Solar PV Power Forecasting Using Machine Learning

Tesi di Laurea Magistrale in
Electrical Engineering - Ingegneria Elettrica

Author: **Saloni Dhingra**

Student ID: 994054
Advisor: Prof. Giancarlo Storti Gajani
Academic Year: 2022-23

Abstract

Solar photovoltaic (PV) power forecasting is a crucial aspect of efficient energy management in the renewable energy sector. This thesis explores the application of various types of artificial neural networks (ANNs) for predicting PV power output by considering various variables that affect the power output. The proposed ANNs are used to forecast the output power for different PV technologies while considering different prediction horizons. Additionally, the impact of panel ageing is investigated using different machine learning models. To evaluate the performance of the proposed ANNs, real-world PV power data was collected and preprocessed. The preprocessed data was then used to train and test different ANNs, including recurrent neural networks, autoencoders and convolutional neural networks. The experimental results show that the proposed ANNs can accurately predict PV power output, with LSTM demonstrating the best performance for short-term forecasting. Furthermore, the impact of panel ageing on PV power was analyzed using different machine learning models, including linear regression and predictive analysis. The results show that the machine learning models can effectively predict the degradation of PV panel performance over time.

To improve the accuracy of predictions, the effects of splitting the dataset into two distinct datasets - sunny and cloudy - is investigated. Furthermore, a separate prediction model is utilized for each of these datasets. The results indicate that clustering the dataset leads to improved prediction accuracy. Overall, this thesis provides a comprehensive analysis of the application of different ANNs for solar PV power forecasting and the impact of panel ageing on PV power output. The results demonstrate the potential of using machine learning techniques for accurate and reliable solar PV power forecasting.

Keywords: Artificial Neural Networks, Hyperparameters, Long-Short Term Memory, Panel Ageing, Dataset Clustering

Abstract in lingua italiana

La previsione dell'energia solare fotovoltaica (PV) è un aspetto cruciale per una gestione efficiente dell'energia nel settore delle energie rinnovabili. Questa tesi esplora l'applicazione di vari tipi di reti neurali artificiali (ANN) per prevedere l'output dell'energia solare PV, considerando diverse variabili che influenzano l'output energetico. Le ANN proposte vengono utilizzate per prevedere l'output energetico per diverse tecnologie PV, considerando diversi orizzonti di previsione. Inoltre, viene analizzato l'impatto dell'invecchiamento dei pannelli utilizzando diversi modelli di apprendimento automatico. Per valutare le prestazioni delle ANN proposte, sono stati raccolti e preelaborati dati reali sull'energia solare PV. I dati preelaborati sono stati quindi utilizzati per addestrare e testare diverse ANN, tra cui reti neurali feedforward, reti neurali ricorrenti e reti neurali convoluzionali. I risultati sperimentali mostrano che le ANN proposte possono prevedere con precisione l'output dell'energia solare PV, con il modello LSTM che offre le migliori prestazioni per la previsione a breve termine. Inoltre, l'impatto dell'invecchiamento dei pannelli sull'output dell'energia solare PV è stato analizzato utilizzando diversi modelli di apprendimento automatico, tra cui la regressione lineare e l'analisi predittiva. I risultati mostrano che i modelli di apprendimento automatico possono prevedere in modo efficace il degrado delle prestazioni dei pannelli PV nel tempo.

Per migliorare l'accuratezza delle previsioni, viene studiato l'impatto della suddivisione del dataset in due sottodataset: soleggiato e nuvoloso. Inoltre, viene utilizzato un modello di previsione dedicato per ciascun sottodataset. I risultati indicano che la suddivisione del dataset porta a un miglioramento dell'accuratezza delle previsioni. In generale, questa tesi fornisce un'analisi esaustiva dell'applicazione di diverse ANN per la previsione dell'energia solare PV e dell'impatto dell'invecchiamento dei pannelli sull'output dell'energia PV. I risultati dimostrano il potenziale dell'utilizzo delle tecniche di apprendimento automatico per una previsione accurata e affidabile dell'energia solare PV.

Parole chiave: Reti Neurali Artificiali, Iperparametri, Memoria a Lungo-Corto Termine, Invecchiamento dei Pannelli, Clustering del Dataset

Contents

1 | Introduction

1.1. Motivation

Due to mounting concerns regarding the emissions of greenhouse gases and environmental pollution stemming from the excessive utilization of fossil fuel-based energy sources, the significance of renewable energy sources in the field of power generation has escalated [28]. Consequently, it has become imperative to explore alternative energy sources to supplant fossil fuels. By the end of 2019, the cost-effectiveness of generating energy from wind and photovoltaic (PV) installations had surpassed that of conventional fossil fuel-powered plants in some regions. Moreover, in specific regions, the installation of new wind and solar PV facilities has been found to be more cost-effective than the ongoing operation of existing fossil fuel-based power plants [25]. Considering the characteristics of solar energy, such as its clean nature, abundant and free availability, and renewable attributes, there has been a significant increase in the deployment of PV panels for harnessing solar power in recent years [41]. Due to nations' interest in investing in renewable energy sources, it is probable that the installation of PV panels will continue to rise. Figure 1.1 depicts the worldwide capacity of solar photovoltaics from 2011 to 2021.

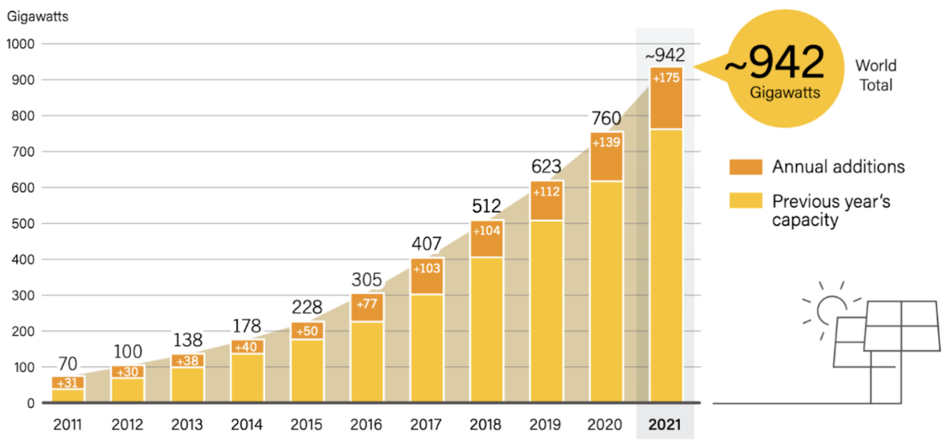


Figure 1.1: Global capacity of Solar PV and annual additions

Solar photovoltaics (PV) is experiencing rapid growth as an energy technology. In terms of solar PV capacity, India ranks among the top ten countries globally. It holds the position of the second-largest market in Asia for new solar PV capacity and the third-largest market worldwide. This growth in the Indian solar PV market is primarily attributed to a strong emphasis on domestic manufacturing. After two years of decline, the country witnessed significant expansion in annual solar installations in 2021, with an additional 13 gigawatts (GW) of capacity installed. This amount is more than double the installations in 2020 and surpasses any previous year, establishing a new record. Figure 1.2 presents a visual representation of the countries with the highest solar PV capacity globally, highlighting the significant position of India in the solar energy landscape [25].

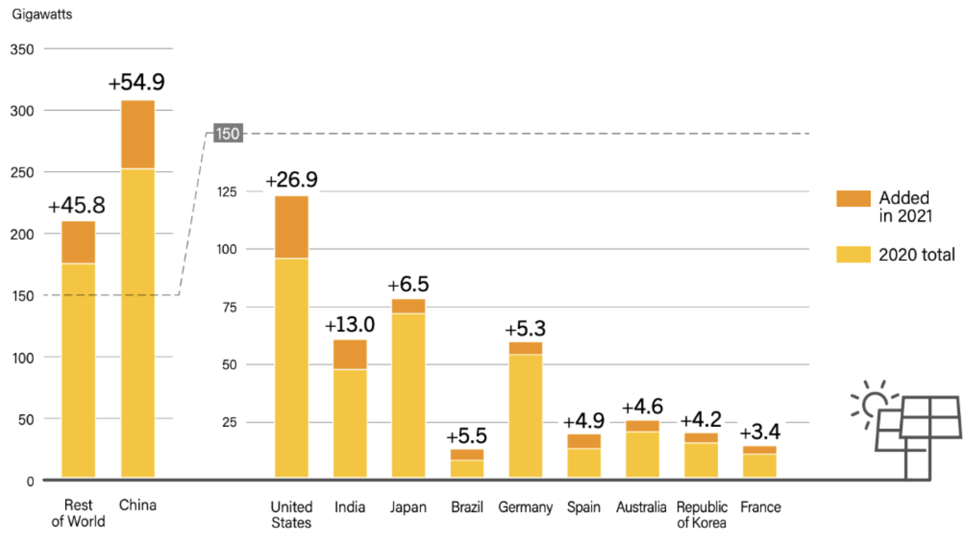


Figure 1.2: Solar PV capacity of differnt countries

Photovoltaic (PV) installations are a critical component in microgrid systems. A microgrid is a localized energy grid that can operate independently from the main power grid. It is typically composed of various energy sources such as solar panels, wind turbines, and energy storage systems that work together to provide a reliable and stable energy supply to the local community. One advantage of a PV installation in a microgrid system is that it provides a decentralized and clean energy source that can reduce the community's dependence on fossil fuels. This can help to reduce greenhouse gas emissions and improve air quality, which are critical issues in many parts of the world. Figure 1.3 illustrates a schematic representation of a microgrid system as an example. Incorporating solar energy into microgrids comes with its own set of challenges. The main issue is that solar power is not constant and can vary depending on the weather conditions. Factors like sunlight intensity, temperature, and humidity affect how much electricity can be generated from solar panels. As a result, the power output from solar panels can fluctuate due to these variables [9].

Figure 1.4 demonstrates the variability in power output within a PV array under diverse conditions, thereby posing a potential threat to the reliability and stability of the power system.

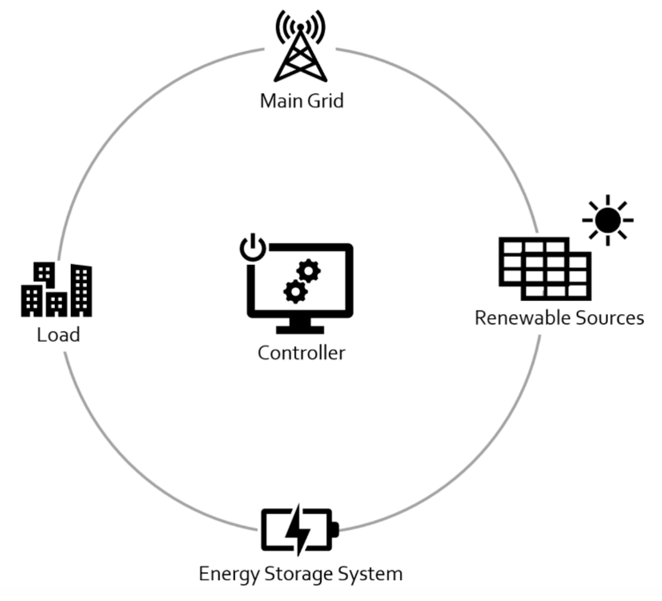


Figure 1.3: A microgrid system example

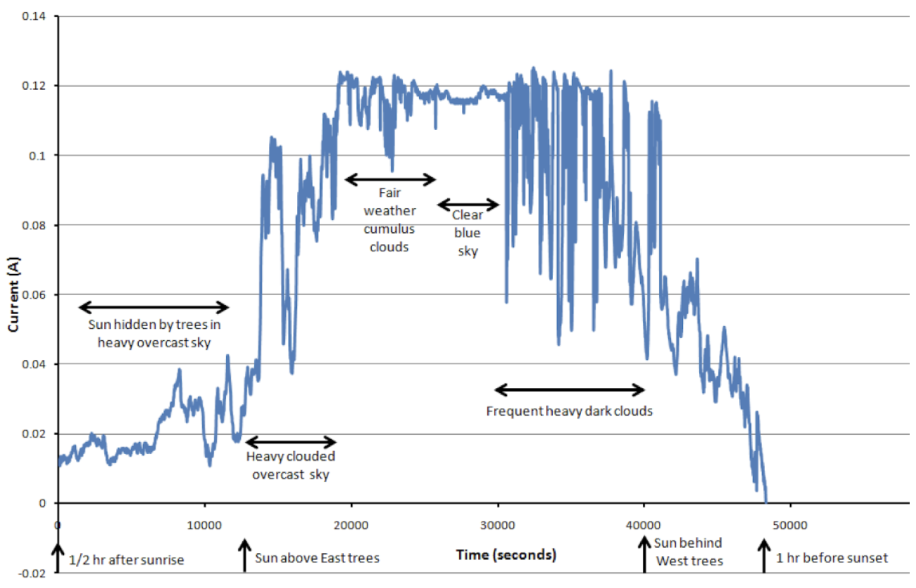


Figure 1.4: Output Power variation in a PV array in different conditions

The increasing proliferation of PV installations in microgrid systems highlights the growing need for precise solar prediction. However, forecasting solar energy poses significant challenges due to the inherent variability of meteorological factors that influence the PV output power. Particularly, solar prediction becomes more challenging as the prediction horizon extends further into the future, specifying the time span for which the target variable is to be anticipated.

Given the escalating importance of accurate solar prediction, the field of solar forecasting has garnered considerable attention as a promising area of research. Consequently, numerous studies and research endeavors have been conducted in recent years to enhance the precision of solar predictions in order to meet the required standards. Accurate prediction of photovoltaic (PV) power output is important for optimizing the performance of solar power systems [37]. Here are some tools that can be used to achieve this:

1. Solar irradiance models: These models use meteorological data to predict the amount of solar irradiance that will be received at a particular location and time. Some commonly used models include the Clear Sky Model and the Linke Turbidity Model.

2. Machine learning algorithms: Machine learning algorithms can be trained on historical data to predict PV power output. Some popular algorithms include Artificial Neural Networks (ANNs), Support Vector Machines (SVMs), and Random Forest Regression (RFR).

3. Physical models: Physical models can be used to simulate the behavior of a solar panel and predict its power output. These models take into account factors such as temperature, shading, and panel orientation to make predictions.

4. Hybrid models: Hybrid models combine the strengths of different types of models to achieve more accurate predictions. For example, a hybrid model may use a machine learning algorithm to make predictions based on historical data, and then use a physical model to adjust these predictions based on real-time conditions.

5. Monitoring systems: Monitoring systems can be installed on solar power systems to collect real-time data on power output, temperature, and other variables. This data can then be used to refine predictions and ensure that the system is operating at peak performance.

In addition to these tools, it is also important to consider other factors that can affect PV power output, such as the age of the panel, its efficiency, and the level of maintenance it has received.

1.2. Thesis Objectives

The primary objective of this thesis is to facilitate the advancement of an algorithm capable of accurately forecasting power generation derived from a solar photovoltaic (PV) system. The entire world is being compelled to focus on the usage of renewable energy sources in order to minimize power outages, reduce carbon dioxide emissions, and maintain pollution control. As a result, there is a steady growth in interest and development in solar energy. Forecasting photovoltaic power is critical for reliable and cost-effective grid operation. Furthermore, photovoltaic (PV) power forecasting is required for the installation of big PV producing plants. Accurate PV power prediction can be challenging due to several meteorological characteristics such as temperature, cloud quantity, and dust. The use of machine learning models has increased the forecasting accuracy of solar energy models that use historical data dramatically. Because a machine learning model is often data-dependent, different types (e.g., input variables) and resolutions (e.g., time periods) of data can be evaluated to see how well they predict the desired output. Nonetheless, the impacts of topographical changes at PV plant locations and fluctuations in meteorological conditions on PV power output are poorly understood, particularly when searching for possible sites for solar PV systems over large areas. A smart forecasting approach could immediately anticipate the PV output power based on some prior information or easily accessible data.

1.3. Thesis Structure

Chapter 2 provides a comprehensive review of the existing literature on solar forecasting techniques. The research investigates diverse methodologies encompassing statistical approaches, sky imagers, satellite imaging, and numerical weather prediction systems. The chapter also discusses different prediction horizons and prediction modes, and appropriate values are selected based on the specific application requirements. Subsequently, a meticulous evaluation of various techniques and architectures tailored for time series forecasting, such as Long Short-term Memory (LSTM), is conducted to identify a suitable forecasting approach.

Chapter 3 presents a detailed description of the data and tools utilized in this study.

Chapter 4 is dedicated to data exploration and preprocessing, which are crucial steps in the research. The chapter also introduces the performance metrics employed in this study to evaluate the prediction models. Hyperparameter tuning is conducted to identify the optimal set of hyperparameters for the models. The obtained results from different model architectures are then compared and analyzed. Furthermore, an analysis of PV system aging is conducted, providing valuable insights into the degradation of prediction accuracy over time.

Moreover, the chapter investigates the potential of splitting the dataset into sunny and overcast days and explores the use of multiple prediction models to enhance the accuracy of solar predictions. All the obtained results, along with their interpretations, are presented comprehensively in this chapter.

Finally, in Chapter 5, the thesis study concludes by summarizing the key findings and outcomes of the study. Additionally, potential areas for improvement and future research directions are proposed, offering avenues for further advancement in this field.

2 | Literature Review

This chapter is dedicated to conducting a comprehensive literature review and studying existing solar forecasting methods. It also entails a meticulous examination of the distinct characteristics associated with each technique under consideration.

2.1. Solar Power Forecasting Methods

The existing methods for solar power prediction are categorized into different groups based on their prediction horizons. These categories include statistical methods, sky imagers, satellite imaging, and Numerical Weather Prediction (NWP) [41]. All these techniques involve the anticipation of solar irradiance, which is then converted into power, with the exception of statistical approaches that may directly predict power without the need for conversion. Figure 2.1 provides an overview of the various approaches and their corresponding prediction horizons.

2.1.1. Statistical Methods

Artificial neural networks (ANNs), regression models, support vector machines (SVMs), and Markov chains represent statistical methodologies commonly utilized for solar power forecasting in the research field [41]. These techniques leverage historical observations to facilitate the training of models, enabling the computation of predictions by analyzing the past values of input variables [42]. Consequently, depending on the input variables used, the power output can be predicted directly or indirectly. In the indirect approach, solar irradiance is forecast, and the result is then converted into power. The different types of statistical methods used for solar power forecasting are as follows:

2.1.1.1. Time Series based Forecasting Techniques

Time series data offers valuable statistical insights that can be harnessed for predicting the characteristics and patterns of a quantifiable entity. These observations are typically gathered over time at regular intervals, which may span quarterly, monthly, weekly, daily,

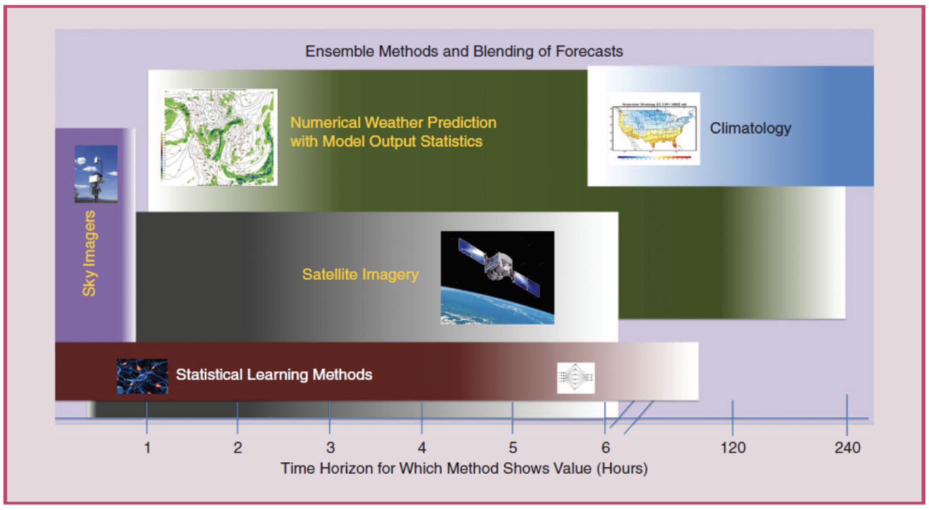

Ensemble Methods and Blending of Forecasts
Numerical Weather Prediction with Model Output Statistics
Climatology
Sky Imagers
Satellite Imagery
Statistical Learning Methods
1
2
3
4
5
6
120
240
Time Horizon for Which Method Shows Value (Hours)

Figure 2.1: Different PV Power forecasting methods based on prediction horizon

or even finer timeframes, contingent upon the temporal dynamics of the specific variable under consideration [1]. Several time series-based forecasting techniques include:

- **Exponential Smoothing:** The exponential smoothing approach, also known as the exponentially weighted moving average (EWMA), is a statistical technique used for forecasting based on historical time series data. This method assigns varying weights to past observations, with more weight given to recent data points and gradually diminishing weights for older observations. Despite its simplicity, the method is capable of learning and making decisions based on underlying assumptions. It was initially developed by Brown in 1956 [34] and has since found numerous applications. The approach was further developed by Holt in 1957 and Winter in 1960, resulting in the well-known Holt-Winter's approach [40]. The governing equation for exponential smoothing is as follows:

$$\hat{Y}_{t+1} = \alpha Y_t + (1-\alpha)\hat{Y}_t = \hat{Y}_t + \alpha(Y_t - \hat{Y}_t) \tag{2.1}$$

  In Equation (2.1), Y_t represents the current observation, $\hat{Y}_t$ is the predicted value, and α is the smoothing constant, which ranges between 0 and 1. The forecasting equation calculates the predicted value at time $t+1$ as the sum of the last predicted value $\hat{Y}_t$ and the forecast adjustment factor $\alpha(Y_t - \hat{Y}_t)$.

- **Autoregressive Moving Average Model (ARMA):** ARMA (Autoregressive Moving Average) is a widely used statistical analysis method for time series forecasting. It has been extensively studied and applied in various applications, including solar irradiance and wind speed forecasting, consistently demonstrating high prediction accuracy. The ARMA model incorporates two polynomials, AR (Autoregressive) and MA (Moving Average), to predict PV output based on historical data [12]. The mathematical expression for the ARMA model is as follows [1]:

$$X(t) = \sum_{i=1}^{p} \alpha_i X(t-i) + \sum_{j=1}^{q} \beta_j e(t-j) \tag{2.2}$$

In Equation (2.2), X(t) represents the expected PV output, which is a combination of the AR and MA functions. The parameters p and q denote the order or number of processes in the model, while α_i and β_j are the coefficients for the AR and MA components, respectively. The term e(t) represents white noise, generated randomly and unrelated to the model's predictions. ARMA (Autoregressive Moving Average) models exhibit significant flexibility in capturing diverse patterns within time series data, as they can be configured with different orders.

- **Autoregressive Integrated Moving Average Model (ARIMA):** The Autoregressive Integrated Moving Average (ARIMA) model is a widely-used method for analyzing and forecasting time series data. It combines three components: autoregressive (AR), integrated (I), and moving average (MA) terms [21].

  (i) Autoregressive (AR) term: This term signifies that the future values of a time series depend on its past values. In other words, the current value of a time series can be predicted based on its previous values.

  (ii) Moving Average (MA) term: This term indicates that the future values of a time series are influenced by the errors or residuals of past observations. In other words, the current value of a time series can be predicted based on the errors made in predicting its previous values.

  (iii) Integrated (I) term: This term refers to the requirement of differencing to transform a time series into a stationary one, where the mean and variance remain constant over time.

  ARIMA models are defined by three parameters: p, d, and q. The parameter p represents the order of the autoregressive term, d represents the order of differencing needed to achieve stationarity, and q represents the order of the moving average term. Mathematically, an ARIMA(p, d, q) model is represented as:

$$y(t) = c + \phi(1)y(t-1) + \cdots + \theta(1)e(t-1) + \cdots + \theta(q)e(t-q) + e(t) \tag{2.3}$$

Where y(t) is the value of the time series at time t; c is a constant term or intercept; $\phi(1), \ldots, \phi(p)$ are the coefficients of the autoregressive terms of the model, which represent the effect of the past values of the time series on its current value; $\theta(1), \ldots, \theta(q)$ are the coefficients of the moving average terms of the model, which represent the effect of the past errors or residuals on the current value of the time series; e(t) is the error or residual term at time t, which represents the difference between the predicted value and the actual value of the time series at that time. d is the order of differencing required to make the time series stationary.

2.1.1.2. Machine Learning based Forecasting - Artificial Neural Networks

Machine learning techniques can be used to forecast solar PV power output based on weather data, historical power output data, and other relevant factors.

An artificial neural network (ANN) is a computational model inspired by the structure and functioning of biological neural networks in the human brain. It is a powerful machine learning technique used for various tasks such as pattern recognition, classification, regression, and decision-making [5]. The basic unit of computation in an ANN is a neuron, which receives input from other neurons or external sources. Each neuron performs a simple calculation on its inputs and produces an output. This calculation involves applying a weighted sum of the inputs and passing the result through an activation function. The weights associated with the inputs determine the strength of the connections between neurons and play a crucial role in learning and information processing. Figure 2.2 shows the basic architecture of an artificial neural network.

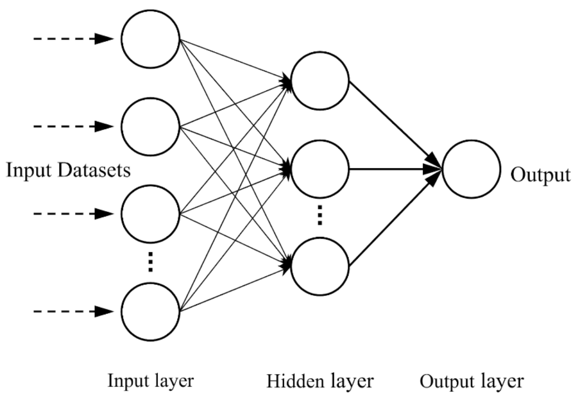


Figure 2.2: Basic architecture of an Artificial Neural Network

Neurons are organized into layers in an ANN. The most common arrangement is a layered architecture consisting of an input layer, one or more hidden layers, and an output layer. The input layer receives the initial input data, while the output layer produces the final output of the network. The hidden layers are intermediary layers between the input and output layers, responsible for extracting and transforming features from the input data. Some of the commonly used network topologies in ANNs for forecasting include:

- **Multi-layer Perceptron Neural Network (MLPNN):** Several studies use the MLPNN model as a reference [7], [6]. It is a method for designing and predicting using a simple and effective ANN approach. It is so powerful that it is employed in universal approximation, nonlinear modeling, and complex problems that an ordinary single layer neural network cannot handle [32],[29],[27].Multi-Layer Perceptron (MLP) is a type of artificial neural network that typically consists of three or more layers of interconnected nodes. These nodes in each layer are connected to nodes in the subsequent layer through specific weights.

  The number of nodes in the hidden layer plays a crucial role in the MLP's capacity to capture complex nonlinear functions. Research conducted by Hegazy et al. in [20] has demonstrated that a single hidden layer with a sufficient number of hidden nodes is capable of effectively representing complex nonlinear functions. However, as the number of nodes in a Multi-Layer Perceptron Neural Network (MLPNN) increases, challenges such as overfitting and training issues can arise [8]. Figure 2.3 illustrates a simplified architecture of an MLPNN.

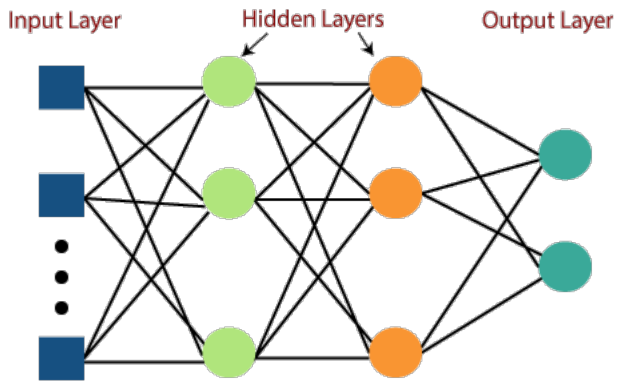


Figure 2.3: Multi-layer perceptron neural network

- **Recurrent Neural Network (RNN):** A Recurrent Neural Network (RNN) is a form of neural network that uses a feedback loop to handle sequential or time-series input. Unlike classic neural networks, which accept fixed-size inputs, RNNs can handle variable-length input sequences [2]. The feedback loop enables RNNs to store data and capture its temporal dynamics. Each time step's output is determined not just by the input at that time step, but also by the output of the preceding time step. This means that RNNs can represent the context of the input sequence, which is very useful for natural language processing, speech recognition, and music production.

An RNN's design generally comprises a hidden state that is changed at each time step based on the input and the preceding hidden state. The output at that time step is then generated using this concealed state. The hidden state is a learnt representation of prior inputs that is carried over to the next time step.

Several RNN variants, such as Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU), have been created to overcome the problem of vanishing gradients, which can occur during regular RNN training. Additional gates govern the flow of information in the network, allowing for higher memory retention and more steady training. Figure 2.4 shows the basic architecture of a Recurrent Neural Network.

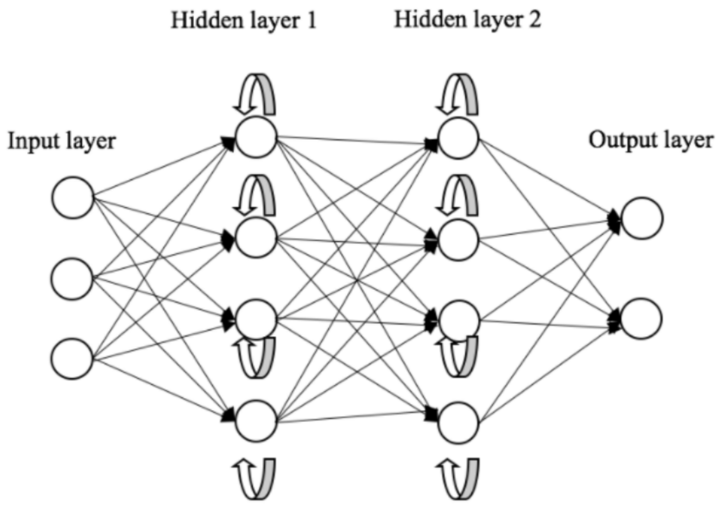


Figure 2.4: Recurrent Neural Network

- **Extreme Learning Machine (ELM):** The extreme learning machine (ELM) is a machine learning method used for a variety of applications, including time series forecasting. ELM is a feedforward neural network with a single hidden layer that is described as a quick and accurate alternative to regular neural networks [17]. In ELM, the input layer is directly linked to the hidden layer, and the weights between these two layers are produced at random. The output layer is then computed by combining the hidden layer outputs in a linear fashion. In contrast to standard neural networks, ELM's weights are not modified during training, and only the output layer weights are computed using a technique known as pseudo-inverse. Basic

ELM structure is shown in Figure 2.5.

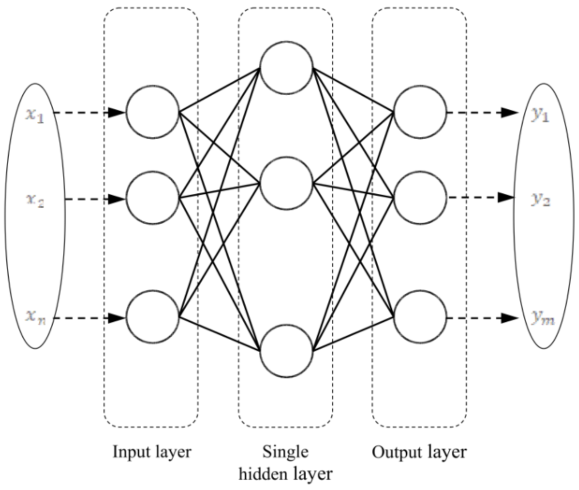


Figure 2.5: Basic Extreme Learning Machine Model

For time series forecasting with ELM, the input variables are the past observations of the time series, and the output variable is the forecasted value of the time series at the next time step. The ELM algorithm can learn the underlying patterns and dependencies in the time series and use them to predict the future values. One of the advantages of ELM for time series forecasting is its fast training speed, which is due to the use of random weights and the pseudo-inverse method. This makes ELM suitable for handling large datasets and high-dimensional input spaces. However, ELM may not be as accurate as traditional neural networks in some cases, and careful selection of hyperparameters may be necessary to achieve optimal performance.

2.1.1.3. Deep Learning: State-of-the-art in Machine Learning

Deep learning neural networks are a sort of machine learning model that uses interconnected nodes (neurons) that can learn from data to imitate how the human brain operates. These neural networks include numerous layers that can extract increasingly complex aspects from input data, making them particularly well-suited for tasks like picture and audio recognition, forecasting, natural language processing, and even game play [19]. Deep learning neural networks function by taking in input data (such as a picture or a text) and sending it through a sequence of layers, with each layer non-linearly altering the input to build a more abstract representation of the data. The output is then used to make a prediction or classification. Basic structure of a DNN is shown in Figure 2.6.

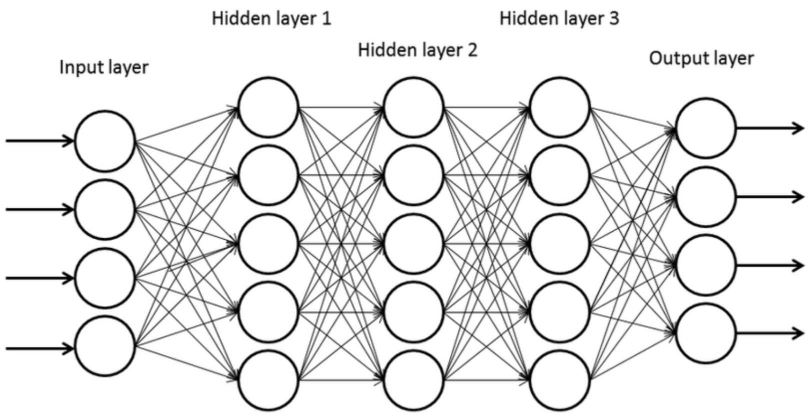


Figure 2.6: Basic Deep learning Neural Network

Deep learning neural networks require a vast amount of data to be trained accurately, and the training process involves revising the neurons' weights and biases depending on the difference between the expected and real output [15]. This is usually accomplished through the use of a technique known as backpropagation, which computes the gradient of the error with respect to each weight and bias in the network. There are many different types of deep learning neural networks, but some of the most commonly used architectures include:

- **Convolutional Neural Networks (CNNs):** A convolutional neural network (CNN) is a type of deep learning algorithm that is commonly used for image recognition and classification tasks. The key feature of CNNs is the use of convolutional layers, which apply filters to the input image to extract features that are then used for classification [3]. The basic overview of the CNN architecture is:

  Input Layer: The input layer receives the raw image data, which is typically represented as a matrix of pixel values.

  Convolutional Layer: The convolutional layer applies a set of filters to the input image to extract features. Each filter is a small matrix of weights, which is applied to a small portion of the input image at a time (a sliding window). The output of the convolutional layer is a set of feature maps, where each map represents the response of a specific filter. The output feature map is calculated as follows:

$$F_{i,j} = \sum_{m} \sum_{n} I_{i+m,j+n} \times K_{m,n} \tag{2.4}$$

  where $F_{i,j}$ is the output value at position (i, j) in the feature map, I is the input image, K is the filter matrix, and m and n are the indices of the filter matrix.

ReLU Layer: The ReLU (rectified linear unit) layer applies the element-wise activation function $f(x) = \max(0, x)$ to the output of the convolutional layer. This introduces non-linearity into the model and helps to improve its performance.

Pooling Layer: The pooling layer reduces the dimensionality of the feature maps by down-sampling them. The most common pooling operation is max-pooling, which takes the maximum value within a sliding window. This operation is applied separately to each feature map.

Fully Connected Layer: The fully connected layer takes the output of the previous layer and applies a linear transformation to produce the final output. This layer is similar to the one in a standard neural network, except that it receives a set of feature maps instead of a vector. The output of the fully connected layer is calculated as follows$y = Wx+b$ where x is the input vector, W is the weight matrix, b is the bias vector, and y is the output vector.

Softmax Layer: The softmax layer applies the softmax function to the output of the fully connected layer to produce a probability distribution over the classes. This function normalizes the output vector so that the values sum up to 1.

The softmax function is defined as:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{2.5}$$

where $\sigma(z)_j$ is the output probability for class j, z is the input vector, and the sum is taken over all classes.

Output Layer: The output layer produces the final prediction by selecting the class with the highest probability from the softmax distribution. This is a basic overview of the CNN architecture and its mathematical equations. In practice, there are many variations and extensions of this model, including different types of layers, regularization techniques, and optimization methods.

- **Recurrent Neural Networks (RNNs):** RNNs are frequently used to process sequential data, such as voice or text. They are designed to keep a stored memory of prior inputs, allowing them to handle data sequences having a temporal component. There are several types of Recurrent Neural Networks (RNNs) that differ in how they process and utilize the sequential data like Vanilla, LSTM, Gated Recurrent Unit, Bidirectional RNNs, etc. RNNs are useful for a wide range of applications, including language modeling, forecasting, speech recognition, and machine translation [2]. A Long Short-Term Memory (LSTM) network and Gated Recurrent Unit (GRU) are discussed in detail in this thesis.

Gated Recurrent Unit (GRU) is a type of Recurrent Neural Network (RNN) that is commonly used for time series forecasting tasks. The GRU is designed to help overcome the vanishing gradient problem that is commonly encountered in traditional RNNs [4]. The vanishing gradient problem occurs when the gradients in the backpropagation process become very small, making it difficult to update the network weights. The GRU accomplishes this by introducing gating mechanisms that allow the network to selectively update and forget information [10]. The gating mechanisms consist of two gates: the reset gate and the update gate.

The reset gate determines how much of the previous hidden state should be forgotten, while the update gate determines how much of the new input should be incorporated into the current hidden state [30]. These gates are controlled by sigmoid functions that output values between 0 and 1.

Mathematically, the update gate is defined as follows:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{2.6}$$

where z_t is the update gate at time t, σ is the sigmoid function, W_z, U_z, and b_z are the weight matrices and bias vector associated with the update gate, x_t is the input at time t, and h_{t-1} is the hidden state from the previous time step. The reset gate is defined similarly:

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{2.7}$$

where r_t is the reset gate at time t, W_r, U_r, and b_r are the weight matrices and bias vector associated with the reset gate. Using the reset gate, we can determine how much of the previous hidden state should be forgotten:

$$h'_t = tanh(W_h x_t + r_t * U_h h_{t-1} + b_h) \tag{2.8}$$

where h'_t is the new candidate hidden state at time t, tanh is the hyperbolic tangent activation function, W_h, U_h, and b_h are the weight matrices and bias vector associated with the candidate hidden state. Finally, we use the update gate to determine how much of the new candidate hidden state should be incorporated into the current hidden state:

$$h_t = (1 - z_t) * h_{t-1} + z_t * h'_t \tag{2.9}$$

where h_t is the new hidden state at time t, and h_{t-1} is the previous hidden state.

Long Short-term Memory (LSTM) is a type of recurrent neural network (RNN) that is designed to deal with the vanishing gradient problem often encountered in traditional RNNs [22].

In an LSTM network, there are three main components: the input gate, the forget gate, and the output gate [16]. Each of these gates has its own set of weights and biases, which are learned during the training process. The input and output gates control the flow of information into and out of the LSTM cell, while the forget gate determines which information to discard from the cell state [13]. Figure 2.7 shows basic architecture of an LSTM unit.

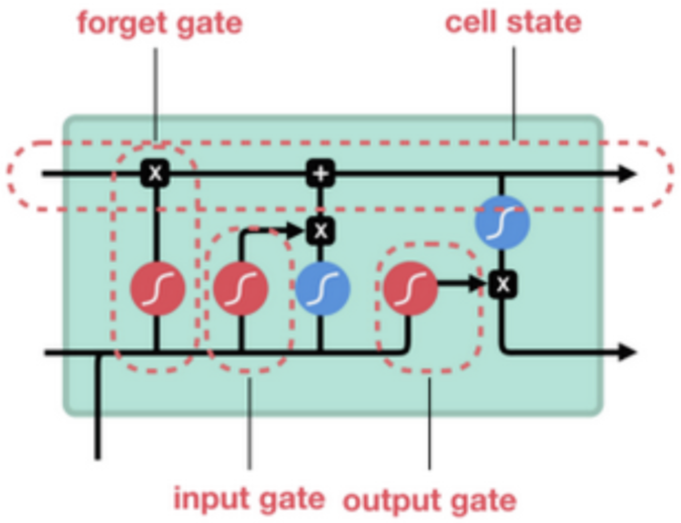


Figure 2.7: LSTM unit architecture

The LSTM cell updates its internal state based on the input, the previous cell state, and the output from the previous time step. The internal state of the cell is also known as the hidden state. The cell state is updated as follows:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{2.10}$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{2.11}$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.12}$$

$$g_t = tanh(W_g[h_{t-1}, x_t] + b_g) \tag{2.13}$$

$$c_t = f_t * c_{t-1} + i_t * g_t \tag{2.14}$$

$$h_t = o_t * tanh(c_t) \tag{2.15}$$

where x_t is the input at time step t; h_t is the hidden state at time step t; c_t is the cell state at time step t; f_t, i_t, and o_t are the forget, input, and output gates at time step t, respectively; g_t is the candidate cell state at time step t; σ is the sigmoid activation function which represents element-wise multiplication; tanh is the hyperbolic tangent activation function; W_f, W_i, W_o, W_g are weight matrices for the forget, input, output, and candidate gate, respectively; b_f, b_i, b_o, b_g are bias terms for the forget, input, output, and candidate gate, respectively

The output of the LSTM cell at time step t is the hidden state, h_t. This output can be fed into another layer for further processing or used directly as the forecasted value. LSTM is a powerful tool for time series forecasting tasks, especially when dealing with long-term dependencies. Its internal architecture allows it to selectively retain or discard information from the input, ensuring that relevant information is propagated through the network while irrelevant information is filtered out [38].

- **Autoencoders** are a type of neural network commonly used for unsupervised learning tasks, such as dimensionality reduction and feature extraction. In the context of time series forecasting, autoencoders can be used to learn a compressed representation of the time series data, which can then be used to make predictions.

  The basics of autoencoders for time series forecasting along with some mathematical equations are:

  Encoding: The first step in an autoencoder is to encode the input time series data into a lower-dimensional representation. This is done using an encoder function, which maps the input time series x(t) to a lower-dimensional representation z(t) using a set of learned parameters. The encoder function can be written as:

$$z(t) = f(x(t); \theta) \tag{2.16}$$

  where f is the encoder function, θ are the learned parameters, and z(t) is the encoded representation of x(t).

  Decoding: The next step is to decode the encoded representation z(t) back into a reconstruction of the original time series data. This is done using a decoder function, which maps the encoded representation z(t) back to the original time series x(t). The decoder function can be written as:

$$\hat{x}(t) = g(z(t); \phi) \tag{2.17}$$

where g is the decoder function, ϕ is the learned parameter, and $\hat{x}(t)$ is the reconstruction of x(t) from z(t).

Loss function: To train the autoencoder, we need a way to measure how well the reconstructed time series matches the original time series. This is done using a loss function, which measures the difference between the reconstructed time series $\hat{x}(t)$ and the original time series x(t). The loss function can be written as:

$$L(x, \hat{x}) = ||x - \hat{x}||^2 \tag{2.18}$$

where ||.|| represents the Euclidean norm.

Training: The autoencoder is trained by minimizing the loss function L using an optimization algorithm such as stochastic gradient descent. The optimization algorithm adjusts the parameters θ and ϕ to minimize the loss function L.

Forecasting: Once the autoencoder is trained, it can be used for time series forecasting by feeding in a sequence of input time steps and predicting the next time step using the encoded representation. This is done by encoding the input time steps using the encoder function, and then using the encoded representation to make a prediction using a linear or nonlinear regression model.

In summary, autoencoders can be used for time series forecasting by learning a compressed representation of the time series data, which can then be used to make predictions. The basic components of an autoencoder for time series forecasting are the encoder function, the decoder function, the loss function, and the optimization algorithm.

2.1.2. Sky Imagers

Sky imagers are sophisticated digital imaging devices engineered to acquire high-resolution sky images. These images are instrumental in predicting cloud conditions, which in turn affect solar irradiation at the Earth's surface. By analyzing consecutive photos, sky imagers can detect the presence of clouds, estimate their height above the ground, and calculate their motion velocity. Moreover, the detection of cloud shadows allows sky imagers to identify sudden changes in solar irradiance [42].

However, it is important to note that sky imagers have a relatively limited prediction horizon, typically up to 30 minutes. This means their forecasting capability is more suitable for short-term predictions [41].

2.1.3. Satellite Imaging

Geostationary satellites can offer data on cloud conditions and movement. Clouds may be detected and their features assessed using satellite imagery. Irradiance may be predicted up to six hours in advance using the information acquired [42].

2.1.4. Numerical Weather Prediction (NWP)

Based on numerical dynamic atmospheric modeling, the NWP model can estimate solar irradiance. NWP forecasts the status of the atmosphere based on present conditions. NWP is useful for forecasts up to two weeks in advance. When the forecast horizon exceeds 4 hours, NWP techniques outperform satellite imaging predictions in terms of accuracy. However, due to geographical and temporal constraints, most cloud properties remain unresolved, and as a result, NWP approaches are not practically feasible for forecasting timeframes shorter than several hours [42]. The different types of NWP are as follows :

2.1.4.1. Persistence Forecast

The persistence model is a frequently employed method for solar forecasting at very short-term and short-term horizons due to its advantages of low computational cost, minimal time delay, and reasonable accuracy [1]. This forecasting method involves predicting the power output of a solar photovoltaic (PV) system based on historical data of its performance.

The fundamental principle of the persistence model is based on the assumption that the power output remains relatively constant if weather conditions and external factors remain unchanged. The forecasting equation for this model is represented as follows:

$$P(t+k|t) = \frac{1}{T}\sum_{i=0}^{n-1} P(t - i\Delta t) \tag{2.19}$$

In Equation (2.19), the variable k represents the forecast time period, and $P(t + k|t)$ denotes the predicted power at time t + k based on the available information at time t. T denotes the duration of the forecast interval, while n represents the number of historical measurements considered. The term $P(t-i\Delta t)$ corresponds to the actual power measured at time t and at time steps i within the forecast interval T. Additionally, Δt signifies the time difference between consecutive measurements in the time series.

It is important to note that the persistence model is often used as a benchmark for evaluating the performance of other forecasting methods. However, as the time horizon increases, the accuracy of the persistence model decreases significantly, particularly when climate conditions change over time [1].

2.1.4.2. Physical Model

A physical model for forecasting solar PV power utilizes physical equations and models to predict the output of a solar PV system based on known variables such as weather conditions, system design, and relevant parameters [24].

In contrast to persistence forecasting that relies solely on historical data, physical models take into account the fundamental physics and engineering principles that govern the performance of solar PV systems. These models consider the dynamic properties of the atmosphere and can provide forecasts for a time horizon of more than 15 days. They utilize a set of numerical equations to mathematically describe and model the physical conditions and their interactions.

Mathematically, a physical model can be represented as:

$$\frac{\Delta A}{\Delta t} = F(A) \tag{2.20}$$

In Equation (2.20), ΔA denotes the change in the value of the forecasted response at a specific spatial location, Δt represents the change in time or temporal horizon, and $F(A)$ signifies the mathematical function that incorporates the variables influencing the value of A. The physical model aims to capture the relationships and interactions among these variables in order to predict the changes in the forecasted response over time.

2.2. Prediction Method Selection

Prediction method selection refers to the process of choosing the most appropriate algorithm for a given prediction problem. The choice of the prediction method depends on several factors, including the nature of the problem, the type of data, the desired prediction accuracy, prediction horizon and computational resources available.

2.2.1. Prediction Horizon Selection

The selection of an appropriate forecasting approach depends on the specified prediction horizon, which refers to the time period into the future for which the PV output power is being forecasted. To better understand the concept of the prediction horizon and its relationship with the window length, Figure 2.8 provides an illustration.

Based on the forecast horizon, solar power generation can be categorized into four groups, as described in previous studies [42],[11].

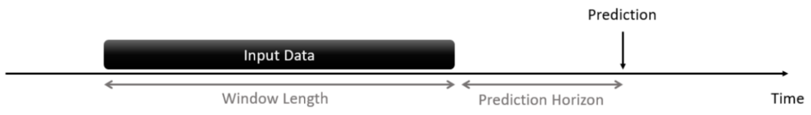


Figure 2.8: Window Length and Prediction horizon

Very short-term Prediction Forecasting: This category involves predicting PV output within a few seconds to minutes. It is suitable for applications such as PV and storage control, as well as anticipating sudden perturbations in power generation.

Short-term Prediction: For short-term prediction, the forecast horizon typically spans up to 72 hours into the future. This type of forecasting is useful for tasks like unit commitment, scheduling, electrical power dispatching, and other operational purposes.

Medium-term Prediction: Medium-term prediction allows forecasting up to a week in advance. This horizon is valuable for maintenance planning and scheduling activities.

Long-term Prediction: Long-term prediction involves forecasting up to a year ahead. It is particularly useful for generation expansion planning and making informed decisions regarding future energy infrastructure development.

Figure 2.9 provides an overview of the applications of different prediction horizons, showcasing the suitability of statistical methods, especially Artificial Neural Networks (ANNs), for solar forecasting tasks.

2.2.2. Prediction Mode Selection

Solar photovoltaic (PV) forecasting can be achieved through two main methods: indirect and direct forecasting. Indirect forecasting involves predicting environmental factors such as irradiance, temperature, and humidity, which are then used as inputs in a PV simulator to estimate the PV power output. Conversely, direct forecasting relies on historical PV power output data to directly anticipate future power generation. To enhance the accuracy of the prediction, additional meteorological data can be incorporated alongside the PV power output in both indirect and direct forecasting approaches.

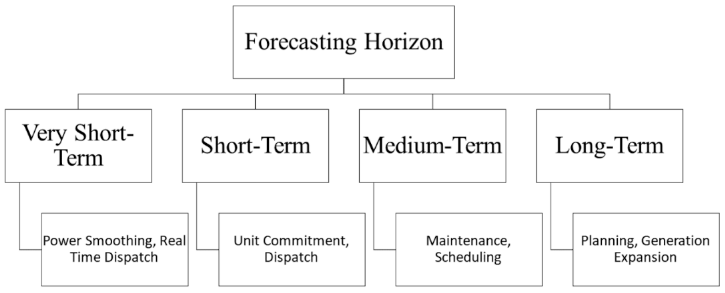

Forecasting Horizon
Very Short-Term
Short-Term
Medium-Term
Long-Term
Power Smoothing, Real Time Dispatch
Unit Commitment, Dispatch
Maintenance, Scheduling
Planning, Generation Expansion

Figure 2.9: Different prediction horizons and their applications

In their publication [14], Gigoni, Betti, Crisostomi, and their collaborators investigated the reasons behind the superior accuracy of direct methods over indirect methods in PV solar forecasting. One of the key factors they identified is that indirect prediction models used in the estimation of PV output are mere approximations of real-world PV systems, lacking the ability to capture every possible physical phenomenon that may occur. Moreover, the physical characteristics of PV systems gradually deteriorate over time, leading to diminished reliability in the forecasts generated by indirect methods. Consequently, the research project under consideration focuses on exploring and evaluating direct techniques for PV solar forecasting.

3 | Data and Tools

3.1. Data

The data are sourced from the National Renewable Energy Laboratory Photovoltaic Data Acquisition (NREL PVDAQ), which is a large-scale time-series database containing system metadata and performance data from a variety of experimental PV sites like San Francisco, USA as shown in Figure 3.1 and commercial public PV sites [26].

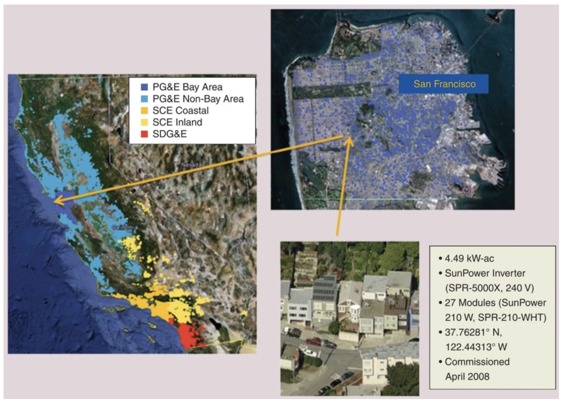
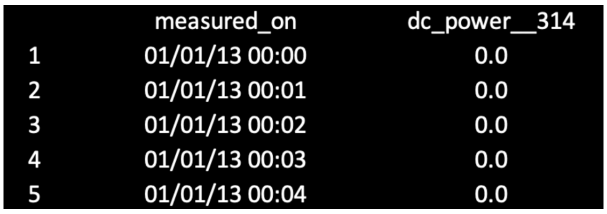

🟦 PG&E Bay Area
🟦 PG&E Non-Bay Area
🟨 SCE Coastal
🟨 SCE Inland
🟥 SDG&E
San Francisco
• 4.49 kW-ac
• SunPower Inverter (SPR-5000X, 240 V)
• 27 Modules (SunPower 210 W, SPR-210-WHT)
• 37.76281° N, 122.44313° W
• Commissioned April 2008

Figure 3.1: PV Panels installed in San Francisco

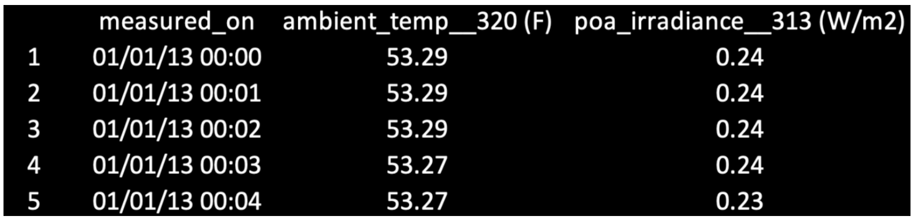
Photovoltaic field array data are made up of time-series, raw performance data collected by a number of sensors linked to a PV system. Two datasets are utilized - One ranging from 2013 to 2018 with one-minute sampling interval and the other from 2011 to 2019 with 15-minute sampling interval. Figure 3.2 displays a selection of rows from the power dataset, offering contextual information.



	measured_on	dc_power__314
1	01/01/13 00:00	0.0
2	01/01/13 00:01	0.0
3	01/01/13 00:02	0.0
4	01/01/13 00:03	0.0
5	01/01/13 00:04	0.0

Figure 3.2: An Example of power dataset

Some meteorological data was also available like ambient temperature and solar irradiance. For some systems, this data was available in different datasets [26]. To provide context, Figure 3.3 showcases a selection of rows from the weather dataset, where the sampling rate is also set at one minute.



	measured_on	ambient_temp__320 (F)	poa_irradiance__313 (W/m2)
1	01/01/13 00:00	53.29	0.24
2	01/01/13 00:01	53.29	0.24
3	01/01/13 00:02	53.29	0.24
4	01/01/13 00:03	53.27	0.24
5	01/01/13 00:04	53.27	0.23

Figure 3.3: An Example of weather dataset

Furthermore, the data lake offered data for 156 PV systems deployed in the previously indicated locations. Because there were data discrepancies involving some of the systems. The System 4 dataset was used for this study since it had comprehensive data from 2013 to 2018. Furthermore, because these units were connected to the grid, a massive amount of data pertaining to them was also available. The dataset had inconsistencies such as negative values and extremely high values. These issues will be addressed in Chapter 4.

3.2. Implementation Tools

During the course of this project's implementation, the following tools and libraries were used:

Jupyter Notebooks are a web-based interactive computing platform that allows users to create and share documents that contain live code, equations, visualizations, and narrative text. The Jupyter notebook interface consists of cells. Each cell can contain code, Markdown text, or raw text. You can run individual cells and see the output immediately, which makes it easy to experiment and iterate.

Python Programming Language is one of the most popular programming languages for machine learning and data science due to its simplicity and versatility. It offers a number of libraries and frameworks that make it easy to implement and experiment with machine learning algorithms.

Matplotlib is a plotting library for the Python programming language. Matplotlib provides a large library of customizable plot types, including line plots, scatter plots, bar plots, error bars, histograms, bar charts, pie charts, box plots, violin plots, density plots, and more.

NumPy is a Python library used for scientific computing and data analysis. It provides a high-performance multidimensional array object and tools for working with these arrays.

Pandas is a fast, powerful, flexible and easy to use open-source data analysis and data manipulation library built on top of the Python programming language. It provides data structures for efficiently storing large datasets and tools for working with them.

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn is used for visualizing statistical relationships between multiple variables.

TensorFlow is an open-source software library for machine learning and deep learning. It provides a flexible and powerful platform for building and deploying machine learning models, allowing users to create and train neural networks for a wide range of tasks such as image classification, natural language processing, and time-series forecasting.

4 | Implementation and Results

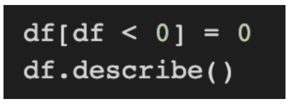
4.1. Data Exploration and Pre-processing

The accuracy and reliability of forecasting heavily depend on the quality of input data. Historical time series data of PV output and specific meteorological information for power stations and their geographical locations have been widely utilized in research projects for modeling purposes. However, these datasets often contain intermittent static or spike elements resulting from weather variations, seasonal changes, fluctuations in electricity demand, and occasional power system failures. These outliers lack a discernible trend, are influenced by chance events, and significantly impact forecast accuracy. Additionally, data may be corrupted or contain missing values due to sensor defects or erroneous recordings. Therefore, it is crucial to preprocess distorted input data using techniques such as decomposition, interpolation, or seasonal adjustments (i.e., data cleansing and structural modifications).

To transform the dataset into a usable format, each year's data is considered. The Date-Time column is set as the index of the data frame, enabling the processing of datasets based on the index variable. Each day of the year corresponds to its own dataset, which may exhibit inconsistencies such as varying numbers of columns. To address these discrepancies, the datasets for each year are combined, and unnecessary columns are removed.

4.1.1. Negative Values

The subsequent step involves handling negative values in the dataset. The dataset had negative values relating to power and solar irradiance. Because all of the negative numbers in the dataset correlate to time values during the night, which are irrelevant. As a result, such values are set to null to further sanitize the data. An example of how negative values in the data frame are made to zero is depicted in Figure 4.1.



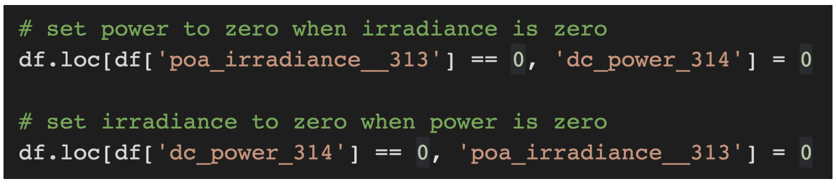
```
df[df < 0] = 0
df.describe()
```

Figure 4.1: Setting Negative values to zero

Efficient pre-processing steps were implemented to filter out any solar irradiance values with missing associated PV power values, as well as PV power records with missing irradiance data. The detailed steps are as follows:

- During the early and late hours of the day, negative solar radiation values and missing associated power values are often observed. These occurrences can be attributed to sensor offsets and inverter failures, respectively. To address this, it is advisable to set the radiation and PV output values to zero (0) in such instances.
- The absence of solar radiation and output power data during mid-day periods may be ascribed to malfunctions in solar irradiation sensors, inverters, or network disruptions. In order to ensure accurate analysis, it is recommended to exclude these data points from further processing.

An example of how missing values in the data frame are made to zero is depicted in Figure 4.2.



```
# set power to zero when irradiance is zero
df.loc[df['poa_irradiance__313'] == 0, 'dc_power_314'] = 0

# set irradiance to zero when power is zero
df.loc[df['dc_power_314'] == 0, 'poa_irradiance__313'] = 0
```

Figure 4.2: Setting missing values to zero

4.1.2. Statistical Measures

Statistical measures can be used to better comprehend the data:

Count: The count refers to the number of records or observations for each attribute.

Mean: The mean represents the average value of each attribute, obtained by summing all the values and dividing by the count.

Std: The standard deviation (std) indicates the amount of dispersion or variability of data around the mean. It provides information about the spread of values within the dataset.

Min: The minimum value is the smallest observed value among the data points for a given attribute.

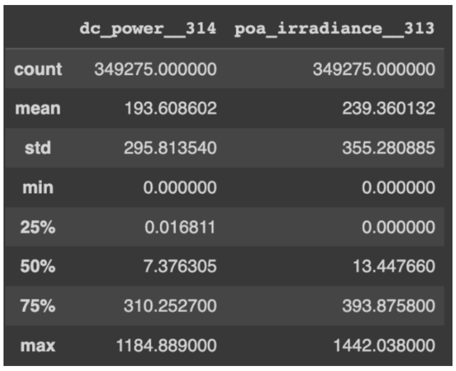
25%: The 25th percentile, also known as the lower quartile, divides the data into four equal parts, with 25

50%: The 50th percentile represents the median, which is the middle value in a sorted dataset. It indicates the value that divides the data into two equal halves, revealing information about the distribution's skewness.

75%: The 75th percentile, or the upper quartile, indicates the threshold below which 75

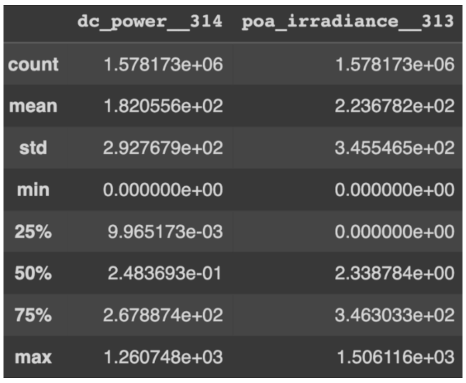
Max: The maximum value is the largest observed value among the data points for a given attribute.

Table 4.1 displays statistical analysis for the dataset corresponding to 2018 and Table 4.2 displays statistical analysis for the dataset corresponding to 2014-2016 which has been used for training, testing and validating the model.



	dc_power__314	poa_irradiance__313
count	349275.000000	349275.000000
mean	193.608602	239.360132
std	295.813540	355.280885
min	0.000000	0.000000
25%	0.016811	0.000000
50%	7.376305	13.447660
75%	310.252700	393.875800
max	1184.889000	1442.038000

Table 4.1: Statistical analysis (2018)



	dc_power__314	poa_irradiance__313
count	1.578173e+06	1.578173e+06
mean	1.820556e+02	2.236782e+02
std	2.927679e+02	3.455465e+02
min	0.000000e+00	0.000000e+00
25%	9.965173e-03	0.000000e+00
50%	2.483693e-01	2.338784e+00
75%	2.678874e+02	3.463033e+02
max	1.260748e+03	1.506116e+03

Table 4.2: Statistical analysis (2014-2016)

4.1.3. Data Visualization

4.1.3.1. Periodicity of Power output

The algorithm used in this case first creates two new features, "Day sin" and "Day cos", as well as "Year sin" and "Year cos", using sinusoidal signals that represent the minute of the day and day of the year respectively. Then, the FFT (Fast Fourier Transform) of DC power output is computed and the spectrum of the signal is plotted, which shows the periodicity of the signal in terms of days and years. The peaks in the spectrum are also calculated. Finally, the algorithm gives the dominant frequencies of the signal in days.

In terms of days, the PV power output is affected by several factors that cause variations in the amount of solar radiation reaching the panels. These factors include the position of the sun in the sky, the presence of clouds, and atmospheric conditions such as humidity and air pollution. As a result, the PV power output typically follows a daily pattern, with a peak around midday when the sun is highest in the sky and the least amount of shading occurs. In this case, we have found a high periodicity in terms of 24 hours and 12 hours.

In terms of years, the PV power output is affected by the seasonal changes in solar radiation. This is because the sun's angle in the sky changes throughout the year, which affects the amount of solar radiation reaching the panels. In general, PV power output is highest in the summer months when the sun is highest in the sky and days are longest, and lowest in the winter months when the sun is at its lowest angle and days are shortest. Figure 4.3 depicts the spectrum of the power signal.

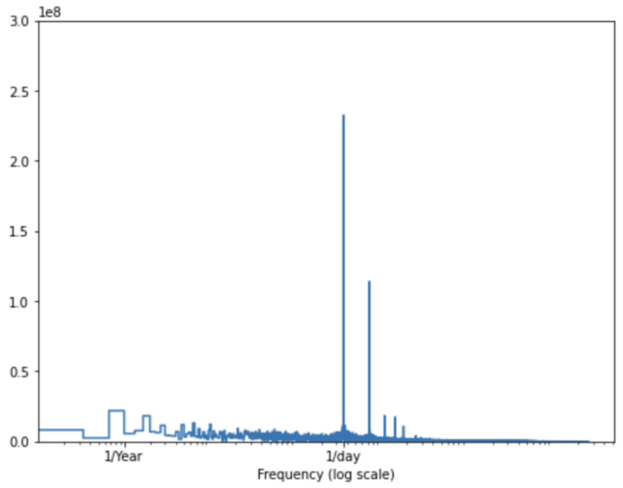


Figure 4.3: Spectrum of PV Power signal

4.1.3.2. Periodicity of Solar Irradiance

Solar Irradiance is the amount of solar radiation received by the Earth's surface per unit area. It varies over different time scales due to various factors such as the Earth's rotation, revolution, and the Sun's activity. On a daily basis, solar irradiance varies due to the Earth's rotation. The solar irradiance is highest at solar noon, which is the time when the Sun is directly overhead. Solar noon occurs around noon local time and varies depending on the observer's longitude and the time of the year. The solar irradiance is lower in the morning and evening when the Sun is lower in the sky. On a yearly basis, solar irradiance varies due to the Earth's revolution around the Sun. In this case, the FFT (Fast Fourier Transform) of Solar Irradiance is computed and the spectrum of the signal is plotted, which shows the periodicity of the signal in terms of days and years. Figure 4.4 depicts the spectrum of the solar irradiance signal.

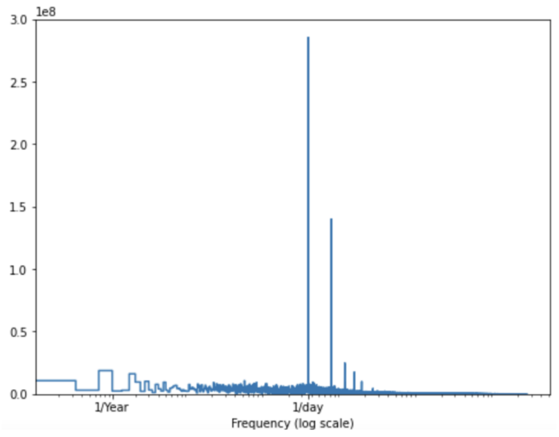
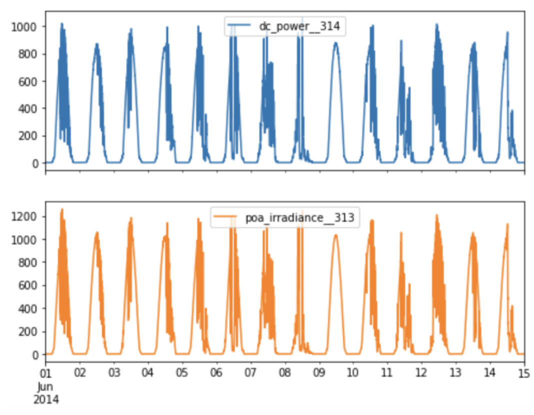


Figure 4.4: Spectrum of Solar irradiance signal

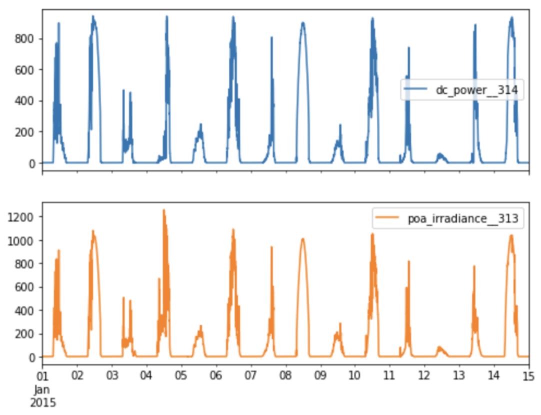
The correspondence between PV output power and solar irradiance can be clearly observed. Therefore, as solar irradiance levels vary throughout the day depending on the weather conditions, the output power of a PV system will also fluctuate and understanding the correspondence between PV output power and solar irradiance is important for designing and optimizing PV systems for maximum efficiency and output.

4.1.3.3. Temporal Progression of PV Power and Solar Irradiance

Figure 4.5 depicts the temporal progression of generated power and solar irradiance. Power and solar irradiance have comparable trends, with higher values in the summer and lower values in the winter, as expected.



(a) Power and Irradiance (June 2014)



(b) Power and Irradiance (January 2015)

Figure 4.5: Visualization of data

4.1.4. Features Selection

Feature selection aims to enhance performance by eliminating irrelevant and redundant data. An effective feature subset consists of attributes that are strongly associated with the output variable but not strongly correlated with each other [18]. A negative correlation indicates an inverse relationship between variables, where an increase in one variable leads to a decrease in another, and vice versa. Considering the high correlation between power and solar irradiance, as demonstrated previously, the study focuses on two features: power and solar irradiance.

4.1.5. Data Normalization

A Python function is used to normalize the data. The function takes a pandas DataFrame as an input and returns a new DataFrame with each column normalized to have minimum value 0 and maximum value 1.

Some of the benefits of normalizing the data are:

Improved numerical stability: Many machine learning algorithms, especially those that involve optimization, perform better when the input data is normalized. This is because normalization helps prevent numerical instability and optimizes the numerical accuracy of the computations.

Better handling of scale: Normalizing the data helps handle the differences in scale between different features or variables. This can help prevent some features from dominating others, which can skew the results and make the model less interpretable.

Improved interpretability: Normalizing the data can make it easier to compare and interpret the results, especially when comparing variables or features with different scales.

Easier feature engineering: Normalizing the data can make it easier to engineer new features or to combine existing features, since the data is on the same scale.

4.1.6. Splitting of Dataset

Splitting a dataset into train, test, and validation sets is a common practice in machine learning to evaluate the performance of a model. The train set is used to train the model and optimize its parameters. The larger the training set, the better the model can learn the patterns in the data. The test set is used to evaluate the performance of the model after it has been trained. It provides an unbiased estimate of the model's accuracy on new data. The validation set is used to tune the hyperparameters of the model. The idea is to use the validation set to determine the best set of hyperparameters that lead to the best performance on the test set. In this case, datasets corresponding to 2014-2016 are used in which data of 2014 and 2015 is used to train the model, data of 2016 is equally split to test and validate the model.

4.2. Performance Metrics

Performance metrics are statistical measures used to evaluate the quality and accuracy of a model. They provide a way to determine the effectiveness of a model by comparing its predictions to actual results. The choice of performance metric depends on the problem being solved and the specific requirements of the problem [36]. In this case, following performance metrics are used:

4.2.1. Mean Squared Error (MSE)

Mean Squared Error (MSE) is a commonly used loss function in supervised machine learning, especially in regression problems. It is defined as the average of the squared differences between the predicted values and the true values. Mathematically, the MSE for a set of predictions is calculated as:

Supposing power time series is $Y_i = Y_1, Y_2, ..., Y_n, \hat{Y}_i$ is the predicted time series, and indicates time.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{4.1}$$

where n is the number of instances in the dataset and the $\sum$ symbol represents the sum over all instances.

The MSE measures the average magnitude of the errors in the predictions, and penalizes large differences more than smaller ones. By squaring the differences, the loss function ensures that the error is positive and assigns a higher loss value to larger errors. The MSE is commonly used because it's easy to compute and differentiable, making it suitable for optimization with gradient-based methods.

4.2.2. Mean Absolute Error (MAE)

Mean Absolute Error (MAE) is a loss function used in regression problems, where the goal is to predict a continuous value output. It is calculated as the average of the absolute differences between the true values and the predicted values. Mathematically, it can be represented as:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \tag{4.2}$$

where y_i is the prediction and x_i is the true value. The advantage of using MAE as the loss function is that it is easy to interpret and understand, as it gives the average magnitude of the error in the units of the target variable.

4.2.3. Root Mean Squared Error (RMSE)

Root Mean Squared Error (RMSE) is a commonly used loss function for regression problems. The RMSE is calculated as the square root of the average of the squared differences between the actual and predicted values. Mathematically, it can be represented as:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}} \tag{4.3}$$

where $y_i = y_1, y_2, ..., y_n$ are observed values, $\hat{y}_i$ is a predicted time series and i indicates time. The RMSE penalizes larger differences more heavily than smaller differences, which makes it suitable for cases where the range of the target variable is large. Moreover, the RMSE is in the same units as the target variable, which makes it easy to interpret and compare the results with the actual values.

4.3. Hyperparameters Tuning

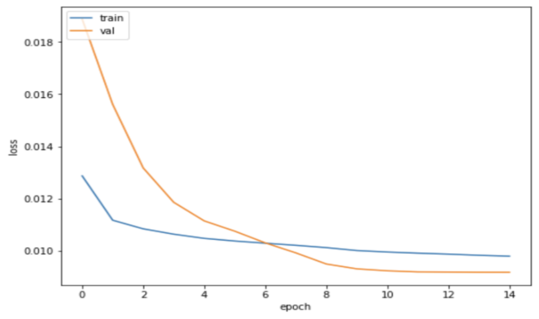
Hyperparameter tuning is the process of finding the optimal set of hyperparameters for a machine learning model to achieve the best performance on a given dataset. Hyperparameters are parameters that are not learnt from data during training but are instead defined by the user before the model is trained [35]. Learning rate, regularization strength, and the number of hidden layers in a neural network are all examples of hyperparameters. The method of tuning hyperparameters entails selecting a range of values for each hyperparameter and then assessing the model's performance on a validation set with each combination of hyperparameters.

In machine learning, the **learning rate** is a hyperparameter that determines the step size at each iteration during the optimization process of a neural network. It is a scalar value that determines the size of the update made to the model's weights and biases during training. If the learning rate is too small, the model will take a long time to converge to the minimum of the loss function, while if it is too large, the model may overshoot the minimum and fail to converge.

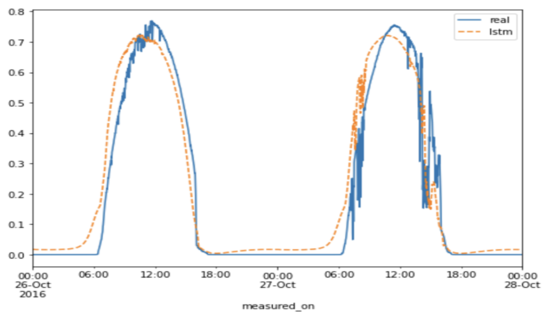
The number of **epochs** is a hyperparameter that can significantly affect the performance of a neural network. Too few epochs may result in underfitting, where the model is not able to learn the underlying patterns in the data, while too many epochs can result in overfitting, where the model learns to fit the training data too well and performs poorly on new data.

Learning rate, and epoch numbers are modified in this study. Also, in all cases power output for 1 hour in the future has been predicted using 3 hours in the past.

Figures 4.6 - 4.10 show the outcomes of hyperparameter adjustment for the LSTM architecture.

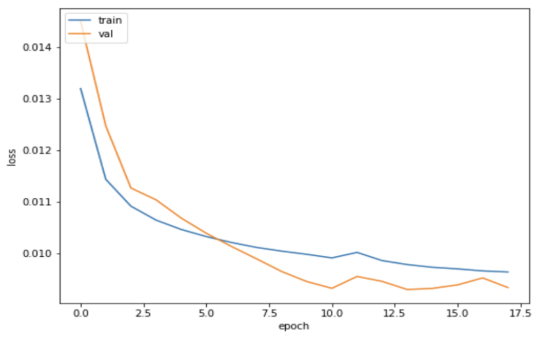


(a) Training Curve

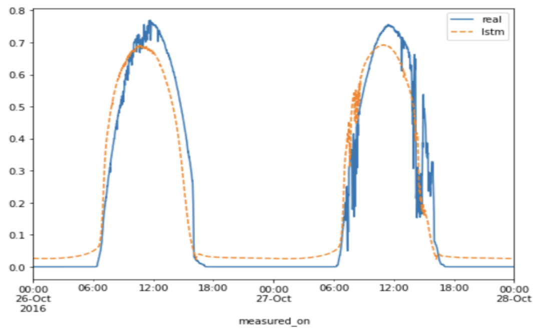


(b) Forecasting Results

Figure 4.6: Training and Forecasting results, Learning rate = 0.0001, Epochs = 15

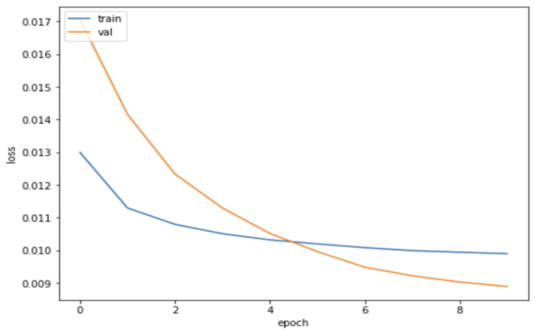


(a) Training Curve

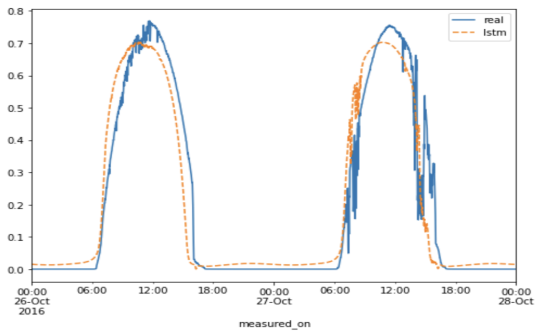


(b) Forecasting Results

Figure 4.7: Training and Forecasting results, Learning rate = 0.001, Epochs = 18

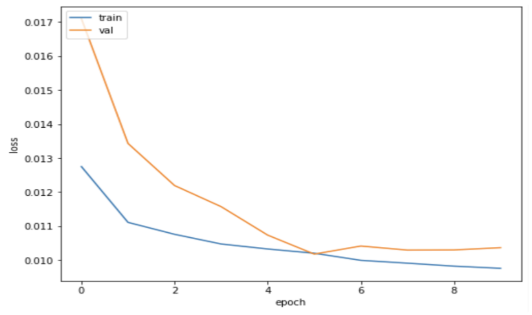


(a) Training Curve

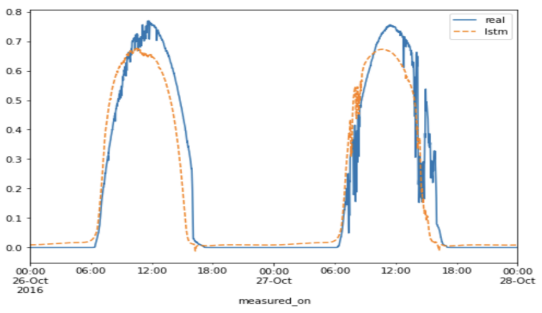


(b) Forecasting Results

Figure 4.8: Training and Forecasting results, Learning rate = 0.01, Epochs = 10

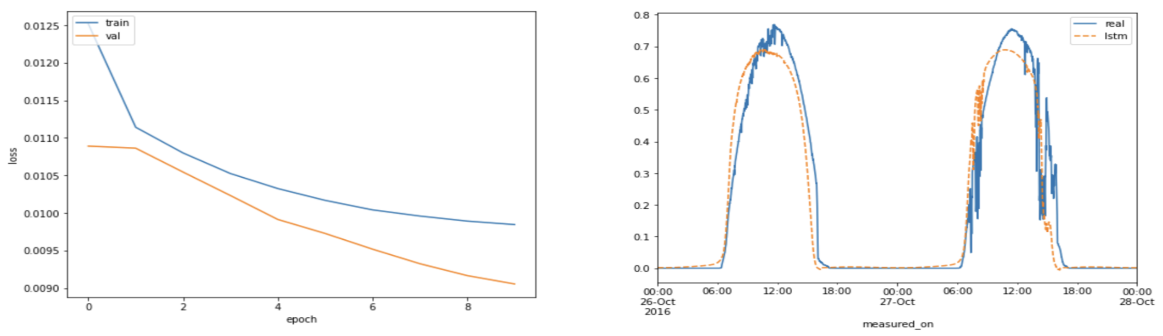


(a) Training Curve



(b) Forecasting Results

Figure 4.9: Training and Forecasting results, Learning rate = 0.001 for epoch ≤ 6; Learning rate = 0.0001 for epoch > 6, Total Epochs = 10



(a) Training Curve (b) Forecasting Results

Figure 4.10: Training and Forecasting results, Exponentially Decaying Learning rate; Initial value = 0.01; Decay steps = 1000 and Decay rate = 0.9, Epochs = 10

Comparison

Table 4.3 summarizes hyperparameters tuning results.

Learning Rate	Total Epochs	Best Epoch Number	Test MSE	Test MAE	Test RMSE
0.0001	15	11	0.0099	0.0512	0.0997
0.001	18	9	0.0100	0.0528	0.1002
0.01	10	8	0.0100	0.0521	0.1000
0.001 for Epoch Number $\leq$6; 0.0001 for Epoch Number $>$ 6	10	8	0.0099	0.0500	0.0995
Exponentially Decaying with Initial value = 0.01; Decay steps = 1000, Decay rate = 0.9	10	7	0.0100	0.0510	0.1002

Table 4.3: Hyperparameters tuning results

Based on the results in this table, we can see that different combinations of learning rate and epoch number can significantly affect the performance of a machine learning model :

- Lower learning rates (0.0001 and 0.001) tend to result in better performance on the test dataset, as measured by lower MAE and RMSE values. However, these lower learning rates may require more epochs to achieve good performance, as seen in the higher epoch numbers for the 0.0001 and 0.001 learning rates.

- Exponentially decaying learning rates can also be effective, as seen in the last row of the table, which achieved favorable test MAE and RMSE.
- Learning rate of 0.001 for Epoch Number ≤ 6 and 0.0001 for Epoch Number > 6 can also be considered as a good combination based on the results in the table. This setting achieved a test MAE of 0.0500 and a test RMSE of 0.0995, which are among the lowest values in the table. Additionally, it achieved good performance with only 8 epochs of training, which is fewer than some of the other settings in the table.

It's worth noting that the best epoch numbers for each learning rate can vary, suggesting that finding the best hyperparameters requires experimentation and tuning.

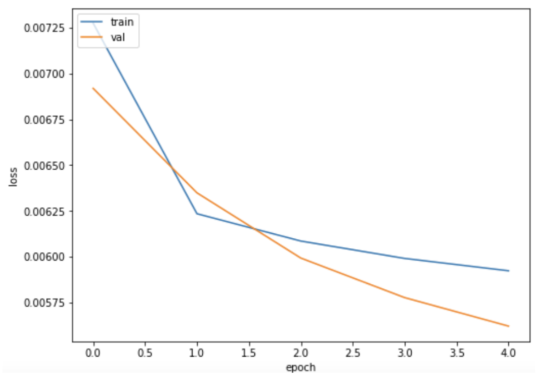
4.4. Short-time Prediction

The prediction horizon of PV power output refers to the length of time over which future solar power generation can be accurately predicted. A shorter prediction horizon generally refers to a timeframe of a few hours to a day.

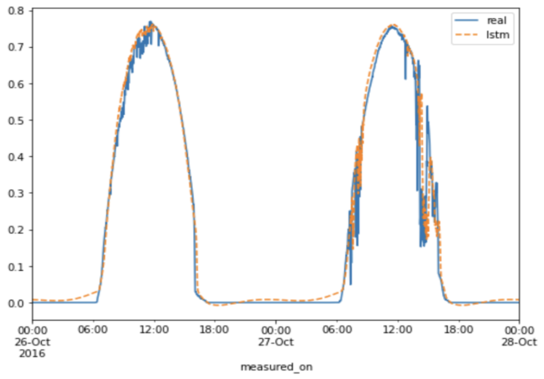
One of the key benefits of a shorter PV power output prediction horizon is that it can lead to greater accuracy in predicting solar power generation. This is due to the fact that weather conditions and other factors affecting solar power output may change rapidly, and a shorter prediction horizon allows for more up-to-date information to be integrated into the forecast. With a shorter horizon, more current meteorological data can be incorporated, resulting in more accurate predictions. Another benefit of a narrower prediction horizon is that it allows for better integration of solar power into the power network. Grid operators can better manage electricity supply and demand with more precise estimates of solar power generation over shorter periods.

It is feasible to improve the charging and discharging of energy storage systems by properly estimating the amount of solar power that will be generated over a shorter duration. This can assist to cut expenses and increase the energy storage system's efficiency. Lastly, shorter forecast horizons can aid renewable energy producers in asset management. Reliable forecasts of solar power output over shorter durations can assist producers in properly scheduling maintenance, planning for equipment improvements, and making other critical operational decisions. To demonstrate the difference of the accuracy achieved for shorter prediction horizons, a prediction horizon of 15 minutes using data of past 3 hours has been taken as an example in this case.

Figure 4.11 shows the learning curve and simulation results.



(a) Training Curve



(b) Forecasting Results

Figure 4.11: Training and Forecasting results, Prediction Horizon = 15 minutes

4.4.1. Hyperparameters Tuning for Short-time Prediction

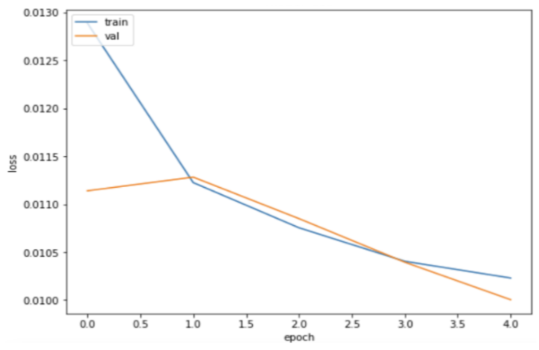
In this section, hyperparameters are tuned for short-time prediction. The following hyperparameters are considered:

Learning Rate: As discussed before, learning rate is a hyperparameter that determines the step size at each iteration during the optimization process of a neural network. It is a scalar value that determines the size of the update made to the model's weights and biases during training.

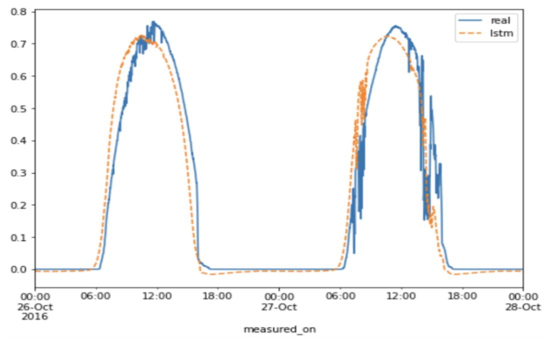
Window Length: Window length refers to the number of time steps that the network considers when processing a sequence of input data. In practice, the window length can be chosen based on the length of the input sequences and the desired level of temporal dependency that the network should capture. A longer window length can capture longer-term dependencies in the input sequences, but may also require more computational resources to process. A shorter window length may be more computationally efficient but may not capture long-term dependencies as effectively.

Prediction Horizon: Prediction horizon refers to the number of time steps into the future that the network is trained to predict. The prediction horizon is determined by the output layer of the network, which is typically designed to produce a sequence of output vectors that correspond to the predicted values for each time step in the future. In practice, the prediction horizon can be chosen based on the task at hand and the desired level of accuracy in the predictions.

Figures 4.12 - 4.29 show the outcomes of hyperparameter adjustment for the LSTM architecture for short-time prediction.

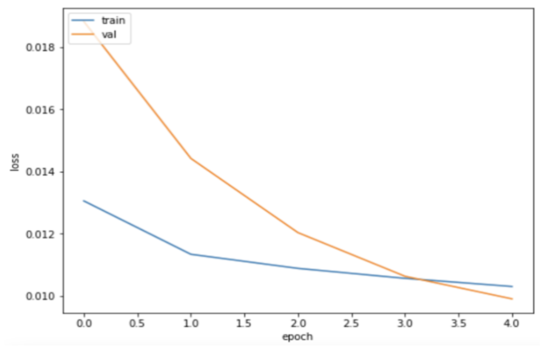


(a) Training Curve

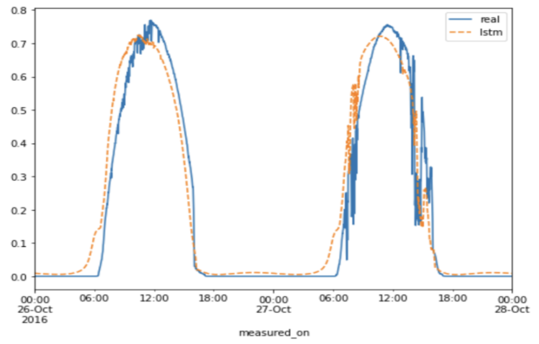


(b) Forecasting Results

Figure 4.12: Training and Forecasting results, Learning rate = 0.001, Window Length = 6hr, Predicion Horizon = 1hr

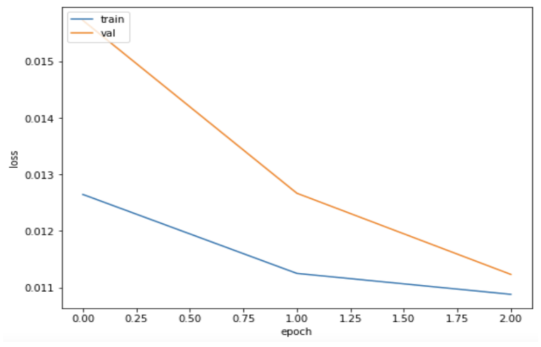


(a) Training Curve

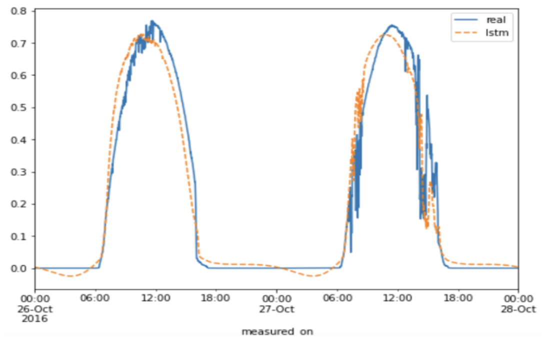


(b) Forecasting Results

Figure 4.13: Training and Forecasting results, Learning rate = 0.01, Window Length = 6hr, Predicion Horizon = 1hr

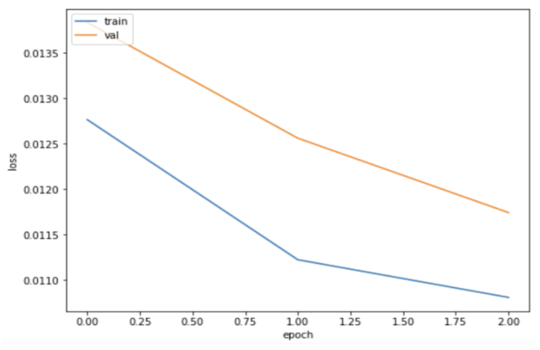


(a) Training Curve

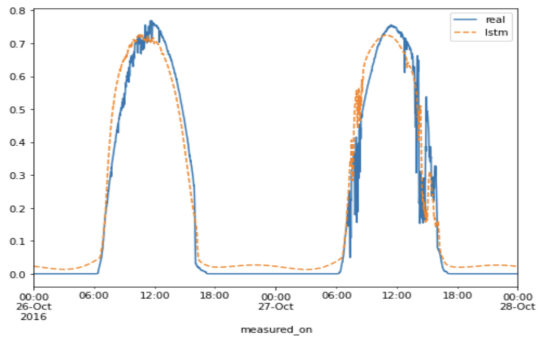


(b) Forecasting Results

Figure 4.14: Training and Forecasting results, Learning rate = 0.001, Window Length = 9hr, Predicion Horizon = 1hr

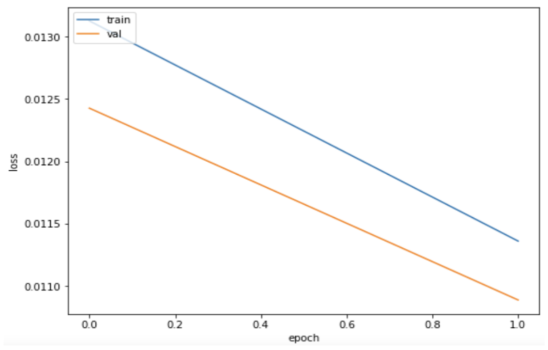


(a) Training Curve

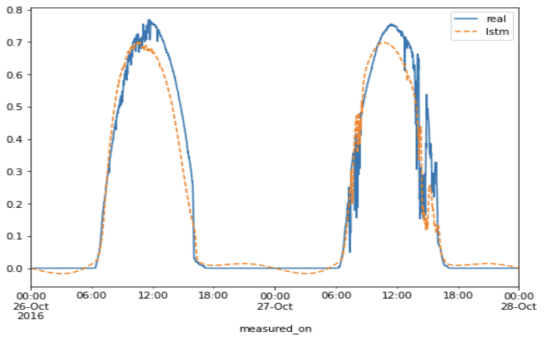


(b) Forecasting Results

Figure 4.15: Training and Forecasting results, Learning rate = 0.01, Window Length = 9hr, Predicion Horizon = 1hr

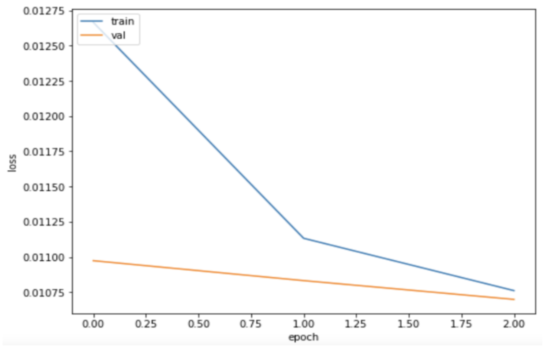


(a) Training Curve

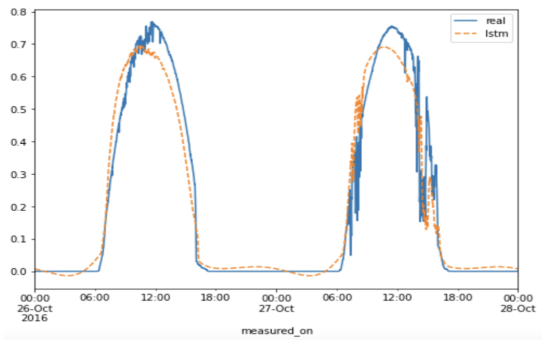


(b) Forecasting Results

Figure 4.16: Training and Forecasting results, Learning rate = 0.001, Window Length = 12hr, Predicion Horizon = 1hr

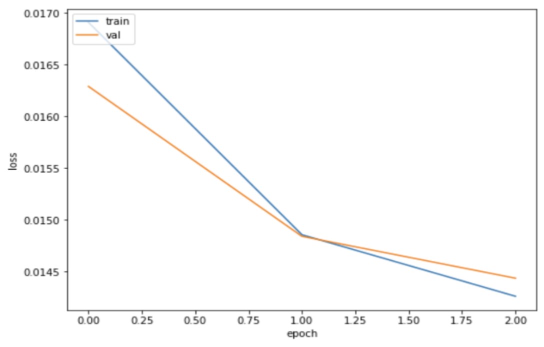


(a) Training Curve

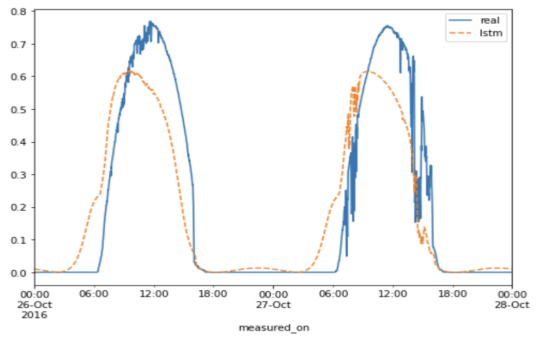


(b) Forecasting Results

Figure 4.17: Training and Forecasting results, Learning rate = 0.01, Window Length = 12hr, Predicion Horizon = 1hr

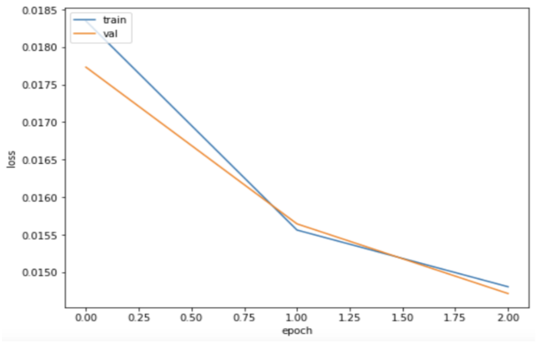


(a) Training Curve

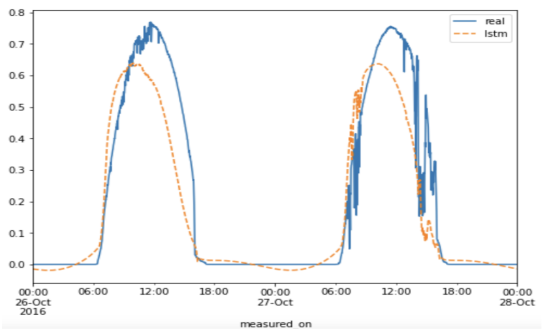


(b) Forecasting Results

Figure 4.18: Training and Forecasting results, Learning rate = 0.001, Window Length = 6hr, Predicion Horizon = 2hr

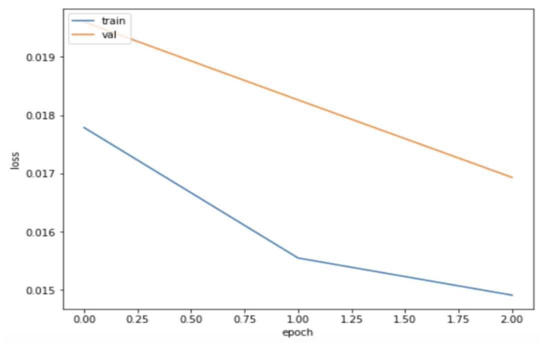


(a) Training Curve

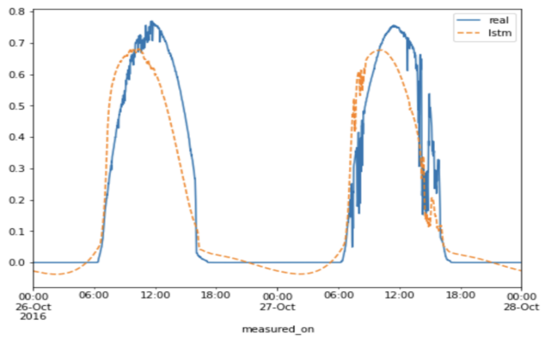


(b) Forecasting Results

Figure 4.19: Training and Forecasting results, Learning rate = 0.01, Window Length = 6hr, Predicion Horizon = 2hr

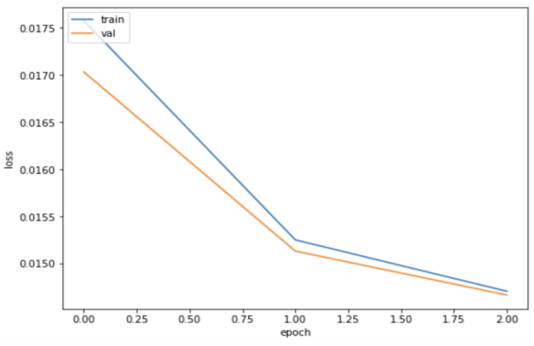


(a) Training Curve

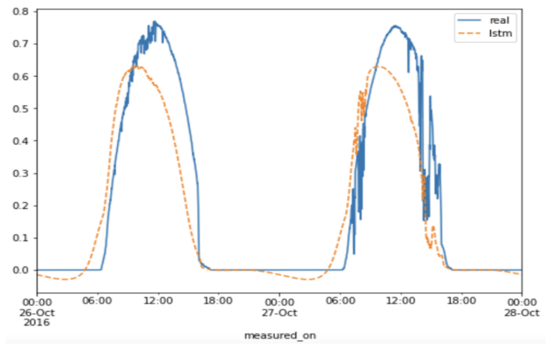


(b) Forecasting Results

Figure 4.20: Training and Forecasting results, Learning rate = 0.001, Window Length = 9hr, Predicion Horizon = 2hr

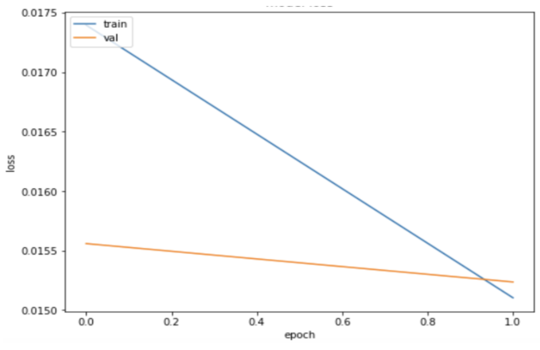


(a) Training Curve

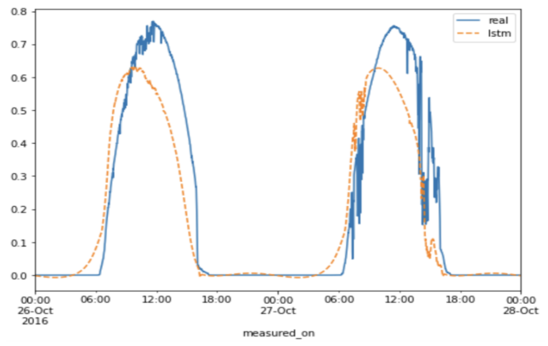


(b) Forecasting Results

Figure 4.21: Training and Forecasting results, Learning rate = 0.01, Window Length = 9hr, Predicion Horizon = 2hr

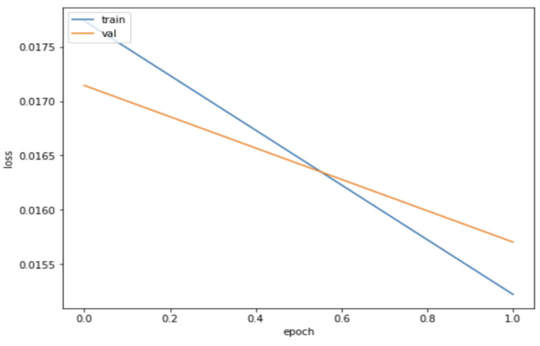


(a) Training Curve

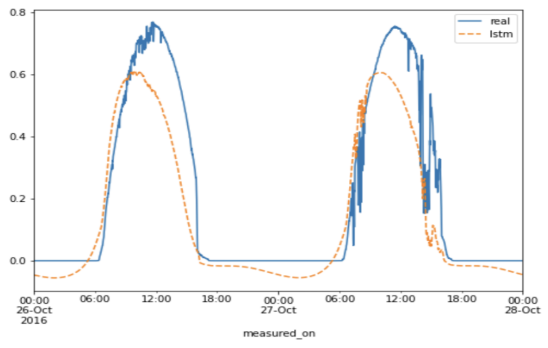


(b) Forecasting Results

Figure 4.22: Training and Forecasting results, Learning rate = 0.001, Window Length = 12hr, Predicion Horizon = 2hr

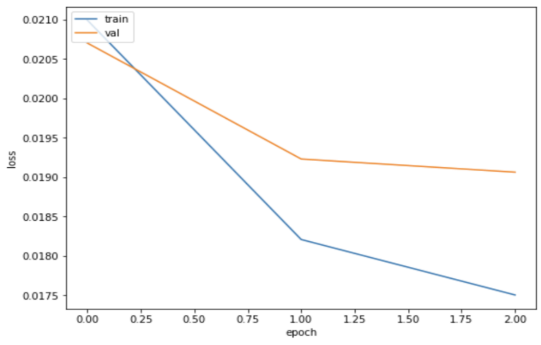


(a) Training Curve



(b) Forecasting Results

Figure 4.23: Training and Forecasting results, Learning rate = 0.01, Window Length = 12hr, Predicion Horizon = 2hr


(a) Training Curve

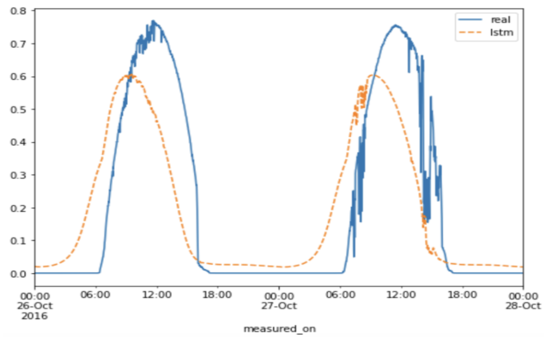
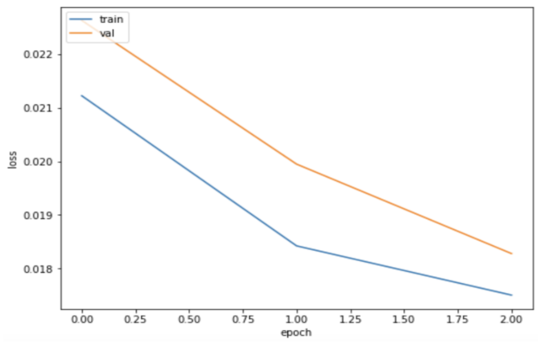

(b) Forecasting Results

Figure 4.24: Training and Forecasting results, Learning rate = 0.001, Window Length = 6hr, Predicion Horizon = 3hr


(a) Training Curve

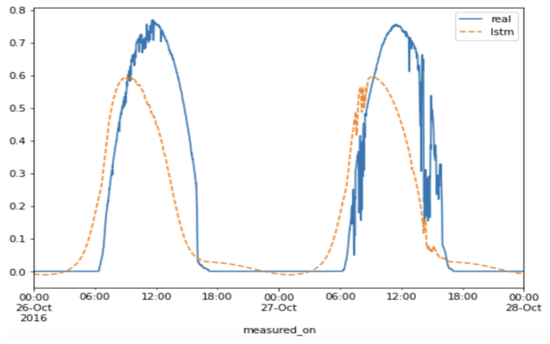
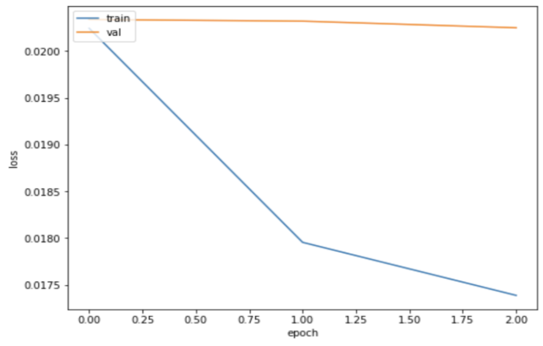

(b) Forecasting Results

Figure 4.25: Training and Forecasting results, Learning rate = 0.01, Window Length = 6hr, Predicion Horizon = 3hr


(a) Training Curve

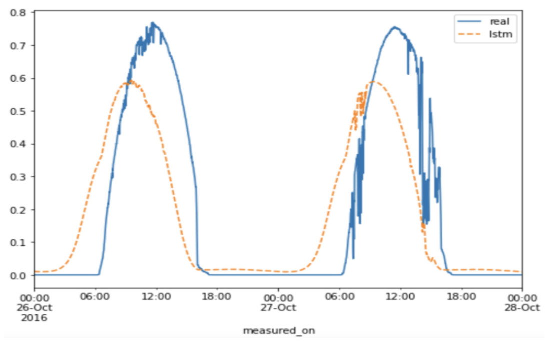
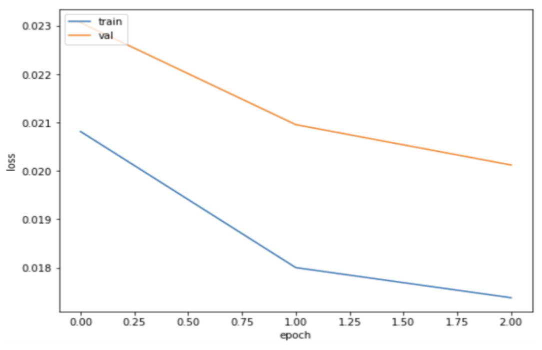
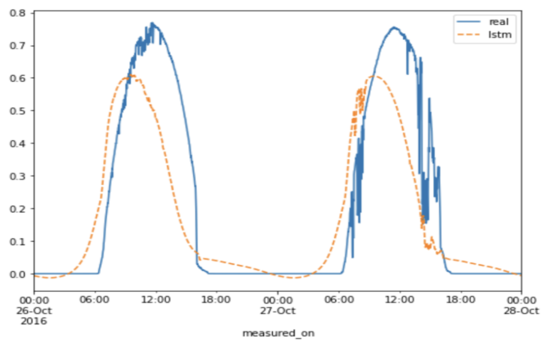

(b) Forecasting Results

Figure 4.26: Training and Forecasting results, Learning rate = 0.001, Window Length = 9hr, Predicion Horizon = 3hr

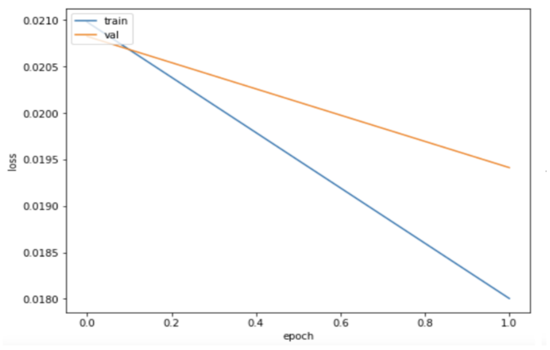


(a) Training Curve

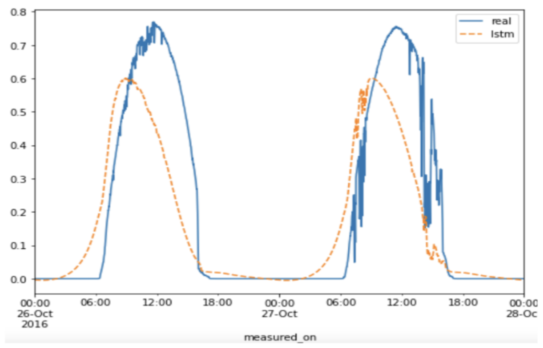


(b) Forecasting Results

Figure 4.27: Training and Forecasting results, Learning rate = 0.01, Window Length = 9hr, Predicion Horizon = 3hr

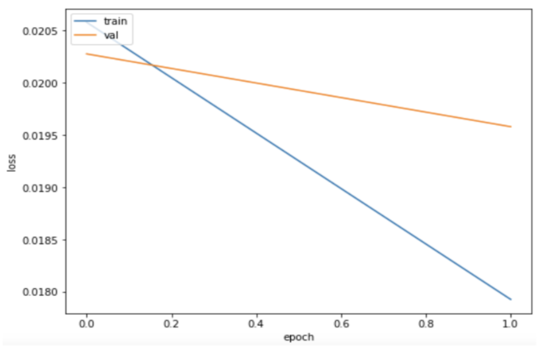


(a) Training Curve

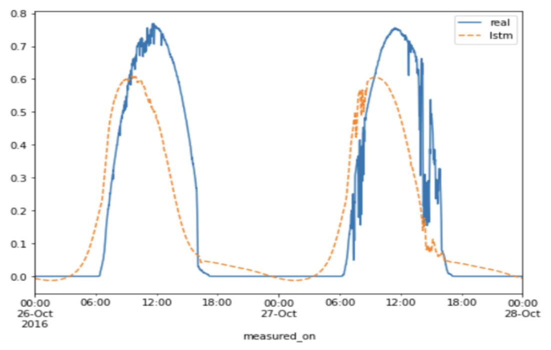


(b) Forecasting Results

Figure 4.28: Training and Forecasting results, Learning rate = 0.001, Window Length = 12hr, Predicion Horizon = 3hr



(a) Training Curve



(b) Forecasting Results

Figure 4.29: Training and Forecasting results, Learning rate = 0.01, Window Length = 12hr, Predicion Horizon = 3hr

Comparison

Table 4.4 summarizes hyperparameters tuning results for short-time prediction.

Learning Rate	Window Length(hr)	Prediction Hori-zon(hr)	Test MAE	Test MSE	Test RMSE
0.001	6	1	0.0527	0.0102	0.1011
0.01	6	1	0.0596	0.0109	0.1042
0.001	9	1	0.0587	0.0109	0.1043
0.01	9	1	0.0588	0.0108	0.1040
0.001	12	1	0.0624	0.0114	0.1066
0.01	12	1	0.0578	0.0108	0.1037
0.001	6	2	0.0687	0.0143	0.1194
0.01	6	2	0.0734	0.0148	0.1217
0.001	9	2	0.0735	0.0149	0.1221
0.01	9	2	0.0700	0.0147	0.1213
0.001	12	2	0.0732	0.0151	0.1229
0.01	12	2	0.0684	0.0146	0.1211
0.001	6	3	0.0793	0.0175	0.1323
0.01	6	3	0.0801	0.0175	0.1323
0.001	9	3	0.0779	0.0174	0.1319
0.01	9	3	0.0796	0.0174	0.1318
0.001	12	3	0.0848	0.0180	0.1342
0.01	12	3	0.0776	0.0172	0.1316

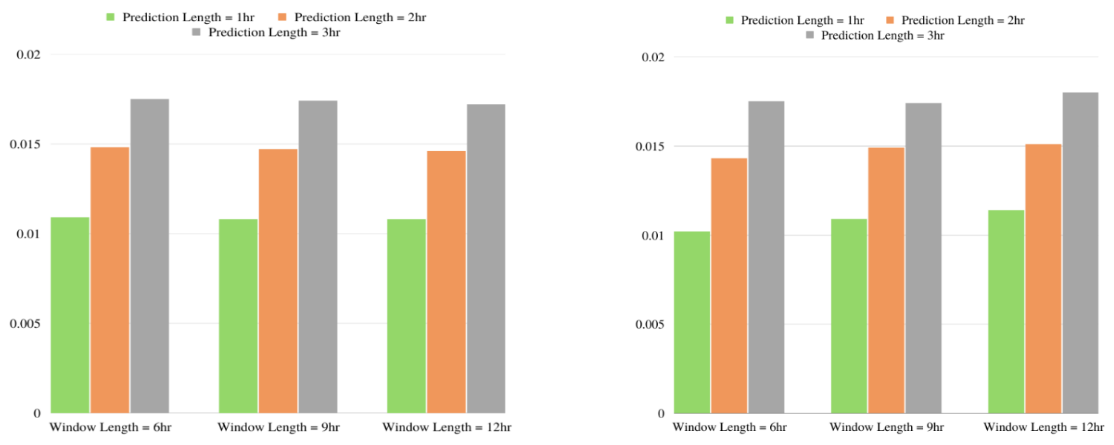
Table 4.4: Hyperparameters tuning results for short-time prediction

Based on the results in this table, we can see that different combinations of learning rate, window length and prediction horizon can significantly affect the performance of a machine learning model :

- The learning rate of 0.01 generally led to better results than the learning rate of 0.001 across most hyperparameter combinations.

- The model's performance is generally improved as the window length is increased. In this case, a window length of 12 hours seemed to produce the best results.
- The model's performance is better for shorter prediction horizons, with the best results being achieved when the horizon is 1 hour which shows that increasing the prediction horizon appears to increase the errors as well, which is not surprising since forecasting farther into the future is generally more challenging than predicting for a shorter duration.

All these results can be analyzed graphically as shown below in Figure 4.30.



(a) Learning Rate = 0.01 (b) Learning Rate = 0.001

Figure 4.30: Hyperparameters Tuning results as a function of Test Set Loss

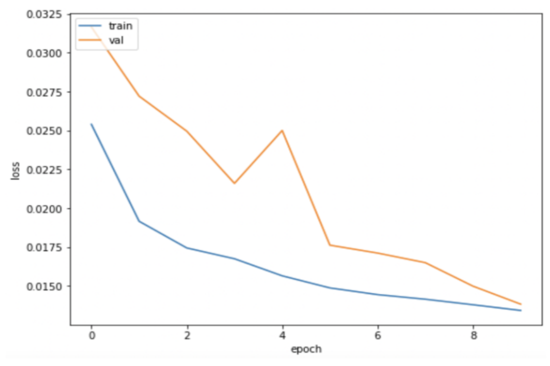
This table provides valuable information for selecting the most effective hyperparameter configuration when using this particular model for a time-series forecasting task. It also provides insights into which hyperparameters are most influential for achieving the best performance. The results suggest that the model may perform better when using larger window lengths and smaller prediction horizons. This suggests that the model may be better at making short-term predictions as compared to longer-term forecasts.

4.4.2. Short-time Prediction using Simpler Networks and Comparison with LSTM

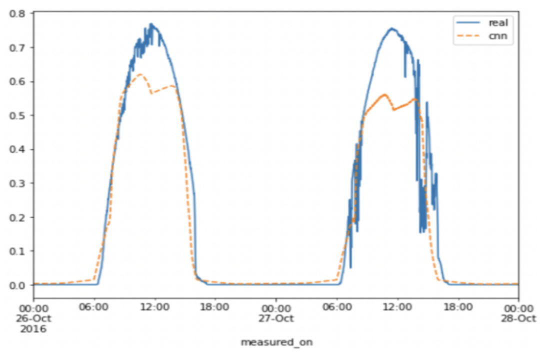
Networks simpler than LSTM can also be used for Solar PV power prediction as simpler networks offer several advantages over LSTM. Firstly, training time is much faster as simpler neural networks have fewer parameters compared to LSTM networks, which require a lot of data and computational resources to be trained properly. Additionally, simpler neural networks are more interpretable, making it easier to understand how the network is making its predictions. They are also less prone to overfitting, which occurs when the network learns to memorize the training data instead of generalizing to new data. Simpler neural networks require less data to train effectively and are easier to deploy in production due to their low computational requirements.

However, it's important to note that simpler neural networks may not always perform as well as LSTM networks, especially in tasks that require processing sequential data or long-term dependencies. In such cases, LSTM networks may be the better choice. Ultimately, the choice of neural network architecture depends on the specific task and available resources.

Figures 4.31 - 4.33 show outcomes concerning the simpler networks.

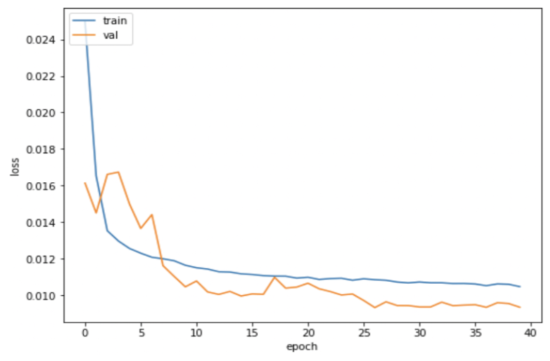


(a) Training Curve

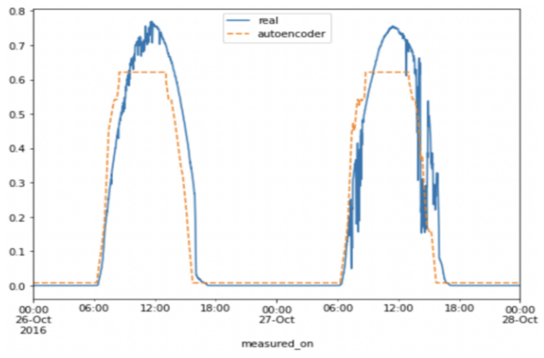


(b) Forecasting Results

Figure 4.31: Convolutional Neural Network (CNN), Window Length = 12hr, Prediction Horizon = 1hr

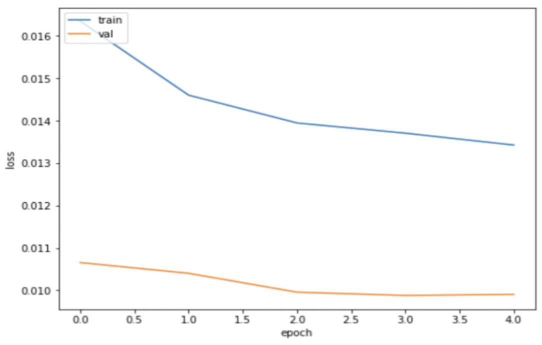


(a) Training Curve

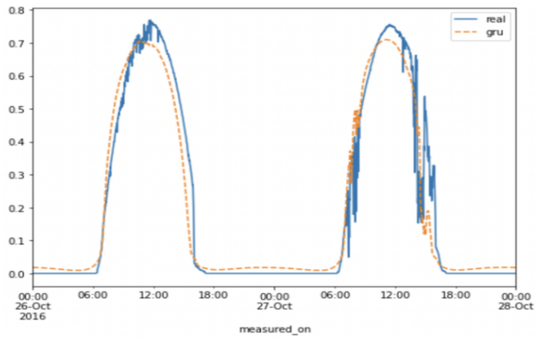


(b) Forecasting Results

Figure 4.32: Autoencoders, Window Length = 12hr, Prediction Horizon = 1hr



(a) Training Curve



(b) Forecasting Results

Figure 4.33: Gated Recurrent Unit (GRU), Window Length = 12hr, Prediction Horizon = 1hr

Comparison

Table 4.5 summarizes results for short-time prediction to compare simpler networks with LSTM.

Network	Window Length(hr)	Prediction Horizon(hr)	Test MAE	Test MSE	Test RMSE
Convolutional Neural Network (CNN)	12	1	0.0649	0.0134	0.1174
Autoencoders	12	1	0.0534	0.0109	0.1162
Gated Recurrent Unit (GRU)	12	1	0.0498	0.0104	0.1159
Long Short-Term Memory (LSTM)	12	1	0.0463	0.0101	0.1025

Table 4.5: Results for short-time prediction using different networks

Based on the results in this table, the four different neural network models were trained and tested on the time series dataset. The results show that the LSTM model performed the best among the four models, as it had the lowest MAE, MSE, and RMSE, with values of 0.0463, 0.0101, and 0.1025, respectively. The next best-performing model was the GRU, and the CNN model had the highest MAE, MSE, and RMSE values among the four models. The results are graphically represented in Figure 4.34.

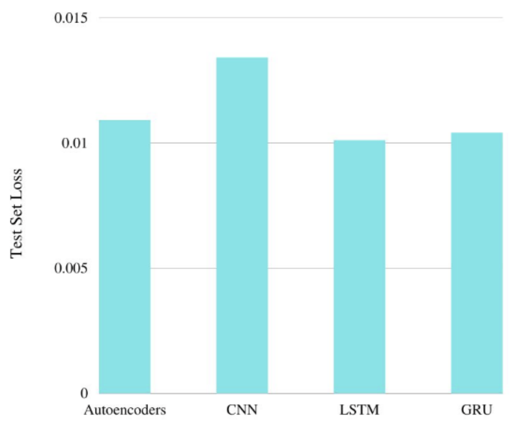


Figure 4.34: Comparison between different networks

Overall, the results suggest that the LSTM and GRU models are the most effective for predicting future values in this time series dataset, while the Autoencoder model also performs well, but the CNN model is less effective for this specific problem.

4.5. Ageing

Solar panels are known to degrade over time due to exposure to the environment, temperature variations, and other factors. This degradation is commonly referred to as "ageing" and can have a significant impact on the power output of the solar panels. As solar panels age, their efficiency in converting sunlight into electricity decreases gradually. This is because the materials used in solar panels, such as silicon, can deteriorate over time and lose their ability to absorb light and generate electrical current [33]. The rate of degradation depends on various factors such as the quality of the materials used, the design of the panel, and the operating conditions.

Some of the reasons for solar panel aging in detail are:

Exposure to the elements: Solar panels are typically installed outdoors and are exposed to various weather conditions such as rain, snow, wind, and hail. These environmental factors can cause wear and tear on the solar panels, leading to cracks, corrosion, and other forms of damage.

Temperature changes: Solar panels are designed to operate in a range of temperatures, but extreme temperatures can cause them to age more quickly. For instance, if the solar panel gets too hot, it may experience thermal stress that can cause the cells to crack or delaminate.

UV radiation: Solar panels are exposed to sunlight, which contains ultraviolet (UV) radiation that can cause the panel's materials to degrade over time. The UV radiation can break down the encapsulant (like Ethylene-vinyl acetate, EVA) that holds the solar cells in place and cause yellowing of the panel surface, which can reduce the panel's efficiency.

Humidity: High humidity levels can cause moisture to seep into the solar panel, which can lead to corrosion of the metal parts and damage to the electrical connections.

Manufacturing defects: Sometimes, solar panels may have manufacturing defects that can cause premature aging. For instance, if the panel's cells are not properly soldered, it can cause hotspots that can reduce the panel's efficiency.

Dust and debris: Solar panels can collect dust and debris over time, which can reduce the amount of sunlight that reaches the cells. This can cause the panel's efficiency to decrease over time.

Figure 4.35 shows defects in PV modules installed in Ghana [31].

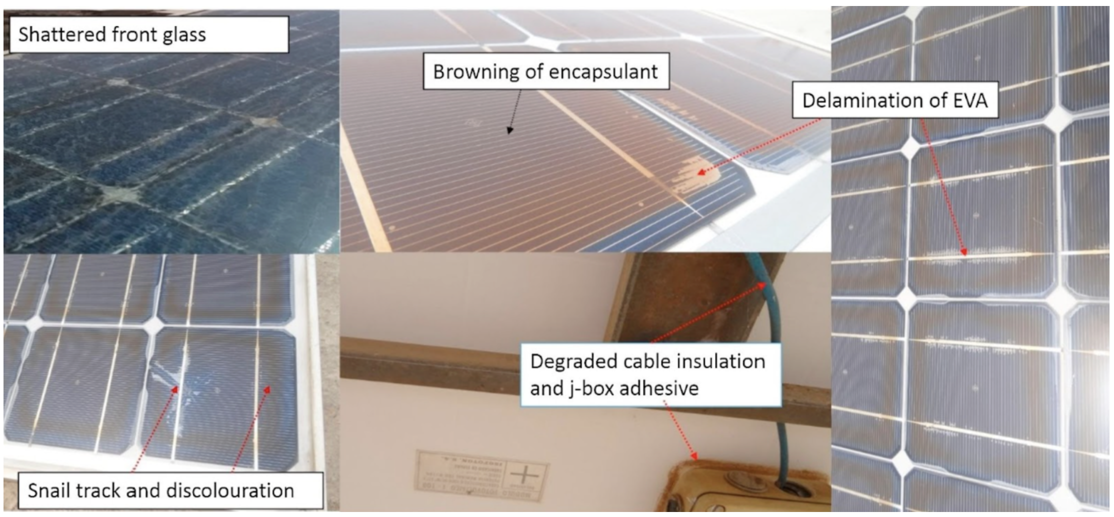

Shattered front glass
Browning of encapsulant
Delamination of EVA
Degraded cable insulation and j-box adhesive
Snail track and discolouration

Figure 4.35: Ageing of PV modules

The power output of solar panels typically decreases by around 0.5% to 1% per year. This means that a solar panel that originally produced 200 watts of power may produce only 180 watts after ten years of use. The degradation rate can vary depending on the specific panel and the environmental conditions it is exposed to. Regular maintenance and cleaning can help extend the lifespan of solar panels and minimize the impact of ageing on their performance.

4.5.1. Simulation Results Using one-minute Resolution Dataset

LSTM is used to train the model to observe the effect of ageing on solar PV power output as LSTM can model complex non-linear relationships between input and output signals, allowing it to capture the complex dynamics of the signal in panel ageing. Data is retrieved from freshly installed PV panels (i.e. "old data" with respect to the present day) and the model is compared with more recent data. It was expected that predictions at times immediately after the data used to create the model should be predicted quite nicely, while predictions made at times that are more in the future should overestimate real data. Thus, the difference between the actual and predicted values over several years in future can be used to observe the effect of ageing. Therefore, data corresponding to 2014-2015 has been used for training, 2016 for validation and 2017-2021 for testing and the prediction is done for 30 minutes in the future using 6 hours in the past.

Figure 4.36 shows learning curve and Figures 4.37 - 4.41 show simulation results for different days in the future.

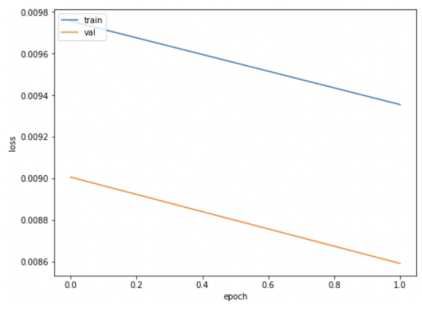
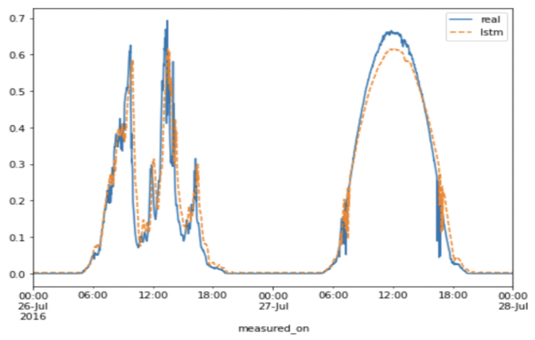
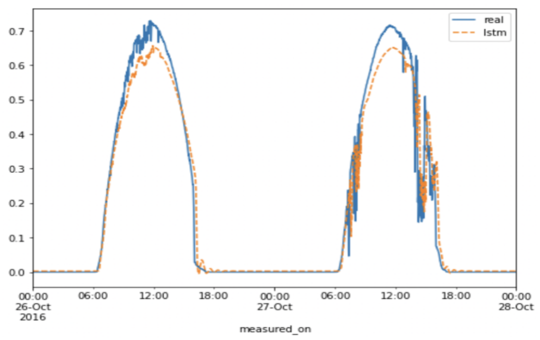


Figure 4.36: Learning Curve

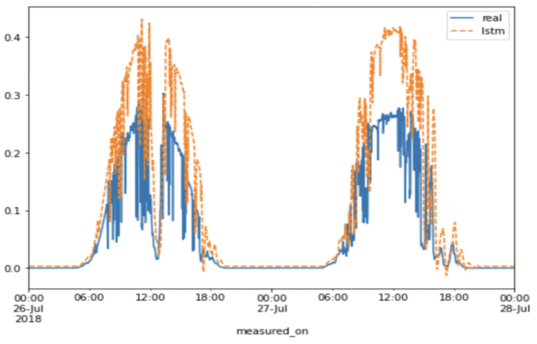


(a) Simulation Results, July 2016

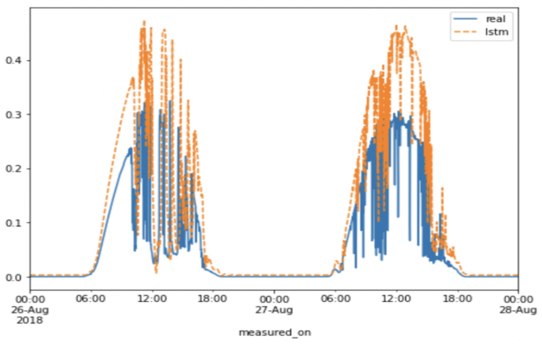


(b) Simulation Results, October 2016

Figure 4.37: Simulation Results, 2016

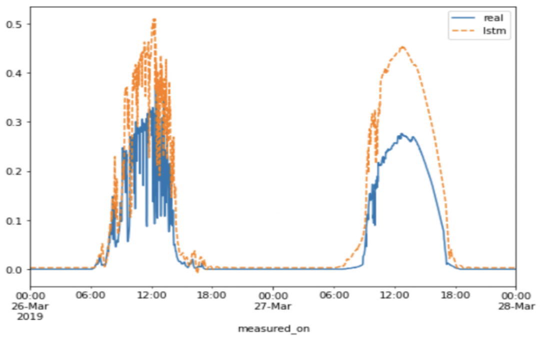


(a) Simulation Results, July 2018

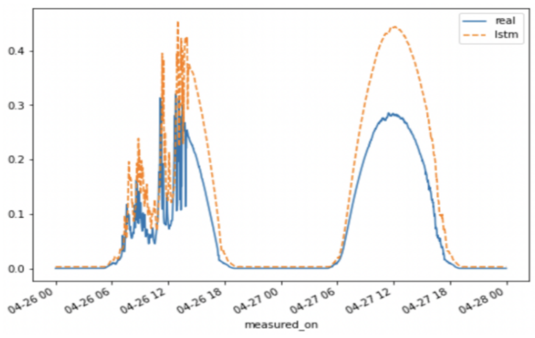


(b) Simulation Results, August 2018

Figure 4.38: Simulation Results, 2018

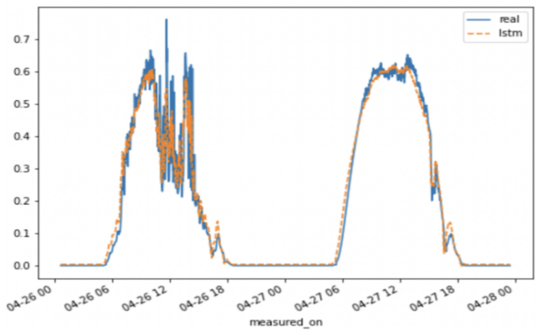


(a) Simulation Results, March 2019

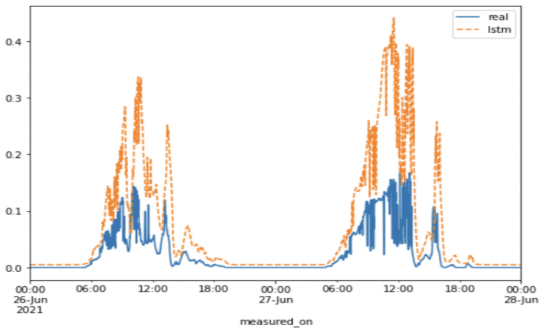


(b) Simulation Results, April 2019

Figure 4.39: Simulation Results, 2019

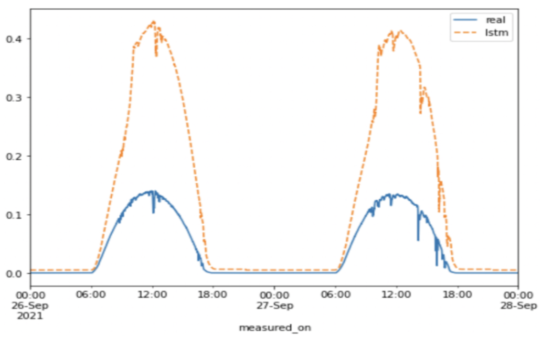


(a) Simulation Results, April 2020

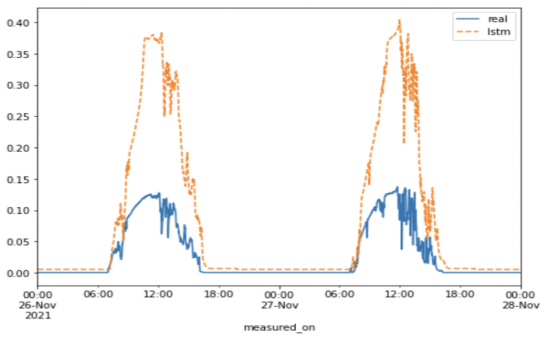


(b) Simulation Results, June 2020

Figure 4.40: Simulation Results, 2020



(a) Simulation Results, September 2021



(b) Simulation Results, November 2021

Figure 4.41: Simulation Results, 2021

Observations

Table 4.6 summarizes the changing difference between actual and predicted values over time due to ageing.

Prediction Month, Year	Prediction view time span(Days)	Window Length(hr)	Prediction Horizon(hr)	Average difference between real and predicted values over the time span
July, 2016	2	6	0.5	0.0264
October, 2016	2	6	0.5	0.0265
July, 2018	2	6	0.5	0.0464
August, 2018	2	6	0.5	0.0477
March, 2019	2	6	0.5	0.0489
April, 2019	2	6	0.5	0.0491
April, 2020	2	6	0.5	0.0273
June, 2020	2	6	0.5	0.0534
September, 2021	2	6	0.5	0.0557
November, 2021	2	6	0.5	0.0562

Table 4.6: Changing difference between actual and predicted values due to ageing

The table summarizes the effect of panel ageing on the accuracy of predicted values. It shows that as the time passes, the difference between actual and predicted values increases, indicating that the accuracy of the prediction model decreases due to panel ageing. The table can be useful in understanding the limitations of the prediction model and making decisions based on the predicted values.

Also, there is an inconsistency in the results for April 2020 compared to the surrounding time points. A possible reason for this inconsistency is that there was a change in the conditions of the panels during that time period, which could have affected the accuracy of the predictions. For example, there could have been a temporary improvement in the environmental conditions, or some maintenance work might have been done on the panels during that time period.

4.5.2. Ageing using Predictive Analysis

In this case, the evaluation of aging is accomplished by employing data spanning the period of 2011-2019 with a sampling interval of 15 minutes. Due to lesser number of available samples in this case, it is advantageous to extend the training duration of the model in order to more effectively capture the underlying trends present within the data. Ageing is assessed by comparing the difference between predicted power and actual power over several years.

4.5.2.1. Pre-processing of Dataset

Initially, the data corresponding to a photovoltaic (PV) system was downloaded, which was stored in a separate file for each day. Subsequently, the files were merged to create distinct datasets for each year. The data was found to have numerous negative and NaN values, along with extraneous columns. Python code was utilized to remove such values and columns. Following the data cleaning process, a dataset analysis was conducted, which revealed the presence of missing and inconsistent values of power output for certain days. To address this issue, the corresponding rows were eliminated from the dataset. Upon completion of data cleaning, the final dataset was analyzed and the results are presented below in Figure 4.42.

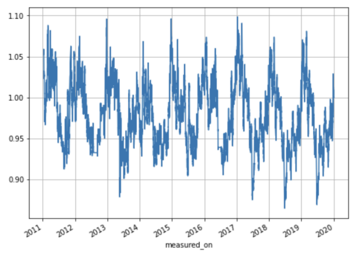
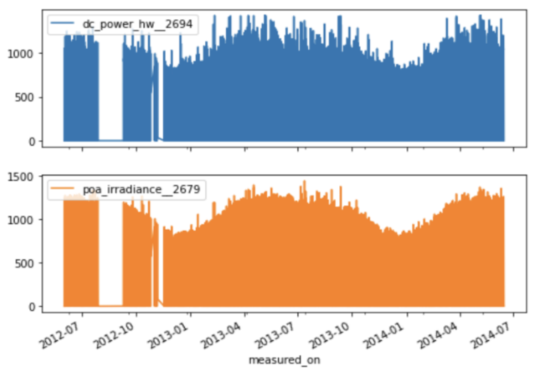


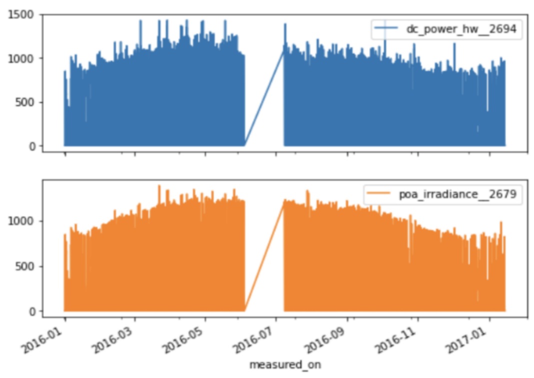
Figure 4.42: DC Power and Irradiance trends

It is clear that DC Power and Solar Irradiance follows an almost fixed pattern over seasons and years.

Figure 4.43 shows the relationship between Power output and solar irradiance, and the data removed during preprocessing.



(a) July 2012 - July 2014



(b) January 2016 - January 2017

Figure 4.43: Relationship between Power and Solar Irradiance, and the data removed during pre-processing

4.5.2.2. Simulation Results

Table 4.7 summarizes the difference between actual and predicted values over time.

Prediction Month, Year	Prediction view time span(Days)	Window Length(hr)	Prediction Horizon(hr)	Average difference between real and predicted values over the time span
January, 2015	10	90	0.5	0.1150
February, 2015	10	90	0.5	0.1283
March, 2015	10	90	0.5	0.1455
April, 2015	10	90	0.5	0.1711
May, 2015	10	90	0.5	0.1609
June, 2015	10	90	0.5	0.1530
July, 2015	10	90	0.5	0.1473
August, 2015	10	90	0.5	0.1411
September, 2015	10	90	0.5	0.1503
October, 2015	10	90	0.5	0.1353
November, 2015	10	90	0.5	0.1342

December, 2015	10	90	0.5	0.1187
January, 2016	10	90	0.5	0.0968
February, 2016	10	90	0.5	0.1432
March, 2016	10	90	0.5	0.1553
April, 2016	10	90	0.5	0.1745
May, 2016	10	90	0.5	0.1583
July, 2016	10	90	0.5	0.1432
August, 2016	10	90	0.5	0.1370
September, 2016	10	90	0.5	0.1474
October, 2016	10	90	0.5	0.1336
November, 2016	10	90	0.5	0.1226
December, 2016	10	90	0.5	0.0932
January, 2017	10	90	0.5	0.1064
February, 2017	10	90	0.5	0.1172
March, 2017	10	90	0.5	0.1507
April, 2017	10	90	0.5	0.1666
May, 2017	10	90	0.5	0.1665
June, 2017	10	90	0.5	0.1411
July, 2017	10	90	0.5	0.1320
August, 2017	10	90	0.5	0.1446
September, 2017	10	90	0.5	0.1505
October, 2017	10	90	0.5	0.1427
November, 2017	10	90	0.5	0.1176
December, 2017	10	90	0.5	0.1112
January, 2018	10	90	0.5	0.1060
February, 2018	10	90	0.5	0.1168
March, 2018	10	90	0.5	0.1501
April, 2018	10	90	0.5	0.1660
May, 2018	10	90	0.5	0.1659
June, 2018	10	90	0.5	0.1408

July, 2018	10	90	0.5	0.1317
August, 2018	10	90	0.5	0.1442
September, 2018	10	90	0.5	0.1499
October, 2018	10	90	0.5	0.1422
November, 2018	10	90	0.5	0.1172
December, 2018	10	90	0.5	0.1092
January, 2019	10	90	0.5	0.1031
February, 2019	10	90	0.5	0.1116
March, 2019	10	90	0.5	0.1464
April, 2019	10	90	0.5	0.1646
May, 2019	10	90	0.5	0.1679
June, 2019	10	90	0.5	0.1477
July, 2019	10	90	0.5	0.1424
August, 2019	10	90	0.5	0.1544
September, 2019	10	90	0.5	0.1546
October, 2019	10	90	0.5	0.1421
November, 2019	10	90	0.5	0.1148
December, 2019	10	90	0.5	0.1092

Table 4.7: Difference between actual and predicted values over time

The corresponding simulation results are shown in Figure 4.44 - 4.48. The results are also graphically represented in Figure 4.49.

The table spans from January 2015 to December 2019, and for each month within that time frame, the table provides the prediction view time span (number of days), window length and prediction horizon (in hours), and the average difference between the actual and predicted values over the time span. Based on table, we can infer that the accuracy of the predictions varies over time. The average difference between the actual and predicted values is quite small for some months (e.g., January 2015, January 2016), while for other months, the difference is relatively large (e.g., April 2015, April 2018).

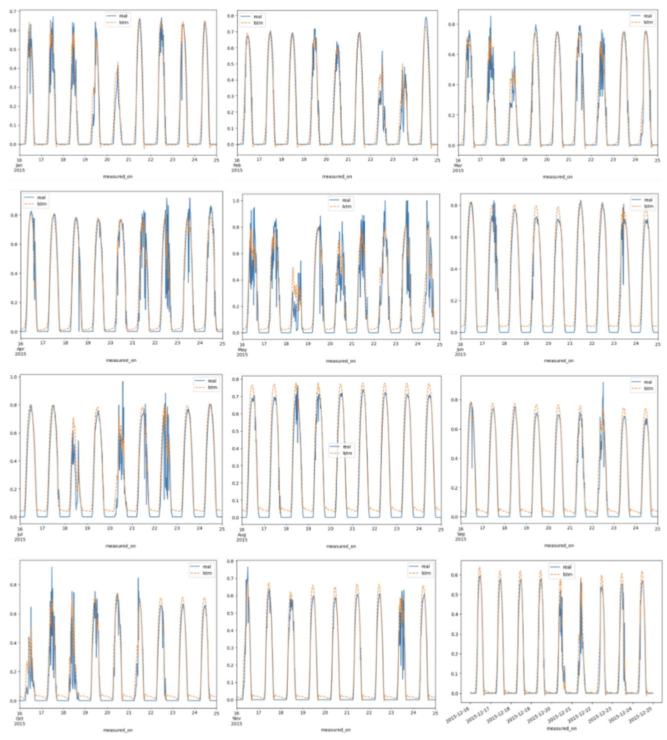


Figure 4.44: Simulation Results, Year 2015

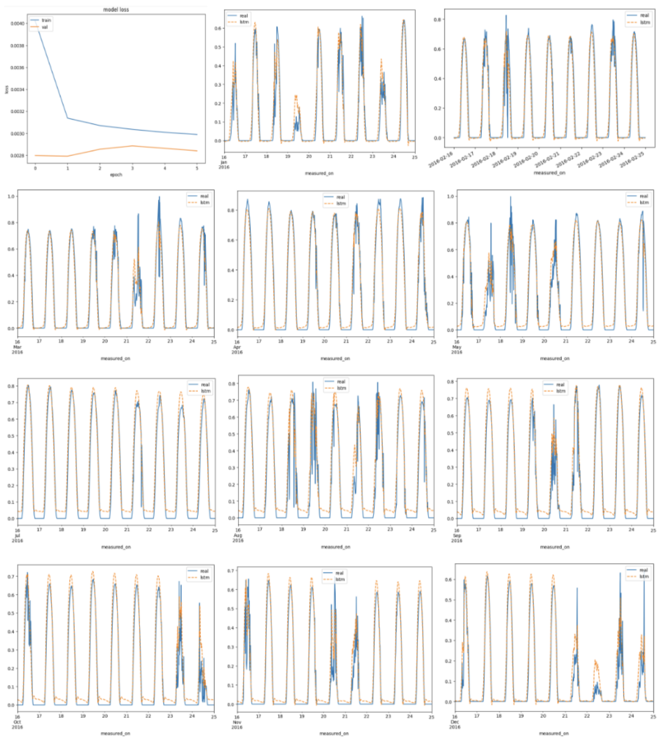


Figure 4.45: Simulation Results, Year 2016

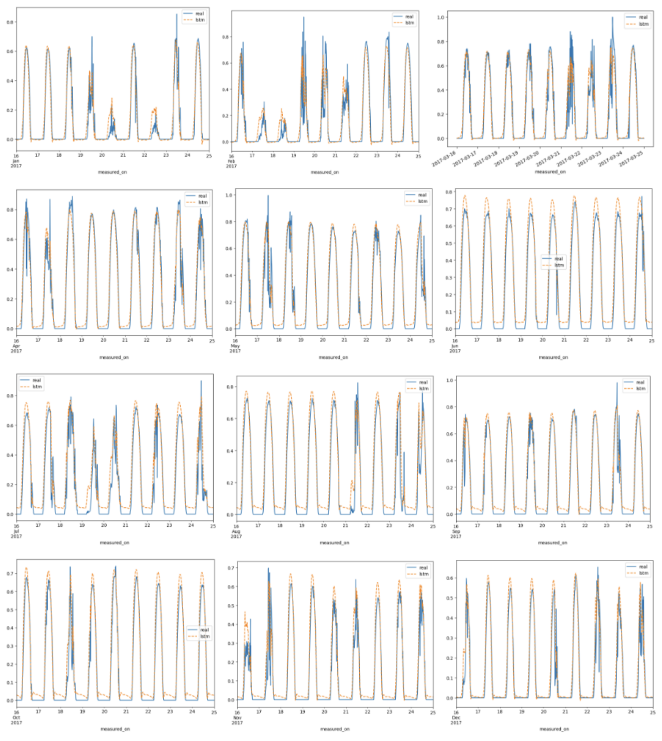


Figure 4.46: Simulation Results, Year 2017

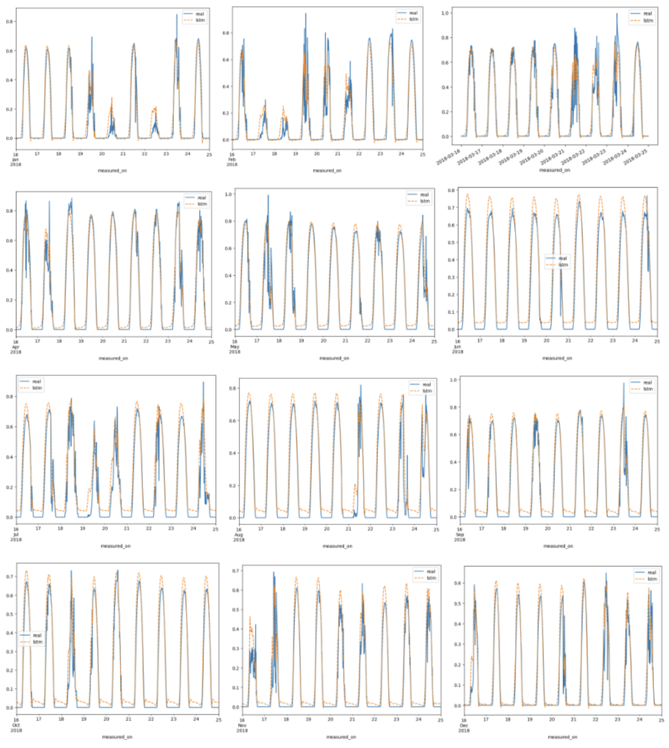


Figure 4.47: Simulation Results, Year 2018

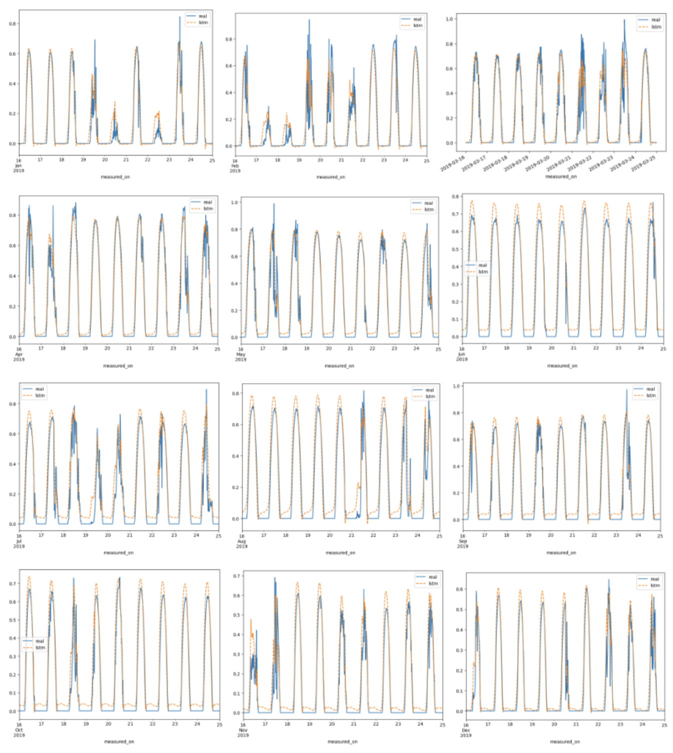


Figure 4.48: Simulation Results, Year 2019

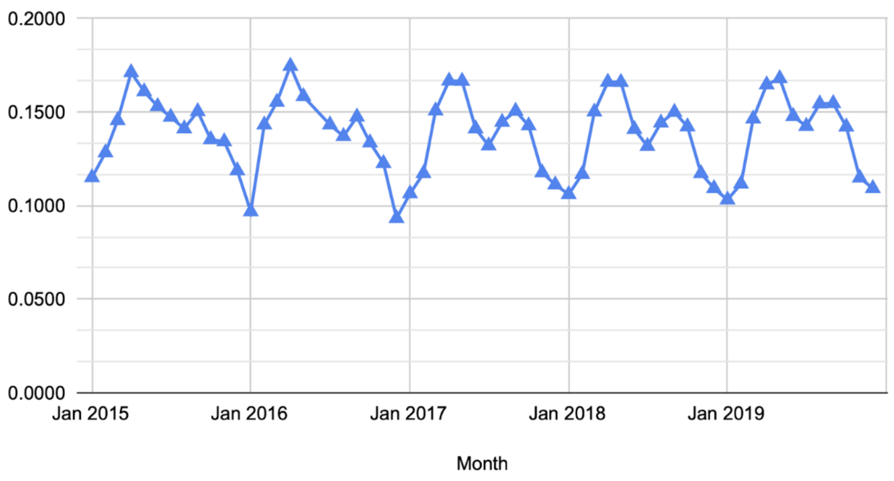


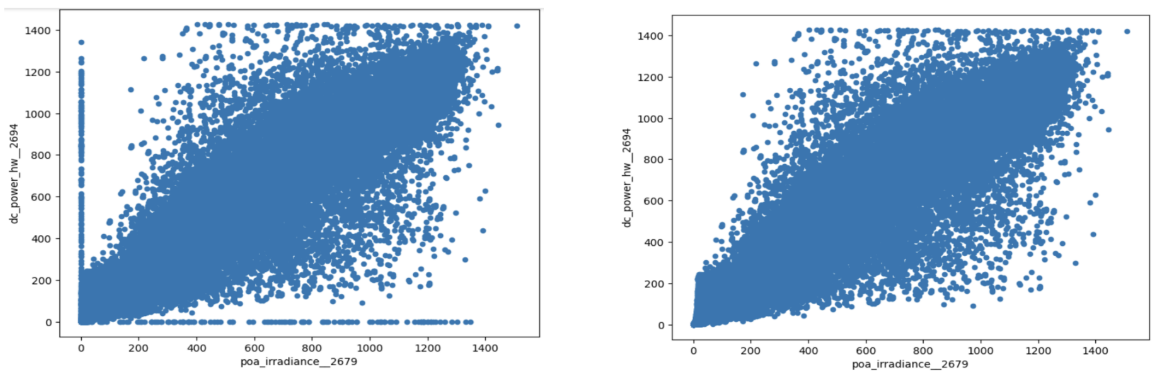
Figure 4.49: Graphical representation of difference between actual and predicted values over months

There are many factors that can contribute to the difference between actual and predicted values in different months. Some of the factors that can influence the accuracy of predictions include changes in external factors, such as weather or maintenance patterns.

Improving prediction accuracy by clustering the dataset for different weather conditions is discussed in detail in section 4.6.2.

4.5.3. Ageing using Data Analysis

Scatter plots are used to study ageing using data analysis. Analyzing solar panel ageing using the slope of a scatter plot between irradiance and power during different years can provide insights into how the panels are performing over time. In general, the slope of a scatter plot between irradiance and power can indicate the efficiency of the solar panel, with a steeper slope indicating a more efficient panel. Over time, however, the slope may change, indicating that the panel's performance is deteriorating due to ageing, damage, or other factors. To analyze the slope of a scatter plot over time, I have plotted the data for each year and calculated the slope for each plot. As it is clear from the scatter plot shown in Figure 4.50(a) that there are outliers in the data and there are some data points on both the vertical and horizontal zero axis, corresponding to data where there either is energy production without sun or there is sun with no energy production. While the latter is possible the former is clearly impossible. In all cases, both cases are removed from the dataset which is shown in the scatter plot in Figure 4.50(b).



(a) (b)

Figure 4.50: Scatter plot of Irradiance vs Power from 2011-2019

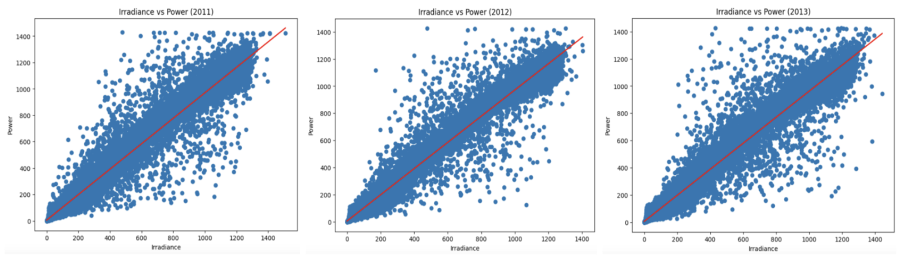
4.5.3.1. Ageing using Linear Interpolation

Table 4.8 shows the value of slope, intercept and R-value corresponding to the plots in Figure 4.51.

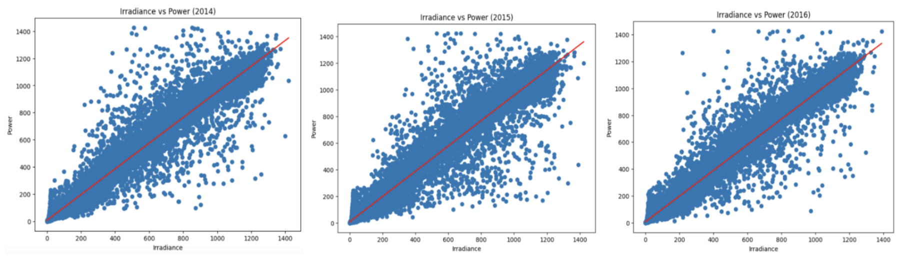
To analyze ageing, a scatter plot corresponding to each year along with the interpolating line is shown in Figure 4.51.

Year	Slope	Intercept	R-Value
2011	0.9642	5.8532	0.9845
2012	0.9661	5.7446	0.9863
2013	0.9561	5.7877	0.9837
2014	0.9468	6.3851	0.9848
2015	0.9507	7.0541	0.9806
2016	0.9567	6.0614	0.9851
2017	0.9497	6.1416	0.9851
2018	0.9431	4.6355	0.9850
2019	0.9419	4.4810	0.9849

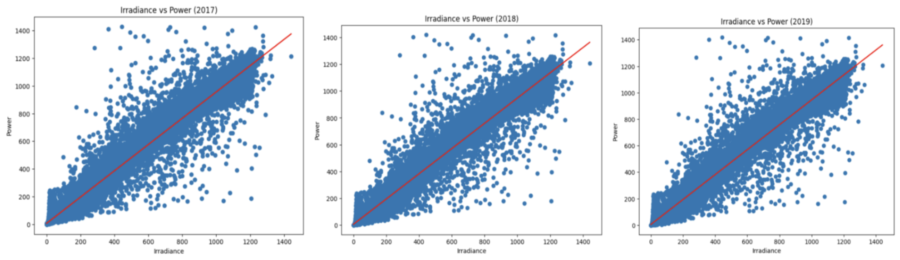
Table 4.8: Slope, intercept and R-value corresponding to the plots in Figure 4.51



(a)



(b)



(c)

Figure 4.51: Scatter plot of irradiance vs power from 2011-2019 with the interpolating line

The slope for each year is graphically represented in Figure 4.52.

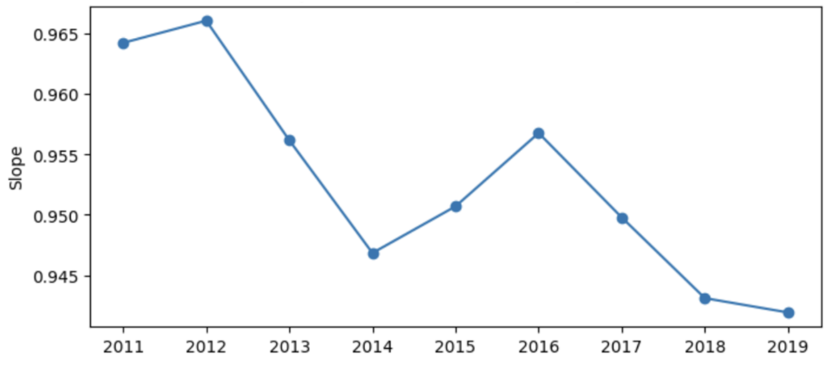


Figure 4.52: Slope of each year with linear interpolation

The graph and table represents the results of a linear regression analysis performed on a time series dataset consisting of 9 years of observations of solar irradiance and power. The slope, intercept, and R-value are calculated for each year to describe the linear relationship between the two variables. The slope represents the rate of change in power for a unit change in irradiance, while the intercept represents the estimated value of power when irradiance is zero. The R-value, also known as the correlation coefficient, is a measure of the strength and direction of the linear relationship between the two variables. An R-value of 1 indicates a perfect positive correlation, while an R-value of -1 indicates a perfect negative correlation. An R-value of 0 indicates no correlation between the two variables.

The R-values ranging from 0.9806 to 0.9863 indicate a strong positive linear correlation between the two variables. This means that as irradiance increases, power output also increases. This suggests that there is a direct relationship between the amount of solar radiation received and the power output.

The slope values for each year, ranging from 0.9419 to 0.9661, indicate that the rate of change in power output per unit change in irradiance is relatively consistent over time. This suggests that the relationship between irradiance and power output is stable and reliable. The intercept and slope values for each year are not identical, indicating that there are yearly differences in the relationship between irradiance and power output. This could be due to various factors such as changes in weather patterns, maintenance of equipment, or modifications to the system.

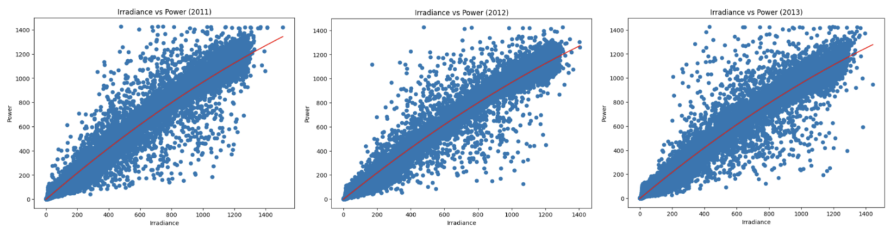
If panel aging occurred, it could potentially affect the slope values over time, as the performance of solar panels can decline with age due to factors such as weathering, degradation of materials, or the accumulation of dirt and debris.

Based on the data in the table, there is a clear evidence of a consistent decrease in slope values over the 9-year period covered by the data. While there is some variability in slope values from year to year but there is a certain trend indicating a decrease in slope values over time. Factors such as maintenance practices, changes in environmental conditions, and changes in technology could also potentially affect the performance of solar panels over time.

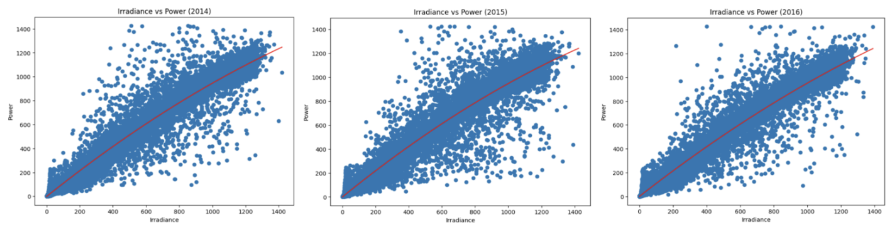
4.5.3.2. Ageing using Quadratic Interpolation

Quadratic interpolation can be used to estimate the relationship between solar irradiance and power output. This technique involves fitting a quadratic curve to the data points, which can then be used to estimate the power output at any given level of solar irradiance.

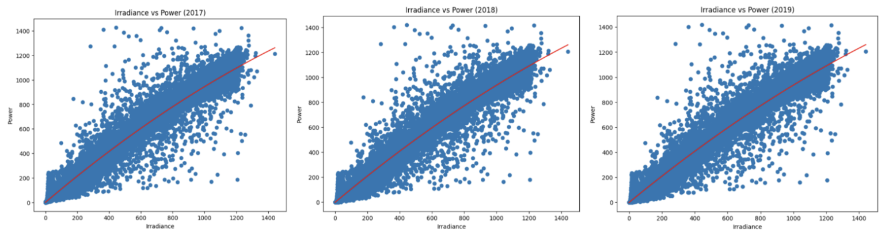
To analyze ageing and relationship between irradiance and power, a scatter plot corresponding to each year along with the quadratic interpolation is shown in Figure 4.53.



(a)



(b)



(c)

Figure 4.53: Scatter plot of irradiance vs power with quadratic interpolation

The quadratic equation for each year is given in Table 4.9.

Year	Quadratic fit
2011	$y = -0.00015x^2 + 1.12x - 1.70$
2012	$y = -0.00015x^2 + 1.13x - 1.90$
2013	$y = -0.00016x^2 + 1.14x - 1.91$
2014	$y = -0.00016x^2 + 1.11x - 1.30$
2015	$y = -0.00018x^2 + 1.13x - 1.00$
2016	$y = -0.00016x^2 + 1.10x - 0.41$
2017	$y = -0.00016x^2 + 1.10x - 0.73$
2018	$y = -0.00015x^2 + 1.08x - 1.61$
2019	$y = -0.00015x^2 + 1.08x - 1.63$

Table 4.9: Quadratic equation for each year

The quadratic term (ax^2) represents the curvature of the relationship between irradiance and power output. In the case of these equations, a is negative, which means that the relationship between irradiance and power is "inverted U-shaped" or "downward-sloping". This implies that the power output initially increases as the irradiance level increases, but eventually reaches a maximum and then decreases as the irradiance level continues to increase.

The magnitude of the coefficient of x^2 is also important. The values provided range from -0.00014 to -0.00018, which means that the curvature of the relationship is relatively small. In other words, the maximum power output is not very far from the point where the rate of increase in power output begins to slow down. The linear term (bx) represents the rate of change in power output with respect to irradiance. In the case of these equations, b is generally positive but relatively small in magnitude (between 1.08 and 1.13). This means that the power output increases as the irradiance level increases, but not very rapidly.

The constant term (c) represents the power output when the irradiance level is zero. In other words, it represents the "baseline" power output of the PV panel. The values provided range from -1.91 to -0.41, which means that the baseline power output is negative but relatively small in magnitude.

The quadratic equations provided suggest that the power output of PV panels increases as the irradiance level increases, but eventually reaches a maximum and then decreases. The rate of increase in power output is not very rapid, and the baseline power output is relatively small. These equations can be used to model the performance of PV panels over time and optimize their operation.

Moreover, to understand the ageing, years vs PV power output graph considering quadratic fitting is plotted in Figure 4.54 for a fixed value of irradiance (1200 W/m2).

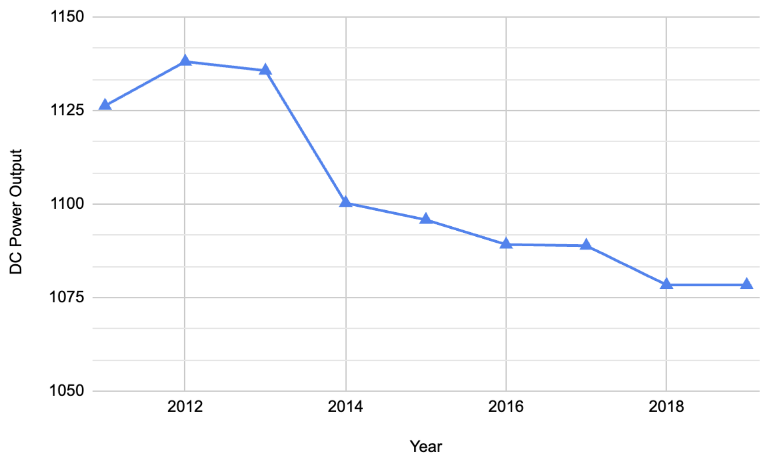


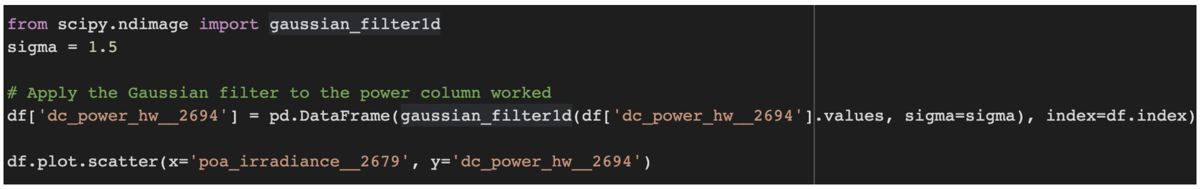
Figure 4.54: Power Output each year with quadratic interpolation with Irradiance = 1200 W/m2

It is evident that the output power is gradually decreasing each year with the same value of irradiance, as observed through linear interpolation. This indicates that a quadratic fit may provide a more suitable description of the correlation between irradiance and power, and also aid in assessing the effects of ageing.

4.5.3.3. Ageing using Quadratic Interpolation after removing noise

A Gaussian filter is applied with a sigma value of 1.5 to the DC Power column. The Gaussian filter is a smoothing filter that can be used to remove noise or reduce the impact of outliers in a dataset.

The sigma value in a Gaussian filter determines the width of the Gaussian distribution used to smooth the data. Specifically, it determines the standard deviation of the Gaussian distribution. A larger sigma value will result in a wider distribution and a smoother output. When the sigma value is small, the Gaussian filter only smooths out high-frequency noise in the data, leaving the low-frequency information intact. However, as the sigma value increases, the filter begins to smooth out lower frequency information as well, resulting in a loss of detail in the data. The algorithm used to apply the gaussian filter is shown in Figure 4.55

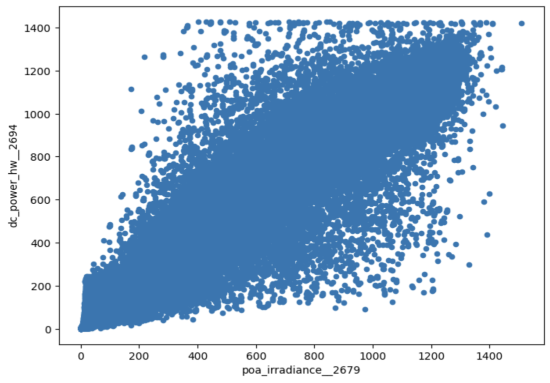


```
from scipy.ndimage import gaussian_filter1d
sigma = 1.5

# Apply the Gaussian filter to the power column worked
df['dc_power_hw__2694'] = pd.DataFrame(gaussian_filter1d(df['dc_power_hw__2694'].values, sigma=sigma), index=df.index)

df.plot.scatter(x='poa_irradiance__2679', y='dc_power_hw__2694')
```

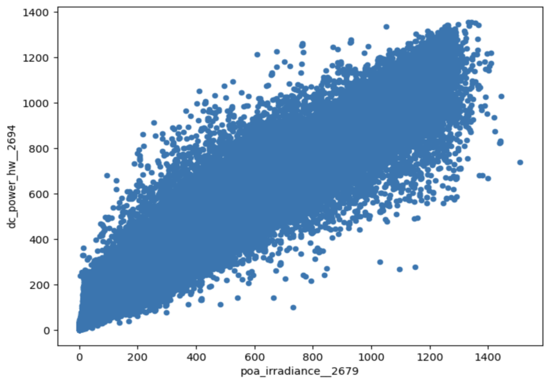
Figure 4.55: Gaussian filter algorithm

Therefore, the choice of sigma value depends on the nature of the data and the degree of smoothing required. If the data contains a lot of high-frequency noise, a smaller sigma value may be appropriate. On the other hand, if the data is relatively smooth and we want to remove outliers or reduce noise without losing important details, a larger sigma value may be more appropriate.

It is also important to note that excessively large sigma values can result in oversmoothing and loss of important features in the data. Therefore, After experimenting with different sigma values to find the optimal value for this specific dataset, Sigma = 1.5 has been used. The scatter plot before and after applying this filter over the whole dataset is shown in Figure 4.56.



(a) Without filter

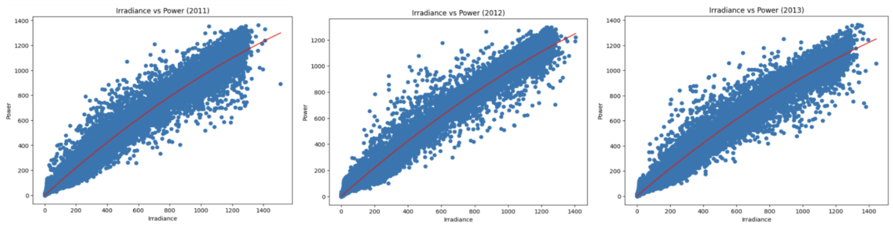


(b) With filter

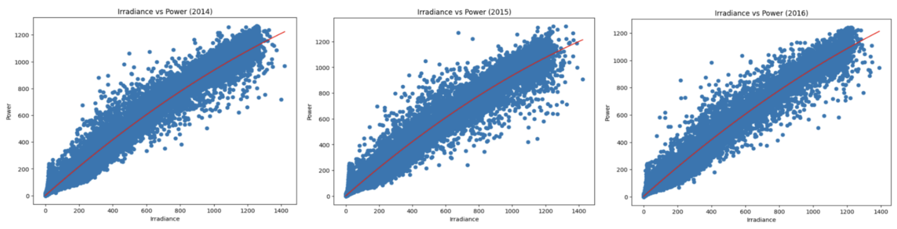
Figure 4.56: Scatter plot of Irradiance vs Power from 2011-2019

The application of a Gaussian filter has effectively reduced outliers and noise within the dataset, as shown in the figures.

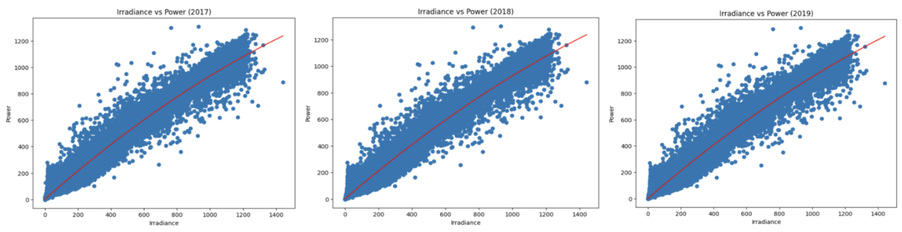
To analyze ageing and relationship between irradiance and power after applying the filter and removing the noise significantly, a scatter plot corresponding to each year along with the quadratic interpolation is shown in Figure 4.57.



(a)



(b)



(c)

Figure 4.57: Scatter plot of Irradiance vs Power with Gaussian filter and quadratic interpolation

The quadratic equation for each year after applying filter is given in Table 4.10.

Year	Quadratic fit
2011	$y = -0.00018x^2 + 1.12x + 1.05$
2012	$y = -0.00017x^2 + 1.13x + 0.82$
2013	$y = -0.00018x^2 + 1.12x + 0.87$
2014	$y = -0.00018x^2 + 1.11x + 1.26$
2015	$y = -0.00019x^2 + 1.12x + 1.78$
2016	$y = -0.00018x^2 + 1.10x + 2.26$
2017	$y = -0.00019x^2 + 1.10x + 2.02$
2018	$y = -0.00018x^2 + 1.08x + 1.07$
2019	$y = -0.00018x^2 + 1.08x + 1.02$

Table 4.10: Quadratic equation for each year after filter

Moreover, to understand the effect of applying the filter on ageing, years vs PV power output graph considering quadratic fitting is plotted in Figure 4.58 for a fixed value of irradiance (1200 W/m2).

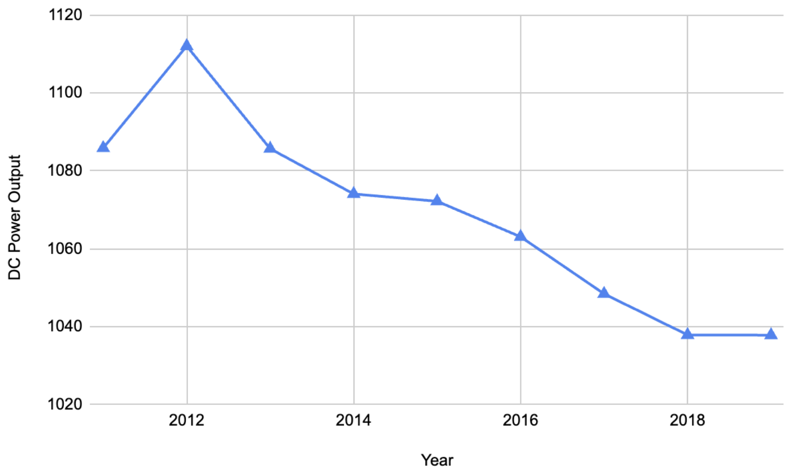


Figure 4.58: Power Output each year with Gaussian filter and quadratic interpolation with Irradiance = 1200 W/m2

Observations using quadratic interpolation after applying filter on the output power with constant irradiance have revealed a gradual annual decrease. Although the application of a filter has successfully removed noise from the dataset, the similar trend of decrease in output power suggests an aging effect that remains unaffected.

4.6. Model Performance Improvements

Two different approaches are used for improving prediction accuracy. The first approach involves modifying the best performing architecture, while the second approach involves using a dataset clustering approach. Both approaches have their own benefits and limitations.

4.6.1. Architectural Modifications Using LSTNet

LSTnet (Long-Short-Term network) is a deep learning model that combines the strengths of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) for time-series forecasting. The model was proposed in the paper "LSTNet: Learning Long- and Short-term Dependencies Together for Precipitation Forecasting" by Lai et al. (2018) in [23]. The model consists of three main components: a CNN encoder, an RNN decoder, and a skip-connection network. The CNN encoder is used to extract features from the input time-series data, the RNN decoder is used to model the temporal dependencies, and the skip-connection network is used to help preserve the information from the input directly to the output. The basic architecture of LSTNet is shown in Figure 4.59.

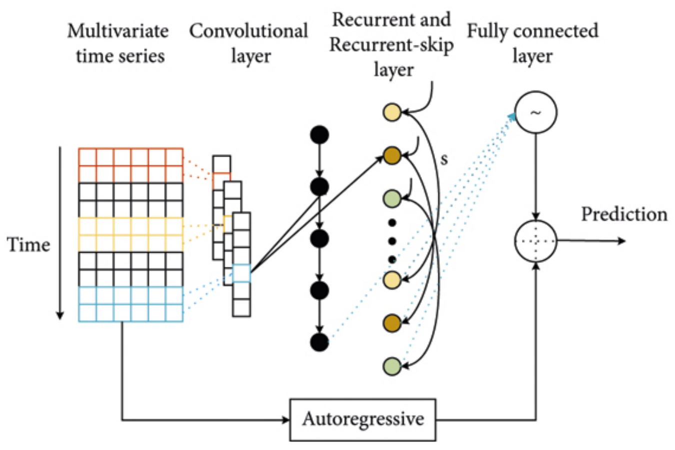


Figure 4.59: Basic architecture of LSTNet

The detailed working of model is described as follows:

CNN Encoder: The input time-series data is first passed through a CNN encoder, which extracts local patterns from the input time-series data. The encoder consists of a series of convolutional layers, which are followed by a max-pooling layer. The output of the encoder is a sequence of feature maps, which represent the extracted local patterns.

RNN Decoder: The sequence of feature maps from the CNN encoder is then fed into an RNN decoder, which models the temporal dependencies between the extracted features. The RNN decoder is typically a type of gated RNN, such as an LSTM or GRU, which is capable of modeling long-term dependencies. The output of the RNN decoder at each time step is given by:

$$h_t = f(h_{t-1}, y_{t-1}) \tag{4.4}$$

where h_t is the hidden state at time t, h_{t-1} is the hidden state at the previous time step, and y_{t-1} is the output at the previous time step. f is the recurrent function, which is defined as:

$$f(h_{t-1}, y_{t-1}) = \text{LSTM}(h_{t-1}, y_{t-1}) \tag{4.5}$$

where LSTM is the Long Short-Term Memory cell.

Skip-connection Network: The skip-connection network is used to help preserve the information from the input directly to the output. The skip-connection network consists of a series of fully connected layers, which are used to combine the output of the CNN encoder with the output of the RNN decoder. The final output of the model is given by:

$$\hat{y}_t = g(h_t, s_t) \tag{4.6}$$

where $\hat{y}_t$ is the predicted value at time t, h_t is the hidden state of the RNN decoder at time t, and s_t is the skip connection output at time t. g is a fully connected layer, which is used to combine the two inputs.

Overall, LSTnet is a powerful and effective model for time-series forecasting, which is capable of capturing both long-term and short-term dependencies in the data.

The simulations results corresponding to LSTNet are shown in Figure 4.60.

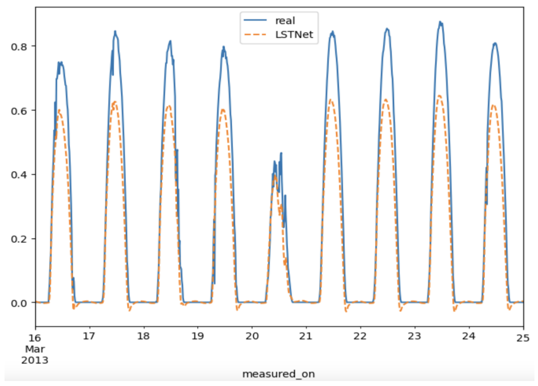
Comparison with LSTM

The comparison between simulation results of LSTNet and LSTM as a function of test set loss are shown in Table 4.11.

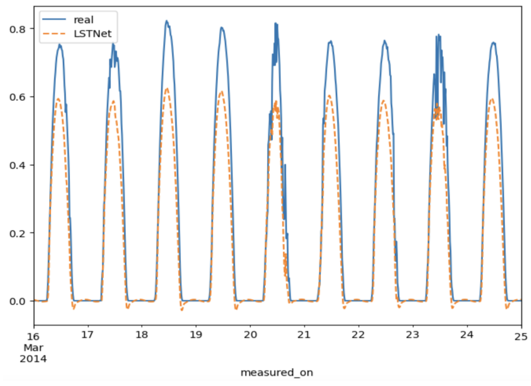
Month, Year	LSTNet Test Loss	LSTM Test Loss
March, 2013	0.0661	0.0674
March, 2014	0.0638	0.0646
March, 2015	0.0602	0.0567
March, 2016	0.0600	0.0632
March, 2017	0.0628	0.0657
March, 2018	0.0617	0.0606

Table 4.11: Comparison between LSTNet and LSTM results

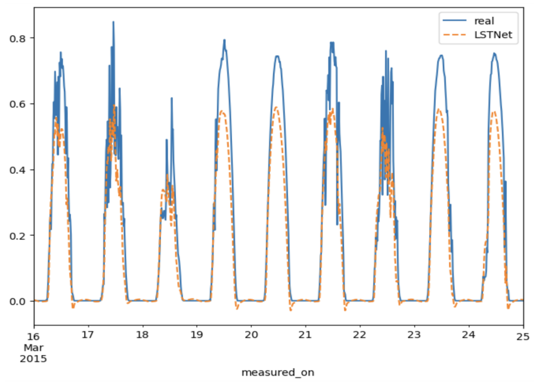
LSTNet generally exhibits slightly better performance compared to LSTM. Across most years, LSTNet consistently achieves lower test loss values, indicating its superior predictive capability. However, there are instances where LSTM outperforms LSTNet. Specifically, in March 2015 and March 2018, LSTM achieved marginally lower test loss values compared to LSTNet. This suggests that LSTM may have had a better ability to capture the underlying patterns and dependencies in the data during those particular years. While the differences in test loss values between the models are relatively small, these instances indicate that LSTM can occasionally exhibit competitive performance with LSTNet. The choice between the two models may depend on specific requirements, such as the dataset characteristics or the desired balance between accuracy and computational efficiency. Further analysis and experimentation is necessary to gain a comprehensive understanding of the models' relative strengths and weaknesses.



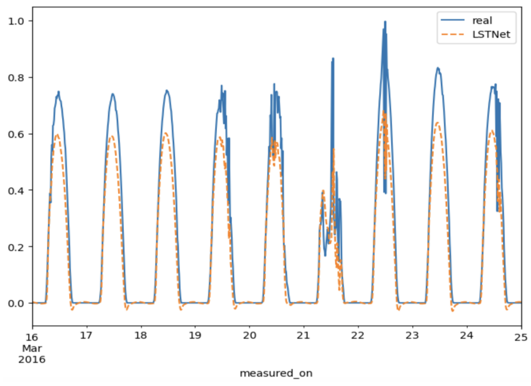
(a)



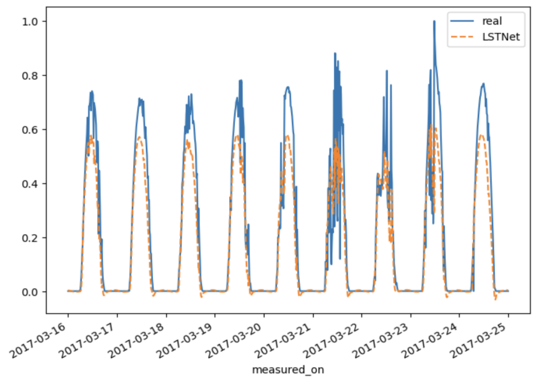
(b)



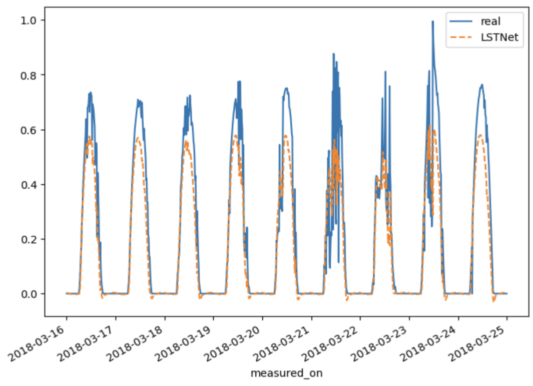
(c)



(d)



(e)



(f)

Figure 4.60: Simulation results from 2013-2018 using LSTNet

4.6.2. Dataset Clustering

The objective of dataset clustering is to systematically aggregate days exhibiting analogous patterns in photovoltaic (PV) power generation relative to solar irradiance. This can help to reduce the variability in the data and make it easier for the model to learn patterns [39]. Here are the basic steps involved in this approach:

Preprocess the data: Before clustering the dataset, it is important to preprocess the data by normalizing it and removing any outliers or missing values.

Select a clustering algorithm: There are many different clustering methods to choose from. The choice of the method will depend on the size of the dataset, the desired number of clusters, and the computational resources available.

Cluster the data: Once the algorithm has been defined, the dataset can be clustered using the selected algorithm. The output of this step will be a set of clusters, each containing days with similar trends in PV power generation.

Train a separate model for each cluster: Finally, a separate model can be trained for each cluster using the days within that cluster. This can help to improve the accuracy of the models, since they will be trained on data with lower variability.

To address the challenge of uncertainty in power values on a day-to-day basis, the dataset is divided into two categories: sunny days and overcast days. Each category is then trained separately using dedicated models. This approach involves the use of two distinct models, one specifically trained for sunny days and another for overcast days. The model trained specifically on sunny days is employed to forecast power values under sunny conditions, whereas the model trained on overcast days is utilized for power prediction during overcast conditions. By grouping days with similar power production characteristics into the same category, the variability within each set is reduced. This clustering approach allows for a more targeted and accurate prediction by tailoring the models to specific weather conditions.

Daily mean irradiance is used to split the dataset. To split the PV power dataset into cloudy and sunny days using daily mean irradiance, the daily mean irradiance values for each day in the dataset are obtained. The daily mean irradiance is usually calculated by averaging the irradiance values over a day. Then, a threshold value is determined that will serve as the cutoff between cloudy and sunny days. This threshold value depends on the specific dataset.

In order to determine the most suitable threshold value, various thresholds were evaluated.

The daily mean irradiance value for each day was compared to the threshold value. If the irradiance value exceeded the threshold, the day was classified as sunny. Conversely, if the irradiance value fell below the threshold, the day was classified as cloudy. This classification process resulted in the creation of a new dataset where days were grouped into either sunny or cloudy categories based on the threshold comparison.

Flowchart of the above process is illustrated in Figure 4.61.

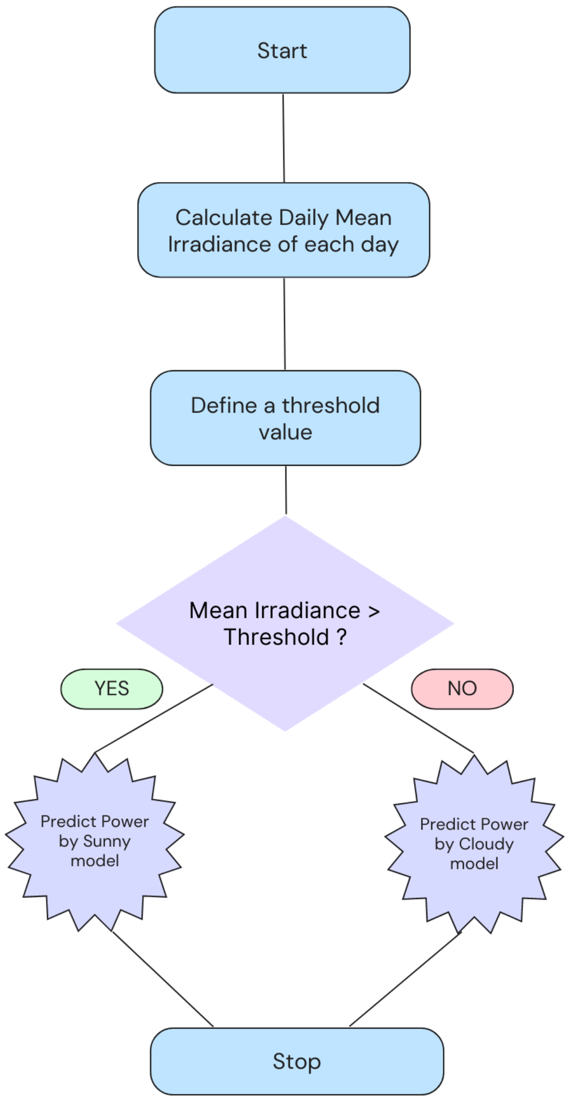
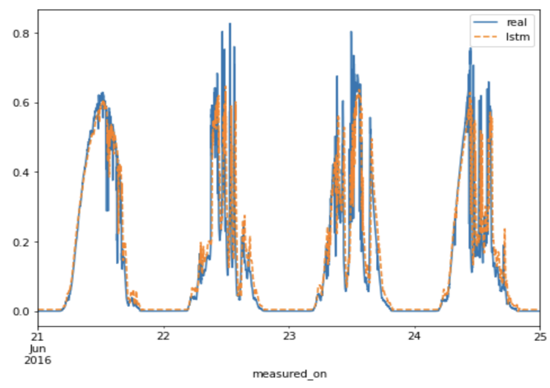

Start
Calculate Daily Mean Irradiance of each day
Define a threshold value
Mean Irradiance > Threshold ?
YES
NO
Predict Power by Sunny model
Predict Power by Cloudy model
Stop

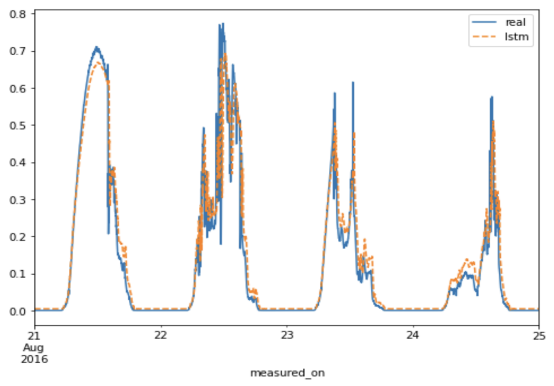
Figure 4.61: Flowchart of dataset clustering approach

Simulation Results

The simulation results for cloudy and sunny days with a threshold value of mean daily irradiance equal to 250 W/m^2 are shown below from Figures 4.62 - 4.63.

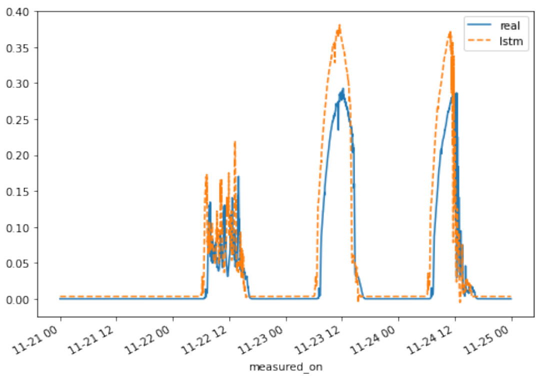


(a) June 2016, MAE = 0.0324

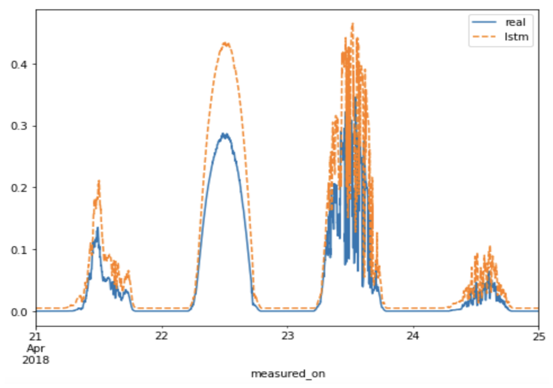


(b) August 2016, MAE = 0.0371

Figure 4.62: Sunny Days



(a) November 2017, MAE = 0.0621



(b) April 2018, MAE = 0.0779

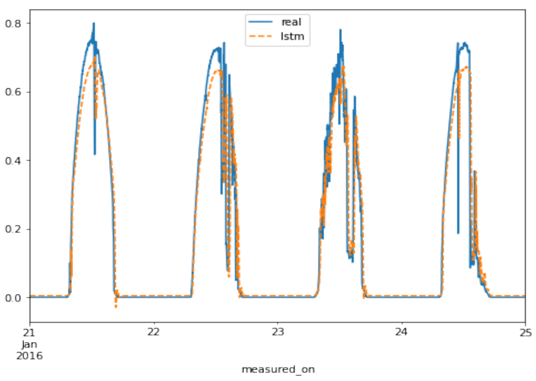
Figure 4.63: Cloudy Days

The clustering of the dataset has been shown to effectively reduce prediction errors. It is worth noting that the predictive accuracy is significantly lower for overcast days compared to sunny days. One possible explanation for this disparity is the substantial variability in power values within the overcast dataset, whereas the sunny dataset tends to exhibit more consistent patterns. Consequently, training models on the sunny dataset leads to more repetitive and reliable forecasts.

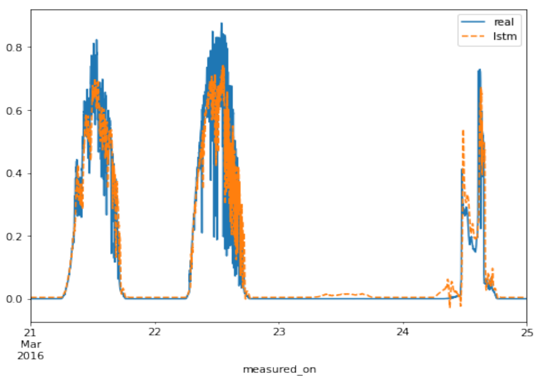
As a result, during cloudy weather conditions, model performance tends to suffer due to the inherent variability in both power and irradiance values within the dataset, making prediction task more challenging.

The performance of utilizing a model trained on one weather condition (sunny or overcast) for predicting power values on the opposite condition (overcast or sunny, respectively) can also be assessed.

Figures 4.64 - 4.65 shows the simulation results of employing a model trained on one weather condition for predicting power values on the opposite condition.

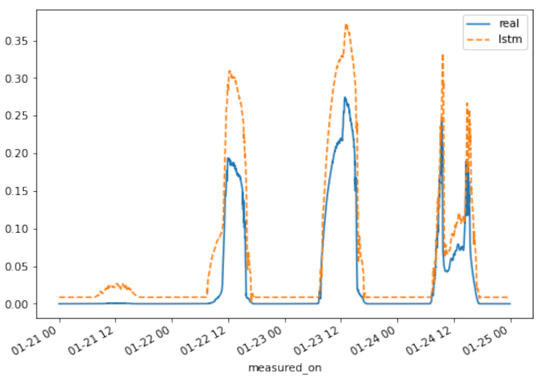


(a) January 2016, MAE = 0.0293

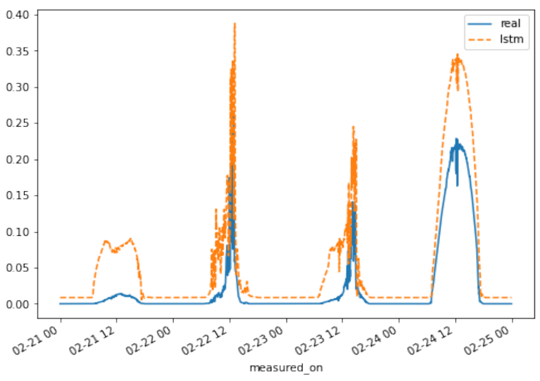


(b) March 2016, MAE = 0.0378

Figure 4.64: Cloudy day model used for sunny days



(a) January 2017, MAE = 0.0736



(b) February 2017, MAE = 0.0854

Figure 4.65: Sunny day model used for cloudy days

The results demonstrate that using a model trained on the sunny dataset, to predict power values on overcast days led to higher MAE compared to its performance on sunny days. Similarly, utilizing a model trained on the overcast dataset, for power prediction on sunny days resulted in higher MAE compared to its performance on overcast days.

These findings indicate that there are notable differences in power production characteristics between sunny and overcast conditions. The models trained on their respective datasets have learned specific patterns and relationships relevant to the corresponding weather conditions. When applied to opposite weather conditions, the models struggled to capture the nuanced dynamics, leading to decreased performance.

In conclusion, utilizing a model trained on one weather condition to predict power values on the opposite condition resulted in decreased performance compared to using the models on their original datasets. These findings highlight the importance of tailoring models to specific weather conditions for accurate power predictions. Therefore, it is recommended to employ separate models trained specifically for sunny and overcast conditions to address the challenge of uncertainty in power values on a day-to-day basis effectively.

5 | Conclusions and Future Developments

The objective of this research was to develop an algorithm capable of accurately predicting the power generation from a solar photovoltaic (PV) system which is crucial for optimizing the performance of solar power systems. To accomplish this, various machine learning models were explored, including recurrent neural networks (RNNs), convolutional neural networks (CNNs), and a hybrid architecture.

Simulations were conducted to evaluate the performance of these models and determine the most accurate prediction. The hyperparameters of the models such as learning rate, number of epochs, and window length, while examining different prediction horizons using two datasets with varying sampling intervals of 1 minute and 15 minutes, were also fine-tuned to identify the optimal set of parameters that would yield the best performance on the datasets. The results of the simulations demonstrated that the LSTM model outperformed all other networks in terms of prediction accuracy. The findings revealed a notable decrease in prediction accuracy as the prediction horizon increased, indicating the increased challenge of forecasting with longer timeframes.

The study also involved a meticulous investigation into the aging of PV systems. Two methods were employed to analyze aging effects. Firstly, data was collected from PV panels that were installed some time ago (referred to as "old data" relative to the present day), and the model's performance was compared with more recent data. It was expected that predictions following the data used for model creation would be more accurate, while predictions further into the future would overestimate actual data. Consequently, the difference between predicted and actual values over multiple years was examined to observe the impact of aging. The results demonstrated an increasing disparity between predicted and actual values over time, indicating a decrease in prediction model accuracy due to panel aging. The second method employed in the aging analysis involved calculating the slope of the scatter plot between irradiance and power over different years. The results indicated a gradual decrease in the output power for each year, despite the irradiance remaining constant. This observation further supports the notion of aging affecting the performance of PV systems.

Furthermore, a clustering approach was employed to classify the dataset into sunny and cloudy days. Individual prediction models were then developed for each category, leading to an improvement in prediction accuracy. Through training and testing the prediction model on separate datasets that represent distinct climate conditions, it was observed that locations with more consistent and stable sunny weather conditions, with fewer occurrences of cloudy days throughout the year, achieved higher prediction accuracy. This finding can be attributed to the increased variability in power and solar irradiance within datasets characterized by unstable weather conditions, which poses a greater challenge for accurate predictions.

Therefore, inference from the results suggests that LSTM architecture is the most effective model for accurately predicting power generation from a solar PV system. The research findings indicate that increasing the prediction horizon poses a challenge to forecasting accuracy, highlighting the importance of considering shorter prediction intervals for more precise predictions. Furthermore, the study reveals that longer window lengths contribute to improved prediction accuracy, emphasizing the significance of selecting appropriate window lengths during model training. Additionally, the analysis of PV system aging provides valuable insights into the deterioration of prediction accuracy over time, as the difference between actual and predicted values increases with panel aging. This suggests the need for periodic recalibration or retraining of the prediction model to account for the changing characteristics of aging PV panels.

Future implications of this research are significant for the optimization and performance enhancement of solar power systems. The identified superior performance of LSTM architecture in power output prediction can serve as a benchmark for future studies and industry applications. These findings can guide the development of more accurate and reliable prediction models, leading to improved operational efficiency and better resource planning for solar energy installations. The understanding that prediction accuracy decreases with panel aging highlights the importance of monitoring and maintenance practices in ensuring the long-term viability and productivity of PV systems. Moreover, the analysis of climate conditions and its impact on prediction accuracy provides insights into the challenges associated with unstable weather patterns and emphasizes the need for tailored prediction models that account for regional variations. Overall, the research outcomes contribute to the advancement of renewable energy technologies and pave the way for future developments in solar power forecasting and system optimization.

Bibliography

[1] R. Ahmed, V. Sreeram, Y. Mishra, and M. Arif. A review and evaluation of the state-of-the-art in pv solar power forecasting: Techniques and optimization. *Renewable and Sustainable Energy Reviews*, 124, 2020. doi: 10.1016/j.rser.2020.109792.

[2] A. Aussem and F. Murtagh. Dynamical recurrent neural networks — towards environmental time series prediction. *International Journal of Neural Systems*, 6:145–170, 1995. doi: 10.1142/s0129065795000123.

[3] S. Bai, J. Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint*, 2018.

[4] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.

[5] S. Bhatti and H. Manzoor. Machine learning for accelerating the discovery of high performance low-cost solar cells: a systematic review. *arXiv preprint*, 2022. doi: 10.48550/arXiv.2212.13893.

[6] D. T. Bui. Hybrid intelligent model based on least squares support vector regression and artificial bee colony optimization for time-series modeling and forecasting horizontal displacement of hydropower dam. *Handbook of neural computation, Academic Press*, pages 279–293, 2017.

[7] K. Bui, T. Bui, J. Zou, V. D. C, and R. I. A novel hybrid artificial intelligent approach based on neural fuzzy inference model and particle swarm optimization for horizontal displacement modeling of hydropower dam. *Neural Comput Appl*, 29: 1495–1506, 2016.

[8] T. Bui, V. Nhu, and N. Hoang. Prediction of soil compression coefficient for urban housing project using novel integration machine learning approach of swarm intelligence and multi-layer perceptron neural network. *Adv Eng Inf*, 38:144–152, 2018.

[9] C. Chen, S. Duan, T. Cai, and B. Liu. Online 24-h solar power forecasting based

on weather type classification using artificial neural network. *Solar energy*, 85(11): 2856–2870, 2011. doi: 10.1016/j.solener.2011.08.027.

[10] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint*, 2014.

[11] U. K. Das and K. S. Tey. Forecasting of photovoltaic power generation and model optimization: A review. *Renewable and Sustainable Energy Reviews*, 81:912–928, 2018. doi: 10.1016/j.rser.2017.08.017.

[12] M. David, F. Ramahatana, P. Trombe, and P. Lauret. Probabilistic forecasting of the solar irradiance with recursive arma and garch models. *Sol Energy*, 133:55–72, 2016.

[13] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. *Neural Comput Appl*, 1999. doi: 10.1049/cp:19991218.

[14] L. Gigoni, A. Betti, E. Crisostomi, and A. Franco. Day-ahead hourly forecasting of power generation from photovoltaic plants. *IEEE Transactions on Sustainable Energy*, 9:831–842, 2018. doi: 10.1109/TSTE.2017.2762435.

[15] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT press, 2016.

[16] A. Graves. Long short-term memory. *Supervised sequence labeling with recurrent neural networks*, pages 37–45, 2012. doi: 10.1007/978-3-642-24797-2_2.

[17] H. Guang and Z. Qin. Extreme learning machine: A new learning scheme of feedforward neural networks. *IEEE International Conference on Neural Networks*, 2: 985–990, 2004. doi: 10.1109/IJCNN.2004.1380068.

[18] M. A. Hall and L. A. Smith. Feature selection for machine learning: comparing a correlation-based filter approach to the wrapper. *FLAIRS conference*, 1999:235–239, 1999.

[19] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[20] T. Hegazy, P. Fazio, and O. Moselhi. Developing practical neural network applications using back-propagation. *Sol Energy*, 9:145–159, 1994.

[21] S. Hillmer and G. Tiao. An arima-model-based approach to seasonal adjustment. *Journal of the American Statistical Association*, 77:63–70, 1982. doi: 10.1080/01621459.1982.10477767.

[22] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

[23] G. Lai, W. Chang, Y. Yang, and H. Liu. Modeling long-and short-term temporal patterns with deep neural networks. *arXiv preprint*, 2017. doi: 10.1145/3209978.3210006.

[24] G. G. M. Mayer. Extensive comparison of physical models for photovoltaic power forecasting. *Journal of Applied Energy*, 283, 2021. doi: 10.1016/j.apenergy.2020.116239.

[25] H. Murdock and D. Gibb. Renewables 2022 global status report. Technical report, United Nations Environment Programme, 2022.

[26] NREL. Solar power data for studies, 2022. URL `https://data.openei.org/submissions/4568`.

[27] G. Pala, J. Hola, and L. Sadowski. Non-destructive neural identification of the moisture content of saline ceramic bricks. *Construct Build Matter*, 113:144–152, 2016.

[28] N. Panwar, S. Kaushik, and S. Kothari. Role of renewable energy sources in environmental protection: A review. *Renewable and Sustainable Energy Reviews*, 15: 1513–1524, 2011. doi: 10.1016/j.rser.2010.11.037.

[29] B. Pham, T. Bui, I. Prakash, and M. Dholakia. Hybrid integration of multilayer perceptron neural networks and machine learning ensembles for landslide susceptibility assessment at himalayan area (india) using gis. *Catena*, 149:52–63, 2017.

[30] M. Phi. Illustrated guide to lstm's and gru's: A step by step explanation, 2018. URL `https://towardsdatascience.com/illustrated-guide-to-lstms-and-grusastepbystepexplanation44e9eb85bf21`.

[31] D. Quansah and M. Adaramola. Ageing and degradation in solar photovoltaic modules installed in northern ghana. *Solar Energy*, 173:834–847, 2018. doi: 10.1016/j.solener.2018.08.021.

[32] L. Sadowski, J. Hola, s. Czarnecki, and D. Wang. Pull-off adhesion prediction of variable thick overlay to the substrate. *Autom ConStruct*, 85:10–23, 2018.

[33] S. Santos, N. Torres, and R. Lameirinhas. The impact of aging of solar cells on the performance of photovoltaic panels. *Energy Conversion and Management*, 10:82–100, 2021. doi: 10.1016/j.ecmx.2021.100082.

[34] G. Sbrana and A. Silvestrini. Random switching exponential smoothing and inventory forecasting. *Int J Prod Econ*, 156:283–294, 2014.

[35] M. Sipper. High per parameter: A large-scale study of hyperparameter tuning for machine learning algorithms. *Algorithms*, 315, 2022. doi: 10.3390/a15090315.

[36] M. Steurer, R. Hill, and N. Pfeifer. Metrics for evaluating the performance of machine learning based automated valuation models. *Journal of Property Research*, 38:99–129, 2021. doi: 10.1080/09599916.2020.1858937.

[37] Y. Su. A comparative analysis of the performance of a grid-connected photovoltaic system based on low- and high-frequency solar data. *International Journal of Green Energy*, 12:1206–1214, 2015. doi: 10.1080/15435075.2014.893880.

[38] M. Sundermeyer, R. Schluter, and H. Ney. Lstm neural networks for language modeling. *Thirteenth annual conference of the international speech communication association*, 2012.

[39] T. Talakoobi. Solar power forecast using artificial neural network techniques. Master's thesis, Politecnico di Torino, 10 2020. Department of Control and Computer Engineering.

[40] L. F. Tratar and E. Strmcnik. The comparison of holt–winters method and multiple regression method: a case study. *Energies*, 109:266–276, 2016.

[41] A. Tuohy, J. Zack, S. E. Haupt, and J. Sharp. Solar forecasting: methods, challenges, and performance. *IEEE Power and Energy Magazine*, 13(6):50–59, 2015.

[42] C. Wan, J. Zhao, Y. Song, Z. Xu, J. Lin, and Z. Hu. Photovoltaic and solar power forecasting for smart grid energy management. *CSEE Journal of Power and Energy Systems*, 1(4):38–46, 2015. doi: 10.17775/CSEEJPES.2015.00046.

6 | Appendix

Data Analysis and LSTM Algorithm

```
import os
import datetime
import IPython
import IPython.display
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
!pip install -q tensorflow-model-optimization
import tensorflow-model-optimization as tfmot

mpl.rcParams['figure.figsize'] = (8, 6)
mpl.rcParams['axes.grid'] = False

df=pd.concat(map(pd.read_csv["contentSystem34Combined.csv"]
print(df)
```

This block of code imports various libraries that are commonly used for data analysis, visualization, and machine learning. Some of the libraries imported include os (operating system interactions), datetime (date and time operations), matplotlib and seaborn (plotting libraries), numpy (numerical computing library), pandas (data manipulation library), tensorflow (machine learning framework), and tensorflow-model-optimization (library for model optimization). Next line reads the CSV file "contentSystem34Combined.csv" using pd.read-csv["contentSystem34Combined.csv"]. The resulting DataFrames are then assigned to the variable df.

```
df['measured_on'] = pd.to_datetime(df['measured_on'],
 infer_datetime_format=True)
df.set_index('measured_on', inplace=True)
df.describe()

def plot_range(start, end):
  plot_features = df[['dc_power_hw__2694', 'poa_irradiance__2679']]
 [start:end]
  _ = plot_features.plot(subplots=True)
plot_range(datetime.datetime(2012, 6, 1), datetime.datetime(2014, 6, 15))
```

First, the code preprocesses the data by converting the 'measured-on' column to datetime format and setting it as the index of the DataFrame. This helps in working with time-series data efficiently. Next, the code generates descriptive statistics for the DataFrame using the describe method. This provides an overview of the numerical columns in the DataFrame, including count, mean, standard deviation, minimum, maximum, and quartiles. These statistics can help in understanding the distribution and characteristics of the data. Finally, the code defines a function called plot-range which takes two parameters: start and end. This function allows you to plot a specific range of data from the DataFrame. Inside the plot-range function, a subset of the DataFrame is created using the specified columns ('dc-power-hw–2694' and 'poa-irradiance–2679') and the specified date range (start to end). The plot method is then used to generate a plot of the subset of data, with subplots=True indicating that each column should be plotted on separate subplots. The last line of code calls the plot-range function with specific start and end dates, allowing you to visualize a specific range of data from the DataFrame.

```
minute = df.index.map(lambda d: d.hour * 60 + d.minute)
day = df.index.map(lambda d: d.timetuple().tm_yday - 1)
day_minutes = 24 * 60
year_days = 365.25

df['Day sin'] = np.sin(minute * (2 * np.pi / day_minutes))
df['Day cos'] = np.cos(minute * (2 * np.pi / day_minutes))
df['Year sin'] = np.sin(day * (2 * np.pi / year_days))
df['Year cos'] = np.cos(day * (2 * np.pi / year_days))
```

```
fft = tf.signal.rfft(df['dc_power_hw__2694'])
f_per_dataset = np.arange(0, len(fft))
n_samples = len(df['dc_power_hw__2694'])
years_per_dataset = n_samples/(24*365.25*4)

f_per_year = f_per_dataset/years_per_dataset
plt.step(f_per_year, np.abs(fft))
plt.xscale('log')
plt.ylim(0, 30000000)
plt.xlim([0.1, max(plt.xlim())])
plt.xticks([1, 365.25], labels=['1/Year', '1/day'])
_ = plt.xlabel('Frequency (log scale)')
```

The code begins by creating two new variables: minute and day. The minute variable represents the minutes of the day (0 to 1439) calculated from the DataFrame's index, while the day variable represents the day of the year (0 to 364) calculated from the DataFrame's index.

Next, the code defines two constants: day-minutes, representing the total number of minutes in a day, and year-days, representing the average number of days in a year. These constants are used in the subsequent calculations.

The code then adds four new columns to the DataFrame df: 'Day sin', 'Day cos', 'Year sin', and 'Year cos'. These columns contain sinusoidal transformations of the minute and day variables, which convert the minutes and days into periodic signals that capture their cyclical nature. The sinusoidal transformation allows the model to capture the time-related patterns in the data. After adding the new features, the code computes the Fast Fourier Transform (FFT) of the 'dc-power-hw–2694' column using tf.signal.rfft from TensorFlow. The FFT is used to analyze the frequency components of the signal and identify any periodic patterns.

The code then calculates the frequency range for the FFT using np.arange and defines years-per-dataset as the number of years represented in the dataset. This information is used to convert the frequency values into meaningful units. Finally, the code generates a plot using plt.step to visualize the FFT results. The x-axis is scaled logarithmically using plt.xscale('log'), and the y-axis limits are set with plt.ylim(0, 30000000). The plot is further customized with appropriate x-axis labels ('1/Year', '1/day') and an x-axis title ('Frequency (log scale)').

```
def normalize_df(df):
    return (df-df.min())/(df.max()-df.min())
column_indices = {name: i for i, name in enumerate(df.columns)}
n = len(df)
groups = [group[1] for group in df.groupby(df.index.year)]
train_df = pd.concat(groups[:4])
val_df = groups[4]
test_df = pd.concat(groups[5:])
num_features = df.shape[1]
train_df = normalize_df(train_df)
val_df = normalize_df(val_df)
test_df = normalize_df(test_df)
```

The normalize-df function takes a DataFrame as input and applies normalization to each column. The normalization formula (df - df.min()) / (df.max() - df.min()) subtracts the minimum value of each column from its values and divides it by the range (maximum - minimum) of that column. This normalization ensures that all values in each column are scaled between 0 and 1.

Next, the code defines a dictionary column-indices to map column names to their respective indices in the DataFrame. This mapping will be used later during the modeling process. The variable n is assigned the length of the DataFrame df, representing the total number of rows. The code then groups the data in the DataFrame by the year component of the index using df.groupby(df.index.year). This results in a list of groups, where each group contains data from a specific year.

The training data is created by concatenating the data from the first four groups using pd.concat(groups[:4]). This combines the data from the first four years into the train-df DataFrame. The validation data is assigned as the fifth group, groups[4], which contains the data from the fifth year.

The test data is created by concatenating the data from all the remaining groups, starting from the sixth group, using pd.concat(groups[5:]). This combines the data from the sixth year onwards into the test-df DataFrame. The variable num-features is assigned the number of columns in the DataFrame, representing the total number of features in the dataset. Finally, the normalize-df function is applied to the training, validation, and test DataFrames to normalize their respective data.

```python
def generate_dataset(df, target=["dc_power_hw__2694"],
  exclude_input=None, pasth=60, futureh=0.5, batch_size=32,
  sampling_period=15, sequence_stride=1):
    sampling_rate = 1
    # futureh and pasth are in hours
    past = pasth*60//sampling_period
    future = int(futureh*60//sampling_period)

    if not exclude_input:
        x = df[:-(past + future)].values
    else:
        x = df[:-(past + future)][[c for c in df.columns if not c in
        exclude_input]].values
    y = df.iloc[past+future:][target]
    return tf.keras.preprocessing.timeseries_dataset_from_array(
        x,
        y,
        sequence_length=past,
        batch_size=batch_size,
        sampling_rate=sampling_rate,
        sequence_stride=sequence_stride,
        shuffle=False
    )
```

The code provided defines a function generate-dataset that takes a DataFrame df and generates a dataset for time-series forecasting. Inside the function, the variables past and future are calculated based on the provided pasth and futureh values.

The code then checks whether to exclude any input features based on the exclude-input parameter. If exclude-input is None, all columns except the target columns are used as input features (x). Otherwise, the columns specified in exclude-input are excluded from the input features. The target variable(s) (y) are extracted from the DataFrame, starting from the index corresponding to the past + future offset. Finally, the function uses tf.keras.preprocessing.timeseries-dataset-from-array to create a time-series dataset from the input (x) and target (y) arrays. The sequence length is set to past, representing the length of the input sequence. Other parameters such as batch size, sampling rate, sequence stride, and shuffle are also provided to configure the dataset generation. The function returns the generated time-series dataset.

```
dataset_train = generate_dataset(train_df, pasth=60, sampling_period=15)
dataset_val = generate_dataset(val_df, pasth=60, sampling_period=15)
dataset_test = generate_dataset(test_df, pasth=60, sampling_period=15)

for batch in dataset_train.take(1):
    inputs, targets = batch
print("Input shape:", inputs.numpy().shape)
print("Target shape:", targets.numpy().shape)
```

In this code snippet, the generate-dataset function is used to generate time-series datasets for training, validation, and testing from the respective DataFrames train-df, val-df, and test-df.

The generate-dataset function is called three times with different DataFrame inputs and the same values for pasth (60) and sampling-period (15) parameters. This ensures consistency in the length of the input sequence (past) and the sampling interval. After generating the datasets, a loop is used to iterate over the first batch of the dataset-train dataset using dataset-train.take(1). Each iteration of the loop provides a batch of data, which is unpacked into inputs and targets variables.

The next lines of code print the shapes of the inputs and targets arrays using the .shape attribute. The inputs array represents the input sequences of the time-series data, and the targets array represents the corresponding target values to be predicted. The output of the code snippet provides the shape of the inputs array as (batch-size, sequence-length, num-features) and the shape of the targets array as (batch-size, num-targets). This gives information about the dimensions of the input and target arrays, which is useful for understanding the structure of the generated datasets.

```
from collections import defaultdict

MAX_EPOCHS = 20

def compile_and_fit(model, train, val, patience=4,
 epochs=MAX_EPOCHS, verbose=1):
  early_stopping =
  tf.keras.callbacks.EarlyStopping(monitor='val_loss',
  patience=patience, mode='min')
```

```
model.compile(loss=tf.losses.MeanSquaredError(),
optimizer=tf.optimizers.Adam(), metrics=
 [tf.metrics.MeanAbsoluteError(), tf.metrics.RootMeanSquaredError()])
history = model.fit(train, epochs=epochs,
                      verbose = verbose,
                      validation_data=val,
                      callbacks=[early_stopping])
  print(model.summary())


  return history

def plot_history(history):
  plt.plot(history.history['loss'])
  plt.plot(history.history['val_loss'])
  plt.title('model loss')
  plt.ylabel('loss')
  plt.xlabel('epoch')
  plt.legend(['train', 'val'], loc='upper left')
  plt.show()

performance = {'val': defaultdict(lambda: defaultdict(dict)),
 'test': defaultdict(lambda: defaultdict(dict))}
```

This code snippet imports the defaultdict class from the collections module and sets a constant MAX-EPOCHS to 20. The compile-and-fit function is defined, which takes a model object, training data train, validation data val, and several optional parameters. Inside the function, an instance of tf.keras.callbacks.EarlyStopping is created with the specified monitor, patience, and mode parameters. This callback monitors the validation loss during training and stops the training process if the validation loss does not improve for a specified number of epochs. The model is compiled using a mean squared error loss function (tf.losses.MeanSquaredError()) and the Adam optimizer (tf.optimizers.Adam()). Additionally, two metrics are specified: mean absolute error (tf.metrics.MeanAbsoluteError()) and root mean squared error (tf.metrics.RootMeanSquaredError()).

The model.fit method is called to train the model. The training data train is used, and the number of epochs is specified. The verbose parameter determines the level of verbosity during training. The validation data val is provided for evaluation during training. The early-stopping callback is included as a callback to monitor the validation loss and stop training if needed. After training, the model summary is printed using model.summary().

The compile-and-fit function returns the training history, which contains information about the loss and metrics values during training. The plot-history function is defined to plot the training and validation loss over epochs. It takes the training history as input and uses plt.plot to create a line plot of the loss values. The function also adds labels to the plot and displays it using plt.show(). The performance variable is defined as a nested dictionary structure using defaultdict. It is intended to store performance metrics for validation and test datasets.

```
model_inputs = Input(inputs.numpy().shape[1:])
lstm_out = tf.keras.layers.LSTM(64)(model_inputs)
outputs = tf.keras.layers.Dense(units=1, activation='linear')(lstm_out)
lstm = tf.keras.Model(inputs=model_inputs, outputs=outputs)
history = compile_and_fit(lstm, dataset_train, dataset_val, epochs=3)
plot_history(history)
performance['val']['lstm'] = lstm.evaluate(dataset_val, verbose=0)
performance['test']['lstm'] = lstm.evaluate(dataset_test, verbose=0)
```

In this code snippet, a model architecture is defined, trained, and evaluated using the LSTM (Long Short-Term Memory) layer.

First, an input layer model-inputs is created with the shape of the inputs array obtained from the previous code snippet using Input(inputs.numpy().shape[1:]). This defines the shape of the input data for the model. Next, an LSTM layer with 64 units is added to the model using tf.keras.layers.LSTM(64)(model-inputs). This layer processes the input sequences and extracts relevant features. The output of the LSTM layer is passed to a dense layer with a single unit and linear activation function using tf.keras.layers.Dense(units=1, activation='linear')(lstm-out).

The model is created using tf.keras.Model by specifying the input and output layers: lstm = tf.keras.Model(inputs=model-inputs, outputs=outputs). The compile-and-fit function is called to compile and train the model. The LSTM model, training dataset (dataset-train), and validation dataset (dataset-val) are provided as arguments. The model's performance is evaluated on the validation dataset and the test dataset using lstm.evaluate(dataset-val, verbose=0) and lstm.evaluate(dataset-test, verbose=0), respectively.

```
def yield_windows(arr, n, width, step=1):
  for i in range(0, n):
    try:
      x = arr[i * step:i * step + width]
      x = x[None, :, :]
      yield x
    except IndexError:
      raise StopIteration

def view_predictions(models, start, end, step=1, window_width=12*20,
 shift=0, df=test_df):
  arr = df[start - datetime.timedelta(minutes=15*(window_width +
 shift)):end].to_numpy()
  out = df[['dc_power_hw__2694']][start:end:step]
  out.rename(columns={'dc_power_hw__2694': 'real'}, inplace=True)
  windows = [w for w in yield_windows(arr, len(out), window_width,
 step=step)]
  style = ["-"]
  for k,v in models.items():
    style.append("--")
    predictions = calculate_predictions(v, windows)
    out[k] = predictions
    differences = [abs(predictions[i]-out['real'].iloc[i]) for i in
 range(len(out))]
    print(f"{k} average difference:
 {sum(differences)/(len(differences)+1)}")out.plot(style=style)

def calculate_predictions(model, windows):
  return [model.predict(w, verbose=0).flatten()[0] for w in windows]
```

The code snippet provides utility functions for generating and viewing predictions using trained models. The yield-windows function is a generator that yields overlapping windows of a specified width from an input array (arr). It iterates n times with a step size of step, and for each iteration, it extracts a window of width width starting from index i * step. The window is reshaped to have a shape of (1, window-width, num-features) and yielded using yield x. This function is useful for dividing the input array into windows for making predictions.

The view-predictions function takes a dictionary of models, a start and end datetime range, and an optional step size, window width, shift, and dataframe (df). It prepares the input array by extracting a slice from the dataframe df based on the specified datetime range and converts it to a numpy array. It also prepares the output dataframe (out) by selecting the target column and renaming it to 'real'.

The calculate-predictions function takes a model and a list of windows and returns a list of predictions. It uses the model.predict method to make predictions for each window and flattens the output to obtain a scalar prediction. The predictions are stored in a list and returned.

```
view_predictions({"lstm": lstm }, start=datetime.datetime(2015, 1,
 16), end=datetime.datetime(2015, 1, 25), step=1,window_width=12*20,
 shift=0, df=val_df)

 view_predictions({"lstm": lstm }, start=datetime.datetime(2016, 1,
 16), end=datetime.datetime(2016, 1, 25), step=1,window_width=12*20,
 shift=0, df=test_df)
```

This code calls the view-predictions function to generate and visualize predictions using the LSTM model. Two sets of predictions are generated and displayed:

The first call to view-predictions generates predictions using the LSTM model for the datetime range from January 16, 2015, to January 25, 2015. It uses the validation dataframe (val-df) for the predictions. The second call to view-predictions generates predictions using the LSTM model for the datetime range from January 16, 2016, to January 25, 2016. It uses the test dataframe (test-df) for the predictions.

By calling view-predictions, the code generates predictions for the specified datetime ranges using the LSTM model and displays the predictions alongside the real values.

List of Figures

List of Tables

Acknowledgements

I stand at the culmination of an incredible journey, and as I turn the pages of this thesis, I am overwhelmed with gratitude for the numerous individuals who have accompanied me along this path. It is with heartfelt appreciation that I express my deepest gratitude to those who have contributed to the completion of my Master's thesis.

First and foremost, I extend my utmost gratitude to my esteemed supervisor, Professor Giancarlo Storti Gajani for his guidance, invaluable support, and immense knowledge. His expertise and dedication have been instrumental in shaping the direction and quality of this research. I have been truly fortunate to have had such an exceptional mentor.

I am also indebted to the faculty members of Politecnico di Milano, whose wisdom, expertise, and passion for their respective fields have enriched my academic experience. Their commitment to fostering an environment of learning and intellectual curiosity has been a constant source of inspiration throughout my journey.

My heartfelt appreciation also goes out to my family and friends who have supported me unconditionally throughout this journey. I reserve a special place of appreciation for my dear friend and colleague, Prateek Pati. Throughout this transformative journey, Prateek's support, camaraderie, and intellectual companionship have been truly remarkable.

Lastly, I would like to acknowledge the countless authors, researchers, and scholars whose works have formed the foundation of my research. Their pioneering efforts and groundbreaking discoveries have paved the way for my own investigations, and I am grateful for the wealth of knowledge they have contributed to the academic community.

I am humbled by the opportunity to have been part of this remarkable academic journey, and I am profoundly grateful to everyone who has played a role, big or small, in its realization. Their collective contributions have undoubtedly shaped the person I have become today, both academically and personally.

With deep appreciation and sincere gratitude,

Saloni Dhingra