POLITECNICO DI TORINO

Department of Control and Computer Engineering
(DAUIN)

Master of Science in Computer Engineering

Master Degree Thesis

Solar Power Forecast Using Artificial Neural Network Techniques

Supervisor
Prof. Elena BARALIS

Student
Talaye TALAKOOBI
matricola: 260271

Company Supervisor
PUNCH Torino S.p.A.
Prof. Massimiliano MELIS

October 2020

Abstract

Solar energy is a clean, available and renewable source of energy. Deployment of photovoltaic panels is trending to take advantage of solar energy for electrical power generation. Intermittent nature of solar energy results in variable generated power. This uncertainty could be alleviated by taking advantage of energy storage systems and accurate solar power forecast. This thesis aims to implement a solar forecast module to take part in an optimized Energy Management System (EMS). Different solar power prediction methods are studied including statistical methods, sky imagers, satellite imaging and Numerical Weather Prediction (NWP). Artificial neural networks (ANNs) which are a subset of statistical methods are chosen as prediction method to satisfy requirements imposed by EMS. The requirements include precise forecast in short-term prediction horizon for proper functionality of the EMS. Prediction horizon is amount of time in future for which prediction is needed. Multiple ANN architectures suitable for time-series prediction are investigated and compared, including Long Short-term Memory (LSTM), Long- and Short-term Time Series Network (LSTNet) and Temporal Convolutional Network (TCN). Although all the aforementioned methods have acceptable performance, TCN architecture shows more promising results.

In order to further improve the prediction accuracy, effect of clustering the dataset into sunny and cloudy sub-datasets and using a dedicated prediction module for each sub-dataset is studied. Results show that prediction accuracy is improved by clustering the dataset for all the models. Moreover, simulation results on datasets of multiple geographical locations with different climate conditions show that prediction accuracy is higher in locations having more stable weather and sunny days.

A Graphical User Interface (GUI) is implemented to simplify working with the forecast module. Service is made available to users through exposing REST APIs on a remote server to facilitate user interactions with the forecast service.

Acknowledgements

I would like to express my sincere gratitude to my supervisors professor Massimiliano Melis and professor Elena Baralis for the trust and the opportunity I was given to further my research.

I would also like to thank my tutors and colleagues from my internship and thesis at Punch Torino specifically, Silvia Cantagalli and Giovanni Spoleti for their collaboration and support.

Last but not least, I would like to thank my husband Iman for all the understanding and support during this period of our life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Nowadays, renewable energy sources have become of paramount importance in power generation due to concerns regarding greenhouse gas emissions and environmental pollution issues that are consequences of excessive consumption of fossil fuel energy sources [1]. Moreover, fossil fuel energy sources are limited and will run out someday in future. Therefore, it's necessary to look for other sources of energy to substitute fossil fuels. By end of 2018, in some locations electricity generation from new wind and photovoltaic (PV) plants had become more economical than power generation from fossil fuel-fired plants. Also, in some places building new wind and solar PV plants cost less than continuing execution of current existing fossil fuel power plants [2]. Since solar energy is a clean, available, free and renewable source of energy, deployment of PV panels has increased in recent years in order to generate electricity from solar energy [3]. Due to interest of countries in investment of renewable sources of energies, it is likely that installment of PV panels continue to increase. Fig. 1.1 shows solar photovoltaics global capacity from 2008-2018.

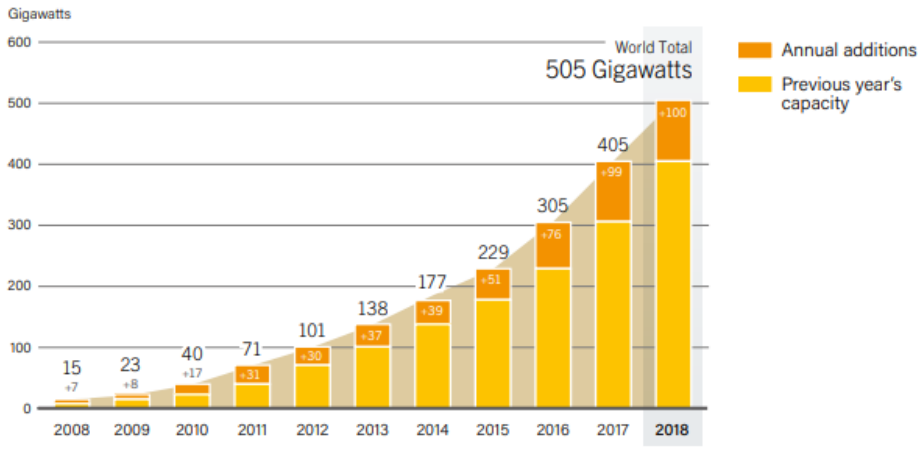


Figure 1.1. Solar PV global capacity and annual additions [2]

Since solar photovoltaics is one of the energy technologies that are growing rapidly and Italy is one of the top ten countries in the sense of solar PV capacity, research in these fields are crucial and inevitable. Fig. 1.2 illustrates top ten countries having the most solar PV capacity around the world [2].

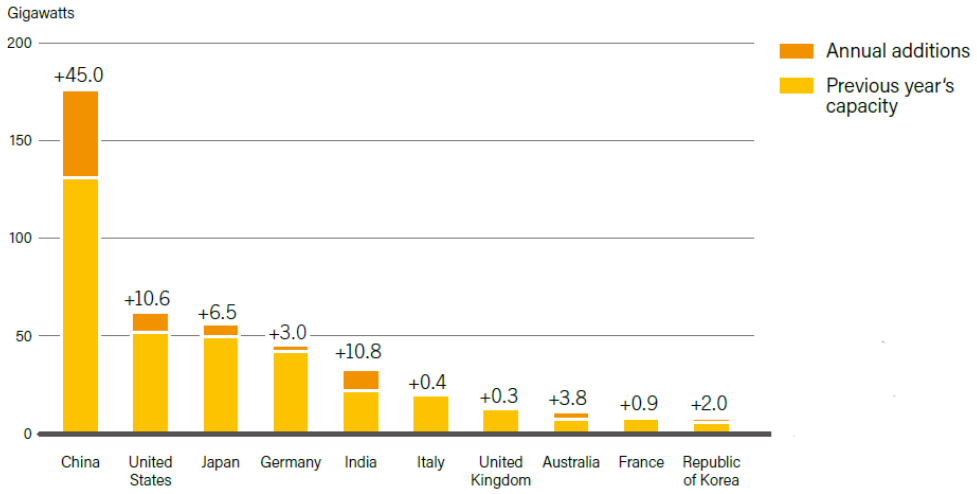


Figure 1.2. Solar PV capacity of top 10 countries

Use of photovoltaic panels is spreading for commercial and residential applications. With technical advances and economic feasibility, integrating renewable resources in microgrids is becoming more trending. Microgrids are localized energy networks that utilize local energy sources like solar energy and wind power to generate electricity. A microgrid contains a cluster of loads, electricity generator units and energy storage systems that operate in coordination. Microgrids can operate in isolated mode independently of the main grid system or they can operate in grid-connected mode which is in collaboration with the main grid. Fig. 1.3 shows schematic of a microgrid system. However, integration of solar energy in microgrids is challenging in sense of operation due to intermittent nature of solar energy. Power generation of photovoltaic panels depends on meteorological factors such as solar irradiance, air temperature and relative humidity [4]. As a result, output power of photovoltaic panels varies with respect to the other parameters. The uncertainty of output power can strongly affect reliability and stability of the power system. Fig. 1.4 depicts the sudden power output variation in a PV array in Florida that can jeopardize reliability and stability of the power system. But the uncertainty can be alleviated by taking advantage of energy storage systems. Energy Management Systems (EMS) monitor, control and optimize the performance of the grid system. EMS responsibility is making sure of grid operation in reliable, secure and economical way both in grid-connected and isolated mode [5].

Energy management system can optimize utilization cost and energy storage level of energy storage systems. Various ways exist for microgrid energy storage systems optimization. One possible way of microgrid management that is used in [6], is implementation of hierarchical state-machine based energy management method to reduce cost of energy

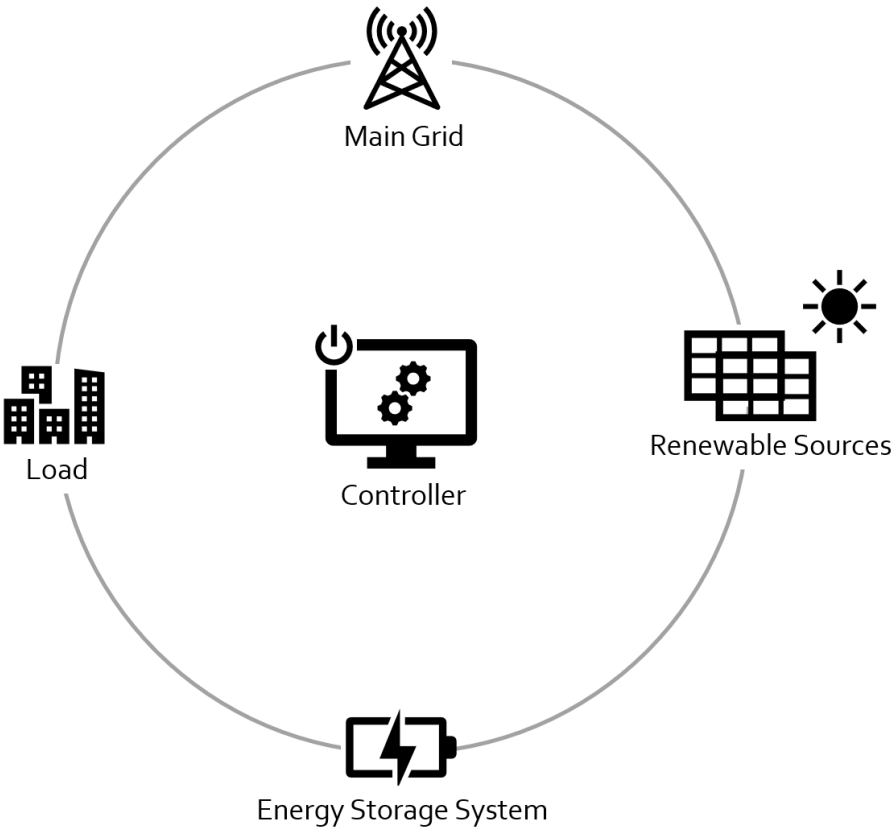


Figure 1.3. An example of microgrid system schematic

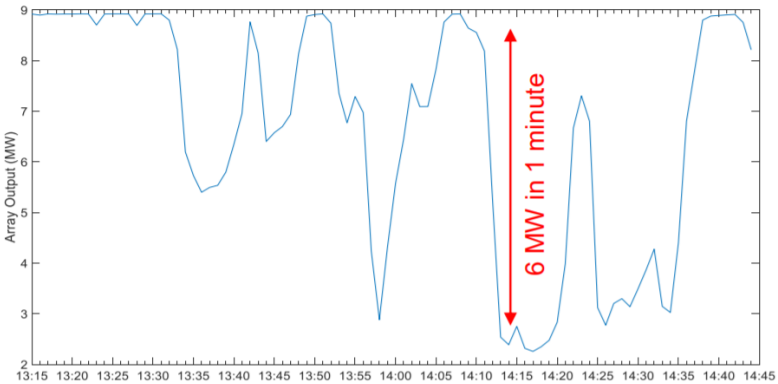


Figure 1.4. Sudden PV array power output variation

storage system and maintaining the energy storage level. This method also supports use of multiple energy storage systems. The method contains two management layers. Each

layer has a different responsibility. The bottom layer is responsible of controlling storage devices and the top layer oversees the state machine. In any case, energy storage system performance must be optimized by advanced control techniques using EMS system. It is worth mentioning that precise prediction is essence of efficiency in energy management systems.

Due to increase of PV installment in microgrid systems, need of accurate solar prediction is becoming more evident. On the other hand, solar prediction is a challenging task due to variation of meteorological variables that have direct effect on output power of PV panels. Specifically, the prediction task becomes more challenging when prediction horizon increases. Prediction horizon indicates the amount of time in the future that target variable should be predicted.

Increasing importance of accurate solar prediction, has made solar forecasting an attractive field of research. Therefore, a lot of research and studies took place in the recent years in order to fulfill the required prediction precision. Moreover, advances in machine learning has result in progress in a lot of research areas including time series prediction. Various machine learning models and techniques have been proposed in recent years to deal with time series forecast tasks. Also, computational capacity of devices is improving which is helpful for machine learning tasks that require a lot of computations. Another helpful advance is availability of high-quality data that is an essential requirement for proper training of machine learning models.

1.2 Thesis Objectives

Current research took place in Punch Torino facility. One of the project opportunities that the innovation team of Punch torino is investigating is about the energy management systems in microgrids. The thesis activities were focused into development of an algorithm for the forecasting of power generation from a solar array. The output of the calculation will be used by the Energy Management System, in development phase by the Innovation team of Punch Torino, in charge of the optimized operational planning of a microgrid.

Due to the importance of precise solar forecasting, the thesis works aims to address the necessity to have accurate solar power forecast for an optimized EMS and finding an efficient method for forecasting solar power to be included in the microgrid control system. Solar power prediction can help to optimize the usage of energy storing devices based on the final user consumption and estimated generated power.

Moreover, solar forecast module deployment phase needs to be considered to simplify user interaction with the implemented forecast module. Implementation of a Graphical User Interface (GUI) and REST APIs are valid choices for deployment that can ease user interactions with solar forecast module.

1.3 Thesis Structure

The rest of the thesis is organized as follows. Chapter 2 provides literature review about solar forecasting techniques. Reviewed techniques include statistical methods, sky imagers, satellite imaging and numerical weather prediction methods. Then different prediction horizons and prediction modes are discussed, and proper values are selected to satisfy

the application requirements. Based on selected values for prediction horizon and mode, suitable forecasting technique is selected among the mentioned techniques and different architectures suitable for time series forecasting such as Long short-term Memory (LSTM), Long- and Short-term Time Series Network (LSTNet) and Temporal Convolutional Network (TCN) are further investigated.

Description of data and tools used during this research are presented in chapter 3.

Chapter 4 discusses data exploration and preprocessing. Performance metrics used during this research are introduced. Hyperparameter tuning is used to find the best set of hyperparameters. Results obtained from different architectures are compared. In order to further improve prediction accuracy, possibility of dataset clustering into sunny and cloudy days and using multiple prediction modules is studied. All the results and their interpretation are presented. After finding the best method for this research, model performance is studied in different locations and weather conditions around the world to understand how well model performs in different weather conditions.

Chapter 5 discusses deployment scenarios. A Graphical User Interface (GUI) implementation is presented. Moreover, implementation of a remote server and APIs are described.

Finally, chapter 6 concludes the thesis research by pointing out the findings and outcome of this work. Moreover, possibile improvements are proposed for further research.

Chapter 2

Literature Review

This chapter is dedicated to literature review and study of existing solar forecast methods and characteristics of each technique.

2.1 Solar Power Prediction Methods

Existing solar power prediction methods are divided into different categories based on prediction horizon, including statistical methods, sky imagers, satellite imaging and Numerical Weather Prediction (NWP) [3]. In all the methods, solar irradiance is predicted and converted to power. Except for statistical methods, which can also predict power directly without need of conversion. Different methods and their proper prediction horizons are depicted in **Fig. 2.1**.

2.1.1 Statistical Methods

Artificial neural networks (ANNs), regression models, support vector machines and Markov chains are different kinds of statistical methods [3]. Historical observations are used to train statistical methods. Output predictions are computed based on historical values of input variables [7]. For this reason, depending on input variables, power output can be predicted in direct mode and also it can be predicted indirectly by forecasting irradiance and converting the result to power. Pros and cons of both options are investigated in section 2.2.2.

2.1.2 Sky Imagers

Sky imagers are digital cameras that provide high quality ground-based images from sky. Clouds have strong effect on solar irradiance at ground's surface. For this reason, determining clouds state is useful for solar irradiance forecasting. Sky imagers detect clouds, estimate cloud height from ground and detect cloud motion velocity from consecutive images. Since sky imagers recognize cloud shadows, they can detect abrupt changes in solar irradiance [7].

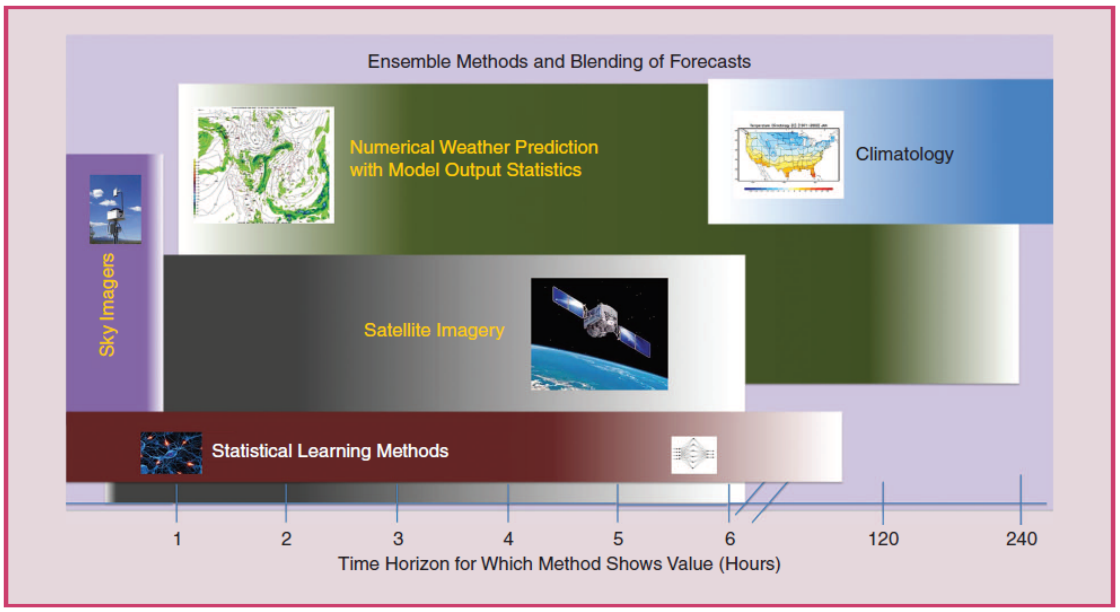


Figure 2.1. Overview of solar prediction methods based on forecast horizon [3]

However, Prediction horizon of sky imagers is very short and is up-to 30 minutes. Also, Sky imagers are expensive to use with respect to other forms of short-term prediction methods [3].

2.1.3 Satellite Imaging

Geo-stationary satellites can provide information about clouds states and their movement. From images gathered by satellites clouds can be detected, and their characteristics can be determined. From the obtained information, irradiance can be predicted as far as six hours ahead [7].

2.1.4 Numerical Weather Prediction (NWP)

NWP model can predict solar irradiance based on numerical dynamic atmosphere modeling. NWP predicts state of atmosphere based on current state of atmosphere and correct physical laws. NWP is suitable for predictions up-to two weeks. In general NWP methods outperform satellite imaging predictions in sense of accuracy when prediction horizon is beyond 4 hours. On the other hand, due to spatial and temporal limitations, characteristics of most clouds is unresolved and for this reason NWP methods are not suitable for prediction horizons less than several hours [7].

2.2 Prediction Method Selection

2.2.1 Prediction Horizon Selection

In order to be able to choose a proper method, prediction horizon should be determined. Prediction horizon is the future amount of time for which PV output power is predicted. The concept of prediction horizon and window length is illustrated in Fig. 2.2. Solar power generation can be classified into four categories based on prediction horizon [7], [8].

1. Very short-term Prediction Forecasting of generated power is done from a few seconds to minutes. This type of forecasting is suitable for PV and storage control and forecasting sudden fluctuations in power generation.

2. Short-term Prediction This category includes prediction horizon up to 72 hours ahead and is appropriate for ensuring unit commitment, scheduling, dispatching of electrical power, etc.

3. Medium-term Prediction Forecasting is done for up to a week ahead and is useful for maintenance and scheduling

4. Long-term Prediction Includes prediction horizon up to a year which is useful for generation expansion planning.

Different prediction horizons and their applications are depicted in Fig. 2.3.

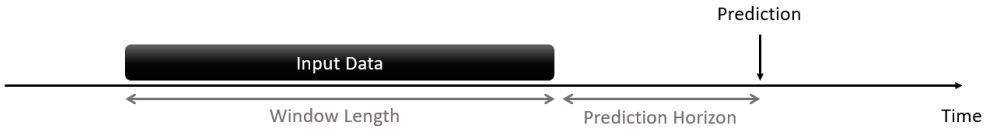


Figure 2.2. Prediction horizon and window length concept

Since very short-term and short-term predictions are suitable for plant operation, scheduling and storage control in energy management systems, two different prediction horizons have been chosen: 5-minutes horizon and 24-hours horizon.

Based on specified prediction horizons, statistical methods in particular ANNs are selected for solar prediction task.

2.2.2 Prediction Mode Selection

PV solar prediction can be performed in indirect and direct mode. In indirect mode, environmental variables like irradiance, temperature and humidity are predicted and are inputted in PV simulator in order to predict output power. On the other hand, in direct mode historical data of PV power output is used for predicting the output power. In addition to output power, some other related meteorological data can be used for prediction.

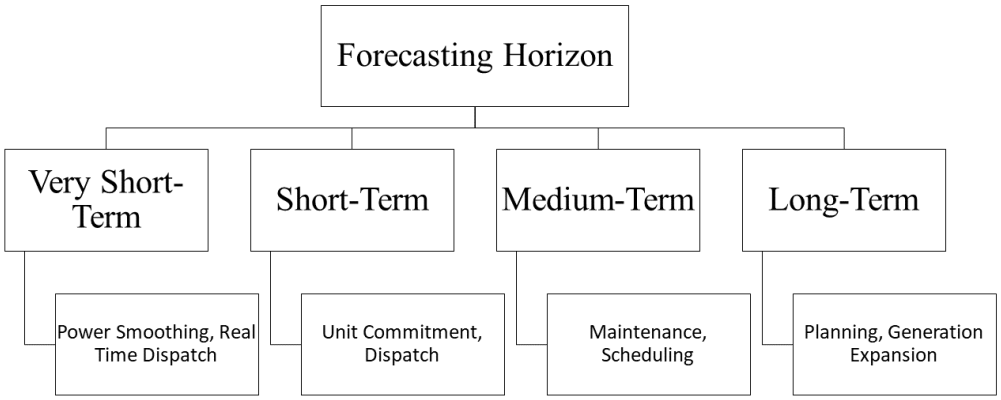


Figure 2.3. Classification of different forecasting horizons and their application

Gigoni, Betti, Crisostomi, *et al.* in [9] discussed the reasons why direct methods predictions are more accurate than indirect methods predictions. One of the mentioned reasons is that PV models used in indirect prediction are approximations of the real plant and are not able model every possible physical phenomenon occurring in real-world. Moreover, physical variables degrade over time due to aging which again makes the predictions less accurate. As a result, for this project direct methods are considered.

2.3 Time Series Forecasting with Artificial Neural Networks (ANNs)

Solar power forecasting is a subset of Time Series Forecasting which is an important field in machine learning. Time Series Forecasting is challenging because of existence of mixture of short-term and long-term repeating patterns.

Recurrent neural networks (RNN) are mainly used in sequential data processing. Long Short-Term Memory (LSTM) [10], and Gated Recurrent Unit (GRU) [11] are one of the most effective variants of RNNs [12]. In order to improve performance, variant versions of hybrid architectures using advanced RNNs have been developed. This includes Long- and Short-term Time Series Network (LSTNet) architecture, which combines strengths of convolutional and recurrent neural networks. The convolutional layer detects short-term patterns, the recurrent layer detects long-term patterns. Moreover, Recurrent-skip structure detects very long-term dependence patterns [13].

In 2018 Bai, Kolter, and Koltun [14] introduced Temporal Convolutional Network (TCN) family of architectures based on CNNs for sequential tasks. TCN models use dilated causal convolutions, also they take advantage of residual connections [15].

In this project, LSTM architecture is chosen as a baseline for comparison of other models. LSTNet model which is an advanced hybrid architecture is a suitable candidate for solar prediction task. Moreover, since TCN architecture outperforms baseline recurrent architectures in several sequence modeling tasks, it is motivating to use TCN architecture on solar prediction task.

2.3.1 Long Short-term Memory (LSTM)

Training Recurrent Neural Networks through time is difficult due to vanishing gradient problem [16]. Vanishing gradient problem causes exponential decrease or increase in influence of a given input on hidden layer and output during backpropagation [17].

LSTM is able to preserve information over long periods of time and does not suffer from vanishing gradient problem due to usage of gates [18].

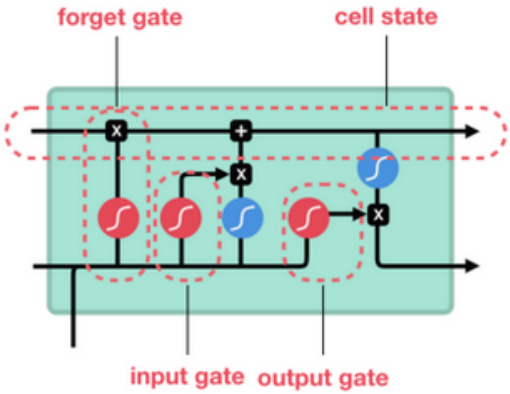


Figure 2.4. LSTM unit architecture [19]

LSTM architecture is made of a series of recurrently connected LSTM units also called memory blocks. Each LSTM unit has three different gates including input gate, output gate and forget gate. Input gate controls what to store in memory block, output gate controls output flow of information to the rest of network [10]. Forget gate allows LSTM unit to self-reset memory contents when they become irrelevant [20]. LSTM unit is illustrated in **Fig. 2.4** and can be described mathematically through the following equations:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{2.1}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2.2}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{2.3}$$

$$\tilde{c} = \tanh(W_c x_t + U_c h_{t-1}) \tag{2.4}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{2.5}$$

$$h_t = o_t \odot \tanh(c_t) \tag{2.6}$$

W and U are weights of the connections. And b is bias vector parameters. x_t is input of LSTM and h_t is output also called hidden state of LSTM unit. c_t refers to memory cell state at time t. $\tilde{c}$ is new memory cell state candidate which is generated from current input and past hidden state and includes aspects of new input x_t. Input gate (i) it is responsible for determining valuable parts of new input by considering x_t and previous hidden state. Forget gate (f) assess usefulness of past memory state for computation of current memory state. New memory cell is generated by forgetting a portion of previous cell state determined by forget gate and gating a portion of new cell state determined by input gate. Output gate (o) separates final memory from hidden state by determining what portion of cell state is required to be stored on hidden state.

2.3.2 Long- and Short-term Time Series Network (LSTNet)

LSTNet [13] is a strong hybrid model introduced by Lai *et al.* in 2017. Architecture of LSTNet is illustrated in **Fig. 2.4** . LSTNet is composed of convolutional component, recurrent component, recurrent-skip component and autoregressive component. Each component is described and formulated.

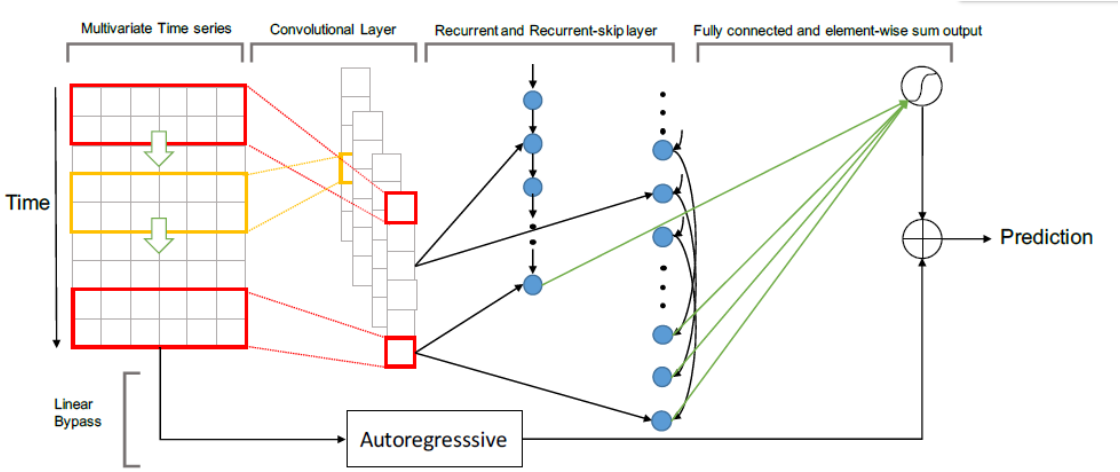


Figure 2.5. LSTNet architecture [13]

Convolutional Component

Convolutional layer is responsible for short-term pattern extraction and finding dependencies among variables. Output of each filter of this layer can be formulated mathematically in equation 2.7:

$$h_k = RELU(W_k * x + b_k) \tag{2.7}$$

Recurrent Component

Output of Convolutional layer is fed into recurrent component. GRU is the recurrent component of LSTNet. Authors of [13] believe that RELU activation function has better

performance with respect to tanh for usage in GRU gates due to easier gradient backpropagation. Recurrent component can be described mathematically by following equations:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{2.8}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{2.9}$$

$$\tilde{h}_t = RELU(r_t \odot (U_h h_{t-1}) + W_h x_t + b_h) \tag{2.10}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{2.11}$$

r is reset gate and is responsible to find importance of previous hidden state (h_{t-1}) in computation of new memory ($\tilde{h}_t$). Update gate z determines how much of new memory and previous hidden state should be carried forward to next hidden state (h_t).

Recurrent-skip Component

Although gating has alleviated gradient vanishing problem in LSTM and GRU, these architectures still have difficulties in capturing very-long term patterns in data. In case of solar forecasting, prediction is more efficient when both most recent records and records of same hour in adjacent days are used, due to the 24-hour daily basis pattern that exists in solar data. But this 24-hour periodic pattern is difficult to catch by RNNs due to long length of each period. To solve this issue recurrent skip-connections are used. Recurrent skip-connections extend temporal span and connect hidden units that have same phase in different periods.

$$z_t = \sigma(W_z x_t + U_z h_{t-p} + b_z) \tag{2.12}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-p} + b_r) \tag{2.13}$$

$$\tilde{h}_t = RELU(r_t \odot (U_h h_{t-p}) + W_h x_t + b_h) \tag{2.14}$$

$$h_t = (1 - z_t) \odot h_{t-p} + z_t \odot \tilde{h}_t \tag{2.15}$$

In above equations p is the number of hidden cells skipped through. Skip length in case of solar forecasting is set to 24 for hourly prediction.

A dense layer combines output of recurrent layer and recurrent-skip layer. Hidden state at time t is composed of hidden state of recurrent component at time t and p hidden states of recurrent-skip component from timestamp $t - p + 1$ to t. output of dense layer can be formulated as 2.16

$$h_t^D = W^R h_t^R + \sum_{i=0}^{p-1} W_i^S H_{t-i}^S + b \tag{2.16}$$

Autoregressive Component

Neural networks output is not sensitive to scale of inputs due to non-linearity of CNNs and RNNs. This is problematic in case of solar forecast that input data scale changes constantly and results in reduction of prediction accuracy.

LSTNet output is composed of two parts, linear part and non-linear part. The non-linear part is the output of recurrent units that focuses on recurring patterns. Linear part focuses on scaling issue. Autoregressive model is used in LSTNet for handling linear part of architecture.

Output prediction of LSTNet is the result of integration of linear and non-linear parts.

2.3.3 Temporal Convolutional Network (TCN)

TCN [14] was introduced in 2018 by Bai, Kolter, and Koltun. as a substitute for recurrent neural networks in time series forecasting tasks. TCN is based on CNN architecture and has two main characteristics:

1. Causal convolutions are used in TCN architecture, therefore there is no leak from future to past.
2. Like RNNs, TCN can have input sequence of arbitrary length and maps it to output sequence of same length.

Main components of TCN architecture are dilated causal convolutions with residual connections. For having same sequence length in input and output, TCN uses 1D fully connected convolutional network (FCN) with zero padding.

Moreover, for not having leakage from future to past TCN uses causal convolutions so that output at time t is result of convolutions from time t and earlier times in previous layers.

Causal convolutions receptive field grows linear with depth of the network. In order to alleviate difficulties of handling tasks that require larger receptive field, dilated convolutions are used in TCN architecture. The advantage of using dilated convolutions is exponential growth of receptive field. Dilation adds fix-sized steps between adjacent filter taps. In TCN dilation is increased exponentially with depth of the network. In this way all inputs are hit by filters in receptive field.TCN block is illustrated in **Fig. 2.6**.

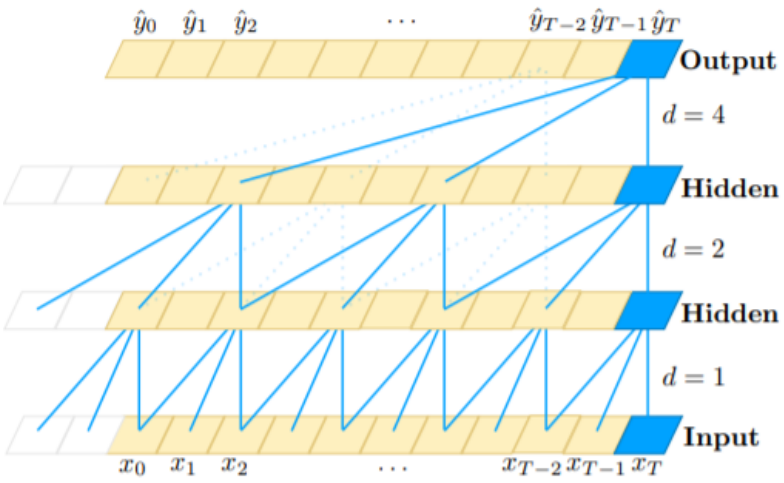


Figure 2.6. TCN architecture [14]

Effective history of each layer is computed by $(k-1)d$. K is filter size and d is dilation factor. Therefore, for increasing receptive factors filter size and/or dilation factor can be increased.

Residual connections are useful specially in very deep networks. For having very large receptive field sometimes it is necessary to use very deep TCN architecture. For more

stabilization of deep networks TCN takes advantage of residual connections. TCN residual block is demonstrated in **Fig. 2.7**.

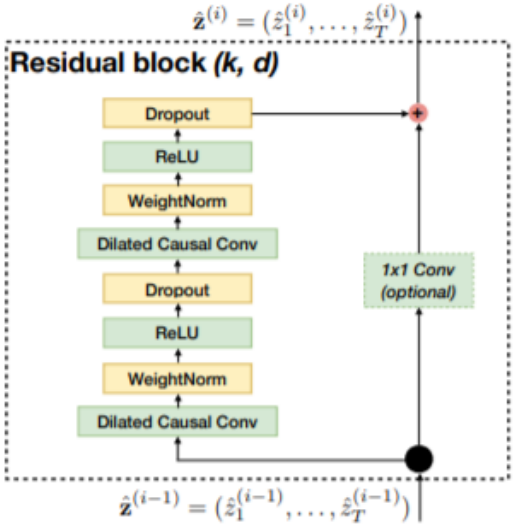


Figure 2.7. TCN block architecture [14]

Each residual block contains two dilated causal convolution layers, ReLU non-linearity, weight normalization and spatial dropout for regularization.

Since in TCN input and output can have different widths, residual connections are occurred through 1x1 convolutions.

One of the main advantages of TCN is the ability to compute convolutions in parallel, because same filter is used in all layers. Therefore, unlike sequential processing in RNNs, TCN can compute long input sequence as a whole.

TCN does not suffer from gradient vanishing because backpropagation path is different from temporal direction of sequence.

Chapter 3

Data and Tools

3.1 Data

Public power output data from a PV panel installation of 227 MW capacity in Florida, USA is used for better demonstration of results [21]. The historical data is regarding year 2006 and contains information about generated power (MW) with 5 minutes sampling rate. In order to have a general view, some rows of the power dataset are illustrated in Fig. 3.1.

	LocalTime	Power(MW)
0	01/01/06 00:00	0.0
1	01/01/06 00:05	0.0
2	01/01/06 00:10	0.0
3	01/01/06 00:15	0.0
4	01/01/06 00:20	0.0

Figure 3.1. General view of power dataset

Since, meteorological data was not available another dataset is used that contains information about solar irradiance ($\frac{w}{m^2}$), relative humidity (%), temperature (F) and amount of rain fall (in) of same location during 2006 [22]. In the weather dataset sampling rate is one hour. In order to have a general view, some rows of the weather dataset are illustrated in Fig. 3.2.

	DateTime	Temperature (F)	SolarIrradiance (w/m^2)	RelativeHumidity (pct)	RainTotal (in)
0	1 Jan 2006 12:00 AM	57.68	0.38	94	0.0
1	1 Jan 2006 1:00 AM	57.21	0.38	95	0.0
2	1 Jan 2006 2:00 AM	58.46	0.43	95	0.0
3	1 Jan 2006 3:00 AM	58.29	0.44	96	0.0
4	1 Jan 2006 4:00 AM	58.60	0.44	96	0.0

Figure 3.2. General view of weather dataset

The two datasets are merged to form a unit dataset. The required steps for merging the two datasets are further explained in Chapter 4. The resulting dataset is used to find the best architecture.

After finding the best architecture, in order to test the model performance on different weather conditions around the world public datasets containing power and meteorological data during year 2016 for Italy, Spain and Oman are used [23]. Data refers to an installed peak PV power of 1 KW with system loss of 14%. The datasets contain information about date and time, generated power (W), solar irradiance on the plane of the PV arrays ($\frac{w}{m^2}$) and air temperature ($C°$). Italy, Spain and Oman datasets do not contain the field regarding relative humidity. In order to have a general view of the datasets some rows of the aforementioned datasets are illustrated in Fig. 3.3-3.5.

	DateTime	Power (W)	SolarIrradiance (w/m^2)	Temperature (C)
0	20160101:0010	0.0	0.0	0.16
1	20160101:0110	0.0	0.0	-0.31
2	20160101:0210	0.0	0.0	-0.79
3	20160101:0310	0.0	0.0	-1.27
4	20160101:0410	0.0	0.0	-1.14

Figure 3.3. General view of Italy dataset

	DateTime	Power (W)	SolarIrradiance (w/m^2)	Temperature (C)
0	20160101:0010	0.0	0.0	-1.29
1	20160101:0110	0.0	0.0	-0.12
2	20160101:0210	0.0	0.0	1.05
3	20160101:0310	0.0	0.0	2.22
4	20160101:0410	0.0	0.0	2.58

Figure 3.4. General view of Spain dataset

Data and time format in the datasets are different. This issue will be dealt with in Chapter 4.

	DateTime	Power (W)	SolarIrradiance (w/m^2)	Temperature (C)
0	20151231:2348	0.00	0.00	16.22
1	20160101:0048	0.00	0.00	16.09
2	20160101:0148	0.00	0.00	15.95
3	20160101:0248	0.00	0.00	15.82
4	20160101:0348	231.08	208.08	17.38

Figure 3.5. General view of Oman dataset

3.2 Implementation Tools

The following tools and libraries have been used during implementation of this project:

- **Visual Studio Code (VS Code)** is a lightweight, powerful source code editor. VS Code supports Python through usage of extensions. One of the points that has made VS Code such a strong editor is possibility of using the third-party extensions that has made the programmers lives easier.
- **Python programming language** is one of the languages that are commonly used for Machine Learning purposes and a lot of Machine Learning libraries exist for Python language.
- **PyTorch** is and open source Machine Learning library that facilitates building Machine Learning projects.
- **Numpy** manages array and matrix data structures.
- **Pandas** is built on NumPy package and uses DataFrames to manipulate tabular data.
- **Pvlib** is used to estimate clear sky irradiance.
- **Matplotlib** is a plotting library for Python are used.
- **Flask** is a micro web framework written in Python. In section 5 Flask is adopted for server implementation.
- **TKinter** is the standard Python interface to the TK GUI toolkit and it is used to create a GUI (Graphical User Interface) to simplify modification of configuration parameters.
- **Python threading module** is used to facilitate running multiple threads at the same time. Threading is used in GUI to prevent it from freezing and it is used in server to manage different requests from users.

Chapter 4

Implementation and Results

4.1 Data Exploration and Preprocessing

In order to merge the datasets and create a unit dataset some preliminary steps have been taken:

1. The time column format in the two datasets are not compatible and has string format, therefore *strptime*() function from Python datetime module is used to create datetime object from the string.
2. The created DateTime column is set as index of the data frame so that the datasets could be merged based on the index variable.
3. In order to solve inconsistency due to different sampling rates in the two datasets, data is averaged on hourly basis.
4. Then the two datasets are merged based on date and time in order to create a single dataset.

4.1.1 Missing Values

Next step is dealing with missing values. Missing data is detected and results are shown in Fig. 4.1. Four missing values exist in Temperature, SolarIrradiance, RainTotal and RelativeHumidity features. There are no missing values in Power feature. Interpolation method is used to fill missing values instead of hard coding them.

```
1 # sanity check - number of nan
2 print(data.isnull().sum())
```

```
Power               0
Temperature         4
RelativeHumidity    4
SolarIrradiance     4
RainTotal           4
dtype: int64
```

Figure 4.1. Missing values detection

4.1.2 Statistical Measures

In order to better understand the data, statistical measures can be used:

- **Count** indicates number of records for each attribute.
- **Mean** indicates the average value of each attribute.
- **Std** indicates the standard deviation of each attribute, that can be used to understand data dispersion around the average.
- **Min** minimum value of the attribute.
- **25%** the lower percentile
- **50%** the median, unlike average value it provides information on the distortion of the distribution
- **75%** the upper percentile
- **Max** maximum value of the attribute

Tab. 4.1 shows statistical measures. Visualizing these numerical results graphically gives better intuition of data.

	count	mean	std	min	25%	50%	75%	max
Power	8760.0	44.494698	56.630253	0.00	0.0000	0.008333	97.160417	179.025
Temperature	8760.0	72.793039	11.703850	32.46	65.7375	73.420000	80.822500	98.970
RelativeHumidity	8760.0	70.456221	20.467894	14.00	54.0000	75.000000	89.000000	97.000
SolarIrradiance	8760.0	194.258273	268.693574	0.00	0.3600	8.235000	380.322500	970.250
RainTotal	8760.0	0.004682	0.051183	0.00	0.0000	0.000000	0.000000	1.390

Table 4.1. Statistical measures

4.1.3 Data Visualization

Fig. 4.2 shows evolution of features over time. It can be seen that Power and SolarIrradiance have similar trends and have higher values during spring and summer. Moreover, temperatures value is higher during summer months as expected then it reduces.

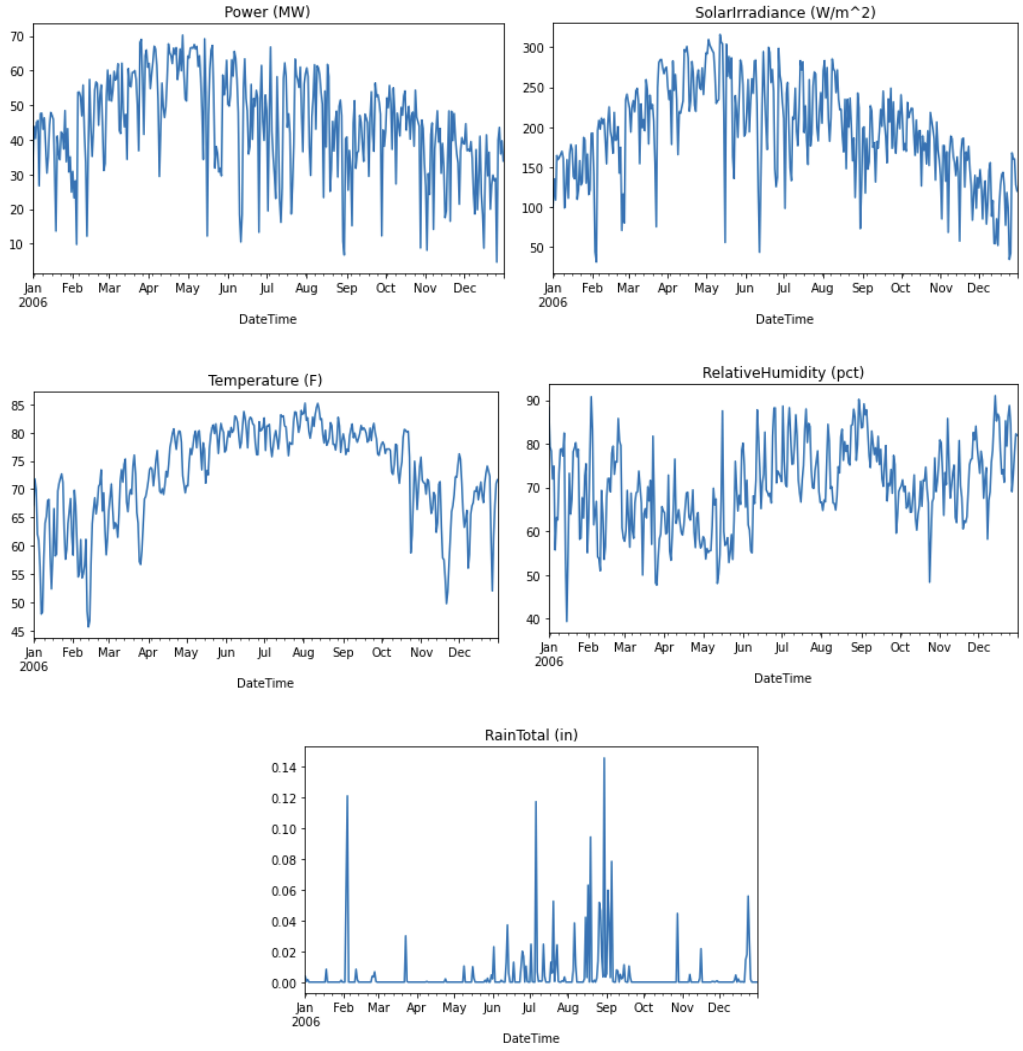


Figure 4.2. Data visualization

4.1.4 Box Plots

Box plots can be used for demonstration of data distribution based on minimum, first quartile, median, third quartile and maximum. From box plots illustrated in Fig. 4.3 it can be seen that RainToral mostly contains outliers. Data is scaled for better demonstration of features in same figure.

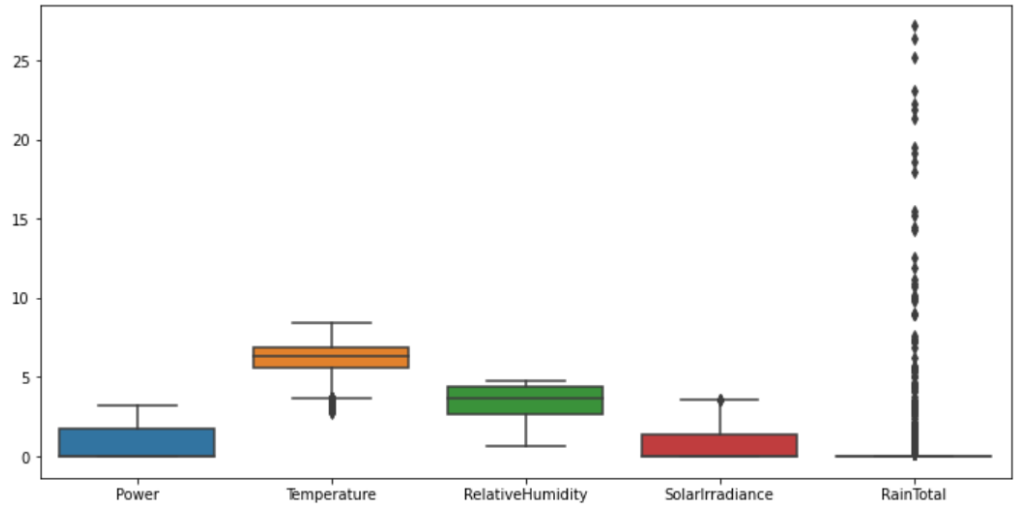


Figure 4.3. Box plots

4.1.5 Features Selection

The goal of feature selection is removing irrelevant and redundant data in order to improve performance. A proper feature subset contains features that are highly correlated with the output but are uncorrelated with each other [24]. For better demonstration of correlation between features, correlation matrix is demonstrated in Fig. 4.4.

Correlation between features is defined by a number between [−1,1]. Negative correlation coefficient indicates there is negative relationship between variables, which means increase in one variable results in decrease in another variable, and vice versa. Absolute value of correlation coefficient demonstrates intensity of correlation between the features. 0 indicates there is no correlation between the two features. |1| means variables are completely correlated.

Fig. 4.4 shows strong correlation between power and irradiance (0.86). Moreover, there is a moderate positive correlation between power and temperature (0.52) and a moderate negative correlation exists between power and relative humidity (-0.59), but there is no correlation between power and rain fall amount. For this reason, the selected features are power, solar irradiation, relative humidity, and temperature. For better comparison, analysis is performed both in single variant mode by considering only power and in multivariate mode by considering all the selected features for power prediction.

Adam [25] optimization is used as optimization method for this project.

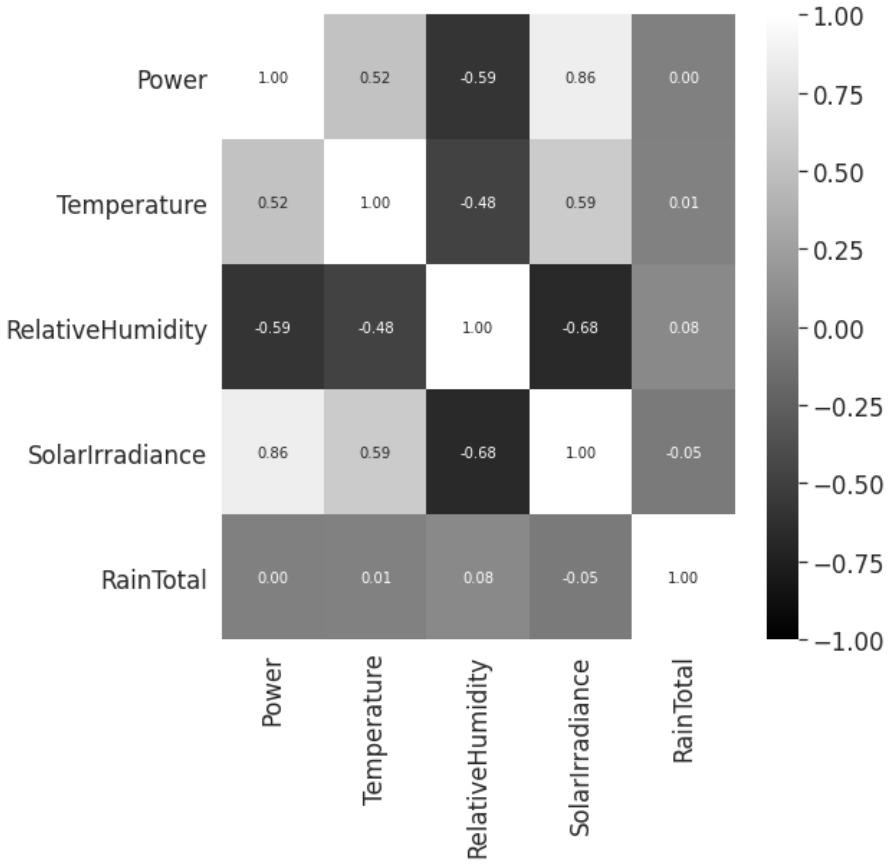
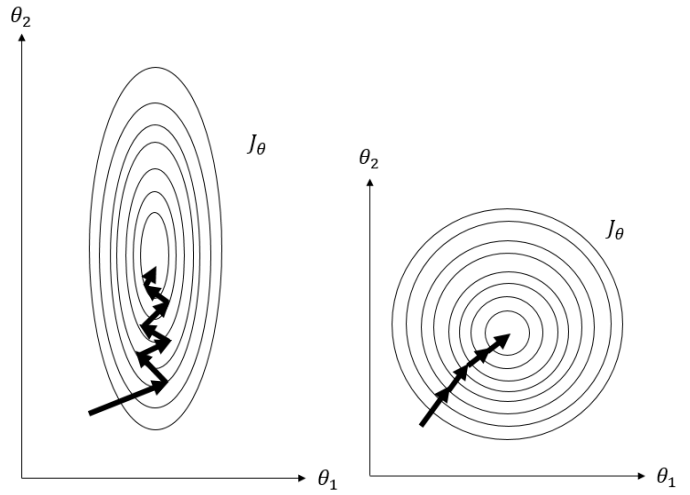


Figure 4.4. Correlation matrix

4.1.6 Data Scaling

Cost function is easier and faster to optimize when features have similar scale. Otherwise, there could be more emphasis on parameters with larger scale. Therefore, all the features are scaled by maximum value of the corresponding feature. Effect of scaling is illustrated in Fig. 4.5 on cost function optimization.



(a) Feature with larger scale dominates the update

(b) Scaled parameters have similar influence

Figure 4.5. Effect of features scaling on cost function optimization

4.2 Performance Metrics

Supposing power time series is $Y_t = y_1, y_2, ..., y_t$, $\hat{Y}$ is predicted time series, t indicates time and Ω_{Test} is the set of time stamps used for testing.

Two evaluation metrics are considered for better comparison of results.

4.2.1 Root Relative Squared Error (RSE)

RSE is scaled version of Root Mean Square Error (RMSE). The reason of choosing this metric is having more readable evaluation, regardless of data scale [13]. The lower the value of RSE the better is the result.

$$RSE = \frac{\sqrt{\sum_{t \in \Omega_{Test}} (Y_t - \tilde{Y}_t)^2}}{\sqrt{\sum_{t \in \Omega_{Test}} (Y_t - mean(Y))^2}} \tag{4.1}$$

4.2.2 Empirical Correlation Coefficient (CORR)

Empirical Correlation Coefficient shows how correlated is prediction with respect to real output. Higher values for correlation are better.

$$CORR = \frac{\sum_t (Y_t - mean(Y))(\tilde{Y}_t - mean(\tilde{Y}))}{\sqrt{\sum_t (Y_t - mean(Y))^2 (\tilde{Y}_t - mean(\tilde{Y}))^2}} \tag{4.2}$$

4.3 Hyperparameters Tuning

In order to find best hyperparameters, hyperparameters tuning is necessary. The process of finding best hyperparameters to have better accuracy is called hyperparameter tuning. Different approaches exist for hyperparameter tuning including manual tuning, grid search and random search.

- In manual tuning, based on current value of hyperparameters and obtained results new hyperparameters are chosen manually without automation of selection process.
- In grid search model is trained for every possible combination of hyperparameters values and selects the best performing set of hyperparameters on validation set.
- Random search does not try all the hyperparameters sets and chooses hyperparameter sets randomly instead of performing exhaustive search.

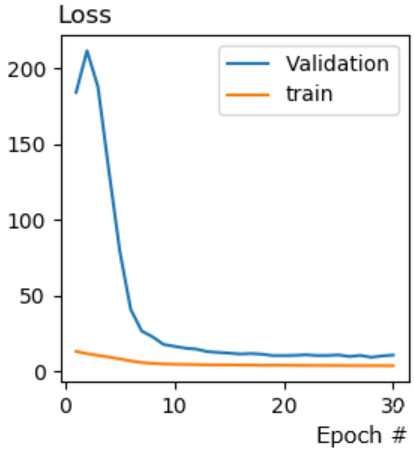
Selecting few sets of hyperparameters decreases the chance of finding the best combination. And selecting too many hyperparameter sets increases processing time.

Some hyperparameters are more important with respect to other hyperparameters and their value has more influence on prediction accuracy. In this research grid search method was chosen for hyperparameters tuning and window length, learning rate and epoch numbers are tuned.

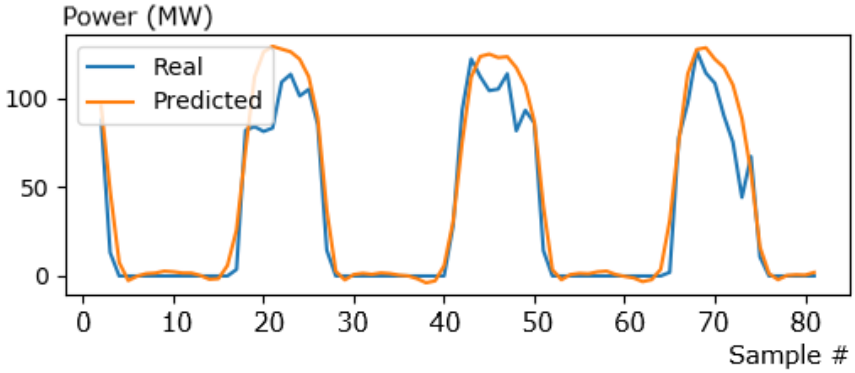
Model is trained for 30 epochs at each set and the number of epochs that obtained best results on validation set is chosen and used for testing the model on test set.

The results of hyperparameters tuning for TCN architecture are illustrated in Fig. 4.6-4.14. In all the simulation results regarding hyperparameters tuning prediction horizon is set to 24 hours.

WindowLength = 24, LearningRate = 0.0001, MaxEpochNumber =30



(a) Learning Curve

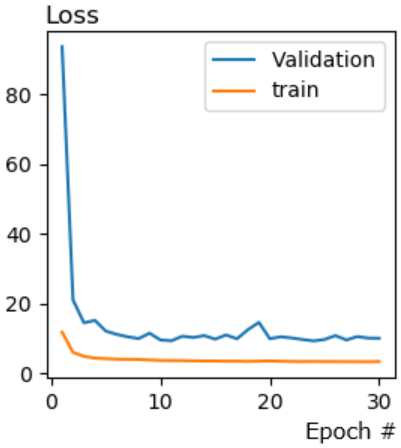


(b) Simulation Results

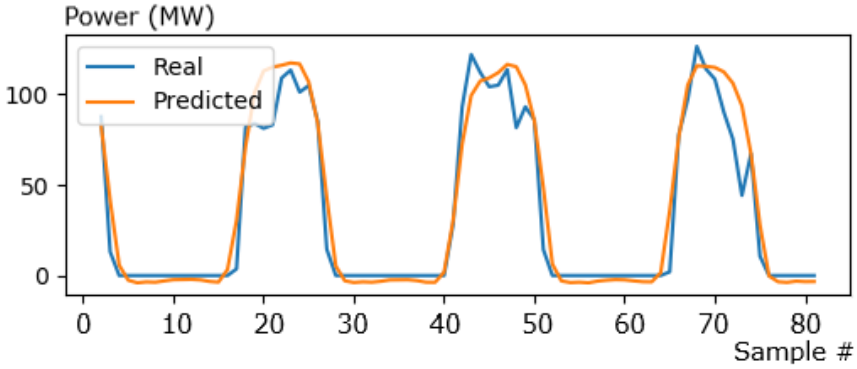
Figure 4.6. Learning and simulation results, window = 24, lr = 0.0001, max-epoch = 30

Test set RSE = 0.4994
Test set CORR = 0.8875

WindowLength = 24, LearningRate = 0.001, MaxEpochNumber =30



(a) Learning Curve

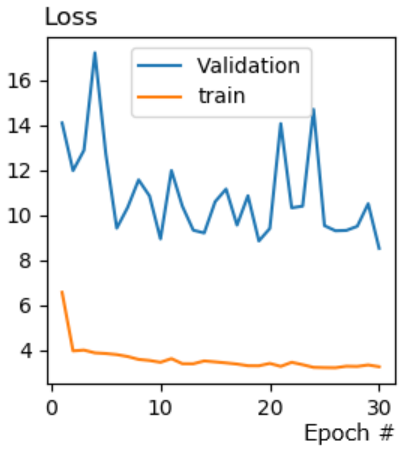


(b) Simulation Results

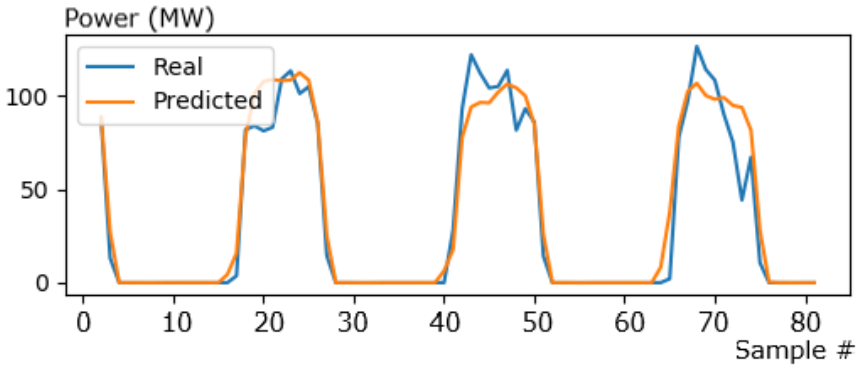
Figure 4.7. Learning and simulation results, window = 24, lr = 0.001, max-epoch = 30

Test set RSE = 0.4804
Test set CORR = 0.8840

WindowLength = 24, LearningRate = 0.01, MaxEpochNumber =30



(a) Learning Curve

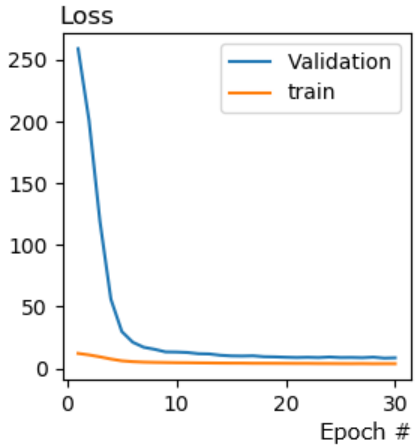


(b) Simulation Results

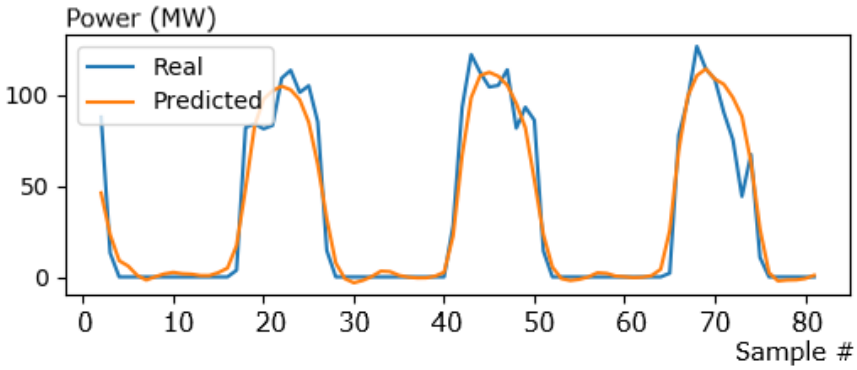
Figure 4.8. Learning and simulation results, window = 24, lr = 0.01, max-epoch = 30

Test set RSE = 0.4638
Test set CORR = 0.8877

WindowLength = 100, LearningRate = 0.0001, MaxEpochNumber =30



(a) Learning Curve

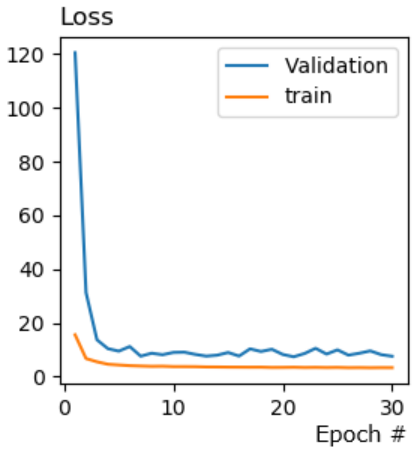


(b) Simulation Results

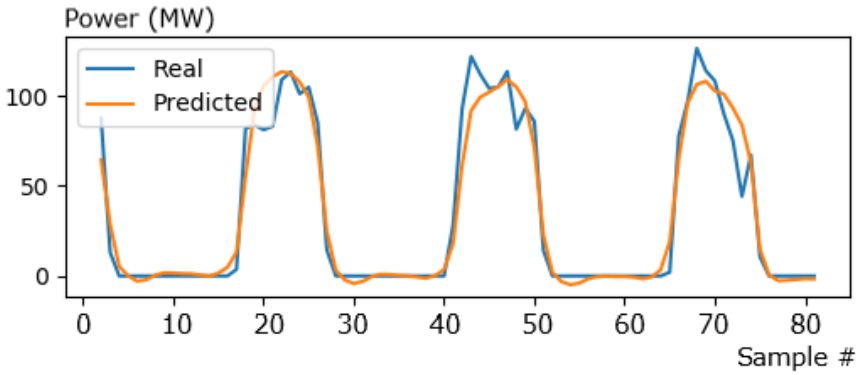
Figure 4.9. Learning and simulation results, window = 100, lr = 0.0001, max-epoch = 30

Test set RSE = 0.4679
Test set CORR = 0.8944

WindowLength = 100, LearningRate = 0.001, MaxEpochNumber =30



(a) Learning Curve

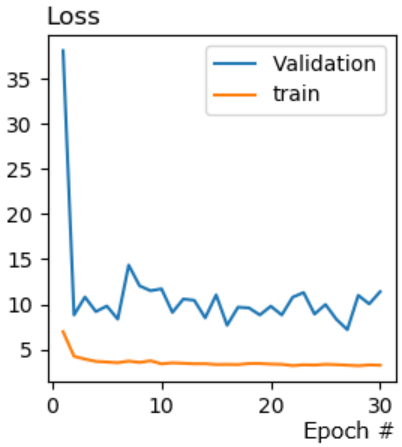


(b) Simulation Results

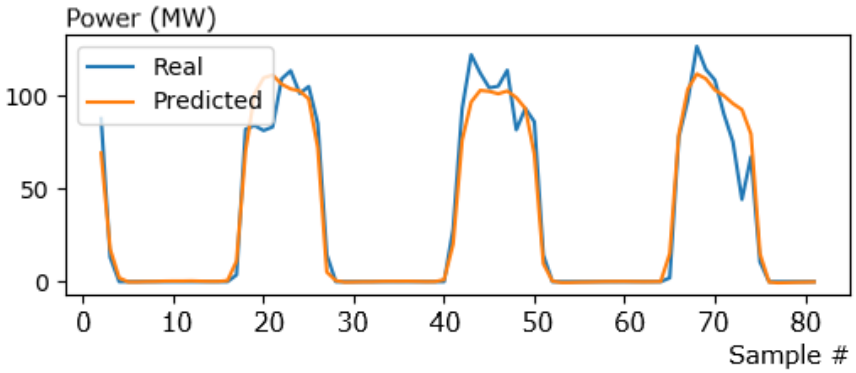
Figure 4.10. Learning and simulation results, window = 100, lr = 0.001, max-epoch = 30

Test set RSE = 0.4421
Test set CORR = 0.8992

WindowLength = 100, LearningRate = 0.01, MaxEpochNumber =30



(a) Learning Curve

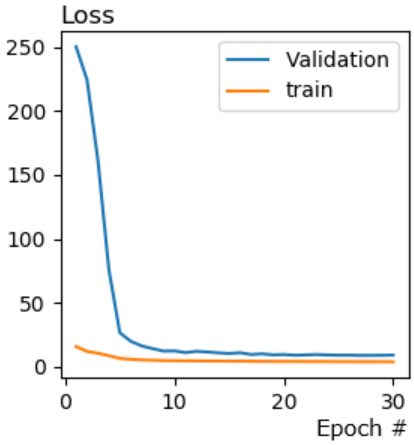


(b) Simulation Results

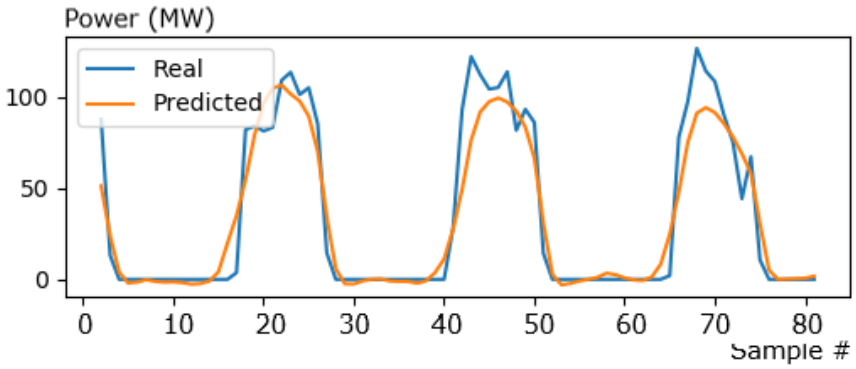
Figure 4.11. Learning and simulation results, window = 100, lr = 0.01, max-epoch = 30

Test set RSE = 0.4451
Test set CORR = 0.8991

WindowLength = 168, LearningRate = 0.0001, MaxEpochNumber =30



(a) Learning Curve

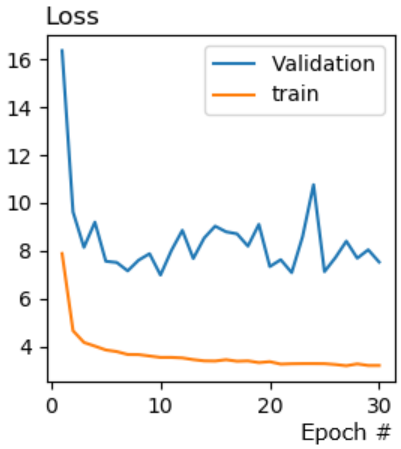


(b) Simulation Results

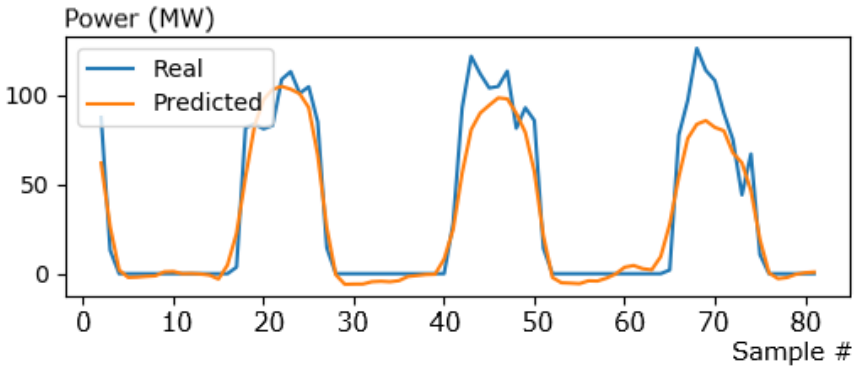
Figure 4.12. Learning and simulation results, window = 168, lr = 0.0001, max-epoch = 30

Test set RSE = 0.4755
Test set CORR = 0.8844

WindowLength = 168, LearningRate = 0.001, MaxEpochNumber =30



(a) Learning Curve

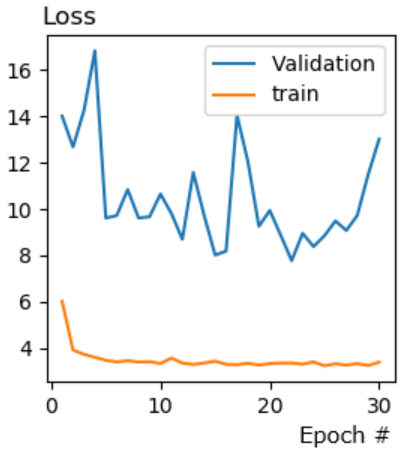


(b) Simulation Results

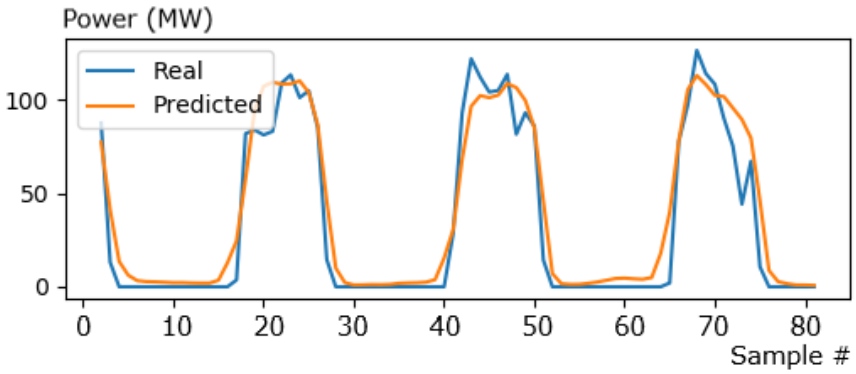
Figure 4.13. Learning and simulation results, window = 168, lr = 0.001, max-epoch = 30

Test set RSE = 0.4524
Test set CORR = 0.8941

WindowLength = 168, LearningRate = 0.01, MaxEpochNumber =30



(a) Learning Curve



(b) Simulation Results

Figure 4.14. Learning and simulation results, window = 168, lr = 0.01, max-epoch = 30

Test set RSE = 0.4801
Test set CORR = 0.8847

Comparison

Tab. 4.2 summarizes hyperparameters tuning results. Best set of hyperparameters is (window length = 100, learning rate = 0.001 and number of epochs = 6).

Window Length	Learning Rate	Max Epoch Number	Best Epoch Number	Test RSE	Test CORR
24	0.0001	30	28	0.4994	0.8875
24	0.001	30	6	0.4804	0.8840
24	0.01	30	21	0.4638	0.8877
100	0.0001	30	28	0.4679	0.8944
100	0.001	30	6	0.4421	0.8992
100	0.01	30	10	0.4451	0.8991
169	0.0001	30	16	0.4755	0.8844
169	0.001	30	4	0.4524	0.8941
169	0.01	30	2	0.4801	0.8847

Table 4.2. Hyperparameters tuning results summary

Learning Rate is a hyper-parameter that controls how much the weights of the network will be adjusted. Generally, large learning rate allows the model to learn faster, at the cost of arriving on a sub-optimal final set of weights. A smaller learning rate may allow the model to achieve higher prediction accuracy, but it may take significantly longer time to train. A very large learning rate can cause undesirable divergent behavior in the loss function. Learning rate values used for hyperparameters tuning are 0.01, 0.001, 0.0001.

Window Length represents number of samples in time series that are considered as input for prediction. Sliding window is used in this research. Concept of sliding window is illustrated in Fig. 4.15. The window slid over time series. At each step elements within window are used as input of the model to make predictions, then the window is slid further and this procedure repeats. Larger windows consider more information to make predictions but very large window can decrease sensitivity of the predictions and results in very smooth predictions. Window length values used for hyperparameters tuning are 24, 100, 168.

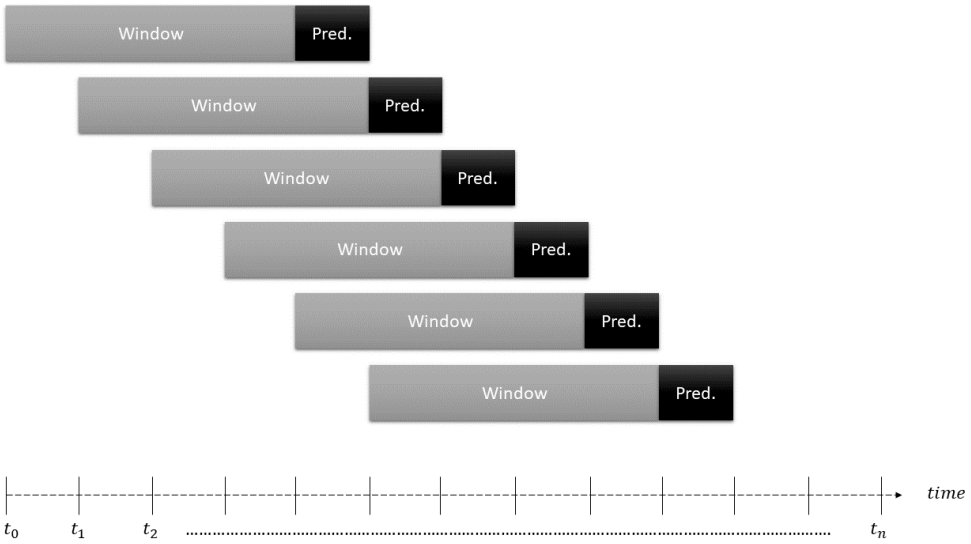


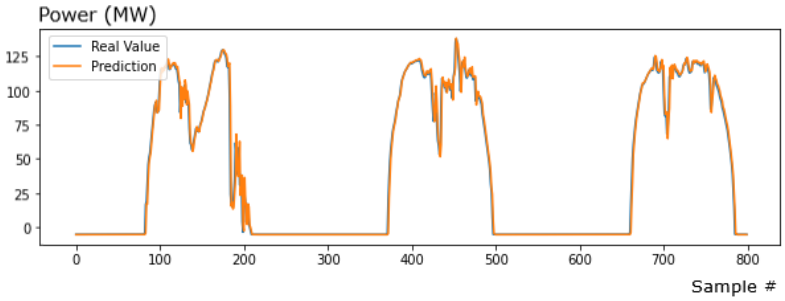
Figure 4.15. Sliding window illustration

4.4 Model Architecture Selection

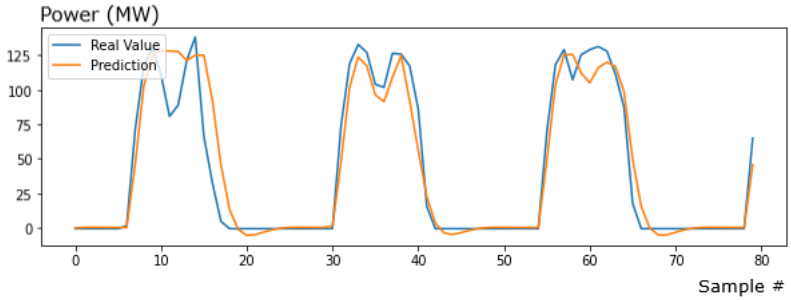
Results obtained from multivariate and univariate time series forecasting using LSTM, LSTNet and TCN architectures are summarized in Tab. 4.3. In multivariate forecasting power is predicted from power and meteorological variables. In univariate forecasting power is only predicted from data regarding power in previous timestamps. Prediction horizon is set to 5 minutes and 24 hours in univariate mode. Since sampling rate is 5 minutes in univariate prediction both short-term and very short-term horizons can be investigated. On the other hand, in multivariate prediction sampling rate is one hour, and prediction horizon is set to 24 hours.

4.4.1 LSTM

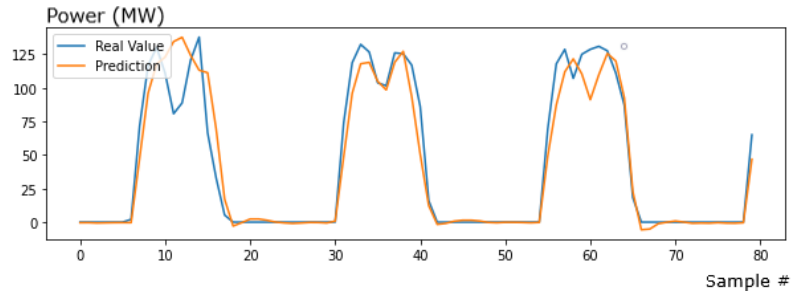
Simulation results using LSTM architecture are illustrated in Fig. 4.16.



(a) Univariate prediction, Prediction horizon = 5 Minutes, RSE = 0.1466, CORR = 0.9895



(b) Univariate prediction, Prediction horizon = 24 Hours, RSE = 0.5634, CORR = 0.8385

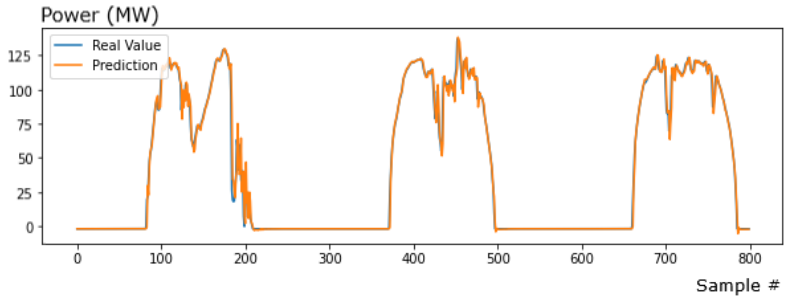


(c) Multivariate prediction, Prediction horizon = 24 Hours, RSE =0.5162, CORR = 0.8633

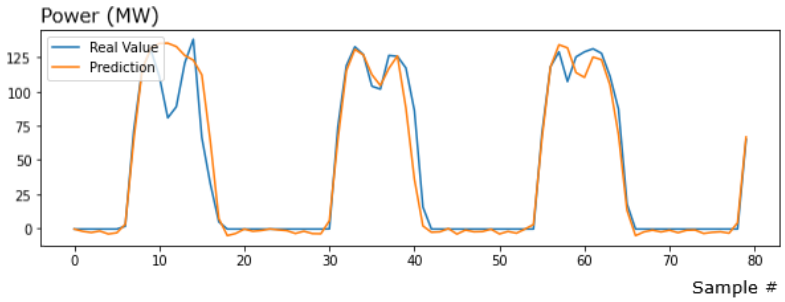
Figure 4.16. Power prediction using LSTM

4.4.2 LSTNet

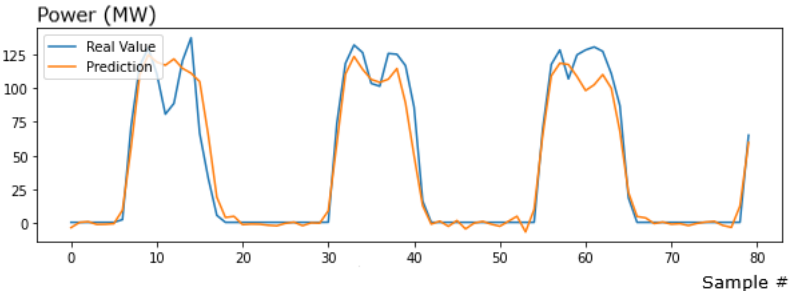
Simulation results using LSTNet architecture are illustrated in Fig. 4.17.



(a) Univariate prediction, Prediction horizon = 5 Minutes, RSE = 0.1389, CORR = 0.9904



(b) Univariate prediction, Prediction horizon = 24 Hours, RSE = 0.4891, CORR = 0.8827

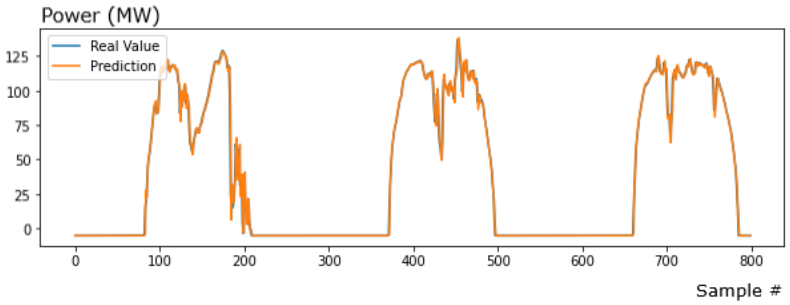


(c) Multivariate prediction, Prediction horizon = 24 Hours, RSE =0.4613, CORR = 0.8904

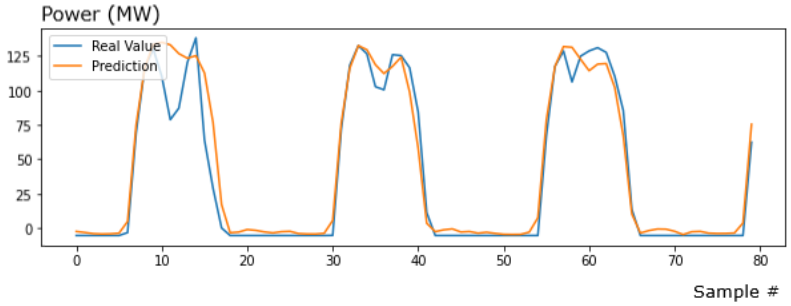
Figure 4.17. Power prediction using LSTNet

4.4.3 TCN

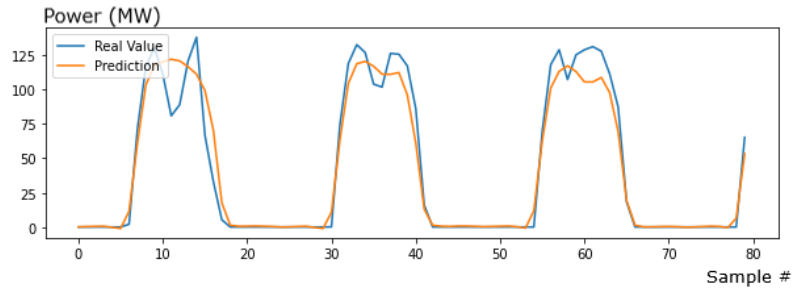
Simulation results using TCN architecture are illustrated in Fig. 4.18.



(a) Univariate prediction, Prediction horizon = 5 Minutes, RSE = 0.1412, CORR = 0.9901



(b) Univariate prediction, Prediction horizon = 24 Hours, RSE = 0.4767, CORR = 0.8972



(c) Multivariate prediction, Prediction horizon = 24 Hours, RSE =0.4421, CORR = 0.8992

Figure 4.18. Power prediction using TCN

4.4.4 Comparison

		Univariate		Multivariate
		Horizon		Horizon
Method	Metric	5 Minutes	24 Hours	24 Hours
LSTM	RSE	0.1466	0.5634	0.5162
	CORR	0.9895	0.8385	0.8633
LSTNet	RSE	0.1389	0.4891	0.4613
	CORR	0.9904	0.8827	0.8904
TCN	RSE	0.1412	0.4767	0.4421
	CORR	0.9901	0.8972	0.8992

Table 4.3. Results summary of all methods

For better comparison of methods, results of output power prediction in same time span is also demonstrated in Fig. 4.16-4.18.

Multivariate approaches have better results in terms of accuracy and correlation with respect to univariate approaches. As expected by increasing prediction horizon, accuracy of prediction decreases and forecasting very large horizons (for example 24 hours) is a challenging task. In case prediction horizon is set to 24 hours, TCN outperforms LSTNet and LSTM. When prediction horizon is set to 5 minutes, all methods have acceptable performance and LSTNet has the best performance in this case. LSTNet performs better that simple LSTM due to enhancements of architecture. LSTM has the simplest architecture among the three models and has higher RSE error with respect to other models. On the other hand, LSTNet that takes advantage of both CNN networks and RNN networks has impressive results, but the problem is high computational cost due to complex architecture. At last, TCN architecture which is the newest architecture among the three architectures and has medium level of architecture complexity was able to achieve the best simulation results. This conclusion is very promising in sense of use of Convolutional Neural Networks for time series prediction tasks instead of Recurrent Neural Networks that have been traditionally used for time series predictions.

4.5 Model Performance Improvements

In order to improve prediction accuracy model can be improved by applying some modifications. Two approaches were chosen for performance improvements. First approach is about modifying the best performing architecture to further improve the prediction accuracy. For, this reason TCN architecture which is the best performing model among the investigated models is chosen. Some modifications were applied using TCN architecture and results of the modifications is studied in section 4.5.1.

Since in cloudy days much less power is generated with respect to sunny days, power value varies significantly from day to day in the dataset. This results in lower prediction accuracy since variability in power value is significant. Dataset clustering approach can alleviate this issue by clustering the dataset into sunny and cloudy days so that days having

more similar trend are clustered into same category. The influence of dataset clustering is studied in section 4.5.2.

4.5.1 Architectural Modifications Using TCN

Inspired by LSTNet architecture, a convolutional component can be added to the architecture to extract short-term pattern and find dependencies among variables. The convolutional component is added before TCN architecture and output of the convolutional component is fed to TCN. The proposed architecture is depicted in Fig. 4.19.

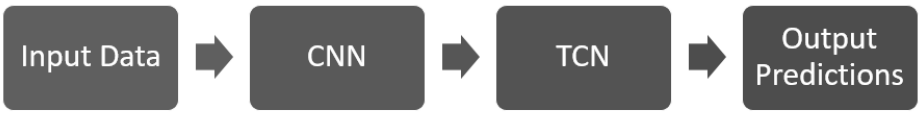


Figure 4.19. Proposed architectural modifications

Input data can be formulated as $X_T = \{y_1, y_2, ..., y_T\} \in R^{n \times T}$ which T denotes the timestamp and n is number of variables. The convolutional layer filters has width ω and height n. The height of the filters is set to the number of variables. Sweeping k^{th} filter through input data produces the following output:

$$h_k = RELU(W_k * X + b_k) \tag{4.3}$$

h_k is output vector of matrix, $*$ shows convolution operation and $RELU$ is the RELU activation function formulated as $RELU(x) = max(0, x)$. h_k has length T due to applied zero padding on matrix X. The input of TCN is output of convolutional layer which has size of $d_c \times T$. d_c is number of filters. The rest of the architecture is kept intact.

The simulation results are illustrated in Fig. 4.20. The prediction results RSE value is 0.4437 and $CORR$ value is 0.9039. The results did not show any significant improvements, therefore TCN architecture is still the best performing architecture.

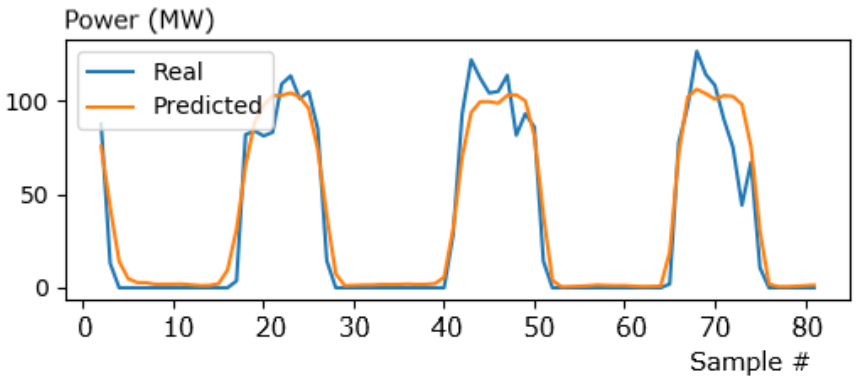


Figure 4.20. Simulation results of the modified architecture

4.5.2 Dataset Clustering

In order to deal with variability of power value from day to day which has made prediction task more challenging, dataset is clustered into sunny and cloudy days [26] and each set of days are trained on a separate model. In this method two models are used, one model for sunny days and the other for the cloudy days. Sunny days are trained on the model dedicated to sunny days and this model will be used for power prediction of sunny days. Also, the situation is similar for the cloudy days. By this way the days that are more like each other in sense of generated power are clustered in same category and variability of data is reduced within each set. The effect of data clustering on prediction accuracy is investigated in this section.

First step is clustering the dataset. Since solar irradiance is highly correlated with power output as discussed earlier in section 4.1, the dataset can be classified by considering the mean value of irradiance during each day. Clear sky condition can be defined as absence of visible clouds across the sky and clear sky irradiance is the global horizontal irradiance (GHI) that occurs during clear sky condition. Clear sky models estimate the clear sky irradiance as a function of location and atmospheric conditions. To compute clear sky irradiance PVLIB python library is used [27]. PVLIB *get_clearsky()* function receives location's latitude, longitude and elevation level from sea and requested time span to deliver the time series of clear sky irradiance for the corresponding time span. Different clear sky models can be used to compute clear sky irradiance. Ineichen clear sky model is used for estimating clear sky irradiance during this research [28].

Clear sky models are used to compute clearness index (K) which is computed by dividing the irradiation ($G_{Irradiance}$) by the estimated irradiation under clear sky conditions ($G_{ClearSkyIrradiance}$) (i.e. the output of clear sky model)

$$K = \frac{G_{Irradiance}}{G_{ClearSkyIrradiance}} \tag{4.4}$$

A threshold value could be set for clearness index so that if clearness index is higher than the threshold the day is classified as sunny otherwise it is classified as cloudy. Higher threshold values consider less days as sunny and lower threshold values considers more days as sunny.

Dataset Clustering Approach

This section focuses on predicting next day generated power. Three Forecasting modules are required in this approach. For better result comparison experiments were executed with LSTM, LSTNet and TCN. The modules are named 'Irradiance Module', 'Sunny Module' and 'Cloudy Module'. Flowchart of the proposed procedure is illustrated in Fig. 4.21. The elaborated steps of the procedure are:

1. 'Irradiance Module' predicts next day solar irradiance.
2. Clearness index is computed based on predicted irradiance and output of clear sky model.
3. If clearness index is higher than the specified threshold next day is classified as sunny, otherwise next day is classified as cloudy.
4. If next day is classified as sunny, 'Sunny Model' that is only trained on sunny days is used for power prediction. Otherwise, 'Cloudy Model' that is trained on cloudy days is used for power prediction.

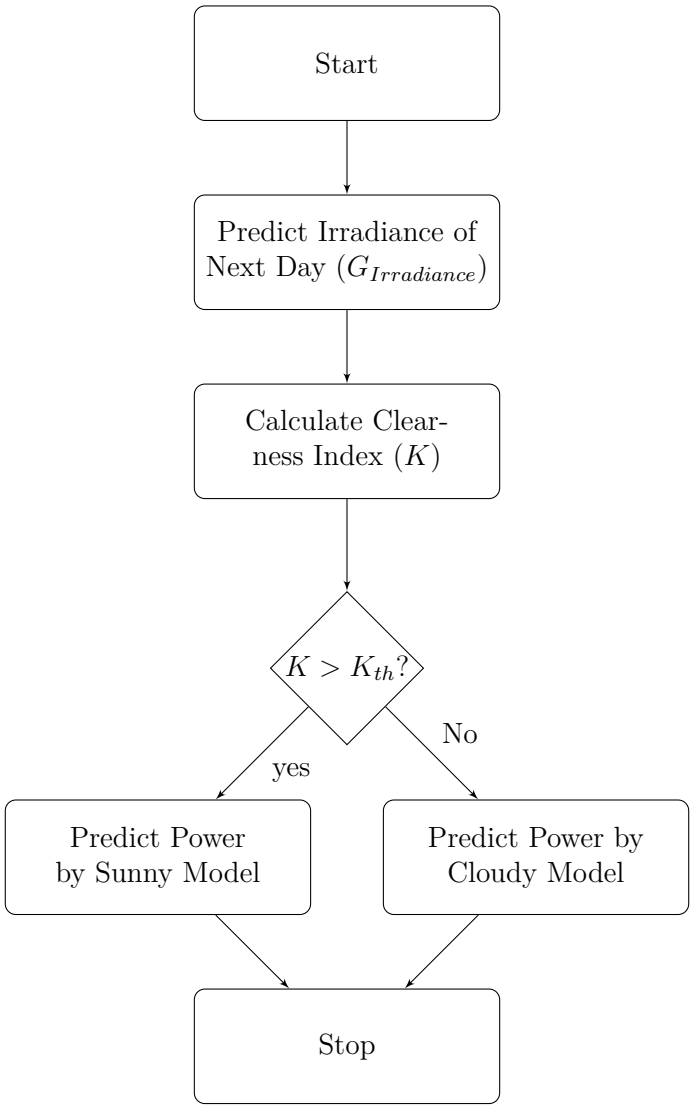


Figure 4.21. Flowchart of dataset clustering approach

4.5.3 Dataset Clustering Optimization

In order to find the optimized value of clearness index for the current dataset, multiple threshold values are used for clearness index and each time dataset is divided into sunny sub-dataset and cloudy sub-dataset. Each sub-dataset is split into train set, validation set and test set with ratio of 0.6, 0.2 and 0.2 respectively. For each sub-dataset RSE value of power prediction should be computed, therefore a model is trained on train set, best performing model on validation set is saved to be used for testing the model performance on test set. In this way the RSE value for each sub-dataset is computed, but in order to compare the performance of different threshold values RSE should be computed on

the whole dataset instead of separately on each sub-dataset. To solve this issue, after forecasting power value for test set the prediction results should be added into the original dataset. In this way real power values and prediction values both exist in the original dataset and RSE could be computed on the whole dataset. By using this method computed RSE values can be utilized to compare different threshold values of clearness index. This procedure is graphically illustrated in Fig. 4.22.

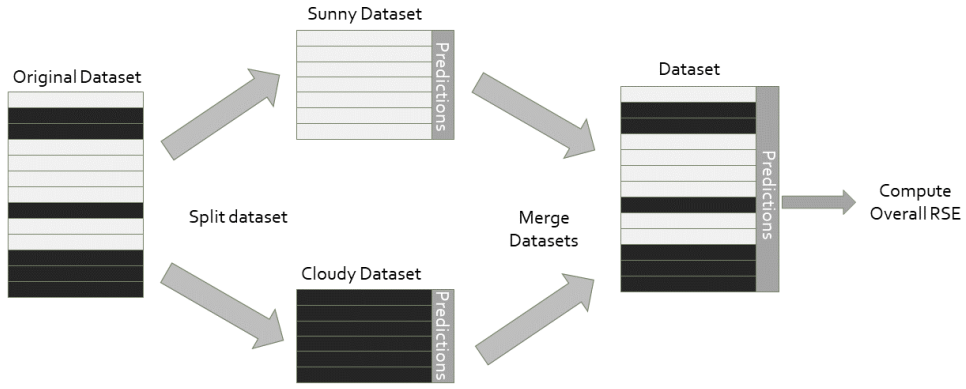


Figure 4.22. Visialization overall RSE computation

Simulations

In this sections simulation results of dataset clustering using TCN architecture are illustrated in Fig. 4.23-4.28.

Threshold = 0.6 $OverallRSE = 0.3981$, $Totalnumberofdays = 365$

Sunny Dataset $RSE = 0.3737$, $Numberofsunnydays = 317$

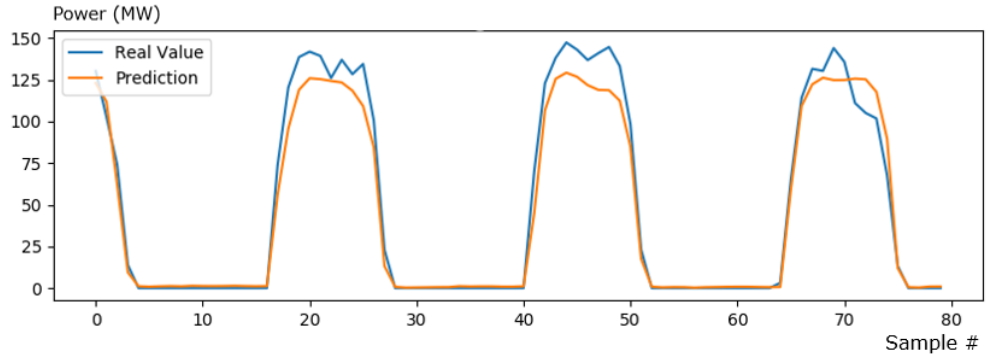


Figure 4.23. Simulation results for sunny dataset with threshold value equal to 0.6

Cloudy Dataset $RSE = 0.6136$, $Numberofcloudydays = 48$

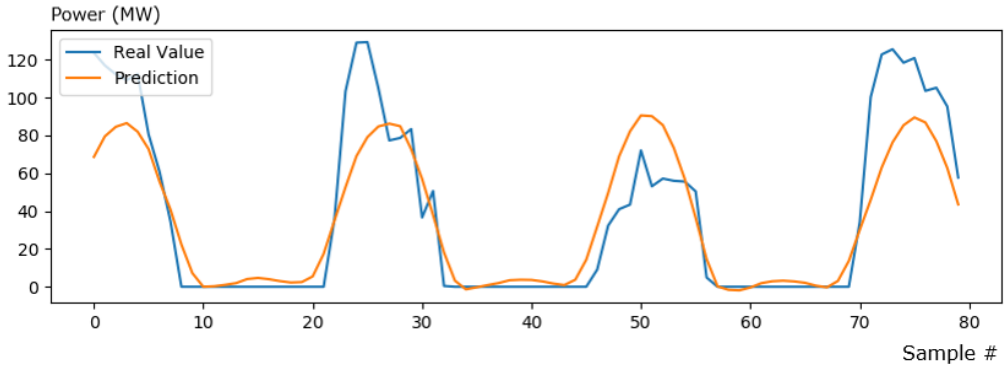


Figure 4.24. Simulation results for cloudy dataset with threshold value equal to 0.6

Threshold = 0.7 Overall RSE = 0.3982, Total number of days = 365

Sunny Dataset RSE = 0.3500, Number of sunny days = 277

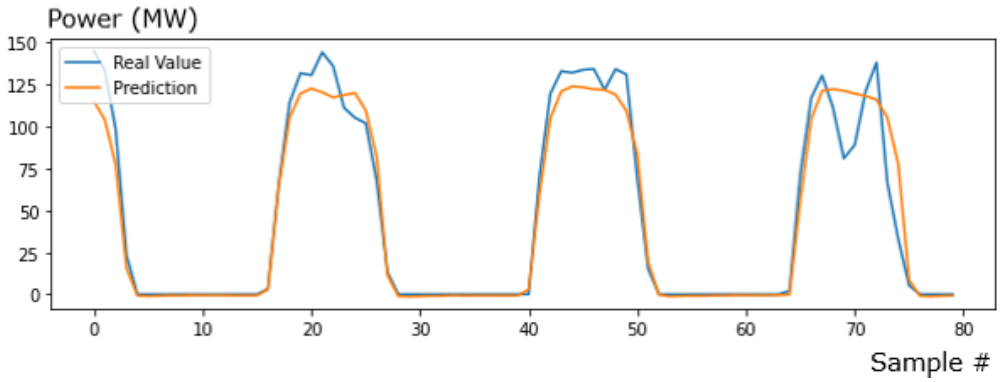


Figure 4.25. Simulation results for sunny dataset with threshold value equal to 0.7

Cloudy Dataset RSE = 0.5649, Number of cloudy days = 88

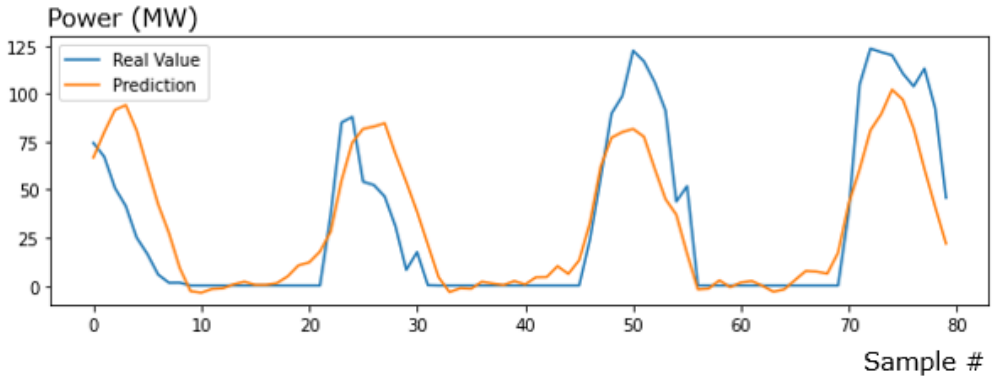


Figure 4.26. Simulation results for cloudy dataset with threshold value equal to 0.7

Threshold = 0.8 Overall RSE = 0.4301, Total number of days = 365

Sunny Dataset RSE = 0.3289, Number of sunny days = 203

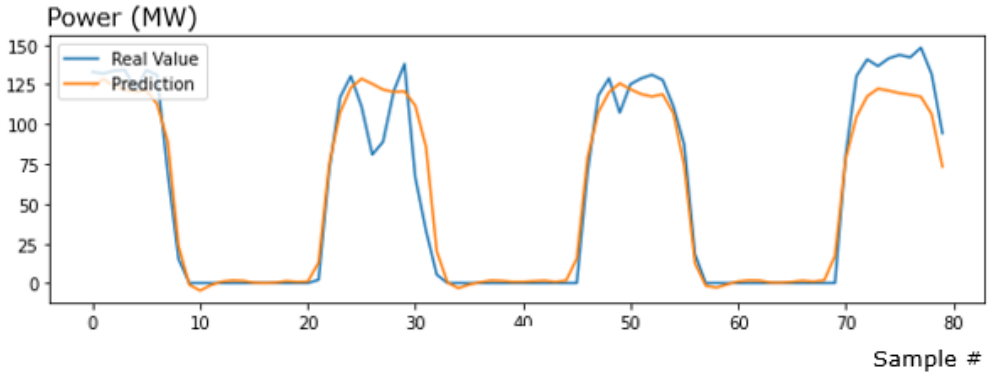


Figure 4.27. Simulation results for sunny dataset with threshold value equal to 0.8

Cloudy Dataset RSE = 0.5810, Number of cloudy days = 162

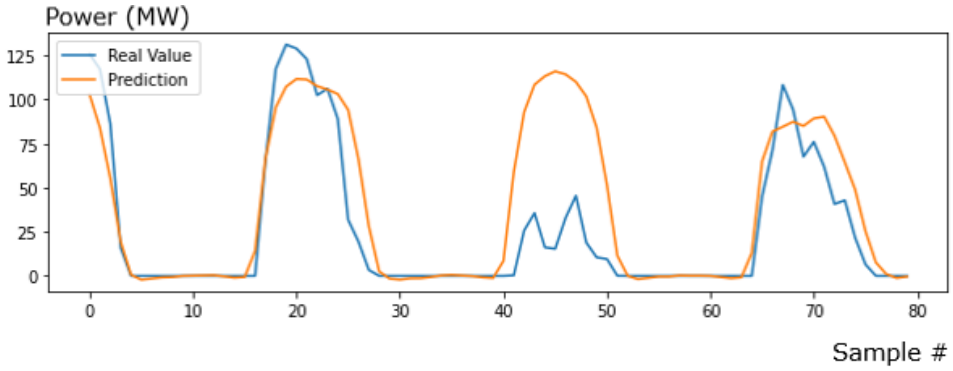


Figure 4.28. Simulation results for cloudy dataset with threshold value equal to 0.8

Results

Results of simulation without performing database clustering and with performing database clustering using different values for clearness index threshold and different architectures are summarized in Tab. 4.4

	Target	Threshold	# Days	LSTM		LSTNet		TCN		
				RSE	CORR	RSE	CORR	RSE	CORR	Overall RSE
Wole Dataset	Irradiance	-	365	0.4862	0.8767	0.4189	0.9139	0.4467	0.9165	0.4467
Whole Dataset	Power	-	365	0.5162	0.8633	0.4613	0.8904	0.4421	0.8992	0.4421
Sunny	Power	0.6	317	0.4380	0.9045	0.3932	0.9225	0.3737	0.9275	0.3985
Cloudy			48	0.9476	0.6610	0.6421	0.7926	0.6136	0.8046	
Sunny	Power	0.7	277	0.4001	0.9196	0.3499	0.9370	0.3500	0.9393	0.3982
Cloudy			88	0.7352	0.7515	0.5409	0.8494	0.5649	0.8268	
Sunny	Power	0.8	203	0.3794	0.9328	0.3391	0.9448	0.3289	0.9435	0.4301
Cloudy			162	0.6537	0.7700	0.5823	0.8400	0.5810	0.8472	

Table 4.4. Simulation results summary for Florida dataset

From Tab. 4.4 valuable information can be observed. First of all the results are consistent with results obtained from Table 4.3. TCN has the best prediction accuracy and LSTM has the worst prediction accuracy due to the reasons mentioned in previous sections.

One other observation is that clustering the dataset reduced prediction error, and from the experiments it is evident that threshold value of 0.8 has improved prediction accuracy less that other threshold values. The goal of dataset clustering is grouping the days with more similar trends together and increasing the threshold value classifies more days in cloudy category, this experiment helped us to find the appropriate threshold value for this dataset and threshold 0.6 is chosen as the best threshold value for Florida dataset.

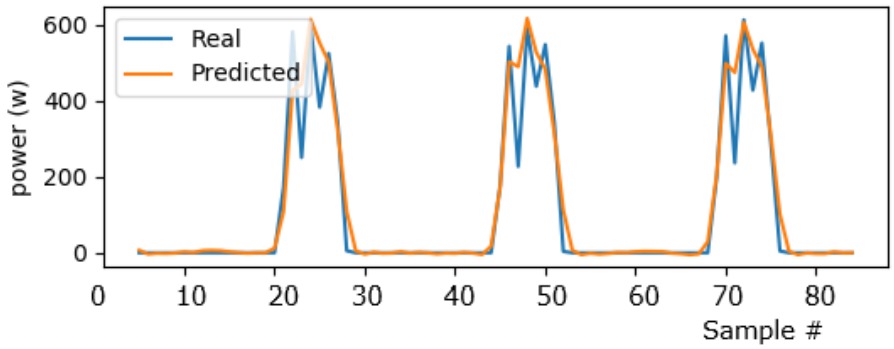
Forecasting performance for cloudy days is significantly lower than sunny days. A possible explanation could be high variability of power variable within cloudy dataset, while in sunny dataset, days usually follow the same trend and their peek value is closer to maximum obtainable value, therefore training on sunny dataset leads to more repeated predictions.

4.6 Model Performance in Different Locations

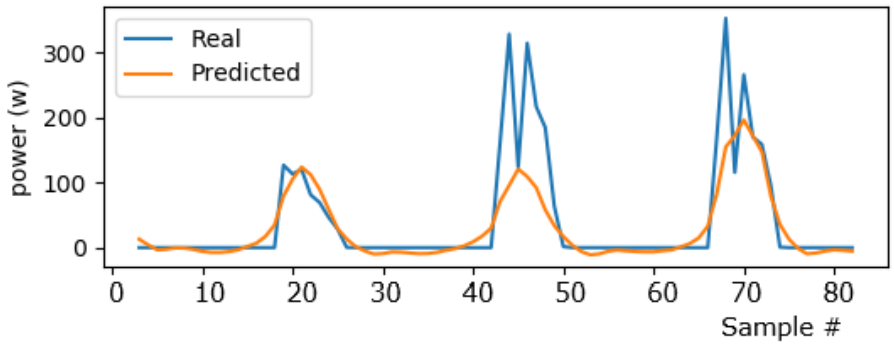
Different locations around the world have different climates. In order to understand how well model generalizes to different weather conditions the experiments were repeated on datasets containing power and weather information from Italy and Oman.

4.6.1 Italy Dataset

Threshold = 0.6



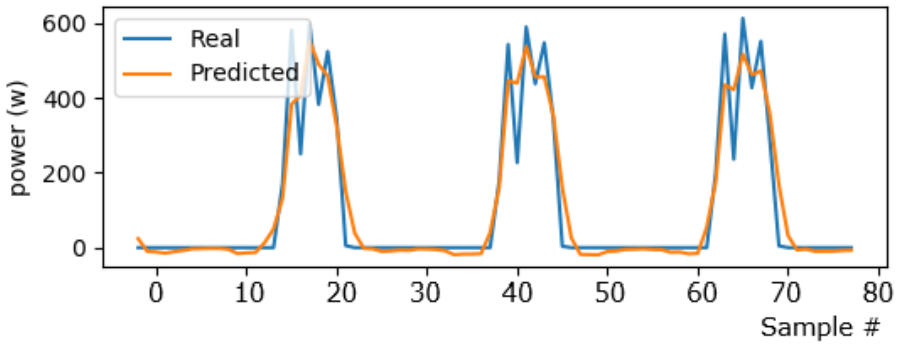
(a) Sunny Sub-dataset, Threshold = 0.6



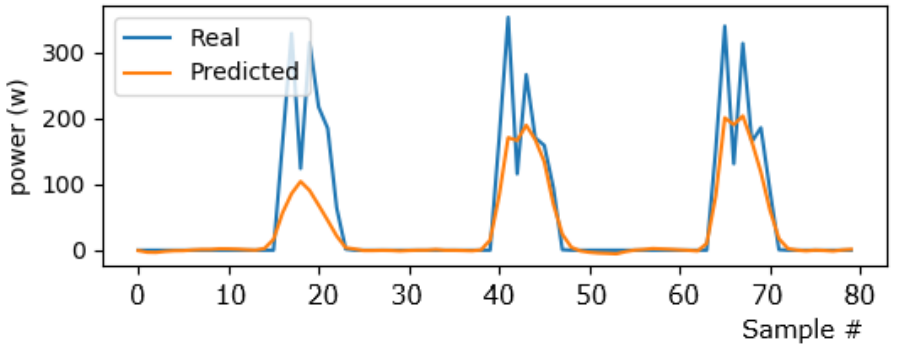
(b) Cloudy Sub-dataset, Threshold = 0.6

Figure 4.29. Italy dataset simulation results, Threshold = 0.6

Threshold = 0.7



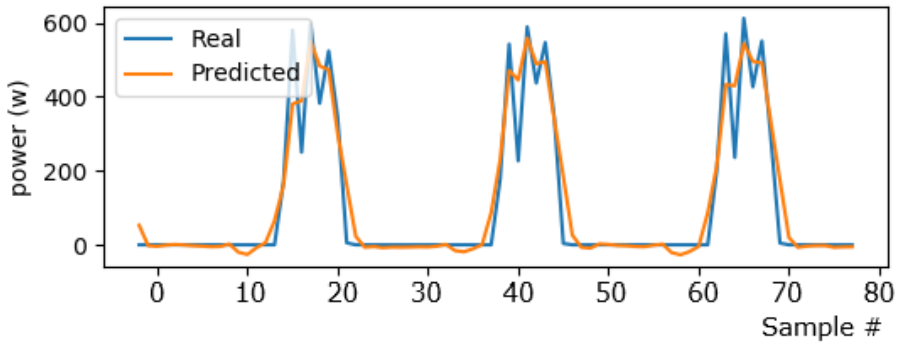
(a) Sunny Sub-dataset, Threshold = 0.7



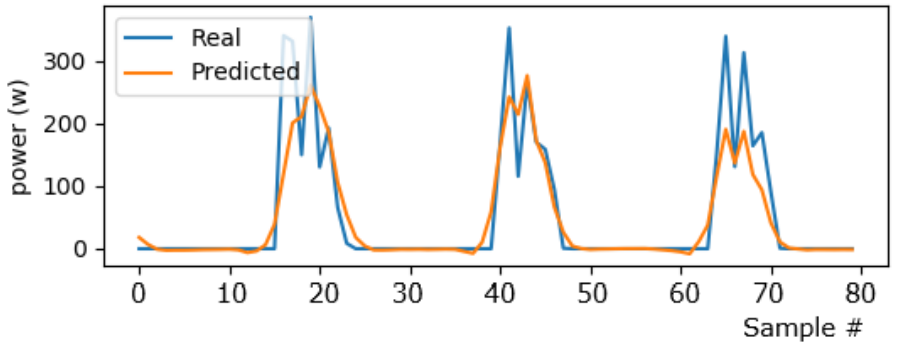
(b) Cloudy Sub-dataset, Threshold = 0.7

Figure 4.30. Italy dataset simulation results, Threshold = 0.7

Threshold = 0.8



(a) Sunny Sub-dataset, Threshold = 0.8



(b) Cloudy Sub-dataset, Threshold = 0.8

Figure 4.31. Italy dataset simulation results, Threshold = 0.8

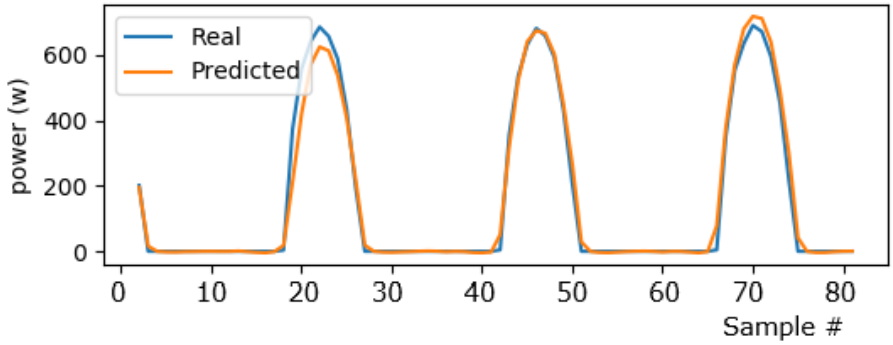
Results

Italy Dataset	Target	Threshold	# Days	TCN		
				RSE	CORR	Overall RSE
Sunny	Power	0.6	153	0.4849	0.8951	0.5451
Cloudy			213	0.7908	0.7039	
Sunny	Power	0.7	139	0.4676	0.8871	0.5467
Cloudy			227	0.7503	0.7222	
Sunny	Power	0.8	112	0.4045	0.9152	0.5317
Cloudy			254	0.6875	0.7470	

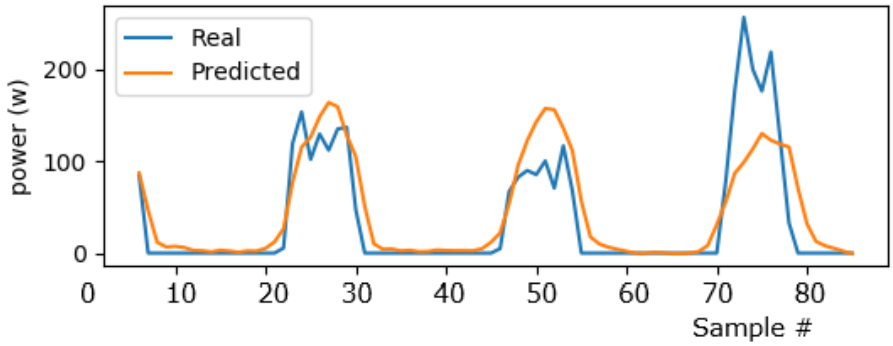
Table 4.5. Simulation results summary for Italy dataset

4.6.2 Spain Dataset

Threshold = 0.6



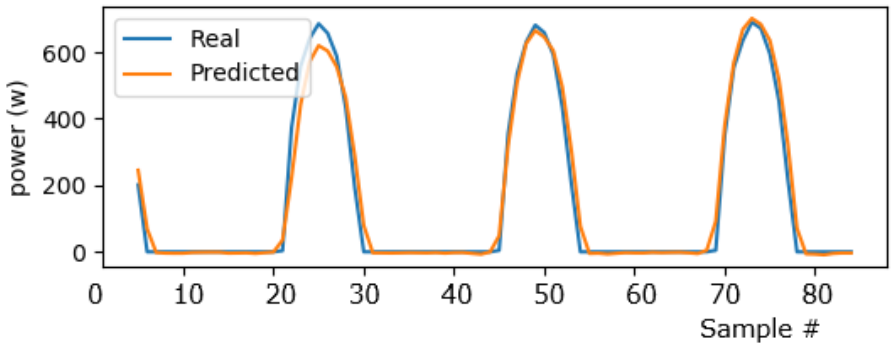
(a) Sunny Sub-dataset, Threshold = 0.6



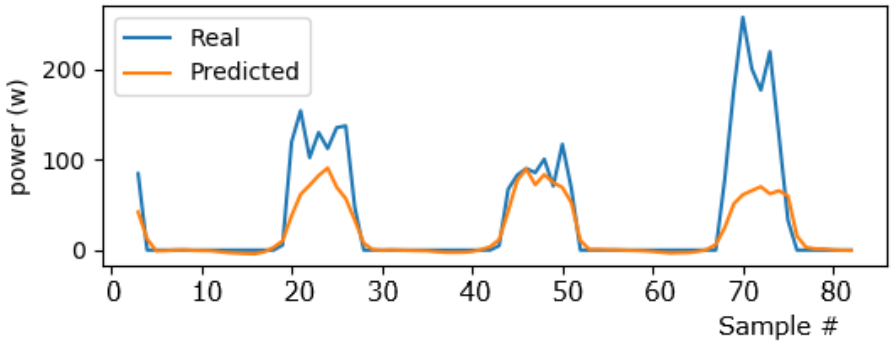
(b) Cloudy Sub-dataset, Threshold = 0.6

Figure 4.32. Spain dataset simulation results, Threshold = 0.6

Threshold = 0.7



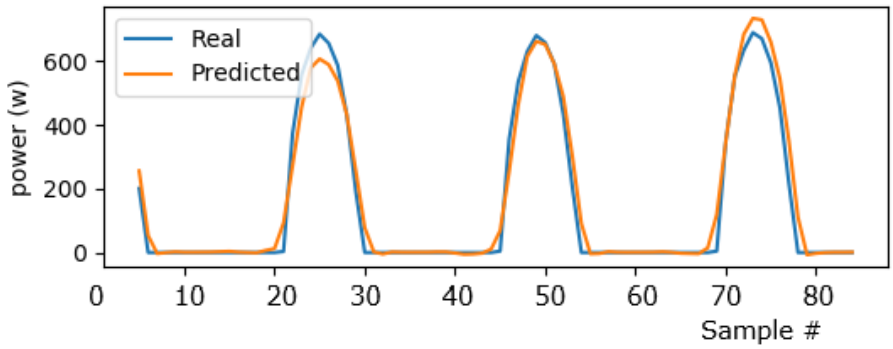
(a) Sunny Sub-dataset, Threshold = 0.7



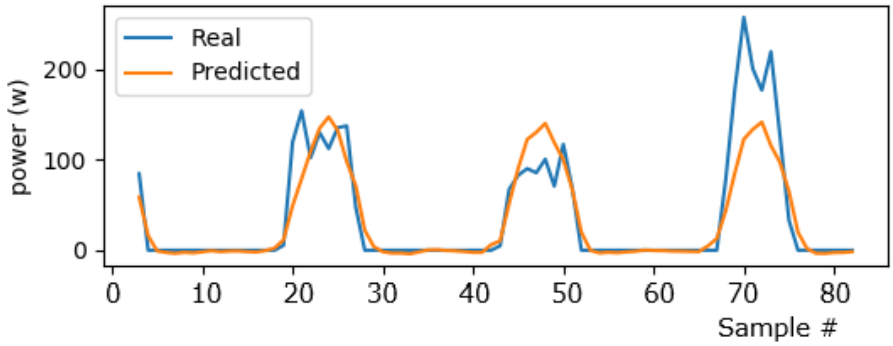
(b) Cloudy Sub-dataset, Threshold = 0.7

Figure 4.33. Spain dataset simulation results, Threshold = 0.7

Threshold = 0.8



(a) Sunny Sub-dataset, Threshold = 0.8



(b) Cloudy Sub-dataset, Threshold = 0.8

Figure 4.34. Spain dataset simulation results, Threshold = 0.8

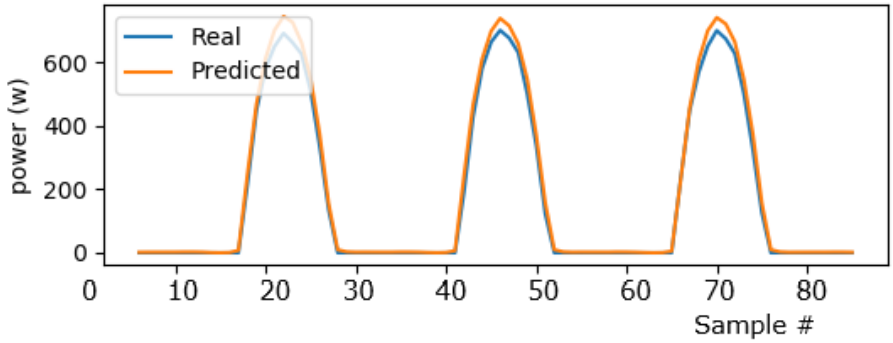
Results

Spain Dataset	Target	Threshold	# Days	TCN		
				RSE	CORR	Overall RSE
Sunny	Power	0.6	242	0.4314	0.9184	0.4811
Cloudy			124	0.8239	0.7126	
Sunny	Power	0.7	228	0.4196	0.9207	0.4909
Cloudy			138	0.7984	0.7231	
Sunny	Power	0.8	112	0.4058	0.9288	0.4788
Cloudy			254	0.6955	0.7910	

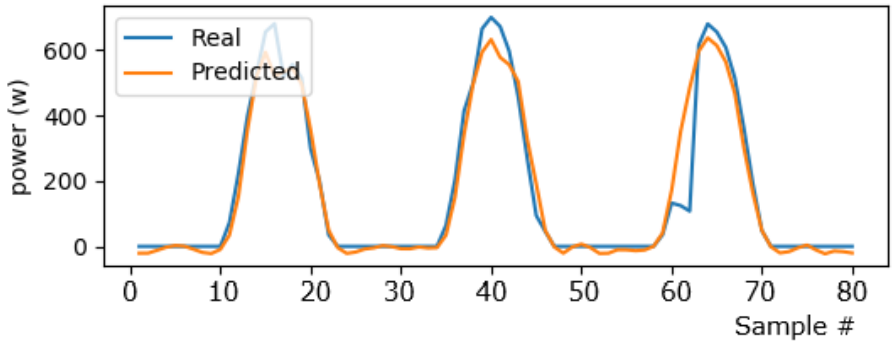
Table 4.6. Simulation results summary for Spain dataset

4.6.3 Oman Dataset

Threshold = 0.6



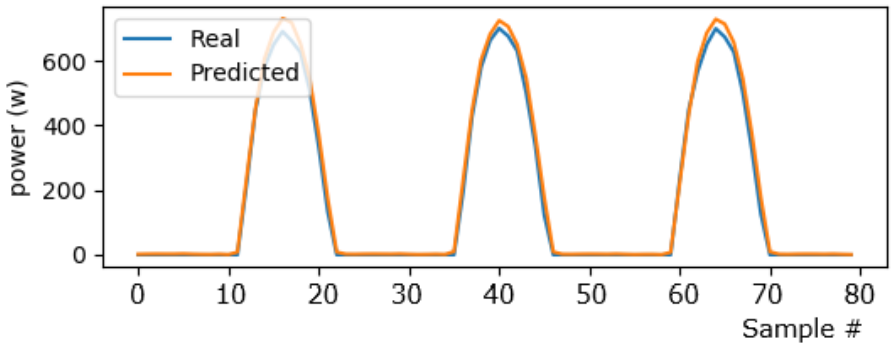
(a) Sunny Sub-dataset, Threshold = 0.6



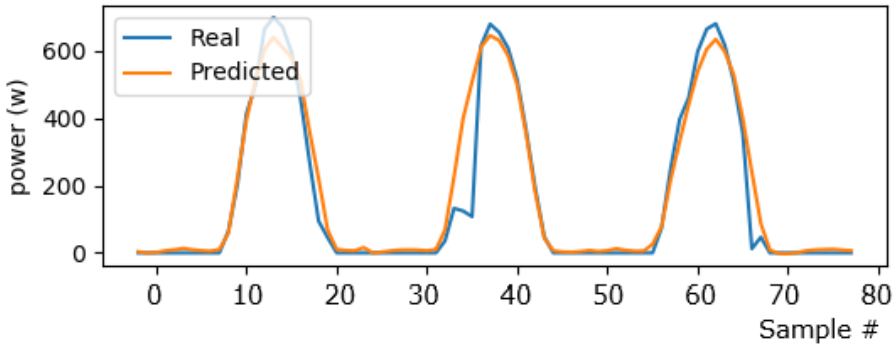
(b) Cloudy Sub-dataset, Threshold = 0.6

Figure 4.35. Oman dataset simulation results, Threshold = 0.6

Threshold = 0.7



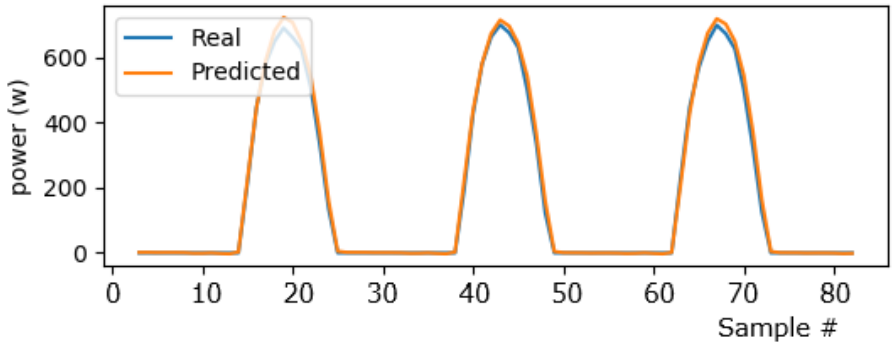
(a) Sunny Sub-dataset, Threshold = 0.7



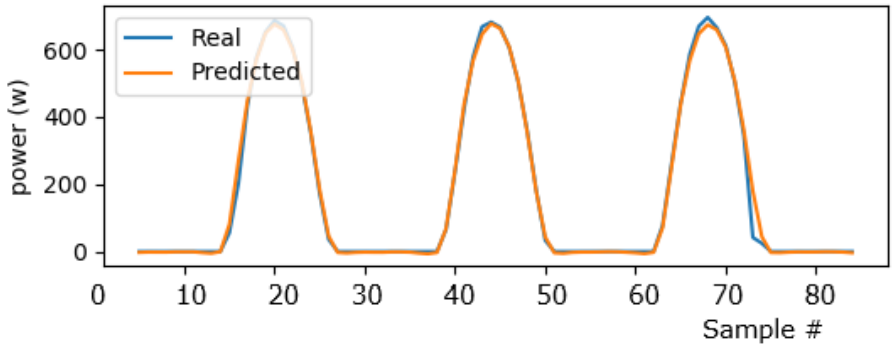
(b) Cloudy Sub-dataset, Threshold = 0.7

Figure 4.36. Oman dataset simulation results, Threshold = 0.7

Threshold = 0.8



(a) Sunny Sub-dataset, Threshold = 0.8



(b) Cloudy Sub-dataset, Threshold = 0.8

Figure 4.37. Oman dataset simulation results, Threshold = 0.8

Results

Oman Dataset	Target	Threshold	# Days	TCN		
				RSE	CORR	Overall RSE
Sunny	Power	0.6	320	0.1123	0.9949	0.1388
Cloudy			46	0.2843	0.9590	
Sunny	Power	0.7	277	0.1159	0.9940	0.1451
Cloudy			89	0.2241	0.9754	
Sunny	Power	0.8	238	0.097	0.9954	0.1207
Cloudy			128	0.1605	0.9872	

Table 4.7. Simulation results summary for Oman dataset

4.6.4 Comparison

Tab. 4.8 summarizes best model performance in different locations and climate conditions. Countries like Oman which most of the time sky is clear, more days are sunny and follow the same trend tend to have higher prediction accuracy. In weather conditions that tend to change and there are numerous cloudy days with different cloud intensity model performance decreases due to variability of power and irradiance within the dataset that makes prediction task more challenging. Optimum threshold value depends on target location's climate and it's better to be tuned for each dataset.

Dataset	Data	Best Threshold	# Days	Test RSE	Test CORR	Overall RSE
Florida	Sunny	0.7	277	0.3500	0.9393	0.3982
	Cloudy		88	0.5649	0.8256	
Italy	Sunny	0.8	112	0.4045	0.9152	0.5317
	Cloudy		254	0.6875	0.7470	
Spain	Sunny	0.8	112	0.4058	0.9288	0.4788
	Cloudy		254	0.6955	0.7910	
Oman	Sunny	0.8	238	0.0987	0.9954	0.1207
	Cloudy		128	0.1605	0.9872	

Table 4.8. Summary of model performance in different locations

Chapter 5

Deployment

For code deployment different scenarios are considered. One possible scenario is that user has access to the code and wants to run the code on an arbitrary device. In this case user should be able to modify configuration parameters to satisfy the requirements.

Solar forecast module is configured to read the configuration parameters from a specified JSON file. Users can customize configuration parameters by modifying the JSON file. In order to simplify configuration parameters customization, a Graphical User Interface (GUI) is also implemented. In this way in case GUI is not available user is still able to modify the configuration parameters by modifying the JSON file.

Another possible scenario is running the code on a remote server and exposing the service to the users through APIs.

In first scenario code can be executed on any arbitrary machine and user should be able to modify the configuration parameters. For this purpose, two methods are implemented. User can directly modify the JSON file that solar forecast module uses for reading the configuration parameters from. Another method is using a GUI to modify the JSON file and automatically starting solar forecast execution.

5.1 Configuration Parameters JSON File

Configuration parameters can be updated through a JSON file. User can modify parameters.json file. When solar forecast code runs, configuration parameters are captured from the JSON file. This implementation is clean and easy and is always available even to the users that their device does not support GUI. Fig. 5.1 shows the structure of the parameters.json file. Users are able to specify whether GPU or CPU executes the code. Moreover, number of epochs used for training the model can be modified by the user. Another configuration parameter is prediction horizon and window length. Prediction horizon specifies the time span in future that model should predict, and window length specifies the number of samples used as input of the model used for predictions. Also, user can specify the target variable. If only power is chosen as target variable predictions take place without dataset clustering. In case both power and irradiance are selected dataset is clustered into sunny and cloudy and predicted irradiance value is used for classifying the target day as sunny or cloudy then power prediction takes place.

Figure 5.1. Configuration parameters JSON file

5.2 Graphical User Interface (GUI)

GUI is implemented using Tkinter which is the standard Python interface to the TK GUI toolkit. TK is a cross-platform toolkit that provides a library of basic GUI elements. TK allows GUI implementation in different programming languages.

5.2.1 Implementation Steps

In order to use Tkinter for developing GUI application following steps are required:

1. **Import Tkinter Module**

   Tkinter library should be imported by using the usual command for importing libraries.

   ```
   from tkinter import *
   ```

2. **Implement Main Window**

   Tk() method of Tkinter library creates the main window. Main window can be implemented using the following command:

   ```
   window = Tk()
   ```

   Moreover, Tkinter mainloop method is used when application is ready for execution. Mainloop is an infinite loop that constantly refreshes the window and checks for modifications in GUI window as long as window is open.

3. **Add GUI Widgets to Main Window**

   Grid method is used to organize the widgets in table-like structure inside the window. Tkinter offers different widgets. The widgets are described in further subsections.

4. **Capture Entered Information**

Button widgets can be used for this purpose, because user is able to associate a user-defined function to each button. Therefore, when button is pressed Tkinter automatically calls the associated function.

In the associated function entered value to each widget can be captured and in this project the JSON file containing configuration parameters is updated accordingly.

5.2.2 Widgets

The widgets included in GUI implementation are explained below:

Button

In order to add button to GUI application Button widget can be used and a user-defined function can be associated to the button. When the button is pressed Tkinter automatically calls the associated function.

In the below code a button is implemented in the parent window that is shown by window parameter. "text" parameters define the text that is displayed on the button and "command" parameter calls the user-defined function when button is clicked.

```
btn = Button(window, text = "Start", command = clicked)
```

Radio Button

Radio button allows choosing one option among multiple choices. The following code is used for radio button creation. "text" parameter defines the text that is displayed for the radio button. When a radio button is selected the value is set to the variable.

```
rad1 = Radiobutton(window,text='CPU', value='cpu', variable=
    ↪ device_value)
```

Check Button

Check buttons allow choosing multiple choices. A variable is associated to each check button. In the following code the variable is Boolean and its default value is true.

```
prediction_1_state = BooleanVar()
prediction_1_state.set(True)
prediction_1 = Checkbutton(window, text='Irradiance', var=
    ↪ prediction_1_state)
```

Entry

Single line text entry is possible by this widget. The window and width of the entry box are defined in the following code:

```
epochs_value = Entry(window,width=10)
```

Label

Is a widget that can be used to display a text f image. "text" parameters define the displayed text on label widget. An example of label widget implementation is shown in the following code:

```
window_label = Label(window, text="Window length:")
```

The implemented GUI is illustrated in Fig. 5.2. GUI execution steps are described as follows:

1. gui.py execution
    - GUI window appears
2. Entering the configuration parameters in GUI window
3. Pressing the Start button in GUI window
    - JSON file containing configuration parameters is updated by new entered parameters
    - Solar forecast module starts execution
    - GUI window terminates
    - Solar forecast module captures configuration parameters from JSON file
    - Solar forecast module continues execution and computes the predictions

5.2.3 GUI Freezing Issue

When solar forecast module is executed on the same thread as mainloop method of Tkinter library, mainloop stops until the code execution is finished, this results in GUI freezing. Multi-threading programming is able to solve this issue. When solar forecast code is executed on a separate thread GUI does not freeze anymore. In order to use multi-threading programming Python threading library is used. In the implemented solution main thread creates two threads. One thread is responsible of GUI and the other thread executes the solar forecast module. Main thread starts the GUI thread and then waits for the GUI thread and the solar forecast thread to join. When user clicks on the start button second thread that is responsible for solar forecast module starts. Then GUI window is closed by using *window.destroy*() command and first thread terminates.

5.3 Remote Server

One possible scenario is to run the solar forecast code on a remote server and expose the service to clients through APIs. One of the advantages of this method is that service does not depend on the user device characteristics. Therefore, users can take advantage of the service by sending HTTP requests without any concerns about their device characteristics. Moreover, when updates are available users do not have to download any updates and they

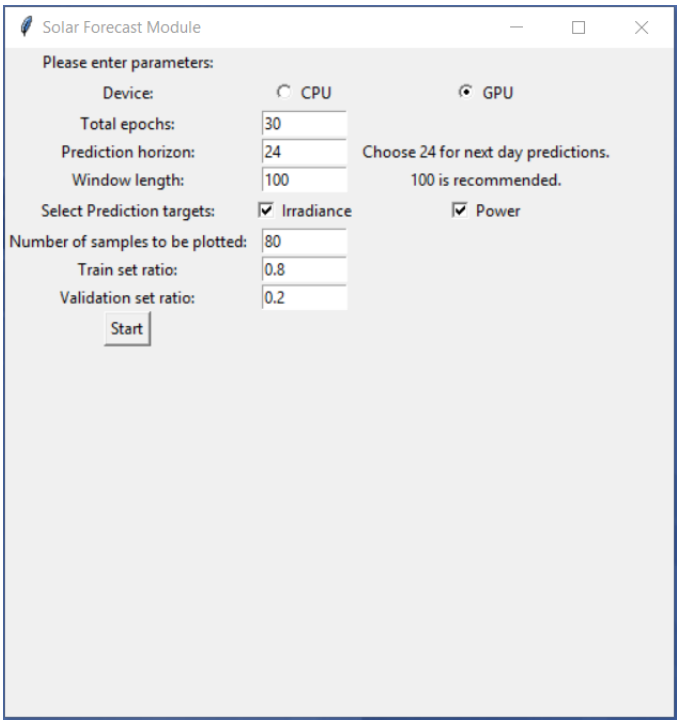


Figure 5.2. GUI implementation

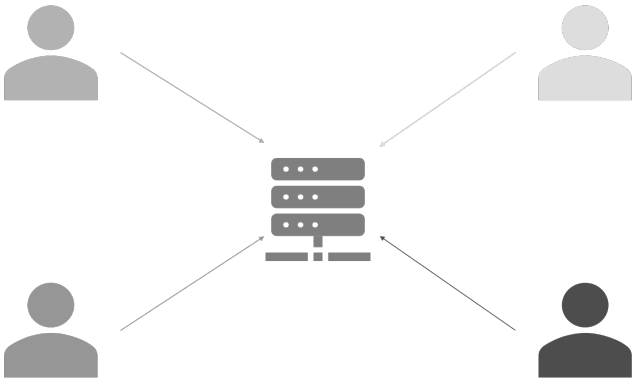


Figure 5.3. Remote server

can continue using the service without any extra effort. The main idea of using a shared server among the users is illustrated in Fig. 5.3.

To implement the paradigm illustrated in Fig. 5.3, a web server is required. The web server should be able to handle incoming HTTP requests, perform some initial checks on the current system status and hand them over to the Machine Learning engine running as the python code. Since, the webserver shall also be able to execute the CPU/GPU consuming

tasks at the same time, conventional web hosting plans are not the best choice for this task. Instead, more resources should be allocated to perform this task. At this point there are mainly two options to choose from, dedicated servers or virtual private servers. Dedicated servers can provide a great amount of resources in exchange to an expensive subscription fee. However, VPS plans are designed to exploit the powerful features of dedicated servers such as the freedom to install specific operating systems and setup the environment in a customized manner while spending less amount of money for the subscription fees. In the VPS architecture, a dedicated server shares its resources to the underlying virtual private servers. These virtual private servers are isolated from each other thanks to the virtual machine architecture. For the purpose of this research, since for the moment there is only one potential user to access the service, Virtual Private Server (VPS) is a well-fitting candidate. There are multiple web hosting and cloud infrastructure providers which rent virtual private servers. In order to minimize the costs of this research, DigitalOcean Company has been selected to rent a virtual machine and setup the remote server. In the following sections, the steps to setup the environment is introduced.

5.3.1 Environment Setup

1. The first step is to create a new account on DigitalOcean website[1].

2. As soon as the account is ready and payment method is configured, a new project can be started. The project is the location to gather all the occupied resources.

3. In DigitalOcean terms, a virtual machine running on the VPS is called a Droplet. Next step is to create a droplet and selecting the required specifications. At this step the amount of resources dedicated to the virtual private server should be carefully chosen. It will be possible to increase some of the resources such as memory in future however down-sizing is not natively supported by the platform. For the starting point of this project, a VPS with following specifications has been created. Different starting plans of DigitalOcean droplets are listed in figure Fig. 5.4.

   - Operating System: Ubuntu 18.04.3 LTS (No GUI)
   - CPU: 1 vCPU
   - Memory: 2GB
   - SSD: 50GB
   - Transfer Traffic: 2TB

   The data center can be selected based on the available locations. In order to have the VPS in Europe, Amsterdam data center has been chosen. For accessing the droplet, two methods can be utilized. A password key or SSH keys. SSH keys are more reliable and secure, so in the next step, an SSH key is generated to attach to the VPS.

4. Generate SSH key using git bash on the local PC. If ssh public and private key has not been generated for the current user on local PC, they can be generated using git

[1] https://cloud.digitalocean.com/registrations/new

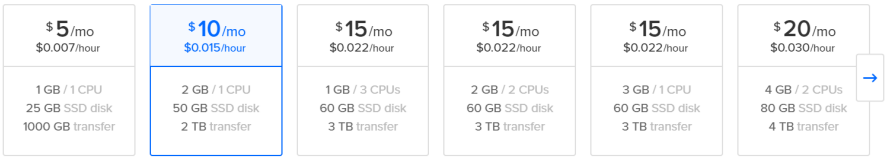


Figure 5.4. Plans for VPS resources

bash and the private key should be added to the droplet. Figure Fig. 5.5 illustrates the generated public key. The file content is added to the droplet settings.

```
Asus@Talaye MINGW64 ~/.ssh
$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCef7wgA8NSYvWFrQMUi1xdxGgIsDVCvnGXiLzVS2t0
orgR/L496fBsuoR/sSrrlNnjVkvXm9QfLf2fjmaaATcGUFst7XwmdShA1A+1aYdgI+DcHNt2oZXtADrR
JVg6CA2tgF7WXsrOGTF2HUzdZilitV4cjEg3+agq1joKXG5fCusyFLYMfhkmxSlYlU+UkKgGayM6ZZT1
W9NIPSbB9t/12OVXojl3ibsLbD+24JUVgpOKYYpT5vKPnS5ky63Px1FCYUxdmz6VhPPTEpWQQtQS31/H
yzzd//wu5ZC9fAWOD/ORB4xf0CZ12hbd1FeECz9zW5IJi9vE2CDDOr+KSXI5 Asus@Talaye
```

Figure 5.5. Fetch SSH public key

5. After setting up the droplet, the platform requires some time to process the request, deploy the virtual machine, set up the requested operating system and enable the access. In the DigitalOcean platform portal the progress is tracked with a progress bar. It usually takes up to 10 minutes to set up the whole droplet. When setup is complete, the ssh connection can be tested by using git bash terminal. To access the droplet with SSH, the ip address of the VPS can be fetched from the DigitalOcean portal. Note that no password would be required to authenticate as the public key is already shared with the VPS previously. *ssh root@remote_server_ip*. Figure Fig. 5.6 shows that the droplet is deployed correctly and it is up and running.

6. For faster access, a meaningful name can be chosen for the ip address to store it inside the ssh config file. This will eliminate the necessity to enter the ip address every time to access to the VPS. So the VPS can be accessible with a command similar to *ssh server*.

7. Now that the SSH connection is up and running, the machine learning engine code can be cloned into the server. To do so, a new directory is created under home directory and the project is cloned.

8. Before starting to run the code, it is necessary to install following programs and libraries.

   - Anaconda: *apt-get install libgl1-mesa-glx libegl1-mesa libxrandr2 libxrandr2 libxss1 libxcursor1 libxcomposite1 libasound2 libxi6 libxtst6*
   - PyTorch: *conda install pytorch torchvision cpuonly -c pytorch* Note that PyTorch can be also used with GPU, however the affordable droplets do not have GPU installed, so the PyTorch is installed in CPU only mode.

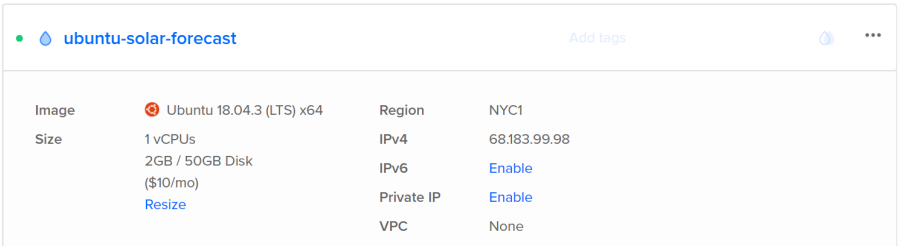


Figure 5.6. Droplet creation status on DigitalOcean portal

- Matplotlib: To generate the graph results, this library is also required and can be installed using python well known package manager pip. *pip install matplotlib*

9. At this point, all the required materials are ready to execute the code. To setup additional tools to ease the remote development following actions have been performed.

10. On local PC, install VS Code. VS Code is a brand-new multi-platform IDE developed by Microsoft. This idea exploits the power of plugins and extensions to add powerful features to the integrated development environment.

11. A very interesting extension developed by Microsoft is called Remote Development extension. Upon activation, it deploys a micro server on the target remote machine and then it is possible to open the code physically stored on remote machine seamlessly. So, all the modifications are applied in real time to the remote code and also the terminal section will be executed on remote server. To install this remote extension, it is enough to open the VS code, navigate to extensions section, select and install the remote development extension. Then it automatically reads the SSH configuration file to find already configured targets. It is also possible to add manually the target. When the SSH connection is established with the target and VS code initialized the remote environment, it would be possible to open the cloned code to eventually modify it. The IDE also integrates the git functionality, so new commits could be made directly from the IDE.

5.3.2 Application Setup

Flask [29] is a micro web framework written in Python and can be used to run a web server and let the server listen to incoming HTTP requests. Flask application object implements a Web Server Gateway Interface (WSGI) application. WSGI is a web server that runs Python code to create a web application.

Application setup code is demonstrated in Fig. 5.7. Where web server is implemented using Flask and listens to incoming post HTTP requests to target Uniform Resource Locator (URL) and sends a success message in case service is alive. Detailed steps are enumerated below:

1. Flask class should be imported to provide the WSGI application.

2. An instance of Flask class should be created.

3. route() decorators specify the Uniform Resource Locators (URLs) that trigger Flask application functions.
4. when function is triggered by receiving post request on the target URL root route, /, status code 200 that indicates the request is successfully received and server is alive is set with a message body showing success.

```
from flask import Flask

HOST = '0.0.0.0'
PORT = 80

app = Flask(__name__)

@app.route('/', methods = ['GET', 'POST'])
def index():
result = {
        'message': 'success'
}
result = jsonify(result)
return result, status.HTTP_200_OK

if __name__ == '__main__':
        app.run(host = HOST, port = PORT, debug = False)
```

Figure 5.7. Web application setup

5.3.3 Application Programming Interface (API)

After application setup, APIs need to be defined and implemented. In this scenario, users need to send request with configuration parameters to the server and receive the prediction results from the server. Moreover, when server is busy running the solar forecast module new training requests should be rejected until solar forecast module execution is completed. Two different APIs are required for this project. One API for receiving requests from the users and starting execution of solar forecast module, another API for delivering the prediction results to the users. Therefore, in this case two different APIs are defined. */execute* API receives configuration parameters and starts solar forecast module execution. */get − results* API sends the prediction results to the user when results are ready. The schematic of APIs are illustrated in Fig. 5.8.

As of Flask version 1.0, Flask runs in threaded mode by default. Therefore, Flask is able to handle multiple requests simultaneously by running each request in a separate thread and serve multiple users at the same time. Prior to version 1.0 or in case threaded mode is disabled, server runs in single-threaded mode. This means that server is only able to handle one request at a time.

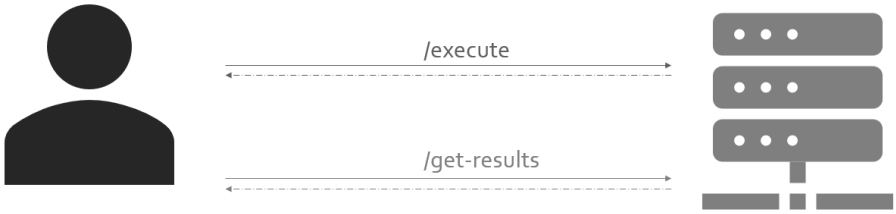


Figure 5.8. APIs illustration

During server implementation two points need to be considered:

- When server is busy with execution of solar forecast module, new train requests are rejected in order to prevent server from overloading.
- Results should be available to users only after solar forecast module execution is properly finished and prediction results are available, otherwise a proper message should be delivered to the users indicating execution is already in progress.

In order to deal with the mentioned points, a global variable is used in API implementations indicating busy status. When server is busy with solar forecast module execution, Busy global variable is set to True. Therefore, new execution and get-results requests are terminated by sending proper messages to users. When solar forecast module execution is completed Busy global variable, value is set to False and new requests can be accepted.

Execute API

When a new request triggers */execute* API and Busy global variable is False, if request is valid in sense of method and variables, Busy global variable is set to True and new thread is created for execution of solar forecast module. Because, solar forecast module execution may take several minutes to complete, another thread is created for handling the execution. Also, user will be notified by receiving a response with status code 200 and a message indicating start of solar forecast module execution.

Get Results API

Upon triggering of $/get-results$ API if request is valid and Busy global variable is False, the csv file containing the prediction results will be sent to the user with status code 200 that shows success. Otherwise, user will be notified by proper messages showing possible errors or if code execution is already in progress.

Chapter 6

Conclusion

This research aimed to develop an accurate solar power forecast module to be used in optimized Energy Management Systems (EMS). EMS systems require very short and short prediction horizons for plant operation, scheduling and storage control. Due to strength of machine learning models in solar forecast prediction tasks for aforementioned horizons, different architectures from recurrent neural networks, convolutional neural networks and also a hybrid architecture were further investigated. The studied architectures are LSTM, TCN and LSTNet respectively. Simulation results indicate that TCN outperforms LSTM and LSTNet in terms of prediction accuracy and correlation. Two different prediction horizons are used during simulations, 5 minutes and 24 hours. By comparing prediction accuracy of these simulations, it is evident that increase of prediction horizon results in decrease of the prediction accuracy. It can be concluded that forecasting becomes more challenging when prediction horizon increases.

By clustering dataset into sunny and cloudy days and using a separate prediction code module for each set, prediction accuracy was improved for all the models. Clustering dataset using TCN architecture has the best prediction results among the models. Moreover, results show that multivariate forecasting by considering meteorological parameters in addition to output power, results in better prediction accuracy with respect to univariate forecasting that only uses output power for prediction. Training and testing the prediction module on different datasets containing data of different locations and climate conditions shows that prediction accuracy is higher for places that have more stable sunny weather conditions and have less cloudy days around the year. This is due to variability of power and solar irradiance within the datasets having unstable weather conditions that makes prediction task more challenging.

A Graphical User Interface (GUI) is implemented to ease the use of prediction module for users who access the source code. GUI receives configuration parameters from the user and starts prediction module execution. Configuration parameters can also be customized by modification of the JSON file containing the configuration parameters directly. In order to deal with GUI freezing issue multi-threading programming is used. Moreover, service is made available by utilizing a remote server and exposing REST APIs to the users. In this case users send HTTP requests containing proper configuration parameters to server and receive a CSV file containing the prediction results. The implemented prediction module can be included in EMS to optimize the system performance and also can be utilized

directly by users for further analysis.

Further studies could improve prediction accuracy by improving architecture of prediction module. Moreover, a front-end website could be developed to create a user-friendly environment to the users.

Appendix A

GUI Implementation Details

```
from tkinter import *
import json
import os
from main import main as solarforecast
import threading



def gui():
    window = Tk()

    window.title("Solar Forecast Module")

    window.geometry('500x500')
    welcome = Label(window, text="Please enter parameters:")
    welcome.grid(column=0, row=0)


    device_value = StringVar()
    device = Label(window, text="Device:")
    rad1 = Radiobutton(window,text='CPU', value='cpu', variable=
        ↪ device_value)
    rad2 = Radiobutton(window,text='GPU', value='gpu', variable=
        ↪ device_value)
    device.grid(column=0, row=1)
    rad1.grid(column=1, row=1)
    rad2.grid(column=2, row=1)


    epochs_label = Label(window, text="Total epochs:")
    epochs_label.grid(column=0, row=2)
```

```
epochs_value = Entry(window,width=10)
epochs_value.grid(column=1, row=2)


horizon_label = Label(window, text="Prediction horizon:")
horizon_label.grid(column=0, row=3)
horizon_value = Entry(window,width=10)
horizon_value.grid(column=1, row=3)
horizon_comment = Label(window, text="Choose 24 for next day
    ↪ predictions.")
horizon_comment.grid(column=2, row=3)


window_label = Label(window, text="Window length:")
window_label.grid(column=0, row=4)
window_value = Entry(window,width=10)
window_value.grid(column=1, row=4)
window_comment = Label(window, text="100 is recommended.")
window_comment.grid(column=2, row=4)


prediction_label = Label(window, text="Select Prediction
    ↪ targets:")
prediction_label.grid(column=0, row=5)
prediction_1_state = BooleanVar()
prediction_1_state.set(True) #set check state
prediction_1 = Checkbutton(window, text='Irradiance', var=
    ↪ prediction_1_state)
prediction_1.grid(column=1, row=5)
prediction_2_state = BooleanVar()
prediction_2_state.set(True)
prediction_2 = Checkbutton(window, text='Power', var=
    ↪ prediction_2_state)
prediction_2.grid(column=2, row=5)


plot_label = Label(window, text="Number of samples to be
    ↪ plotted:")
plot_label.grid(column=0, row=6)
plot_value = Entry(window,width=10)
plot_value.grid(column=1, row=6)

train_label = Label(window, text="Train set ratio:")
train_label.grid(column=0, row=7)
train_value = Entry(window,width=10)
train_value.grid(column=1, row=7)
```

```
    validation_label = Label(window, text="Validation set ratio:"
        ↪ )
    validation_label.grid(column=0, row=8)
    validation_value = Entry(window,width=10)
    validation_value.grid(column=1, row=8)

    def clicked():
        parameters ={
        "device" : device_value.get(),
        "epochs" : epochs_value.get(),
        "horizon" : horizon_value.get(),
        "window" : window_value.get(),
        "Prediction_irradiance" : prediction_1_state.get(),
        "Prediction_power" : prediction_2_state.get(),
        "points" : plot_value.get(),
        "train_ratio" : train_value.get(),
        "validation_ratio" : validation_value.get()
        }

        # Serializing json
        json_object = json.dumps(parameters, indent = 4)

        # Writing to parameters.json
        with open(os.path.join(os.getcwd(), 'data','parameters.
            ↪ json'), "w") as outfile:
            outfile.write(json_object)

        t2.start()
        window.destroy()


    btn = Button(window, text="Start", command=clicked)

    btn.grid(column=0, row=10)

    window.mainloop()

t1 = threading.Thread(target=gui, args=[])
t2 = threading.Thread(target=solarforecast, args=[])
t1.start()
t1.join()
t2.join()
```

Appendix B

Remote Server Execute API Implementation Details

```
@app.route('/execute', methods = ['POST'])
def execute():
    global BUSY
    if (BUSY == False):
        worker = threading.Thread(target=dispatcher, args=[])
        try:
            # check if http method is POST
            if request.method == 'POST':
                device = request.values.get('device')
                epochs = request.values.get('epochs')
                horizon = request.values.get('horizon')
                window = request.values.get('window')
                Prediction_irradiance = request.values.get('
                    ↪ Prediction_irradiance')
                Prediction_power = request.values.get('
                    ↪ Prediction_power')
                points = request.values.get('points')
                train_ratio = request.values.get('train_ratio')
                validation_ratio = request.values.get('
                    ↪ validation_ratio')

                BUSY = True
                worker.start()
                result = {
                'message': 'training is started'
                }
                result = jsonify(result)
                return result, status.HTTP_200_OK
```

```
            else:
                # specify the method must be POST
                # return error code 400
                result = {
                    'message': 'accepts POST parameters'
                }
                result = jsonify(result)
                return result, status.HTTP_400_BAD_REQUEST
        except:
            result = {
                'message': 'Error during execution'
            }
            result = jsonify(result)
            return result, status.HTTP_400_BAD_REQUEST
    else:
        result = {
            'message': 'Execution already in progress...'
        }
        result = jsonify(result)
        return result, status.HTTP_400_BAD_REQUEST


def dispatcher():
    result = solarforecast()
    global BUSY
    BUSY = False
    return 0
```

Appendix C

Remote Server Get Results Implementation Details

```
@app.route('/get-results', methods=['POST'])
def get_results():
    global BUSY
    if (BUSY == False):
        try:
            # check if http method is POST
            if request.method == 'POST':
                path = request.values.get('path')
                if path is None:
                    result = {
                        'message': 'missing or bad parameters {}'
                            ↪ .format(request)
                    }
                    result = jsonify(result)
                    return result, status.HTTP_400_BAD_REQUEST

                try:
                    return send_file(path, as_attachment=True),
                        ↪ status.HTTP_200_OK
                except Exception as e:
                    return str(e)

            else:
                # specify the method must be POST
                # return error code 400
                result = {
                    'message': 'accepts POST parameters'
                }
                result = jsonify(result)
```

```
            return result, status.HTTP_400_BAD_REQUEST
    except:
        result = {
            'message': 'Error during operation'
        }
        result = jsonify(result)
        return result, status.HTTP_400_BAD_REQUEST
else:
    result = {
        'message': 'execution in progress...'
    }
    result = jsonify(result)
    return result, status.HTTP_400_BAD_REQUEST
```

Bibliography

[1] N. Panwar, S. Kaushik, and S. Kothari, "Role of renewable energy sources in environmental protection: A review", *Renewable and Sustainable Energy Reviews*, vol. 15, no. 3, pp. 1513–1524, 2011. DOI: 10.1016/j.rser.2010.11.037.

[2] H. E. Murdock, D. Gibb, T. André, F. Appavou, A. Brown, B. Epp, B. Kondev, A. McCrone, E. Musolino, L. Ranalder, *et al.*, "Renewables 2019 Global Status Report", 2019.

[3] A. Tuohy, J. Zack, S. E. Haupt, J. Sharp, M. Ahlstrom, S. Dise, E. Grimit, C. Mohrlen, M. Lange, M. G. Casado, *et al.*, "Solar forecasting: methods, challenges, and performance", *IEEE Power and Energy Magazine*, vol. 13, no. 6, pp. 50–59, 2015.

[4] 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*, vol. 85, no. 11, pp. 2856–2870, 2011. DOI: 10.1016/j.solener.2011.08.027.

[5] D. E. Olivares, A. Mehrizi-Sani, A. H. Etemadi, C. A. Cañizares, R. Iravani, M. Kazerani, A. H. Hajimiragha, O. Gomis-Bellmunt, M. Saeedifard, R. Palma-Behnke, *et al.*, "Trends in microgrid control", *IEEE Transactions on smart grid*, vol. 5, no. 4, pp. 1905–1919, 2014. DOI: 10.1109/TSG.2013.2295514.

[6] Y. Pu, Q. Li, W. Chen, and H. Liu, "Hierarchical energy management control for islanding DC microgrid with electric-hydrogen hybrid storage system", *International Journal of Hydrogen Energy*, vol. 44, no. 11, pp. 5153–5161, 2019. DOI: 10.1016/j.ijhydene.2018.10.043.

[7] 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*, vol. 1, no. 4, pp. 38–46, 2015. DOI: 10.17775/CSEEJPES.2015.00046.

[8] U. K. Das, K. S. Tey, M. Seyedmahmoudian, S. Mekhilef, M. Y. I. Idris, W. Van Deventer, B. Horan, and A. Stojcevski, "Forecasting of photovoltaic power generation and model optimization: A review", *Renewable and Sustainable Energy Reviews*, vol. 81, pp. 912–928, 2018. DOI: 10.1016/j.rser.2017.08.017.

[9] L. Gigoni, A. Betti, E. Crisostomi, A. Franco, M. Tucci, F. Bizzarri, and D. Mucci, "Day-Ahead Hourly Forecasting of Power Generation From Photovoltaic Plants", *IEEE Transactions on Sustainable Energy*, vol. 9, pp. 831–842, 2018. DOI: 10.1109/TSTE.2017.2762435.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling", *arXiv preprint arXiv:1412.3555*, 2014.

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

[13] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, "Modeling Long-and Short-Term Temporal Patterns with Deep Neural Networks", *arXiv preprint arXiv:1703.07015*, 2017. DOI: 10.1145/3209978.3210006.

[14] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling", *arXiv preprint arXiv:1803.01271*, 2018.

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

[16] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult", *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994. DOI: 10.1109/72.279181.

[17] A. Graves, "Long short-term memory", in *Supervised sequence labelling with recurrent neural networks*, Springer, 2012, pp. 37–45. DOI: 10.1007/978-3-642-24797-2_2.

[18] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM neural networks for language modeling", in *Thirteenth annual conference of the international speech communication association*, 2012.

[19] M. Phi, *Illustrated Guide to LSTM's and GRU's: A step by step explanation*, 2018. [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

[20] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM", 1999. DOI: 10.1049/cp:19991218.

[21] *Solar Power Data for Integration Studies*. [Online]. Available: https://www.nrel.gov/grid/solar-power-data.html.

[22] *Florida Automated Weather Network*. [Online]. Available: https://fawn.ifas.ufl.edu/data/reports/.

[23] *JRC Photovoltaic Geographical Information System (PVGIS) - European Commission*. [Online]. Available: https://re.jrc.ec.europa.eu/pvg_tools/en/tools.html#PVP.

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

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

[26] A. Nespoli, E. Ogliari, S. Leva, A. Massi Pavan, A. Mellit, V. Lughi, and A. Dolara, "Day-ahead photovoltaic forecasting: A comparison of the most effective techniques", *Energies*, vol. 12, no. 9, p. 1621, 2019. DOI: 10.3390/en12091621.

[27] W. F. Holmgren, C. W. Hansen, and M. A. Mikofski, "pvlib python: A python package for modeling solar energy systems", *Journal of Open Source Software*, vol. 3, no. 29, p. 884, 2018. DOI: 10.21105/joss.00884.

[28] P. Ineichen and R. Perez, "A new airmass independent formulation for the Linke turbidity coefficient", *Solar Energy*, vol. 73, no. 3, pp. 151–157, 2002. DOI: 10.1016/S0038-092X(02)00045-2.

[29] *Flask*. [Online]. Available: https://flask.palletsprojects.com/en/1.1.x/api/#flask.Flask.