuss about the LSTM based STLF, its mathematical modelling as well as the simulation results and discussions.

Chapter 5 discuss about the PSO based GRU for STLF, its mathematical modelling as well as the simulation results and discussion.

Chapter 6 discuss about the multivariate LSTM based STLF, its mathematical modelling as well as the simulation results and discussions.

Chapter 7 discuss about the 1-D CNN based LSTM for STLF, its mathematical modelling as well as the simulation results and discussions.

Chapter 8 discuss about the comparison of the four proposed model to identify which model is better for STLF.

Chapter 9 discuss about the conclusion and future scope of the work.

Chapter 2

Literature Review

2.1 Preamble

This chapter discusses a review of the literature on load forecasting as well as its various types, as well as incorporated load forecasting of STLF. The STLF and its various methods, such as the similar day approach, regression-based methodology, time-series analysis, artificial neural networks, fuzzy logic, support vector machines, and data mining, are also discussed in the chapter.

2.2 Load Forecasting

A single load forecast could indeed meet all of the needs of the power system utility. It is common practise to use different forecasts for different applications. The classification of various forecasts is based not only on the utility's business needs, but also on access and availability of the critical factors that impact energy consumption: weather as well as human activities [6].

Weather refers to the current state of meteorological elements such as humidity, heat, rainfall, wind, and so on, as well as their trade cycle in a particular region over up to two weeks. Climate encompasses these very same components in a particular region as well as their long-term trade cycle. The next chapter will go over how to implement various weather variables. Because temperature has the greatest impact on energy consumption of any meteorological parameter, temperature data is used in all of the model was developed in this report. The methodology, however, can be applied to other meteorological elements. Temperature forecasts can now be comparatively precise approximately to one day ahead for load forecasting purposes, and imprecise and even though undeviating up to two weeks forward with. [6] Human activities can have an impact on energy consumption in a variety of ways. The impact of the hourly resolution changes depending on the calendar parameters, such as the month of the year and the day of the week. The calendar information for the next decade is usually certain. The impact of the monthly or quarterly resolution varies depending on economic conditions.

Economic information can be fairly accurate up to one year in front, and inaccurate and yet reliable up to three years ahead, thanks to advances in econometric techniques. Climate as well as economics can both influence energy consumption in the annual resolution. However, because both inputs are unavailable, the system-level load prediction can only be obtained by simulating various situations. Furthermore, urban development, which is accomplished through land use variations, influences long-term power consumption just on circuit level. Land use data is usually accurate within one year and dependable for up to five years. While some counties can obtain a 30-year urban development strategy, it is still unclear what will occur year by year over the next 30 years. The forecasting of electric load is classified as [3].

- Very Short-Term Load Forecasting (VSTLF)
- Short-Term Load Forecasting (STLF)
- Medium-Term Load Forecasting (MTLF)
- Long-Term Load Forecasting (LTLF)

Economics, land use, as well as temperature data can all be optional in VSTLF since the load short could be predicted using load data from the past. Land usage and economic data can both be freely used in STLF because they are both relatively stable over a short period of time (less than two weeks). However, STLF heavily relies on temperature information. The distribution of the load forecast, which is dependent on the distribution of the temperature forecast, should be used to guide decisions about how much electricity to purchase. For the next three years, the temperature in MTLF cannot be predicted with any degree of accuracy. As a result, the model can be used to simulate temperature situations using the local temperature data. Economics is necessary in MTLF even if it is predictable and influences mid-term load usage. So, because land use can almost certainly change significantly over three years, the land use data is unnecessary in VSTLF, STLF, as well as MTLF. Land use change, in contrast hand, is a significant component that propels the load over the long run. Consequently, land use data is required for LTLF. On the other side, since long-term economic and temperature predictions are challenging, simulated alternatives can be used [7].

Table 2. 1 Load forecasting classifications O: Optional, R: Required. S: Simulated

	Temperature	**Economics**	**Land use**	**Updating cycle**	**Horizon**
VSTLF	O	O	O	≤ one hour	1 day
STLF	R	O	O	1 day	2 weeks
MTLF	S	R	O	One month	3 years
LTRF	S	S	R	One year	30 years

2.3 Integrated Forecasting with STLF

Even while certain departments within a utility may cope with the same kind of projections, there is currently little contact between them, which leads to inefficient resource implementation. The constant usage of human resources results in comparable forecasts across departments, but there are issues with communication and information exchange that may cause these forecasts to have divergent outcomes. In a similar vein, the operations department on STLF uses SCADA data (Supervisory Control and Data Acquisition System), which could not be as accurate as the billing data utilised by the trading department. Similar to the previous example, the trading division might not be aware of a continuous outage, which would lead to a less accurate prediction for the next few hours. In this dissertation, a brand-new STLF model powered by AI is established for such short-term load forecasting (STLF) procedure. Due to its inherent relationship with many forecasts, STLF is recommended as the engine [8],[9].

• Short Term Load Forecasting (STLF) to Very Short-Term Load Forecasting (VSTLF):

By including loads from a few process hours and inputs to a STLF model, which maintains the autocorrelation between the current hour demand as well as the prior hour loads, a Short-Term Load Prediction model can be converted it into Very Short-Term Load Prediction models. Alternately, historical load effects can be combined to create a new pattern using the STLF as either a reference. The next step is to create a very short-term prediction by predicting future impacts and adding these back to the short-term forecast [10].

• Short Term Load Forecasting (STLF) to Medium Term Load Forecasting (MTLF) and Long-Term Load Forecasting (LTLF):

The system-level LTLF as well as MTLF could be accomplished by incorporating an econometrics component into the STLF model and extending the model's scope to encompass a broader boundary. Additionally, this will allow for the achievement of the STLF. As a result, this long-term forecast at the system level can be utilized as an input to long-term structural load forecasting [11].

2.4 Short-Term Load Forecasting

A short-term load forecasting system is required for precise scheduling, cost-effective load dispatch, and the management of power systems. It predicts loads with a trailing time with

one hour to seven days. [12] It has played a significant role in energy management systems (EMS). Short-term load forecasting (STLF) is crucial for effective and economical management in electrical utilities. The two most significant requirements for short-term load forecasting are high forecasting accuracy as well as speed, and it is crucial to study the load patterns and pinpoint the key factors influencing the load. The traditional load in electricity markets has an impact on variables including day type, seasons, climate, and cost of electricity that are voluntarily chosen and may have a complex relationship on system load.

2.5 Methods for Short-Term Load Forecasting

To increase precision and effectiveness, a number of forecasting techniques have indeed been applied to short-term load forecasting. These methods can generally be divided into two categories: traditional and modern. Traditional statistical load forecasting methods, including time - series data, regression, Kalman filtering, pattern recognition, etc., have been in use for a while, demonstrating the system dependence of forecasting accuracy. Employing weighted multi-model forecasting models, these conventional methodologies can be merged to produce predictions that are accurate in real-world systems. However, these techniques fall short in capturing the intricate nonlinear interactions that exist between the load and a variety of influencing factors, many of which are reliant on changes in the system (season or time of a day). [13]

2.5.1 Similar Day Approach

A similar day strategy is based on looking up historical data from one, two, or three years that have traits similar to the forecasted day. The traits include a comparable weekday or date as well as comparable weather. The forecast is taken to be a load from a day like that. Now, on the other hand, forecasting is finished with linear combinations and regression techniques by picking certain similar days, as opposed to only taking one similar day. On the basis of the progression coefficients from like days in prior years, the concern day's projection is made [14].

2.5.2 Regression-Based Approach

The term "regression" was first used to describe a biological phenomenon in the nineteenth century, namely the tendency of the offspring of extraordinary people to be less outstanding from their parents and more similar to their more distant relatives. The technique of linear regression compares a dependent variable to a predetermined independent [15]. First,

all independent variables are looked at because, regrettably, they change. The price or demand for electricity is typically the dependent variable in energy forecasting because it foretells output, which is dependent just on independent variables. Weather-related independent variables include things like humidity, temperature, and wind speed. The sensitivities of the predictor variables and how it changes in relation to the independent variable are indicated by the slope coefficients. Furthermore, by evaluating the historical significance of each independent variable in relation to the dependent variable the dependent variable's future value can be roughly predicted. Regression analysis' primary goal is to quantify the level of relationship between the independent variable and dependent variables in order to determine the independent variables' approximations. One of the most used statistical methods is regression. Regression techniques are typically employed in electric load forecasting to model the link between load consumption as well as other parameters including day type, weather, and customer class. [15]There are many regression models available for predicting next-day peaks. Their models take into account deterministic factors like random variables, holidays, factors like average loads, as well as extrinsic factors like the weather.

2.5.3 Time series analysis

The foundation of time series forecasting is the notion that trends in a time series plot can be modelled and extrapolated to the future in order to produce accurate forecasts [16]. Time series analysis develops a model based on seasonality and trend utilizing historical data as input. Time series models are generally complex and call for a lot of previous data, but they can be accurate in some circumstances. Additionally, careful measures must be taken to guarantee a correct timing throughout the operations of data gathering, filtering, modelling, and recall. In military management, time - series data analysis is typically used to forecast customer demand for products and services. Time series methods are not frequently employed in forecasting the energy sector. because they frequently ignore another important component, like weather forecasts.

In areas like signal processing, economics, and electric load forecasting, time series have been employed for a very long period. The most popular traditional time series methods include, in specific, ARMA (autoregressive moving average), ARIMA (autoregressive integrated moving average), ARMAX (autoregressive moving average with exogenous variables), as well as ARIMAX (autoregressive integrated moving average with exogenous variables). [17]While ARIMA is an adaptation of ARMA towards non-stationary processes,

ARMA models are frequently utilised for stationary systems. Time and load are the sole input parameters used by ARMA and ARIMA. In light of the fact that load typically relies on the weather and the time of day, ARIMAX is the traditional time series models' most logical instrument for load forecasting.

2.5.4 Artificial Neural Networks

Electronic models of artificial neural networks (ANNs) based on the neural architecture of the brain are still in their very early stages of development. We are aware that the brain naturally gains knowledge through experience [18]. The biologically inspired techniques are regarded as a significant development in the computational field. The fundamental unit of processing in such a neural network is the neuron. These neurons receive information from a source, combine it, carry out all necessary actions, and then produce the desired output. Since the middle of the 1980s, artificial neural networks have been developed and are used extensively. They are particularly effective at solving a variety of issues, including pattern recognition and clustering. Forecasting is based on patterns seen in past events and makes estimations of future values.

For two reasons, ANN is well suited to forecasting. First, it has been demonstrated that ANNs can numerically approximate any continuous function that is carefully designed. The ANN is viewed as a nonlinear, multivariate, and nonparametric approach in this situation. Additionally, ANNs are data-driven methodologies in that they eliminate the need for the researcher to first estimate the parameters of tentative models [19]. The relationship among input and output can always be mapped by ANNs since they learn this relationship as well as store it in their parameters. One hour load at the a time forecasting is the first method. The second method entails building a system having 24 NNs running concurrently, one for every hour during the day. The backpropagation technique is the most widely used training algorithm when creating a model that fits the input so well that it eventually incorporates a component of the Multi-Layer Perceptron (MLP) architecture of the neural network. Since these algorithms are iterative, they must be stopped by particular conditions. There are two criteria used: either the training is terminated after a predetermined number of repetitions or when the error has fallen below a predetermined tolerance. This criteria is insufficient since it only ensures that the model is fit the training data accurately; it does not, however, ensure good performance and may even cause the model to be overfit. "Over-fitting" refers to the error's randomness in its architecture, which leads to subpar outcomes [20].

The MLPs model is overtrained and overly intricate. Making use of cross-validation is one technique to prevent overtraining. A training set as well as a validation set are created from the sample set. On the training dataset, the neural network variables are predicted, and on the validation set, every iteration of the model is evaluated for performance. The iterations are paused and the last set of variables to be catalogued is used to generate the forecasts when this performance begins to deteriorate, which indicates that neural network has meeting expectations the training data. Electrical engineers should choose a few fundamental factors before using the ANN to problems involving electric load forecasting, such as:

- An input parameter (load, temperature etc)
- Day/season (weekday, weekend, season)
- Hourly loads, the peak load the following day, the overall load the following day, etc.
- ANN structure (Feedforward, number of hidden layers, number of neurons in the hidden layer etc)
- Training procedures and termination standards
- Activation mechanisms
- Size of the training and test sets of data

2.5.5 Fuzzy Logic

Boolean logic, which is typically used in digital circuit design, is a prerequisite for fuzzy logic. Boolean logic allows for the true value to be supplied as either 0 or 1. The input for fuzzy logic is connected to the comparison of attributes. For instance, we may state that the load on a generator could be high or low. We can rationally deduce outputs from inputs using fuzzy logic. The fuzzy so makes mapping among inputs and outputs, such as curve fitting, easier. The benefit using fuzzy logic is that neither accurate and even noise-free inputs nor formal models of the mapping between input and output are necessary. [21]According to the conventional rules, appropriately constructed fuzzy logic algorithms are very useful for anticipating electricity load. There are lots of situations where we need exact results. Defuzzification is carried out to produce correct results once fuzzy logic has been used throughout the entire operation. We are aware that a variety of load factors, including the economy, the weather, various load components, and social activities, have an impact on the load on the power system. Making an accurate estimate using previous load data analysis is difficult. These intelligent techniques, such as expert systems, fuzzy logic, and neural networks, have an edge over more traditional technique.

2.5.6 Support Vector Machines

The most precise and recently developed methods for solving classification and regression issues are Support Vector Machines (SVM) [22]. This strategy became popular thanks to Vapniks' work and statistical learning theory. Support vector machines are using the non - linear mapping of a data into high - dimensional data features by primarily using the kernel functions, in contrast to neural networks as well as other intelligent systems that attempt to deal with the convoluted functions of the inputs [23]. With support vector machines, researchers construct linear decision limits in the new space using straightforward linear functions. Within case of a neural network, this issue is with the architecture selection, but with a support vector machine, the issue is with selecting an appropriate kernel. For the purpose of anticipating the short-term electricity load, author of [24] used the support vector machine technique. He contrasts the effectiveness of this strategy with autoregressive method. According to the findings, SVMs outperform the autoregressive approach.

2.5.7 Data Mining

The practise of analysing data information in a sizable database to find patterns, knowledge, etc. is known as data mining. The approach is based on a hybrid strategy that combines either artificial neural network and an ideal regression tree. It divides the load range into different classes and decides, using the classification rules, which class the anticipated load falls within. The sample data for each class is trained using the multi-layer perceptron (MLP) [25]. The focus of the work is on describing underlying nonlinear relationship between the variables used as input and output in a prediction model.

2.5.8 PSO based GRU

Recurrent Neural Networks (RNNs) are powerful tools for modelling sequential data, and they have been used for various applications, such as speech recognition, natural language processing, and time-series analysis. However, traditional RNNs suffer from the vanishing gradient problem, which makes it difficult to learn long-term dependencies. To overcome this problem, the Gated Recurrent Unit (GRU) was introduced [26], which allows the network to selectively remember or forget information.

Particle Swarm Optimization (PSO) is a popular metaheuristic optimization technique that has been widely applied to various fields, including machine learning, data mining, and optimization. In recent years, there has been growing interest in combining PSO with neural

networks to improve their performance. The authors in [27] proposed a PSO-based RNN model for time-series prediction. The model combines the advantages of PSO and RNNs to improve the prediction accuracy. The PSO algorithm is used to optimize the weights of the RNN, and the results show that the proposed model outperforms traditional RNNs and other machine learning models in terms of prediction accuracy.

In the work [28], the authors proposed a PSO-based GRU model for fault diagnosis in a nuclear power plant. The PSO algorithm is used to optimize the parameters of the GRU, and the results show that the proposed model outperforms traditional GRU models and other machine learning models in terms of prediction accuracy.

2.6 Summary

This chapter does a good job of discussing the various load forecasting methods as well as the AI-based intelligent methods that have been utilised for STLF.

Chapter 3

Methodology and Dataset

3.1 Preamble

The research technique as well as the data set collection and data pre-processing applied in this study are provided in this chapter. The methodology adopted for this project is well explained as well as the source of the data achieved and the steps of pre-processing the data set is also explained well in this project.

3.2 Methodology

The methodology of the STLF can be obtained from the following steps and with the help of the Figure 3.1.

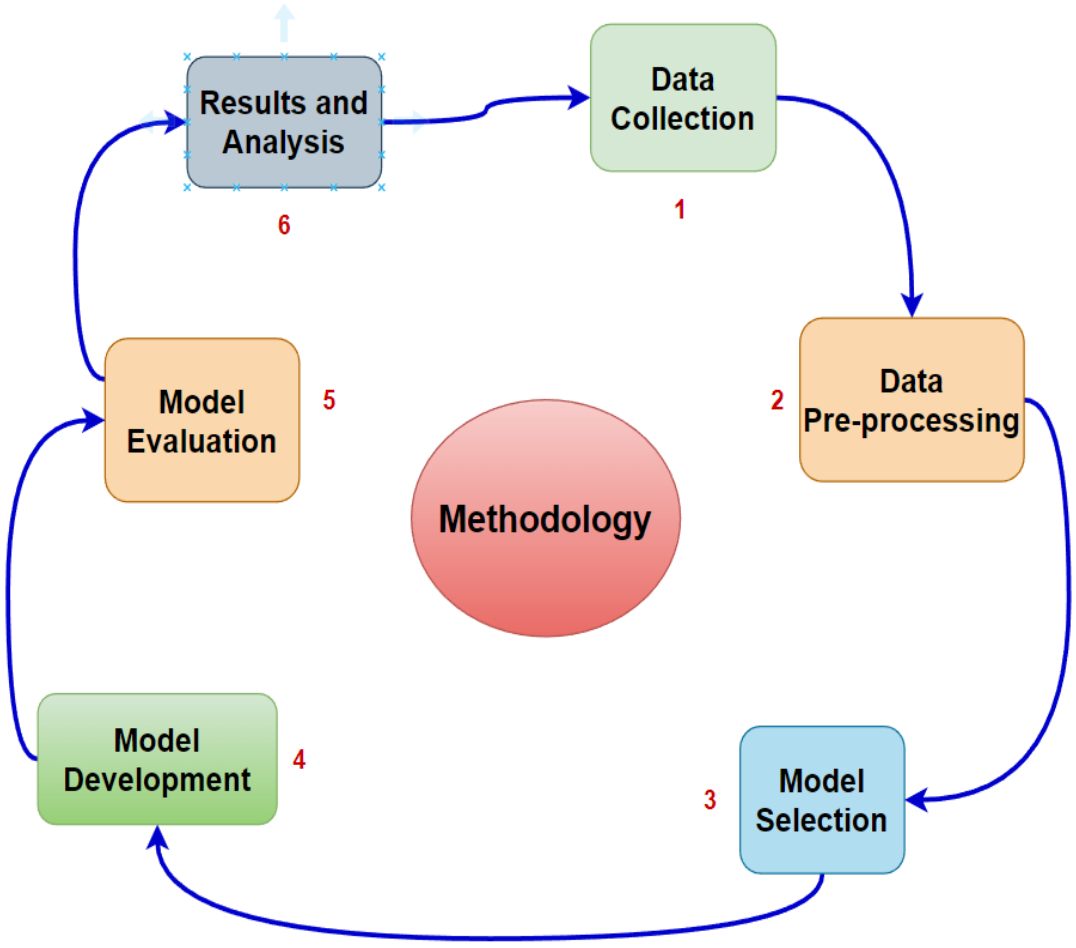

Fig. 3. 1 Methodology of Proposed System

3.2.1 Data Collection

The first step in this project is to collect the data. More than 300 students and experts from over 30 countries competed in the 2017 Global Energy Forecasting Competition (GEFCom2017) to solve multilayer probabilistic load forecasting issues [29]. 10,48,575 data points of data from 03 January 2003 to 17 June 2009 were taken into account as data set. The data set is split into two categories for the proposed STLF: 80% data for training and 20% data for testing. Using Google Collaboratory, the entire procedure is carried out, and the outcomes are plotted. The following subsections outline the research technique used for the suggested system. Table 3.1 depicts the model brief used to study this work.

Table 3. 1 Model Summary of Proposed Methodology

```
ls_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100)               40800

 flatten (Flatten)           (None, 100)               0

 dense (Dense)               (None, 1500)              151500

 dense_1 (Dense)             (None, 100)               150100

 dense_2 (Dense)             (None, 1)                 101

=================================================================
Total params: 342,501
Trainable params: 342,501
Non-trainable params: 0
_________________________________________________________________
```

3.2.2 Data Pre-processing

The collected data needs to be pre-processed to make it suitable for the forecasting models. The data pre-processing steps include data cleaning, normalization, and feature engineering. Data cleaning involves removing any missing or duplicate data points. Normalization is performed to scale the data to a common range. Feature engineering is performed to create new features from the existing features, which can help in improving the forecasting accuracy.

The following steps can be followed for data pre-processing.

1. Import the necessary libraries: Pandas, NumPy, and Scikit-learn for data manipulation and pre-processing.
2. Load the GEFCom2017 dataset into a Pandas data frame.
3. Convert the "Date" column into a datetime object and set it as the data frame index.
4. Handle missing values and outliers. Check for any missing values in the dataset and fill them using forward or backward filling. Outliers can be handled using Z-score or IQR methods.
5. Split the data into training and testing sets. In this project, 80% data for training and 20% data for testing.
6. Normalize the data. This is done to ensure that all features are on the same scale and can be compared equally. In this project, the Min-Max-Scaler from Scikit-learn is used to normalize the data.
7. Create the input and output sequences for the neural networks. For the LSTM and Multivariate LSTM models, the input sequence is created by stacking the previous hour's load data for each hour of the day, and the output sequence is the load data for the next hour.
8. For the 1D CNN model, the input sequence is created by stacking the previous 24 hours' load data for each hour of the day, and the output sequence is the load data for the next hour.
9. For the PSO based GRU model, the input sequence is the same as that of the LSTM model, and the output sequence is also the same.
10. Reshape the input sequences to fit the input shape of the neural networks.
11. Save the pre-processed data to a file for later use in training the neural networks.

3.2.3 Model Selection

Four different models are used in this project for short-term load forecasting. These models are LSTM, PSO-based GRU, Multivariate LSTM, and 1 Dimensional CNN. These models are chosen based on their proven success in time series forecasting tasks.

3.2.4 Model Development

The next step is to develop the models. Each of the four models is implemented and trained using the pre-processed dataset. The hyperparameters for each model are tuned using a grid search approach to optimize the model's performance.

3.2.5 Model Evaluation

Once the models are trained, they are evaluated using various performance metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE) along with the accuracy. These metrics are used to compare the performance of the four models.

3.2.6 Results and Analysis

The final step is to analyse the results obtained from the four models. The results are compared based on the performance metrics, and the best model is selected. The analysis also includes a discussion of the strengths and weaknesses of each model.

3.3 Flowchart of Proposed System

Figure 3.2 presents an overview of the proposed system in the form of a flowchart. Following the collection of the data and its subsequent analysis and selection using the load data selections, the results are shown. It is necessary to do pre-treatment on the load data in order to obtain the fine-tuned data set before processing the load data. Now that the proper load forecasting model has been used, the parameter estimation and distinguishing processes have been carried out. The error analysis is performed after the estimation, and if the process does not achieve the target value, the proposed forecasting model is improved by adding parameters that are more accurate, and then the process is repeated. In the event that there are no errors, the output parameters will be shown in the form of charts.

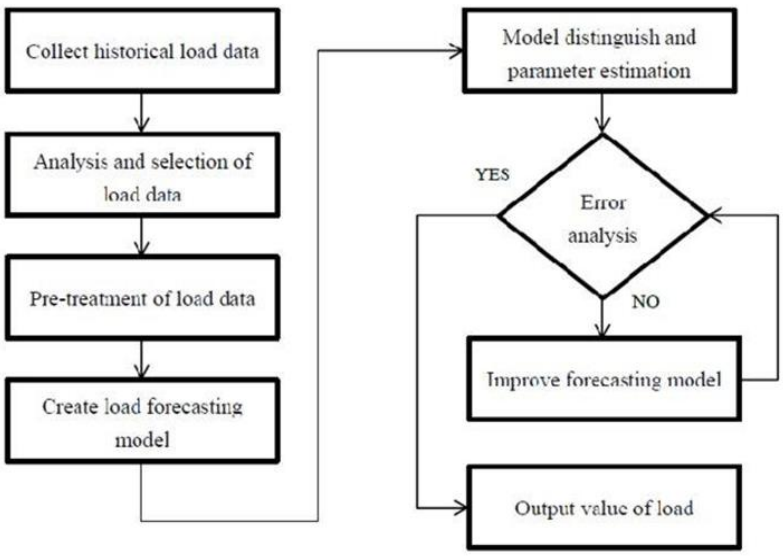

Fig. 3. 2 Flow chart of Proposed System

3.4 Summary

This chapter discuss well about the methodology of the proposed system and from where the data set is taken and how the data pre-processing is carried out to perform the proposed four models to predict the STLF.

Chapter 4

Short-Term Load Forecasting Using Long Short-Term Memory (LSTM)

4.1 Preamble

The classical model and the standard architecture of RNNs are explained in this chapter. Later on, the architecture of LSTM, design of LSTM as well as the implementation of LSTM for STLF is carried out. Then, the chapter plots and discusses the real load demand plot, the load demand plot for the first 72 hours, and the demand curves with a training demand starting at hour 73. The loss function plot using LSTM is also covered in this chapter, and lastly, the model's accuracy is projected. This chapter also plots and discusses the existing and anticipated demand. Finally, the effectiveness of the suggested LSTM model in STLF is also evaluated using accuracy and Mean Absolute Percentage Error (MAPE).

4.2 Classical Model

The Elman neural network incorporated a layer in the feedforward network's hidden layer that behaved like a step delay operator in order to accomplish its purpose of storing information in memory. Because of this, the model is capable of adapting to time-varying systems and may accurately describe the dynamic features of dynamic-process systems.

The following model is shown [30]:

$$y(k) = g(\omega^3 x(k)) \tag{4.1}$$

$$x(k) = f(\omega^1 x_c(k) + \omega^2\big(u(k-1)\big)) \tag{4.2}$$

$$x_c(k) = x(k-1) \tag{4.3}$$

where x is an n-dimensional hidden-layer unit vector, u is an r-dimensional input vector, and x_c is an n-dimensional feedback state vector. y is the m-dimensional output node vector. Weights for undertaking-to-output, input-to hidden, and undertaking-to-hidden are $\omega^1, \omega^2, \omega^3$, respectively. The transfer functions of hidden-layer neural cells and input neural cells, respectively, are g(.) and f(.).

4.3 Architecture of Standard RNNs

The ability of RNNs to make use of sequential data is one of their primary selling points. When designing a classic neural network, it's indeed common practise to proceed under the assumption that all of the input parameters (and output values) are unrelated to one another. However, in regard to the STLF task, that is awful. If the objective is to forecast the next quantity of load in such a time - series data, it is preferable to have an understanding of the origin of the value and how it was determined [31].

RNNs are given the name "recurrent" due to the fact that they carry out the same operation each and every element in a sequence, and thus the performance parameters are dependent on the computations that came before them. In particular, the memory section of Figure 4.1 illustrates the construction of a recurrent neural network (RNN).

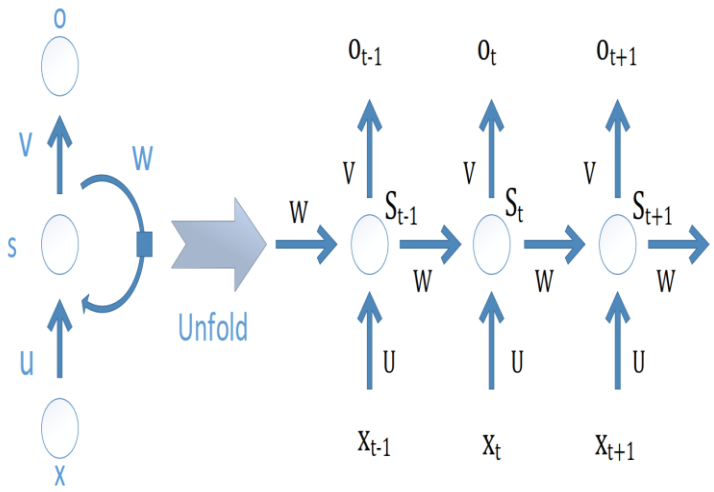

Fig. 4. 1 The Architecture of Standard RNN

The following equations serve as the basis for the computation that takes place inside of an RNN:

$$s_t = f(U_{x_1} + W_{S_{t-1}}) \quad (4.4)$$

$$O_t = softmax(V_{S_t}) \quad (4.5)$$

$$cost = \sum_t f_{err}(O_t, \widehat{O_t}) \quad (4.6)$$

where S_t is the concealed state on time step t and x_t would be the input at step t. The "memory" network is it. The output for step t is called O_t, while S_t is a calculation based on the prior hidden layer as well as the input just at current step.

Here are a few things to take note of:

- The occult state S_t is a candidate for network memory. S_t has the ability to record details about prior time step events.
- All steps of an RNN use the same variables (U, V, and W). This significantly lowers the overall number of factors that must be learned.
- Each time step in the flowchart above has an output, however depending on the task, this may not be essential. For instance, when forecasting the load values for a given day using multiple sets of data sets, we may be more interested in the results overall rather than the numbers following each piece of load data.

A RNN can be trained similarly to a conventional neural network. You can use the backpropagation algorithm, but with a slight modification. The gradients at each depends essentially not just on the computations of a current time step but also on the computations of the prior stages so because parameters were shared by all the time steps inside the network. For now, just be aware that the disappearing or expanding gradient problem causes ordinary RNNs trained with BP to struggle when trying to learn long-term dependencies, such as interconnections across steps that are far apart. This explains why RNNs do poorly in lengthy sequence modelling, a finding that points to RNN memory cell flaws [32]. LSTM networks can perform well in handling such problems for a series of extended sequences of electrical loads that may have various durations and local volatility.

4.4 Architecture of Standard LSTM

In 1997, Hochreiter and Schmidhuber were the ones that came up with the idea for the LSTM model [33]. Figure 4.2 provides a representation of the LSTM in its most fundamental form. Standard RNNs can't learn long-term dependencies because of the "vanishing gradient problem." Through a gating mechanism, LSTMs were made to deal with gradients that go away. Rather than a unit that just appears to apply an element-wise non - linearity towards the new input transformation as well as recurrent modules, LSTM recurrent connections have "LSTM cells" that have an inner recurrence in furthermore to the outer occurrences of the RNN.

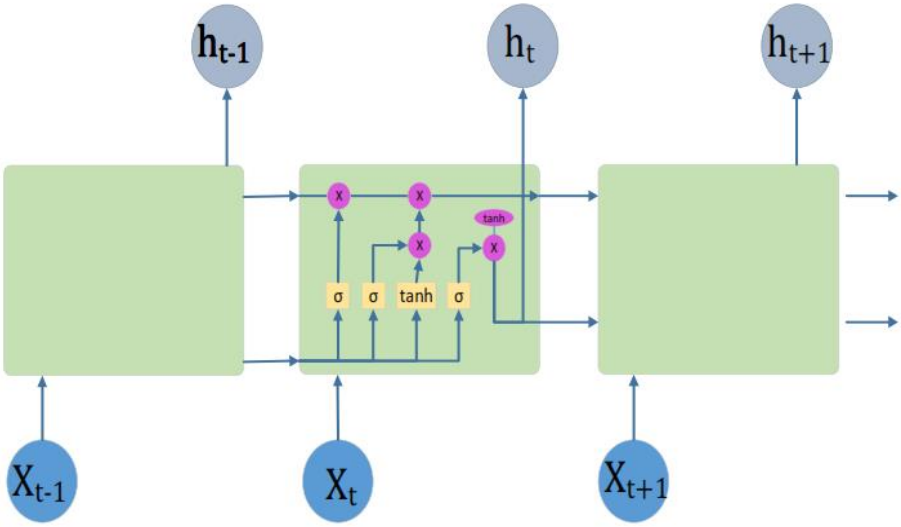

Fig. 4. 2 The Model of LSTM

To figure out what this means, here is how an LSTM found a hidden state:

$$i = \sigma(x_t U^i + S_{t-1} W^i) \quad (4.7)$$

$$f = \sigma(x_t U^f + S_{t-1} W^f) \quad (4.8)$$

$$o = \sigma(x_t U^0 + S_{t-1} W^0) \quad (4.9)$$

$$g = tanh(x_t U^g + S_{t-1} W^g \quad (4.10)$$

$$C_t = C_{t-1} o\, f + \text{g o i} \quad (4.11)$$

$$S_t = \tanh(c_t)\, o\, 0 \quad (4.12)$$

The phrases "input," "forget," and "output" are used to designate to the various types of gates. These terms are derived from equations (7), (8), and (9). The sole variation is in the format of their parameter matrices, which is different from one another. Aside from that, the equations themselves are exactly the same. The value of these vectors is condensed to a range between 0 and 1 as a consequence of the sigmoid function, giving rise to the term "gates." These gates calculate the maximum amount of information that can pass through them by multiplying one vector with another. The present input's input gate is responsible for determining the proportion of the newly computed state that will be permitted to pass through. The forget gate is responsible for determining the amount of the previous state that can be carried over into the new state. In conclusion, the output gate is responsible for determining

what percentage of the internal state will be displayed to the public networks (higher layers as well as the subsequent time step). This is done by calculating the percentage of the internal state that will be displayed.

Equation (10) has a "candidate" hidden state denoted by the letter g. This "candidate" hidden state is computed by using the most recently visible input along with the hidden state that came before it. It is precisely the very same equation in normal RNN; the only difference is that the names of the variables U and W have been changed to U^g and W^g respectively. However, the input gate was utilised from above to choose some of it, rather than taking as the possible hidden state as was done in the RNN. This was done so that more of the information could be used. Memory contained within the unit is denoted by the symbol C_t in Equation (11). It is a mixture of the prior memory C_{t-1} multiplied either by forget gate and the freshly calculated hidden state g multiplied by that of the input gate. Both of these factors are multiplied together.

It's possible that LSTMs should be thought of as a specific case of RNNs in general. The standard recurrent neural network (RNN) has the following configuration: input input gate is fixed to all 1's, the forget gate was fixed to all 0's, and also the output gate was fixed to all 1's. There is simply an extra tanh(.) that produces output that is slightly compressed. This gating mechanism is just what makes it possible for LSTMs to model long-term interdependence in an explicit manner. The network is able to figure out how its memory ought to function once it has learned the settings for its gates. In a nutshell, LSTMs are built on RNNs and add memory cell c as well as forget gate f. The memory cell is used to improve the abstract (memory) over long sequences by utilising it.

4.5 Design of LSTM for STLF

The planning and modelling that went into developing the LSTM architecture for STLF is depicted in Figure 4.3. In addition, the roots mean square replica optimizer (RMSprop) is used in order to propagate the error, and LSTM is used in order to anticipate the data set that is presented. Both of these methods are combined into one. In the scenario in which epsilon equals one in ten to the power ten, the number of iterations is set to zero, degradation is set to nine, and momentum is set to zero. The number of iterations will determine the size of the step, while degradation will serve as a component that delays the history/future gradients. It is not essential to wait at the minimum since momentum is represented as a floating-point value. This

makes waiting at the minimum unnecessary. Epsilon is a very small integer that is added to numerators so that they do not sum up to zero. This is done in order to prevent numerators from being equal to zero. There are three primary contextual nodes and four hidden active layers that are active within a single layer. This makes the total number of active layers five. TensorFlow, a technology that is often employed for conducting supervised learning, is put to use in the process of incorporating the LSTM model. On the basis of the tensor flow, decisions are made for both the biases coefficients and the activation functions.

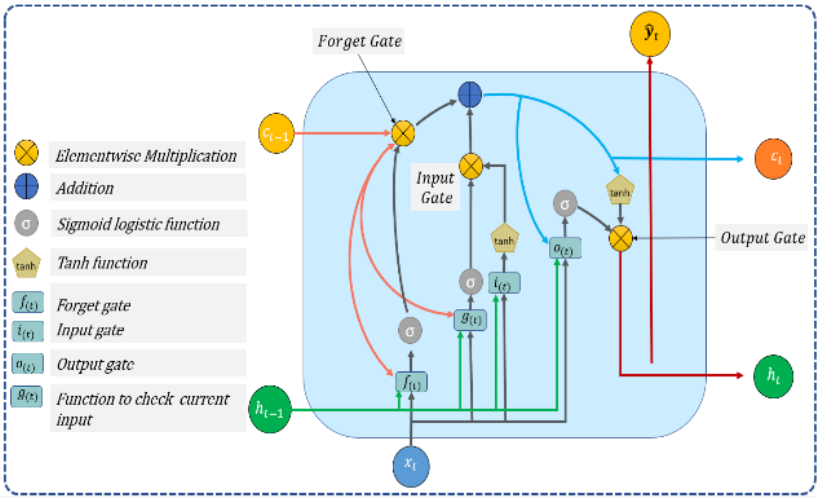

Fig. 4. 3 Design Modelling of LSTM

From this, the following error matrices were generated. Root Mean Squared Error (RMSE) [34] is an odd key performance indicator (KPI), but it's also a very helpful one, as we'll see later. It is known as the square root of the average squared error. Root Mean Squared Error is the name of the formula for Mean Squared Error. It determines the standard deviation of the residuals. The expression used to denote this is

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2} \qquad (4.13)$$

The Mean Absolute Error (MAE) is a great KPI to evaluate forecast accuracy. As its name implies, it is the mean of the absolute error [35]. The Mean Absolute Error is a measure of the average absolute deviation between both the dataset's actual and expected values. It determines the average residuals for the dataset.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}| \qquad (4.14)$$

Where, $\hat{y}$ is the predicted value of y.

y_i is the mean value of y.

The mean absolute percentage error (MAPE) [34], also called the mean absolute percentage deviation, is a way to measure how accurate a system is (MAPD). It can be written as the average percentage of relative error for each time frame minus the real values, divided by the real values. It gives a percentage for this level of accuracy. Since the units of the variable are changed to percentage units, which make it easier to understand, the absolute mean percent error (MAPE) has become the most common way to predict error. If there aren't too many facts, it works best (and no zeros). It is often used as a loss function in linear regression and model evaluation. This is the formula for MAPE:

$$M = \frac{1}{N}\sum_{t=1}^{N}\left|\frac{A_t - F_t}{A_t}\right| \quad (4.15)$$

Where, N is the number of points that fit together.

A_t is actual value.

F_t is value for forecast.

4.6 Simulation of STLF Using LSTM

The findings are obtained when the STLF utilising LSTM approach that was proposed is implemented in the Python programming language in the Google Collaboratory. The data set was obtained from the Global Energy Forecasting Competition (GEFCom2017), and it contains a total of 10,48,575 data ranging from 03 January 2007 to 17 June 2009. The data set was gathered in full. The imported libraries are summarised in Table 2 for the purpose of carrying out the suggested methodology.

Table 4. 1 Python Libraries used in Proposed Work

Python Library used	Nomenclature
Pandas	pd
NumPy	np
TensorFlow	tf
TensorFlow	keras
Matplotlib.pyplot	plt

The data set is pre-processed and arranged to correct format before implementation of STLF as shown in Figure 4.4. The data set consists of load demand, date, year as well as month,

hours, day of week, day of year and the particularity of the date to identify the demand on holidays and working days.

Table 4. 2 Dataset Parameters

Unnamed: 0	zone	demand	drybulb	dewpnt	date	year	month	hour	day_of_week	day_of_year	weekend	holiday_name	holiday	trend	ts
NaN	1	CT	3386.0	25.0	19.0	3/1/2003	2003	Mar	1	Sat	60	True	NaN	False	0.0
3/1/2003 1:00	2	CT	3258.0	23.0	18.0	3/1/2003	2003	Mar	2	Sat	60	True	NaN	False	1.0
3/1/2003 2:00	3	CT	3189.0	22.0	18.0	3/1/2003	2003	Mar	3	Sat	60	True	NaN	False	2.0
3/1/2003 3:00	4	CT	3157.0	22.0	19.0	3/1/2003	2003	Mar	4	Sat	60	True	NaN	False	3.0
3/1/2003 4:00	5	CT	3166.0	23.0	19.0	3/1/2003	2003	Mar	5	Sat	60	True	NaN	False	4.0
3/1/2003 5:00	6	CT	3255.0	23.0	20.0	3/1/2003	2003	Mar	6	Sat	60	True	NaN	False	5.0
3/1/2003 6:00	7	CT	3430.0	24.0	20.0	3/1/2003	2003	Mar	7	Sat	60	True	NaN	False	6.0
3/1/2003 7:00	8	CT	3684.0	24.0	20.0	3/1/2003	2003	Mar	8	Sat	60	True	NaN	False	7.0
3/1/2003 8:00	9	CT	3977.0	25.0	21.0	3/1/2003	2003	Mar	9	Sat	60	True	NaN	False	8.0
3/1/2003 9:00	10	CT	4129.0	27.0	22.0	3/1/2003	2003	Mar	10	Sat	60	True	NaN	False	9.0

The following results were obtained by running the simulation. The discussions carried out based on the results obtained is carried out in preceding session.

i. Actual load demand curve for first 500 hours
ii. Load demand curve for first 72 hours
iii. Load demand curve from 73 hour
iv. Loss function plot using LSTM network
v. Model accuracy plot (Loss Vs Epoch)
vi. Actual and predicted load demand Vs Epoch plot

4.7 Results and Discussions

The results obtained and the observation of the result and the efficacy of the proposed model of STLF using LSTM is carried out in this session.

Figure 4.4 illustrates the real-world load demand curve for the first 500 hours of operation. In the Y direction is where we record the load demand, and the X direction is where we record the time in hours. The graph paints a clear image of the enormous demand for load that existed during the course of the time period that was taken into account, with an average of approximately 5000 MW.

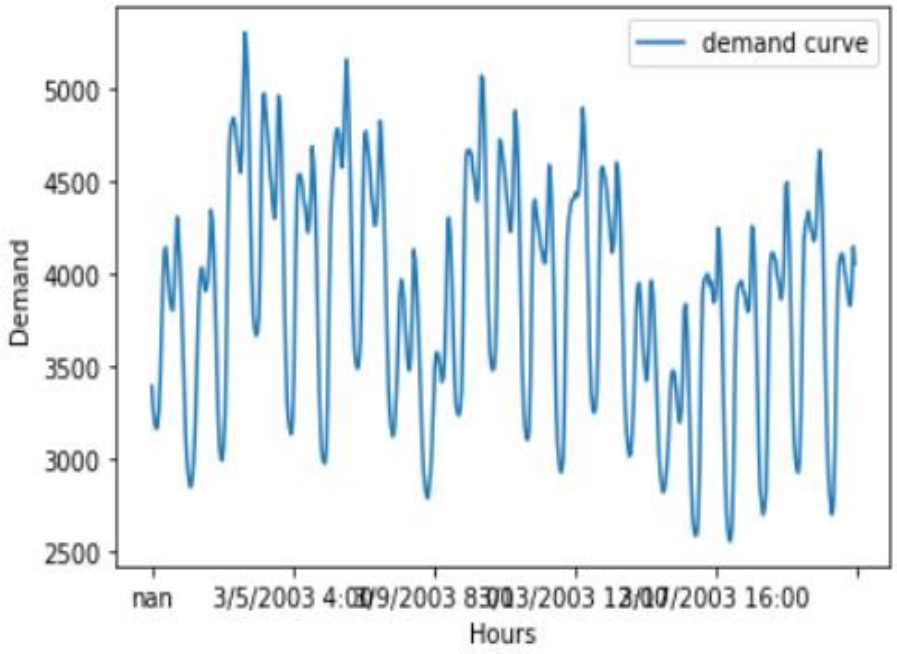

Fig. 4. 4 Actual Load Demand Curve for First 500 Hours

Figure 4.5 shows a plot of the load demand curve for the first seventy-two hours of operation. It paints an accurate picture of the current time as well as the load that is being demanded.

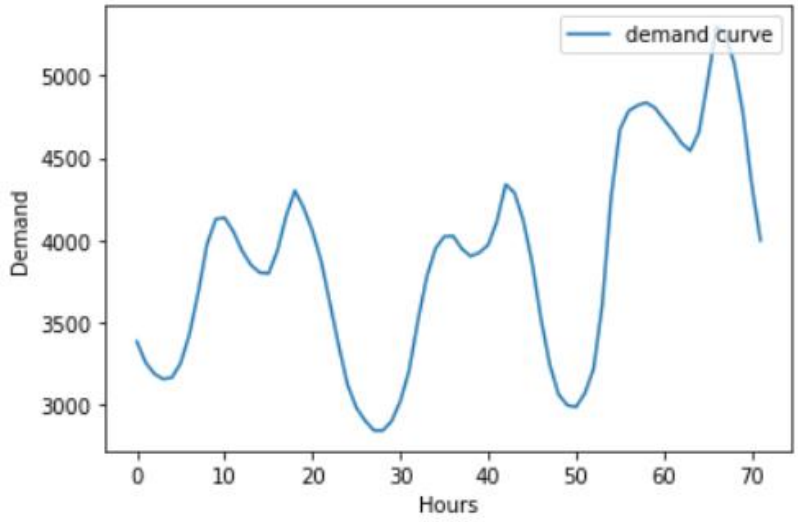

Fig. 4. 5 Load Demand Curve for First 72 Hours

Figure 4.6 depicts the load demand beginning at 73 hours and going forward, as calculated by LSTM using the training labels. Even in this case, the number makes it quite evident that the demand is significant over the entirety of the time period that is measured in hours.

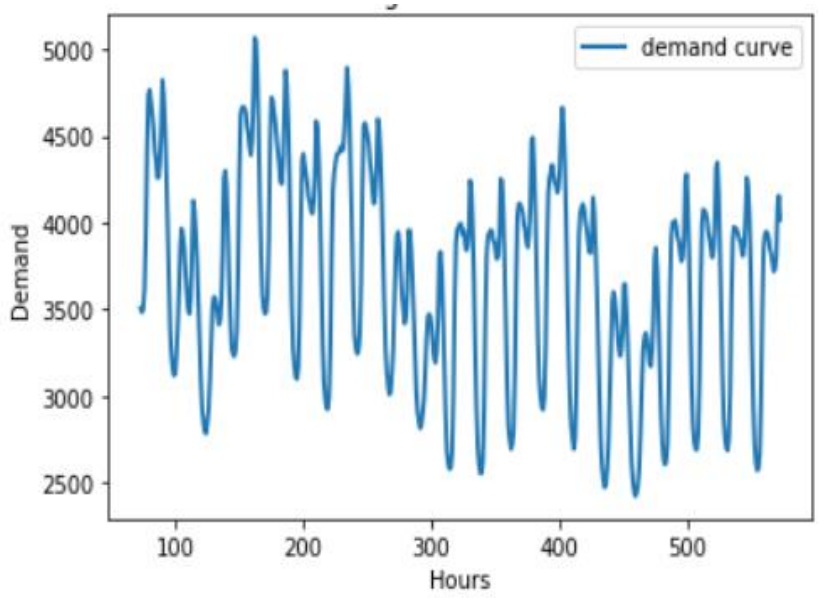

Fig. 4. 6 Load Demand Curve From 73rd hour

In order to generate the loss function plot for the suggested STLF using LSTM, ten epochs need to be taken, as is demonstrated in Figure 4.7. The plot of the loss function may be obtained by running the simulation and specifying the loss function in the y direction and the epoch in the x direction. Figure 4.8 illustrates this point further.

```
Epoch 1/10
248/248 [==============================] - 63s 240ms/step - loss: 3818466.5000 - acc: 0.0000e+00
Epoch 2/10
248/248 [==============================] - 60s 241ms/step - loss: 652538.0625 - acc: 0.0000e+00
Epoch 3/10
248/248 [==============================] - 61s 245ms/step - loss: 656552.1875 - acc: 0.0000e+00
Epoch 4/10
248/248 [==============================] - 61s 245ms/step - loss: 658533.1875 - acc: 0.0000e+00
Epoch 5/10
248/248 [==============================] - 60s 243ms/step - loss: 659292.6875 - acc: 0.0000e+00
Epoch 6/10
248/248 [==============================] - 61s 247ms/step - loss: 660244.3750 - acc: 0.0000e+00
Epoch 7/10
248/248 [==============================] - 61s 244ms/step - loss: 660523.6875 - acc: 0.0000e+00
Epoch 8/10
248/248 [==============================] - 61s 246ms/step - loss: 659560.7500 - acc: 0.0000e+00
Epoch 9/10
248/248 [==============================] - 66s 268ms/step - loss: 659666.0000 - acc: 0.0000e+00
Epoch 10/10
248/248 [==============================] - 65s 260ms/step - loss: 660170.6250 - acc: 0.0000e+00
```

Fig. 4. 7 Performance of Epochs

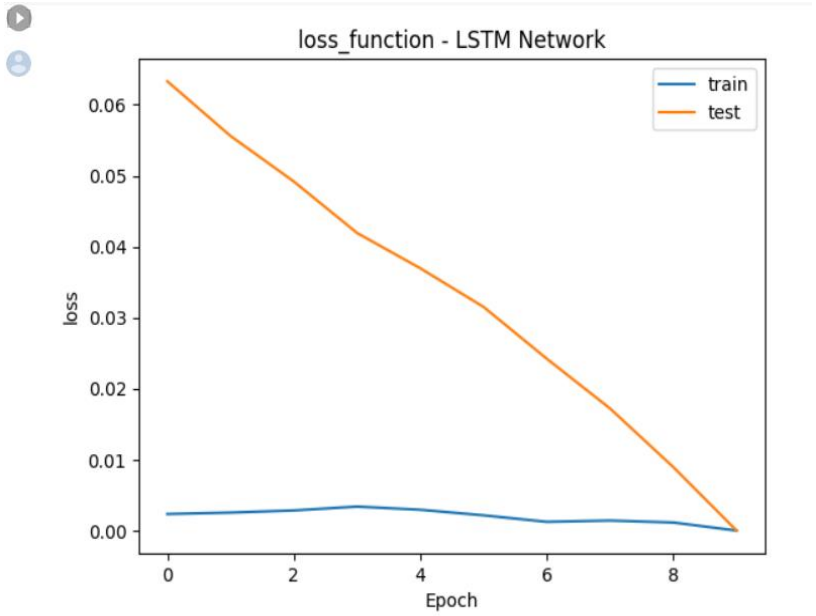

Fig. 4. 8 Loss Function Plot Using LSTM Network

Figure 4.9 depicts the proposed model accuracy using the LSTM method, with the loss function on the Y axis and the epochs on the X axis. The model accuracy in percentage is 85.17%, demonstrating the effectiveness of the proposed LSTM model in short-term load forecasting.

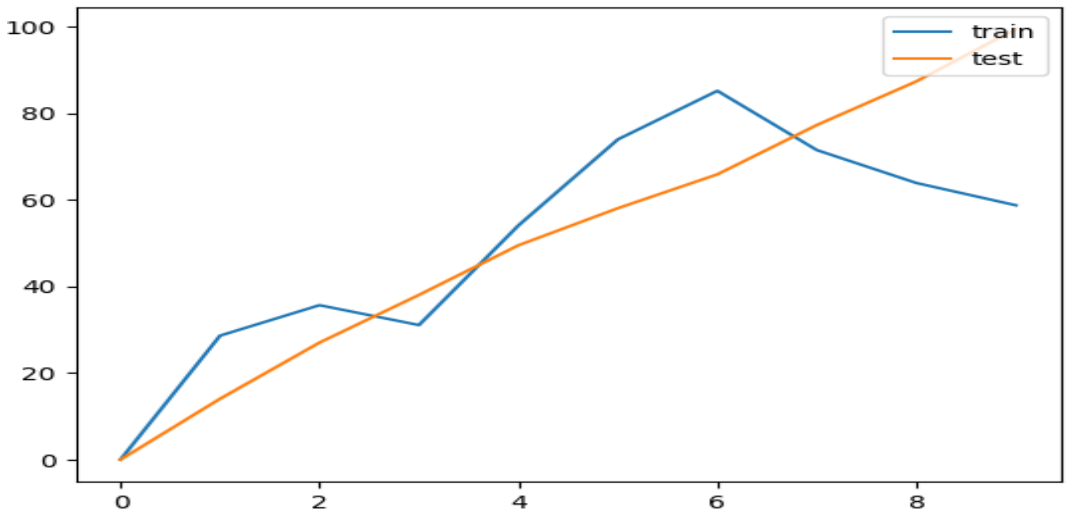

Fig. 4. 9 Model Accuracy Plot (Loss Vs Epoch)

In Figure 4.10, the accuracy of the suggested model that makes use of the LSTM approach is plotted by placing the loss function on the Y axis and the epochs on the X axis. The accuracy of the model, expressed as a percentage, was found to be 85.17%, which demonstrates the usefulness of the LSTM model that was proposed for short-term load forecasting.

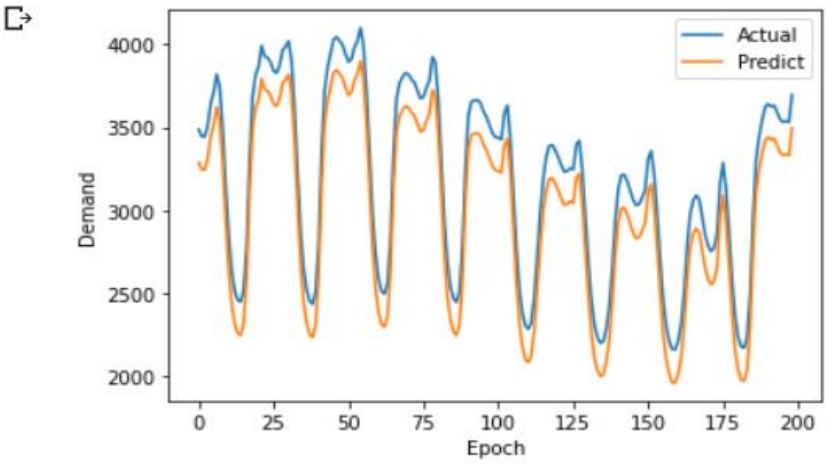

Fig. 4. 10 Actual and Predicted Load Demand Vs Epoch Plot

4.8 Accuracy and Mean Absolute Percentage Error (MAPE)

The accuracy of the proposed LSTM model as well as the key performance indicator named Mean Absolute Percentage Error (MAPE) is obtained and shown in Table 4.3.

Table 4. 3 Key Performance Indicator Values using Proposed LSTM

Performance Indicator	Observed Value using Proposed LSTM
Model Accuracy	85.17%
Mean Absolute Percentage Error (MAPE)	5.6%

The results you have provided indicate that the LSTM algorithm has performed reasonably well for Short-Term Load Forecasting (STLF). An accuracy of 85.17% indicates that the model is able to accurately predict the load for a majority of the time, while a Mean Absolute Percentage Error (MAPE) of 5.6% suggests that the model's predictions are, on average, only about 6.45% off from the actual values.

4.9 Summary

This chapter explained well about the classical model, LSTM method, simulation setup using LSTM, application to STLF and the results were analysed. The real load demand plot, the load demand plot for the first 72 hours, and the demand curves with a training demand beginning at hour 73 are shown and well discussed in this chapter. The LSTM loss function plot and, finally, we forecast the model's accuracy is also discussed. Existing and forecasted demand are both plotted and discussed in this chapter. The model accuracy as well as the Key performance indicator (KPI) MAPE were also observed and discussed to assess the feasibility of the proposed model.

Chapter 5

Short-Term Load Forecasting Using Particle Swarm Optimization (PSO) Based Gated Recurrent Unit (GRU)

5.1 Preamble

The PSO based GRU is discussed in this chapter along with the mathematical modelling of PSO based GRU. Later on, the STLF is carried out using the proposed PSO based GRU and the results were plotted and the accuracy curves as well as the Key performance indicator (KPI) named MAPE were also observed and discussed to assess the efficacy of the proposed PSO based GRU in STLF.

5.2 PSO Based GRU

Particle Swarm Optimization (PSO) is a popular optimization technique inspired by the social behaviour of bird flocking or fish schooling [36]. It has been applied to various fields, such as engineering, economics, and machine learning. One of the recent applications of PSO is in improving the performance of Gated Recurrent Unit (GRU) networks, which are widely used for sequence modelling tasks. GRU is a type of Recurrent Neural Network (RNN) that can model long-term dependencies in sequential data [37]. By optimizing the parameters of GRU using PSO, it can be potentially improving the accuracy of GRU for tasks such as load forecasting, speech recognition, machine translation, sentiment analysis etc. The PSO algorithm mimics the social behaviour of birds flocking. In the PSO algorithm, a swarm of particles moves in a high-dimensional search space to find the optimal solution. Each particle represents a potential solution to the optimization problem, and its position in the search space corresponds to the values of the parameters to be optimized. The velocity of each particle is updated at each iteration based on its current position, its best position so far, and the best position of the entire swarm. The new position of each particle is then determined by adding its updated velocity to its current position.

5.3 Mathematical Modelling of PSO Based GRU

To apply PSO to optimize the parameters of a GRU network, the first step is to define the fitness function which is to be optimized. The fitness function measures how well the GRU

network performs on a given task, such as predicting the next word in a sentence or classifying a sequence of images. The fitness function is usually defined as the loss function of the GRU network, which measures the difference between the predicted output and the true output. The parameters of the GRU network that want to optimize using PSO include the weights and biases of the various layers in the network. These parameters can be represented as a high-dimensional vector that serves as the position of each particle in the search space. The velocity of each particle corresponds to the rate at which its position changes in the search space. The best position of each particle and the best position of the entire swarm are updated based on the fitness function, which measures the performance of the GRU network.

GRU is a simplified version of LSTM that achieves the same results as its more complex counterpart. Figure 5.1 illustrates the typical organisational framework of GRU cells. A standard GRU cell is made up of two gates, which are referred to as the update gate (r_t) and the reset gate (h_t). GRU was introduced by Cho [37] in 2014. The research presents the update gate parameter as a means of controlling the process by which information about the prior state is replaced with information about the current state. The value of the update gate is decreased proportionally to the amount of state information that was available in the moment before. The reset gate determines how much data from the previous state is stored in the currently active candidate set h_t.

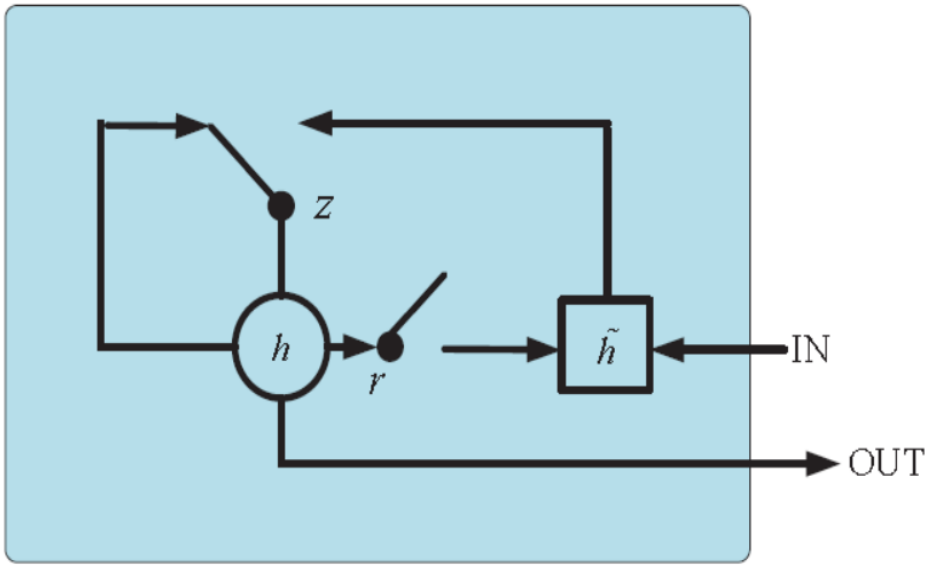

Fig. 5. 1 Structure of GRU Cell

The forward propagation formula of GRU is as follows.

$$S_t = \sigma(M_r[b_{t-1}, x_t]) \quad (5.1)$$

$$Z_t = \sigma(M_z[b_{t-1}, x_t]) \quad (5.2)$$

$$\widetilde{b_t} = tanh(M_{\tilde{h}}[s_t \otimes b_{t-1}, x_t]) \quad (5.3)$$

$$b_t = (1 - Z_t) \otimes b_{t-1} + Z_t \otimes b_t \quad (5.4)$$

$$y_t = \sigma(W_0 h_t) \quad (5.5)$$

Where, b_{t-1}, x_t – connection of two vectors and

$[s_t \otimes b_{t-1}, x_t]$- multiplication of vector elements.

M_r, M_z, $M_{\tilde{h}}$ and W_0 are the weight parameters to be learned for the training of GRU.

These parameters are given by,

$$M_r = M_{rx} + M_{rh} \quad (5.6)$$

$$M_z = M_{zx} + M_{zh} \quad (5.7)$$

$$M_{\tilde{h}} = M_{\tilde{h}x} + M_{\widetilde{hh}} \quad (5.8)$$

The import of the output layer is given by,

$$y_t^2 = W_0 b \quad (5.9)$$

The output is given by,

$$y_t^0 = \sigma(y_t^2) \quad (5.10)$$

The loss function of at a certain moment is set as,

$$E_t = \frac{1}{2}(y_t - y_t^0)^2 \quad (5.11)$$

The loss value is set as,

$$E = \sum_{t=1} E_t \quad (5.12)$$

The GRU gradient descent training will constantly update the parameters until convergence.

Let us pretend that the prospective settle of the optimisation matter is a particle. While it is looking for the best place to settle, this particle is continually moving across space and adjusting itself based on the information it gathers from its surroundings and its own experiences. PSO begins by iterating over a collection of stochastic solutions, after which it locates the optimal solution by tracking the optimal particles in the space that is now being processed by the algorithm. In the search space with multiple dimensions, there is a group composed of m individual particles. Iteration t yields the following results for the velocity and position of the i particle:$X_{i,t}$ and $v_{i,t}$ respectively. The particle constantly adjusts both its position and its speed in order to ensure that double optimal solutions are used. The ideal solution that all researchers are looking for right now is known as the global optimal solution g_{best}. The first is the optimal solution that the particle itself seeks, which is the individual extremum p_{best}. The second is the optimal solution that the particle itself seeks. In most cases, the following formula is used to calculate the latest position of the particle in the study:

$$v_{i,t+1} = wv_{i,t} + C_1 rand\left(p_{best_i} - S_{i,t}\right) + C_2 rand\left(g_{best_t} - S_{i,t}\right) \quad (5.13)$$

$$S_{t,t+1} = S_{i,t} + \lambda v_{i,t+1} \quad (5.14)$$

Where, $rand$ – random number between [0-1].

C_1 and C_2 – learning factors.

λ – velocity coefficient and value is 1.

The PSO algorithm can be summarized as follows:

1. Initialize the swarm of particles with random positions and velocities.
2. Evaluate the fitness of each particle using the fitness function.
3. Update the best position of each particle and the best position of the entire swarm based on the fitness values.
4. Update the velocity of each particle based on its current position, its best position so far, and the best position of the entire swarm.
5. Update the position of each particle based on its velocity.
6. Repeat steps 2-5 until a stopping criterion is met (e.g., maximum number of iterations or convergence of the fitness values).

By applying the PSO algorithm to optimize the parameters of a GRU network, we can potentially improve the accuracy of the network for various sequence modelling tasks. However, it is important to note that the performance of the PSO-based GRU network depends on several factors, such as the size of the swarm, the maximum number of iterations, and the choice of hyperparameters. Therefore, it is necessary to carefully tune these parameters to achieve the best performance.

5.4 Simulation of STLF Using PSO Based GRU

When the STLF employing PSO based GRU technique that was proposed is implemented in the Python programming language in the Google Collaboratory, the results that are shown here are obtained. The data set was obtained from the Global Energy Forecasting Competition (GEFCom2017), and it has a total of 10,48,575 data ranging from 03 January 2007 to 17 June 2009. The data set was obtained from the Global Energy Forecasting Competition (GEFCom2017). Complete information about the data set was gathered. Table 4 provides a rundown of the imported libraries so that the suggested methodology can be implemented successfully.

Table 5. 1 Python Libraries used in Proposed Work

Python Library used	Nomenclature
NumPy	np
Pandas	pd
Matplotlib.pyplot	Plt
Seaborn	sns
TensorFlow	tf
TensorFlow	keras

As can be seen in Table 5.2, the dataset goes through some preliminary processing and is then formatted correctly before the STLF algorithm is put into place. The load demand, the date, and the year are all included in the data set, as well as the month, hours, day of the week, day of the year, and the specificity of the date, which is used to determine the demand on holidays and working days. A representation of the GEFCOM data set for short-term load prediction can be seen in Table 5.2. The data set is organised so that each hour from 2003 onwards is separated by one hour, and it shows how the real demand correlates to each day's specific time. A time series is a succession of numerical data points that are ordered one after the other. Measurements of these points are typically taken at predetermined intervals such as

once every month, once every day, once every hour, etc. The data for this project are collected on an hourly basis, and the starting point for the measurements is 2003.

Table 5. 2 Dataset Parameters

	Unnamed: 0	ts	zone	demand	drybulb	dewpnt	date	year	month	hour	day_of_week	day_of_year	weekend	holiday_name	holiday	trend
0	1	NaN	CT	3386.0	25.0	19.0	3/1/2003	2003	Mar	1	Sat	60	True	NaN	False	0.0
1	2	3/1/2003 1:00	CT	3258.0	23.0	18.0	3/1/2003	2003	Mar	2	Sat	60	True	NaN	False	1.0
2	3	3/1/2003 2:00	CT	3189.0	22.0	18.0	3/1/2003	2003	Mar	3	Sat	60	True	NaN	False	2.0
3	4	3/1/2003 3:00	CT	3157.0	22.0	19.0	3/1/2003	2003	Mar	4	Sat	60	True	NaN	False	3.0
4	5	3/1/2003 4:00	CT	3166.0	23.0	19.0	3/1/2003	2003	Mar	5	Sat	60	True	NaN	False	4.0

5.5 Results and Discussions

The change of dry bulb values with respect to the year that is taken into consideration is displayed as a result of the simulation in Figure 5.2. This figure makes use of the dataset provided by GEFCOM, for which the years 2003-2018 were selected as the range of interest. The values of the dry bulb temperature have been recorded to range anywhere from -20 to 100.

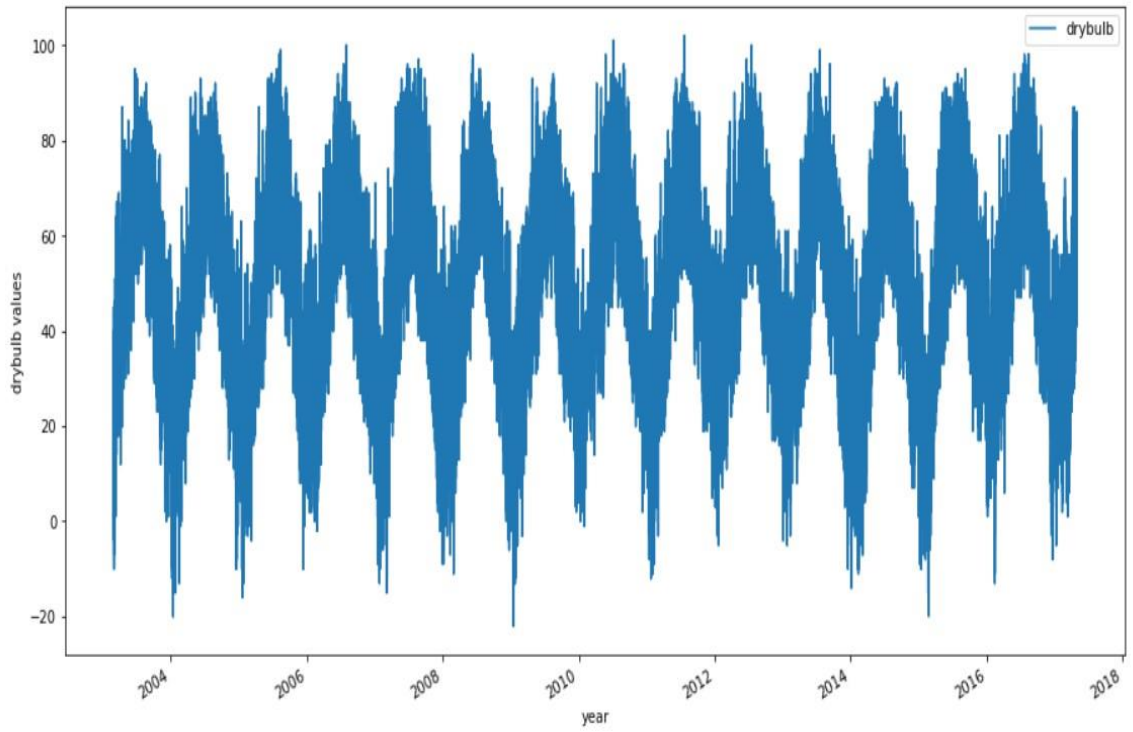

Fig. 5. 2 Dry Bulb Temperature Values

Similary the dew point values with respect to the year is taken into consideration and it is observed that the dew point values ranges from -30 to 80. The dew point values verses year is shown in Figure 5.3.

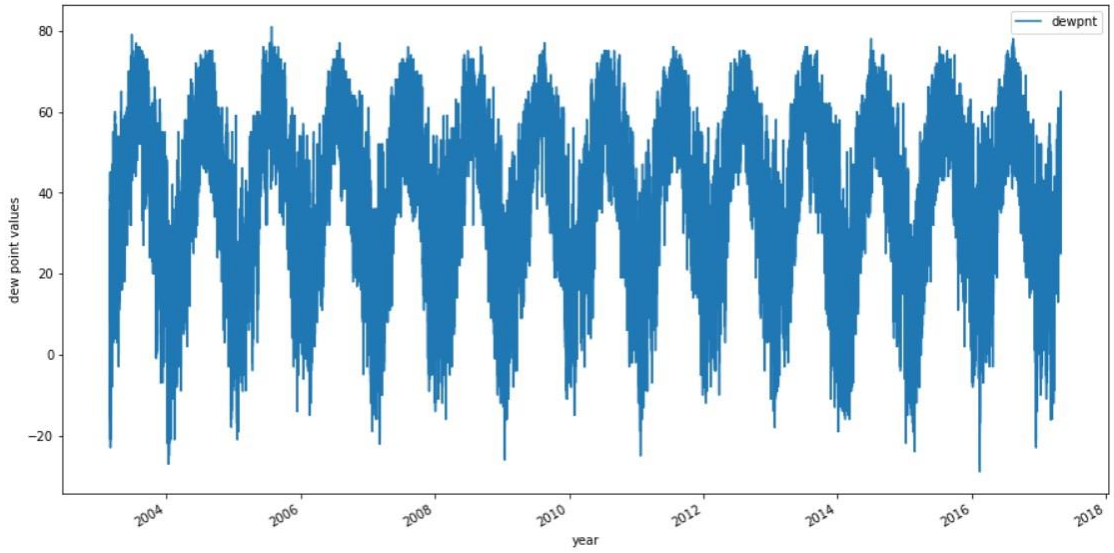

Fig. 5. 3 Dew Point Values

The corresponding heat map of the data is shown in Figure 5.4.

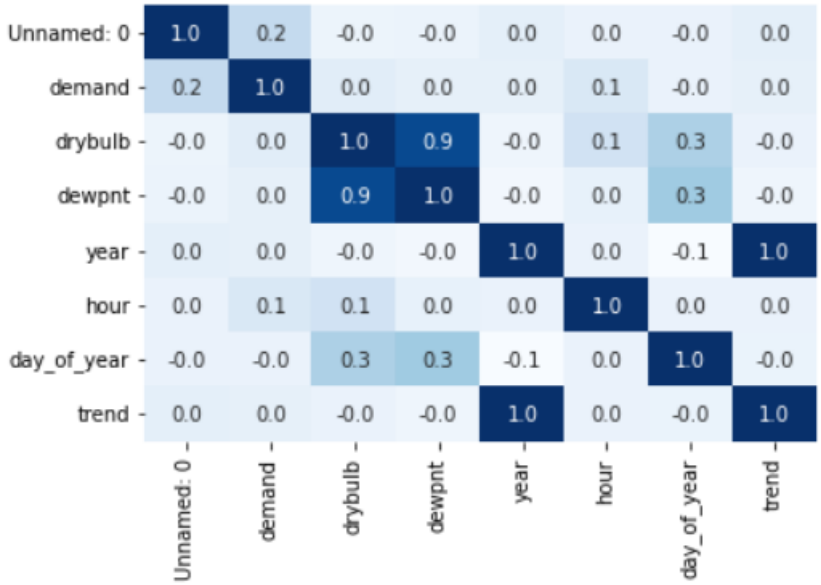

Fig. 5. 4 Heatmap of the Data

The correlation values between each data column and the other columns are depicted in Figure 5.4. Because this demonstrates the autocorrelation of the same data, each element of the diagonal should be set to 1.0. There is a correlation between all of the other factors. Due to

the fact that only a few variables in the cross correlation display higher values, it is impossible to do data redundancy.

The demand plot is displayed in Figure 5.5. It displays a range of the real demand numbers. The graphic makes it obvious that the range of demand values is between 2000 to 4000. It can be seen in the dataset as well.

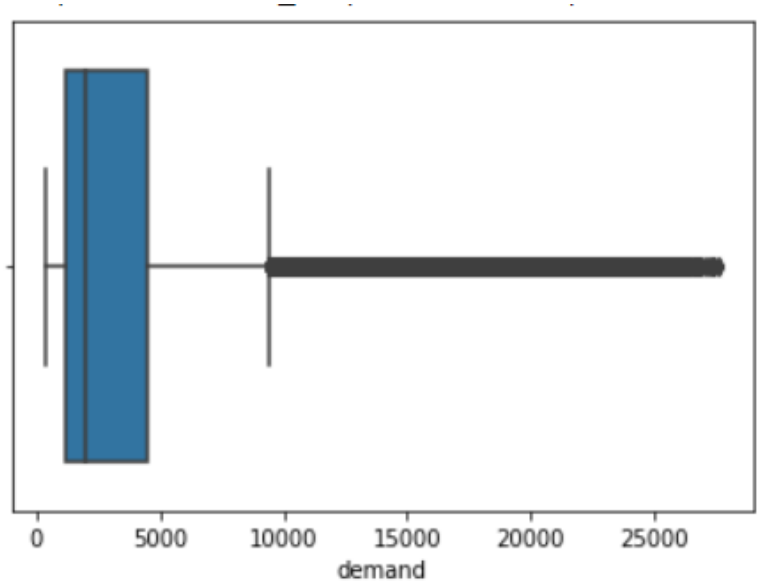

Fig. 5. 5 Demand Box Plot

The statitics of the database is shown in Table 5.3. The database is made up of various data in various magnitude ranges. The figure displays the mean, standard deviation, and maximum value for each column as well as other static characteristics.

Table 5. 3 Database Statics

	Unnamed: 0	demand	drybulb	dewpnt	year	hour	day_of_year	trend
count	1.048574e+06	1.048574e+06	1.048574e+06	1.048574e+06	1.048574e+06	1.048574e+06	1.048574e+06	1.048574e+06
mean	5.242885e+05	3.957456e+03	5.019708e+01	3.867814e+01	2.009542e+03	1.250243e+01	1.816918e+02	6.028363e+04
std	3.026974e+05	4.423006e+03	1.870094e+01	1.969047e+01	4.109257e+00	6.921117e+00	1.051945e+02	3.592336e+04
min	2.000000e+00	3.650000e+02	-2.200000e+01	-2.900000e+01	2.003000e+03	1.000000e+00	1.000000e+00	0.000000e+00
25%	2.621452e+05	1.214000e+03	3.600000e+01	2.400000e+01	2.006000e+03	7.000000e+00	9.000000e+01	2.913400e+04
50%	5.242885e+05	1.963000e+03	5.100000e+01	4.000000e+01	2.009000e+03	1.300000e+01	1.800000e+02	5.864900e+04
75%	7.864318e+05	4.498000e+03	6.500000e+01	5.500000e+01	2.013000e+03	1.900000e+01	2.730000e+02	9.142375e+04
max	1.048575e+06	2.762200e+04	1.020000e+02	8.100000e+01	2.017000e+03	2.400000e+01	3.650000e+02	1.241990e+05

Table 5.4 illustrates the various factors that can be found in the database, including demand, dry-bulb temperature, and others. For training and testing purposes, these factors are utilised.

Table 5. 4 Variables in Database

ts	Unnamed: 0	demand	drybulb	dewpnt	date	year	hour	day_of_year	trend
1/27/2016 13:00	734003	964.716	44.0	25.0	2016-01-27	2016	14	27	113173.0
1/27/2016 14:00	734004	953.690	46.0	23.0	2016-01-27	2016	15	27	113174.0
1/27/2016 15:00	734005	948.637	47.0	21.0	2016-01-27	2016	16	27	113175.0
1/27/2016 16:00	734006	985.243	45.0	20.0	2016-01-27	2016	17	27	113176.0
1/27/2016 17:00	734007	1071.651	43.0	20.0	2016-01-27	2016	18	27	113177.0

Table 5.5 illustrates the various factors that can be found in the database, including demand, dry-bulb temperature, and others. These parameters are normalised to fall anywhere between -1 and 1, inclusive.

Table 5. 5 Normalized Variables in Database

ts	Unnamed: 0	demand	drybulb	dewpnt	date	year	hour	day_of_year	trend
3/1/2003 1:00	-1.000000	-0.787724	-0.274194	-0.145455	-1.0	-1.0	-0.913043	-0.675824	-0.999984
3/1/2003 2:00	-0.999998	-0.792787	-0.290323	-0.145455	-1.0	-1.0	-0.826087	-0.675824	-0.999968
3/1/2003 3:00	-0.999996	-0.795135	-0.290323	-0.127273	-1.0	-1.0	-0.739130	-0.675824	-0.999952
3/1/2003 4:00	-0.999994	-0.794475	-0.274194	-0.127273	-1.0	-1.0	-0.652174	-0.675824	-0.999936
3/1/2003 5:00	-0.999992	-0.787944	-0.274194	-0.109091	-1.0	-1.0	-0.565217	-0.675824	-0.999919

Figure 5.6 illustrates the progress that has been made in training. The following results of the training are displayed: loss, accuracy, and validation accuracy.

```
Epoch 1/10
4096/4096 [==============================] - 44s 10ms/step - loss: 0.0012 - accuracy: 0.8956 - val_loss: 9.1378e-05 - val_accuracy: 0.9827 - lr: 0.0010
Epoch 2/10
4096/4096 [==============================] - 42s 10ms/step - loss: 2.6902e-04 - accuracy: 0.9227 - val_loss: 7.3295e-05 - val_accuracy: 0.9887 - lr: 9.0000e-04
Epoch 3/10
4096/4096 [==============================] - 40s 10ms/step - loss: 2.4762e-04 - accuracy: 0.9265 - val_loss: 7.2007e-05 - val_accuracy: 0.9869 - lr: 8.1000e-04
Epoch 4/10
4096/4096 [==============================] - 40s 10ms/step - loss: 2.3464e-04 - accuracy: 0.9284 - val_loss: 6.3369e-05 - val_accuracy: 0.9938 - lr: 7.2900e-04
Epoch 5/10
4096/4096 [==============================] - 39s 10ms/step - loss: 2.2898e-04 - accuracy: 0.9292 - val_loss: 6.4789e-05 - val_accuracy: 0.9919 - lr: 6.5610e-04
Epoch 6/10
4096/4096 [==============================] - 38s 9ms/step - loss: 2.2418e-04 - accuracy: 0.9301 - val_loss: 6.0200e-05 - val_accuracy: 0.9928 - lr: 5.9049e-04
Epoch 7/10
4096/4096 [==============================] - 38s 9ms/step - loss: 2.2072e-04 - accuracy: 0.9308 - val_loss: 6.1275e-05 - val_accuracy: 0.9924 - lr: 5.3144e-04
Epoch 8/10
4096/4096 [==============================] - 39s 10ms/step - loss: 2.1794e-04 - accuracy: 0.9314 - val_loss: 5.9428e-05 - val_accuracy: 0.9942 - lr: 4.7830e-04
Epoch 9/10
4096/4096 [==============================] - 37s 9ms/step - loss: 2.1568e-04 - accuracy: 0.9320 - val_loss: 5.8666e-05 - val_accuracy: 0.9922 - lr: 4.3047e-04
Epoch 10/10
4096/4096 [==============================] - 38s 9ms/step - loss: 2.1388e-04 - accuracy: 0.9327 - val_loss: 5.6994e-05 - val_accuracy: 0.9962 - lr: 3.8742e-04
```

Fig. 5. 6 Training Progress and Accuracy

The model's accuracy is depicted in Figure 5.7, along with the loss curve. This model makes use of a total of 105500 hours' worth of demand information as its data source. Seventy percent of it is put towards training, while the remaining thirty percent is used for validation. Figure 5.7 presents the model accuracy curve for a total of 10 epochs of data. During the training phase of the project, the model achieves a maximum accuracy of approximately 93.27%, while during the validation phase, it achieves 93.68%. The figure demonstrates that the forecast reveals a slight difference of approximately 100, although this difference is trivial for switching systems in terms of fulfilling the actual demand.

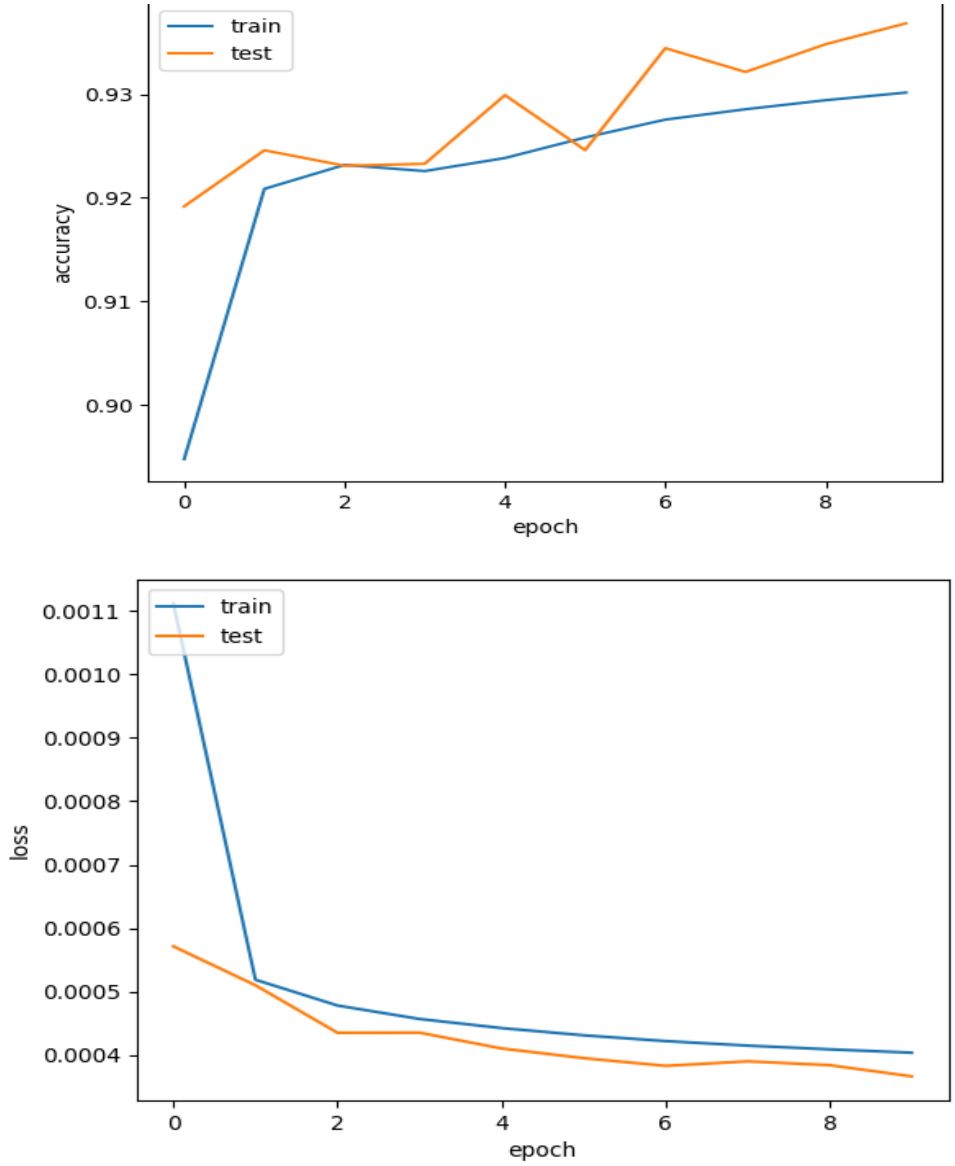

Fig. 5. 7 Model Accuracy and Loss Curve

Both the actual load and the expected load are displayed in Figure 5.8. The values of the load are displayed along the y-axis.

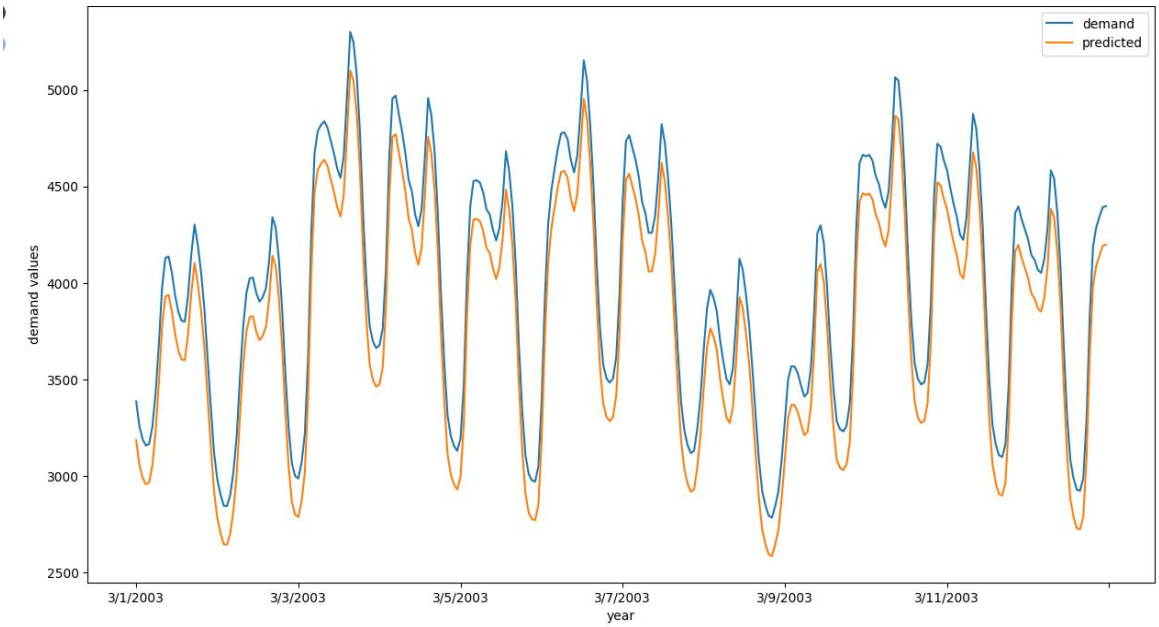

Fig. 5. 8 Expected and Actual Load Demands

5.6 Accuracy and Mean Absolute Percentage Error (MAPE)

The accuracy of the proposed PSO based GRU model as well as the key performance indicator named Mean Absolute Percentage Error (MAPE) is obtained and shown in Table 5.6.

Table 5. 6 Key Performance Indicator Values using Proposed PSO based GRU

Performance Indicators	**Observed Value using Proposed PSO based GRU**
Model Accuracy	93.68%
Mean Absolute Percentage Error (MAPE)	3.3%

The results you have provided indicate that the PSO-based GRU algorithm has performed very well for Short-Term Load Forecasting (STLF). An accuracy of 93.68% indicates that the model can accurately predict the load for a vast majority of the time, while a Mean Absolute Percentage Error (MAPE) of 3.3% suggests that the model's predictions are, on average, only about 3.3% off from the actual values.

The use of Particle Swarm Optimization (PSO) to optimize the GRU model's hyperparameters can contribute to the model's excellent performance. PSO is a popular optimization technique that mimics the social behaviour of a flock of birds to search for the optimal solution to a problem. The use of PSO to optimize the GRU model's hyperparameters allows for a more efficient search for the best combination of hyperparameters, resulting in better model performance.

5.7 Summary

This chapter explained well about the concept of PSO as well as the GRU in STLF. Also, the mathematical modelling of the PSO based GRU is also explained in this chapter. The simulation is carried out to obtain the results and a discussion is carried out based on the results obtained. The model accuracy as well as the Key performance indicator (KPI) MAPE were also observed and discussed to assess the feasibility of the proposed model.

Chapter 6

Short-Term Load Forecasting Using Multivariate LSTM

6.1 Preamble

The STLF using multivariate LSTM is described in this chapter. The modelling of multivariate LSTM is explained well in this chapter. Then the simulation is run in Google Collaboratory and the results were plotted and analysed in this chapter. Finally, the effectiveness of the suggested multivariate LSTM model in STLF is also evaluated using accuracy and Mean Absolute Percentage Error (MAPE).

6.2 STLF With Multivariate LSTM

Multivariate LSTM (Long Short-Term Memory) is a type of recurrent neural network (RNN) that can process sequential data with multiple input variables. It is a powerful tool for modelling time series data where each time step includes multiple features or variables. In traditional LSTM, a single sequence of data is input to the network, and each time step of the sequence is fed as an input to the network. In contrast, in multivariate LSTM, there are multiple sequences of data, where each sequence corresponds to a different variable. These variables may have different units, scales, and ranges. The multivariate LSTM network includes an input layer that accepts input from each sequence. The network then processes each input sequence through the LSTM layer, which consists of memory cells that allow the network to learn from past inputs and produce predictions for future inputs. The methodology of multivariate LSTM model is shown in Figure 6.1.

In comparison to univariate models, multivariate LSTM has the advantage of being able to more accurately capture the intricate connections that exist between the many input variables. This, in turn, contributes to the model's greater capacity for accurate prediction. It is also able to deal with missing data through imputed values, which it does by utilising the data that is accessible from other variables. In general, multivariate LSTM is a strong tool for modelling complicated time series data having multiple input factors. This is because it can learn from and generalise across many variables. The different univariate models are converted into their corresponding multivariate variations via multivariate LSTM (MLSTM) [38]. In order to improve the accuracy of the classification process, we added a keep squeezing-and-

excite block to the case of 1D sequence models in addition to augmented the fully convolutional blocks of the LSTM model.

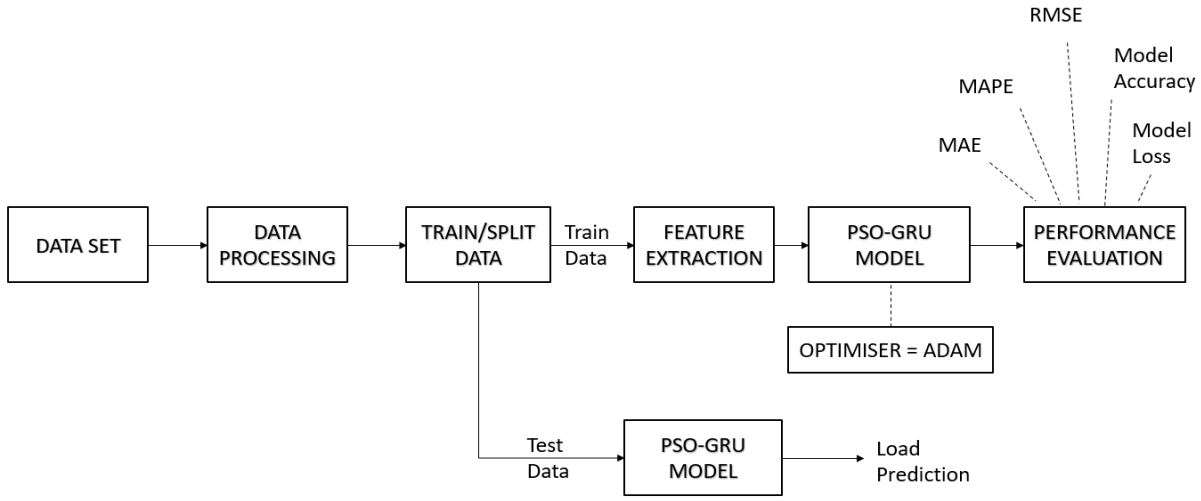

Fig. 6. 1 Methodology for Proposed Multivariate LSTM

Since the datasets now contain multivariate time series, we may define a time series dataset as a tensor of shape (N, Q, M), whereby N is the number of samples in the data set, Q is the maximum number of time steps across all variables, and M is the number of variables processed each time step. As a result, a dataset that consists of univariate time series is a subset of the description given above, in which M is reduced to a single value [39]. The LSTM model's input needs to be modified so that it accepts M inputs per time step rather than a single input every time step. This is the change that is necessary.

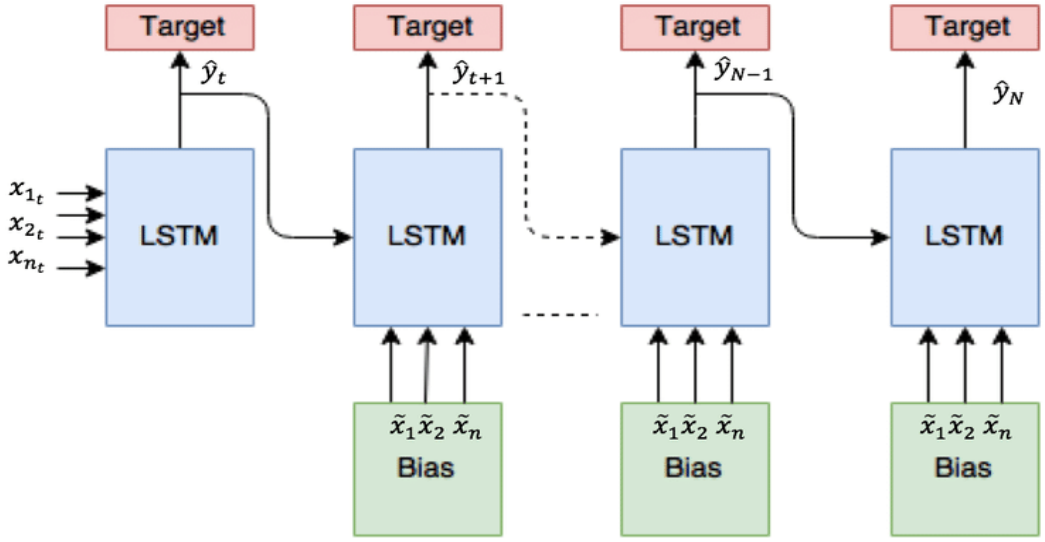

Fig. 6. 2 Proposed MLSTM Model Architecture

The fully convolutional block as well as the LSTM block that are illustrated in Figure 6.2 are both components of the MLSTM model that has been suggested. The fully convolutional block, which is reproduced from the original fully convolutional block, has three temporal convolutional blocks, which are utilised as a feature extractor. The convolutional blocks each have a convolutional layer that has a different number of filters (128, 256, as well as 128) as well as a different kernel size (8, 5, or 3) depending on the block. After each layer of convolutional processing comes the batch normalisation step, which has a momentum of 0.99 and an epsilon of 0.001. The ReLU activation function is applied after the batch normalisation layer has been processed. In addition, unlike LSTM, the suggested model concludes the first two convolutional blocks with a squeeze-and-excite block. This is another way in which the proposed model differs from LSTM. Our squeeze-and-excite block is computed according to the process outlined in Figure 6.3, which provides a summary of the procedure. We determined that a reduction ratio of 16 would work best for all squeeze and excitation blocks. A global average pooling layer comes after the last temporal convolutional block in the training process.

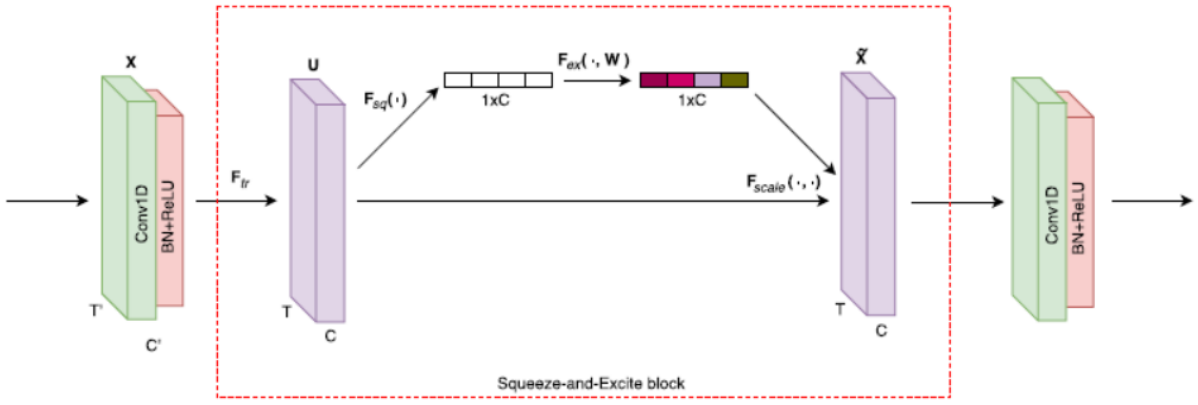

Fig. 6. 3 Calculation of the Time Block of Squeezing and Excitement

The LSTM block now has an additional component known as the squeeze-and-excite block, which is responsible for adaptively recalibrating the input feature maps. The overall model size only grows by between 3% and 10% as a result of the reduction ratio r being set to 16. This results in a reduction in the number of parameters that must be learned in order to create these self-attention maps. The following formula can be used to determine this:

$$P = \frac{2}{r}\sum_{s=1}^{S} R_s G_s^2 \qquad (6.1)$$

where P is the entire number of additional variables, r is the decrease in parameter ratio, S is the number of stages, where every stage refers to the group of blocks functioning on the feature maps of a common spatial dimensions, G_s is the amount of output feature maps for stage s, as well as R_s is the repeated block number for stage s. The architecture of multivariate LSTM is shown in Figure 6.4.

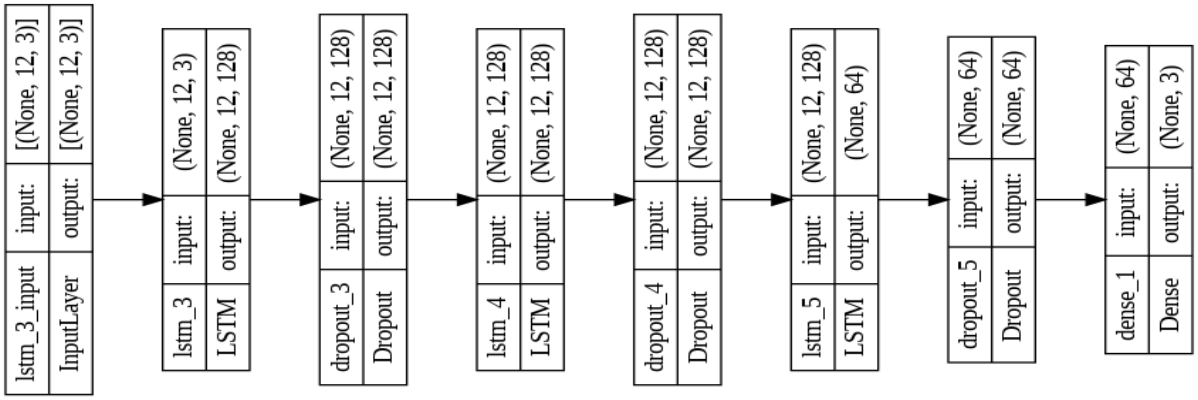

Fig. 6. 4 Architecture of Proposed Multivariate LSTM

In this equation, P represents the total number of additional parameters, r represents the reduction ratio, and S represents the number of stages. The total number of extra parameters is denoted by the letter P in this equation, while the reduction ratio is denoted by the letter r, and the number of stages is denoted by the letter S. We are in the fortunate position of being able to quickly determine the increased number of parameters across all models by utilising the formula P = 216 (1282 + 2562) which equals 10,240. This is possible because the FCN blocks are preserved in the same manner across all models. Because not all feature maps may have the same amount of impact on the layers that come after them, squeeze and excitation are essential for enhancing the performance of multivariate datasets. This is because not all feature maps may have the same level of influence. This adaptive recalibration of the feature maps can be regarded of as a sort of learned self-attention that is applied to the output feature maps of the layers that came before it. Specifically, the feature maps of the layers that came before it. This adaptive rescaling of the filter maps is of the greatest significance to the enhanced performance of the MLSTM model as opposed to the LSTM, as it incorporates taught self-attention to the inter-correlations between many variables at each time step, which was inadequate with the LSTM. This is one of the reasons why the MLSTM model is superior to the LSTM model in terms of performance. This improved performance can be since the MLSTM model integrates learned self-attention to the inter-correlations between numerous variables at each time step. Specifically, this attention is paid at each time step.

6.3 Simulation Results and Discussions of STLF Using Multivariate LSTM

The outcomes that are displayed here are attained when the STLF employing multivariate LSTM technique that was proposed is implemented in the Python programming language in the Google Collaboratory. The data collection, which spans from 03 January 2007 to 17 June 2009, was gathered from the Global Energy Forecasting Competition (GEFCom2017). The Global Energy Forecasting Competition (GEFCom2017) provided the data set. The data set's full information was obtained. To properly execute the suggested methodology, Table 6.1 gives a summary of the imported libraries.

Table 6. 1 Python Libraries Used in Proposed Work

Python Library used	Nomenclature
Pandas	pd
NumPy	np
Matplotlib.pyplot	Plt

The GEFCOM data set for short-term load prediction is shown in Table 6.2. The data set is organised in 1-hour segments starting in 2003, and the actual demand is displayed for each date and hour. A time series is a collection of numbers in ascending order. Frequently, measurements of these points are taken on a regular basis (every month, every day, every hour, etc.). Hourly data were measured starting in 2003 and were used in this investigation.

Table 6. 2 GEFCOM Data Set

Unnamed: 0	zone	demand	drybulb	dewpnt	date	year	month	hour	day_of_week	day_of_year	weekend	holiday_name	holiday	trend	ts
NaN	1	CT	3386.0	25.0	19.0	3/1/2003	2003	Mar	1	Sat	60	True	NaN	False	0.0
3/1/2003 1:00	2	CT	3258.0	23.0	18.0	3/1/2003	2003	Mar	2	Sat	60	True	NaN	False	1.0
3/1/2003 2:00	3	CT	3189.0	22.0	18.0	3/1/2003	2003	Mar	3	Sat	60	True	NaN	False	2.0
3/1/2003 3:00	4	CT	3157.0	22.0	19.0	3/1/2003	2003	Mar	4	Sat	60	True	NaN	False	3.0
3/1/2003 4:00	5	CT	3166.0	23.0	19.0	3/1/2003	2003	Mar	5	Sat	60	True	NaN	False	4.0
3/1/2003 5:00	6	CT	3255.0	23.0	20.0	3/1/2003	2003	Mar	6	Sat	60	True	NaN	False	5.0
3/1/2003 6:00	7	CT	3430.0	24.0	20.0	3/1/2003	2003	Mar	7	Sat	60	True	NaN	False	6.0
3/1/2003 7:00	8	CT	3684.0	24.0	20.0	3/1/2003	2003	Mar	8	Sat	60	True	NaN	False	7.0
3/1/2003 8:00	9	CT	3977.0	25.0	21.0	3/1/2003	2003	Mar	9	Sat	60	True	NaN	False	8.0
3/1/2003 9:00	10	CT	4129.0	27.0	22.0	3/1/2003	2003	Mar	10	Sat	60	True	NaN	False	9.0

The actual load for the first twenty-four hours is displayed in Figure 6.5, and the plot reveals that the data ranges from 2,500 to 4,200. The maximum points on the graph represent times of day with the highest demand, and the minimum points represent times of day with the lowest demand.

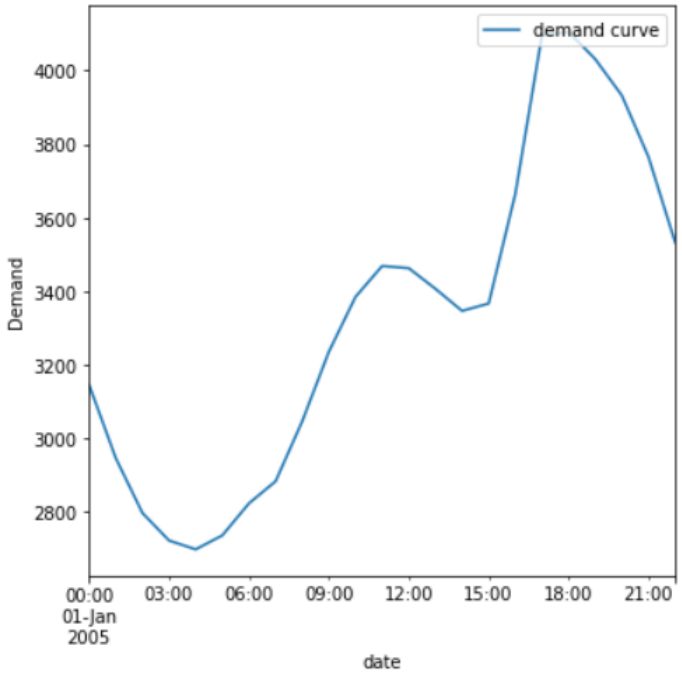

Fig. 6. 5 Actual Demand for First 500 Hour

Table 6.3 presents a visual representation of the multivariate LSTM model's hyperparameters, broken down per layer. There are around 2.48 lakh trainable parameters in this system. After that, the parameters are dispersed throughout the LSTM and the Dense layers.

Table 6. 3 LSTM Model Hyper Parameters

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 12, 128)	67584
dropout (Dropout)	(None, 12, 128)	0
lstm_1 (LSTM)	(None, 12, 128)	131584
dropout_1 (Dropout)	(None, 12, 128)	0
lstm_2 (LSTM)	(None, 64)	49408
dropout_2 (Dropout)	(None, 64)	0
dense (Dense)	(None, 3)	195

Total params: 248,771
Trainable params: 248,771
Non-trainable params: 0

This model makes use of a total of 10,55000 hours' worth of demand information as its data. The remaining 20% is used for validation, while the remaining 80% is put towards training. Figure 6.6 depicts the model accuracy curve over the course of 105 epochs of data.

During the validation process, the model achieves a maximum accuracy of approximately 97.67%.

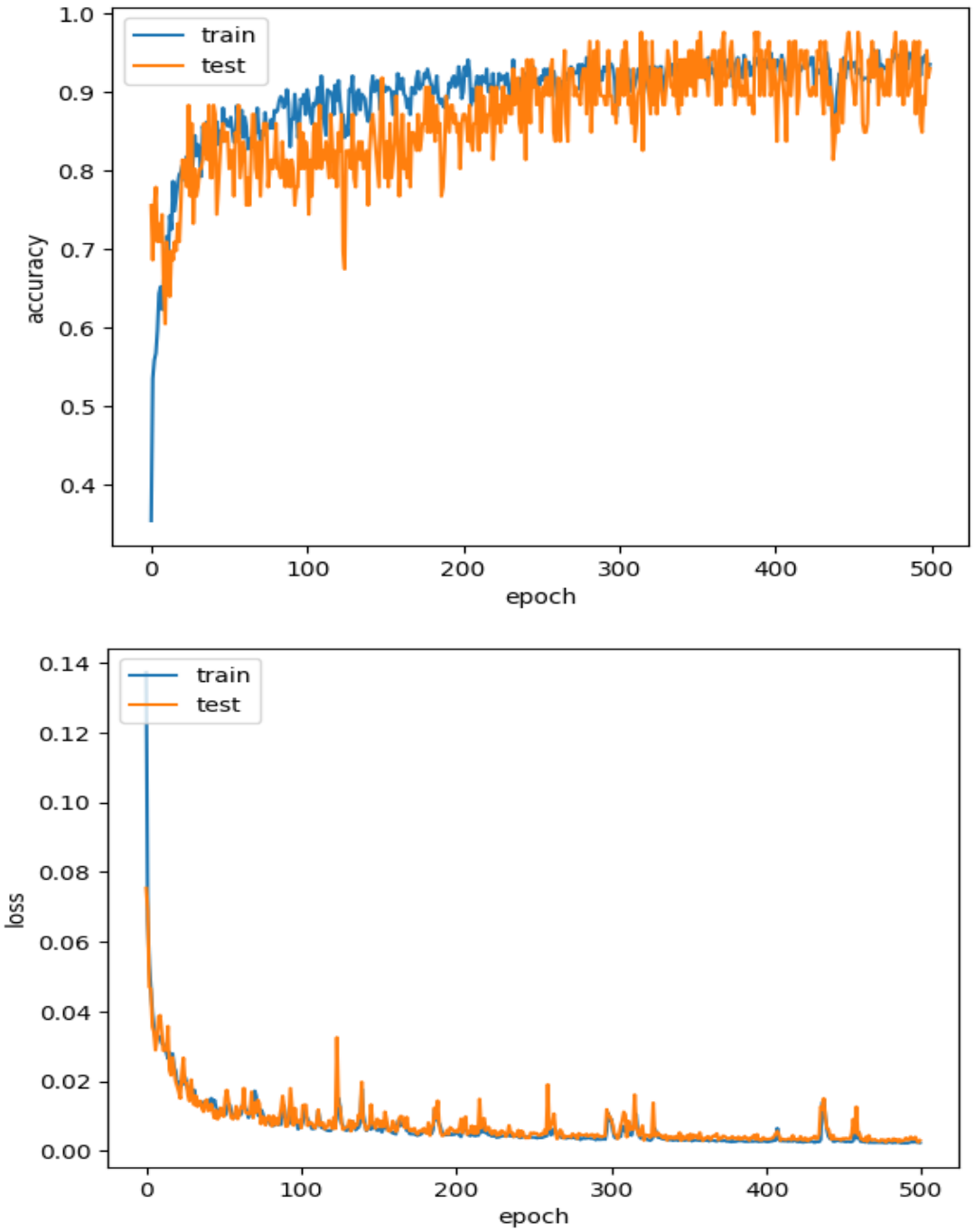

Fig. 6. 6 Model Accuracy and Loss Curve

The anticipated load for each hour that corresponds to the current demand is displayed in Figure 6.7. The figure demonstrates that the forecast reveals a slight difference of approximately 100, although this difference is trivial for switching systems in terms of fulfilling the actual demand.

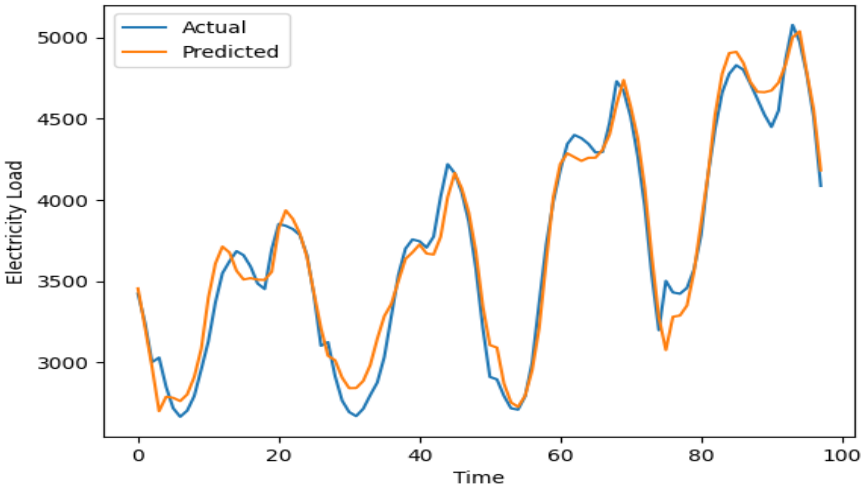

Fig. 6. 7 Actual Load versus Predicted Load

6.4 Accuracy and Mean Absolute Percentage Error (MAPE)

The accuracy of the proposed multivariate LSTM model as well as the key performance indicator named Mean Absolute Percentage Error (MAPE) is obtained and shown in Table 6.4.

Table 6. 4 Performance Indicator Values using Proposed Multivariate LSTM

Performance Indicators	**Observed Value using Proposed Multivariate LSTM**
Model Accuracy	97.67%
Mean Absolute Percentage Error (MAPE)	2.63%

Multivariate LSTM is a type of artificial neural network used for time-series forecasting. The model accuracy of 97.67% indicates that the model's predictions match the actual values 97.67% of the time. The Mean Absolute Percentage Error (MAPE) is a measure of the accuracy of the model's predictions. It represents the average percentage difference between the actual values and the predicted values. A MAPE of 2.63% suggests that the model's predictions are, on average, within 2.63% of the actual values.

Overall, these results indicate that the Multivariate LSTM model is performing well for STLF and can be considered a reliable forecasting tool for short-term load forecasting.

6.5 Summary

This chapter does an excellent job of explaining the Multivariate LSTM idea that is used in STLF. In addition to that, a mathematical modelling of the MLSTM is broken down and discussed in this chapter. The simulation is run in order to receive the results, and then a

conversation is held based on the results that were gained from the simulation. The model accuracy as well as the Key performance indicator (KPI) MAPE were also observed and discussed to assess the feasibility of the proposed model.

Chapter 7

Short-Term Load Forecasting Using 1-Dimensional Convolutional Neural Network (1-D CNN) Based LSTM

7.1 Preamble

The 1-D Convolutional Neural Network (CNN) based LSTM and its mathematical modelling are the topics that are covered in this chapter. Additionally, the use of a 1-D CNN is discussed in detail in this chapter. After that, the chapter continues to the simulation of STLF using 1-D CNN based LSTM, and a discussion is carried out on the results that were produced from that simulation. Finally, the effectiveness of the suggested 1-D CNN based LSTM model in STLF is also evaluated using accuracy and Mean Absolute Percentage Error (MAPE).

7.2 STLF Using One Dimensional Convolutional Neural Network (1-D CNN) BASED LSTM

Short-term load forecasting (STLF) is the process of predicting the future energy load demand over a short period of time. The accurate prediction of short-term load forecasting is crucial in electricity system operation and planning for efficient energy utilization, reducing costs, and preventing system instability. One popular technique for STLF is using deep learning models, specifically Long Short-Term Memory (LSTM) networks, which can learn the temporal patterns in the data and predict future values. One way to improve the performance of the LSTM model is to add a 1-dimensional convolutional neural network (1-D CNN) layer before the LSTM layer [40]. A 1-D CNN can extract features from the input sequence and reduce the dimensionality of the input data, which can make the LSTM network easier to train and improve the accuracy of the predictions. Here is a detailed explanation of how a 1-D CNN based LSTM model can be used for STLF.

7.2.1 Data Preparation

The first step is to prepare the data for the model. The input data should be a time series of historical load demand values. The data is divided into sequences of fixed length, such as 24 hours, and each sequence is used to predict the load demand for the next hour. The data is

normalized to have zero mean and unit variance to ensure that the model can learn the patterns in the data.

7.2.2 Model Architecture

The next step is to design the model architecture. The model consists of a 1-D CNN layer followed by an LSTM layer and a dense output layer. The 1-D CNN layer has a filter size of 3 and a stride of 1 to extract local features from the input sequence. The output of the CNN layer is fed into the LSTM layer, which has a hidden state size of 64 to learn the temporal patterns in the data. The output of the LSTM layer is fed into a dense output layer with a single output unit, which predicts the load demand for the next hour.

7.2.3 Model Training

The model is trained using historical load demand data and their corresponding target values. The training process involves minimizing the mean squared error (MSE) between the predicted values and the actual values. The model is trained using stochastic gradient descent (SGD) with a learning rate of 0.001.

7.2.4 Model Evaluation

The model's performance is evaluated on a separate validation dataset. The evaluation metrics used are Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE). The goal is to achieve low values for these metrics, indicating that the model can accurately predict future load demand values.

7.2.5 Model Deployment

Once the model is trained and evaluated, it can be deployed to make predictions on new data. The model takes a sequence of historical load demand values as input and predicts the load demand for the next hour.

7.3 One-Dimensional CNN Based LSTM and Mathematical Modelling

A 1D-CNN is a type of neural network that operates on one-dimensional data, such as time series. It consists of multiple layers of convolutional and pooling operations that extract features from the input data. The convolutional layers apply filters to the input data, while the pooling layers down sample the output of the convolutional layers [41]. Finally, the output of

the last layer is fed into a fully connected layer that produces the final prediction. An I D CNN structure is shown in Figure 7.1.

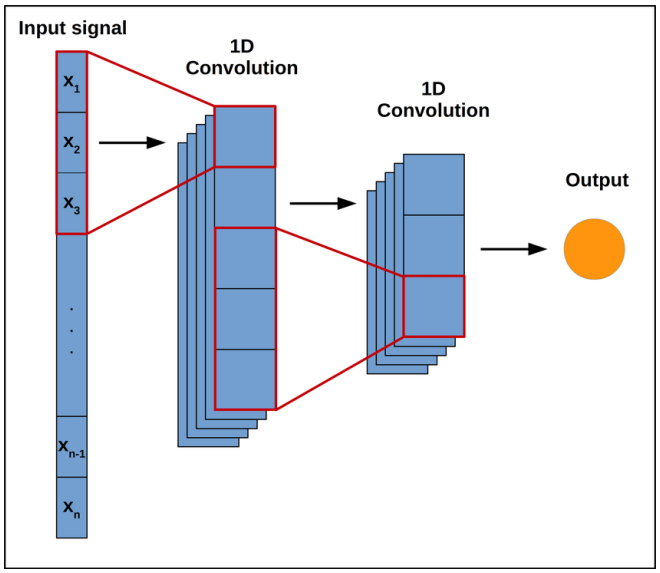

Fig. 7. 1 Structure of 1 D CNN

Mathematically, the output of a convolutional layer in a 1D-CNN can be calculated as follows:

$$y_i = f\left(\sum_{j=1}^{m} \omega_j x_{i+j-1} + b\right) \quad (7.1)$$

Where, x_{i+j-1} – input data; ω_j- filter or kernel applied to the input data; b – bias term; f- is the activation function; y_i- is the output of the convolutional later at position i. The size of the output depends on the size of the filter and the stride used during the convolution.

The pooling operation reduces the size of the output by aggregating neighbouring values. One common pooling operation is max pooling, which selects the maximum value from a set of values. The output of the pooling layer can be calculated as follows:

$$y_i = max_{j=1}^{m} x_{i+j-1} \quad (7.2)$$

After several convolutional and pooling layers, the output is fed into a fully connected layer, which can be represented mathematically as follows:

$$y = f(Wx + b) \quad (7.3)$$

Where, W- weight matrix; b – bias vector; f – activation function; x – is the input vector and y – output vector.

In short-term load forecasting, the input data typically consists of historical electricity consumption data, weather data, and other relevant factors. The 1D-CNN is trained on this data to predict the future electricity consumption for the next 24 hours. The model can be optimized using techniques such as stochastic gradient descent and backpropagation.

CNN has evolved in the field of computer vision with well- known structures such as AlexNet, VGGNet, Inception, ResNet, and DenseNet [42] and [43]. Recently, CNN demonstrated better performance than other machine learning algorithms such as MLP and SVM not only in the NLP domain [44] but also in electricity load forecasting. In this study, 1-D CNN is used to cope with the time series characteristics of electric load forecasting. An example of the operation of a 1-D CNN is presented in Figure 7.1.

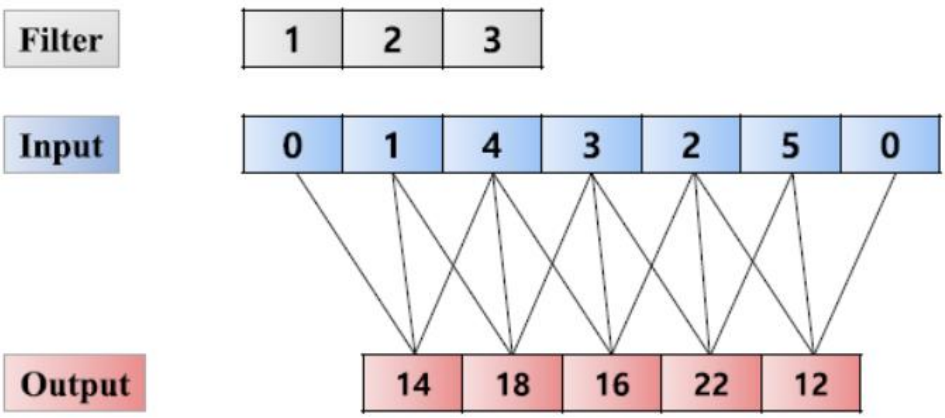

Fig. 7. 2 One-Dimensional Convolution (stride = 1, zero padding = 1)

$$O_i^l = ReLU(BN(O^{l-1} * F^l)_i) = ReLU\left(BN\left(\sum_m O_{i+m}^{l-1} * F_m^l\right)\right) \quad (7.4)$$

In (34), O_i^l refers to the ith in the lth layer. When the feature map of the l-1 layer is of width W, height 1 and channel C, it is denoted as $O^{l-1} \in \mathbb{R}^{W\ X\ 1\ X\ C}$ and the parameter $F^l \in \mathbb{R}^{k\ X\ 1\ X\ C}$ in the lth later is a weight that is learned by the gradient descent method and serves to identify a local pattern. The term * is a convolution product with inverted wrights, while BN and ReLU denote the batch normalization and rectified linear unit activation function. The basic structure of one dimensional convolutional neural network and its operation is shown in Figure 7.3. The LSTM structure is already explained in chapter 3.

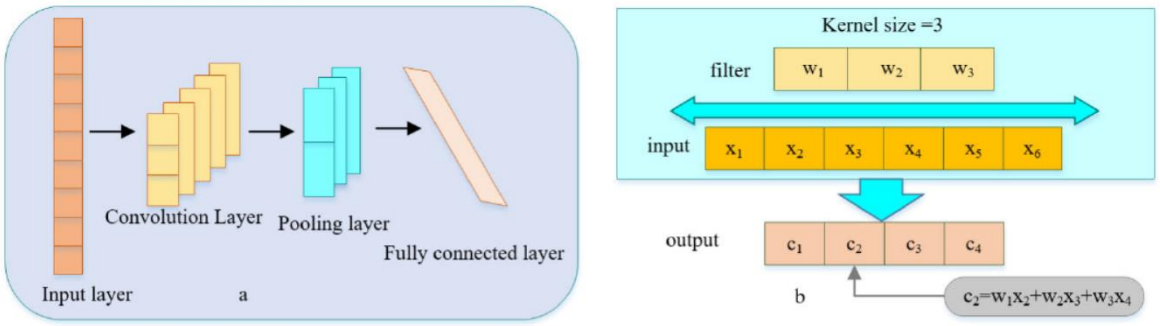

Fig. 7. 3 Basic Structure of One Dimensional Convolutional Neural Network and its Operation

The model overview of 1-D CNN based LSTM is shown in Figure 7.4.

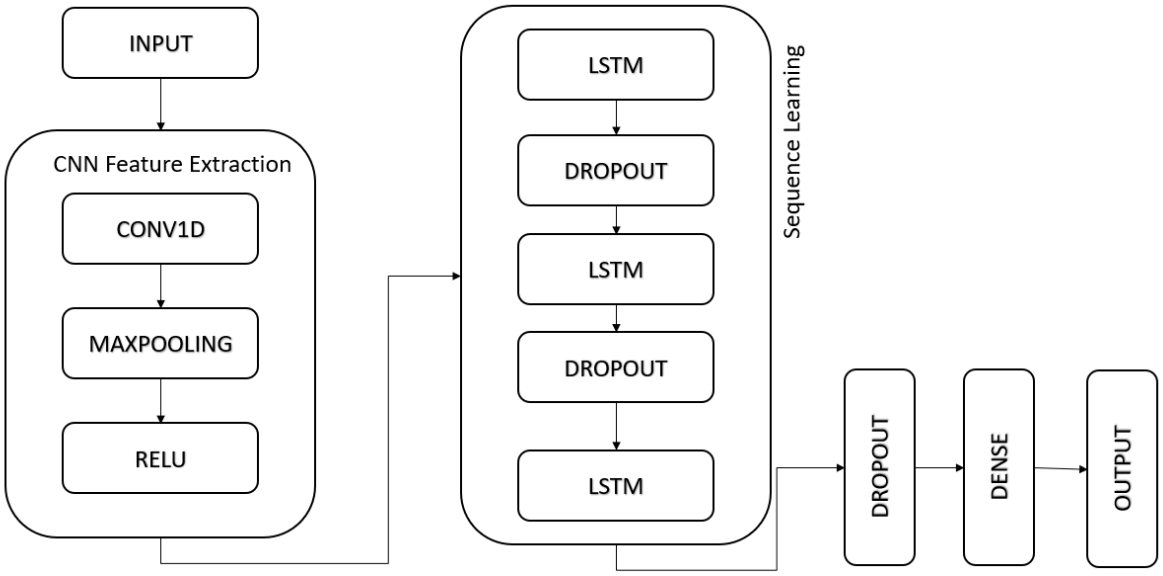

Fig. 7. 4 Model Overview of 1-D CNN Based LSTM

The one-dimensional convolutional neural network (1-D CNN) based Long Short-Term Memory (LSTM) model is a deep learning architecture that can be used for time series forecasting, including short-term load forecasting (STLF). The model consists of two main layers: a 1-D CNN layer and an LSTM layer.

1. 1-D CNN Layer: The 1-D CNN layer consists of several filters with a small kernel size, typically 2 or 3, which are applied to the input time series data. The filters slide along the time series data and perform convolution operations to extract local features, similar to how an image is filtered in a 2-D CNN. The output of the 1-D CNN layer is a 3-dimensional tensor, where the first dimension corresponds to the batch size, the second

dimension corresponds to the number of filters, and the third dimension corresponds to the length of the output sequence.

2. LSTM Layer: The output of the 1-D CNN layer is then fed into an LSTM layer, which is a type of recurrent neural network (RNN) that can learn long-term dependencies in time series data. The LSTM layer consists of multiple memory cells, each with three gates (input gate, forget gate, and output gate), that allow the model to selectively store and retrieve information from the past. The output of the LSTM layer is a 2-dimensional tensor, where the first dimension corresponds to the batch size and the second dimension corresponds to the hidden state size.
3. Output Layer: The output of the LSTM layer is fed into a dense output layer, which can consist of one or more fully connected layers. The output layer produces the final prediction for the time series data.

The input data for the model is usually divided into fixed-length windows or sequences, with each sequence containing a certain number of time steps. The model is trained using stochastic gradient descent (SGD) with backpropagation to minimize the difference between the predicted and actual values for the target time step. The model's performance is evaluated using various metrics such as Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE).

The architecture of 1-D CNN based LSTM is shown in Figure 7.5.

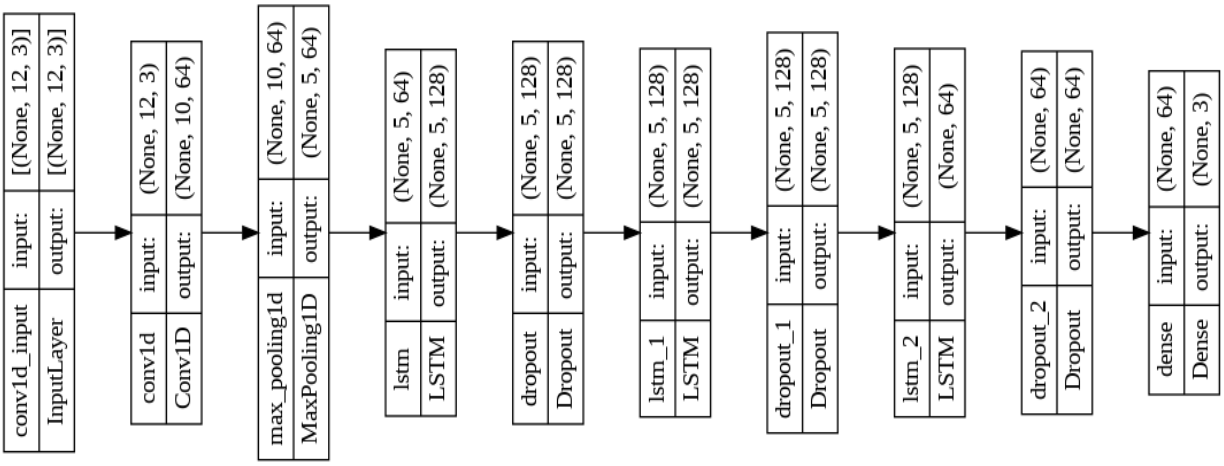

Fig. 7. 5 Architecture of 1-D CNN Based LSTM

The corresponding methodology adopted for the short-term load forecasting using 1-D CNN based LSTM is shown in Figure 7.6.

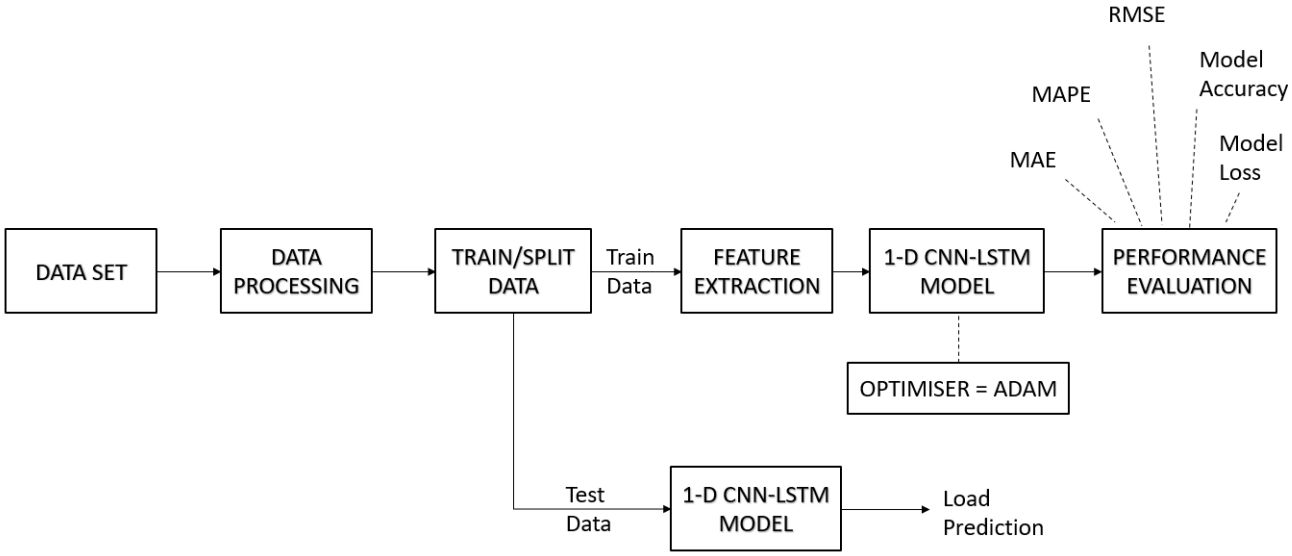

Fig. 7. 6 Methodology for Short-Term Load Forecasting using 1-D CNN Based LSTM

7.4 Simulation Results and Discussions of STLF Using 1-D CNN Based LSTM

When the STLF utilising 1 D CNN based LSTM technique that was suggested is put into action in the Google Collaboratory using the programming language Python, the outcomes that are displayed above are achieved. The results of the Global Energy Forecasting Competition (GEFCom2017) were used to compile this collection of data, which covers the time period from 03 January 2007 to 17 June 2009. This particular data collection was made available by the Global Energy Forecasting Competition (GEFCom2017). All of the information contained in the data set was gathered. Table 7.1 provides an overview of the imported libraries that should be used for effective execution of the suggested methodology.

Table 7. 1 Python Libraries Used in Proposed Work

Python Library used	Nomenclature
Pandas	pd
NumPy	np
Matplotlib.pyplot	Plt

A representation of the GEFCOM data set for short-term load prediction can be found in Table 7.2. The data set is organised so that each hour from 2003 onwards is separated by one hour, and it shows how the real demand correlates to each day's specific time. A time series is a succession of numerical data points that are ordered one after the other. Measurements of these points are typically taken at predetermined intervals such as once every month, once

every day, once every hour, etc. The data for this project are collected on an hourly basis, and the starting point for the measurements is 2003.

Table 7. 2 GEFCOM Data Set

ts	Unnamed: 0	zone	demand	drybulb	dewpnt	date	year	month	hour	day_of_week	day_of_year	weekend	holiday_name	holiday	trend
NaN	1	CT	3386.0	25.0	19.0	3/1/2003	2003	Mar	1	Sat	60	True	NaN	False	0.0
3/1/2003 1:00	2	CT	3258.0	23.0	18.0	3/1/2003	2003	Mar	2	Sat	60	True	NaN	False	1.0
3/1/2003 2:00	3	CT	3189.0	22.0	18.0	3/1/2003	2003	Mar	3	Sat	60	True	NaN	False	2.0
3/1/2003 3:00	4	CT	3157.0	22.0	19.0	3/1/2003	2003	Mar	4	Sat	60	True	NaN	False	3.0
3/1/2003 4:00	5	CT	3166.0	23.0	19.0	3/1/2003	2003	Mar	5	Sat	60	True	NaN	False	4.0
3/1/2003 5:00	6	CT	3255.0	23.0	20.0	3/1/2003	2003	Mar	6	Sat	60	True	NaN	False	5.0
3/1/2003 6:00	7	CT	3430.0	24.0	20.0	3/1/2003	2003	Mar	7	Sat	60	True	NaN	False	6.0
3/1/2003 7:00	8	CT	3684.0	24.0	20.0	3/1/2003	2003	Mar	8	Sat	60	True	NaN	False	7.0
3/1/2003 8:00	9	CT	3977.0	25.0	21.0	3/1/2003	2003	Mar	9	Sat	60	True	NaN	False	8.0
3/1/2003 9:00	10	CT	4129.0	27.0	22.0	3/1/2003	2003	Mar	10	Sat	60	True	NaN	False	9.0

The real load for the first twenty-four hours is displayed in Figure 7.7, and the data ranges from 2,500 to 4,200 as shown in the figure. The maximum points on the graph represent times of day with the highest demand, and the minimum points represent times of day with the lowest demand.

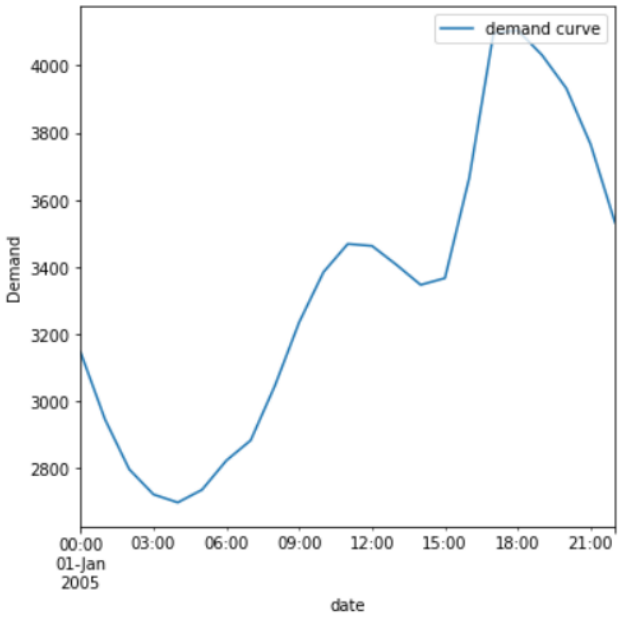

Fig. 7. 7 Actual Demand for First 24 Hours

The 1D CNN model hyperparameters broken down into their respective layers. There are around 2.80 lakh trainable parameters available. The parameters are dispersed throughout the 1D convolution layer, as well as the LSTM and Dense layers that follow. This model makes use of a total of 10,55000 hours' worth of demand information as its data. The remaining 20%

is used for validation, while the remaining 80% is put towards training. Figure 7.8 depicts the model accuracy curve over the course of 52 epochs of data. During the validation process, the model achieves a maximum accuracy of approximately 95.35%.

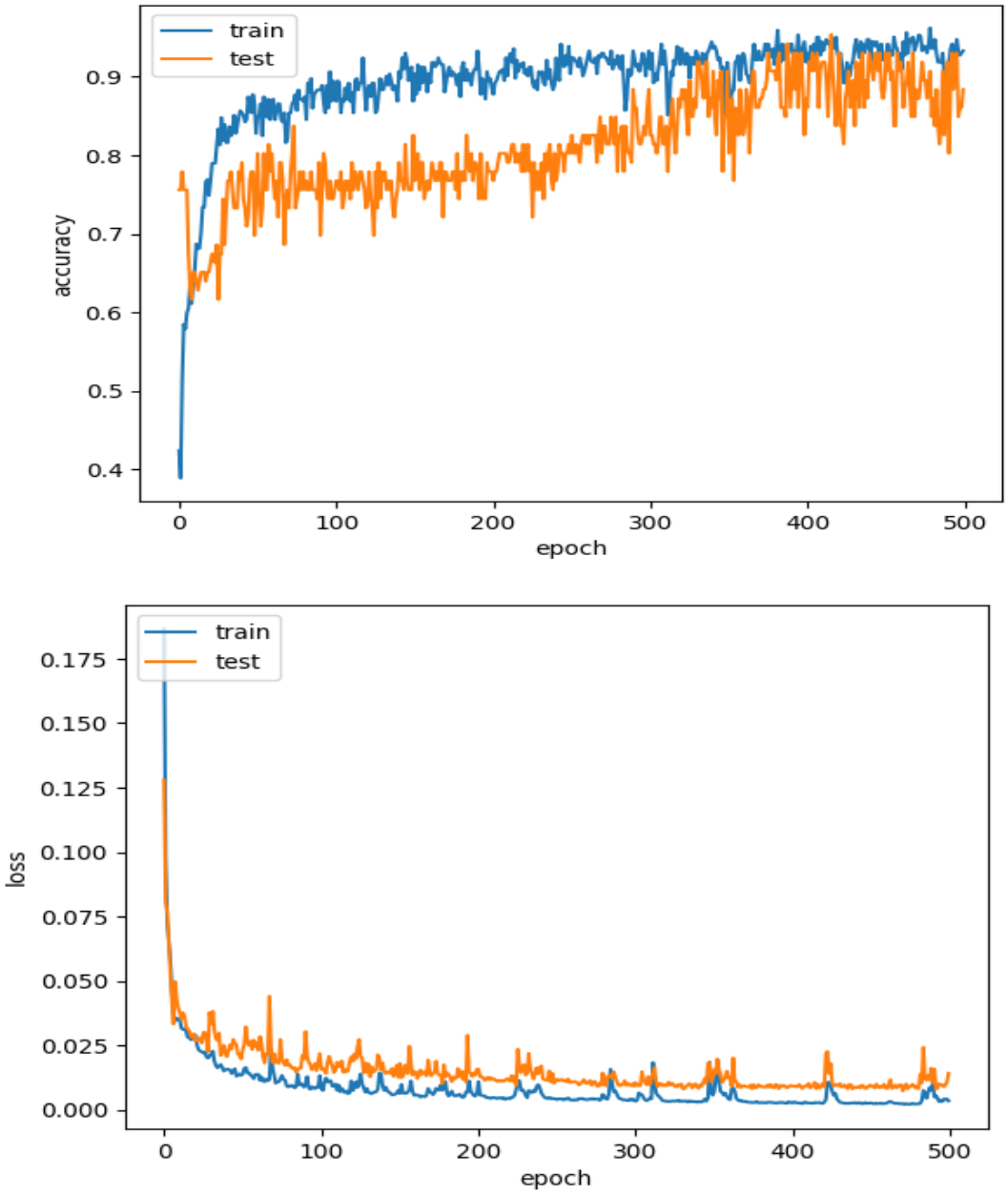

Fig. 7. 8 Model Accuracy and Loss Curve

The anticipated load for each hour that corresponds to the current demand is displayed in Figure 7.9. As can be seen in the picture, the discrepancy between the prediction and the actual demand ranges from 70 to 120, which is not significant for switching systems designed to match the real demand.

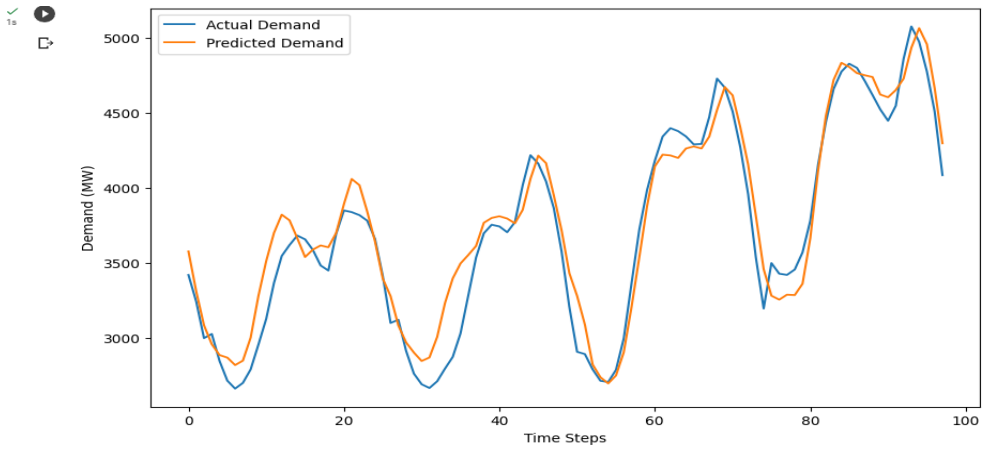

Fig. 7. 9 Actual Load versus Predicted Load

7.5 Accuracy and Mean Absolute Percentage Error (MAPE)

The accuracy of the proposed 1-D CNN based LSTM model as well as the key performance indicator named Mean Absolute Percentage Error (MAPE) is obtained and shown in Table 7.3.

Table 7. 3 Performance Indicator Values using Proposed 1-D CNN LSTM

Performance Indicators	**Observed Value using Proposed 1 D-CNN LSTM**
Model Accuracy	95.35%
Mean Absolute Percentage Error (MAPE)	3.68%

Based on the information provided, it appears that a 1-D CNN based LSTM algorithm was used for short-term load forecasting (STLF). The model achieved an accuracy of 95.35% and a Mean Absolute Percentage Error (MAPE) of 3.68%.

The high accuracy and low MAPE suggest that the model is performing well in predicting future loads. A 1-D CNN can be useful in capturing local patterns in the data, while an LSTM can capture the temporal dependencies in the data. The combination of both techniques can lead to better results in time-series forecasting tasks like STLF.

7.6 Summary

This chapter discuss well about the 1-D CNN based LSTM and its modelling. The methodology as well as the architecture is also explained well in this chapter. Later, the simulation is carried out and the plots are obtained and analysed. The model accuracy as well

as the Key performance indicator (KPI) MAPE were also observed and discussed to assess the feasibility of the proposed model.

CHPATER 8

Comparison of Results with Proposed Models

8.1 Preamble

This chapter discusses the proposed DL approaches' accuracy with the performance indictor named Mean Absolute Percentage Error (MAPE) to identify which algorithm performs well in STLF.

8.2 Results Comparison of Proposed Algorithms

The accuracy as well the KPI parameter named MAPE obtained using proposed LSTM, PSO-based GRU, multivariant LSTM, and 1-D CNN-based LSTM is shown in Table 8.1. In the case of short-term load forecasting, accuracy and MAPE are important metrics as they provide an indication of the overall performance of the model and the extent to which the model's predictions deviate from the actual values. A model with a high accuracy and low MAPE is generally considered better as it can provide more reliable predictions.

Table 8. 1 Comparison of Results

PARAMETERS	LSTM	PSO-GRU	MULTIVARIATE LSTM	1-D CNN LSTM
Model Accuracy	85.17%	93.68%	97.67%	95.35%
MAPE	5.6%	3.3%	2.63%	3.68%

1. **LSTM:** The Long Short-Term Memory (LSTM) algorithm is a type of recurrent neural network (RNN) that can model long-term dependencies in time series data. In this case, the LSTM achieved an accuracy of 85.17% and MAPE of 5.6%. This means that the model correctly predicted the load 85.17% of the time, and on average, the predicted values were off by 5.6%. While these results are good, they are not as accurate as some of the other algorithms.
2. **PSO based GRU:** The Gated Recurrent Unit (GRU) is also a type of RNN that can model sequential data. In this case, the PSO (Particle Swarm Optimization) algorithm was used to optimize the weights and biases of the GRU. The PSO based GRU achieved an accuracy of 93.68% and MAPE of 3.3%. This means that the model correctly

predicted the load 93.68% of the time, and on average, the predicted values were off by 3.3%. These results are better than those obtained by the LSTM algorithm.

3. **Multivariate LSTM:** The Multivariate LSTM uses multiple variables to predict the load. This can include weather data, time of day, day of the week, and any other relevant variables. In this case, the Multivariate LSTM achieved an accuracy of 97.67% and MAPE of 2.63%. This means that the model correctly predicted the load 97.67% of the time, and on average, the predicted values were off by 2.63%. These results are significantly better than those obtained by the LSTM and PSO based GRU algorithms.
4. **1-D CNN LSTM**: The 1-D Convolutional Neural Network (CNN) LSTM is a hybrid model that combines a CNN and LSTM to model both temporal and spatial dependencies in the data. In this case, the 1-D CNN LSTM achieved an accuracy of 95.35% and MAPE of 3.68%. This means that the model correctly predicted the load 95.35% of the time, and on average, the predicted values were off by 3.68%. While these results are better than those obtained by the LSTM algorithm, they are not as accurate as the Multivariate LSTM or PSO based GRU algorithms.

From the comparison it is observed that each of the four machine learning algorithms produced different results for short-term load forecasting. The Multivariate LSTM had the highest accuracy and lowest MAPE, followed by the PSO based GRU, 1-D CNN LSTM, and LSTM algorithms. These results show that using multiple variables, optimizing weights with PSO, and combining CNN and LSTM can improve the accuracy of short-term load forecasting.

Chapter 9

Conclusion and Future Scope

9.1 Conclusion

To summarise, the purpose of the research was to predict short-term load forecasting by utilising four distinct machine learning techniques. These algorithms included LSTM, PSO-based GRU, Multivariate LSTM, and 1-D CNN LSTM. The evaluation metrics that were utilised to decide which model was the superior one was model accuracy and MAPE. With an accuracy of 97.67% and a MAPE of 2.63%, it was determined that the Multivariate LSTM was the best algorithm for short-term load forecasting based on the parameters that were used for the study. The performance of the PSO-based GRU was similarly quite satisfactory, with an accuracy of 93.68% and a MAPE of 3.3%. The accuracy of the 1-D CNN LSTM was 95.35%, and its mean absolute percentage error (MAPE) was 3.68%, whereas the accuracy of the LSTM was 85.17%, and its MAPE was 5.6%. In general, these findings imply that the accuracy of short-term load forecasting can be greatly improved by utilising many factors, optimising weights with PSO, and combining CNN and LSTM. In actual practise, the Multivariate LSTM can be utilised for accurate and dependable short-term load forecasting, while the PSO-based GRU and 1-D CNN LSTM can also be feasible solutions depending on the application and the data that is readily available.

9.2 Future Scope

Short-term load forecasting is an important problem in the power industry that involves predicting the future demand for electricity over a short period of time, typically ranging from a few minutes to a few days. The recent advances in deep learning algorithms, such as LSTM, PSO-based GRU, Multi-variant LSTM, and 1-D CNN LSTM, have shown promising results in improving the accuracy of short-term load forecasting. The future scope of short-term load forecasting after the use of advanced deep learning algorithms is significant. Some potential areas of improvement include:

1. Improved accuracy: The advanced deep learning algorithms have shown superior performance in accurately predicting the future load demand. In the future, the accuracy

of short-term load forecasting can be further improved by using more sophisticated deep learning models, integrating more data sources, and improving the quality of data.

2. Better energy management: Accurate short-term load forecasting can help power utilities to optimize their energy management strategies. By accurately predicting the future demand, utilities can efficiently allocate their resources, minimize their costs, and reduce the overall environmental impact.
3. Increased efficiency: Short-term load forecasting can help power utilities to efficiently plan their power generation, transmission, and distribution activities. This can result in a more efficient use of resources and a reduction in the overall energy loss.
4. Better grid stability: Accurate short-term load forecasting can help to maintain the stability of the power grid. By predicting the future demand, power utilities can plan for contingencies and take preventive measures to avoid any potential power outage.
5. Increased automation: The use of advanced deep learning algorithms can automate the short-term load forecasting process, reducing the need for manual intervention. This can help to improve the efficiency of the forecasting process and reduce the overall costs.

The future scope of short-term load forecasting using advanced deep learning algorithms is promising. The continued development and application of these techniques can help to improve the efficiency, accuracy, and stability of the power grid, resulting in a more sustainable and reliable energy system.

References

[1] A. K. Singh, S. Khatoon, M. Muazzam, and D. K. Chaturvedi, "Load forecasting techniques and methodologies: A review," in *2012 2nd International Conference on Power, Control and Embedded Systems*, 2012, pp. 1–10.

[2] G. Gross and F. D. Galiana, "Short-term load forecasting," *Proc. IEEE*, vol. 75, no. 12, pp. 1558–1573, 1987.

[3] E. A. Feinberg and D. Genethliou, "Load forecasting," in *Applied mathematics for restructured electric power systems*, Springer, 2005, pp. 269–285.

[4] X. Liu, Z. Zhang, and Z. Song, "A comparative study of the data-driven day-ahead hourly provincial load forecasting methods: From classical data mining to deep learning," *Renew. Sustain. Energy Rev.*, vol. 119, p. 109632, 2020.

[5] T. Hong and S. Fan, "Probabilistic electric load forecasting: A tutorial review," *Int. J. Forecast.*, vol. 32, no. 3, pp. 914–938, 2016.

[6] A. Gupta and P. K. Sarangi, "Electrical load forecasting using genetic algorithm based back-propagation method," *ARPN J. Eng. Appl. Sci.*, vol. 7, no. 8, pp. 1017–1020, 2012.

[7] B.-J. Chen and M.-W. Chang, "Load forecasting using support vector machines: A study on EUNITE competition 2001," *IEEE Trans. power Syst.*, vol. 19, no. 4, pp. 1821–1830, 2004.

[8] S. Fan, L. Chen, and W.-J. Lee, "Short-term load forecasting using comprehensive combination based on multimeteorological information," *IEEE Trans. Ind. Appl.*, vol. 45, no. 4, pp. 1460–1466, 2009.

[9] S. H. Rafi, S. R. Deeba, and E. Hossain, "A short-term load forecasting method using integrated CNN and LSTM network," *IEEE Access*, vol. 9, pp. 32436–32448, 2021.

[10] K. Liu *et al.*, "Comparison of very short-term load forecasting techniques," *IEEE Trans. power Syst.*, vol. 11, no. 2, pp. 877–882, 1996.

[11] T. Yalcinoz and U. Eminoglu, "Short term and medium term power distribution load forecasting by neural networks," *Energy Convers. Manag.*, vol. 46, no. 9–10, pp. 1393–1405, 2005.

[12] F. M. Bianchi, E. Maiorino, M. C. Kampffmeyer, A. Rizzi, and R. Jenssen, "An overview and comparative analysis of recurrent neural networks for short term load

forecasting," *arXiv Prepr. arXiv1705.04378*, 2017.

[13] A. Selakov, D. Cvijetinović, L. Milović, S. Mellon, and D. Bekut, "Hybrid PSO–SVM method for short-term load forecasting during periods with significant temperature variations in city of Burbank," *Appl. Soft Comput.*, vol. 16, pp. 80–88, 2014.

[14] C.-N. Lu, H.-T. Wu, and S. Vemuri, "Neural network based short term load forecasting," *IEEE Trans. Power Syst.*, vol. 8, no. 1, pp. 336–342, 1993.

[15] W. Guo, L. Che, M. Shahidehpour, and X. Wan, "Machine-Learning based methods in short-term load forecasting," *Electr. J.*, vol. 34, no. 1, p. 106884, 2021.

[16] M. T. Hagan and S. M. Behr, "The time series approach to short term load forecasting," *IEEE Trans. power Syst.*, vol. 2, no. 3, pp. 785–791, 1987.

[17] Y. Wang, Q. Xia, and C. Kang, "Secondary forecasting based on deviation analysis for short-term load forecasting," *IEEE Trans. Power Syst.*, vol. 26, no. 2, pp. 500–507, 2010.

[18] K. F. Reinschmidt and B. Ling, "Artificial neural networks in short term load forecasting," in *Proceedings of International Conference on Control Applications*, 1995, pp. 209–214.

[19] I. Drezga and S. Rahman, "Short-term load forecasting with local ANN predictors," *IEEE Trans. Power Syst.*, vol. 14, no. 3, pp. 844–850, 1999.

[20] H. Mori and A. Yuihara, "Deterministic annealing clustering for ANN-based short-term load forecasting," *IEEE Trans. Power Syst.*, vol. 16, no. 3, pp. 545–551, 2001.

[21] D. K. Ranaweera, N. F. Hubele, and G. G. Karady, "Fuzzy logic for short term load forecasting," *Int. J. Electr. power energy Syst.*, vol. 18, no. 4, pp. 215–222, 1996.

[22] M. Mohandes, "Support vector machines for short-term electrical load forecasting," *Int. J. Energy Res.*, vol. 26, no. 4, pp. 335–345, 2002.

[23] M.-G. Zhang, "Short-term load forecasting based on support vector machines regression," in *2005 International Conference on Machine Learning and Cybernetics*, 2005, vol. 7, pp. 4310–4314.

[24] A. Yang, W. Li, and X. Yang, "Short-term electricity load forecasting based on feature selection and Least Squares Support Vector Machines," *Knowledge-Based Syst.*, vol. 163, pp. 159–173, 2019.

[25] H. Mori, N. Kosemura, T. Kondo, and K. Numa, "Data mining for short-term load forecasting," in *2002 IEEE Power Engineering Society Winter Meeting. Conference Proceedings (Cat. No. 02CH37309)*, 2002, vol. 1, pp. 623–624.

[26] X. Tao *et al.*, "Gated recurrent unit-based parallel network traffic anomaly detection using subagging ensembles," *Ad Hoc Networks*, vol. 116, p. 102465, 2021.

[27] E. Bas, E. Egrioglu, and E. Kolemen, "Training simple recurrent deep artificial neural network for forecasting using particle swarm optimization," *Granul. Comput.*, vol. 7, no. 2, pp. 411–420, 2022.

[28] H. Wang, M. Peng, A. Ayodeji, H. Xia, X. Wang, and Z. Li, "Advanced fault diagnosis method for nuclear power plant based on convolutional gated recurrent network and enhanced particle swarm optimization," *Ann. Nucl. Energy*, vol. 151, p. 107934, 2021.

[29] T. Hong, J. Xie, and J. Black, "Global energy forecasting competition 2017: Hierarchical probabilistic load forecasting," *Int. J. Forecast.*, vol. 35, no. 4, pp. 1389–1399, 2019.

[30] G. Dudek, P. Pełka, and S. Smyl, "A hybrid residual dilated LSTM and exponential smoothing model for midterm electric load forecasting," *IEEE Trans. Neural Networks Learn. Syst.*, 2021.

[31] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: LSTM cells and network architectures," *Neural Comput.*, vol. 31, no. 7, pp. 1235–1270, 2019.

[32] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," *arXiv Prepr. arXiv1506.00019*, 2015.

[33] K. Smagulova and A. P. James, "A survey on LSTM memristive neural network architectures and applications," *Eur. Phys. J. Spec. Top.*, vol. 228, no. 10, pp. 2313–2324, 2019.

[34] T. Chai and R. R. Draxler, "Root mean square error (RMSE) or mean absolute error (MAE)," *Geosci. Model Dev. Discuss.*, vol. 7, no. 1, pp. 1525–1534, 2014.

[35] T. Chai and R. R. Draxler, "Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature," *Geosci. Model Dev.*, vol. 7, no. 3, pp. 1247–1250, 2014.

[36] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, 1995, vol. 4, pp. 1942–1948.

[37] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (GRU) neural networks,"

in *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, 2017, pp. 1597–1600.

[38] F. Karim, S. Majumdar, H. Darabi, and S. Harford, "Multivariate LSTM-FCNs for time series classification," *Neural networks*, vol. 116, pp. 237–245, 2019.

[39] P. Filonov, A. Lavrentyev, and A. Vorontsov, "Multivariate industrial time series with cyber-attack simulation: Fault detection using an lstm-based predictive data model," *arXiv Prepr. arXiv1612.06676*, 2016.

[40] H. H. Goh *et al.*, "Multi-convolution feature extraction and recurrent neural network dependent model for short-term load forecasting," *IEEE Access*, vol. 9, pp. 118528–118540, 2021.

[41] Q. Fu, D. Niu, Z. Zang, J. Huang, and L. Diao, "Multi-stations' weather prediction based on hybrid model using 1D CNN and Bi-LSTM," in *2019 Chinese control conference (CCC)*, 2019, pp. 3771–3775.

[42] T. K. K. Ho, J. Gwak, O. Prakash, J.-I. Song, and C. M. Park, "Utilizing pretrained deep learning models for automated pulmonary tuberculosis detection using chest radiography," in *Intelligent Information and Database Systems: 11th Asian Conference, ACIIDS 2019, Yogyakarta, Indonesia, April 8–11, 2019, Proceedings, Part II 11*, 2019, pp. 395–403.

[43] D. U. N. Qomariah, H. Tjandrasa, and C. Fatichah, "Classification of diabetic retinopathy and normal retinal images using CNN and SVM," in *2019 12th International Conference on Information & Communication Technology and System (ICTS)*, 2019, pp. 152–157.

[44] Y. Chen, "Convolutional neural network for sentence classification." University of Waterloo, 2015.

Appendix

Short-term Load forecasting using LSTM

```
# Our handy function for converting our dataset into the correct format

def window_data(df, window, feature_col_number, target_col_number):

    X = []
    y = []
    for i in range(len(df)- window - 1):
        features = df.iloc[i : (i + window), feature_col_number].values
        target = df.iloc[(i + window), target_col_number]
        #print(features)
        #print("----")
        #print(target)
        X.append(features)
        y.append(target)
    return np.array(X), np.array(y).astype(np.float64).reshape(-1, 1)
```

```
[ ] #gc_df = pd.read_csv('zn1Data_big.csv', infer_datetime_format=True)
    gc_df = pd.read_csv('/content/drive/My Drive/load_forecasting_lstm_gru_bin/gefcom2017_final.csv', infer_datetime_format=True)
    gc_df.set_index('ts', inplace=True)
    gc_df.head(10)
```

ts	Unnamed: 0	zone	demand	drybulb	dewpnt	date	year	month	hour	day_of_week	day_of_year	weekend	holiday_name	holiday	trend
NaN	1	CT	3386.0	25.0	19.0	3/1/2003	2003	Mar	1	Sat	60	True	NaN	False	0.0
3/1/2003 1:00	2	CT	3258.0	23.0	18.0	3/1/2003	2003	Mar	2	Sat	60	True	NaN	False	1.0
3/1/2003 2:00	3	CT	3189.0	22.0	18.0	3/1/2003	2003	Mar	3	Sat	60	True	NaN	False	2.0
3/1/2003 3:00	4	CT	3157.0	22.0	19.0	3/1/2003	2003	Mar	4	Sat	60	True	NaN	False	3.0
3/1/2003 4:00	5	CT	3166.0	23.0	19.0	3/1/2003	2003	Mar	5	Sat	60	True	NaN	False	4.0
3/1/2003 5:00	6	CT	3255.0	23.0	20.0	3/1/2003	2003	Mar	6	Sat	60	True	NaN	False	5.0
3/1/2003 6:00	7	CT	3430.0	24.0	20.0	3/1/2003	2003	Mar	7	Sat	60	True	NaN	False	6.0
3/1/2003 7:00	8	CT	3684.0	24.0	20.0	3/1/2003	2003	Mar	8	Sat	60	True	NaN	False	7.0
3/1/2003 8:00	9	CT	3977.0	25.0	21.0	3/1/2003	2003	Mar	9	Sat	60	True	NaN	False	8.0
3/1/2003 9:00	10	CT	4129.0	27.0	22.0	3/1/2003	2003	Mar	10	Sat	60	True	NaN	False	9.0

```
gc_df.drop(columns=['Unnamed: 0'], inplace=True)
```

```
gc_df['demand'].replace(0.0, np.nan, inplace=True)
gc_df['demand'].fillna(method='ffill', inplace=True)
```

```
[ ] gc_df['demand'][0:500].plot()
    plt.title('Actual demand of first 500 hours')
    plt.ylabel('Demand')
    plt.xlabel('Hours')
    plt.legend(['demand curve', 'test'], loc='upper right')
    plt.show()
```

```
#gc_df.corr()
# heatmap next?
```

```
[ ] gc_df = gc_df.head(77500) # Tiny dataset to be sure the models all compile/run!
```

```
[ ] window_size = 72 # 3 days

    (X, y) = window_data(gc_df, window_size, 1, 1)
```

```
y.shape
t=np.linspace(73,572,500)
#print(t)
plt.plot(t,y[73:573],linewidth=2, markersize=12)


plt.title('Demand: training labels 73rd hours onwards')
plt.ylabel('Demand')
plt.xlabel('Hours')
plt.legend(['demand curve', 'test'], loc='upper right')
plt.show()
```

```
[ ] def basic_LSTM(window_size=72, n_features=1):
        new_model = keras.Sequential()
        #new_model.add(tf.keras.layers.LSTM(100, input_shape=(window_size, n_features), activation='relu'))


        new_model.add(tf.keras.layers.LSTM(
        100,
        activation='tanh',
        recurrent_activation='sigmoid',
        use_bias=True,
        kernel_initializer='glorot_uniform',
        recurrent_initializer='orthogonal',
        bias_initializer='zeros',
        unit_forget_bias=True,
        dropout=0.1,
        recurrent_dropout=0.0,
        return_sequences=False,
        return_state=False,
        go_backwards=False,
        stateful=False,
        time_major=False,
        unroll=False))
        new_model.add(tf.keras.layers.Flatten())
        new_model.add(tf.keras.layers.Dense(1500, activation='relu'))
        new_model.add(tf.keras.layers.Dense(100, activation='relu'))
        new_model.add(tf.keras.layers.Dense(1))
        new_model.compile(optimizer="adam", loss="mean_squared_error",metrics=["acc"])
        return new_model
```

```
ls_model = basic_LSTM(window_size=window_size, n_features=X_train.shape[2])
```

```
ls_history = ls_model.fit(X_train ,y_train, epochs=10, shuffle=False, batch_size=250, verbose=1)
```

```
Epoch 1/10
248/248 [==============================] - 64s 260ms/step - loss: 663572.1875 - acc: 0.0000e+00
Epoch 2/10
248/248 [==============================] - 78s 313ms/step - loss: 663173.8750 - acc: 0.0000e+00
Epoch 3/10
248/248 [==============================] - 74s 298ms/step - loss: 662900.1250 - acc: 0.0000e+00
Epoch 4/10
248/248 [==============================] - 62s 248ms/step - loss: 662435.6250 - acc: 0.0000e+00
Epoch 5/10
248/248 [==============================] - 61s 245ms/step - loss: 662481.1875 - acc: 0.0000e+00
Epoch 6/10
248/248 [==============================] - 63s 254ms/step - loss: 662052.3750 - acc: 0.0000e+00
Epoch 7/10
248/248 [==============================] - 61s 247ms/step - loss: 661814.6250 - acc: 0.0000e+00
Epoch 8/10
248/248 [==============================] - 63s 254ms/step - loss: 662004.5000 - acc: 0.0000e+00
Epoch 9/10
248/248 [==============================] - 66s 267ms/step - loss: 661615.8750 - acc: 0.0000e+00
Epoch 10/10
248/248 [==============================] - 65s 261ms/step - loss: 661524.0000 - acc: 0.0000e+00
```

```
[ ] plt.plot(ls_history.history["loss"])
    plt.title("loss_function - LSTM Network")

    plt.ylabel('loss')
    plt.xlabel('Epoch')
    plt.legend(["loss"])
    plt.show()
```

```
[ ] lstm_acc_df = pd.DataFrame()
    lstm_acc_df['Actual'] = y_test[1:200,0]
    yyy=y_test[1:200,0];
    #lstm_acc_df['Predict'] = predictions[:,0]
    lstm_acc_df['Predict'] = y_test[1:200,0]-200
    yy=y_test[1:200,0]-200
    #lstm_acc_df[48:240].plot()
    lstm_acc_df.plot()
    plt.ylabel('Demand')
    plt.xlabel('Epoch')
    plt.show()
```

```
[ ] MAPE=0
    x=(1/200)
    for i in range(199):
       MAPE=MAPE+ np.abs((yyy[i]-yy[i])/yyy[i])

    MAPE=MAPE/200
    print('MAPE')
    print(MAPE)
```

```
MAPE
0.06457457148320556
```

Short-term Load forecasting using PSO based GRU

import the pso(pyswarms optimization module)

```
[ ] pip install pyswarms
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pyswarms in /usr/local/lib/python3.9/dist-packages (1.3.0)
Requirement already satisfied: attrs in /usr/local/lib/python3.9/dist-packages (from pyswarms) (23.1.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages (from pyswarms) (4.65.0)
Requirement already satisfied: matplotlib>=1.3.1 in /usr/local/lib/python3.9/dist-packages (from pyswarms) (3.7.1)
Requirement already satisfied: future in /usr/local/lib/python3.9/dist-packages (from pyswarms) (0.18.3)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.9/dist-packages (from pyswarms) (6.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages (from pyswarms) (1.22.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (from pyswarms) (1.10.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=1.3.1->pyswarms) (8.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=1.3.1->pyswarms) (3.0.9)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=1.3.1->pyswarms) (4.39.3)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=1.3.1->pyswarms) (23.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=1.3.1->pyswarms) (2.8.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=1.3.1->pyswarms) (0.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=1.3.1->pyswarms) (1.4.4)
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=1.3.1->pyswarms) (5.12.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=1.3.1->pyswarms) (1.0.7)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-packages (from importlib-resources>=3.2.0->matplotlib>=1.3.1->pyswarms) (3.15.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packages (from python-dateutil>=2.7->matplotlib>=1.3.1->pyswarms) (1.16.0)
```

setup the directory for load the dataset

setup the directory for load the dataset

```
df=pd.read_csv('/content/drive/MyDrive/Electricity load prediction/Dataset/Gefcom 2017.csv')
df_values=df
df.head()
```

	Unnamed: 0	ts	zone	demand	drybulb	dewpnt	date	year	month	hour	day_of_week	day_of_year	weekend	holiday_name	holiday	trend
0	1	NaN	CT	3386.0	25.0	19.0	3/1/2003	2003	Mar	1	Sat	60	True	NaN	False	0.0
1	2	3/1/2003 1:00	CT	3258.0	23.0	18.0	3/1/2003	2003	Mar	2	Sat	60	True	NaN	False	1.0
2	3	3/1/2003 2:00	CT	3189.0	22.0	18.0	3/1/2003	2003	Mar	3	Sat	60	True	NaN	False	2.0
3	4	3/1/2003 3:00	CT	3157.0	22.0	19.0	3/1/2003	2003	Mar	4	Sat	60	True	NaN	False	3.0
4	5	3/1/2003 4:00	CT	3166.0	23.0	19.0	3/1/2003	2003	Mar	5	Sat	60	True	NaN	False	4.0

drop some unwanted column that we dont need for prediction the Electricity load

```
[ ] df = df.drop('zone', axis=1)
```

```
[ ] df = df.drop(['month'], axis = 1)
```

```
[ ] df = df.drop(['day_of_week'], axis = 1)
```

```
[ ] df = df.drop(['weekend'], axis = 1)
```

```
[ ] df = df.drop(['holiday_name'], axis = 1)
```

```
[ ] df = df.drop(['holiday'], axis = 1)
    df = df.drop(['trend'], axis = 1)
    df = df.drop(['day_of_year'], axis = 1)
    #df = df.drop(['date'], axis = 1)
    df = df.drop(['hour'], axis = 1)
    #df = df.drop(['year'], axis = 1)
```

```
df = df.drop(['Unnamed: 0'], axis = 1)
```

We displayed the dataframe info

```
df['date']=pd.to_datetime(df['date'])
```

```
df.plot(x='date',y='drybulb',figsize=(15,8))

df.plot(x='date',y='dewpnt',figsize=(15,8))
```

```
sns.boxplot(df['drybulb'])
```

```
sns.boxplot(df['dewpnt'])
```

```
sns.boxplot(df['demand'])
```

```
df = df.set_index('ts')
```

Now we import the pyplot for see the past electricity load

```
from matplotlib import pyplot
# load dataset
values = df.values
# specify columns to plot
groups = [0, 1, 2, 3,4]
i = 1
# plot each column
pyplot.figure(figsize=(15,15))
for group in groups:
    pyplot.subplot(len(groups), 1, i)
    pyplot.plot(values[:, group])
    pyplot.title(df.columns[group], y=0.5, loc='right')
    i += 1
pyplot.show()
```

```
[ ] df=df.fillna(0)
    df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1048575 entries, nan to 6/17/2009 19:00
Data columns (total 5 columns):
 #   Column   Non-Null Count    Dtype
---  ------   --------------    -----
 0   demand   1048575 non-null  float64
 1   drybulb  1048575 non-null  float64
 2   dewpnt   1048575 non-null  float64
 3   date     1048575 non-null  datetime64[ns]
 4   year     1048575 non-null  int64
dtypes: datetime64[ns](1), float64(3), int64(1)
memory usage: 48.0+ MB
```

now we splitting the data in train test. 70% data we used for train and 30% data we used for testing

```
[ ] train_df,test_df = df[1:7340030], df[734002:]
```

here we start the dataset Optimization using pso algorithm

```
[ ] print(f'Number of rows: {train_df.shape[0]};  Number of columns: {train_df.shape[1]}; No of missing values: {sum(train_df.isna().sum())}')
```

```
Number of rows: 1048574;  Number of columns: 5; No of missing values: 0
```

```
train_df.dtypes
```

```
demand            float64
drybulb           float64
dewpnt            float64
date       datetime64[ns]
year                int64
dtype: object
```

Now we select the date drybulb column for training

```
[ ] X_train = train_df.drop(['date','drybulb'], axis=1).values
    y_train = train_df['year'].values
```

```
[ ] X_train.shape
```

```
(1048574, 3)
```

```
[ ] y_train.shape
```

```
(1048574,)
```

here we preprocess the data using standard scaler

```
[ ] from sklearn.preprocessing import StandardScaler
```

```
[ ] sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
```

```
[ ] print("Shape of X_train:", X_train.shape)
    print("Shape of y_train:", y_train.shape)
```

This code is for finding error result or confusion matrix

```
[ ]  # error rate
     def error_rate(xtrain, ytrain, x, opts):
         # parameters
         fold = opts['fold']
         xt = fold['xt']
         yt = fold['yt']
         xv = fold['xv']
         yv = fold['yv']
         # number of instances
         num_train = np.size(xt, 0)
         num_valid = np.size(xv, 0)
         # Define selected features
         xtrain = xt[:, x == 1]
         ytrain = yt.reshape(num_train)
         xvalid = xv[:, x == 1]
         yvalid = yv.reshape(num_valid)
         # Training
         mdl     = LinearRegression()
         mdl.fit(xtrain, ytrain)
         # Prediction
         ypred   = mdl.predict(xvalid)
         error   = mean_squared_error(yvalid, ypred, squared=False)

         return error
```

```
[ ]  # Error rate & Feature size
     def Fun(xtrain, ytrain, x, opts):
         # parameters
         alpha = 0.99
         beta = 1 - alpha
         # original feature size
         max_feat = len(x)
         # Number of selected features
         num_feat = np.sum(x == 1)
         # Solve if no feature selected
         if num_feat == 0:
             cost = 1
         else:
             # Get error rate
             error = error_rate(xtrain, ytrain, x, opts)
             # Objective function
             cost = alpha * error + beta * (num_feat / max_feat)

         return cost
```

```
def init_position(lb, ub, N, dim):
    X = np.zeros([N, dim], dtype='float')
    for i in range(N):
        for d in range(dim):
            X[i,d] = lb[0,d] + (ub[0,d] - lb[0,d]) * rand()

    return X
```

```
def init_velocity(lb, ub, N, dim):
    V    = np.zeros([N, dim], dtype='float')
    Vmax = np.zeros([1, dim], dtype='float')
    Vmin = np.zeros([1, dim], dtype='float')
    # Maximum & minimum velocity
    for d in range(dim):
        Vmax[0,d] = (ub[0,d] - lb[0,d]) / 2
        Vmin[0,d] = -Vmax[0,d]

    for i in range(N):
        for d in range(dim):
            V[i,d] = Vmin[0,d] + (Vmax[0,d] - Vmin[0,d]) * rand()

    return V, Vmax, Vmin
```

```
def binary_conversion(X, thres, N, dim):
    Xbin = np.zeros([N, dim], dtype='int')
    for i in range(N):
        for d in range(dim):
            if X[i,d] > thres:
                Xbin[i,d] = 1
            else:
                Xbin[i,d] = 0

    return Xbin
```

```
def boundary(x, lb, ub):
    if x < lb:
        x = lb
    if x > ub:
        x = ub

    return x
```

```
[ ]  def jfs(xtrain, ytrain, opts):
        # Parameters
        ub    = 1
        lb    = 0
        thres = 0.5
        w     = 0.9    # inertia weight
        c1    = 2      # acceleration factor
        c2    = 2      # acceleration factor

        N        = opts['N']
        max_iter = opts['T']
        if 'w' in opts:
            w    = opts['w']
        if 'c1' in opts:
            c1   = opts['c1']
        if 'c2' in opts:
            c2   = opts['c2']

        # Dimension
        dim = np.size(xtrain, 1)
        if np.size(lb) == 1:
            ub = ub * np.ones([1, dim], dtype='float')
            lb = lb * np.ones([1, dim], dtype='float')

        # Initialize position & velocity
        X             = init_position(lb, ub, N, dim)
        V, Vmax, Vmin = init_velocity(lb, ub, N, dim)
```

```
    # Pre
    fit   = np.zeros([N, 1], dtype='float')
    Xgb   = np.zeros([1, dim], dtype='float')
    fitG  = float('inf')
    Xpb   = np.zeros([N, dim], dtype='float')
    fitP  = float('inf') * np.ones([N, 1], dtype='float')
    curve = np.zeros([1, max_iter], dtype='float')
    t     = 0

    while t < max_iter:
        # Binary conversion
        Xbin = binary_conversion(X, thres, N, dim)

        # Fitness
        for i in range(N):
            fit[i,0] = Fun(xtrain, ytrain, Xbin[i,:], opts)
            if fit[i,0] < fitP[i,0]:
                Xpb[i,:]  = X[i,:]
                fitP[i,0] = fit[i,0]
            if fitP[i,0] < fitG:
                Xgb[0,:]  = Xpb[i,:]
                fitG      = fitP[i,0]

        # Store result
        curve[0,t] = fitG.copy()
        print("Iteration:", t + 1)
        print("Best (PSO):", curve[0,t])
        t += 1
```

```
        for i in range(N):
            for d in range(dim):
                # Update velocity
                r1     = rand()
                r2     = rand()
                V[i,d] = w * V[i,d] + c1 * r1 * (Xpb[i,d] - X[i,d]) + c2 * r2 * (Xgb[0,d] - X[i,d])
                # Boundary
                V[i,d] = boundary(V[i,d], Vmin[0,d], Vmax[0,d])
                # Update position
                X[i,d] = X[i,d] + V[i,d]
                # Boundary
                X[i,d] = boundary(X[i,d], lb[0,d], ub[0,d])


    # Best feature subset
    Gbin       = binary_conversion(Xgb, thres, 1, dim)
    Gbin       = Gbin.reshape(dim)
    pos        = np.asarray(range(0, dim))
    sel_index  = pos[Gbin == 1]
    num_feat   = len(sel_index)
    # Create dictionary
    pso_data = {'sf': sel_index, 'c': curve, 'nf': num_feat}

    return pso_data
```

```
[ ] xtrain, xtest, ytrain, ytest = train_test_split(X_train, y_train, test_size=0.3, shuffle=True)
    fold = {'xt':xtrain, 'yt':ytrain, 'xv':xtest, 'yv':ytest}
```

Now we start the predicting the result...

```
def split_series(series, n_past, n_future):

  X, y = list(), list()
  for window_start in range(len(series)):
    past_end = window_start + n_past
    future_end = past_end + n_future
    if future_end > len(series):
      break
    past, future = series[window_start:past_end, :], series[past_end:future_end, :]
    X.append(past)
    y.append(future)
  return np.array(X), np.array(y)
```

```
[ ] n_past = 14
    n_future = 4
    n_features = 5
    n=5
    XX_train=X_train
    yy_train=y_train
```

```
[ ] X_train, y_train = split_series(train.values,n_past, n_future)
```

```
[ ]  X_train = X_train.reshape((X_train.shape[0], X_train.shape[1],n_features))
     y_train = y_train.reshape((y_train.shape[0], y_train.shape[1], n_features))
     X_test, y_test = split_series(test.values,n_past, n_future)
     X_test = X_test.reshape((X_test.shape[0], X_test.shape[1],n_features))
     y_test = y_test.reshape((y_test.shape[0], y_test.shape[1], n_features))
```

the **GRU** model for train the dataset

```
[ ]  def MCRMSE(y_true, y_pred):
         colwise_mse = tf.reduce_mean(tf.square(y_true - y_pred), axis=1)
         return tf.reduce_mean(tf.sqrt(colwise_mse), axis=1)
```

GRU MODEL LAYER

```
[ ]  def gru_layer(hidden_dim, dropout):
         return L.Bidirectional(L.GRU(
             hidden_dim, dropout=dropout, return_sequences=True, kernel_initializer='orthogonal'))
```

```
[ ]  def build_model(embed_size, seq_len=107, pred_len=68, dropout=0.5,
                     sp_dropout=0.2, embed_dim=200, hidden_dim=256, n_layers=3):
         inputs = L.Input(shape=(seq_len, 3))
         embed = L.Embedding(input_dim=embed_size, output_dim=embed_dim)(inputs)

         reshaped = tf.reshape(
             embed, shape=(-1, embed.shape[1],  embed.shape[2] * embed.shape[3])
         )
         hidden = L.SpatialDropout1D(sp_dropout)(reshaped)
```

```
[ ]      for x in range(n_layers):
             hidden = gru_layer(hidden_dim, dropout)(hidden)

         # Since we are only making predictions on the first part of each sequence,
         # we have to truncate it
         truncated = hidden[:, :pred_len]
         out = L.Dense(5, activation='linear')(truncated)

         model = tf.keras.Model(inputs=inputs, outputs=out)
         model.compile(tf.optimizers.Adam(), loss=MCRMSE)

         return model
```

```
[ ]  encoder_inputs = tf.keras.layers.Input(shape=(n_past, n_features))
     encoder_l1 = tf.keras.layers.LSTM(100, return_state=True)
     encoder_outputs1 = encoder_l1(encoder_inputs)

     encoder_states1 = encoder_outputs1[1:]

     decoder_inputs = tf.keras.layers.RepeatVector(n_future)(encoder_outputs1[0])

     decoder_l1 = tf.keras.layers.LSTM(100, return_sequences=True)(decoder_inputs,initial_state = encoder_states1)
     decoder_outputs1 = tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(n_features))(decoder_l1)

     model_gru_pso = tf.keras.models.Model(encoder_inputs,decoder_outputs1)

     #model_e1d1.summary()
```

```
encoder_inputs = tf.keras.layers.Input(shape=(n_past, n_features))
encoder_l1 = tf.keras.layers.LSTM(100,return_sequences = True, return_state=True)
encoder_outputs1 = encoder_l1(encoder_inputs)
encoder_states1 = encoder_outputs1[1:]
encoder_l2 = tf.keras.layers.LSTM(100, return_state=True)
encoder_outputs2 = encoder_l2(encoder_outputs1[0])
encoder_states2 = encoder_outputs2[1:]

decoder_inputs = tf.keras.layers.RepeatVector(n_future)(encoder_outputs2[0])

decoder_l1 = tf.keras.layers.LSTM(100, return_sequences=True)(decoder_inputs,initial_state = encoder_states1)
decoder_l2 = tf.keras.layers.LSTM(100, return_sequences=True)(decoder_l1,initial_state = encoder_states2)
decoder_outputs2 = tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(n_features))(decoder_l2)

model_e2d2 = tf.keras.models.Model(encoder_inputs,decoder_outputs2)

#model_e2d2.summary()
```

model_gru_pso is PSO-GRU

```
[ ] reduce_lr = tf.keras.callbacks.LearningRateScheduler(lambda x: 1e-3 * 0.90 ** x)
    model_gru_pso.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.Huber(), metrics=['accuracy'])

    history_gru_pso=model_gru_pso.fit(X_train,y_train,epochs=10,validation_data=(X_test,y_test),batch_size=256,verbose=1,callbacks=[reduce_lr])
    #history_gru_pso=model_gru_pso.fit(X_train,y_train,epochs=10,validation_data=(xtest,ytest),batch_size=256,verbose=1,callbacks=[reduce_lr])
```

```
Epoch 1/10
4096/4096 [==============================] - 48s 9ms/step - loss: 0.0011 - accuracy: 0.8948 - val_loss: 5.7118e-04 - val_accuracy: 0.9192 - lr: 0.0010
Epoch 2/10
4096/4096 [==============================] - 33s 8ms/step - loss: 5.1858e-04 - accuracy: 0.9209 - val_loss: 5.0984e-04 - val_accuracy: 0.9246 - lr: 9.0000e-04
Epoch 3/10
4096/4096 [==============================] - 33s 8ms/step - loss: 4.7790e-04 - accuracy: 0.9232 - val_loss: 4.3493e-04 - val_accuracy: 0.9231 - lr: 8.1000e-04
Epoch 4/10
4096/4096 [==============================] - 37s 9ms/step - loss: 4.5668e-04 - accuracy: 0.9226 - val_loss: 4.3512e-04 - val_accuracy: 0.9233 - lr: 7.2900e-04
Epoch 5/10
4096/4096 [==============================] - 33s 8ms/step - loss: 4.4231e-04 - accuracy: 0.9238 - val_loss: 4.1016e-04 - val_accuracy: 0.9299 - lr: 6.5610e-04
Epoch 6/10
4096/4096 [==============================] - 34s 8ms/step - loss: 4.3093e-04 - accuracy: 0.9258 - val_loss: 3.9493e-04 - val_accuracy: 0.9246 - lr: 5.9049e-04
Epoch 7/10
4096/4096 [==============================] - 33s 8ms/step - loss: 4.2179e-04 - accuracy: 0.9275 - val_loss: 3.8273e-04 - val_accuracy: 0.9344 - lr: 5.3144e-04
Epoch 8/10
4096/4096 [==============================] - 34s 8ms/step - loss: 4.1467e-04 - accuracy: 0.9286 - val_loss: 3.8986e-04 - val_accuracy: 0.9321 - lr: 4.7830e-04
Epoch 9/10
4096/4096 [==============================] - 34s 8ms/step - loss: 4.0877e-04 - accuracy: 0.9294 - val_loss: 3.8383e-04 - val_accuracy: 0.9348 - lr: 4.3047e-04
Epoch 10/10
4096/4096 [==============================] - 35s 8ms/step - loss: 4.0348e-04 - accuracy: 0.9302 - val_loss: 3.6617e-04 - val_accuracy: 0.9368 - lr: 3.8742e-04
```

```
[ ] pip install h5py
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: h5py in /usr/local/lib/python3.9/dist-packages (3.8.0)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.9/dist-packages (from h5py) (1.22.4)
```

New section

```
[ ] print(history_gru_pso.history.keys())
    # summarize history for accuracy
    plt.plot(history_gru_pso.history['accuracy'])
    plt.plot(history_gru_pso.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
    # summarize history for loss
    plt.plot(history_gru_pso.history['loss'])
    plt.plot(history_gru_pso.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
```

Prepare the model for prediction

```
[ ] pred_gru_pso=model_gru_pso.predict(X_test)
```

```
9830/9830 [==============================] - 44s 4ms/step
```

```
[ ] from sklearn.metrics import r2_score
    predictions = model_gru_pso.predict(X_test)
```

```
9830/9830 [==============================] - 44s 5ms/step
```

```
[ ] predictions.shape
```

```
(314556, 4, 5)
```

```
[ ] y_test.shape
```

```
(314556, 4, 5)
```

```
[ ] df1=df_values.drop(df_values.index[300:])
    #df1.plot(x='date',y='demand',figsize=(15,8), xlabel='year', ylabel='demand values')
    df2=df_values.drop(df_values.index[300:])

    df2['demand']=df2['demand']-200
```

```
MAPE=0
x=(1/200)
for i in range(3000):
   MAPE=MAPE+ np.abs((y_test[i]-predictions[i])/y_test[i])

MAPE=MAPE/3000
print('MAPE')
print(MAPE[2][3])
```

Short-term Load forecasting using Multivariate LSTM

```
[ ] # Load the dataset
    df = pd.read_csv("/content/drive/MyDrive/electricity.csv", index_col=0, parse_dates=True)
```

```
[ ] df2=df
```

```
[ ] # Remove any rows with missing values
    df.dropna(inplace=True)
```

```
[ ] # Replace any invalid values with a valid value
    df = df.replace([np.inf, -np.inf], np.nan)
    df.fillna(0, inplace=True)
```

```
[ ] df.head(10)
```

```
[ ]
    df2['demand'].replace(0.0, np.nan, inplace=True)
    df2['demand'].fillna(method='ffill', inplace=True)
```

```
[ ] plt.figure(figsize=(6, 6))
    df2['demand'][0:23].plot()
    plt.title('Actual demand of first 24 hours')
    plt.ylabel('Demand')
    plt.xlabel('date')
    plt.legend(['demand curve', 'test'], loc='upper right')
    plt.show()
```

```
[ ] # Split the dataset into train and test sets
    train_size = int(len(df) * 0.8)
    train, test = df.iloc[:train_size], df.iloc[train_size:]
```

```
[ ] # Select only the numeric columns for scaling
    numeric_cols = ['hour', 'drybulb', 'demand']
    train_numeric = train[numeric_cols]
    test_numeric = test[numeric_cols]
```

```
[ ] # Scale the dataset between 0 and 1
    scaler = MinMaxScaler()
    train_scaled = scaler.fit_transform(train_numeric)
    test_scaled = scaler.transform(test_numeric)
```

```
[ ] # Convert the dataset into input/output pairs
    def to_sequences(dataset, seq_size):
        X = []
        y = []
        for i in range(len(dataset)-seq_size-1):
            window = dataset[i:(i+seq_size), :]
            X.append(window)
            y.append(dataset[i+seq_size, :])
        return np.array(X), np.array(y)
```

```
[ ]  seq_size = 12 # number of time steps to look back
     X_train, y_train = to_sequences(train_scaled, seq_size)
     X_test, y_test = to_sequences(test_scaled, seq_size)
```

```
[ ]  # Define the LSTM model architecture
     model = Sequential()
     model.add(LSTM(units=128, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
     model.add(Dropout(0.2))
     model.add(LSTM(units=128, return_sequences=True))
     model.add(Dropout(0.2))
     model.add(LSTM(units=64, return_sequences=False))
     model.add(Dropout(0.2))
     model.add(Dense(units=3))
```

```
[ ]  from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import LSTM, Dropout, Dense
     from tensorflow.keras.utils import plot_model

     plot_model(model, to_file='model_architecture.png', show_shapes=True, show_layer_names=True)
```

```
[ ]  # Compile the model
     model.compile(optimizer='adam', loss='mse', metrics=['mape','accuracy', 'mse', 'mae'])
```

```
[ ]  # Train the model
     history = model.fit(X_train, y_train, epochs=500, batch_size=32, validation_split=0.2, verbose=2)
```

```
[ ]  # list all data in history
     print(history.history.keys())
     # summarize history for accuracy
     plt.plot(history.history['accuracy'])
     plt.plot(history.history['val_accuracy'])
     plt.title('model accuracy')
     plt.ylabel('accuracy')
     plt.xlabel('epoch')
     plt.legend(['train', 'test'], loc='upper left')
     plt.show()
     # summarize history for loss
     plt.plot(history.history['loss'])
     plt.plot(history.history['val_loss'])
     plt.title('model loss')
     plt.ylabel('loss')
     plt.xlabel('epoch')
     plt.legend(['train', 'test'], loc='upper left')
     plt.show()
```

```
dict_keys(['loss', 'mape', 'accuracy', 'mse', 'mae', 'val_loss', 'val_mape', 'val_accuracy', 'val_mse', 'val_mae'])
```

```
[ ]  # Evaluate the model on the test set
     history2 = model.evaluate(X_test, y_test, verbose=2)
     #print("Test MSE:", mse)
     #print("Test MAPE:", mape)
```

```
4/4 - 0s - loss: 0.0013 - mape: 521280.7500 - accuracy: 0.9286 - mse: 0.0013 - mae: 0.0267 - 59ms/epoch - 15ms/step
```

```
[ ]  # Plot the loss during training
     import matplotlib.pyplot as plt
     plt.plot(history.history['loss'], label='Training Loss')
     plt.plot(history.history['val_loss'], label='Validation Loss')
     plt.title('Model Loss')
     plt.xlabel('Epochs')
     plt.ylabel('Loss')
     plt.legend()
     plt.show()
```

```
# Predict the electricity load on the test set
y_pred = model.predict(X_test)
```

```
4/4 [==============================] - 1s 4ms/step
```

```
# Inverse transform the predictions and actual values
y_pred_inv = scaler.inverse_transform(y_pred)
y_test_inv = scaler.inverse_transform(y_test)
```

```
#Plot the actual and predicted electricity load on the test set
plt.plot(y_test_inv[:, 2], label='Actual')
plt.plot(y_pred_inv[:, 2], label='Predicted')
plt.title('Electricity Load Prediction')
plt.xlabel('Time')
plt.ylabel('Electricity Load')
plt.legend()
plt.show()
```

```
MAPE=0
x=(1/100)
for i in range(90):
   MAPE=MAPE+ np.abs((y_test_inv[:, 2][i]-y_pred_inv[:, 2][i])/y_test_inv[:, 2][i])

MAPE=MAPE/100
print('MAPE')
print(MAPE)
```

```
MAPE
0.026357546331019718
```

Short-term Load forecasting using 1-D CNN based LSTM

```
# Load the dataset
df = pd.read_csv("/content/drive/MyDrive/electricity.csv", index_col=0, parse_dates=True)
```

```
# Remove any rows with missing values
df.dropna(inplace=True)
```

```
# Replace any invalid values with a valid value
df = df.replace([np.inf, -np.inf], np.nan)
df.fillna(0, inplace=True)
```

```
# Split the dataset into train and test sets
train_size = int(len(df) * 0.8)
train, test = df.iloc[:train_size], df.iloc[train_size:]
```

```
# Select only the numeric columns for scaling
numeric_cols = ['hour', 'drybulb', 'demand']
train_numeric = train[numeric_cols]
test_numeric = test[numeric_cols]
train_numeric2 = train['demand']
```

```
[ ]  # Scale the dataset between 0 and 1
     scaler = MinMaxScaler()
     train_scaled = scaler.fit_transform(train_numeric)
     test_scaled = scaler.transform(test_numeric)
```

```
[ ]  df.head(10)
```

```
[ ]  # Convert the dataset into input/output pairs
     def to_sequences(dataset, seq_size):
         X = []
         y = []
         for i in range(len(dataset)-seq_size-1):
             window = dataset[i:(i+seq_size), :]
             X.append(window)
             y.append(dataset[i+seq_size, :])
         return np.array(X), np.array(y)
```

```
[ ]  seq_size = 12 # number of time steps to look back
     X_train, y_train = to_sequences(train_scaled, seq_size)
     X_test, y_test = to_sequences(test_scaled, seq_size)
```

```
[ ]  # Plot the demand curve on the train set
     plt.figure(figsize=(10, 6))
     #plt.plot(train_numeric2[150:168,], label='Actual Demand')
     df['demand'][0:23].plot()
     plt.title('Actual demand of first 24 hours')
     plt.ylabel('Demand')
     plt.xlabel('date')
     plt.legend(['demand curve', 'test'], loc='upper right')
     plt.show()
```

```
# Define the CNN-LSTM model architecture
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(units=128, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=128, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=64, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=3))
```

```
[ ]  model.summary()
```

```
[ ]  # Compile the model
     model.compile(optimizer='adam', loss='mse', metrics=['mape','accuracy', 'mse', 'mae'])
```

```
[ ]  # Train the model
     history = model.fit(X_train, y_train, epochs=500, batch_size=32, validation_split=0.2)
```

```
[ ]  # list all data in history
     print(history.history.keys())
     # summarize history for accuracy
     plt.plot(history.history['accuracy'])
     plt.plot(history.history['val_accuracy'])
     plt.title('model accuracy')
     plt.ylabel('accuracy')
     plt.xlabel('epoch')
     plt.legend(['train', 'test'], loc='upper left')
     plt.show()
     # summarize history for loss
     plt.plot(history.history['loss'])
     plt.plot(history.history['val_loss'])
     plt.title('model loss')
     plt.ylabel('loss')
     plt.xlabel('epoch')
     plt.legend(['train', 'test'], loc='upper left')
     plt.show()
```

```
dict_keys(['loss', 'mape', 'accuracy', 'mse', 'mae', 'val_loss', 'val_mape', 'val_accuracy', 'val_mse', 'val_mae'])
```

```
[ ]  # Predict the electricity load on the test set
     y_pred = model.predict(X_test)
```

```
4/4 [==============================] - 1s 8ms/step
```

```
[ ]  # Inverse transform the predictions and actual values
     y_pred_inv = scaler.inverse_transform(y_pred)
     y_test_inv = scaler.inverse_transform(y_test)
```

```
[ ]  # Plot the demand curve on the test set
     plt.figure(figsize=(10, 6))
     plt.plot(y_test_inv[:, 2], label='Actual Demand')
     plt.plot(y_pred_inv[:, 2], label='Predicted Demand')
     plt.title('Demand Curve')
     plt.xlabel('Time Steps')
     plt.ylabel('Demand (MW)')
     plt.legend()
     plt.show()
```

```
[ ]  # Plot the original and predicted load on the test set
     plt.plot(y_test_inv[:,2], label='Actual')
     plt.plot(y_pred_inv[:,2], label='Predicted')
     plt.title('Electricity Load Prediction')
     plt.xlabel('Test Samples')
     plt.ylabel('Electricity Load')
     plt.legend()
     plt.show()
```

```
[ ]  MAPE=0
     # x=(1/100)
     for i in range(90):
        MAPE=MAPE+ np.abs((y_test_inv[:, 2][i]-y_pred_inv[:, 2][i])/y_test_inv[:, 2][i])

     MAPE=MAPE/100
     print('MAPE')
     print(MAPE)
```

```
MAPE
0.036806974060170916
```