A comparison of deep learning methods for time series forecasting with limited data

Master's thesis
for the degree
M.Sc Mathematics
submitted by
Abinaya Jayaprakash (5391958)

Department of Mathematics and Computer Science
Freie Universität Berlin
WS 2022/23

Abstract

This thesis was written in cooperation with Elia. The goal was to develop a forecaster that would predict the future output of photovolatic (PV) systems that would be then used to optimize energy usage in households. Better knowledge about renewable energy production would result in efficient usage of this energy and reduced energy costs. The historic data used covered 6 months; January to June 2022. Traditional time series forecasting techniques were compared with developing machine learning approaches on their ability to predict future values using the limited input data. The established time series forecasting technique used as a baseline model was simple linear regression. The machine learning techniques consisted of two main types of feed forward neural networks; LSTMs and CNNs. The performance of all methods were measured and compared via a chosen set of evaluation metrics. The best performance of all implemented methods resulted in a CNN model with no hidden layers.

Acknowledgements

I would first like to thank my thesis supervisors; Professor Volker John and Rachel Berryman for their time, support and vital feedback. My thanks also goes to Mason Samuel for his help and guidance. I would also like to thank my friends and family for motivating me throughout and mainly Subodh Singh Khangar for guiding me when I first decided to explore the field of machine learning. Lastly I am also very grateful to Elia for the opportunity, the access to data and coding facilities.

Table of Contents

1. Introduction

1.1 Motivation

Increasing developments in the area of machine learning have evidently outlined the seamless attribution that machines have in producing greater values/outcomes for businesses. Deep learning in specific is one of the most recognized sub areas of machine learning, inspired by the human brain's neural networks, and has set exceptional records of accuracy in the recent years irrespective of the business sector.

Consequently deep learning has strongly established a place for itself in the energy sector too, and one of its main applications has been to maintain the balance between energy supply and demand. According to Forbes [1], "As the world shifts in the direction of personalized digitized services, the energy sector is lagging behind" and this is mainly because this sector heavily depends on predictive diagnostics and the cost of error in it is known to be particularly high.

In attempts to resolve the wastage of excess energy in households, the process of working with renewable energy has proven to be quite challenging given that it requires precise prognosis of its generation and necessity. With respect to solar energy, the uncertainty associated with factors that affect its production such as weather conditions which are constantly subject to change makes it tough. Another hardship faced is gaining access to similar household data from the past with similar geographical settings since installing PV panels in a household is a big-budget and high involvement decision with many barriers that require substantial maintenance, [2].

Achieving an equilibrium in households amidst these drawbacks would lead to energy saving which will then result in reduced utility bills and personalized energy usage/maintenance schemes. Energy providers would be able to dispatch their

resources better, anticipate demand in advance, provide customized services and minimize costs whenever possible, [3].

Having understood the problems faced and the benefits, this thesis revolves around finding the answer to **Can deep learning (or combining best known methods) yield accurate predictions regarding the PV output needed to maintain the balance between the production and usage of solar energy in households with limited input?**

1.2 Related Work

Time series forecasting can be classified as univariate if it involves predicting a single variable varying over time or multivariate if it involves predicting multiple variables varying over time. Various models have been tested, implemented and updated for better prediction in both cases but this has been evidently challenging due to the nature of the data samples and their components such as trend, seasonality and correlation between features. Recent techniques in handling such data in the energy sector have evolved from the increase in use of deep learning approaches, [3]. A myriad of works have already reviewed the existing machine learning and deep learning methods, compared them and identified the most efficient, [4] [5].To summarize, unlike classical machine learning models that were capable of modelling linear relationships, deep learning models take into account the sequential nature of time series data, thus allowing them to map complex non-linear relationships too. Additionally, these models are known to be less sensitive to missing data, easier to incorporate with multivariate time series and include automated feature selection, [6]. Among all reviewed implementations, artificial neural networks (ANNs) have proven to be among the best under many circumstances.

Ref. [4] evaluates the performance in terms of accuracy and efficiency of seven popular ANNs: multilayer perceptron (MLP), Elman recurrent neural network ERNN, long-short term memory (LSTM), gated recurrent unit (GRU), echo state network (ESN), convolutional neural network (CNN) and temporal convolutional network (TCN) and concludes that LSTMs and CNNs are the best choices, with LSTMs yielding the most accurate forecasts and CNNs being more efficient and consistent under different parameter configurations.

1.3 Outline

This thesis is further divided into chapters that elaborate on the different theories and methods used in the thesis.

Chapter 2 explains the concept behind neural networks, how they are trained, their drawbacks and possible ways to counteract these. It also introduces the two main types of neural networks the thesis focuses on and the software needed to develop and train them. Chapter 3 describes the task and the data sets used in detail. It also gives an overview of the different techniques used to process the raw data and then further divide it for training and testing purposes. Chapters 4 and 5 present and evaluate the results obtained via the various approaches chosen. At the end, chapter 6 lays out the outcome of the thesis and how this can be utilized or improved for future use.

2. Theoretical Background

2.1 Machine Learning

The rapid increase in innovation and technological advancement across the globe has resulted in artificial intelligence (AI) and machine learning playing a significant role in transforming businesses, task automation and economic growth, [7]. Machine learning, a subset of AI, entails teaching an algorithm/model to recognize patterns and extract useful insights from data with minimal human intervention in order to produce the desired outcome. These findings are then applied to new and changing data which results in a repeating feedback loop allowing continuous model enhancement to gain further insights with high accuracy. Based on how this technique can be implemented it can be categorized as follows:

- Supervised learning,
- Unsupervised Learning.

Supervised learning refers to the process in which a model is being taught to learn what its desired output should be using previous observations/similar examples. The chosen algorithm is provided with a training data set that includes inputs and the resulting outputs, and thereby it tries to model relationships/dependencies between the target output and the input features. These findings are then used to predict the output based on new input data (test data set), [8]. Unsupervised learning works with data that only involves inputs and the algorithm inspects this data, identifies hidden correlations and sometimes groups it based on a few characteristics without any prior instruction or expected output, [9].

This thesis pays special attention to time series forecasting; one type of supervised learning and I have chosen neural networks to be the learning algorithm.

2.2 Mathematical Background

2.2.1 Artificial Neural Networks (ANNs)

To get an insight of what a neural network is, it is vital to begin with understanding how the scaled-down, miniature building blocks of these neural networks; an artificial neuron, functions.

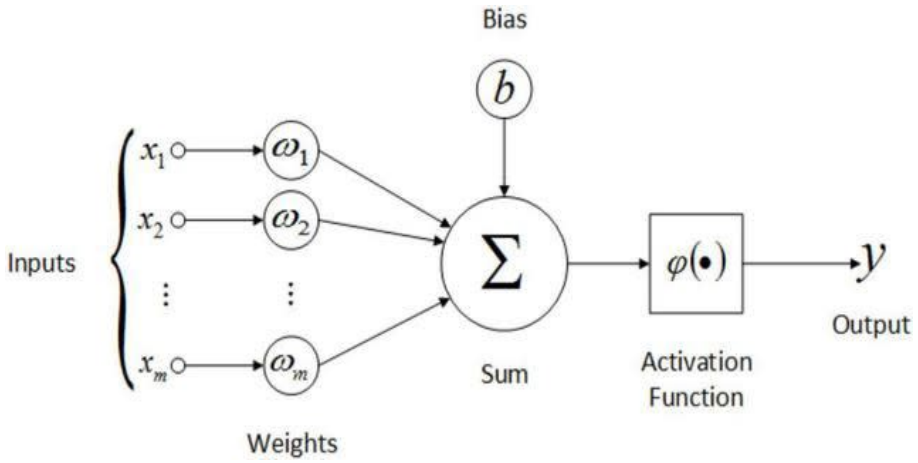

Figure 2.1: Artificial neuron, from [10]

As seen in Figure 2.1, a neuron comprises of four main components. The first being the inputs, which could be information about the inputs in the training data set such as features or the output from a preceding neuron. Second, each input (x_i) is multiplied by a weight (w_i) that controls its level of significance in determining the output (y). The larger the magnitude of the weight multiplied by the input, the more influence it has on the output value. Third, all these weighted inputs ($w_i x_i$) are added to yield a weighted sum. This could also include a bias component (b) in some instances, which is used to adjust the value of the output obtained, thereby ensuring the model is a best fit for the input data. The final component is the activation function (ϕ) which the weighted sum is passed through to output a single number

$$y = \phi\left(\sum_{i=1}^{m} w_i x_i + b\right).$$

The purpose of these activation functions can differ based on their mathematical properties such as linearity, continuity, range and order of differentiation, [11].

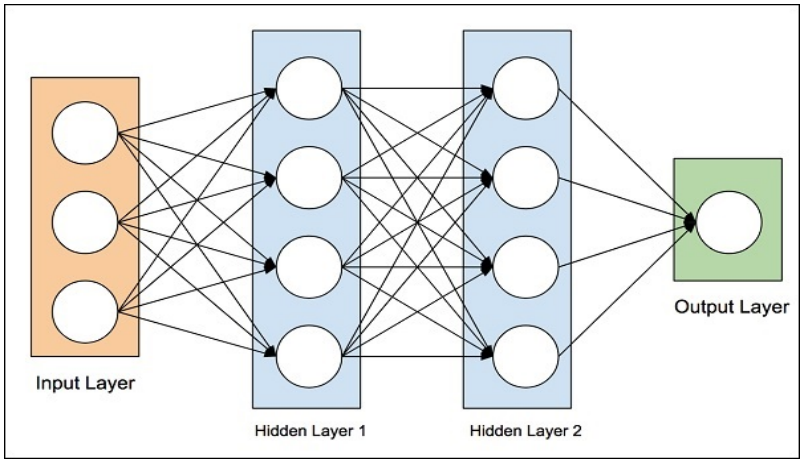

Figure 2.2: Simple neural network

An ANN is a network formed by stacking one or more neurons together in two or more layers. Figure 2.2 is an example of a 3-layer neural network (multi-layer perceptron) with two hidden layers with 4 neurons each, that the input is fed into and then continues to flow in the same direction towards the output layer (forward propagation) that would then generate the expected result. Such ANN frameworks, where information is passed through from the input to the output layer in the forward direction, are called feed-forward neural networks (FNNs).

ANNs can be trained to learn from the input data via a powerful mechanism that will be discussed in the next section.

2.2.2 Training FNNs

In order to train an ANN to study the input data, a technique called back propagation is used.

As the last step of forward propagation, the network error is calculated by comparing the actual output and the value generated by the network. Back propagation is the method of traversing the network in reverse order to update the values of the weights and biases used such that the network error is reduced. The objective function/cost function chosen to calculate the error can vary depending on the network's use case. A learning rate is assigned to be in control of the speed at which back propagation is performed, [12].

The extent to which the values of the weights and biases are altered depends on how responsive the cost function is to a change in these values. Irrespective of the type of the cost function, all cost functions can be represented as a function of all

network outputs and also as an average of all cost functions for individual training samples when there are many. Although the cost function also entails the actual output, it is a fixed parameter and will not be affected by the changes made to the weights/biases, [13].

An epoch refers to one cycle of forward propagation followed by back propagation, [14] . The updated internal parameters are then used for the next epoch. The number of epochs is traditionally large permitting the learning algorithm to run until the network error has reached an optimal minimum (satisfies certain criteria or when the number of iterations exceed the allocated computational budget, [15]).

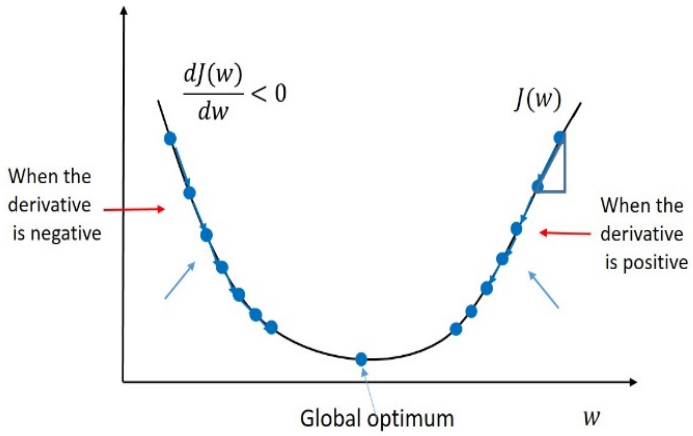

Figure 2.3: Minimizing the cost/error function, from [16]

Figure 2.3 gives us a visual overview of an example of how the one-dimensional cost function, $J(w)$ is minimized iteratively with respect to one of the weights w depending on the magnitude of its gradient with respect to w over multiple epochs. Given that the value of the gradient is positive, the update equation works as follows;

$$w_1 = w_0 - \alpha * \frac{dJ(w)}{dw},$$

where α denotes the chosen learning rate. Suppose the value of the gradient is positive, then

$$w_1 = w_0 + \alpha * \frac{dJ(w)}{dw}.$$

This method referred to as gradient descent is fairly expensive since it requires the computation of the gradient during every single iteration, especially when there are

many training samples. A cheaper replacement works by calculating the gradient with respect to a randomly chosen training point during each epoch instead of all individual training points, and this is referred to as stochastic gradient descent, [15] which will be the optimizer used during the training of the neural networks.

2.2.3 Overfitting

One common problem encountered when training FNNs is that the neural network overfits (learns the training data set too well) and is unable to perform with unseen data, [17]. This issue is particularly frequent for small training data sets like the one used in this context. Several tecnhiques are used to encounter this.

One possibility is to add a regularization/penalty term to the error function (L2 regularization) of the form

$$\frac{\lambda}{2} w^T w,$$

where λ determines its impact. This additional term which is the sum of all weights squared helps in minimizing the values of weights (weight decay) during backpropagation and adds stability by making the model less sensitive to the training data.

Another method we could use is dropout. This method ignores a set of neurons in the neural network at random with set probability. This reduces interdependent learning across the network, makes it simpler and results in better spread out weights, [18].

2.2.4 Vanishing/Exploding gradients

Vanishing/exploding gradients are quite common in deep neural networks. The updated gradient values as a result of backpropagation could keep reducing in value that results in insignificant updates to the parameters (vanishing) or could keep increasing in value resulting in sizeable updates to the parameters (exploding). This could then either stall the learning process or make it unstable, [19]

Choosing an appropriate activation function and proper initialization of weights during training can help overcome this along with normalizing the input data. Based on Ref. [20], I decided to use the exponential linear unit (ELU) activation function along with the He initialization. Normalization is further explained in section 3.4.2 below.

The ELU activation function is a variant of the Rectified linear activation unit (ReLU) activation function and is one of the most popular activation functions

used. It outputs values as follows:

$$ELU(x) = x \text{ if } x \geq 0$$

$$ELU(x) = \alpha(e^x - 1) \text{ if } x < 0$$

Negative entries are handled better by the ELU activation function with the help α that controls the saturation of negative net inputs and therefore avoids the dying ReLU problem and also since its mean output values are closer to zero, it also tends to converge faster and has many other advantages over the ReLU function overall, [20, 21]. The value of α was kept constant at the default value 1.0 for all the models tested.

A well known approach for the initialization of weights along with the ELU activation function is He initialization, known after the last name of its author. The weights would be random number that follow a gaussian probability distribution with a mean of 0 and a standard deviation of $\sqrt{\frac{2}{n}}$, where n denotes the number of inputs fed into the node, [22].

2.3 Types of ANNs

Figure 2.2 is an example of one of the simplest types of ANNs and is called the multi-layer perceptron (MLP) or simply a deep FNN. This contains a series of fully connected layers; all neurons in one layer are connected to all neurons in the next layer. It has also been proven useful for various tasks but is subject to a few limitations, [23].

One of these is the inability to process sequential data since each data element is treated separately, whereas when processing sequential information, elements can be interrelated. To overcome these obstacles and make neural networks more applicable to scenarios that deal with sequential data such as time series forecasting problems, recurrent neural networks (RNNs) and convolutional neural networks (CNNs) were introduced.

2.3.1 RNNs

RNNs are specialized in processing a sequence of values of varying length.

As illustrated in Figure 2.4, RNNs are structured to learn dependencies in sequential input data, by storing previous values in its memory/hidden state for a short period

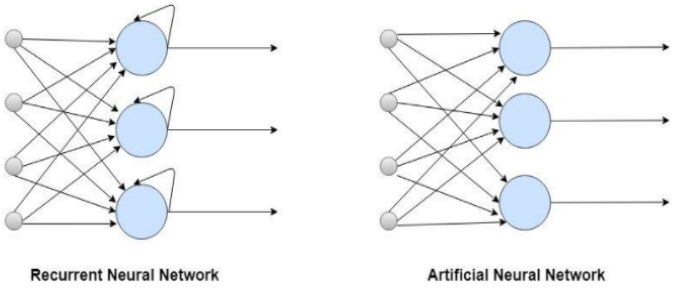

Figure 2.4: Main difference between MLP and RNN, from [24]

of time. The loop in the figure depicts how the output of the current layer would depend on the current and previous inputs, which enables the network handle sequential data well, [25].

2.3.2 CNNs

CNNs were originally designed to solve problems with image data but have also produced impressive results when introduced to sequential data and this is mainly due to the one dimensional convolutional layers it can contain.

Figures 2.5 and 2.6 give us an overview as to how a convolutional layer works as opposed to a fully connected layer.

Figure 2.5: Fully connected layer, from [26]

The pictorial representation of a convolution in Figure 2.6 illustrates the significant details that make it more efficient. Note that we have ignored the biases here to make it simpler. The sliding dot product referring to the vector product of input matrix values and the sliding kernel, ensures that not all inputs affect the

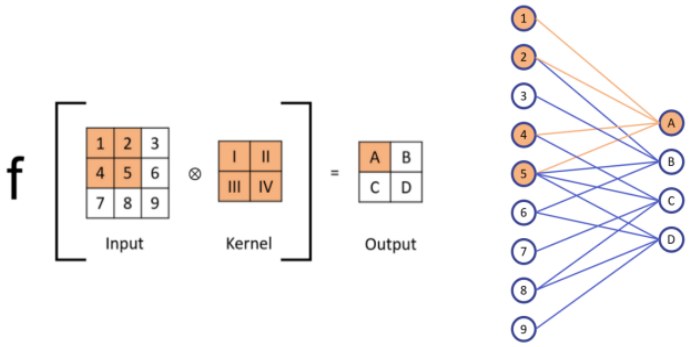

Figure 2.6: Convolutional layer, from [26]

output generated which makes CNNs capable of handling high dimensional data and picking up important features in the data.

The weight matrix in Figure 2.5 would have varying values in all columns of the matrix, whereas the values of weights in the kernel/filter in Figure 2.6 do not change as it slides horizontally and vertically through the input data. This minimizes the number of parameters per layer and teaches the network to flexibly learn and detect important features irrespective of where/how they appear in the input data, which allows for better generalization when a CNN is exposed to new input data, [26].

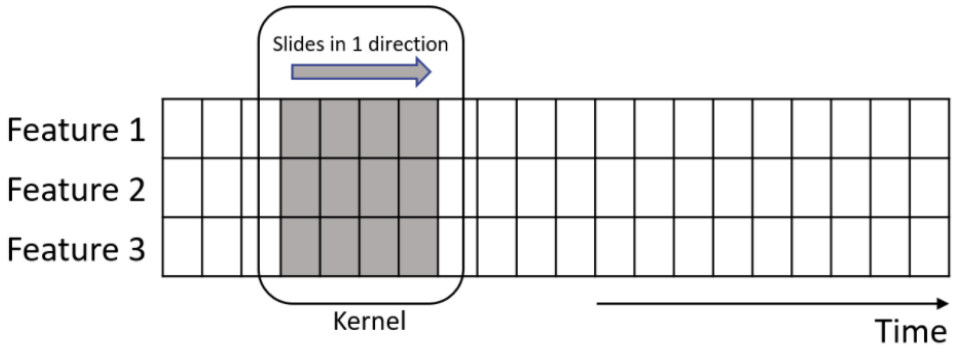

Figure 2.7: 1D convolutional layer, from [27]

As mentioned above, Figure 2.7 shows how the kernel slides only horizontally in a one dimensional convolutional layer used when dealing with time series data. Ref. [28] contains further information on how the time series data was restructured in order to be fed into the 1D convolutional layers.

Similarly, the pooling layers in a CNN also enhance its capability to identify vital features and control the computational power needed to run it via dimension

reduction.

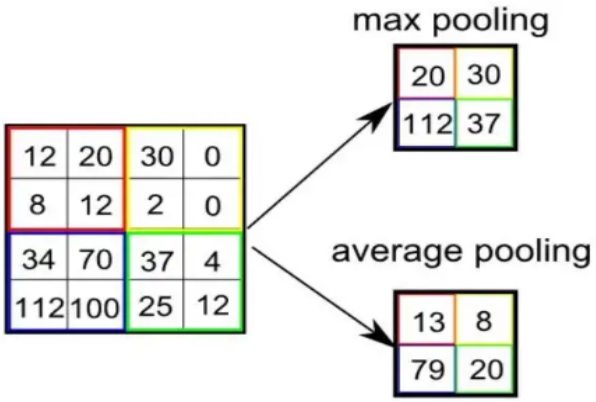

Figure 2.8: Types of pooling and their respective outputs from [29]

As illustrated in Figure 2.8, there are two main types of pooling and their names describe how their functionalities and outputs are different. In Figure 2.8 a window size of 2 and a stride (number of columns/rows to skip when moving horizontally/vertically) value of 2 have been used and these hyperparameters also play a vital role in determining how efficient the pooling layers are. Along with these the type of padding used also matters, [29].

2.4 Multivariate forecasting using LSTMs

Long short term memory (LSTM) is an improved version of a RNN that was developed as a solution to handle a few drawbacks with respect to using RNNs. The special neurons/memory cells in an LSTM help overcome the problem of vanishing/exploding gradients, are capable of storing previous values for a longer period of time and have finer control over which parts of the information is carried forward and how much from the past needs to be forgotten which overall make the LSTM an ideal choice for time series forecasting.

Figures 2.9 and 2.10 provide an insight of how information is passed through each memory cell in an LSTM from left to right. Figure 2.9 illustrates the three gates the memory cell is composed of and their respective functions, while Figure 2.10 shows us what takes place in each gate in order for it to perform its function.

Please refer to Figure 2.10 that contains the terms used in the explanation below. The sigmoid and tanh activation functions are denoted by σ and $tanh$ respectively. Given the new inputs, x_t corresponding to the current timestamp, the LSTM would function as follows, [32];

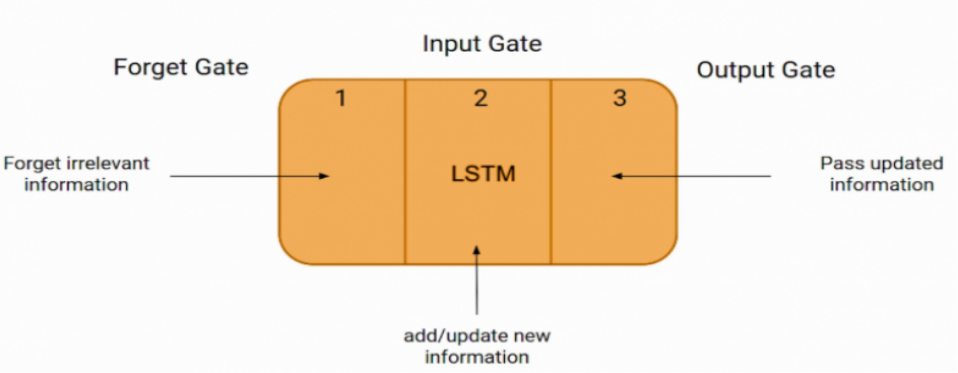

Figure 2.9: Memory cell in an LSTM, from [30]

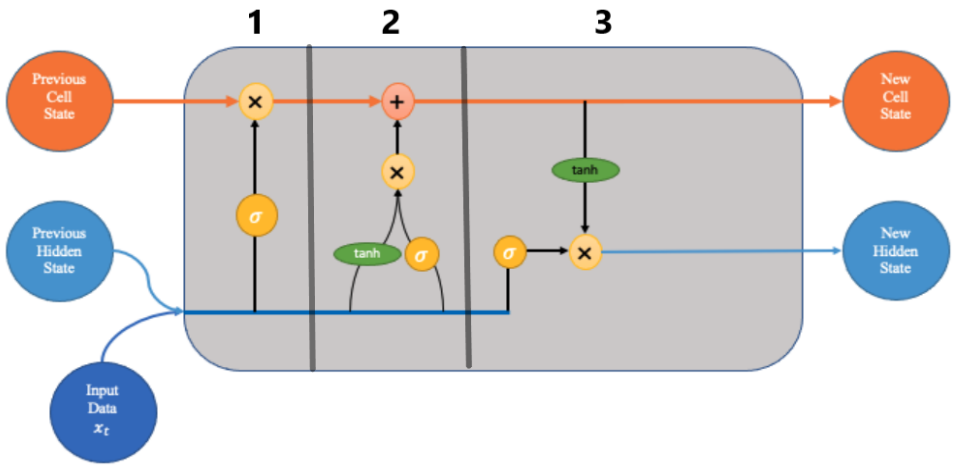

Figure 2.10: Detailed overview of a memory cell, extracted from [31] and edited

- **Forget Gate(1)** - Outputs a value f_t between 0 and 1 which decides what percentage of past data has to be forgotten with respect to the output from the previous timestamp, h_{t-1} (previous hidden state that signifies short term memory) and the current inputs, x_t

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

- **Input Gate(2)** - Updates the new cell state (signifies long term memory) for the current timestamp C_t which would be a combination of a percentage (i_t) of processed input data $\bar{C}_t$ along with the information from the previous

cell state C_{t-1} that needs to be remembered

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

$$\bar{C}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c),$$

$$C_t = f_t * C_{t-1} + i_t * \bar{C}_t.$$

- **Output Gate(3)** - Decides what percentage (o_t) of the new cell state C_t need to be stored short term and carried forward to the next timestamp, also referred to as the new hidden state, h_t

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

$$h_t = o_t * tanh(C_t).$$

The cell state enables information flow through the entire chain of cells, with a few linear interactions that determine which parts of it are stored long term. Constantly updated cell state values help control the gradient values, and thereby avoid vanishing/exploding gradients, [33].

2.5 Hyperparameters

- **Number of hidden layers and neurons** - The number of hidden layers and neurons in each of these layers depends on the use case, input data and largely impact the complexity of the model, [34]. Considering the limited amount of data available, I chose to test with 1 and 2 hidden layers along with the number of neurons within the range of 10 to 50.
- **Dropout probability** - I chose to test with dropout probabilities between 20 to 40% on my hidden layers, [35].
- **Learning rate** - I tested with learning rates ranging from 10^{-3} to 10^{-6} for the chosen optimizer, [36] .
- **Batch size** - I chose to test batch sizes ranging from 32 to 128, [37].
- **Number of epochs** - The number of epochs used for testing varied from 50 to 300.

- **Number and size of convolutional filters** - The number and size of filters/kernels used in the 1D convolutional layers of the CNN massively contribute towards its ability of feature detection [38], and so I decided to test with 16 and/or 32 filters of equal height and width of sizes 1x1 and 2x2, [39]. I chose to maintain the default stride (the number of steps skipped when the kernel slides across the input data) value of 1 in all the convolutional layers.
- **Type of pooling, window size and type of padding** - I chose to test with max pooling with a window size of 2, the default stride value of 1 and valid padding, [29, 40].

2.6 Software used

Given below is an overview of the programming language and software used.

- **Python** - An easy to use programming language [41], that is interoperable with a wide range of other libraries and packages such as Numpy [42], and Pandas [43] that were ideal choices for all numeric computations performed.
- **Scikit-Learn** - An open source machine learning library in python [44], with simple tools for all stages of the training and testing pipeline.
- **TensorFlow** - Its high level and open source programming interface Keras [45], made building and training neural networks a lot quicker.
- **Plotly** - An open source graphing library for python [46], that allows creating various interactive types of graphs and charts to plot, analyze and compare results.

3. Task and Data Analysis

3.1 Task Summary

In order to efficiently optimize overall energy usage in a household, an accurate forecast of how much energy is produced by all energy sources is an important requirement.

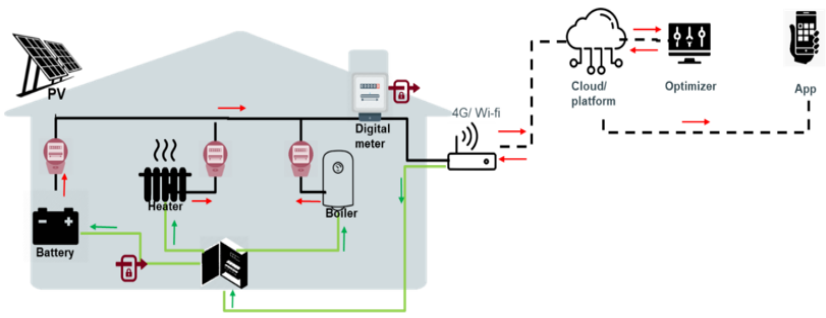

Figure 3.1: Overview of a household

As seen in Figure 3.1, the optimizer designed by Elia, a belgian electricity system operator as part of a research project, focuses on optimizing the energy usage/ minimizing the overall energy costs for all devices agreed upon and controlled by a battery in each household. Some households additionally have solar PV systems installed in them too.
One of the main purposes of the optimizer in this scenario would be to make efficient use of the energy converted to electricity by the PV. In order to carry this out, an accurate forecast of how much electricity is generated in this form throughout different times of the day is absolutely crucial and this thesis contributes to this project by reviewing and analyzing a few approaches that can be used to

forecast this data for a specified period in the future using deep neural networks. Precise predictions would lead to less wastage of the excess solar energy converted to electricity and reduced energy imports from the grid, thereby reducing overall household energy costs.
The following sections would include a more detailed focus on how this was done for one of the households with a PV in the Limburg region in Belgium. A simple linear regression model would be used as the baseline model and act as a reference to compare and evaluate the performance of the chosen methods.

3.2 Data

The models were tested on data over a span of 6 months; January to June 2022.

3.2.1 Household PV data

The data from the household, as seen in Figure 3.2 with 15 minute resolution contained the amount of energy converted to electricity in watts per hour. This would be the target variable which the models would have to predict future values for the next 24 hours.

	negative_active_energy_flow_wh
2022-01-01 12:30:00	37.62
2022-01-01 12:45:00	39.61
2022-01-01 13:00:00	39.61
2022-01-01 13:15:00	41.37
2022-01-01 13:30:00	27.88
...	...

Figure 3.2: Snippet of the pre-processed household PV data

3.2.2 Weather forecast data

Weather is one of the main factors that can influence the output of a PV, [47]. Since the model needs to be trained to predict the electricity generated by the PV in the future for which we wouldn't know how the weather would turn out to be, training the model to predict based on weather forecast data is essential.

The weather forecast dataset used, [48] as seen in Figure 3.3 was extracted from Rebase Energy's database and used as input when training the models. This included values for various weather based variables with hourly resolution for the Limburg region and was also updated every six hours on a daily basis.

valid_datetime	HCC	LCC	MCC	Pressure	RH	SDR	Temp	TCC	TP	WD	WS
2022-01-01 00:00:00	0.44	0.73	0.0	102462.63	80.48	0.00	12.63	80.77	0.0	224.79	5.43
2022-01-01 00:15:00	0.44	0.73	0.0	102462.63	80.48	0.00	12.63	80.77	0.0	224.79	5.43
2022-01-01 00:30:00	0.44	0.73	0.0	102462.63	80.48	0.00	12.63	80.77	0.0	224.79	5.43
2022-01-01 00:45:00	0.44	0.73	0.0	102462.63	80.48	0.00	12.63	80.77	0.0	224.79	5.43
2022-01-01 01:00:00	0.51	0.85	0.0	102468.47	79.81	0.00	12.70	87.48	0.0	222.06	5.65
...	...	...	...	...	...	...	...	...	...	...	...

Figure 3.3: Snippet of the pre-processed weather forecast data

The variables that were taken into consideration are as seen in Figure 3.3. The names of the columns are abbreviations of these variables listed below;

- HCC : High Cloud Cover
- LCC : Low Cloud Cover
- MCC : Medium Cloud Cover
- Pressure : PressureReducedMSL
- RH : Relative Humidity
- SDR : Solar Downward Radiation
- Temp : Temperature
- TCC : Total Cloud Cover
- TP : Total Precipitation
- WD : Wind Direction
- WS : Wind Speed

Ref. [49] contains a detailed insight of what these variables mean and how this data is collected.

3.2.3 Regional solar energy forecast data

This additional data accessed from the Elia open data portal, [50] as seen in Figure 3.4 was also used as an input feature during the training of models. This data contained the values for the total amount of solar energy generated by all facilities in the Limburg region in which the household is situated in, [51].

StartDate	MostRecentForecast
2022-01-01 11:00:00	65.88
2022-01-01 11:15:00	69.95
2022-01-01 11:30:00	73.76
2022-01-01 11:45:00	77.32
2022-01-01 12:00:00	105.66
...	...

Figure 3.4: Snippet of the pre-processed regional forecast data

3.3 Preprocessing

Raw data, or in other terms unformatted real-world data can undoubtedly contain inconsistent values, errors, outliers and sometimes missing entries. Preprocessing is essential in order to address these problems and make the data more consistent, [52].The code used for data analysis and preprocessing can be found at `https://github.com/Abinaya-J/ThesisFiles/blob/main/HHDataAnalysis.ipynb`.

The regional solar forecast data did not have any inconsistent data entries, but however the household PV and the weather forecast datasets did need some preprocessing.

3.3.1 Weather forecast data

The weather forecast data was upsampled as shown in Figure 3.3 from a hourly resolution to a quarter-hourly resolution since the predictions were required to be at 15 minute intervals.

In order to avoid the problem of multicollinearity which can have a negative impact on the results obtained it is also important to check if any of the weather variables are highly correlated with one another, [53, 54].

Figure 3.5 clearly shows that the variable TCC is highly correlated with more than one other variable, and also since the variables HCC,MCC and LCC provide information about cloud cover I chose to remove it completely. Although Temp and SDR were also highly correlated with each other, I chose to keep them since both of them can have a significant impact on the output of the PV when compared with other features.

	HCC	LCC	MCC	Pressure	RH	SDR	Temp	TCC	TP	WD	WS
HCC	1.000000	-0.038695	0.414821	-0.252448	-0.025847	-0.018108	0.176157	0.594200	0.033770	-0.068532	0.040583
LCC	-0.038695	1.000000	0.222517	-0.171030	0.269273	-0.287265	-0.177029	0.590641	0.236719	0.375534	0.434295
MCC	0.414821	0.222517	1.000000	-0.401527	0.064523	-0.139612	0.057109	0.544410	0.276481	0.012737	0.233161
Pressure	-0.252448	-0.171030	-0.401527	1.000000	0.144389	-0.014126	-0.291968	-0.310374	-0.280202	-0.092621	-0.481893
RH	-0.025847	0.269273	0.064523	0.144389	1.000000	-0.658039	-0.609807	0.166605	0.106399	0.099287	-0.148417
SDR	-0.018108	-0.287265	-0.139612	-0.014126	-0.658039	1.000000	0.569591	-0.206100	-0.059723	-0.044796	0.002770
Temp	0.176157	-0.177029	0.057109	-0.291968	-0.609807	0.569591	1.000000	0.028699	0.016000	0.021823	0.023327
TCC	0.594200	0.590641	0.544410	-0.310374	0.166605	-0.206100	0.028699	1.000000	0.168836	0.211461	0.291732
TP	0.033770	0.236719	0.276481	-0.280202	0.106399	-0.059723	0.016000	0.168836	1.000000	0.142552	0.320415
WD	-0.068532	0.375534	0.012737	-0.092621	0.099287	-0.044796	0.021823	0.211461	0.142552	1.000000	0.225882
WS	0.040583	0.434295	0.233161	-0.481893	-0.148417	0.002770	0.023327	0.291732	0.320415	0.225882	1.000000

Figure 3.5: Heatmap indicating correlation between weather variables

	negative_active_energy_flow_wh
2022-04-02 18:00:00	NaN
2022-04-02 18:15:00	NaN
2022-04-02 18:30:00	NaN
2022-04-02 18:45:00	NaN
2022-04-02 19:00:00	NaN

Figure 3.6: Snippet of a few missing values present in the household data

3.3.2 Household PV data

As seen in Figure 3.6 the household data contained about 17% of 'NaN' entries or missing values and as shown by Figure 3.7 the data contained about 1.7% of negative entries as a result of errors that occurred during data collection. In order to decide on how to deal with these entries, it is important to know when and how frequently they appear in the data.

One common observation that can be made by looking closely at Figures 3.8 and 3.9 is that most of these entries occur during the early mornings and evenings; times when there is no sunlight and there is no PV output. This is also clearly seen in Figure 3.10 where the gaps in the data (indicated using red arrows) correspond to these times of the day, but there are still a few that are present during the other times of the day, too.

In order to fill in the missing values and the negative values, I decided to use Ref. [55] in order to obtain the sunrise and sunset times for each day in the data set. I then used this information to replace all missing and negative entries with 0 if they occurred at times before sunrise or after sunset, and otherwise replaced

	negative_active_energy_flow_wh
2022-01-01 02:15:00	-2.842170e-11
2022-01-01 03:00:00	-2.842170e-11
2022-01-02 20:45:00	-2.842170e-11
2022-01-05 17:15:00	-2.842170e-11
2022-01-05 17:45:00	-2.842170e-11

Figure 3.7: Snippet of a few negative values present in the household data

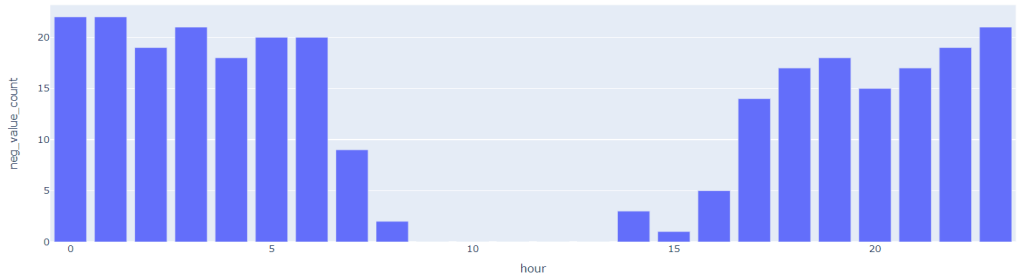

Figure 3.8: Number of negative entries grouped by the hour

them with the mean of the observed values for the same hour over the month of occurrence, [56].

There were also a few occurrences of inconsistent values during the late hours on some days as shown by Figure 3.11 due to errors that didn't correspond to the general pattern observed. Again Ref. [55] was used to cross check the sunrise and sunset times on these days and these inconsistent values were then replaced with zeroes if they occurred after sunset or before sunrise.

Another important thing to check in time series data is if trend and seasonality is present in the data. [57]. Doing this is necessary in order to see what effect these have on the target variable if present, and removing these components would reveal other interesting facts about the data. Since it was quite hard to decide on this just by observing Figure 3.10, I used the Augmented Dickey Fuller (ADF) test for further verification, [58].

The p-value (probability of not rejecting the null hypothesis which states that the data is non-stationary or in other words contains the trend and seasonality components) as observed in Figure 3.12 is very small and the test statistic is less than all critical values at all confidence levels, so the data used is stationary and no further preprocessing steps are needed.

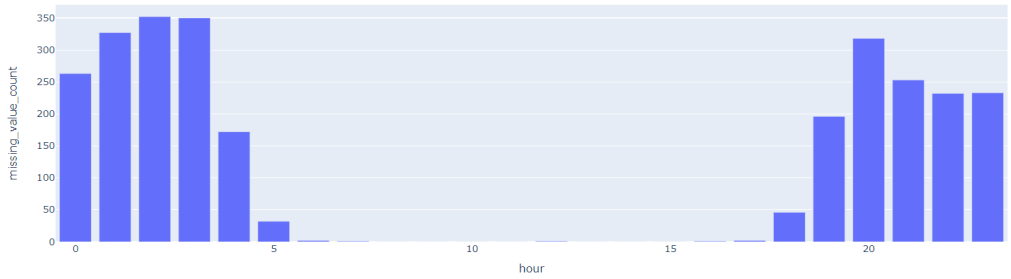

Figure 3.9: Number of missing entries grouped by the hour

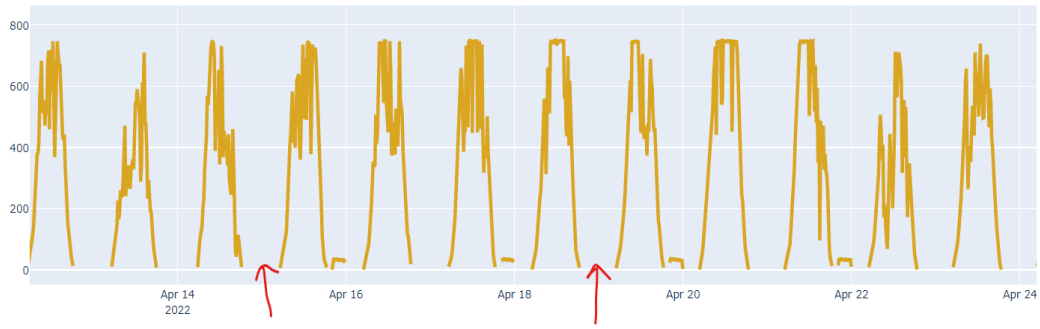

Figure 3.10: Snippet of the household data line graph without negative entries

3.4 Machine learning techniques

3.4.1 Feature engineering

A few additional features that contained information extracted from the raw data were added considering the importance of features in predictive modelling. Better features would enable the model understand the underlying problem quicker and output finer results, [59].

- **Date Time Variables** - Components corresponding to each timestamp such as the hour of the day or month of the year.
- **Lagged Variables** - Values recorded at one or more timestamps before the current timestamp.
- **Rolling window statistics** - The mean over a period of time (window) before the current timestamp, for example the mean of values recorded during the same time over the last week.

2022-07-08 21:00:00+00:00	
2022-07-08 21:15:00+00:00	
2022-07-08 21:30:00+00:00	21.72851560031995
2022-07-08 21:45:00+00:00	37.59765619970858
2022-07-08 22:00:00+00:00	31.933593819849193
2022-07-08 22:15:00+00:00	34.416623956058174
2022-07-08 22:30:00+00:00	36.67712602438405
2022-07-08 22:45:00+00:00	34.42382809985429
2022-07-08 23:00:00+00:00	34.42382809985429
2022-07-08 23:15:00+00:00	34.42382819997147
2022-07-08 23:30:00+00:00	34.42382810031995
2022-07-08 23:45:00+00:00	31.25
2022-07-09 00:00:00+00:00	34.1796875
2022-07-09 00:15:00+00:00	31.25
2022-07-09 00:30:00+00:00	34.42382809985429
2022-07-09 00:45:00+00:00	31.25
2022-07-09 01:00:00+00:00	31.25
2022-07-09 01:15:00+00:00	18.79882809985429
2022-07-09 01:30:00+00:00	
2022-07-09 01:45:00+00:00	

Figure 3.11: Inconsistent values due to errors

```
ADF Statistic: -22.485913
p-value: 0.000000
Critical Values:
        1%: -3.431
        5%: -2.862
        10%: -2.567
```

Figure 3.12: ADF test results

- **Binary indicator** - The information obtained about the sunrise and sunset times for each day in the data set from Ref. [55] was used to add in a binary indicator variable where 1 represented all timestamps in between the sunrise and sunset times and 0 otherwise.

When predicting values for 24 hours in the future, it is not possible to have the actual observed value for timestamps over the last 24 hours with respect to the target timestamp, therefore based on Figures 3.13 and 3.14, the lagged variables and the rolling window statistics were chosen from a period of 4 days before (first few spikes with high auto correlation values) with respect to each timestamp, [60].

Figure 3.13 clearly shows us that the highly correlated values with the target timestamp from any chosen day are the values that correspond to the target timestamp (96^{th} lag), one timestamp before (95^{th} lag) and after this (97^{th} lag) on

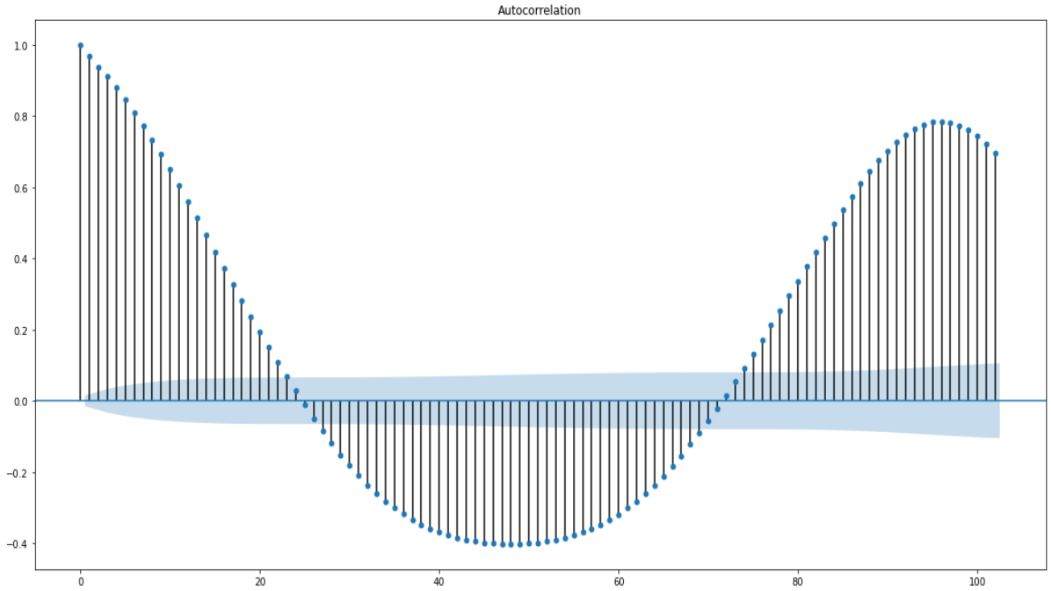

Figure 3.13: Autocorrelation plot obtained for values of the target variable with 102 previous timestamps

the previous day. The mean of these values corresponded to the rolling window statistics that were included.

3.4.2 Normalization

Since the range of values for each feature in our data set varied, normalization was essential in order to prevent features with larger values having a bigger influence on the final output, [61].

Each feature and the target variable were standardized as shown below so that it follows a normal distribution with zero mean and unit variance. Each variables mean value is denoted by $\bar{x}$ and σ denotes its standard deviation

$$x' = \frac{x - \bar{x}}{\sigma}.$$

No dominating variables meant no dominating weights during training that can cause bias, and faster back propagation. Ref. [62] explains this in further detail.

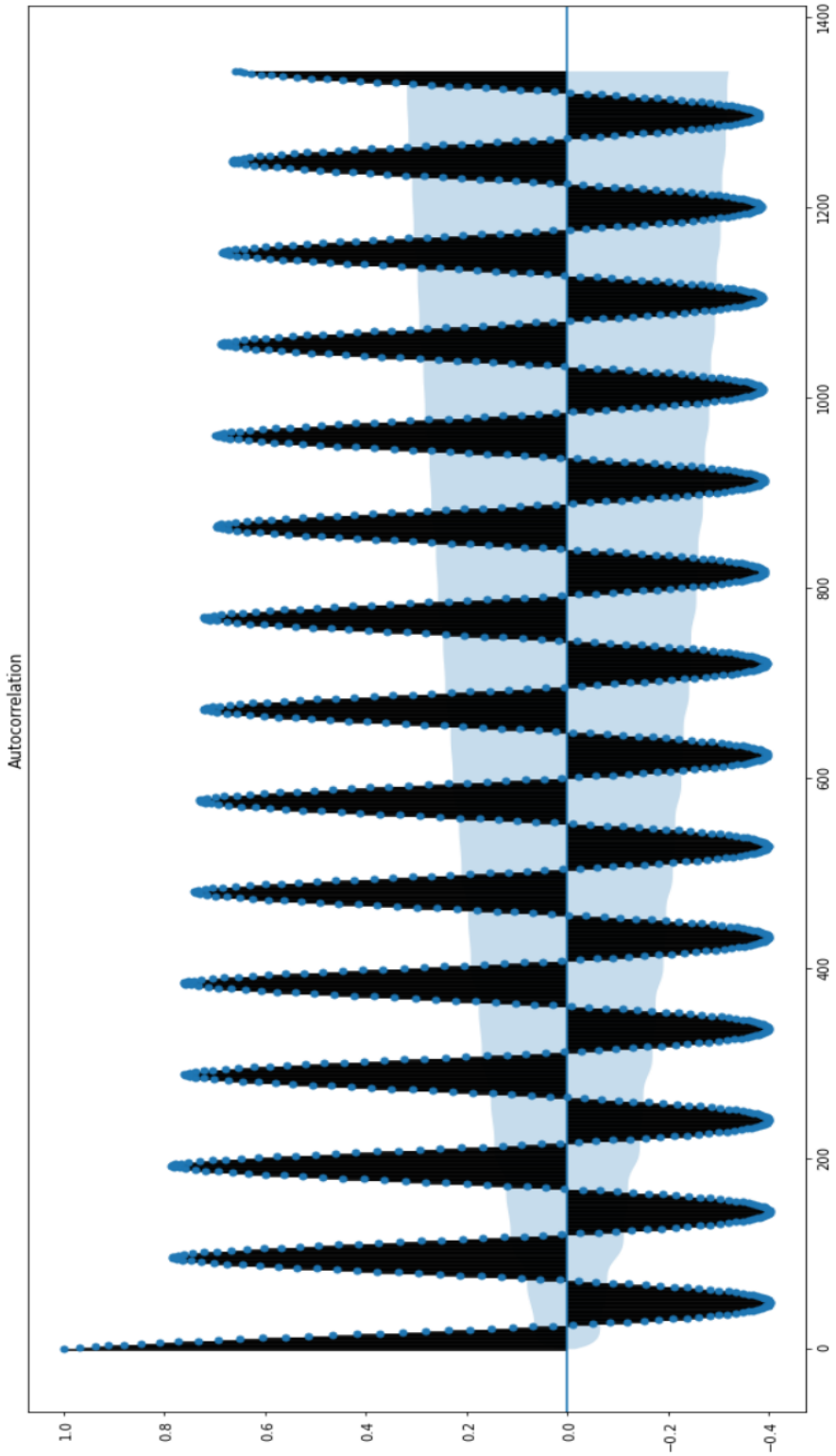

Figure 3.14: Autocorrelation plot obtained for values of the target variable with timestamps over 2 weeks before

3.5 Train and Test Data

The last two weeks of data out of the six months (January to June) were used as test data. The idea of k-fold cross validation was implemented as follows, [63].

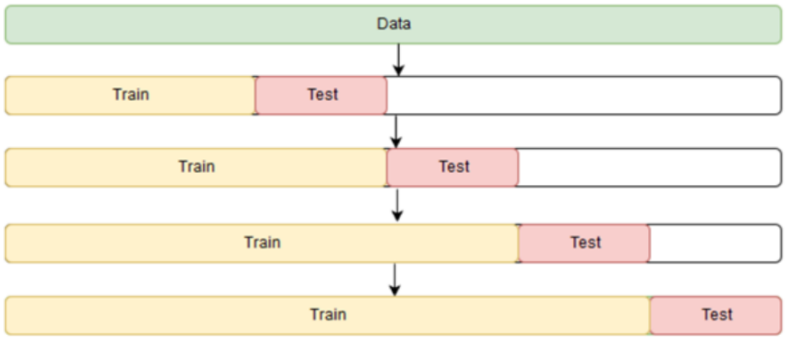

Figure 3.15: k-fold cross validation, from [63]

As illustrated in Figure 3.15, the test data included inputs for 24 hours in the future each time but two major differences in our use case were that:

- Test data sets overlapped since the model was run every 15 minutes during the day and predicted values for 24 hours ahead.
- The train data set was fixed since retraining the model with just one more entry every 15 minutes needed more computational time and is not feasible in 15 minutes.

Model performance on the test data was used to choose the best combination of hyperparameters for each model. Each model along with this combination was then fitted on all 6 months of data (January to June) and was then made to predict for the first two weeks of July (unseen data) and the results obtained were used to compare the suitability of different models.

4. Results

The following four error metrics, [64, 65, 66] were used to assess and compare the performance of the chosen models.

In all of the formulas below, y_i denotes the actual value, $\hat{y}_i$ denotes the forecasted value, n denotes the number of timesteps to predict for in the future, $\bar{y}$ denotes the mean actual value over the n timestamps and k denotes the number of input features.

- **Root mean squared error (RMSE)** - A quadratic scoring rule that measures the average magnitude of the error.

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

- **Mean absolute percentage error (MAPE)** - A measure of the prediction accuracy of the model or in simple terms a measure of how relatively large the errors are when compared to the actual values. This was specifically calculated only for all timestamps for which the actual solar forecast value recorded was non zero.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|.$$

- **R2 score** - This score also called the coefficient of determination is a measure of how well all the model inputs predict the required output.

$$R2score = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}.$$

- **Adjusted R2 score** - A special form of the R2 score. The R2 score can tend to depend on the number of features/inputs and might be high due to a high number of features used, whereas the adjusted R2 score will decrease if insignificant features are present.

$$adjustedR2 = 1 - \frac{(1 - R2score)(n-1)}{(n-k-1)}.$$

Adjusted R2 scores were used when testing with the baseline model in order to choose the best subset of input features and then the rest of the models were tested on the same subset but with varying combinations of hyperparameter values via the following method.

To begin with each model was tested with a subset of hyperparameters that resulted in a not too simple nor complex model in order to avoid undefitting or overfitting given that we were dealing with small amounts of data. The top 20 combinations of hyperparameters were then filtered based on the lowest MAPE value in the results obtained based on the models prediction for the test data set. This information was then used to test with simpler and complex models with the same test data set in order to see if they perform better, and the best model was chosen to be the model that produced the lowest MAPE value overall.

Although two other metrics were also used to assess model performance, MAPE was chosen to be the deciding metric based on the following factors, [67]:

- The fact that it returns the error as a percentage when multiplied by 100 makes it easier to interpret the model's ability to predict and compare its performance across a range of hyperparameter values.
- MAPE is calculated in this scenario by taking only instances where the actual value was non-zero and these are the crucial times of the day when we expect the predictions made by the model to be as accurate as possible. Given that there are fewer data points of this nature overall, it is important that the model learns these equally well and is able to identify the pattern/correlation between the feature and output values corresponding to these data points and MAPE gives us a good idea of how well the model can do this.
- MAPE is not relative to the magnitude of the observed and predicted values which is also another advantage especially given that the PV output can largely vary across the months of the year. This makes it an ideal metric to assess and compare model performance across months/seasons or when more data is used for training in the future.

The loss/error function I chose to use when training the neural networks was mean squared error (MSE), the squared version of RMSE, [68]. The code used for the training and testing of models can be found at `https://github.com/Abinaya-J/ThesisFiles/blob/main/ModelTraining.ipynb`

4.1 Baseline model

I chose a simple Linear Regression model to be my baseline model [69], since it is relatively easy to implement, to work with and also helps in identifying insignificant features through the interpretation of the regression coefficients.

feature	coefficient
HCC	0.000357
LCC	-0.023544
MCC	-0.003385
Pressure	0.018477
RH	-0.045397
SDR	0.167603
Temp	0.02503
TCC	0.002068
TP	0.003059
WD	0.020045
WS	-0.00587
month	-0.070235
dayofmonth	-0.013331
hour	-0.001417
minute	0.006406
LDI	0.026783
MostRecentForecast	0.640047
prev1	-0.070843
prev2	-0.091736
prev3	-0.031127
prev4	0.003496
prev1_mean	0.084997
prev2_mean	0.182954
prev3_mean	0.053087

Figure 4.1: Regression coefficients with respect to each feature

In order to see if removing the less significant features improved the adjusted R2 scores, I first chose to drop the features whose regression coefficient value was less than |0.01|. As per Figure 4.1, the features HCC, MCC, TCC, TP, WS, hour,

model	test_mape	test_rmse	test_r2_score	test_adj_r2_score
LinReg1	0.728	88.615	0.857	0.806

Figure 4.2: Test results with all input features

minute and prev4 were dropped and Figure 4.3 contains the error metrics obtained as a result.

model	test_mape	test_rmse	test_r2_score	test_adj_r2_score
LinReg2	0.695	88.692	0.857	0.825

Figure 4.3: Test results after the first set of features were dropped

When comparing the results in Figures 4.2 and 4.3 it is clear that removing features did result in slight improvement of the adjusted R2, RMSE and MAPE scores. Therefore I then also tried to refine the subset even more by removing the features Pressure, SDR, dayofmonth and prev2mean whose regression coefficient value was less than |0.02| but, as seen in Figure 4.4 this did not improve the results overall and therefore the first subset of features were used to test the rest of the models.

model	test_mape	test_rmse	test_r2_score	test_adj_r2_score
LinReg3	1.478	92.149	0.848	0.823

Figure 4.4: Test results after the second set of features were dropped

An overview of the results obtained using the baseline model (linear regression model fitted to the chosen subset of features) in order to predict for two weeks of unseen data can be observed in Figures 4.5 and 4.16.

The predicted values plotted in the final prediction plot corresponding to each model refer to the latest predictions made by the respective model for each corresponding timestamp.

4.2 LSTM

Initially, a simple LSTM with one hidden layer with 25 and 10 neurons in its first and second layers respectively was tested with a range of values for other hyperparameters.

model	pred_mape	pred_rmse	pred_r2_score
Baseline	0.364	82.781	0.889

Figure 4.5: Final prediction error metrics using the linear regression model

Figure 4.6 was referred to when choosing a suitable subset of values to test further with for each hyperparameter for LSTMs with no hidden layers (Figure 4.7) and two hidden layers (Figure 4.8) and as a result, dropout probabilities of 0.3 and 0.4 were chosen along with learning rates of 0.01 and 0.001, a batch size of 32 and the number of epochs tested for only included 300. I also additionally did test with models that contained 50 neurons in its first LSTM layer.

Having compared all the results obtained, the best result achieved via a LSTM when compared to that of the baseline model corresponded to the first row in Figure 4.8. This model was then used to predict for the two weeks of unseen data and produced the results shown by Figures 4.9 and 4.17.

4.3 CNN

To begin with, a simple CNN with one hidden layer with 32 and 16 filters in its first and second layers respectively was tested with the default kernel size of 1 and a range of values for other hyperparameters.

Figure 4.10 was referred to when choosing a suitable subset of values to test further with for each hyperparameter for CNNs with no hidden layers (Figure 4.11) and two hidden layers (Figure 4.12) and as a result, dropout probabilities of 0.3 and 0.4 were chosen along with learning rates of 0.01 and 0.001, kernel sizes of 1 and 2, a batch size of 32 and the number of epochs tested for only included 300.

Having compared all the results obtained, the best result obtained via a CNN when compared to that of the baseline model corresponded to the first row in Figure 4.11. This model was then used to predict for the two weeks of unseen data and produced the results shown by Figures 4.13 and 4.18.

4.4 CNN-LSTM

As a first step, a simple CNN-LSTM model that consisted of one CNN layer with 16 filters and the default kernel size of 1 and one LSTM layer with 10 neurons was tested with a range of values for other hyperparameters.

layer1_neurons	layer2_neurons	dropout	learning_rate	n_epochs	batch_size	test_mape	test_rmse	test_r2_score
25	10	0.4	0.0010	300	32	1.486	81.851	0.875
25	10	0.4	0.0010	50	32	1.736	92.508	0.846
25	10	0.2	0.0010	300	64	1.864	82.173	0.866
25	10	0.4	0.0010	100	64	1.977	94.558	0.839
25	10	0.3	0.0010	300	32	1.979	82.406	0.869
25	10	0.3	0.0001	300	32	2.051	96.131	0.831
25	10	0.4	0.0010	300	64	2.186	88.638	0.855
25	10	0.3	0.0010	100	64	2.255	88.709	0.854
25	10	0.3	0.0010	100	128	2.273	94.404	0.840
25	10	0.4	0.0010	300	128	2.351	90.542	0.850
25	10	0.2	0.0010	100	64	2.420	90.390	0.843
25	10	0.4	0.0010	100	32	2.432	88.288	0.855
25	10	0.4	0.0001	300	32	2.516	93.575	0.841
25	10	0.2	0.0010	50	32	2.616	88.678	0.850
25	10	0.2	0.0010	50	64	2.694	93.652	0.836
25	10	0.4	0.0001	50	32	2.739	113.428	0.789
25	10	0.3	0.0001	300	64	2.827	93.932	0.840
25	10	0.2	0.0010	100	32	2.945	84.451	0.864
25	10	0.3	0.0010	50	64	3.035	93.918	0.836
25	10	0.2	0.0001	300	64	3.056	94.425	0.837

Figure 4.6: The top 20 entries corresponding to the lowest MAPE scores for the LSTM with one hidden layer

Having compared all the results obtained, the best result obtained via a CNN-LSTM when compared to that of the baseline model corresponded to the first row in Figure 4.14. This model was used to predict for the two weeks of unseen data and produced the results shown by Figures 4.15 and 4.19.

layer1_neurons	dropout	learning_rate	n_epochs	batch_size	test_mape	test_rmse	test_r2_score
10	0.3	0.0001	300	32	1.492	93.115	0.840
50	0.4	0.0001	300	32	1.493	88.056	0.855
50	0.3	0.0001	300	32	1.859	87.014	0.857
25	0.3	0.0001	300	32	1.915	88.165	0.857
10	0.4	0.0001	300	32	2.226	98.078	0.838
50	0.3	0.0100	300	32	2.444	55.178	0.940
50	0.4	0.0100	300	32	2.625	55.820	0.941
10	0.4	0.0100	300	32	2.775	81.654	0.874
10	0.3	0.0100	300	32	3.156	77.194	0.884
25	0.3	0.0100	300	32	3.753	62.605	0.923
25	0.4	0.0001	300	32	3.833	90.119	0.852
25	0.4	0.0100	300	32	4.807	67.156	0.909

Figure 4.7: Results obtained using a LSTM with no hidden layer and the chosen subset of hyperparameters

layer1_neurons	layer2_neurons	layer3_neurons	dropout	learning_rate	n_epochs	batch_size	test_mape	test_rmse	test_r2_score
50	25	10	0.3	0.0100	300	32	0.515	52.877	0.946
50	25	10	0.4	0.0100	300	32	0.540	49.457	0.954
50	25	10	0.4	0.0001	300	32	2.827	97.339	0.820
50	25	10	0.3	0.0001	300	32	3.143	94.530	0.833

Figure 4.8: Results obtained using a LSTM with two hidden layers and the chosen subset of hyperparameters

model	pred_mape	pred_rmse	pred_r2_score
LSTM	0.356	105.81	0.796

Figure 4.9: Final prediction error metrics using the LSTM model

layer1_filters	layer2_filters	kernel_size	dropout	learning_rate	n_epochs	batch_size	test_mape	test_rmse	test_r2_score
32	16	1	0.3	0.0001	300	32	1.447	103.653	0.807
32	16	1	0.3	0.0010	300	32	2.029	84.433	0.862
32	16	1	0.4	0.0010	50	32	2.120	97.694	0.827
32	16	1	0.4	0.0001	300	32	2.123	112.055	0.781
32	16	1	0.3	0.0010	50	64	2.141	110.865	0.773
32	16	1	0.3	0.0010	50	128	2.288	121.609	0.738
32	16	1	0.4	0.0010	300	64	2.374	88.361	0.852
32	16	1	0.4	0.0010	100	128	2.423	105.828	0.793
32	16	1	0.3	0.0001	300	64	2.566	115.438	0.758
32	16	1	0.2	0.0010	100	64	2.708	98.415	0.824
32	16	1	0.3	0.0010	300	128	2.714	93.962	0.831
32	16	1	0.3	0.0010	100	32	2.823	91.754	0.842
32	16	1	0.3	0.0010	50	32	2.872	99.893	0.817
32	16	1	0.2	0.0010	300	64	2.902	88.672	0.849
32	16	1	0.4	0.0010	300	32	2.907	84.162	0.863
32	16	1	0.2	0.0001	300	32	2.918	106.590	0.796
32	16	1	0.2	0.0010	300	128	2.941	96.721	0.828
32	16	1	0.3	0.0001	100	32	3.333	121.134	0.735
32	16	1	0.4	0.0010	300	128	3.371	92.246	0.841
32	16	1	0.2	0.0010	50	32	3.571	98.778	0.814

Figure 4.10: The top 20 entries corresponding to the lowest MAPE scores for the CNN with one hidden layer

layer1_filters	layer2_filters	kernel_size	dropout	learning_rate	n_epochs	batch_size	test_mape	test_rmse	test_r2_score
16	0	2	0.3	0.010	300	32	0.860	77.089	0.881
32	0	2	0.3	0.010	300	32	1.181	77.383	0.882
16	0	2	0.4	0.010	300	32	1.212	78.919	0.876
16	0	1	0.3	0.010	300	32	1.413	80.412	0.873
16	0	1	0.4	0.010	300	32	1.423	80.080	0.873
32	0	2	0.4	0.010	300	32	1.427	78.859	0.878
32	0	1	0.4	0.010	300	32	1.570	77.607	0.880
16	0	1	0.3	0.001	300	32	1.766	83.068	0.864
32	0	2	0.4	0.001	300	32	1.797	83.705	0.863
32	0	2	0.3	0.001	300	32	1.861	84.059	0.860
32	0	1	0.3	0.010	300	32	2.047	77.530	0.880
16	0	2	0.3	0.001	300	32	2.282	84.622	0.862
32	0	1	0.3	0.001	300	32	2.427	82.341	0.865
16	0	2	0.4	0.001	300	32	2.690	84.220	0.862
32	0	1	0.4	0.001	300	32	3.040	83.570	0.864
16	0	1	0.4	0.001	300	32	3.790	86.107	0.854

Figure 4.11: Results obtained using a CNN with no hidden layer and the chosen subset of hyperparameters

layer1_filters	layer2_filters	layer3_filters	kernel_size	dropout	learning_rate	n_epochs	batch_size	test_mape	test_rmse	test_r2_score
64	32	16	1	0.3	0.010	300	32	1.386	67.733	0.910
64	32	16	1	0.4	0.010	300	32	1.575	67.725	0.909
64	32	16	2	0.3	0.010	300	32	2.771	63.103	0.921
64	32	16	1	0.3	0.001	300	32	3.094	82.681	0.866
64	32	16	2	0.4	0.010	300	32	3.152	67.575	0.913
64	32	16	2	0.4	0.001	300	32	4.012	81.968	0.869
64	32	16	2	0.3	0.001	300	32	4.204	81.740	0.870
64	32	16	1	0.4	0.001	300	32	4.451	83.347	0.864

Figure 4.12: Results obtained using a CNN with two hidden layers and the chosen subset of hyperparameters

model	pred_mape	pred_rmse	pred_r2_score
CNN	0.296	75.269	0.907

Figure 4.13: Final prediction error metrics using the CNN model

conv_filters_l1	kernel_size	lstm_nodes_l1	dropout	learning_rate	n_epochs	batch_size	test_mape	test_rmse	test_r2_score
16	1	10	0.3	0.00010	100	64	0.636	107.567	0.801
16	1	10	0.2	0.00100	100	128	0.998	93.149	0.845
16	1	10	0.3	0.00001	300	64	1.022	327.437	-0.600
16	1	10	0.2	0.00100	300	64	1.045	87.783	0.858
16	1	10	0.4	0.00010	50	32	1.079	133.485	0.718
16	1	10	0.4	0.00100	300	128	1.086	94.180	0.842
16	1	10	0.4	0.00100	300	32	1.304	88.106	0.859
16	1	10	0.4	0.00100	50	32	1.386	89.429	0.856
16	1	10	0.2	0.00001	300	128	1.444	136.570	0.663
16	1	10	0.2	0.00010	100	32	1.454	100.592	0.817
16	1	10	0.3	0.00001	100	32	1.575	141.772	0.628
16	1	10	0.4	0.00001	300	128	1.629	219.616	0.219
16	1	10	0.4	0.00100	100	128	1.637	104.952	0.811
16	1	10	0.3	0.00001	300	32	1.662	130.411	0.714
16	1	10	0.3	0.00010	50	64	1.750	121.624	0.735
16	1	10	0.4	0.00010	100	32	1.753	125.660	0.738
16	1	10	0.3	0.00010	100	32	1.894	108.967	0.784
16	1	10	0.3	0.00010	300	128	1.920	110.855	0.797
16	1	10	0.3	0.00100	100	32	1.957	94.537	0.837
16	1	10	0.4	0.00010	300	128	2.073	119.390	0.761

Figure 4.14: The top 20 entries corresponding to the lowest MAPE scores for the CNN-LSTM with one CNN layer and one LSTM layer

model	pred_mape	pred_rmse	pred_r2_score
CNN-LSTM	0.54	102.643	0.835

Figure 4.15: Final prediction error metrics using the CNN-LSTM model

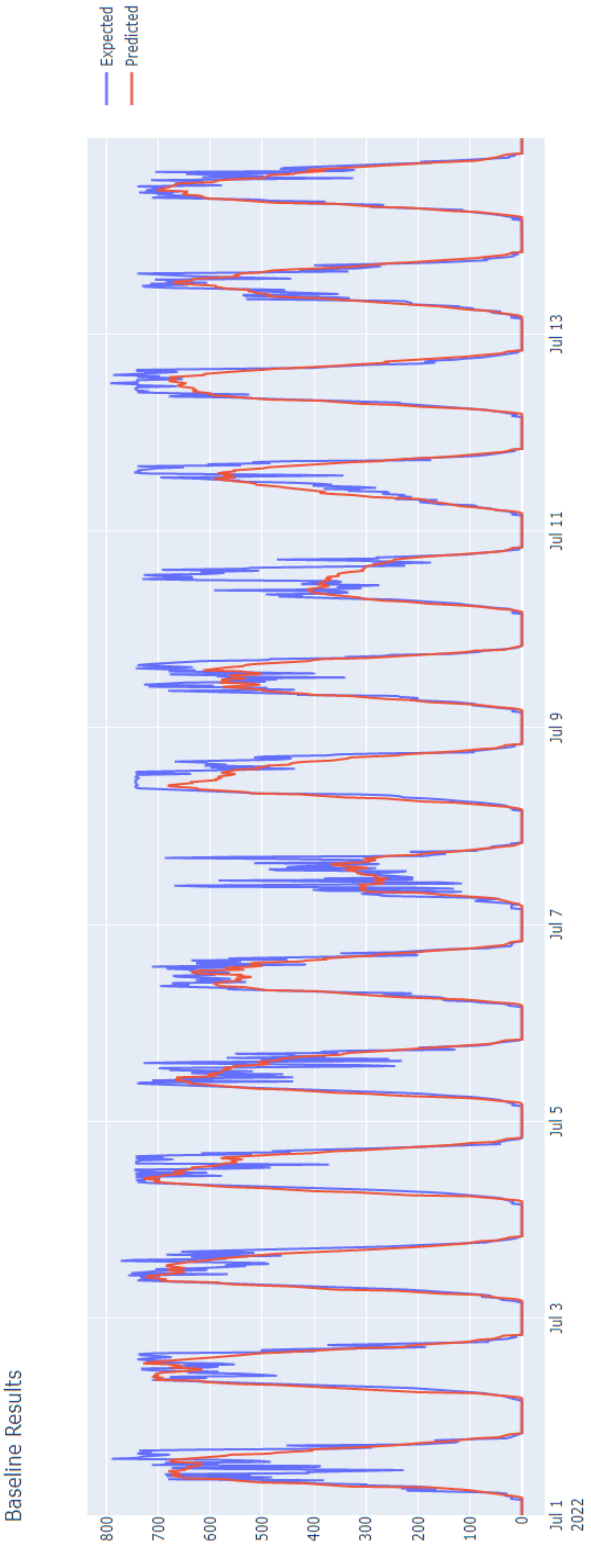

Figure 4.16: Final prediction plot for the baseline model

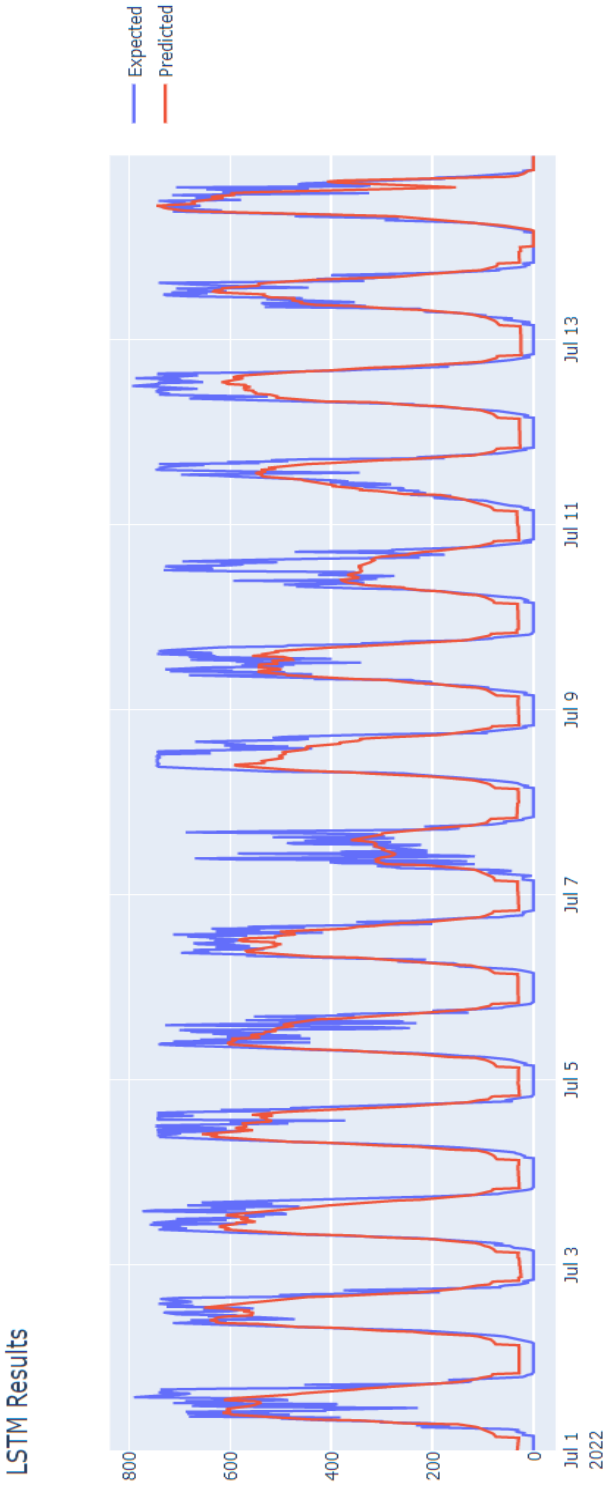

Figure 4.17: Final prediction plot for the LSTM model

Figure 4.18: Final prediction plot for the CNN model

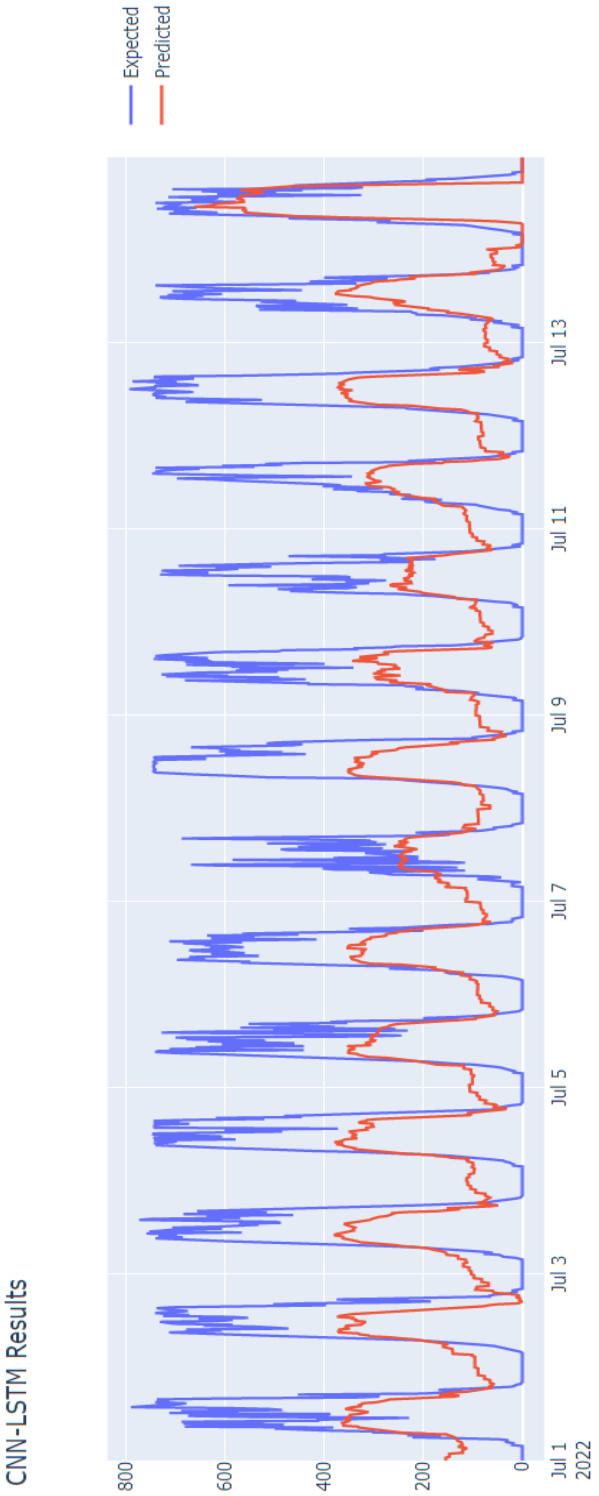

Figure 4.19: Final prediction plot for the CNN-LSTM model

5. Evaluation

To start off, lets analyze the suitability of the various models chosen for this task based on how well they could predict for unseen data.

Model	pred_mape	pred_rmse	pred_r2_score
Baseline	0.364	82.781	0.889
LSTM	0.356	105.81	0.796
CNN	0.296	75.269	0.907
CNN-LSTM	0.54	102.643	0.835

Figure 5.1: Final error metric scores obtained using each model during prediction

Figure 5.1 clearly shows us that the CNN achieved the lowest MAPE and RMSE scores and also the highest R2 score, thereby proving to be the best model. Both the LSTM and the CNN-LSTM did not perform in favour of choosing them over the baseline model. Although the LSTM did achieve a better MAPE score when compared to the baseline model, this was not observed in the other two error metrics.

In order to get an in-depth picture of how the different models performed when compared with one another during both testing and prediction, we will also take a look at the final test and pred MAPE values (since this metric was used to choose the final model used for prediction), the optimal hyperparameter values and the final plots obtained using each of them.

When comparing the final MAPE scores obtained when testing (column 1 in Figure 5.2), the CNN-LSTM evidently outperformed the baseline model itself, while both the LSTM and CNN models did not. In contradiction to this, looking closely at the MAPE values obtained when using these models to predict for unseen data (column 2 in Figure 5.2), both the LSTM and CNN models outperform the baseline

Model	test_mape	pred_mape
Baseline	0.695	0.364
LSTM	1.486	0.356
CNN	1.447	0.296
CNN-LSTM	0.636	0.54

Figure 5.2: Final MAPE scores obtained using each model during testing and prediction

whereas the CNN-LSTM does not. This is one clear example of the CNN-LSTM model being too complex for limited data which has in turn resulted in overfitting and therefore this model fails to generalize when exposed to new data, whereas the LSTM and CNN models seem to generalize well.

Another important fact to note is that the best performing LSTM model needed two hidden layers whereas the CNN needed no hidden layer which means complexity does have an impact on the performance depending on the model. The optimal hyperparameter values for the ones that were common to all models were identical for the LSTM and CNN models (dropout probability of 0.3, learning rate of 0.01, a batch size of 32 and 300 epochs) but mostly varied for the CNN-LSTM model (dropout probability of 0.3, learning rate of 0.0001, a batch size of 64 and 100 epochs).

Closely observing the final plots obtained, Figures 4.16, 4.17, 4.18 and 4.19, they match the results mentioned above. Figure 4.18 clearly establishes how the CNN model outperforms the baseline model and predicts both at the peaks and troughs better then both the LSTM and CNN-LSTM models. As shown in Figure 4.17, although the LSTM performed better then the CNN-LSTM model, it still does not manage to predict well at all peaks and at most troughs. Lastly, Figure 4.19 reveals how the CNN-LSTM fails to predict well at almost all peaks and troughs.

6. Conclusion

In an attempt to find an answer to the question **Can deep learning (or combining best known methods) yield accurate predictions regarding the PV output needed to maintain the balance between the production and usage of solar energy in households with limited input?** by mainly referring to the findings in Ref. [4] the LSTM model, the CNN model and a combination of the two, the CNN-LSTM model were tested for their ability to predict future values when provided with limited data to learn from and were compared against a simple linear regression model that was used as a baseline model. The CNN model performed the best when compared to the other two models used but this can definitely change depending on the features and hyperparameters chosen to work with. Recent advancements in machine learning based techniques, particularly deep learning algorithms, have caused these methods to gain popularity among researchers for time series forecasting and have also proved to be quite effective and accurate as traditional methods which I think was quite evident from the findings presented in this thesis. However, in order to deploy these models over simple models like the baseline model a lot more testing is required with new data in order to see if it constantly outperforms the baseline model and not just by a minor difference in the error metrics mainly due to the extra time and resources needed to train and update these models.

6.1 Future Work

The tested methods and hyperparameters used can act as a starting point when we have less data even for other use cases that require time series forecasting and later customised according to how they perform just like how it was done in this thesis. Regarding the approach used in this thesis, the models, training methods and parameters could be changed or adjusted further to rerun the experiments with or without additional up to date data which might result in better performance.

The subset of features chosen to test models with could also be varied in order to check if this would have an impact on the models ability to predict future values. The models could also be made to predict for shorter periods in the future, for example 6 hours instead of 24 which could also improve accuracy.

References

[1] Olesia Martynova. "Opportunities and Challenges of Artificial Intelligence in the Energy Sector". In: (Feb. 2020). URL: https://intellias.com/opportunities-and-challenges-of-artificial-intelligence-in-the-energy-sector/.

[2] Jenny Palm. "Household installation of solar panels – Motives and barriers in a 10-year perspective". In: *Energy Policy* 113 (Feb. 2018), pp. 1–8. DOI: 10.1016/j.enpol.2017.10.047.

[3] Diogo M. F. Izidio et al. "Evolutionary Hybrid System for Energy Consumption Forecasting for Smart Meters". In: *Energies* 14.7 (Mar. 2021). DOI: 10.3390/en14071794.

[4] Pedro Lara-Benítez, Manuel Carranza-García, and José C. Riquelme. "An Experimental Review on Deep Learning Architectures for Time Series Forecasting". In: *International Journal of Neural Systems* 31.03 (Feb. 2021). DOI: 10.1142/s0129065721300011.

[5] Francisco Martínez-Álvarez et al. "A Survey on Data Mining Techniques Applied to Electricity-Related Time Series Forecasting". In: *Energies* 8.11 (Nov. 2015). ISSN: 1996-1073. DOI: 10.3390/en81112361. URL: http://dx.doi.org/10.3390/en81112361.

[6] Yang Lyla. "A Quick Deep Learning Recipe: Time Series Forecasting with Keras in Python". Apr. 2020. URL: https://towardsdatascience.com/a-quick-deep-learning-recipe-time-series-forecasting-with-keras-in-python-f759923ba64.

[7] Taehee Jeong et al. *BUILDING A SMART PARTNERSHIP FOR THE FOURTH INDUSTRIAL REVOLUTION*. Tech. rep. Atlantic Council, 2018, pp. 15–21. URL: http://www.jstor.org/stable/resrep20947.5.

[8] David Fumo. "Types of Machine Learning Algorithms You Should Know". June 2017. URL: https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861.

[9] Katrina Wakefield. "A guide to machine learning algorithms and their applications". 2019. URL: https://www.sas.com/en_gb/insights/articles/analytics/machine-learning-algorithms.html.

[10] Emma Juliana Gachancipa Castelblanco. *How to choose an activation function?* May 2020. URL: https://www.linkedin.com/pulse/how-choose-activation-function-emma-juliana-gachancipa-castelblanco.

[11] Warren E. Agin. "A Simple Guide to Machine Learning". In: *Business Law Today* (2017), pp. 1–5. ISSN: 10599436, 23758112. URL: https://www.jstor.org/stable/90003559.

[12] Simeon Kostadinov. "Understanding Backpropagation Algorithm". Aug. 2019. URL: https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd.

[13] Michael A Nielsen. "Neural Networks and Deep Learning". Dec. 2019. URL: http://neuralnetworksanddeeplearning.com/chap2.html.

[14] SAGAR SHARMA. *Epoch vs Batch Size vs Iterations*. Sept. 2017. URL: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9.

[15] Catherine F. Higham and Desmond J. Higham. "Deep Learning: An Introduction for Applied Mathematicians". In: *SIAM Review* 61.3 (Jan. 2019), pp. 860–891. DOI: 10.1137/18m1165748. URL: https://arxiv.org/pdf/1801.05894.pdf.

[16] datahacker.rs. "Gradient Descent". Oct. 2018. URL: https://datahacker.rs/gradient-descent/.

[17] Jason Brownlee. *How to Avoid Overfitting in Deep Learning Neural Networks*. Dec. 2018. URL: https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/?source=post_page-----e05e64f9f07----------------------.

[18] Artem Oppermann. *Regularization in Deep Learning — L1, L2, and Dropout*. Aug. 2020. URL: https://towardsdatascience.com/regularization-in-deep-learning-l1-l2-and-dropout-377e75acc036.

[19] Yash Bohra. "Vanishing and Exploding Gradients in Deep Neural Networks". June 2021. URL: https://www.analyticsvidhya.com/blog/2021/06/the-challenge-of-vanishing-exploding-gradients-in-deep-neural-networks/.

[20] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Inc., Sept. 2019. ISBN: 9781492032649. URL: https://www.knowledgeisle.com/wp-content/uploads/2019/12/2-Aur%C3%A9lien-G%C3%A9ron-Hands-On-Machine-Learning-with-Scikit-Learn-Keras-and-Tensorflow_-

Concepts-Tools-and-Techniques-to-Build-Intelligent-Systems-O%E2%80%99Reilly-Media-2019.pdf.

[21] Domas Bitvinskas. *ELU Activation Function*. July 2020. URL: https://closeheat.com/blog/elu-activation-function.

[22] Jason Brownlee. *Weight Initialization for Deep Learning Neural Networks*. Feb. 2021. URL: https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/.

[23] Aravindpai. *CNN vs. RNN vs. ANN - Analyzing 3 Types of Neural Networks*. Feb. 2020. URL: https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/.

[24] Priyal Walpita. *Recurrent Neural Networks in Deep Learning — Part 1*. Mar. 2020. URL: https://medium.datadriveninvestor.com/recurrent-neural-networks-in-deep-learning-part-1-df3c8c9198ba.

[25] Will Koehrsen. "Recurrent Neural Networks by Example in Python". Nov. 2018. URL: https://towardsdatascience.com/recurrent-neural-networks-by-example-in-python-ffd204f99470.

[26] Diego Unzueta. "Convolutional Layers vs Fully Connected Layers". Nov. 2021. URL: https://towardsdatascience.com/convolutional-layers-vs-fully-connected-layers-364f05ab460b.

[27] Macnica's ARIH (AI Research InnovationHub). "Convolutional Neural Network (CNN) for Time Series Classification". Oct. 2020. URL: https://www.macnica.co.jp/business/ai_iot/columns/135112/.

[28] Jason Brownlee. *How to Develop Convolutional Neural Network Models for Time Series Forecasting*. Nov. 2018. URL: https://machinelearningmastery.com/how-to-develop-convolutional-neural-network-models-for-time-series-forecasting/.

[29] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks—the ELI5 way*. Dec. 2018. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[30] Shipra Saxena. *LSTM | Introduction to LSTM | Long Short Term Memor*. Mar. 2021. URL: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/.

[31] Rian Dolphin. *LSTM Networks | A Detailed Explanation*. Mar. 2021. URL: https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9.

[32] H.D. Nguyen et al. "Forecasting and Anomaly Detection approaches using LSTM and LSTM Autoencoder techniques with the applications in supply chain management". In: *International Journal of Information Management* 57 (Apr. 2021), p. 102282. DOI: 10.1016/j.ijinfomgt.2020.102282.

[33] Nir Arbel. *How LSTM networks solve the problem of vanishing gradients*. May 2020. URL: https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577.

[34] Harpreet Singh Sachdev. *Choosing number of Hidden Layers and number of hidden neurons in Neural Networks*. Jan. 2020. URL: https://www.linkedin.com/pulse/choosing-number-hidden-layers-neurons-neural-networks-sachdev#:~:text=1%20Well%20if%20the%20data%20is%20linearly%20separable.

[35] Jason Brownlee. *Dropout Regularization in Deep Learning Models With Keras*. June 2016. URL: https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/.

[36] Saulo Barreto. *Choosing a Learning Rate | Baeldung on Computer Science*. Nov. 2021. URL: https://www.baeldung.com/cs/ml-learning-rate.

[37] Enes Zvornicanin. *Relation Between Learning Rate and Batch Size | Baeldung on Computer Science*. Jan. 2022. URL: https://www.baeldung.com/cs/learning-rate-batch-size.

[38] Renu Khandelwal. *Convolutional Neural Network(CNN) Simplified*. Oct. 2018. URL: https://medium.datadriveninvestor.com/convolutional-neural-network-cnn-simplified-ecafd4ee52c5.

[39] Swarnima Pandey. *How to choose the size of the convolution filter or Kernel size for CNN?* July 2020. URL: https://medium.com/analytics-vidhya/how-to-choose-the-size-of-the-convolution-filter-or-kernel-size-for-cnn-86a55a1e2d15.

[40] Krisha Samir Mehta. *Weights Biases*. Feb. 2022. URL: https://wandb.ai/krishamehta/seo/reports/Difference-Between-SAME-and-VALID-Padding-in-TensorFlow--VmlldzoxODkwMzE.

[41] Python. *Welcome to Python.org*. May 2019. URL: https://www.python.org/.

[42] Numpy. *NumPy*. 2009. URL: https://numpy.org/.

[43] Pandas. *Python Data Analysis Library — pandas: Python Data Analysis Library*. 2018. URL: https://pandas.pydata.org/.

[44] Scikit-Learn. *User guide: contents — scikit-learn 0.22.1 documentation*. 2019. URL: https://scikit-learn.org/stable/user_guide.html.

[45] TensorFlow. *Effective TensorFlow 2 | TensorFlow Core*. URL: https://www.tensorflow.org/guide/effective_tf2.

[46] Plotly. *Plotly Python Graphing Library*. URL: https://plotly.com/python/.

[47] Vidhyashankar Venkatachalaperumal and Afshin Bakhtiari. "How Photovoltaic modules operate in different weather". June 2021. URL: https://ae-solar.com/solar-panels-in-different-weather/.

[48] URL: https://api.rebase.energy/weather/docs/v2/#hirlam-fmi-identifier-fmi_hirlam.
[49] URL: https://api.rebase.energy/weather/docs/v2/#variables-3.
[50] URL: https://www.elia.be/en/grid-data/power-generation/solar-pv-power-generation-data.
[51] URL: https://www.elia.be/en/grid-data/power-generation.
[52] Neha Seth. *What Is Data Preprocessing in Machine Learning, and Its Importance?* Nov. 2021. URL: https://www.analytixlabs.co.in/blog/data-preprocessing-in-machine-learning/.
[53] Will Badr. *Why Feature Correlation Matters .... A Lot!* Jan. 2019. URL: https://towardsdatascience.com/why-feature-correlation-matters-a-lot-847e8ba439c4.
[54] Aishwarya V. Srinivasan. *Why exclude highly correlated features when building regression model ??* Sept. 2019. URL: https://towardsdatascience.com/why-exclude-highly-correlated-features-when-building-regression-model-34d77a90ea8e.
[55] URL: https://sunrise-sunset.org/api.
[56] Akshita Chugh. *How to deal with missing values in data set ?* Jan. 2021. URL: https://medium.com/analytics-vidhya/how-to-deal-with-missing-values-in-data-set-8e8f70ecf155#:~:text=%20How%20to%20deal%20with%20missing%20values%20in.
[57] Shay Palachy. *Detecting stationarity in time series data.* Nov. 2019. URL: https://towardsdatascience.com/detecting-stationarity-in-time-series-data-d29e0a21e638.
[58] Selva Prabhakaran. *Augmented Dickey Fuller Test (ADF Test) – Must Read Guide.* Nov. 2019. URL: https://www.machinelearningplus.com/time-series/augmented-dickey-fuller-test/.
[59] Jason Brownlee. *Basic Feature Engineering With Time Series Data in Python.* Dec. 2016. URL: https://machinelearningmastery.com/basic-feature-engineering-time-series-data-python/.
[60] Alan Anderson and David Semmelroth. *Autocorrelation Plots: Graphical Technique for Statistical Data.* Mar. 2016. URL: https://www.dummies.com/article/technology/information-technology/data-science/big-data/autocorrelation-plots-graphical-technique-for-statistical-data-141241/.
[61] Urvashi Jaitley. *Why Data Normalization is necessary for Machine Learning models.* Oct. 2018. URL: https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029.

[62] Sergey Ioffe. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. URL: https://arxiv.org/pdf/1502.03167.pdf.

[63] Soumya Shrivastava. *Cross Validation in Time Series*. Jan. 2020. URL: https://medium.com/@soumyachess1496/cross-validation-in-time-series-566ae4981ce4.

[64] Shwetha Acharya. *What are RMSE and MAE?* June 2021. URL: https://towardsdatascience.com/what-are-rmse-and-mae-e405ce230383.

[65] Zach. *How to Interpret Adjusted R-Squared (With Examples)*. Mar. 2022. URL: https://www.statology.org/adjusted-r-squared-interpretation/.

[66] Konstantin Rink. *Time Series Forecast Error Metrics you should know*. Nov. 2021. URL: https://towardsdatascience.com/time-series-forecast-error-metrics-you-should-know-cc88b8c67f27.

[67] Stephen Allwright. *RMSE vs MAPE, which is the best regression metric?* July 2022. URL: https://stephenallwright.com/rmse-vs-mape/.

[68] Jason Brownlee. *Loss and Loss Functions for Training Deep Learning Neural Networks*. May 2019. URL: https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/.

[69] Aashish Nair. *Baseline Models: Your Guide For Model Building*. Apr. 2022. URL: https://towardsdatascience.com/baseline-models-your-guide-for-model-building-1ec3aa244b8d.