UiA University of Agder

Forecasting of Photovoltaic Power Production

ENE500 - Master's Thesis Renewable Energy

THALE VASSNES STEFANSSON

SUPERVISORS

Rune Strandberg and Ghali Raja Yakoub

University of Agder, 2022

Faculty of Engineering and Science
Department of Engineering and Sciences

Master

Abstract

Solar irradiance and temperature are some weather parameters that affect the amount of power photovoltaic cells can generate. Based on these and past power production, future production can be predicted. Knowing" future generation may help the integration of this renewable energy source on an even larger scale than today, as well as optimize the use of them today. In this thesis, forecasting of future power generation was made by an artificial neural network (ANN) model, a support vector regression (SVR) model, an auto-regressive integrated moving average (ARIMA) model, a quantile regression neural network (QRNN) model, an ensemble model of ANN and SVR, an ANN ensemble model and an ANN model using only numerical weather predictions (NWPs) as inputs. Correlation techniques and principal component analysis were used for feature reduction for all models.

The research questions for this thesis are, "How will the models perform using random train data to predict August 2021, compared to a random test sample? Will the ensemble models perform better than the standalone models, and will the quantile regression neural network make accurate prediction intervals? How well will the predictions be if the ANN model only uses NWP data as inputs, compared to both historical power and NWPs?". As well as to answer these questions, the objective of this thesis is to provide a model or multiple models that can accurately predict future power production for the PV power system in Lillesand.

All models can predict future power production, but some with less accuracy than others. Of all models, as expected, both ensemble models performed best overall for both tests. The SVR model did however perform with the lowest MAE for the August test. For different fits, these results will probably slightly change, but it is expected that the ensemble models will still perform best overall.

Preface

My master's thesis is before you. The target group for this thesis is those who have similar previous knowledge as I have, who has done some machine learning before, and are familiar with forecasting. My excitement and yearning for working on forecasting photovoltaic power production originate from my passion for the global environment and climate. Improving the forecast may contribute to a better climate with lower greenhouse gas emissions, as the need for more renewable energy sources is acknowledged worldwide. Earlier, I worked on similar smaller projects with wind and solar power, in my first year in the master's program. More recently, last semester, I did a research project also called a pre-project on the same topic as a preparation for this thesis.

I started working on this thesis in January 2022 and was delivered in May 2022, corresponding to 30 study credits. The thesis considers different prediction models for predicting future PV power production. I have managed to complete the project tasks and answer the research questions of this thesis after doing extensive investigations in the field of forecasting photovoltaic power production and with the help of my supervisors. I would like to thank my supervisors for their support, guidance, and availability. In addition, I want to thank Gorines for providing historical power production data from their system, and their desire of being able to efficiently manage their dispatch of power. Thanks also go to my friends and family, for supporting me during this time.

Individual/group Mandatory Declaration

1.	I/We hereby declare that my/our report is my/our own work and that I/We have not used any other sources or have received any other help than mentioned in the report.	[✓ YES]
2.	**I/we further declare that this report:** • has not been used for another exam at another department/university/university college in Norway or abroad; • does not refer to the work of others without it being stated; • does not refer to own previous work without it being stated; • have all the references given in the literature list; • is not a copy, duplicate or copy of another's work or manuscript.	[✓ YES]
3.	I/we am/are aware that violation of the above is regarded as cheating and may result in cancellation of exams and exclusion from universities and colleges in Norway, see Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	[✓ YES]
4.	I/we am/are aware that all submitted reports may be checked for plagiarism.	[✓ YES]
5.	I/we am/are aware that the University of Agder will deal with all cases where there is suspicion of cheating according to the university's guidelines for dealing with cases of cheating.	[✓ YES]
6.	I/we have incorporated the rules and guidelines in the use of sources and references on the library's web pages.	[✓ YES]

Publishing Agreement

I hereby give the University of Agder a free right to make the task available for electronic publishing:	[✓ YES]
Is the report confidential?	[✓ NO]
Is the task except for public disclosure?	[✓ NO]

Contents

List of Figures

List of Tables

Notation and Abbreviations

The list describes several symbols and abbreviations that will be used within the body of the document

Notation

η	Controlling parameter
$\gamma(PICP)$	Variable in CWC/CLC
γ	Kernel coefficient in SVR
$\hat{y}_i$	Predicted/forecasted value
μ	Controlling parameter
ω	Weight vector
$\overline{y}_i$	Average of the true/measured value
ψ	White noise
θ_i	is the jth MA coefficient
ε	Accuracy term in SVR, tube radius
ε_i	Variable in PICP
φ_i	is the ith AR coefficient
$\vartheta(x)$	Non-linear mapping function
b	Bias
C	Penalty factor in SVR
d	Number of nonseasonal differences
f	Function
h	Hour
$h_1(\cdot)$	Activation function

$h_2(\cdot)$	Activation function
I	Input
j	Neuron
k	Number of neighbors in kNN
L	Lag operator
$L\varepsilon$	Vapnik's ε-insensitive loss function
L_i	Lower bound of the prediction interval
m	Number of hidden nodes
N	Number of data/input samples
n	Number of data/input samples
O	Output
p	Order of the AR model
P_t	Predicted/forecasted value
q	Order of the MA error term
R	Structural risk function
r	Size of independent variable
R^2	Coefficient of determination
T	Transposed
t	Time
t_h	Time horizon
U_i	Upper bound of the prediction interval
V_i	Connection weight
w	Weighted coefficient
W_i	Weight vector
X	Independent vector
x	Input
x_{test_1}	Second input test data

x_{test}	Input test data
x_{train}	Input train data
Y	Dependent vector
y	Output
y_i	True/measured value
y_{test_1}	Second output test data
y_{test}	Output test data
y_{train}	Output train data

Abbreviations

$\varepsilon - SVR$	ε-Support Vector Regression
AC	Alternating Current
$Adagrad$	Adaptive Gradient Algorithm
$AnEn$	Analog Ensemble
$ANFIS$	Adaptive Neuro-Fuzzy Inference System
ANN	Artificial Neural Network
$ARIMA$	Auto-Regressive Integrated Moving Average
$ARMA$	Auto-Regressive Moving Average
AVG	Average
$BiLSTM$	Bidirectional Long Short-Term Memory
BP	Backpropagation
BRR	Bayesian Ridge Regressor
CC	Cloud Cover
CLC	Coverage length-based criterion
CNN	Convolutional Neural Network
CWC	Coverage width-based criterion
DBN	Deep Belief Network
DI	Direct Radiation

DNN	Deep Neural Network
Dr	Diffuse Radiation
$ECMWF$	European Centre for Medium Range Weather Forecasting
$EDBN$	Enhanced DBN
ELM	Extreme Learning machine
ET	Extremely Randomized Trees
GBR	Gradient Boosted Regression
GBR	Gradient Boosting Regressor
GHI	Global Horizontal Irradiance
GI	Global Radiation
GPR	Gaussian Process Regressor
GRU	Gated Recurrent Unit
IFS	Integrated Forecasting System
KDE	Kernel Density Estimation
kNN	k-Nearest Neighbor
kNN	k-Nearest Neighbors
kW	Kilowatt
kWh	Kilowatt hours
LAR	Lasso Regressor
LR	Linear Regressor
$LSSVR$	Least Squares Support Vector Regression
$LSTM$	Long Short-Term Memory
MAE	Mean Absolute Error
$MAPE$	Mean Absolute Percentage Error
MF	Mondrian Forest
$MLFF$	Multilayer Feed-Forward
MLP	Multi-layer Perceptron

$MLPNN$	Multi-layer Perceptron Neural Network
$MLPR$	Multi Layer Perceptron Regressor
MMP	Mean Measured Power
MP	Mean Power
MSE	Mean Squared Error
MW	Megawatt
$NARMAX$	Non-linear Auto-Regressive Moving Average with exogenous inputs
$NARX$	Non-linear Auto-Regressive Network with exogenous inputs
$nRMSE$	normalized Root Mean Squared Error
NWP	Numerical Weather Prediction
P	Production
PCA	Principal Component Analysis
PDF	Probability Density Functions
$PeEn$	Persistence Ensemble
$PICP$	Prediction Interval Coverage Probability
$PINAW$	Prediction Interval Normalized Average Width
PNN	Probabilistic Neural Network
$Prec$	Precipitation
PV	Photovoltaic
QR	Quantile Regression
$QRNN$	Quantile Regression Neural Network
$QRNNE-UCV$	Quantile Regression Neural Network based on Epanechnikov kernel function using Unbiased Cross-Validation
$QRNNT$	Quantile Regression Neural Network and Triangle kernel function
RBF	Radial-Basis Function
$ReLU$	Rectified Linear Unit
RF	Random Forest

RFR	Random Forest Regression
RFR	Random Forest Regressor
$RMSE$	Root Mean Squared Error
$RMSProp$	Root Mean Square Propagation
RNN	Recurrent Neural Network
RR	Ridge Regressor
$SARIMA$	Seasonal Auto-Regressive Integrated Moving Average
SGD	Stochastic Gradient Descent
STD	Standard Deviation
SVM	Support Vector Machine
$SVQR$	Support Vector Quantile Regression
SVR	Support Vector Regression
$Temp$	Temperature
$TLFN$	Two-hidden-layer Feed-forward Network
UTC	Coordinated Universal Time
WD	Wind Direction
Wp	Watt-Peak
WRF	Weather Research and Forecasting
WS	Wind Speed
XGB	Extreme Gradient Boosting

1 Introduction

The need for electricity is globally increasing. With global warming in mind, clean renewable energy is a good substitute for traditional fossil fuels. These renewable energy sources are already integrated into the electrical power grid and the integration is rapidly increasing. Because of the highly non-schedulable and randomness of renewable energy sources, demand for intelligent dispatch of power has been prominently increasing. Accurate advanced photovoltaic (PV) power forecasting is a necessary tool to reduce the randomness of solar power, which can optimize the unit commitment and load dispatch, and improve the stability and economic efficiency of both the power grid and standalone power systems [1]. Accurate forecasting of PV power can also be used to better coordinate PV power with energy storage, where during daylight and non-peak hours the PV arrays can supply power to charge the batteries used for energy storage so that during peak hours both PV power and power from energy storage can be utilized.

There are two main approaches to forecasting PV power production, being deterministic forecasting and probabilistic forecasting. The purpose of deterministic forecasts is to accurately determine at time t the power production at time $t + t_h$. This approach ignores information that is highly valuable for utility managers, like the upper and lower bounds of predictions or the percentage of confidence for each value. Probabilistic forecasting addresses this problem by being able to provide a broader knowledge of the predictions, as several plausible values are determined, as well as the probability associated with each of them [1]. They provide probability density functions (PDFs) from which probabilities of future outcomes can be estimated. Probabilistic forecasts also provide information about uncertainty in addition to the commonly provided single-valued (best-estimate) power prediction.

A detailed literature review will now be presented, with details on chosen models, inputs, and prediction results. The thesis description and research questions will be presented afterward, in section 1.2.

1.1 Literature review and state of the art

Various models and methods have appeared in state of the art for solar PV production forecasting. These have both been deterministic and probabilistic forecasting of solar power in different forecasting horizons. The deterministic models have mostly been based on artificial neural networks (ANNs), support vector regression (SVR), and auto-regressive models. Antonanzas et al. did review PV forecasting techniques and found that 24% of the studies used ANN, 18% used SVR and 14% used regressive models like Auto-Regressive Moving Average (ARMA), Auto-Regressive Integrated Moving Average (ARIMA) and Non-linear ARMA with exogenous inputs (NARMAX) [1]. Recently, probabilistic forecasts have mainly been based on prediction interval (PI) approaches including quantile regression [2]–[4]. The most recent deterministic approaches found was based on different

ANNs [5]–[16] and SVR [12], [17]–[19]. Along with these mostly used methods, other methods that also has been used include bayesian ridge regressor (BRR), linear regressor (LR), gaussian process regressor (GPR), gradient boosting regressor (GBR), random forest regressor (RFR), lasso regressor (LAR), ridge regressor (RR), extreme learning machine (ELM), k-nearest neighbor (kNN), extremely randomized trees (ET), deep belief network (DBN), mondrian forest (MF) and different artificial neural networks like multi-layer perceptron (MLP), convolutional neural network (CNN), gated recurrent unit (GRU), bidirectional long short-term memory (BiLSTM), and long short-term memory (LSTM). As these are not among the most recently studied models, theory on them will not be included. They are mentioned for comparison to establish a background for the chosen models in this thesis.

Wang et al. used solar irradiance, wind speed, ambient temperature, and relative humidity as the input weather variables [12]. They used nine machine learning models, which were BRR, LR, GPR, multi-layer perceptron regressor (MLPR), SVR, GBR, RFR, LAR, and RR. In general, they found that the LAR, RFR, GBR, and SVR models showed better performances than the other models. During the periods when the weather was more unstable, the SVR model performed better than the other models. For August 2019, the models with weather type classification got RMSE [W] values in the range of 87.30 and 115.61, where the LAR model got 97.30 and the RFR model got 115.61. Without weather type classification, the RMSE [W] values for August 2019 were in the range of 103.58 from the SVR model and 106.95 from the GBR model. Overall, without time interval, the Mean Relative Error (MRE) ranged from 8.03% from the LAR model and 12.32% from the GBR model.

Massaoudi et al. made an ensemble (combination of two or more independent models with a voted outcome) of ELM, kNN, ET, DBN, MF, and the enhanced DBN (EDBN) [14]. These models were then compared to MLP, GRU, BiLSTM, and LSTM. In their models, their inputs were global horizontal irradiance, diffuse horizontal irradiance, relative humidity, wind direction, sampling time, temperature, and historical power data of PV arrays [kW] from March 1, 2016, to December 1, 2019, in sampling intervals of five minutes. The overall RMSE [kW] for their MF, ELM, ET, kNN, DBN, and EDBN was 27.37, 25.43, 14.45, 16.13, 8.57, and 3.88, respectively.

A hybrid (combination of two or more models that work together to predict an outcome) CNN-LSTM model was made for short-term photovoltaic power forecasting by Zhang et al. The hybrid model was then compared to an MLP- and an LSTM model. The model was used on twenty-one different PV facilities in Germany. The facilities' installed nominal power ranges between 100 kW and 8500 kW. The data set they used includes historical NWP data (solar radiation, sun position, wind speed and direction, relative humidity, temperature, and cloud cover) and the produced PV power in a 3-h resolution for 990 days. For all PV facilities, the average RMSE of the models' predictions is 0.0778 for the MLP model, 0.0714 for the LSTM model, and 0.0689 for the CNN-LSTM model [5]. These RMSE values are very low, and the paper does not specify the unit of the RMSE, but it might be normalized values. Since the nominal power of the PV facilities is in kW, it is assumed that RMSE is in kW.

In less recent years, more models based on SVR, and auto-regressive models were made to predict PV power production as well as ANN models.

An ANN model for PV energy forecasting was proposed by Leva et al. [20]. Their input variables were weather forecast, power and irradiance measurements, and historical data sets. The accuracy of the model was evaluated on a reduced number of hourly samples. The chosen days are sunny days with sunny weather forecasts, partially cloudy days with variable weather forecasts, and cloudy days.

The normalized root-mean-square error nRMSE% (based on the maximum observed power output) is 12.5, 24, and 36.9, respectively, while the normalized mean absolute error nMAE%, based on the rated power of the system is 5.19, 13.2 and 11, respectively. Their errors, as stated by Leva et al., is highly related to the solar irradiance forecasting accuracy. The last case represents one of the worst cases over the entire data set analyzed in the paper. In all the considered cases, they noticed that the most relevant errors occur during sunrise and sunset; therefore, a possible enhancement to their method can be performed by improving the way sunset and sunrise are considered, for instance by adopting hybrid methods.

Mellit et al. developed three distinct artificial neural networks (ANN), to be applied to three typical types of days (sunny, partly cloudy, and overcast). The first model is applied to a sunny day, the second to partly cloudy, and the third to overcast. The proposed ANN models accept as input the future values of in-plane solar irradiance, solar cell temperature, and the present value of the produced power. For sunny, partly cloudy, and overcast days the RMSE [kW] is 0.087, 0.1, and 0.054 kW, respectively. This RMSE seems quite low, as a PV plant of 500 kW_p was used. The nRMSE based on the power of the PV plant would then be 0.0174%, 0.02% and 0.0108%, respectively [21].

Fernandez-Jimenez et al. built and evaluated a set of forecasting models: ARIMA models, k-nearest neighbor (kNN) models, ANN-based models, and Adaptive Neuro-Fuzzy Inference System (ANFIS) models. The inputs used for developing the models were past values of hourly energy production in the PV plant, as well as forecasted values of weather variables obtained with the two first modules of the proposed system. A multilayer perceptron ANN-based model was revealed as the best forecasting model among those evaluated with an nRMSE (RMSE/rated power) of 11.79%. Their two ARIMA models performed with nRMSE of 21.14 % and 17.36% [22].

Forecasting models based on seasonal ARIMA (SARIMA) time-series analysis with and without an exogenous factor (here, solar radiation), and two ANN models were made in [23]. These models were compared to each other, and to a persistence (benchmark) model. For the ANN models, radiation forecasts and historical power was used as inputs. Their models performed with a yearly average nRMSE (normalized to the PV installed capacity) of 13.71% for the persistence, 12.89% for the SARIMA (3,1,2) × (3,1,2)24, 11.12% for the SARIMA (3,1,2) × (3,1,2)24 with exogenous factor, 11.42% for the ANN - Model A and 11.26% for ANN - Model B. Both ANN models perform with a lower nRMSE than the others when forecasting winter and spring even though their yearly average nRMSE is higher than the SARIMA with an exogenous factor.

De Felice et al. aimed to make a short-term forecast of PV power production based on SVR. Their data consisted of historical power production and weather forecasts of solar radiation and temperature provided by the European Centre for Medium-Range Weather Forecasting (ECMWF) Integrated Forecasting System (IFS). The predicted solar power production in Italy obtained by the SVR model was found to be more accurate during summer than in the rest of the year: the percentage error is below the 5% when they used observed meteorological data as predictors and below the 12% when they use forecasted predictors on the entire prediction range. The normalized RMSE (i.e. RMSE divided by the maximum PV plant power output) is below 0.08 and 0.18 respectively [24].

Rana et al. applied an ensemble of ANNs and an SVR algorithm for a very short-term PV power forecast using both univariate and multivariate models. The univariate models use only the previous PV power data, while the multivariate models use in addition previous meteorological data (solar irradiance, temperature, humidity, and wind speed). For the multivariate models they got mean absolute errors (MAEs) in the range 48.26 kW - 127.55 kW for the SVR model and 45.11 kW - 110.27 kW for the ANN ensemble, while the MRE (MAE normalized by the range of the target values) is in the range 4.20% to 11.09% for the SVR model and in range of 3.92% to 9.59% for the ANN ensemble. The first values in the range correspond to prediction horizon 1 in step and 5 minutes in time and the last corresponds to prediction horizon 12 in step and 60 minutes in time. For the smaller forecasting horizons, the performance of the ANN ensemble and SVR was similar, but for the larger horizons, the ANN ensemble outperformed SVR for both univariate and multivariate predictions [25].

Different machine learning algorithms, including Linear Regression, Polynomial Regression, Decision Tree Regression, SVR, Random Forest Regression, Long Short-Term Memory (LSTM), and Multilayer Perceptron (MLP) Regression was used by Mahmud et al., to predict short-term, medium-term, and long-term PV power generation. Various weather parameters including temperature, relative humidity, global horizontal radiation, diffuse horizontal radiation, and daily precipitation were used to train their models and predict the PV power output. They only listed their model's performance for long-term forecasts considering one-year data points. For SVR, MLP and LSTM the MAE is 0.0157, 0.1492 and 0.0447, respectively [26].

Some of the above papers have not used relative error metrics to evaluate their models, making it hard to assess whether the errors are large or not. If the rated power of the arrays is included, it is possible to assess the magnitude. The RMSE or MAE values the models got in the above papers will be divided by the rated power if specified and compared to the models used in this thesis in the comparison section 4.8.

The above-presented papers, all proposed deterministic (point) forecasts, and to improve those forecasts into probabilistic forecasts, interval prediction including quantile regression (QR), analog ensemble (AnEn), and kernel density estimation (KDE) methods can be implemented. To evaluate probabilistic forecasts prediction interval coverage probability (PICP) and prediction interval normalized average width (PINAW) are often measured. More information and theory on them are covered in the theory subsection 2.8.2. Chen et al. used an improved nonlinear autoregressive network with exogenous inputs (NARX) and multivariate KDE to interval predict PV power. The input features are meteorological factors like global horizontal irradiation (GHI), wind speed, ambient temperature, and previous PV power output. PICP and PINAW were calculated for four periods, autumn, winter, spring, and summer. For a 95% confidence interval the PICPs (%) are 95.6, 98.69, 94.91 and 97.15, respectively and the PINAWs(%) are 12.27, 6.54, 14.44 and 8.74, respectively [2]. Their proposed method has a narrow bandwidth, high coverage, and a close distance between the middle of PI and the actual value. It supplies a new way to realize the interval prediction, which is helpful for system reliability assessment of PV power plants and dispatch of the smart grid.

A short-term probabilistic photovoltaic power forecast based on a deep convolutional long short-term memory network and kernel density estimation was proposed by Bai et al. [27]. The prediction results gained by kernel density estimation were PICPs of 0.9789, 0.9590, 0.9556, and 0.9489, for 1-step, 2-step, 3-step, and 4-step, respectively. The fact the PICP approximates 95% means that the obtained 95% probabilistic confidence interval is reliable enough. In their deterministic model, the convolutional long short-term memory network is close to the actual values and the forecast errors are small, with MAREs (MAE divided by rated power) of 0.20%, 0.42%, 0.65%, and 0.89% for 1-step, 2-step, 3-step, and 4-step, respectively.

Cheng et al. proposed a new solar power probabilistic forecasting method based on the dynamic weighting method, kNN algorithm, and QRNN. Their inputs are historical power and numerical weather prediction (NWP) variables including relative humidity, total cloud cover, temperature, solar radiation, and total precipitation. They made five different QRNN models with a PICP ranging from 0.9231 to 0.9846. They also made a comparison of error of point prediction (RMSE) of the five different methods, being in the range of 0.1069 and 0.1314. The QRNN model has low quantile scoring and narrow prediction intervals, meaning the prediction interval is of high reliability The authors concluded that the effective probability prediction method can provide useful information for the energy storage control of a solar grid-connected power generation system [28].

1.2 Thesis description and research questions

Based on the models used in the literature and their results, this thesis will focus on PV power production forecasting, using an artificial neural network (ANN) model, a support vector regression (SVR) model, a quantile regression neural network (QRNN) model, and two ensemble models with inputs of measured PV power production and/or historical forecasted weather data (NWPs) from a forecasting service. All models will be compared to each other and to an auto-regressive integrated moving average (ARIMA) model with an exogenous variable. This ARIMA model will be used as a benchmark model. All models will be evaluated using root mean square error, normalized root mean square error, mean absolute error, and the coefficient of determination R^2. The QRNN model will also be evaluated by employing prediction interval coverage probability (PICP), prediction interval normalized average width (PINAW), and coverage width-based criterion (CWC).

The master's thesis topic intends to contribute toward the United Nations Sustainable Development Goals [29], the Paris Agreement [30], Norway's climate goals [31], as well as to contribute toward a more economically efficient, stable, reliable and uninterrupted power grid with optimized load dispatch, based on renewable energy sources. The mentioned intended contributions are the motivation behind the chosen topic. The topic is also intended to give food producer Gorines the opportunity to predict their system's hourly PV power generation, as they are providing the data with the wish of being able to better control their use of electrical appliances. Daily, the company tries to be as economically efficient and climate positive as possible. If they know how much power they will produce each hour, they would be able to control their electrical use even better. This thesis is a continuation of a previously done pre-project (research project) where the abstract of this report is included in Appendix A.1. The results from this thesis will be compared to the results from the pre-project. The following goals were established. Determine which NWP variables are relevant input variables to be used in the models, and find out the difference in using random test data (shuffled with a low statistical difference from the random train data and not any particular month, just a percentage of the full dataset) and a month as test data, find the optimized hyper-parameters including the number of hidden layers and neurons, optimizers, initializers, activation functions, regularization parameter C, tube radius ε, and kernel coefficient γ, and evaluate the performance of the models. As a lot of research has been done previously in the same field, this thesis is mostly an addition to the field of forecasting with comparisons of different models using the same data and comparisons with models used in literature. The thesis may fill gaps in the literature in terms of training on power data produced in very varying weather conditions, with quite a difference between months and seasons.

The research questions for this thesis are, "How will the models perform using random train data to predict August 2021 compared to a random test sample? Will the ensemble models perform better than the standalone models, and will the quantile regression neural network make accurate prediction intervals? How well will the predictions be if the ANN model only uses NWP data as inputs?".

Since there was no previous experience with artificial neural networks and statistical models, this may limit the full utilization of the possibilities artificial neural networks and statistical models have to offer. The process of learning how to build such models was based on reading literature on related previous work and some guidance from supervisors. For coding, trial and error were used together with searching for a fix of the errors.

The report is structured as follows. First, the theory is described in 2. Section 2.1 consist of theory about the process of preparing data and Section 2.2, 2.3, 2.4, 2.5 and 2.6 include theory on artificial neural networks, support vector regression, autoregressive models, ensemble models and interval prediction, respectively. Section 2.7 covers theory about hyper-parameter optimization and section 2.8 include theory about forecasting performances. Then, a descriptive method of the whole process of preparing, analyzing predicting, optimizing, and evaluating, is included in 3, before the results and discussions are described and made in section 4. Lastly, a conclusion was made in section 5. References used are also included at the end, followed by Appendix chapters with the script made in Spyder (Python 3.8) are also attached at the very end of the report.

2 Theory

This chapter contains theory on both deterministic and probabilistic forecasting, including several important steps most models should go through, as well as theory on the different models and how they may be built.

A photovoltaic power plant may be seen as a box containing several inputs, for example, solar radiation, temperature, precipitation, and wind speed, and one output, the alternating current (AC) flowing to the power grid [32], [33]. Forecasting can be done by taking historical values (measured and/or forecasted) as inputs to the forecasting model and using this history to predict the future. The most important input when considering a time horizon (future) up to two hours ahead, is the available observations of PV power, while numerical weather predictions (weather forecast) are the most important input for longer time horizons [34], [35].

The forecast of PV production is traditionally deterministic. However, such a method does not necessarily provide all the necessary information such as the forecast error margins, and the confidence one can have in the forecast [36]. Deterministic forecasting, also called point forecasting, forecasts the power for the next minutes, hours, or days (time horizon) with a single value of the PV production in each predicted horizon. Forecasting methods depend on the tools and information available like the data from weather stations and satellites, PV system data, and outputs from numerical weather predictions (NWPs). These methods can be classified into three different categories, methods that use exogenous input data, usually based on the NWP model, methods that only use non-exogenous input data, usually based on on-site past values of different atmospheric parameters and/or the power output of the PV system, and methods combining the former two methods [21], [37].

One drawback of known forecasting methods is that they do not take into consideration the degradation effect of the photovoltaic modules due to delamination, hot spots, dust accumulation, soiling effect, light-induced effects, mismatch, and anti-reflection coating degradation. Hence, they do not work as well in long-term prediction. In this case, periodically training should be carried out, using new data [21]. Deterministic forecasting can be categorized into four categories, persistence, statistical, machine-learning, physical, and hybrid method with subcategories [38]–[40].

Different from deterministic predictions is probabilistic forecasts which can acquire more information through constructed prediction intervals (PIs) or probability density functions. Probabilistic interval prediction uses a predictive model to acquire a range of fluctuations at a certain confidence level, which consists of the lower and upper bounds of the interval. Therefore, performing probabilistic interval prediction to analyze their uncertainties has become a major concern in the field of power prediction.

Compared with probability interval prediction, probability density prediction acquires more useful information by estimating the occurrence probability of future outcomes, including a comprehensive evaluation of the uncertainty associated with PV power. This method gives the most detailed information obtained in the field of power prediction and currently has the highest practical application value. It can not only achieve deterministic prediction results in addition to probabilistic prediction results for PV power at any time in the future, but it can also provide a complete probability distribution figure with predictive values [41].

Previous research has primarily been focused on deterministic forecasting. Although it is not clear why this is the case, Hong and Fan suggested that the reason might be the fact that probabilistic forecasts were assessed with the same performance metrics as those used to assess deterministic forecasts, and subsequently performed worse than their deterministic counterparts. Assessing probabilistic forecasts with these metrics could lead to invalid conclusions, as they are quite different. Reliability and sharpness are important properties of a probabilistic forecast, therefore other metrics like prediction interval coverage probability (PICP) and prediction interval normalized average width (PINAW) have been used [42], [43].

Figure 2.1 shows the difference between probabilistic interval forecasting and deterministic forecasting. The confidence forecast refers to the probabilistic forecast [44].

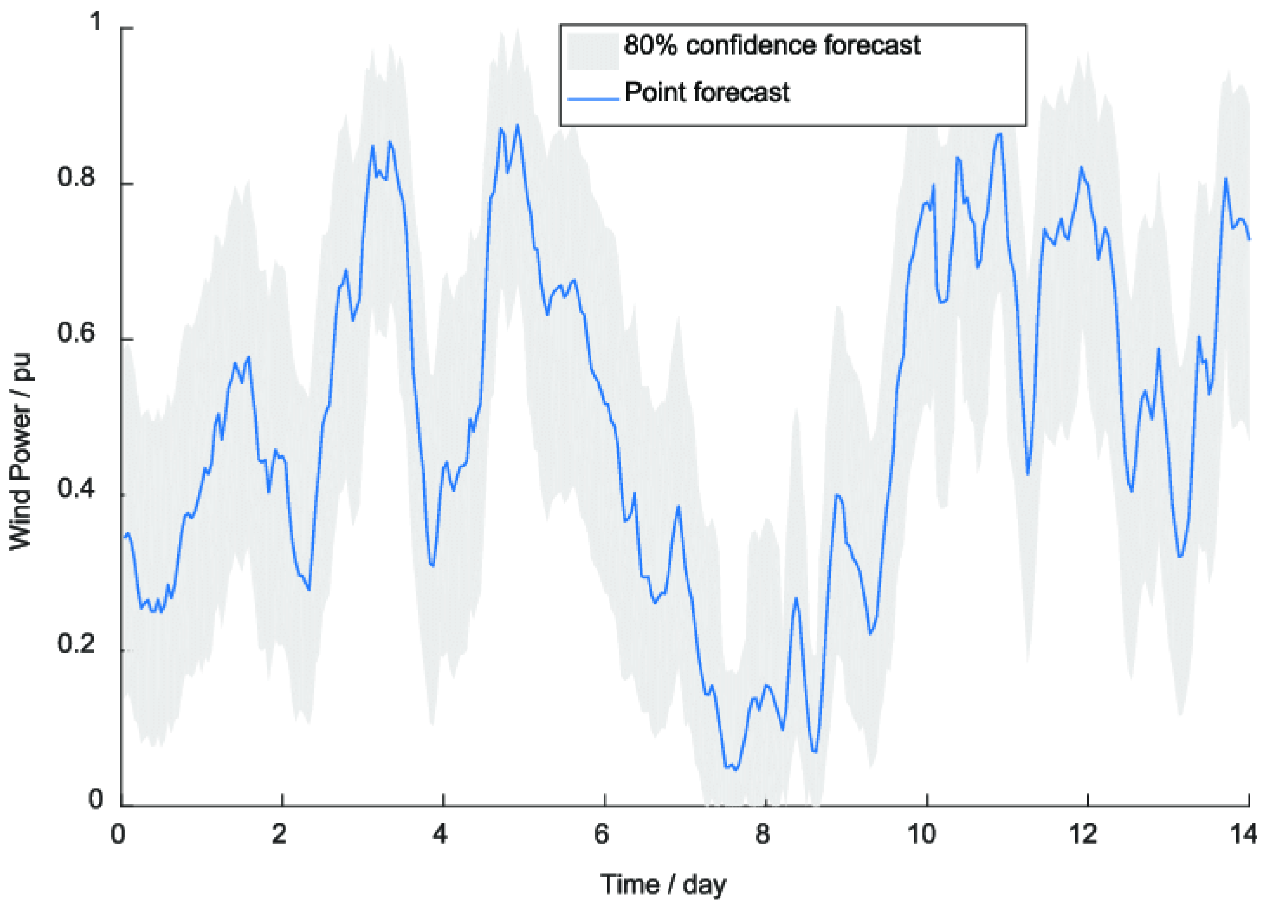


Figure 2.1: A probabilistic prediction interval (confidence) forecast and deterministic (point) forecast of wind power.

2.1 Data analysis, Processing, Input Selection and Data Division

When preparing the data for most of the models, several steps should be taken. These include data analysis with data visualization to understand the data, assessing the amount of missing data, input selection, feature selection and reduction techniques like finding the correlation between features and doing principal component analysis, division of data into train and test, scaling of data to avoid domination of features in magnitude, fitting and transforming of data to establish the mean and variance of the feature. These steps will be explained in this section.

2.1.1 Input selection

The solar irradiation and temperature at each time step are key components that determine the power generation of photovoltaic panels. Other parameters that may influence the power output is wind speed, wind direction, cloud cover and precipitation [38], [39], [45]. In addition, lagged past values of production and relevant variables can tell us the trend the power output follows, and are therefore important when predicting the future. To include lagged past values into a model for predicting the power, all data can be time-shifted as much as wanted. The lagged past value should be further analyzed so that one does not use wrong and uncorrelated information, resulting in bad predictions.

Numerical weather prediction (NWP) models are used and developed for weather forecasting purposes and as inputs in power prediction models. In a very simple way, they are supplied with initial conditions and then, the differential equations describing the evolution of the atmosphere are solved [46]. Numerical weather predictions are often based on an ensemble of different models and give information on future variables such as irradiation, wind speed, wind direction, air temperature, amount of cloud cover, and precipitation.

2.1.2 Correlation

To measure the association between two features and the direction of their relationship, a bivariate analysis called correlation analysis can be used. The strength of association, the correlation coefficient varies between +1 and -1. ± 1 indicates a perfect degree of association between the two features. The closer to zero, the weaker the correlation. The minus sign indicates a negative relationship, and a positive relationship is indicated by the plus sign. The types of correlation often measured are Pearson correlation, Kendall rank correlation, and Spearman correlation [47]. Strongly correlated variables should be used as input vectors to the forecasting model, and weakly correlated variables should be refused. The accuracy of the forecasting model may be improved by using many input vectors but may increase computational cost and complexity. Therefore, constructing a forecasting model with an optimal number of input vectors, based on correlation is important. The pre-processing of input variables may significantly reduce improper training problems, caused by non-stationary data, which is due to the change in weather conditions and missing input data points in the historical data, which is due to recording errors or other unexpected events. Therefore, the accuracy of the forecasting model can be considerably improved by the pre-processing of the input data [38]. A common practice is to assume that if the correlation coefficient is larger than ± 0.5, it means the variables are correlated, and if it is less than ± 0.5, then it means too low a correlation to be considered as input to the models [48].

2.1.3 Missing Data

Missing data may be a big problem if the amount is too large. Traditionally, the missing values are simply omitted or replaced through imputation methods. However, omitting those missing values may cause temporal discontinuity. Imputation methods, on the other hand, may alter the original time series [49]. This could result in bad prediction performance. If the missing data is less than 1% of the total data, the effect is insignificant. Rates among 1% - 5% refer to manageable missing data, while for amounts larger than 5%, processing tools should be employed [50]–[52].

2.1.4 Data Division

Strongly correlated features are divided into train and test data. The training data are used for learning from data, whereas testing data are used to evaluate the forecasting model [38].Further both the train- and test data is divided into features data (x_{train} and x_{test}) containing input variables and target data (y_{train} and y_{test}) containing one or more output variables. The next steps include scaling, transforming, and fitting the data. Scaling ensures input features have a similar range and thereby avoiding the domination of features in magnitude. Fitting and transformation are used to establish the mean and variance of the features of the training set and to transform all the features using the respective mean and variance. To keep the test data "unseen" and a surprise, the test data only gets transformed, using the same mean and variance as is calculated from the training data. This is the standard procedure to scale the data while building a machine learning model, so that our model is not biassed towards a particular feature of the data set, and at the same time prevents our model from learning the features/values/trends of the test data [53]. The training data gets further split into train and validation data during k-fold cross-validation. A validation set is needed to provide an unbiased assessment of a model's fit on the training dataset and the testing set is needed to provide an impartial assessment of a final model's fit to the training dataset [54]. Please refer to the pre-project (research project) for a more thorough description of these steps [55].

2.1.5 Principal Component Analysis

Sometimes reducing the number of features can help decrease the complexity of the study. A method that may do this, while not affecting the output of the forecasting models is Principal Component Analysis (PCA). This unsupervised method can reduce redundant variable information and compute smaller numbers of uncorrelated variables which contain the original variable information as much as possible [56]. It does not use the output information, and the variance is to be maximized. The proportion of variance needed for optimal feature space may vary. In [57] they chose to have as many variables after performing PCA as needed while keeping 99% of the variance. Figure 2.2 shows the principal components of a two-dimensional dataset.

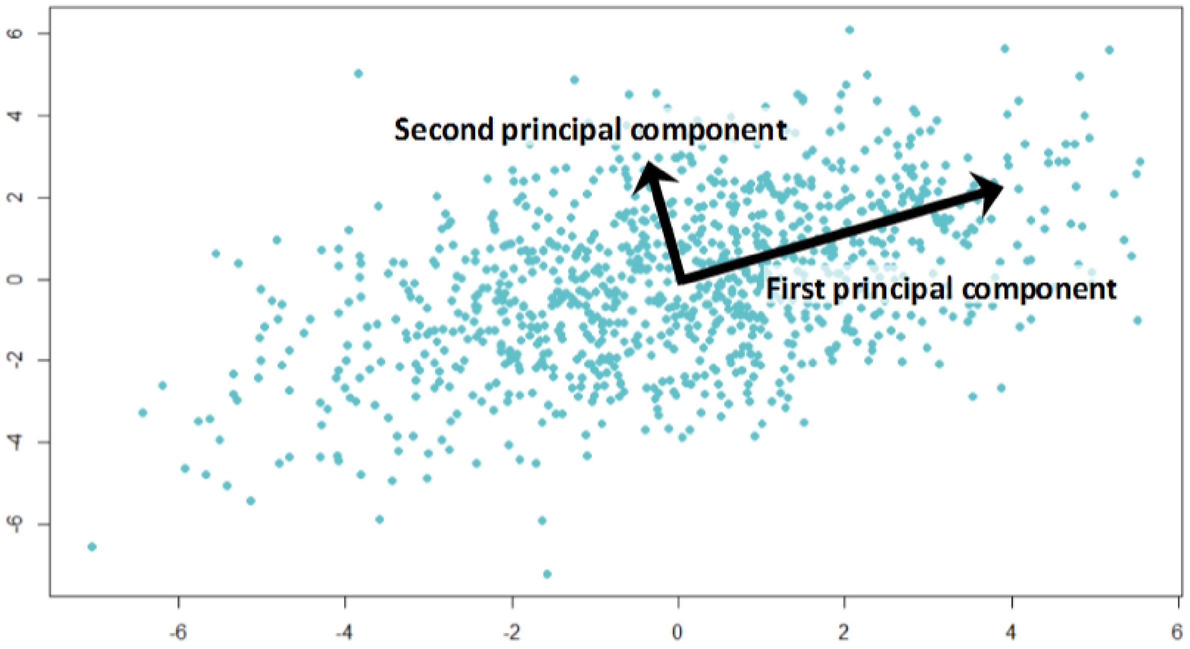


Figure 2.2: The principal components of a two-dimensional dataset

2.2 Artificial Neural Network

Artificial neural networks (ANNs) are highly capable of pattern classification and pattern recognition. Inspired by research into the human brain, ANNs can learn from and generalize from experience by capturing subtle functional relationships among the data even if the underlying relationships are unknown or difficult to describe. Thus ANNs are well suited for problems whose solutions require knowledge that is challenging to specify, but for where sufficient data or observations are available [58]. One disadvantage of ANNs is that there is no specific rule for determining their structure. The appropriate network structure is achieved through experience and trial and error [59]. Another disadvantage of ANNs is that they usually require much more data than traditional machine learning algorithms, as in at least thousands if not millions of labeled samples. This is not an easy problem to deal with and many machine learning problems can be solved well with less data if you use other algorithms. Nevertheless, there are some cases where ANNs do well with little data, but most of the time they do not [60].

After learning the presented data, ANNs can often correctly assume the unseen part, even if the sample data contain noisy information. As forecasting is performed via the prediction of future behavior (the unseen part) from examples of past behavior, it is an ideal application area for neural networks, at least in principle. Since ANNs are not linear they are suitable for forecasting PV power as the amount of irradiation measured during the day is non-linear [58]. The basic structure of ANNs is divided into three sections: input layer, hidden layer, and output layer, comprising artificial neurons and weighted connections. The input layer receives input information, the hidden layer or layers, analyzes the input information, and the output layer receives the analyzed results and provides the output [38], [40]. Types of artificial neural networks include multilayer feedforward (MLFF) network with backpropagation (BP) learning, also called multilayer perceptron (MLP), recurrent neural network (RNN), and probabilistic neural network (PNN).

Several steps can be followed in the development process of ANN models, depending on the available data and the desired outcome. Figure 2.3 shows possible steps in the process of developing an ANN model.

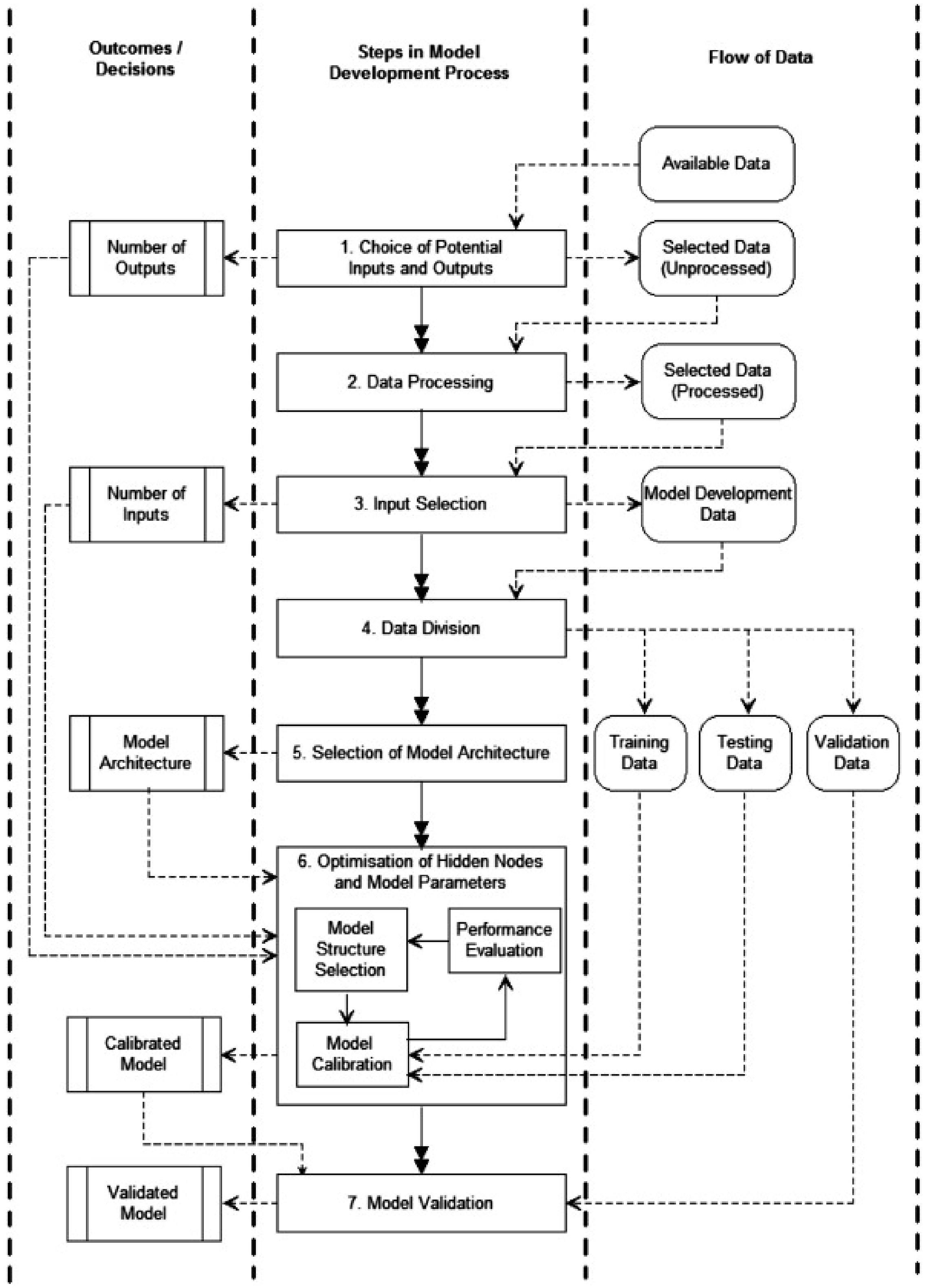


Figure 2.3: Steps in ANN Model Development Process [61].

2.2.1 Multilayer Perceptron

Multiplayer perceptron networks are the most popular and widely-used neural network method, where the data flows in the forward direction from the input to the output layer. The number of hidden layers can be adjusted in accordance with the complexity of the problem. ANNs with two or more hidden layers are called deep networks because the network has become complex with more than one hidden layer. A hidden layer is not seen directly, either on the input side or the output side. MLPs are used in nonlinear modeling and complex problems which cannot be solved by an ordinary single-layer neural network. The backpropagation (BP) algorithm is the most used method for training feedforward ANNs, which is dependent on the gradient descent optimization technique. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and make the model dependable by increasing its generalization. BP in neural networks is a short form for "backward propagation of errors." Figure 2.4 shows the inner structure of a neuron and Figure 2.5 shows the basic structure of MLP using BP training with one input layer, two hidden layers, and an output layer [62]–[65].

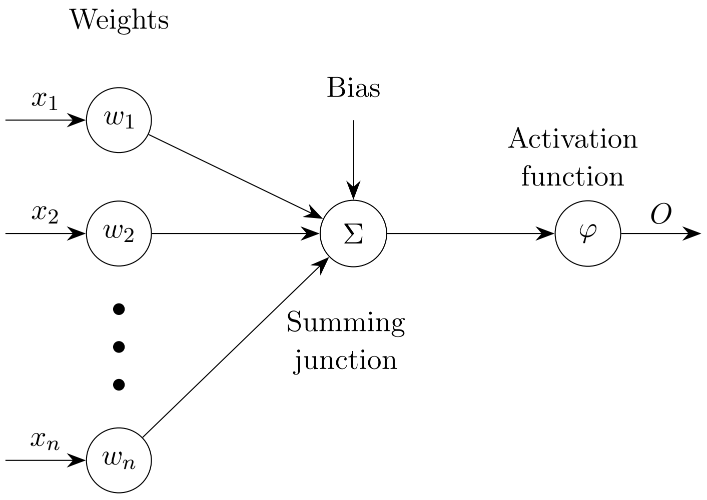


Figure 2.4: The inner structure of a neuron. Inputs are multiplied by weights, and biases are added before passing through an activation function. Outputs are produced, and if it is not an output neuron, it becomes a hidden neuron where the same happens.

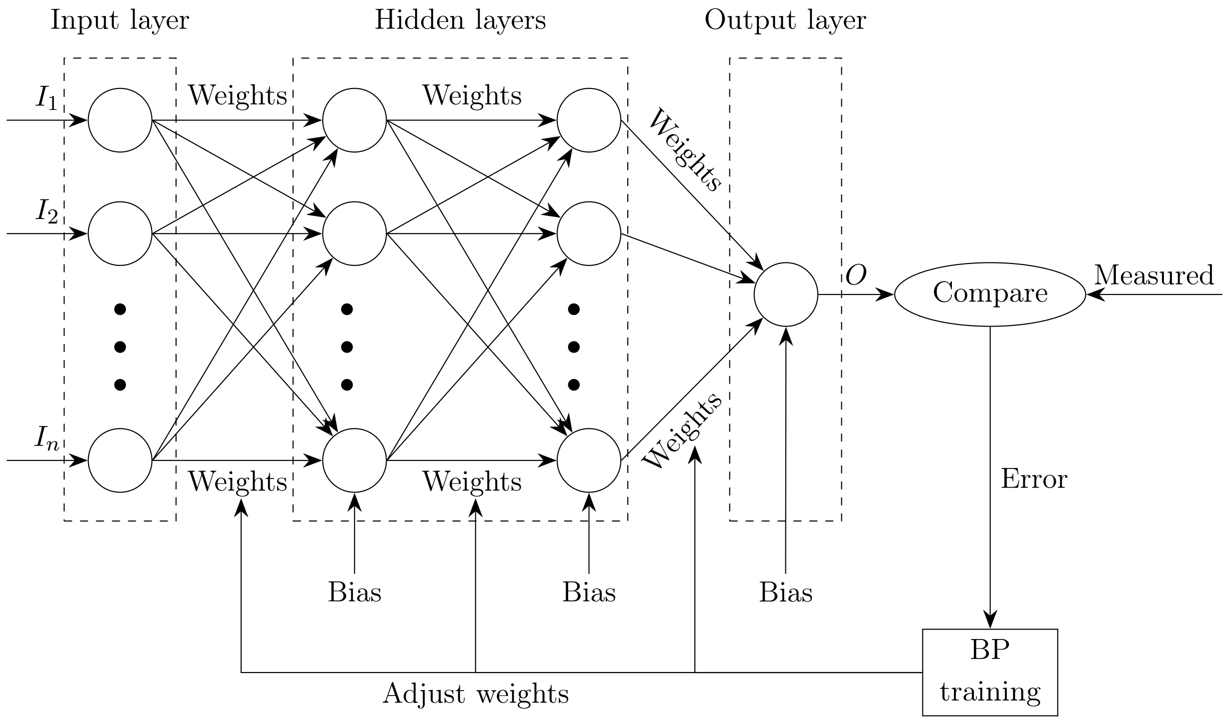


Figure 2.5: MLP network with two hidden layers and BP training. All neurons is connected to another and the weights get adjusted after each epoch

As shown in Figure 2.4, each neuron performs two functions: collecting inputs and producing an output. Each input is multiplied by connection weights, and its products and biases are added before being passed through an activation function to produce an output [38], [40], [58], [66]. These variables can be defined by

$$O = f(b_j + \sum_{i=1}^{N} w_{i,j} x_i), \tag{2.1}$$

where O is the output, f is the activation function, b is the bias, j is a certain neuron, N is the number of input values, w is the weighted coefficient, and x is the input value [67].

The performance and accuracy of the model depend on the input dataset, the number of neurons in the hidden layer or layers, the number of hidden layers, the learning algorithm, batch size and activation functions, weight initializers, and optimizer. The batch size is the number of samples that are passed to the network at once. The main drawbacks of ANN are that they have a tendency of overfitting and that they require a large amount of data during the training process. ANN forecasting techniques have undergone many modifications to accommodate disparate input-output projections and architectures [40], [68].

Hidden layers and neurons

The correlation between the number of artificial neurons and hidden layers is essential. The number of neurons in the output layer is determined by the number of output variables. However, the number of neurons for the input and hidden layers is not predetermined, but a variable parameter to be optimized. The number of hidden layers is also not predetermined. Thomas et al. investigated whether feedforward neural networks with two hidden layers generalize better than those with one. They found that in nine out of ten cases two-hidden-layer feedforward networks (TLFNs) outperformed single-hidden-layer ones, but that the amount of improvement was very case dependent [69]. Azka et al. [70] forecasted PV output power using ANN with both two hidden layers and one hidden layer, and found that in their case, the two hidden layers' neural network has a higher accuracy value. Both cases have the same number of neurons, but on a neural network, two hidden layers can have more parameters. According to them, this happens because one-time training of data in two hidden layers is the same as dozens of times training in one hidden layer. The number of hidden layers needed will not likely exceed two, and for most cases one is sufficient. When in doubt, one can begin with one hidden layer and experiment by adding or pruning layers or neurons to find the best compromise. Generally, the net should be kept as simple as possible to help ensure training time within reasonable bounds and good learnability and generalization [71].

Several papers tested and optimized different numbers of hidden neurons in the learning step of the network, according to a specified criterion such as an RMSE value [72]–[74]. According to Kermanshahi in [74], the selection of the number of hidden neurons is an art rather than mathematics. When the number of hidden neurons is small, the correlation between input and output cannot be well studied and the errors increase, resulting in underfitting. Moreover, when the number of hidden neurons is more than necessary, the error grows more. This may result in overfitting, and the amount of training time can increase to the point that it is impossible to adequately train the neural network. Therefore, a compromise must be reached between too few and too many neurons in the hidden layers [75]. A method for obtaining the number of hidden neurons is trial-and-error, which both Ding et al. and Kermanshahi used [73], [74]. Three methods that can serve as a starting point before implementing trial-and-error are first, the number of hidden neurons should be between the size of the input layer and the size of the output layer. Secondly, the number of hidden neurons should be two-thirds the size of the input layer, plus the size of the output layer. Third and lastly, the number of hidden neurons should be less than twice the size of the input layer [75].

Layer Activation functions and Weight Initializers

Each neuron can be initialized with specific weights. This can be a statistical distribution or a function. Activation Functions are specially used in artificial neural networks to transform an input signal into an output signal, which in turn is fed as input to the next layer in the stack [76]. There is no mention in any literature of the type of activation function to be preferred, but the most frequently used activation functions in ANNs are sigmoid, hyperbolic tangent sigmoid, rectified linear unit (ReLU), softmax, gaussian radial basis, linear, unipolar step function, bipolar step function, unipolar linear function, and the bipolar linear function [8], [38], [76]. Other activation functions include softplus, softsign, hyperbolic tangent, scaled exponential linear unit (SELU), exponential linear unit (ELU) and exponential [77].

A neural network works just like a linear regression model where the predicted output is the same as the provided input if an activation function is not defined. Similar is the case if a linear activation function is used where the output is similar to the input fed along with some error. The choice of activation function is context-dependent, it depends on the task that is to be accomplished. Different activation functions have both advantages and disadvantages of their own. For classification problems, a combination of sigmoid functions gives better results. The rectified linear unit (ReLU) function is the most widely used function and performs better than other activation functions in most cases, however, the ReLU function must only be used in the hidden layers and not in the outer layer. Studies have shown that both sigmoid and hyperbolic tangent functions are not suitable for hidden layers because the slope of function becomes very small as the input becomes very large or very small, which in turn slows down gradient descent. ReLu is therefore the most preferred choice for tasks with hidden layers as the derivative of ReLU is 1 [76]. Elu is a variant of RELU that modifies the slope of the negative part of the function.

Proper initialization of the weights in a neural network is critical to its convergence and to ensuring a model with high accuracy. If the weights are not correctly initialized, it may give rise to the vanishing gradient problem or the exploding gradient problem. When the ReLu activation function is used the He initialization is most suitable, either normal or uniform. ELU has been effective in reducing the vanishing gradient problem [78], and for this activation function, LeCun initialization is preferred [79]. Uniform means the weights are assigned from values of a uniform distribution and normal means that the weights are assigned from values of a normal distribution [80], [81]. SELU is a newer activation function that induces self-normalizing properties like variance stabilization which in turn avoids exploding and vanishing gradients [82]. LeCun initialization is also preferred with SELU [79].

The vanishing gradients problem is one example of unstable behavior that one may encounter when training a neural network. It describes the situation where a multilayer feed-forward network or a recurrent neural network is unable to propagate useful gradient information from the output end of the model back to the layers near the input end of the model. The result is the general inability of models with many layers to learn on a given dataset, or for models with many layers to prematurely converge to a poor solution [83].

Exploding gradients is a problem when large error gradients accumulate and result in large updates to neural network model weights during training. Gradients are used during training to update the network weights, but typically this process works best when these updates are small and controlled. When the magnitudes of the gradients accumulate, an unstable network is likely to occur, which can cause poor prediction results [84].

To prevent these two problems, some steps can be done. These include using the ReLu, ELU, or SELU activation function or the Long-Short Term Memory (LSTM) architecture, Gradient Clipping, and weight regularization [83]–[85].

Loss function and optimizer

Most deep learning algorithms involve optimization of some sort. Optimization refers to the task of either minimizing or maximizing some function $f(x)$ by altering x. The function one wants to minimize or maximize is called the objective function or criterion. When it gets minimized, it is called the cost function, loss function, or error function [86]. From the loss function, the gradients can be derived, which are used to update the weights. The average of all losses constitutes the cost. The loss function is required by the learning algorithm to decide what steps it should take to minimize the loss. While the loss function calculates the error for a single data point (sample), the cost function calculates the loss for the entire dataset [87]. The choice of the loss function is directly related to the activation function used in the output layer of the neural network. If the network is dealing with a regression problem where one predicts a real-value quantity the output layer configuration should be one node with a linear activation unit, and the loss function should be Mean Squared Error (MSE) [88]. Other literature also supports the use of MSE as the loss function [5], [89]–[93]

Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate to reduce the losses. How to go about changing the weights or learning rates of the neural network to reduce the losses is defined by the optimizers one uses. Optimization algorithms or strategies are responsible for reducing the losses and providing the most accurate results possible [94]. In gradient-based BP training algorithms, it is easy to get trapped by local minima and therefore deteriorate the performance of ANNs. Examples of optimizers are Stochastic gradient descent with or without momentum, adaptive gradient algorithm, root mean square propagation, and adaptive moment estimation.

Stochastic gradient descent (SGD) is a variant of gradient descent that tries to update the model's parameters more frequently. In this, the model parameters are altered after the computation of loss on each training example. So, if the dataset contains 1000 rows SGD will update the model parameters 1000 times in one cycle of the dataset instead of one time as in gradient descent. Some disadvantages with this optimizer are the high variance in model parameters, that it may shoot even after achieving global minima and to get the same convergence as gradient descent it needs to slowly reduce the value of learning rate [94].

The adaptive gradient algorithm (Adagrad) is an adaptive learning rate method. It performs larger updates for infrequent parameters and smaller updates for frequent parameters. It is well suited to sparse data as in large-scale neural networks and uses different learning rates for every parameter for every time step. A drawback of this method is that it is computationally expensive and the learning rate is always decreasing resulting in slow training [94], [95].

Root mean square propagation (RMSProp) tries to resolve Adagrad's radically diminishing learning rates by using a moving average of the squared gradient. It utilizes the magnitude of the recent gradient descents to normalize the gradient. In RMSProp learning rate gets adjusted automatically and it chooses a different learning rate for each parameter. It divides the learning rate by the average of the exponential decay of squared gradients.

Adaptive moment estimation (Adam) combines both SGD with momentum to resolve local minima problems and RMSProp, which uses the sum of the square of previous gradients to resolve the same learning rate issue [96]. Adam implements the exponential moving average of the gradients to scale the learning rate instead of a simple average as in Adagrad. It keeps an exponentially decaying average of past gradients. It is computationally efficient and has little memory requirement. This optimizer is one of the most popular gradient descent optimization algorithms [95].

Early Stopping and Overtraining

In MLP models bad generalization might become an issue. This happens when the neural network learns too many input-output examples, and it ends up memorizing the training data. It may do so by memorizing noise or coincidences in the training data, and then predict badly on new data [65]. When the number of hidden neurons is large, the generalization accuracy deteriorates, and the bias/variance dilemma holds (too high number of free parameters). If training continues too long, meaning, as more training epochs (iterations) are made, the error on the training set decreases but the error on the validation set starts to increase beyond a certain point [97]. To overcome the problem of bad generalization and overtraining, learning should be stopped early. Early Stopping monitors the performance of the model for every epoch on a held-out validation set during the training and terminates the training conditional on the validation performance [98]. This means the training stops when the error stops improving.

2.3 Support Vector Machines

A support vector machine (SVM) is a supervised machine learning method based on the structural risk minimization principle. The method minimizes the upper bound of the expected risk. Therefore, it can minimize the error of the training data. Given a training data sample, SVM constructs a hyper-plane as the decision space in such a way that the margin of separation between positive and negative examples is maximized [65]. Support vector regression (SVR) is the application of SVM in time series regression. The input time series data is mapped into a higher dimensional feature space by nonlinear mapping, and then linear regression is performed in that space [38]. The method work as a multiple linear regression using transformed predictors (inputs), while keeping low complexity and a decent fitting of data [1]. The SVR model can prevent overfitting, dismiss iterative tuning of model parameters, require few kernels, make faster computations, and have a good generalization and convergence [99]. SVR models train using a symmetrical loss function, which equally penalizes high and low errors. Using Vapnik's ε-insensitive approach, a flexible tube of the minimal radius is formed symmetrically around the estimated function, such that the absolute values of errors less than a certain threshold ε are ignored, both above and below the estimate. Points outside the tube are penalized, and those within the tube, either above or below the function, receive no penalty [100].

To develop the SVR model for predicting PV power generation, four parameters dominate the performance of the model. These parameters are penalty (C), which determines the penalties for estimation errors, tube radius (ε), which determines the data inside the tube to be ignored in regression, kernel coefficient (γ) and the kernel function's parameter. The suitable values of C and the kernel function's parameter must be selected to develop the appropriate prediction model. The performance of this model depends largely on the selection of the three parameters, which is a limitation of this method [38]. The basic kernels for this method are linear, polynomial, and radial basis function [101]. The basic kernels are visualized in Figure 2.6.

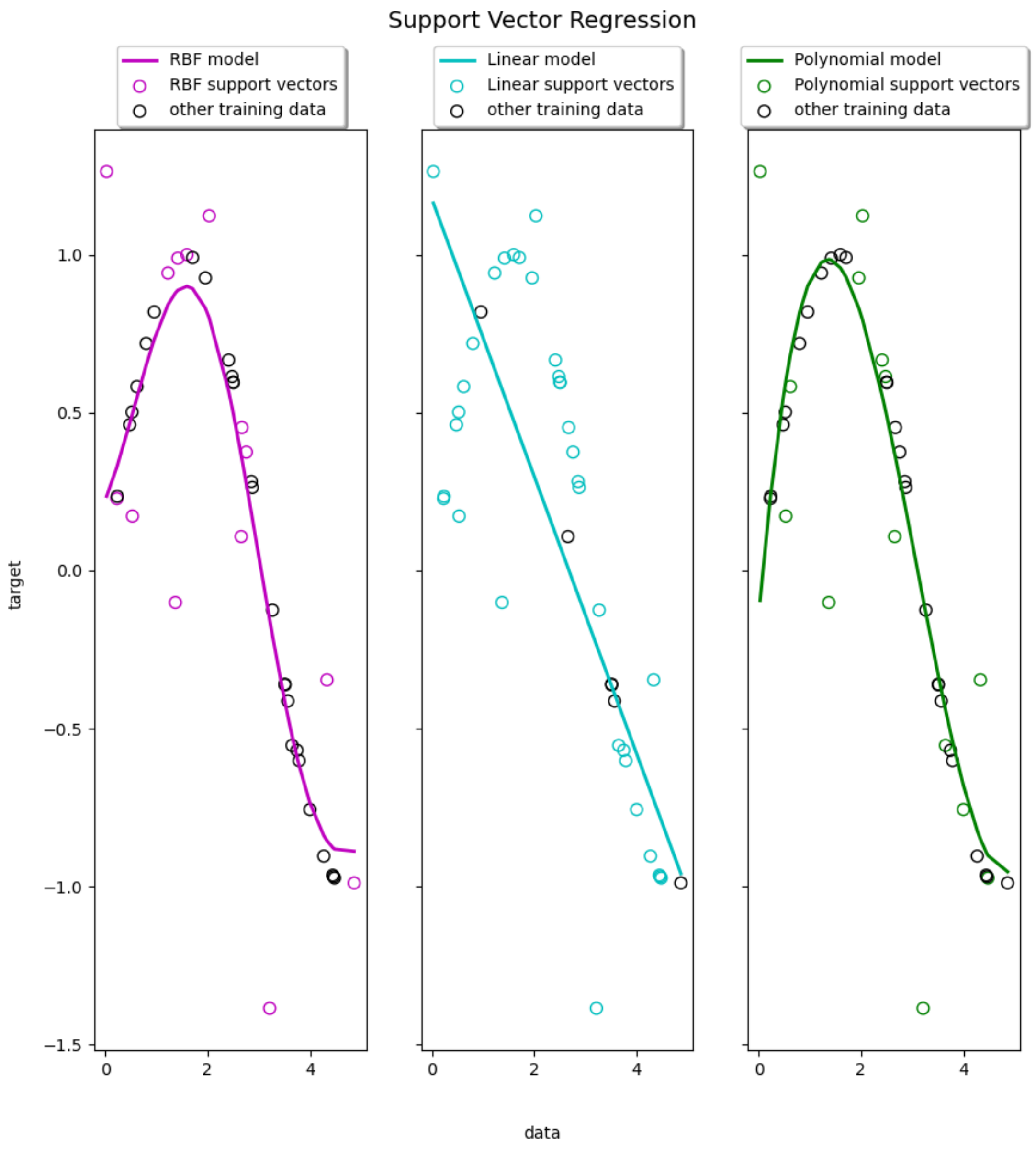


Figure 2.6: The basic kernels of support vector regression

The SVM regression function relates the input x to the output y as follows:

$$f(x) = \omega^T \vartheta(x) + b = y \tag{2.2}$$

where $\vartheta(x)$ is a nonlinear function mapping the input vector to a high-dimensional feature space. ω and b are the weight vector and bias terms, respectively, ω^T is the transposed ω and can be estimated by minimizing the following structural risk function

$$R = \frac{1}{2}\omega^T \omega + C \sum_{i=1}^{N} L\varepsilon(\hat{y}_i) \tag{2.3}$$

where N is the sample size, C represents the trade-off between the model complexity and the empirical error. An increase in C will increase the relative importance of the empirical risk concerning the regularization term. $L\varepsilon$ is Vapnik's ε-insensitive loss function. In general, there are different types of SVM, being linear SVM, LSSVR, v-SVM, and ε-SVR [102].

2.4 Statistical Models

In the statistical methods, the PV power generation is predicted from the statistical analysis of the different input variables. No internal information from the system is needed to model it. It is a data-driven approach that can extract relations on past data to predict future power generation. The quality of the historical data and the selection of a proper training data set is very essential to achieving an accurate forecast. Normally, these methods are adopted for short-term forecasting. The requirement of the input data series in this model is less compared to the machine-learning method [1], [38].

2.4.1 Auto-Regressive Moving Average models

The auto-regressive moving average (ARMA) is one of the most popular time series forecasting models. This is due to the model's ability to extract useful statistical properties. The model is based on two elementary parts: the moving average (MA) and the auto-regressive (AR). The ARMA model can be expressed as

$$P_t = \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \psi_{t-i} \tag{2.4}$$

where the first part is the AR model part and the second is the MA model part and where P_t is the forecasted PV power/irradiance at time t, p is the order of the AR model, φ_i is the i^{th} AR coefficient, q is the order of the MA error term, θ_i is the j^{th} MA coefficient, and ψ denotes the white noise, which is an independent variable with zero mean and constant variance [39]. A thorough search was done to figure out what the AR- and MA coefficients are, but with no luck. However, to use the ARMA model, only the order has to be established as far as the found literature stated. Here, the literature refers to the articles/reports in the literature review and the literature on which the theory is based. ARMA is suitable for forecasting the PV power generation from the specified time-series data [38]. The major limitation of the ARMA model is that the objective time series must be stationary, i.e., the statistical properties of the time series do not change over time [39].

2.4.2 Auto-Regressive Integrated Moving Average models

The auto-regressive integrated moving average (ARIMA) model is an extension of the ARMA model with an acceptable level of accuracy, where an integrated part removes any non-stationarity from the data. This makes it developed for non-stationary random processes. An ARIMA(p, d, q) model can be expressed as

$$(1 - \sum_{i=1}^{p} \varphi_i L^i)(1 - L)^d X_t = (1 - \sum_{i=1}^{q} \theta_i L^i)\varepsilon_t \tag{2.5}$$

where L denotes the lag operator, φ_i is the AR coefficient, θ_i represents the MA coefficients, ε_t is a white noise that is independent and identically distributed random variables with zero mean, p is the order of AR, d is the number of nonseasonal differences, and q is the MA order. The ARIMA model is the most general class of models for time series prediction. The success of ARIMA is because of its exceptional ability to capture the periodical cycle better than other methods [38], [39].

Theoretically, both ARMA and ARIMA cannot involve the process behavior. To consider exogenous inputs, the ARMA model with exogenous inputs, (ARMAX) model is applied, which has proved to be a great tool in time series prediction. ARMAX is an extension of ARIMA and can be more flexible for practical use of PV power prediction because it can include external variables such as temperature, humidity, and wind speed [39].

Some ARIMA model orders are ARIMA(1,0,0), the first-order auto-regressive model, ARIMA(0,1,0), the random walk, ARIMA(1,1,0), the differenced first-order auto-regressive model and ARIMA(0,1,1), the simple exponential smoothing [103].

2.5 Ensemble models

Ensemble learning is a machine learning approach where more than one learner is trained to achieve the same classification or regression goal. Contrary to ordinary machine learning models that try to learn just one pattern from training data, ensemble learning methods form a set of patterns and utilize a combination of them. A major objective of an ensemble model is to decrease the generalization error by reducing the variance or bias. The learner used in an ensemble model is called a base learner and an ensemble has usually better performance than its base learners. Weighted averaging and majority voting are the commonly used two methods for regression and classification respectively [104].

Weighted average ensembles assume that some models in the ensemble have more skill than others and give them more contribution when making predictions. Each model is assigned a fixed weight that is multiplied by the prediction made by the model and used in the sum or average prediction calculation. The challenge of this type of ensemble is how to calculate, assign, or search for model weights that result in a performance that is better than any contributing model and an ensemble that uses equal model weights.

Many approaches can be used to choose the relative weighting for each ensemble member. For example, the weights may be chosen based on the skill of each model, such as the classification accuracy or negative error, where large weights mean a better-performing model. Performance may be calculated on the dataset used for training or a holdout dataset, the latter of which may be more relevant.

The scores of each model can be used directly or converted into a different value, such as the relative ranking for each model. Another approach might be to use a search algorithm to test different combinations of weights.

2.6 Interval Prediction

Different from deterministic prediction results, prediction intervals (PIs) are composed of maximum and minimum prediction results below a certain confidence level. Interval prediction can obtain the upper and lower bound of a future value and is more suitable for solar power due to its high variability. It is more applicable for systems requiring risk management like electricity production. Predicting an interval offers additional variability information than just predicting a single value. When knowing the range of the target point, better energy management of the grid system can be built to minimize operation costs and improve stability [105].

There are mainly two types of approaches to estimate prediction intervals in literature. The first type is the theoretical approach, where a theoretical interval is calculated based on the assumption that forecasting errors follow a determined distribution with zero mean, usually the normal distribution. However, in the real world, where data always involves complex processes, it is hard to ensure the assumption can be fulfilled. The theoretical prediction interval may behave poorly if the aforementioned assumption is not valid. As alternatives, another type of approach has been proposed with no need for considering the forecasting error distribution. The empirical approach is an example of such an approach. This type of approach is claimed to achieve robust performance for the construction of prediction intervals [105].

Probabilistic interval prediction uses a predictive model to acquire a range of possible fluctuations in PV power at a certain confidence level, which consists of the lower and upper bounds of the interval (the midpoint of the interval can be used as point prediction), where the future value is expected to lie between, with a prescribed probability. A prediction interval is an interval estimate for an (unknown) future value. As a future value can be regarded as a random variable at the time the forecast is made, a prediction interval involves a different sort of probability statement from that implied by a confidence interval [41], [106]. Figure 2.7, illustrates prediction intervals for gradient boosting regression.

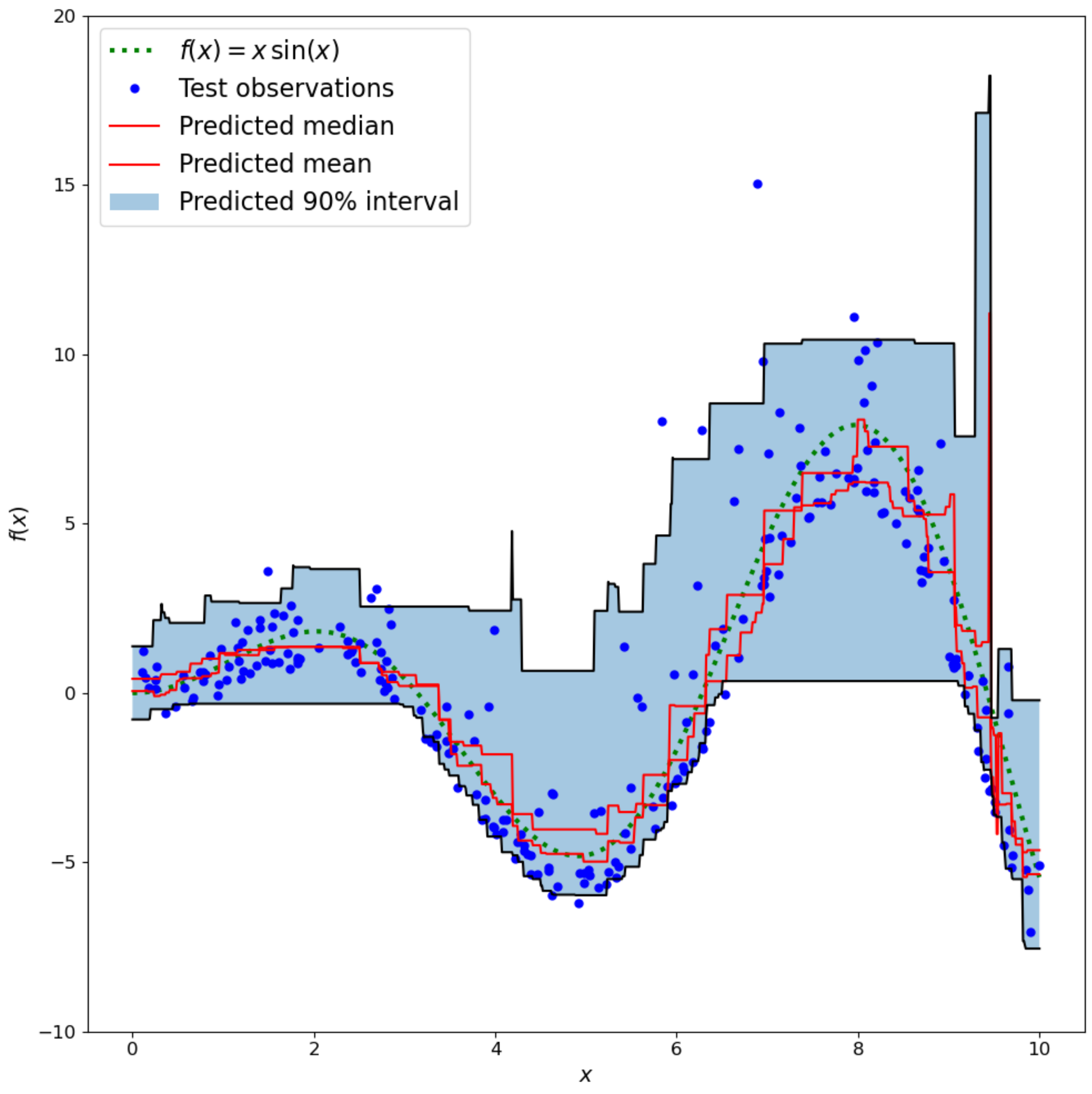


Figure 2.7: Prediction Intervals for Gradient Boosting Regression

2.6.1 Quantile Regression

Comparing with the ordinary regression model, the main advantage of quantile regression (QR) is that it is more stable for dispersed data in the response measurements. The estimation method based on least squares is used to obtain approximate values of predictor variables. Furthermore, QR is capable of comprehensively analyzing the relevant relationships between independent vector $X = [x_1, x_2, ..., x_n]$ and dependent vector $Y = [y_1, y_2, ..., y_n]$, where $x_i = [x_{i,1}, x_{i,2}, ..., x_{i,r}]', x_{i,j}$ is the j-th value of x_i, y_i is the i-th variable of Y, and n is the total number of samples [107].

Quantile regression neural network

A traditional linear quantile regression model limits the influence pattern of explanatory variables to dependent variables. A more realistic behavior is present in nonlinear models, which ANN models provide. Based on the single hidden-layer neural network, the following QRNN model was proposed by Taylor [108].

$$f(x_i, W_i(\tau), V_i(\tau)) = h_2 \left\{ \sum_{k=1}^{m} v_{i,k} h_1 \left[\sum_{j=1}^{r} \omega_{i,j,k}(\tau) x_i \right] \right\} \tag{2.6}$$

where f is function, $W_i = \omega_{i,j,k_{j=1,2,...,r;k=1,2,...,m}}$ is the weight vector, $V_i = v_{i,k_k} = 1, 2, ..., m$ is the connection weight between the hidden layer and output, m is the number of hidden nodes, and r is the size of independent variables. $h_1(\cdot)$ and $h_2(\cdot)$ are activation functions.

2.7 Hyper-parameter Optimization

To optimize the models, hyper-parameter optimization may be used. Grid search is such an optimization process. To perform a grid search in Python GridSearchCV may be used [109]. GridSearchCV tries all the combinations of the values passed in the dictionary (the parameter grid to explore, as a dictionary mapping estimator parameters to sequences of allowed values) and evaluates the model for each combination using cross-validation. Cross-validation is one of the most widely used data re-sampling methods to assess the generalization ability of a predictive model and to prevent overfitting [110]. After using GridSearchCV one gets accuracy/loss for every combination of hyper-parameters and may choose the one with the best performance [111].

2.8 Prediction Performance, Bias and Variance

While making predictions, a difference occurs between the predicted value and the actual value, and this variation is known as bias. When the bias is high, assumptions made by the model are too basic, and it cannot capture the key features of the data. This means that the model has not captured patterns in the training data and hence cannot perform well on new data. Variance is a measurement of a model's sensitivity to fluctuations in the data. It tells how much a random variable is different from its expected value, going from one dataset to another. The model may learn from noise, which will cause the model to consider trivial features as important. For any model, it is important to find the perfect balance between bias and variance. A low bias often causes high variance and low variance can cause bad fitting and high bias. This is called the bias/variance dilemma. A perfect balance ensures that the model captures the essential patterns of the data while ignoring the noise present. This is called a bias-variance trade-off, and it helps optimize the error in the model and keeps it as low as possible. The mean square error (MSE) can be written as the sum of the variance and the square of the bias. If there is bias, this may indicate that the model does not contain the solution, meaning it is underfitting. If there is variance, this may indicate that the model is too general and also learns the noise, meaning it is overfitting [97], [112], [113].

2.8.1 Deterministic Error Metrics

To measure the accuracy of machine learning models, different evaluation metrics have been proposed and applied in literature. Mean Squared Error, Root Mean Square Error (RMSE), normalized Root Mean Square Error (nRMSE), and Mean Absolute Error (MAE), have been commonly used in evaluating the accuracy of deterministic predictions [38]. R^2 score, the coefficient of determination provides an indication of how well the model fits the data. Therefore, it is a measure of how good unseen samples are likely to be predicted by the model. It represents the proportion of variance that has been explained by the independent variables in the model.

The evaluation metrics RMSE, nRMSE, and MAE, as well as the R^2 score, may be calculated using the equations

$$\text{RMSE} = \sqrt{\tfrac{1}{N} \textstyle\sum_{i=0}^{N-1} (y_i - \hat{y}_i)^2} \tag{2.7}$$

$$\text{nRMSE} = \frac{\sqrt{\frac{1}{N} \sum_{i=0}^{N-1} (y_i - \hat{y}_i)^2}}{\hat{y}_{i,max} - \hat{y}_{i,min}} \tag{2.8}$$

$$\text{MAE} = \tfrac{1}{N} \textstyle\sum_{i=0}^{N-1} |y_i - \hat{y}_i| \tag{2.9}$$

$$\text{R}^2 = 1 - \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} (y_i - \overline{y}_i)^2}, \tag{2.10}$$

where y_i and $\hat{y}_i$ represent the true/measured value and the predicted value, respectively. N is the number of data samples for the period, and $\overline{y}_i$ is the average of the measured value. RMSE is a risk metric corresponding to the expected value of the squared error or loss. MAE is the risk metric corresponding to the expected value of the absolute error loss [109].

MAE and RMSE give slightly different results. MAE gives the mean absolute difference between the predicted values and the actual values in a dataset, and RMSE gives the square root of the average squared difference between the predicted values and the actual values in a dataset. If one would like to give more weights to observations that are further from the mean (i.e. if being off by 20 is more than twice as bad as being off by 10) then it may be better to use the RMSE to measure error. because the RMSE is more sensitive to observations that are further from the mean. However, if being off by 20 is twice as bad as being off by 10 then it may be better to use the MAE [114], [115]. In literature, a new metric was observed, called Mean Relative Error (MRE) which is the same as MAE/Wp.

2.8.2 Probabilistic Error Metrics

The main requirement when studying the performance of probabilistic forecasts is reliability. The prediction interval coverage probability (PICP) is a metric that assesses whether the probability distribution of observations lies within the prediction interval. The PICP metric can be formulated as follows.

$$\text{PICP} = \frac{1}{N}\sum_{i=1}^{N} \epsilon_i, \tag{2.11}$$

where ϵ_i is defined as:

$$\epsilon_i = \begin{cases} 1 \text{ if } x_i \in [L_i, U_i] \\ 0 \text{ if } x_i \notin [L_i, U_i], \end{cases} \tag{2.12}$$

where L_i and U_i represent the lower and upper bound of the prediction interval, respectively. A high value for PICP implies that more results lie within the bounds of the prediction interval, which is desirable. The PICP measure is a quantitative expression of reliability and should be higher than the nominal confidence level since these are otherwise invalid and should be discarded [42].

The PICP should be simultaneously analyzed with the prediction interval normalized average width (PINAW), which is a measure that quantitatively assesses the width of the prediction intervals. The PINAW can be defined as follows [42], [43].

$$\text{PINAW} = \frac{1}{N}\sum_{i=1}^{N} \frac{U_i - L_i}{y_{max} - y_{min}}. \tag{2.13}$$

PICP and PINAW usually have a direct relationship in which the high width of the prediction interval implies high coverage (PICP) of results, and therefore a quantitative measure to assess both simultaneously were proposed. This measure is called both coverage length-based criterion (CLC) and coverage width-based criterion (CWC), and can be formulated as follows:

$$\text{CWC} = \text{PINAW}(1 + \gamma(\text{PICP})e^{-\eta(\text{PICP}-\mu)}), \tag{2.14}$$

where η and μ are controlling parameters and $\gamma(\text{PICP}) = 1$ during training. μ represents the preassigned PICP that is to be achieved during the training phase. To select this parameter, the confidence level can be used as guidance. Moreover, η is a penalizing term that will cause CWC grow exponentially if the preassigned PICP is not satisfied. Based on literature, this value is 50 [43], [116], [117]. When PICP $\approx \mu$, one has achieved balance between PICP and PINAW and can continue with testing of the model. Then, CWC is to be determined with γ(PICP) depending on μ, which is formulated as follows [42]:

$$\gamma(\text{PICP}) = \begin{cases} 0 \text{ if PICP} \geq \mu \\ 1 \text{ if PICP} < \mu. \end{cases} \tag{2.15}$$

3 Methods

This chapter describes the progress followed in this thesis, based on the theory of steps one can follow. First some brief information on what had been done and which tools have been used before a thorough description of the steps taken to build the models, make predictions and evaluate them.

The historical power production data from a PV plant placed in Lillesand belonging to food producer Gorines was used as input in addition to Numerical Weather Prediction (NWP) data from the same place. The ARIMA model did only use one NWP variable, and one ANN model was built using only NWP variables. The whole process of building the models, the optimization, the predictions, the testing of performance, and the plotting were coded in Spyder (Python 3.8), and the script is shown in Appendix A.2. All tables coded in Spyder (Python 3.8), were exported from Spyder using the code $tablename.to\_csv('tablename.csv')$ and converted from CSV file to LaTeX Table using a web-based tool [118]. One table was also made in Excel and then converted using the same web-based tool. The main steps used to build these prediction models are based on the steps shown in Figure 2.3.

3.1 Data analysis, Processing, Input Selection and Data Division

The PV plant in Lillesand consists of 2046 solar panels, 1248 panels with 290 W_p per panel, and 798 panels with 315 W_p per panel. In total, the PV plant is 613 kW_p with a battery bank of 350 kWh. The power production from the solar cell system was measured in kWh from three inverters. The NWP model is an enhanced downscaled model data provided by the European Center for Medium-Range Weather Forecasts (ECMWF) Integrated Forecasting System (IFS). ECMWF produces operational ensemble-based analyses and predictions that describe the range of possible scenarios and their likelihood of occurrence. Their provided Atmospheric Global Circulation model describes the dynamical evolution of the atmosphere worldwide on the resolved scale and is used for medium-range, extended medium-range, and seasonal forecasts. It is a general atmospheric model of uniform model physics and structure and is executed on a global scale at several resolutions each appropriate to the forecast period. The model uses the most accurate estimate of the current conditions and the most up-to-date description of the model physics. A single execution of the model does not give definitive results so they also generate an ensemble of perturbed runs [119]. Reading literature, it was observed that the NWP variables that were frequently used are solar irradiance, temperature, wind speed, wind direction, cloud cover, and precipitation. The same variables were therefore included in the input selection for this work.

The historical power production and the NWP data consisted of hourly recorded data. The production dataset had measured power from three inverters, recorded mainly at times 00:00, 01:00, and so on, but in some cases at another minute during the hour. The code for cleaning the data in a way that all data corresponds to the same hour in the 00:00, 01:00 format at the same time zone, as they were initially recorded for different time zones, as well as checking the data for missing values is shown in Appendix A.2 subsection A.2.1. The production from the three inverters was summed and combined into one column to represent the total production. During the analysis, it was noticed that there was forecasted direct and/or diffuse radiation at times when the global radiation was forecasted to zero. Therefore, these two variables were considered as inputs as well, which is different from the research project [55]. The amount of missing data after cleaning was 100 hours of the total of 8760 hours. This corresponds to a missing data percentage of 1.142. As this is less than 5%, no processing methods were used.

From the cleaned data frame, the July month was selected to visualize how the production, the irradiation, and temperature varies. Further, a clear sky day of all months was selected to show the variation in production over a year. Next, to prepare the data for forecasting, columns of time-lags (past values) were made by time-shifting the data. The visualization is presented in Chapter 4, and the codes for visualization and shifting of data, are shown in Appendix A.2 subsection A.2.2 and A.2.3.

To reduce the number of variables two methods were employed. The correlation between variables and their time lags was found utilizing both Pearson, Kendall, and Spearman correlation as these are the types of correlation often measured, and as one or more of them were used in relevant literature [12], [27], [99]. The results are visualized using tables and plots in Chapter 4, where the script is shown in Appendix A.2 subsection A.2.3.

To further prepare the data for the models, the data was split into randomized training data, a randomized testing data, and the month of August was selected as the second testing data. Randomized, means that the data is shuffled and that the statistical difference between the randomized training data and testing data is very small. This choice of two testing data was made to see how well the models do on both types. The data was split so that the August test was 744 hours, being 8.54 % of the whole dataset, the random test was 1539 hours, being 17.67 % of the whole dataset and the random train is 6372 hours, being 73.17 % of the whole dataset. After the data was further split into x and y, where x holds the input data and y the output data, scaling, fitting, and transformation was done. Please refer to the pre-project for further description and all the reasons why [55]. The code for the randomized split is as follows.

```
def randomization(dataset,percentage):
    dataset=pd.DataFrame(dataset)
    index=int(np.ceil(percentage*len(dataset)))
    for i in range(1000000):
        print(i)
        shuffled=dataset.iloc[0:len(dataset) ,:]
        shuffled=shuffled.sample(frac=1)
        train = shuffled.iloc[0:index , :].values
        test=shuffled.iloc[index:len(dataset), :].values
        AV_train=train.mean(0)
        AV_train=AV_train.reshape(1,train.shape[1])
        STD_train=train.std(0)
        STD_train=STD_train.reshape(1,train.shape[1])
        AV_test=test.mean(0)
        AV_test=AV_test.reshape(1,train.shape[1])
        STD_test=test.std(0)
        STD_test=STD_test.reshape(1,train.shape[1])
        AV=np.concatenate((AV_train,AV_test),axis=0)
        STD=np.concatenate((STD_train, STD_test),axis=0)
        CV=STD/AV
        C1=CV[0,:].reshape(1,train.shape[1])
        C2=CV[1,:].reshape(1,train.shape[1])
        C12 = np.vstack([C1, C2])
        MaxC12=C12.max(0).reshape(1,train.shape[1])
        ERR=np.vstack([(abs((C1-C2)/MaxC12))])
        if np.all(ERR <=0.03):
            print("result"+str(i))
            result=shuffled
            break
    return result.iloc[0:index , :],result.iloc[index:len(dataset), :]
```

The whole script for data division, scaling, fitting, and transformation is shown in Appendix A.2 subsection A.2.4.

The second feature reduction technique PCA was done after the last step described above, where the number of variables was chosen so that the data still contained 99% of the variance after PCA was done. The PCA script is shown in Appendix A.2 subsection A.2.5.

The below-described models, except the one that only used NWP variables, used correlated NWP variables at time t, t-1, t-2, and so on, as well as correlated production at time t-1, t-2, and so on, as inputs to establish the relationship between all these inputs and the production at time t. For example, at 08:00, the correlated previous hours of production and NWP variables, and NWP variables at 08:00 are used as inputs.

3.2 Multilayer Perceptron network

The multilayer perceptron network was made using the Keras Sequential model [120]. Based on theory and literature [9], [116], [121], [122], the model was initially set to have 2 hidden layers with $7 \cdot \frac{2}{3} + 1 \approx 6$ neurons in each layer, He uniform initializer, the ReLu activation function and the Adam optimizer. The output neuron was initially set to have a He uniform initializer and linear activation function. The loss function for this model was set to be MSE, based on theory, and early stopping was set with patience of 10, meaning training stops after no improvement in 10 epochs, and with restoring of the best weights. Patience of 10 was set to ensure that more epochs would be used. The batch size (number of samples per gradient update) was initially set to default (32) and the validation split was set to 0.2. The amount of validation split was chosen based on still having enough training data.

Different batch sizes, numbers of neurons in the hidden layers, number of hidden layers, initializers, activation functions, and optimizers were tried in GridSearchCV [109]. Since GridSearchCV takes some time, all parameters will not be tested at the same time.

For each run of the ANN model, one gets slightly different results. To visualize this, a *for*-loop was made, and then all the different predicted results were plotted in one plot. Because of this difference, a decision to make an ensemble of multiple ANN fits was made.

A decision was made to make a forecast using only NWPs as inputs. The input to make this forecast was decided to be all available NWP variables. For this forecast, another decision was made to not include any previous hours, meaning no time-lags were used. After the relation between the NWP data and the production was made using the train data, the model was fed with the two test data, random and August, and then used this information to predict the production for the same timestamp as the fed NWP test data. Similarly, this model was made using the Keras Sequential model [120], with a linear activation function for the output layer. The loss function for this model was also set to be MSE, and early stopping was set with patience of 10, with restoring of the best weights. All other possible variations were chosen with GridSearchCV.

3.3 Support Vector Regression

The Support vector machine model was made using the sklearn.svm.SVR-model [109], which uses the ε-support vector regression. The implementation is based on libsvm, which is an open-source machine learning library. The chosen kernel is the default kernel radial-basis function (RBF), as this was the one used in the pre-project [55], in literature [25], [123], and is one of the basic kernels for this model. To select the best hyper-plane parameters needed for fitting, GridSearchCV was used. It searches for the optimal combination of C, ε, and γ, with a k-fold of 5, and uses a stepwise approach, using the Grid Search algorithm, combined with the cross-validation method to optimize the hyper-parameters.

3.4 Auto-Regressive Integrating Moving Average

The testing and training data for the ARIMA model were changed to no longer consist of previous hour (lagged) values, as this model only accepts one input array. Therefore, the testing data was set to be for August 2021 and the training data, to be for the rest of 2021. In addition, scaling, fitting, transformation, and PCA were not used in this model, as this was not found to be used in the literature. Different orders were tested to see what order gave the best performance. This model uses quite more executing time than the others resulting in only testing some orders. The tested orders were (1,0,0), the first-order autoregressive model, (1,1,0), the differenced first-order autoregressive model, (0,1,1), simple exponential smoothing, and (1,0,1), as this was the order of one of the models in [22].

3.5 Ensemble models

Two different ensemble methods were made. The first one combined the ANN model and the SVR model, using a weighted average method called voting regressor provided by scikit-learn [109]. The same ANN- and SVR model that was used separately, was used as learners. To get the ranking weight, each learner was evaluated using MAE. The second ensemble method combined two different ANN models with different settings and parameters, the first with the same setting as the standalone ANN model, and the second with settings and parameters from another grid search. Both models were run 15 times, making 15 different results, and then combined to make one resulting point prediction using the same weighted average method as for the first ensemble. The script for getting the ranking weight is as follows.

```
def evaluate_models(models, X_train, X_val, y_train, y_val):
    # fit and evaluate the models
    scores = list()
    for name, model in models:
        # fit the models
        model.fit(X_train_scaled, y_train_scaled.ravel())
        # evaluate the model
        yhat = model.predict(X_test_scaled)
        yhat = scaler_y.inverse_transform(yhat.reshape(-1, 1))
        yhat.min()
        yhat[yhat < 0] = 0
        mae = mean_absolute_error(y_test, yhat)
        # store the performance
        scores.append(-mae)
        # report model performance
    return scores

# fit and evaluate each model
scores = evaluate_models(models, X_train_scaled, X_test_scaled,
                         y_train_scaled.ravel(), y_test)
ranking = 1 + argsort(argsort(scores))
```

3.6 Interval Prediction - Quantile Regression

To make a Quantile regression neural network, a change to the ANN model was made. The loss function was changed from MSE to a quantile loss function. The code for the quantile loss function is as follows.

```
def QR_loss(q, y, f):
    e = (y - f)
    return keras.backend.mean(keras.backend.maximum(q * e, (q - 1) * e),
                              axis=-1)
```

The complete script which includes how to implement the loss function is shown in its completeness in Appendix A.2 subsection A.2.9. The interval was set to be within the 5th and 95th percentile.

3.7 Hyper-parameter Optimization, predicting, and post-processing

To find the best settings and hyper-parameters, manually testing and GridSearchCV [109] were used. Grid search was chosen based on earlier success and because it was used in literature as well [24], [36], [101]. For the ANN model, the settings and parameters that were tested are described in section 3.2, while the tested hyper-plane parameters for SVR were ε, C, and γ. Five folds were used as this is the default and since the dataset is not that large.

For SVR, GridSearchCV was ran 2 times, changing the initial set parameter grid a little based on their previous result. The initial set grid was the following.

```
parameters={'C':[1,2,3,7,10], 'gamma':[1, 0.1, 0.01, 0.001],
            'epsilon':[0.01, 0.05, 0.08]}
```

For ANN, there were much more settings and parameters to try. First, GridSearchCV attempted searching all parameter combinations, but even after over 10 hours, the search still continued. Therefore it was stopped and the decision to split the search in two parts was made. First, the number of hidden neurons and layers was tested together with activation functions. Based on theory, the number of hidden neurons was set to search between 6 and 14, and the activation function was set to be ReLU, SELU, or ELU. SELU and ELU were chosen together with RELU, the most used activation function, because they have been shown to reduce or avoid exploding and/or vanishing gradients. The number of hidden layers was set to be searched between 2 and 4 because the network should be kept simple to help ensure training time within reasonable bounds and good learnability and generalization as described 2.2.1. After finding the best combination between these three parameters, GridSearchCV was set to search for the optimal combination of the optimizer, initializer, and batch size. The parameters from the first try were added to the model before searching the second time. Also based on the theory the searched optimizers were Adam, RMSProp, and SDG, the searched initializers were He uniform and He normal if the activation function were searched to be RELU and, Lecun normal and Lecun uniform if the activation function were searched to be SELU or ELU. The searched batch sizes were 32, 64, 128, and 256, based on literature [124], [125].

For the ANN model using only NWPs as inputs, a new GridSearchCV was done. The first and second try was done the same way as earlier, but where on the first try, the result from the previous second try was used for the parameters that were not tested on the first.

To find the settings and parameters for the second ensemble learner for the ANN ensemble, GridSearchCV was run again the same way as GridSearchCV was run for the ANN model with only NWPs as inputs.

After finding the best parameters, all models needed to be fit. This is done using *.fit(...)* , where the insides of the parentheses was *X_ train_ scaled, y_ train_ scaled, epochs = 100, batch_ size = 64, validation_ split = 0.2, callbacks = [callback], verbose = 1* for the MLP- and QRNN models, nothing for the ARIMA model and *X_ train_ scaled,y_ train_ scaled.ravel()* for the SVR model and both ensemble models. The validation split of 0.2 splits the train and validation into 80 % train and 20 % validation. The callback refers to the early stopping and verbose of 1 refers to "seeing" the training progress for each epoch like this:

```
Epoch 1/100
[==============================] - 0s 843us/step - loss: 0.1779 - val_loss: 0.0891
Epoch 1/100
[==============================] - 0s 532us/step - loss: 0.0974 - val_loss: 0.0830
.
.
.
Epoch 100/100
[==============================] - 0s 528us/step - loss: 0.0271 - val_loss: 0.0640
```

The *.ravel()* function returns contiguous flattened array (1D array with all the input-array elements and with the same type as it).

After fitting, the predictions were made using *.predict(...)*, where the insides of the parentheses was *X_ test_ scaled* for predicting on the random test data, *X_ test_ scaled_ 1* for predicting on the august test data, and *X_ train_ scaled* for predicting on the train data. For ARIMA the insides was *start, end, exog = xexog*, where start is the length of train data (len(train) in python = 5065), end is the length of train data added to the length of test data subtracted from one (len(train) + len(test) - 1 in python = 5065), and xexog is the global radiation in August. Running the scripts for optimization did take some time to complete on a rather slow laptop but was completed a lot faster, using a newer and faster desktop computer with the following performance, NVIDIA GeForce

RTX 3070, AMD Ryzen 7 5800X 8-core processor, and a relatively normal but decent WiFi. The GridSearchCV scripts is included in Appendix A.2 sections .

After predictions were made, the arrays had to be inverse transformed to be on the right scale for comparison with test and train data. Since there cannot be negative production, all negative production was set to zero. To be able to test the performance of the model, predictions were made for both test datasets and the train data.

3.8 Prediction Performance

A performance function was made to test the deterministic performance of all models. The RMSE, the MAE and the R^2 score was measured by means of equation 2.7, 2.9 and 2.10. The nRMSE however was measured by dividing the RMSE by the rated capacity of the PV system (613kWp). The function is is shown in Appendix A.2 subsection A.2.6. To measure the RMSE, nRMSE, MAE, and R^2 for the QRNN model, the 50th percentile was used as this is the median. Please refer to the pre-project for a more thorough description of the methods for measuring deterministic performance [55].

In addition to the before-mentioned performance methods, the error (true - predicted) was measured for all data points to create a plot of all errors to more easily visualize at what timestamp the models make the biggest mistakes. The error for each timestamp was found with the script below.

```
def create_error(true, pred):
    all_errors = []
    for i in range(len(pred)):
        error= true[i]-pred[i]
        all_errors.append(error)
    return all_errors
```

The largest error can be presented as a percentage of the rated capacity of the PV system (613kWp) and will be the maximum or minimum of the results from the script above.

The relation between the RMSE of the predicted value from training and testing data was compared by dividing the "train RMSE" by the "test RMSE". If this relation is approximately one, it does not necessarily mean the model does not over-fit or under-fit the data, but it indicates the fit. This relation is in this case called an over-fitting indicator.

The probabilistic error metrics were measured employing equations 2.11 for measuring PICP, 2.13 for measuring PINAW, and 2.14 for measuring CWC. PICP, PINAW, and CWC were measured for a 90 % confidence interval and an 80 % confidence interval. The script for measuring these for the August test at a 90 % confidence interval is shown below.

```
L = pred[['5th']].values
U = pred[['95th']].values

mu = 0.90
eta = 50
epsilon = []
pi = []
for i in range(len(y_test_1)):
    if y_test_1[i] >= L[i] and y_test_1[i] <= U[i]:
        c = 1
    else:
        c = 0
    epsilon.append(c)
    a =np.float64((U[i]-L[i])/(y_test_1.max()-y_test_1.min()))
    pi.append(a)
Picp = sum(epsilon)/y_test_1.shape[0]
Pinaw = sum(pi)/y_test_1.shape[0]

if Picp >= mu:
    gamma = 1
else:
    gamma = 0
Cwc = Pinaw*(1+gamma*np.exp(-eta*(Picp-mu)))
```

The script for random test is the same, except y_test_1 is replaced by y_test and pred is replaced by preds, where pred is the predicted production for August and preds is the predicted production for random test. To measure PICP, PINAW and CWC for a 80 % confidence interval, L and U were `pred[['10th']].values` (`preds[['10th']].values`) and `preds[['90th']].values` (`preds[['90th']].values`), respectively. For a 80 % confidence interval the mu was also change to 0.8

4 Results and Discussions

Results and discussion based on the methodology described in chapter 3 will now be presented. Firstly, it will be on data analysis, processing, input selection, and data division, followed by results and discussion on the developed models. Lastly, the prediction performances of the models will be presented and discussed. Both tables and plots of the results will be presented to give a better overview of the obtained results.

4.1 Data analysis, Processing, Input Selection and Data Division

The input selection was, as mentioned in chapter 3 section 3.1, based on on-site PV power production and NWPs from the same site, provided by the ECMWF IFS. The models used slightly different variables from all the available ones, and this will be specified in this section as well as done in the methodology for a reminder.

To see the nature of the data and to visualize how the variables change during the available time, several plots were made. Figure 4.1 shows the hourly production in July 2021, Figure 4.2 and Figure 4.3 shows the hourly global, direct and diffuse radiation and the hourly air temperature in July 2021, respectively. No information was found as to how or where the radiation variables were measured, but the unit is assumed to be W/m^2.

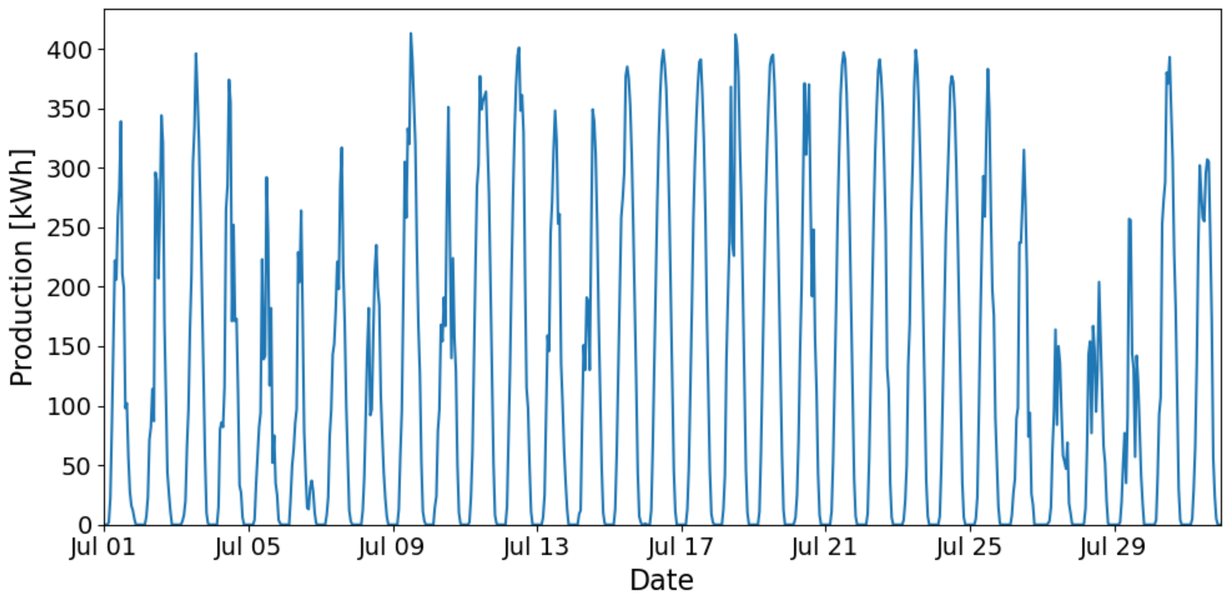


Figure 4.1: Hourly PV power production [kWh] in July 2021.

The production and radiation vary quite similarly throughout July month and between days. This was expected as there is no sunshine during nighttime. As the plots show, when the radiation is

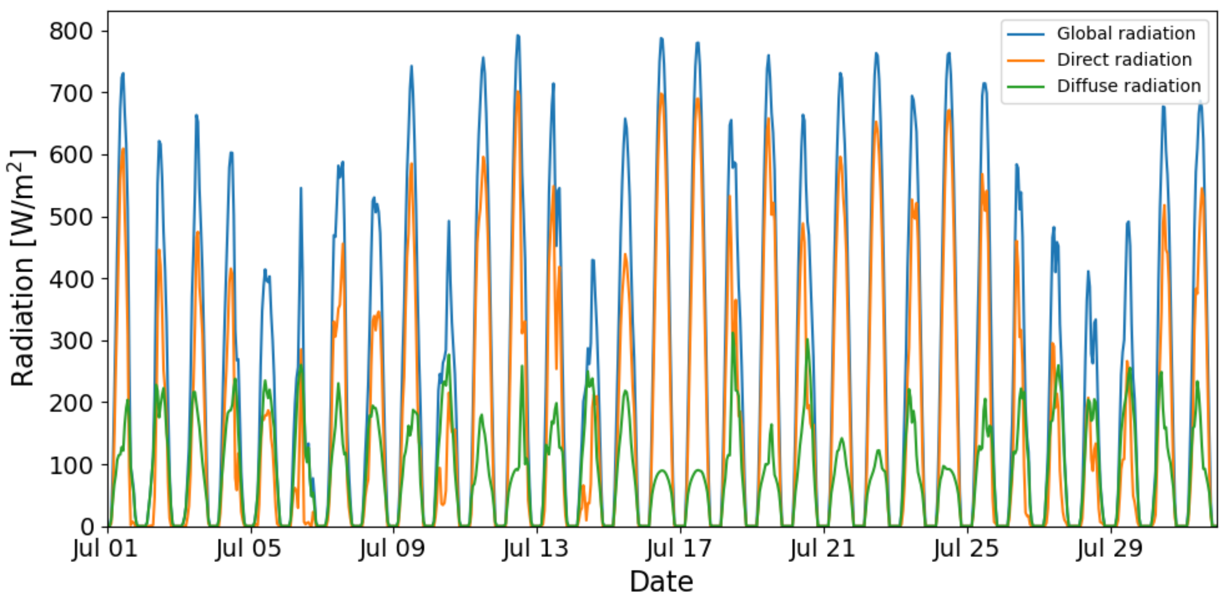


Figure 4.2: Hourly global, direct and diffuse radiation [W/m^2] in July 2021.

high, the production is high and the same goes for low radiation. The diffuse radiation does not vary as much and mostly does not follow the same pattern as the other variables. This indicates that the correlation between the global radiation and the production and between the direct radiation and the production is higher than the diffuse radiation and the production.

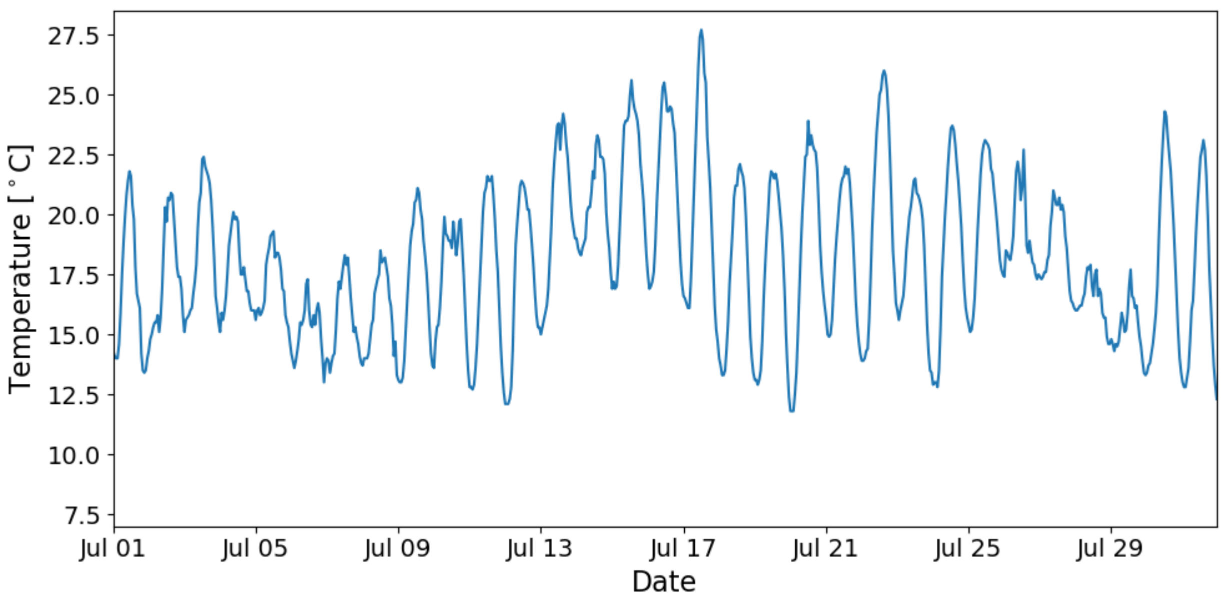


Figure 4.3: Hourly air temperature [°C] in July 2021.

As with the production and radiation, the air temperature also varies. This is in correlation with the radiation as the sun heats the air. Regarding correlation, as may be seen in Figure 4.3, when the temperature is relatively high, the production might not be. It is not easy to read that from the plots, but it is an indication of a lower correlation between the production and air temperature.

To be able to see the variation in production and global radiation over a year, two plots were made. Figure 4.4 shows the hourly production of a clear sky day for all months of 2021 and Figure 4.5 shows the hourly global radiation for the same clear sky day for all months of 2021.

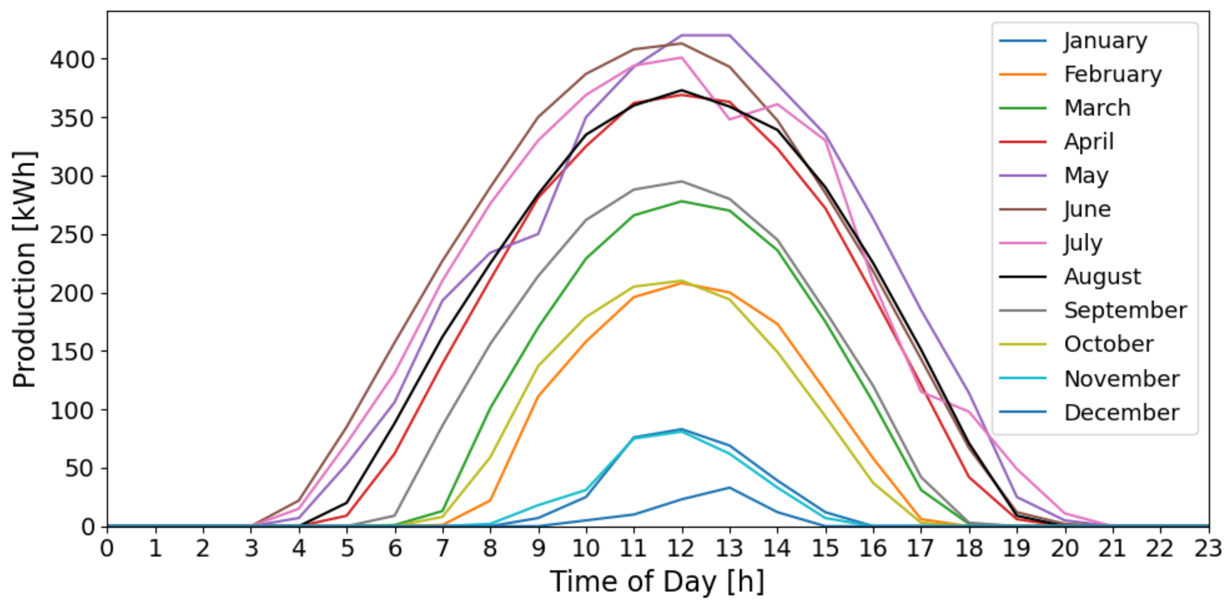


Figure 4.4: Hourly PV power production [kWh] on a clear sky day for for all months in 2021, showing the difference in power production each month.

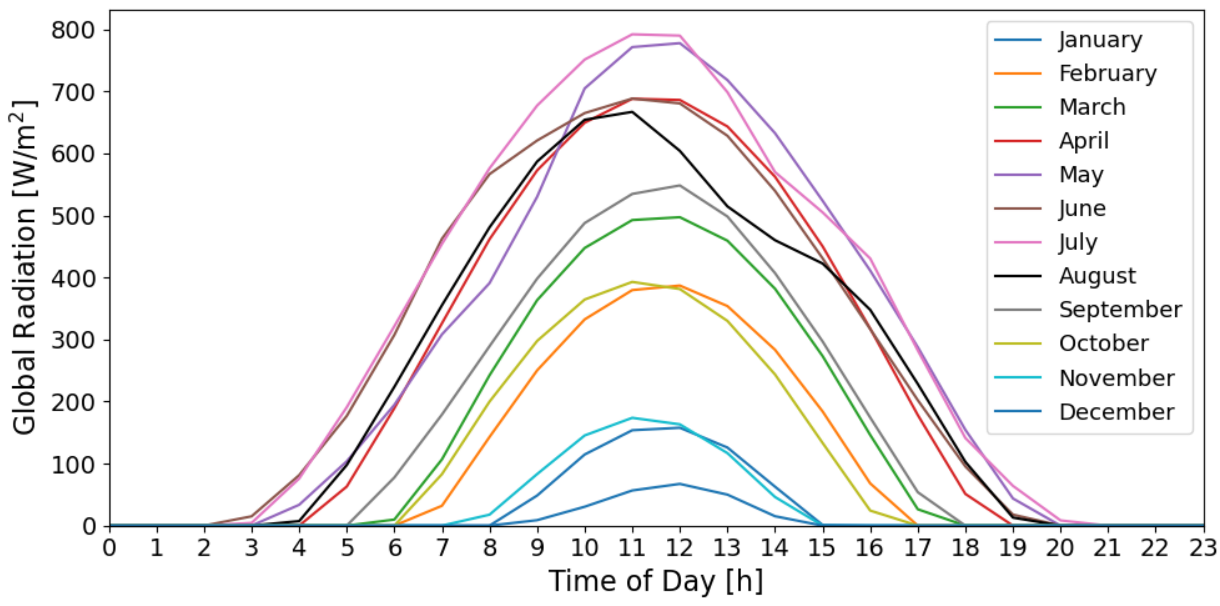


Figure 4.5: Hourly global radiation [W/m^2] on a clear sky day for all months in 2021, showing the difference each month.

The figures show a definite variation in production, radiation, and temperature between months, the days in a month, and the hours of a day. Figure 4.4 and Figure 4.5 also show that the periods that should be equivalent concerning the position of the sun in the sky, are quite similar. With more data over multiple years, this analysis could show even more similarities.

All considered inputs and how they varied the between 08:00 and 20:00 the first day of the recorded data is shown in Table 4.1. Temp is short for temperature, Prec is short for precipitation, and WS, WD, GI, DI, DR, CC, and P corresponds to wind speed, wind direction, global radiation, direct radiation, diffuse radiation, cloud cover, and production, respectively.

Table 4.1: All considered inputs and how they varied the first day of the available data

UTC	Temp	Prec	WS	WD	GI	DI	DR	CC	P
	[°C]	[mm]	[m/s]	[°]	[W/m^2]	[W/m^2]	[W/m^2]	[%]	[kWh]
2021-01-01 08:00:00	0.1	0.09	5.5	23.9	0.0	0.0	0.0	8.0	0.0
2021-01-01 09:00:00	0.2	0.14	5.5	21.3	10.3	0.8	9.5	8.0	1.0
2021-01-01 10:00:00	0.4	0.21	5.4	20.3	27.0	1.3	25.7	8.0	13.0
2021-01-01 11:00:00	0.6	0.16	5.4	17.8	40.3	0.1	40.2	8.0	25.0
2021-01-01 12:00:00	1.0	0.08	5.2	21.0	61.6	11.9	49.7	8.0	29.0
2021-01-01 13:00:00	1.0	0.0	5.7	30.5	60.3	20.8	39.5	6.0	25.0
2021-01-01 14:00:00	0.5	0.0	5.4	24.6	22.7	7.3	15.4	4.0	16.0
2021-01-01 15:00:00	-0.5	0.0	5.7	17.4	0.0	0.0	0.0	4.0	0.0
2021-01-01 16:00:00	-0.8	0.0	6.3	19.5	0.0	0.0	0.0	7.0	0.0
2021-01-01 17:00:00	-0.6	0.0	6.3	21.7	0.0	0.0	0.0	8.0	0.0
2021-01-01 18:00:00	-0.1	0.0	6.5	21.1	0.0	0.0	0.0	7.0	0.0
2021-01-01 19:00:00	0.3	0.0	6.9	31.8	0.0	0.0	0.0	7.0	0.0
2021-01-01 20:00:00	0.3	0.0	6.6	32.9	0.0	0.0	0.0	8.0	0.0

Table 4.2 shows the production at t, t-1, t-2, t-3, and t-4, between 08:00 and 20:00 on the first day of the recorded data to visualize what time lags in the data-frames look like.

Table 4.2: Production at t, t-1, t-2, t-3 and t-4, between 08:00 and 20:00 the first day of the data-frame.

UTC	Production	lag1	lag2	lag3	lag4
	[kWh]	[kWh]	[kWh]	[kWh]	[kWh]
2021-01-01 08:00:00	0.0	0.0	0.0	0.0	0.0
2021-01-01 09:00:00	1.0	0.0	0.0	0.0	0.0
2021-01-01 10:00:00	13.0	1.0	0.0	0.0	0.0
2021-01-01 11:00:00	25.0	13.0	1.0	0.0	0.0
2021-01-01 12:00:00	29.0	25.0	13.0	1.0	0.0
2021-01-01 13:00:00	25.0	29.0	25.0	13.0	1.0
2021-01-01 14:00:00	16.0	25.0	29.0	25.0	13.0
2021-01-01 15:00:00	0.0	16.0	25.0	29.0	25.0
2021-01-01 16:00:00	0.0	0.0	16.0	25.0	29.0
2021-01-01 17:00:00	0.0	0.0	0.0	16.0	25.0
2021-01-01 18:00:00	0.0	0.0	0.0	0.0	16.0
2021-01-01 19:00:00	0.0	0.0	0.0	0.0	0.0
2021-01-01 20:00:00	0.0	0.0	0.0	0.0	0.0

As the table shows, the values from the previous column is shifted one hour for each lag.

A subplot was made to show the difference between Pearson-, Kendall- and Spearman correlation, where the straight blue dashed lines are placed at the correlation coefficient of 0.5, to show what variables correlate well enough. All variables correlate differently depending on how the correlation is measured.

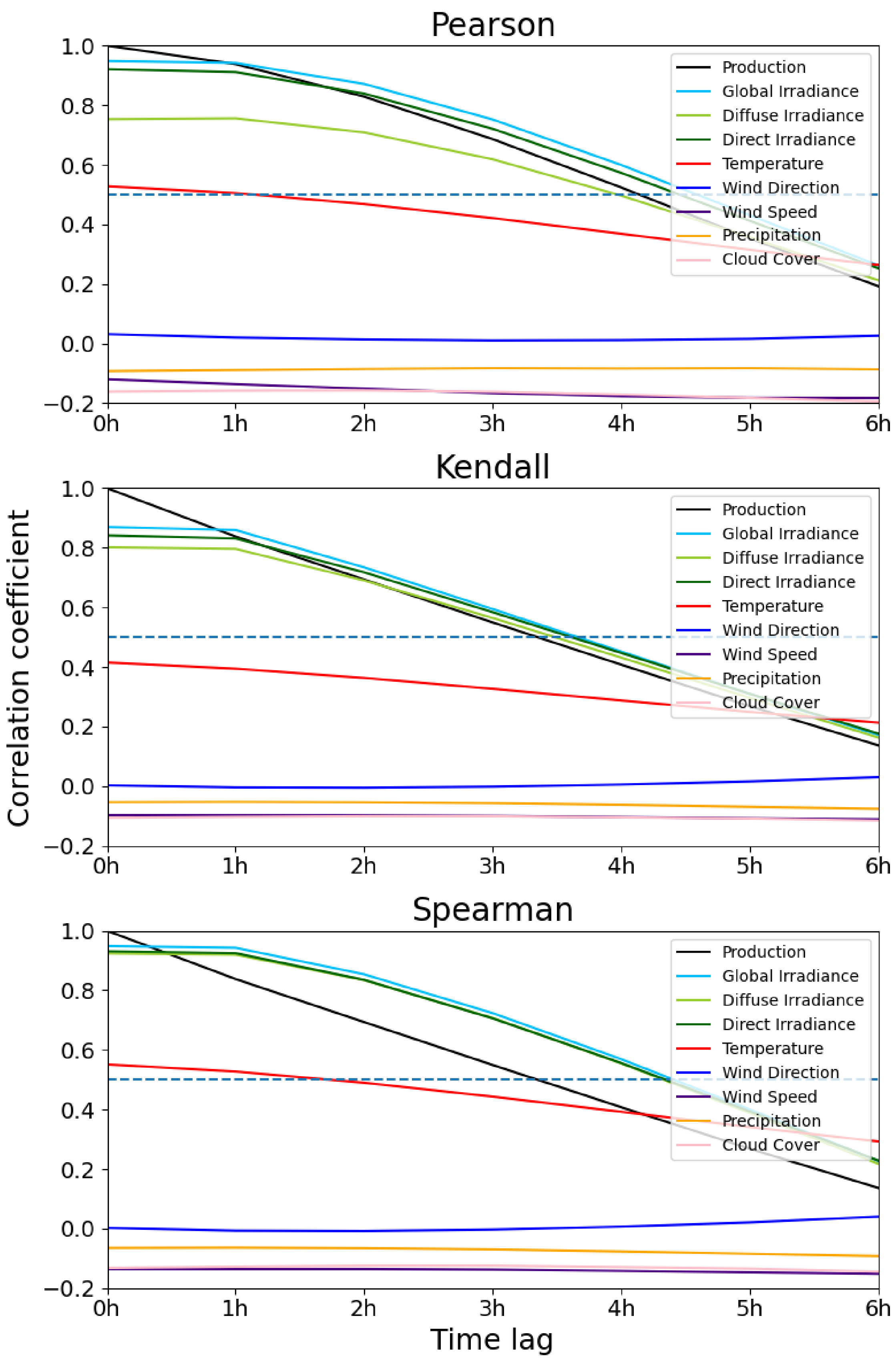


Figure 4.6: The correlation subplot from using Pearson, Kendall and Spearman correlation. For each time-lag meaning each past hour, the relevant variables correlate less.

Table 4.3 shows the difference between Pearson (P)-, Kendall (K)- and Spearman (S) Correlation, where the numbers are the correlation coefficient, representing the correlation between the features at time $t - i$ and the production at time t. Only correlation coefficients larger than 50% with the production are shown in this table as these got selected as inputs. The Spearman method included the highest number of possible features since with this method, the highest number of features has correlation coefficients larger than 50%. Rad is short for radiation.

Table 4.3: All variables having correlation coefficients larger than 50% with the production, showing their correlation coefficients for all three types. P is Pearson, K is Kendall and S is Spearman.

	P	K	S
Temp [°C]	0.525	—	0.546
Global Rad [W/m^2]	0.941	0.861	0.939
Direct Rad [W/m^2]	0.914	0.833	0.922
Diffuse Rad [W/m^2]	0.748	0.793	0.915
Production [kWh]	1.000	1.000	1.000
Production lag1 [kWh]	0.936	0.837	0.928
Temp lag1 [°C]	0.502	—	0.522
Global Rad lag1 [W/m^2]	0.935	0.851	0.934
Direct Rad lag1 [W/m^2]	0.904	0.824	0.916
Diffuse Rad lag1 [W/m^2]	0.751	0.789	0.911
Production lag2 [kWh]	0.825	0.691	0.816
Global Rad lag2 [W/m^2]	0.866	0.728	0.847
Direct Rad lag2 [W/m^2]	0.833	0.711	0.828
Diffuse Rad lag2 [W/m^2]	0.705	0.684	0.827
Production lag3 [kWh]	0.683	0.548	0.675
Global Rad lag3 [W/m^2]	0.748	0.590	0.718
Direct Rad lag3 [W/m^2]	0.717	0.579	0.700
Diffuse Rad lag3 [W/m^2]	0.617	0.560	0.702
Production lag4 [kWh]	0.523	—	0.516
Global Rad lag4 [W/m^2]	0.598	—	0.565
Direct Rad lag4 [W/m^2]	0.571	—	0.551
Diffuse Rad lag4 [W/m^2]	—	—	0.551

All models used one or more NWP variables as inputs. It was just the ARIMA model that used only one, the global radiation at time t as explained in 3.4. For the ANN model using only NWP variables as inputs, all available NWP variables at time t were used, meaning all variables shown in Table 4.1, except the production. For all other models, the inputs were all the variables shown in Table 4.3, with as many correlated time-lags as also shown in the table. The ARIMA model also used the production at time t as input.

4.2 Multilayer Perceptron

The results from performing GridSearchCV are as follows. The number of hidden layers is 2 and the number of hidden neurons is 14. The activation function is SELU and Lecun uniform initializer. The batch size is 64 and the optimizer is RMSProp. As mentioned in the methodology section 3.7, two searches were done for each of the ANN models. The first grid search took 39 minutes to complete, while the second only took 4 minutes to complete. For all models that used an ANN architecture, 100 epochs were used as it was noticed that early stopping happened almost always before all epochs were used.

Figure 4.7 displays the predicted and the actual power production in August 2021, predicted by the MLP model, while Figure 4.8 presents the same but only from August 6 to August 16, 2021, and is presented for better visualization of the difference/error. The MLP model is further referred to as the ANN model.

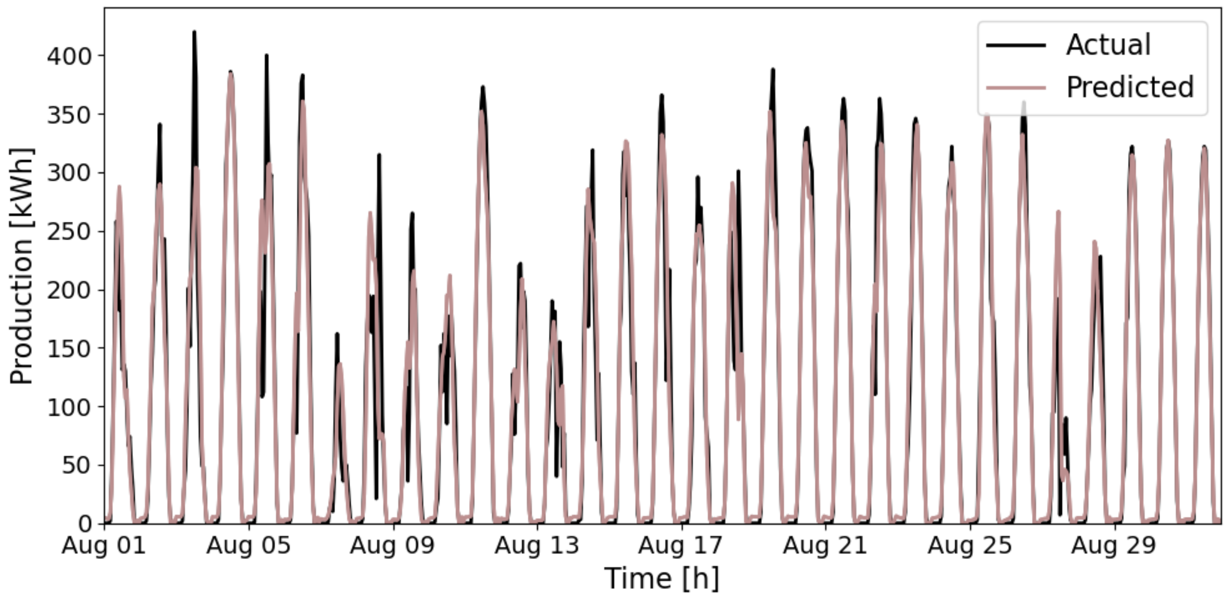


Figure 4.7: The predicted power production by the ANN model and the actual production in August 2021.

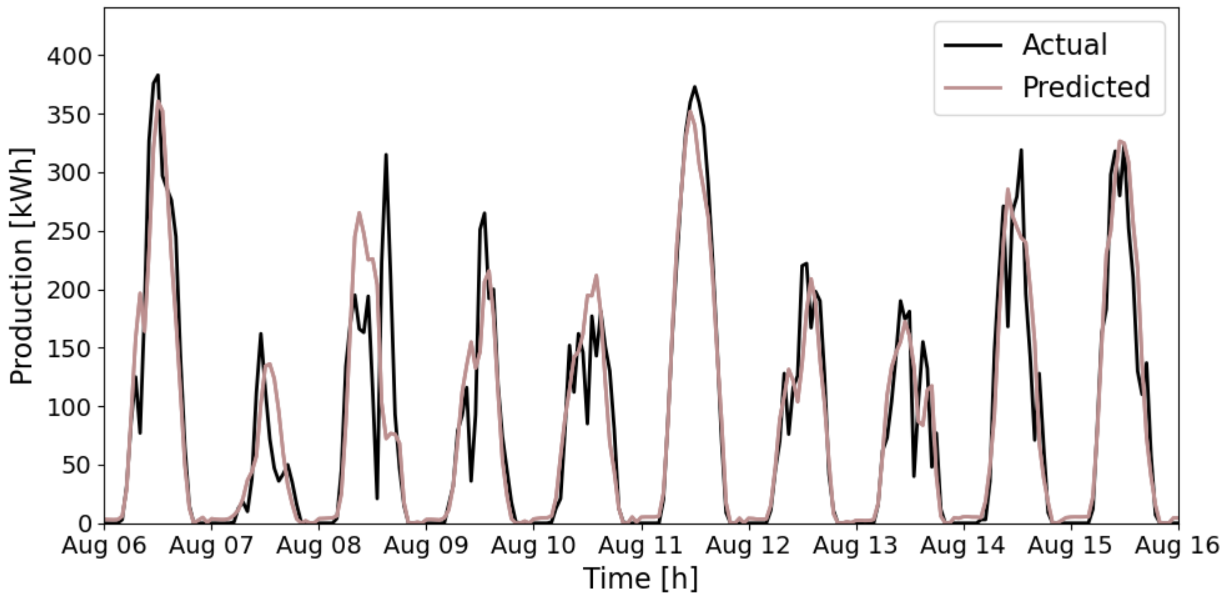
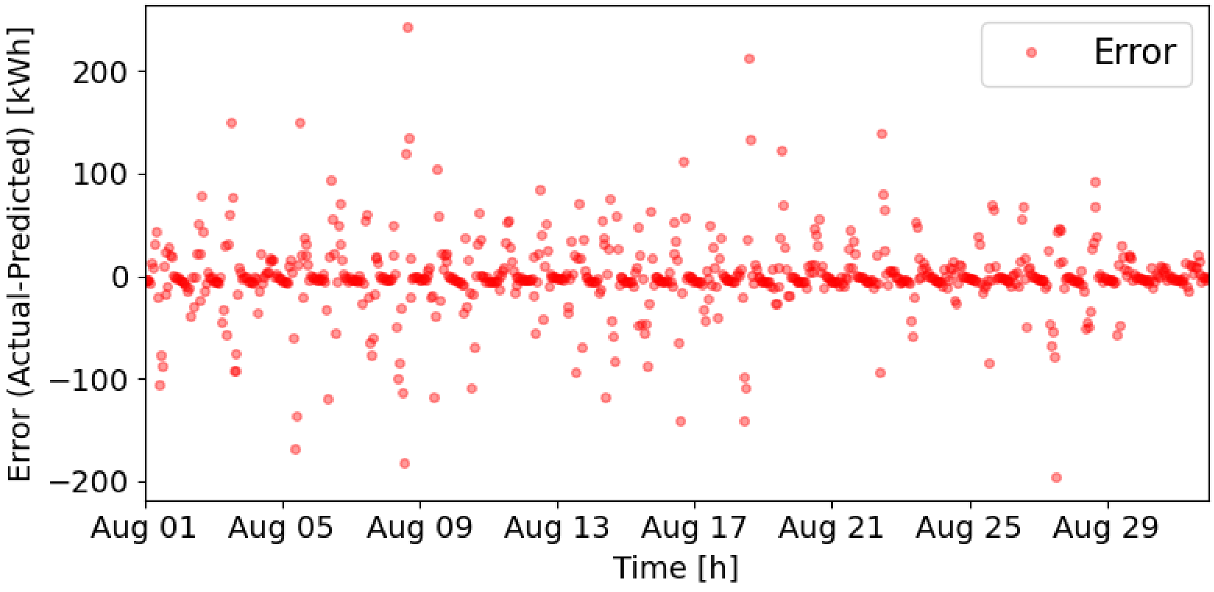
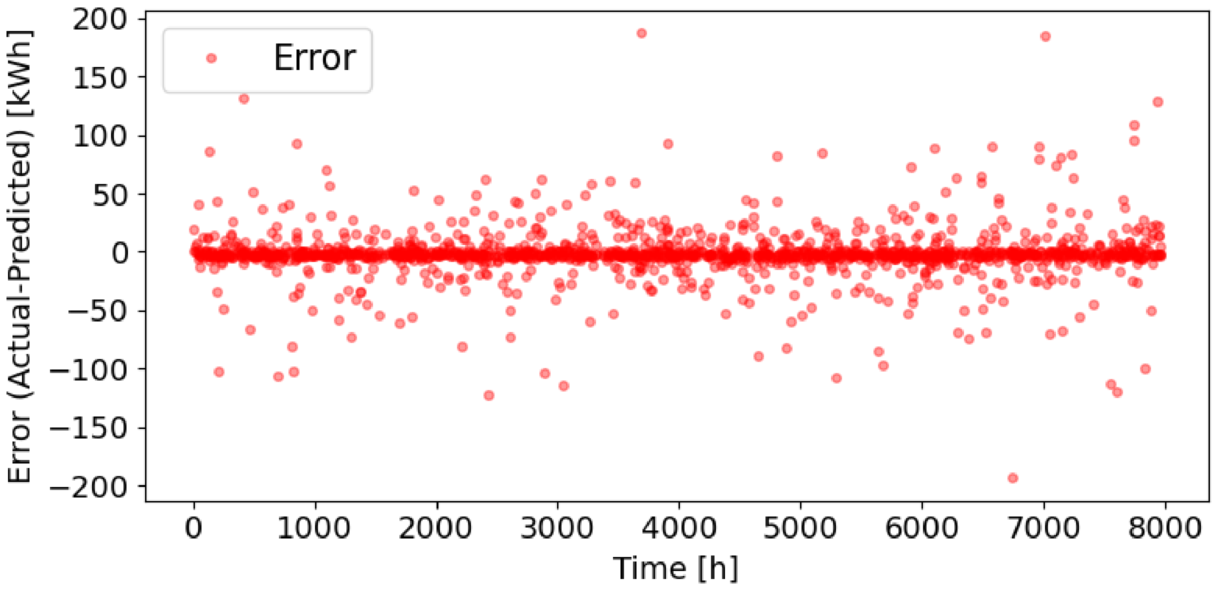


Figure 4.8: The predicted power production by the ANN model and the actual production from August 6 to August 16, 2021.

Figure 4.9 visualizes the errors between both the actual production in the random test data and the predicted power production (sub-figure 4.9a) and the actual production in August and the predicted power production (sub-figure 4.9b).



(a) August test.



(b) Random test.

Figure 4.9: The error between the actual- and predicted power production from the ANN model for random test and August test.

It is clear from these two plots, that the fit on the random test is better, as there is a higher percentage of smaller errors. The biggest error is also measured on the August test.

Figure 4.10 shows a plot of the results for 30 fits (runs) on the same ANN model with no variations, and Figure 4.11 shows the same results, but for only one day in August to even better visualize the difference between the results. In the last figure, one can easily spot the slight difference in each fit, where there is quite a variation between the best and worst fit. It is this variation that was the motivation for making an ANN ensemble, combining different fits of the same model with the same settings and parameters.

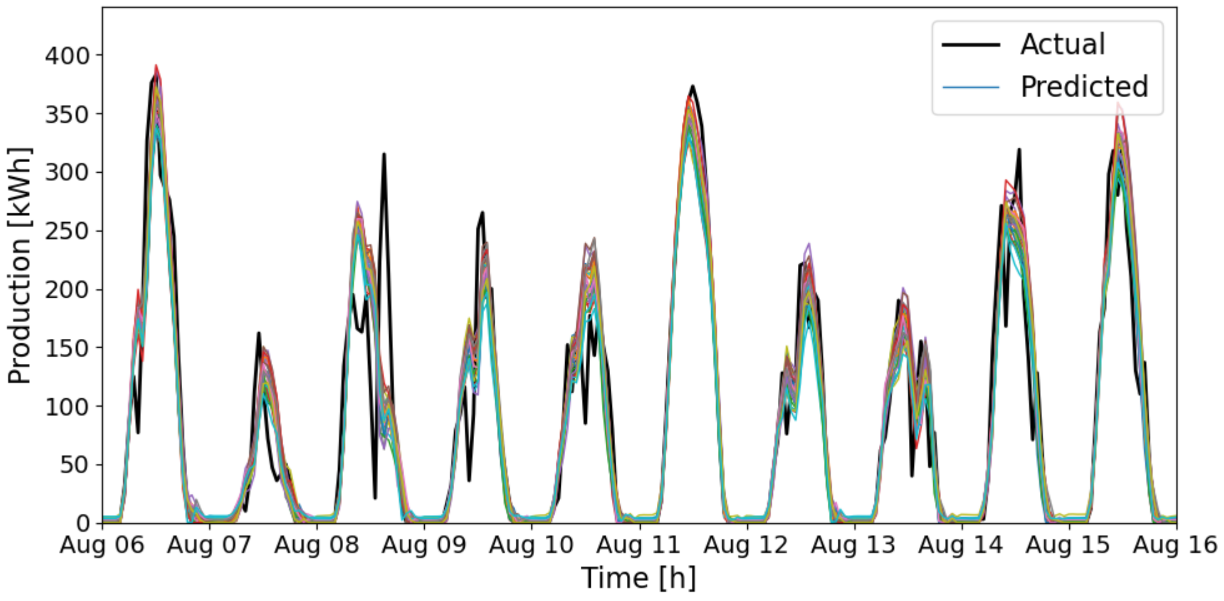


Figure 4.10: The predicted power production by the ANN model's 30 fits and the actual production from August 6 to August 16, 2021. Each line is one fit.

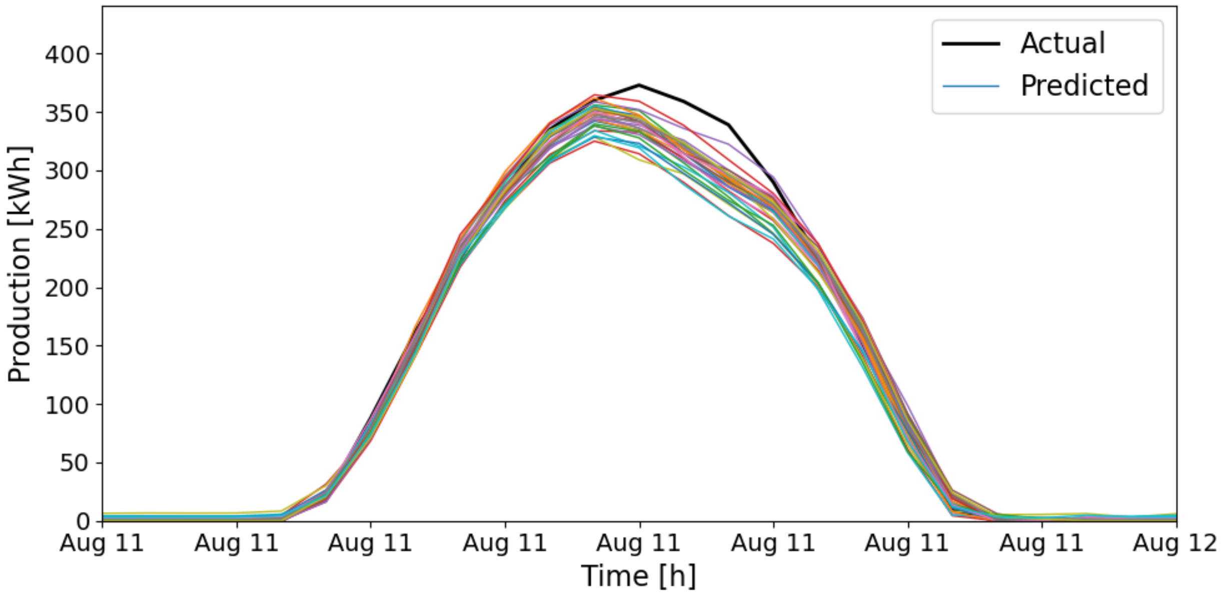


Figure 4.11: Only one day in August of the predicted power production by the ANN model's 30 runs and the actual production. Each line is one fit.

4.2.1 Only numerical weather predictions as inputs

Another GridSearchCV was performed, as the ANN model using only NWPs as inputs, has just that, other inputs, and the results are as follows. The number of hidden layers is 2 and the number of hidden neurons is 14. The activation function is SELU and the initializer is Lecun uniform. The batch size is 32 and the optimizer is RMSProp. The first grid search took 26 minutes to complete, while the second only took 5 minutes to complete.

Figure 4.12 and 4.13 displays the predicted and actual production for the whole of August month and between August 6 and August 16, respectively predicted by the ANN model using only NWPs as inputs.

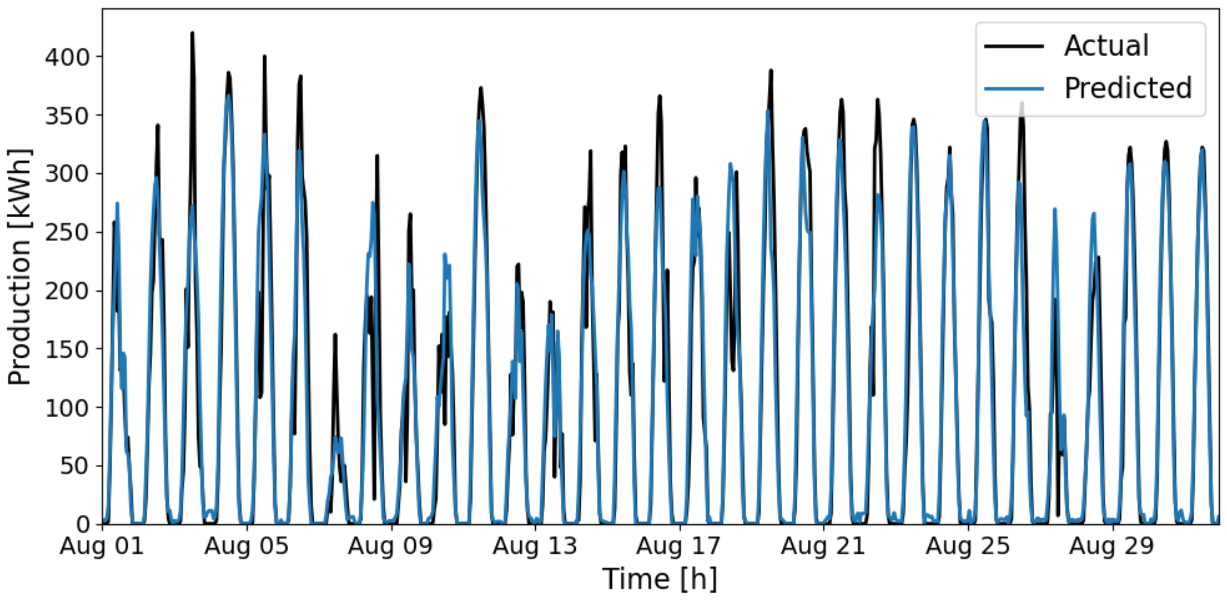


Figure 4.12: The predicted power production by the ANN model trained only on NWPs and the actual production.

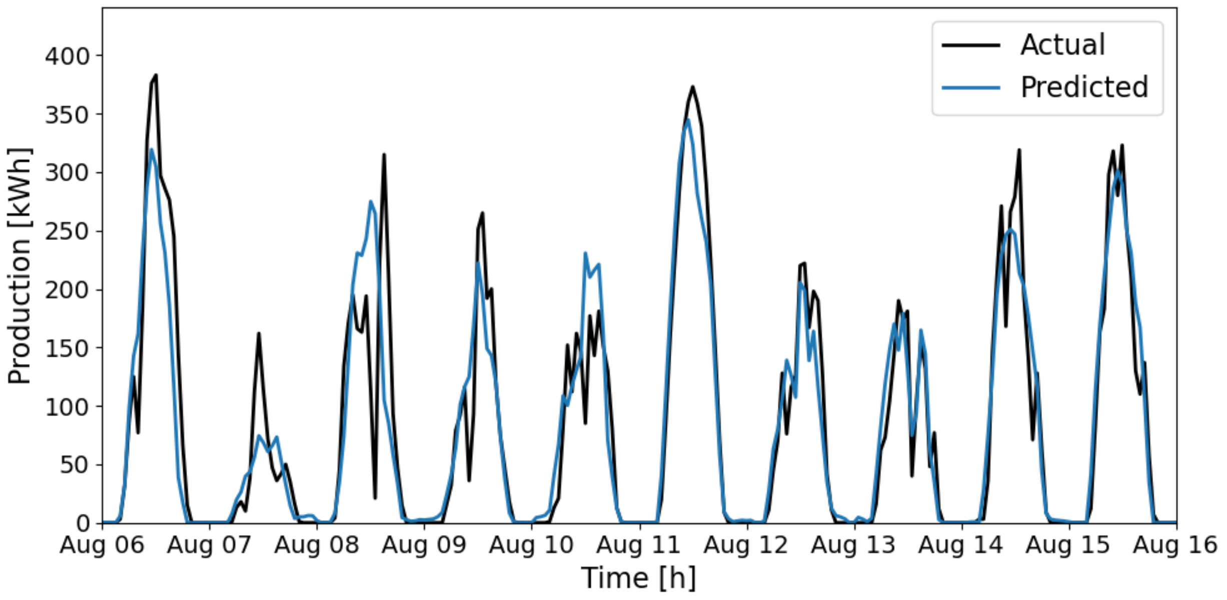
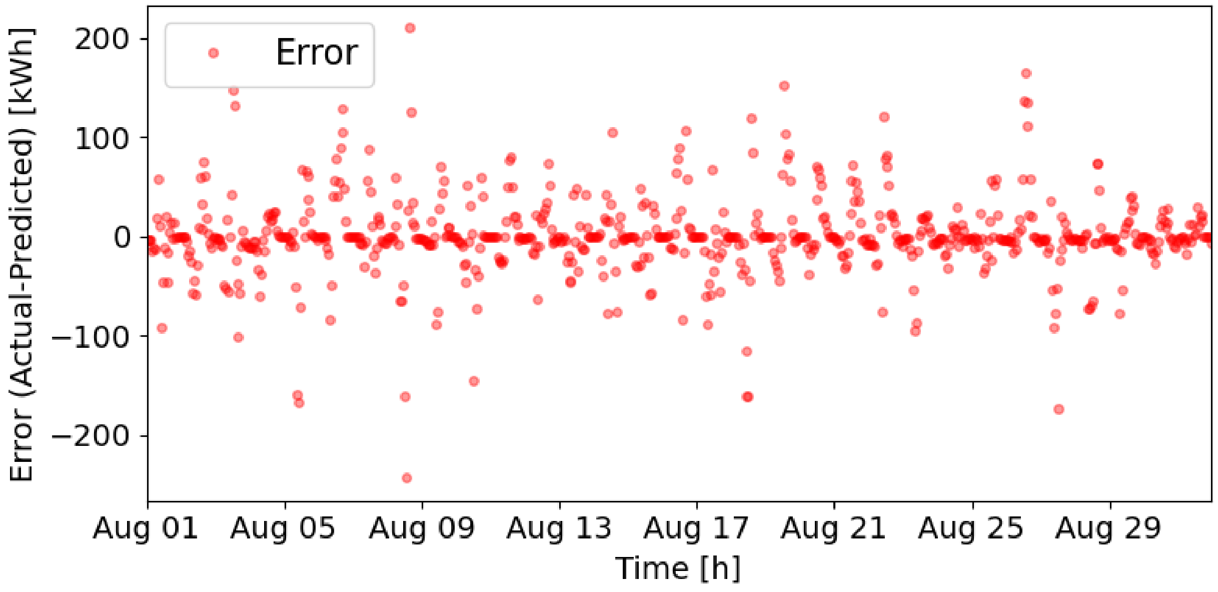


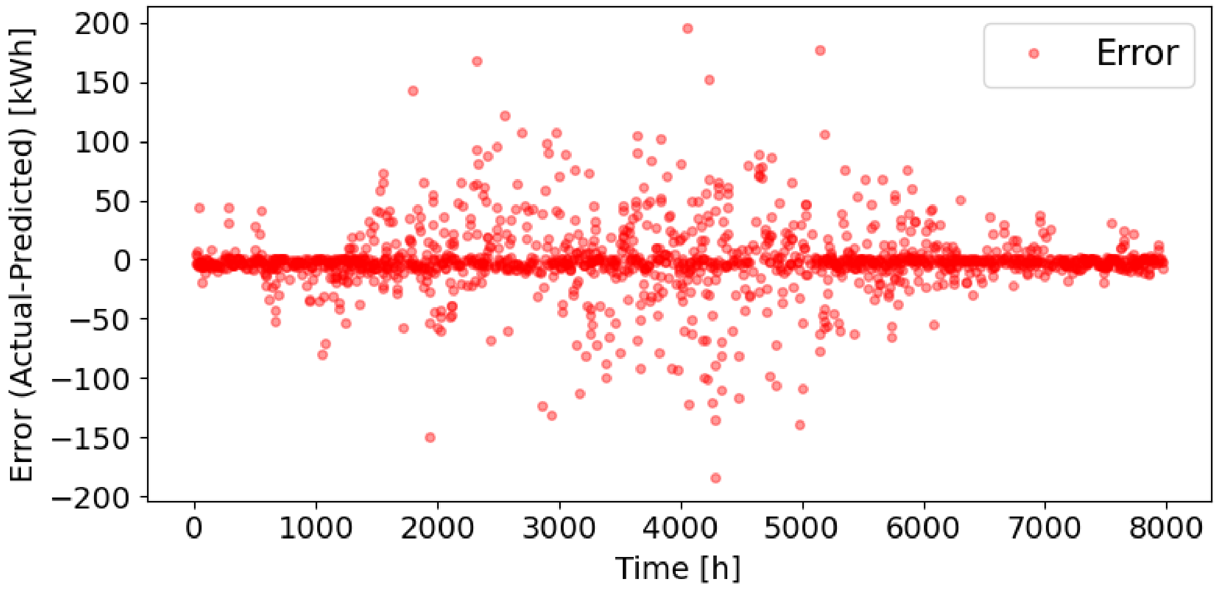
Figure 4.13: A plot of the predicted power production by the ANN model trained only on NWPs and the actual production for August 6 to August 16, 2021.

The figures display a fit that is quite good, considering the model was not trained on any actual production. As expected, the model misses more of the actual production than the ANN model. However, this seems to not be the case for August 12 and 13.

Figure 4.14 shows all errors at each time step for both test data, where Figure 4.14a and Figure 4.14b shows the error for August test and random test, respectively. Comparing the error figures on



(a) August test.



(b) Random test.

Figure 4.14: The error between the actual- and predicted power production from the ANN model trained only on NWPs for random test and August test.

the August test between the ANN model and the ANN model using only NWPs, they both show large measured errors around August 9, August 19, and August 27. On the random test, however, the large errors were not measured around the same hours.

4.3 Support Vector Regression

The SVR model was made for comparison, and to be in an ensemble of ANN and SVR. For a thorough review of the results with a similar SVR model used on only daylight hours as well as all hours, please refer to the pre-project [55]. This comparison showed a low difference and therefore, comparisons were not included in this work.

The resulting hyperplane parameters from GridSearchCV are a C of 9, a ε of 0.05, and a γ of 0.01. Figure 4.15 displays the predicted and the actual power production in August 2021, and Figure 4.16 presents the predicted and actual production from August 6 to August 16, 2021. The last figure is presented to better visualize the performance.

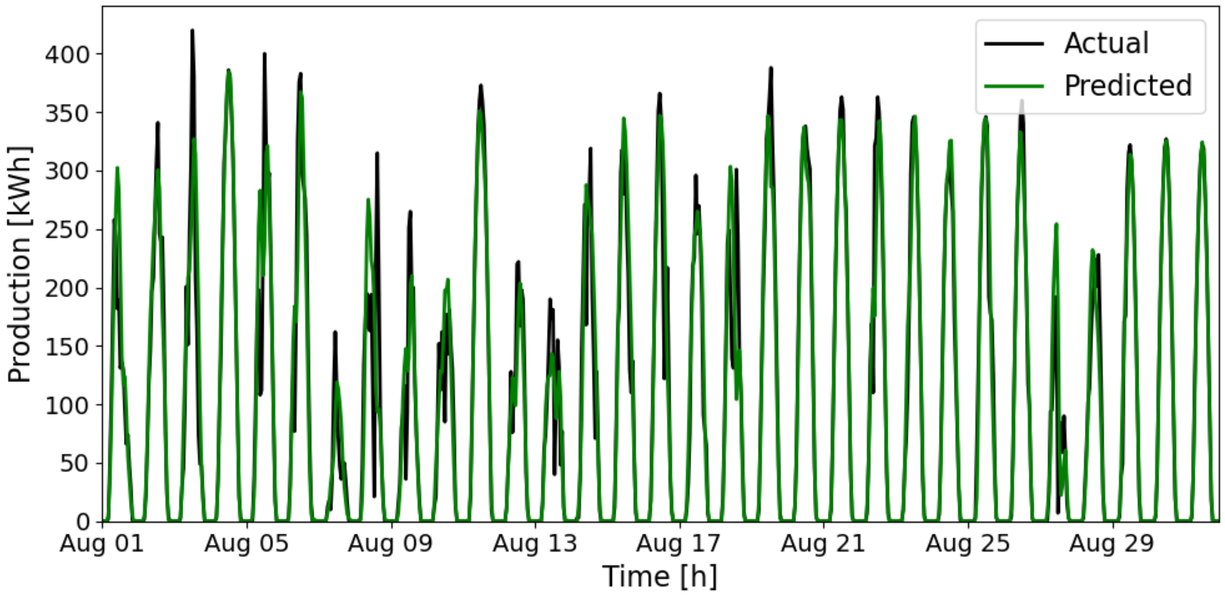


Figure 4.15: The predicted power production by the SVR model and the actual production.

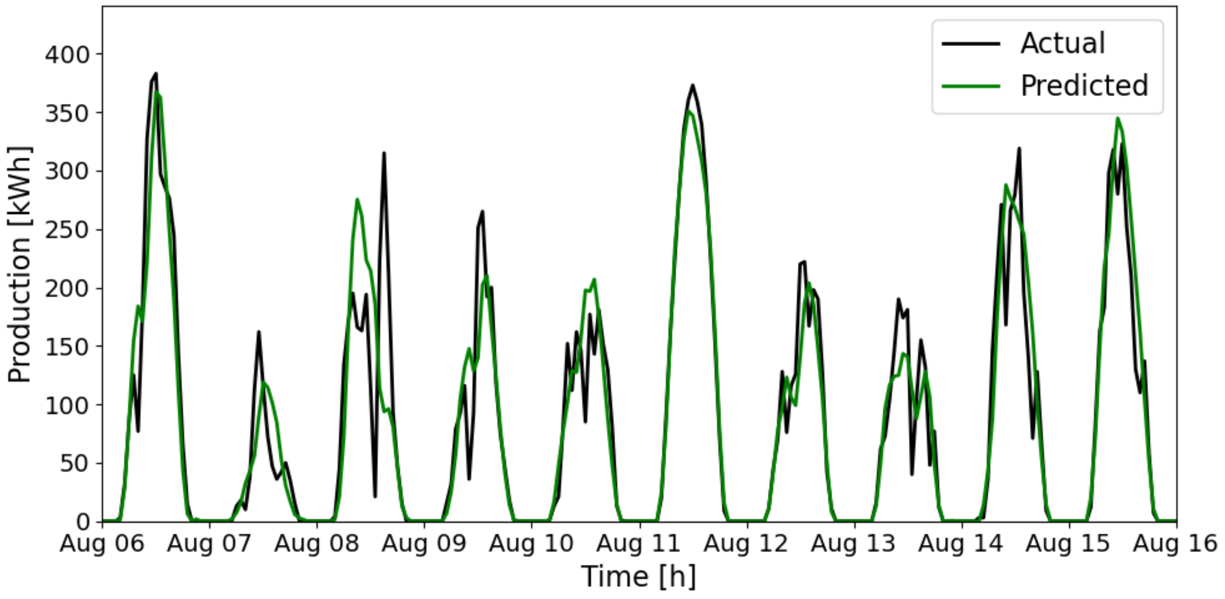
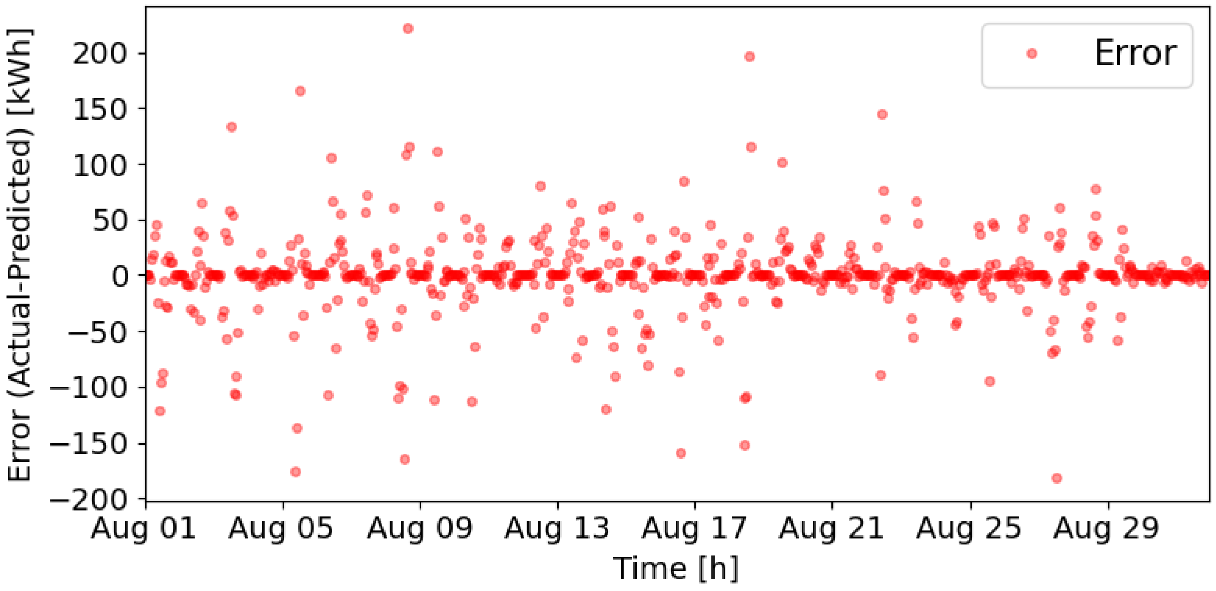


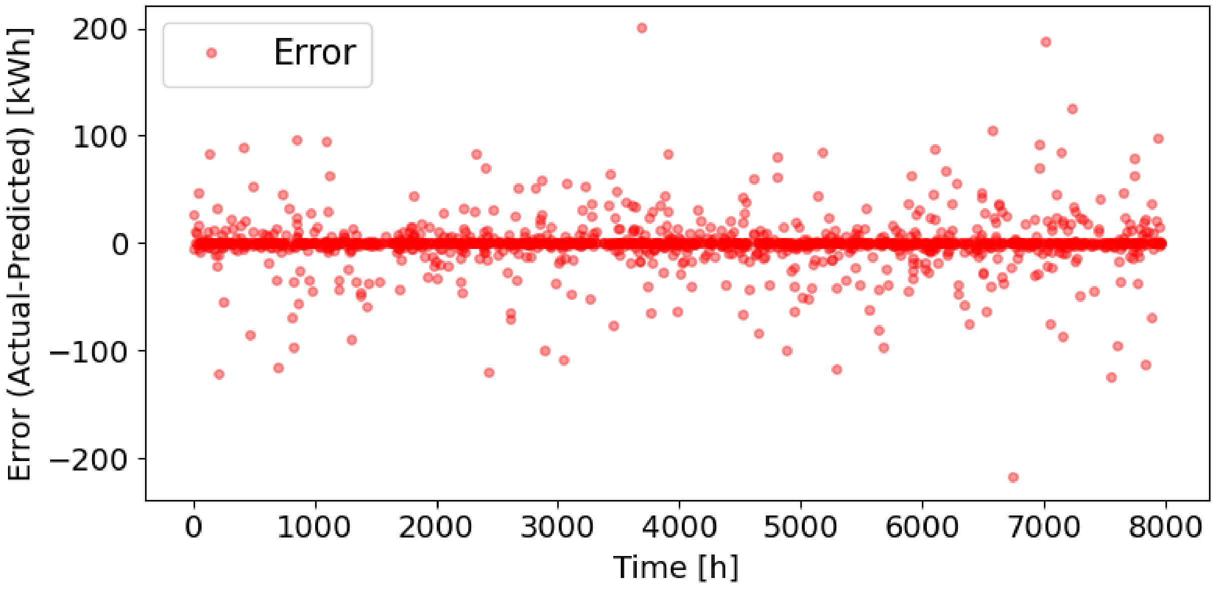
Figure 4.16: A plot of the predicted power production by the SVR model and the actual production, from August 6 to August 16, 2021.

Looking at Figure 4.8 and Figure 4.16, it seems that the ANN model predicts more precisely than the SVR model, while comparing Figure 4.7 and Figure 4.15, it seems that the SVR model predicts more accurately on the last 17 days of August.

Figure 4.17 visualizes the errors between both the actual production in the random test data and the predicted power production, and the actual production in August and the predicted power production for each hour in August and each hour in random test. Sub-figure 4.17a shows the errors between predicted and actual production in August, while the other sub-figure 4.17b shows the same but for the random test.



(a) August test.



(b) Random test.

Figure 4.17: The error between the actual- and predicted power production from the SVR model for random test and August test.

Compared to the errors of the ANN model, the errors of the SVR model seem to be around the same time and hours.

4.4 Auto-Regressive Integrating Moving Average

The ARIMA model was made for comparison with a statistical model. Figure 4.18 displays the models' predicted and the actual power production in August 2021. Figure 4.19 presents the models' predicted and the actual production from August 6, 2021, to August 16, 2021, and is presented for better visualization of the variation between actual and predicted.

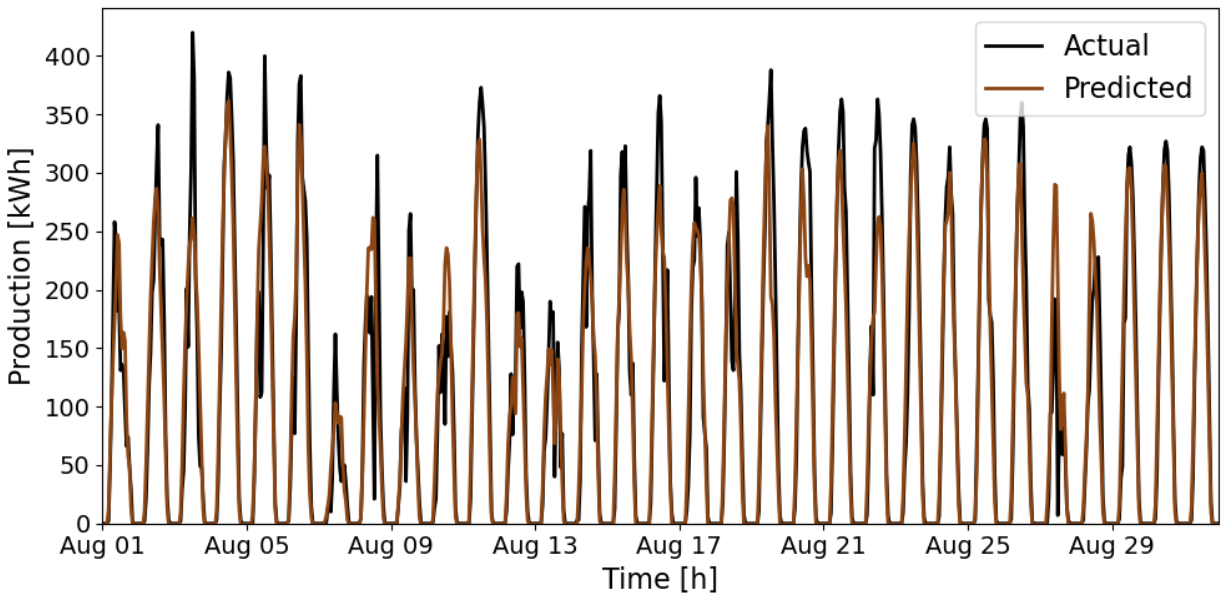


Figure 4.18: The predicted power production by the ARIMA model and the actual production.

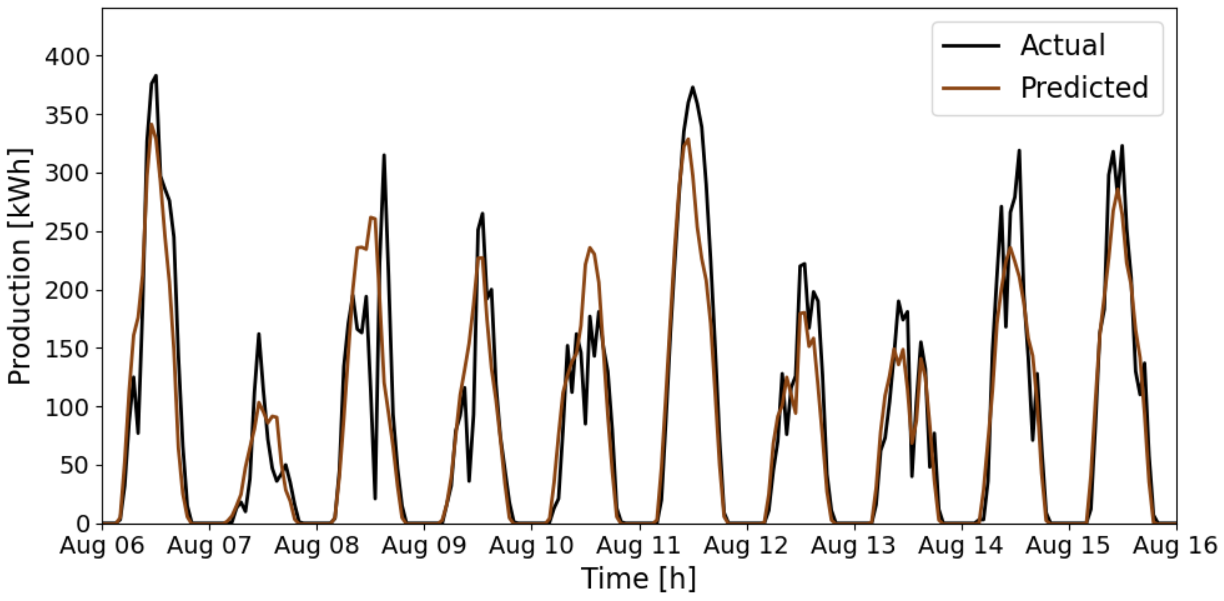


Figure 4.19: A plot of the predicted power production by the ARIMA model and the actual production, from August 6 to August 16, 2021.

By comparing the figures of the actual and the predicted production by both the SVR model and the ANN model, it is clear that the ARIMA model does not perform as correctly but still manages to predict the variation between days and hours. The ARIMA model, however, seems to predict August 12 and 13 a little more accurately than both the SVR and the ANN models.

Figure 4.20 visualizes the errors between the actual production in August and the predicted power production.

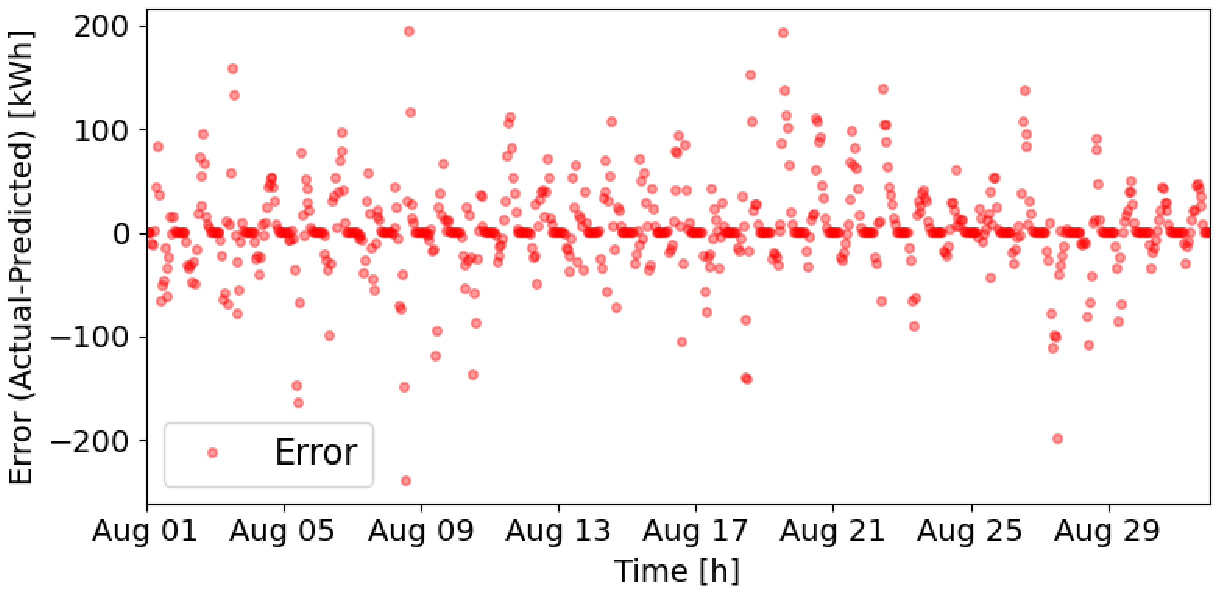


Figure 4.20: The error between the actual- and predicted power production from the SVR model for August test.

Comparing all standalone deterministic models, they all got large errors around August 9, 19, and 27.

4.5 Ensemble models

The same setting and parameters as used for the standalone ANN model are used for the first base learner of the ensemble. For the second base learner, GridSearchCV was used again. From the first search, the number of hidden layers is 2, the number of hidden neurons is 12 and the activation function is SELU. From the second search, the batch size is 64, the initializer is Lecun uniform, and the optimizer is RMSProp. The first search took 25 minutes to complete, while the second only took 4 minutes to complete. As the difference is only the number of hidden neurons, it was decided to use different parameters. The number of hidden layers is still 2, but the activation function was set to RELU, with HE uniform initializer and the Adam optimizer. The batch size was also unchanged.

The forecasted power production by the SVR and ANN ensemble model and the actual power production in August 2021, is shown in Figure 4.21 and 4.22, where the latter only shows the forecasted and actual production from August 6 to August 16, 2021.

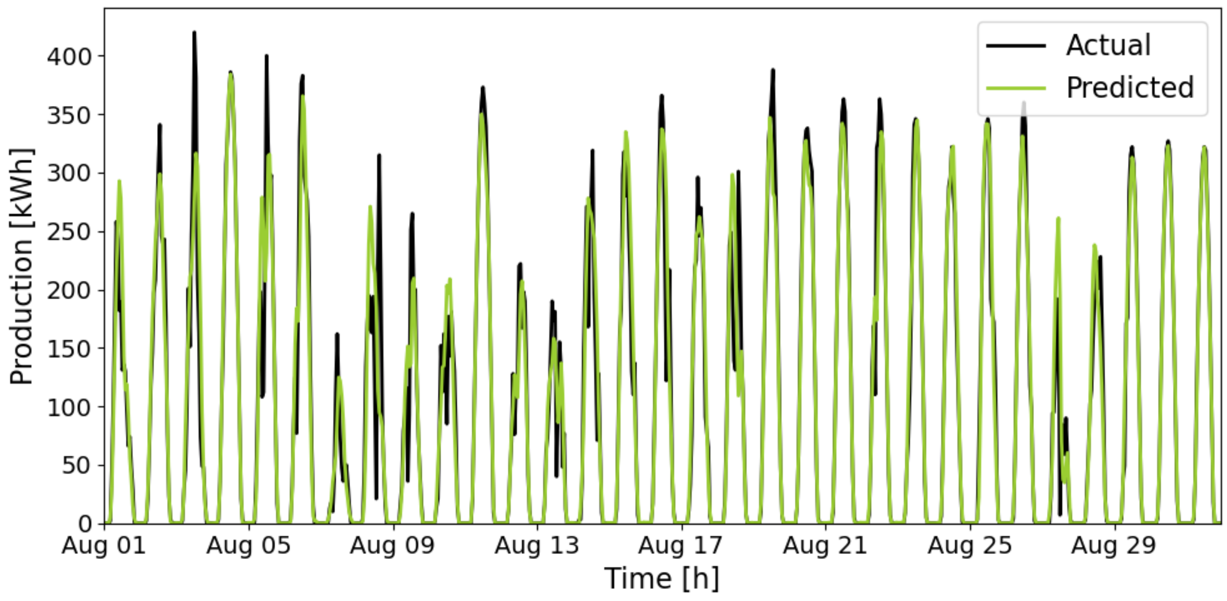


Figure 4.21: The predicted power production by the SVR and ANN ensemble model and the actual production.

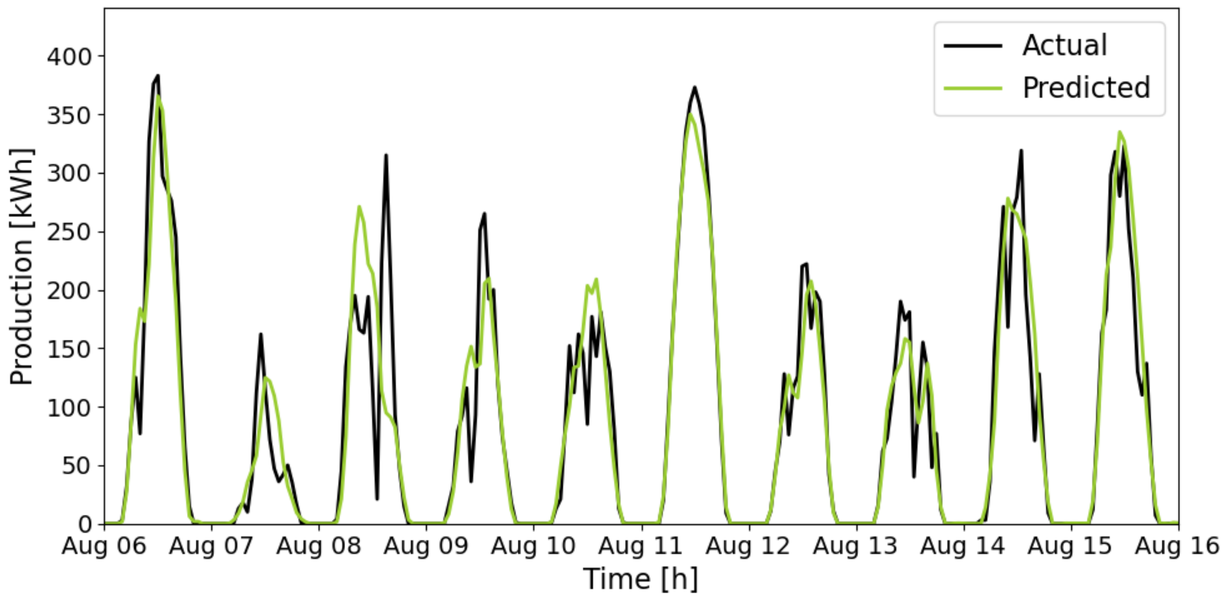
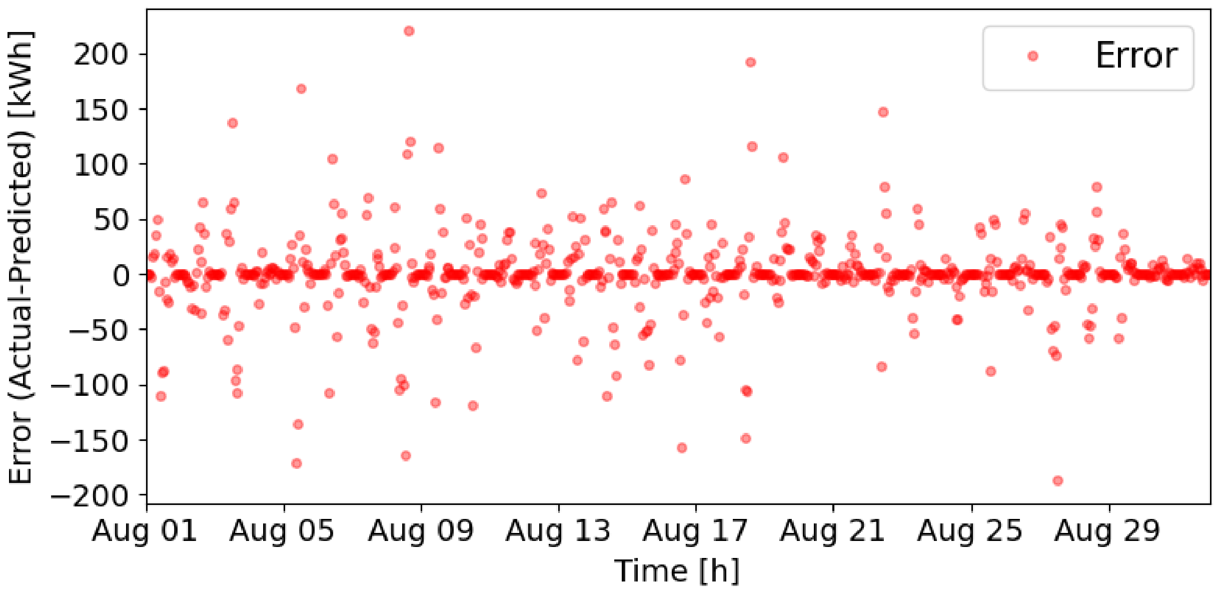


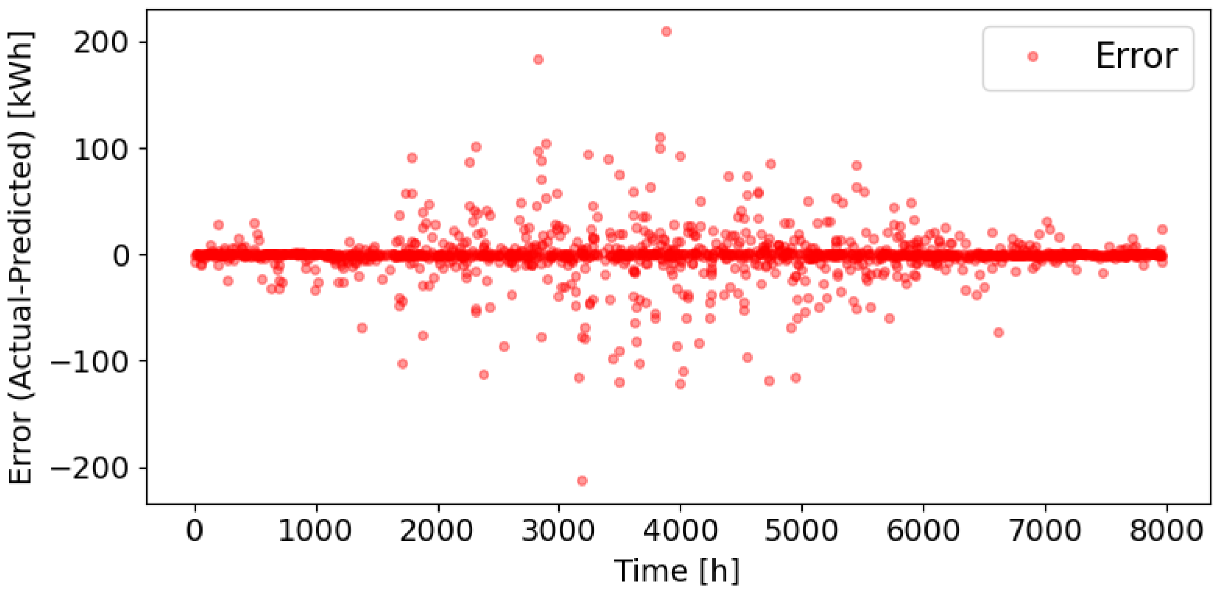
Figure 4.22: The predicted power production by the SVR and ANN ensemble model and the actual production from August 6 to August 16, 2021.

It might not be easy to spot, but it seems like this ensemble model predicts slightly better than both the SVR and the ANN model, by in a way, using the predicted value from the SVR model when it predicted the hour better than the SVR model and the other way around when the ANN model predicted better.

Figure 4.23 visualizes the errors between both the actual production in the random test data and the predicted power production and the actual production in August and the predicted power production. Sub-figure 4.23a shows the errors between the actual production in August and the predicted power production and sub-figure 4.23b shows the other mentioned errors.



(a) August test.



(b) Random test.

Figure 4.23: The measured error between the actual- and predicted power production from the SVR and ANN ensemble model for random test and August test.

As expected, the errors are measured around the same times and hours as for both the SVR and the ANN model.

The predicted production done by the ANN ensemble model of 15 fits of one ANN model and 15 fits of another, for August 2021 is displayed together with the actual production in Figure 4.24. To give a clearer image of the difference between the predicted- and actual power production Figure 4.25 is included, showing August 6 to August 16.

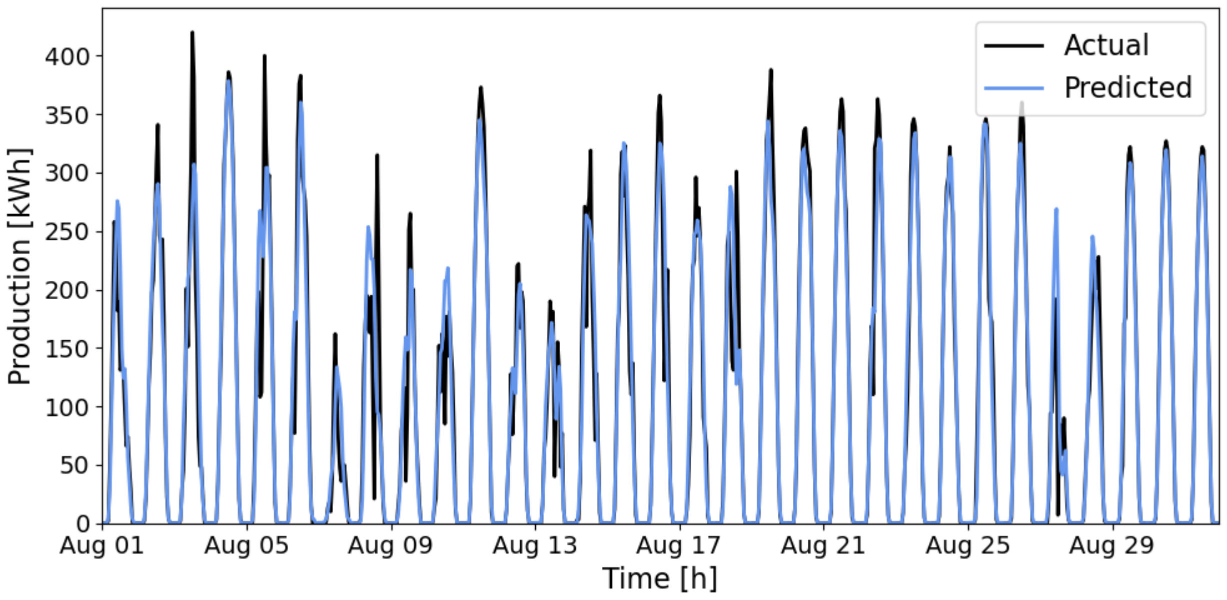


Figure 4.24: The predicted power production by the ANN ensemble model and the actual production.

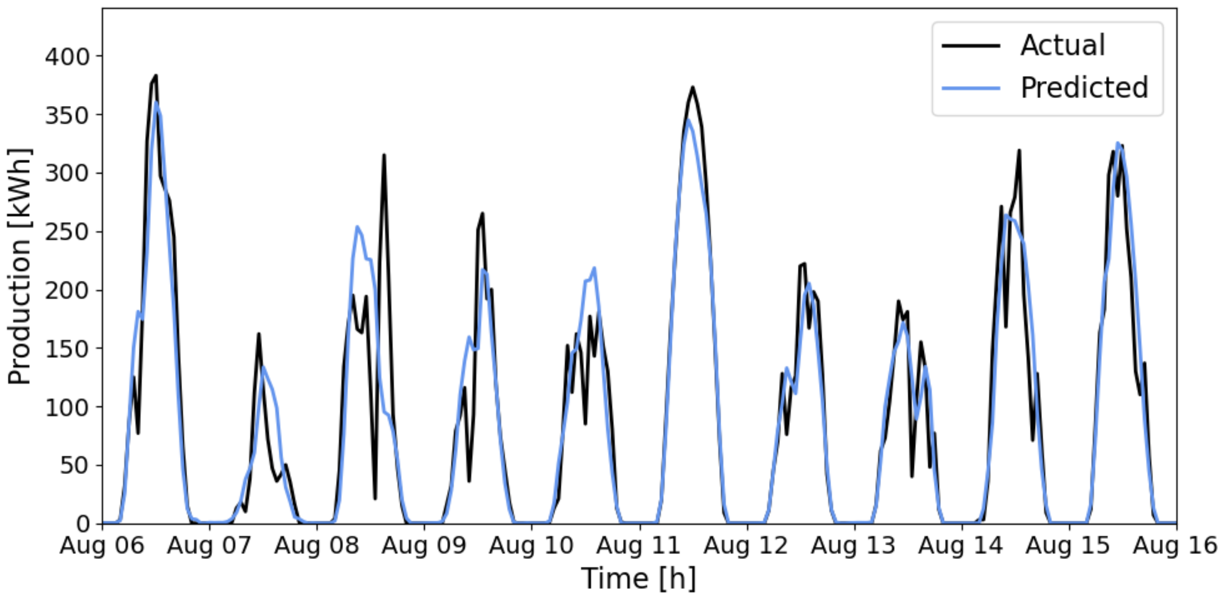
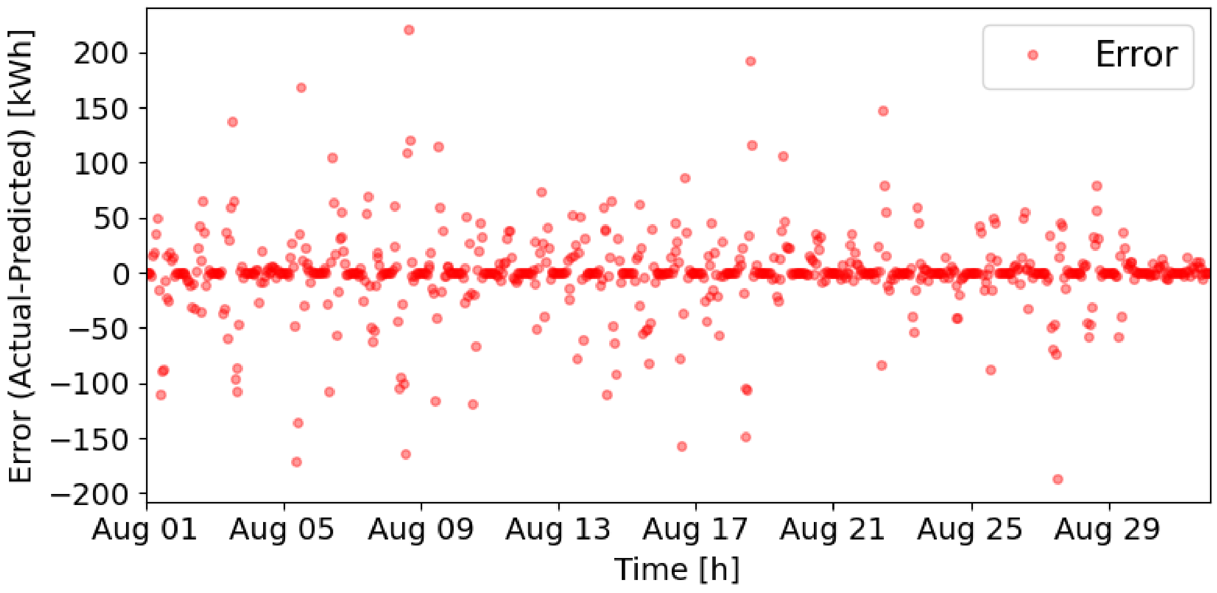
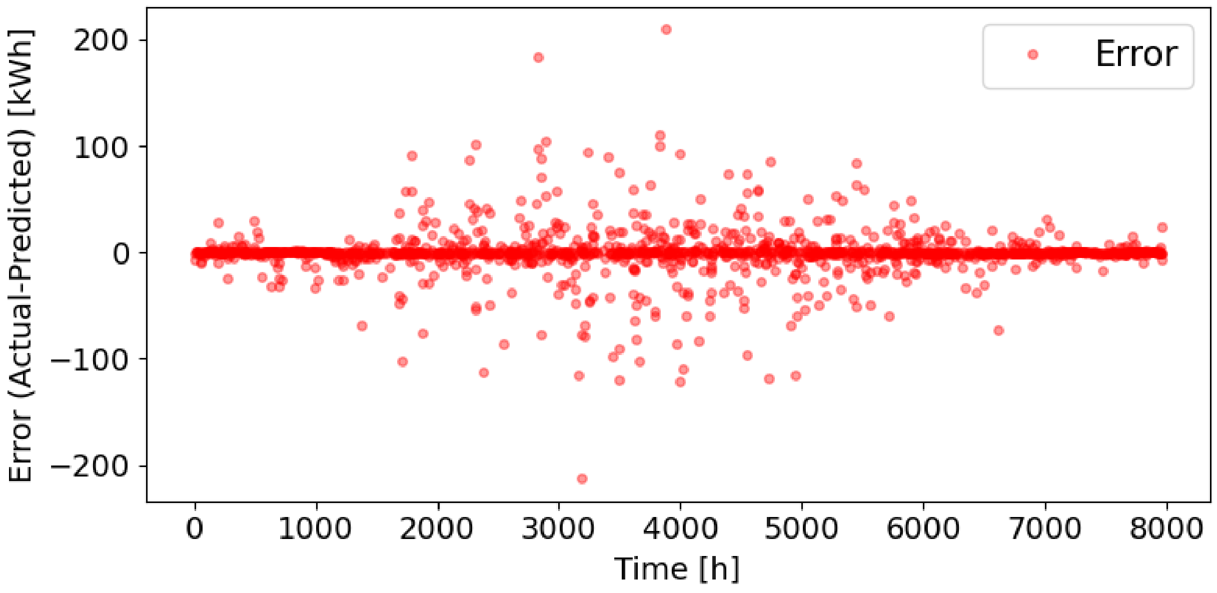


Figure 4.25: A plot of the predicted power production by the ANN ensemble model and the actual production, from August 6 to August 16, 2021.

The errors in prediction done by the ANN ensemble are shown in Figure 4.26, where Sub-figure 4.26a shows the errors in the predictions for August and Sub-figure 4.26b shows the errors for the random test.



(a) August test.



(b) Random test.

Figure 4.26: The error between the actual- and predicted power production from the ANN ensemble model for random test and August test.

Comparing all deterministic models, both standalone and ensembles, all models except the ANN model trained on only NWPs had large measured hours around the same times and hours for August 2021 and random test, meaning the results of the ensembles might have been better or worse if one learner were predicting small errors where the ones in this thesis did not.

4.6 Interval Prediction - Quantile Regression

The quantile regression neural network model's ability to predict an interval (between the 5th and 95th percentile) where the actual production lies within, is illustrated in Figure 4.27 and Figure 4.28, where the latter only shows 2 days of August, to better illustrate the ability of the model.

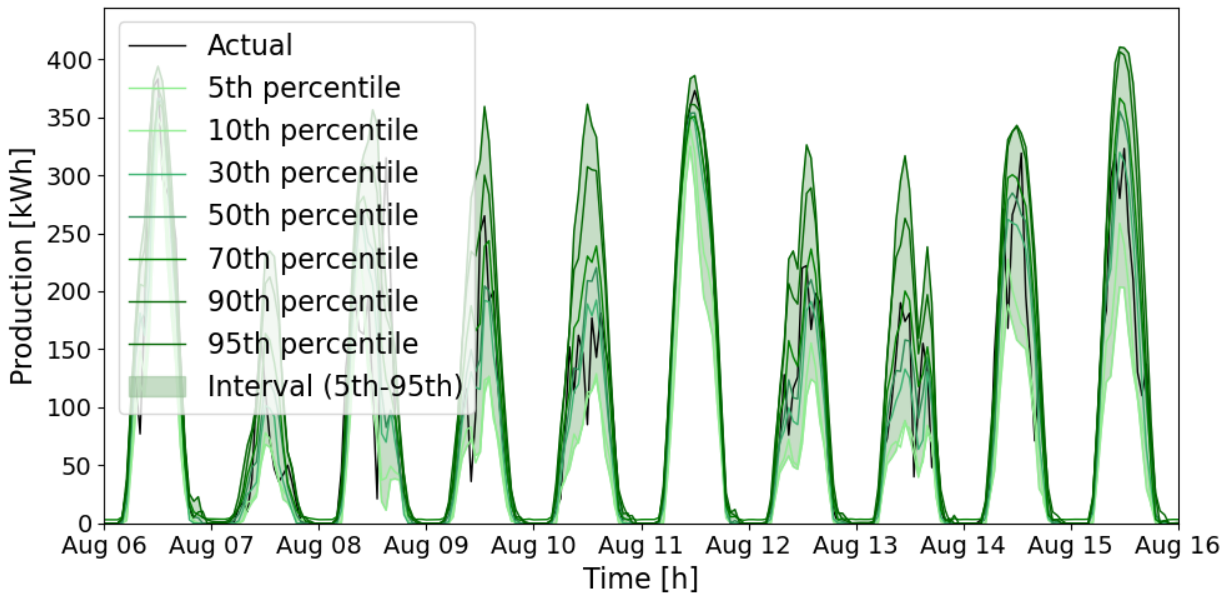


Figure 4.27: A plot of the predicted power production by the QRNN model and the actual production from August 6 to August 16, 2021, with its percentiles and prediction interval. The 50th percentile represents the point prediction from the model.

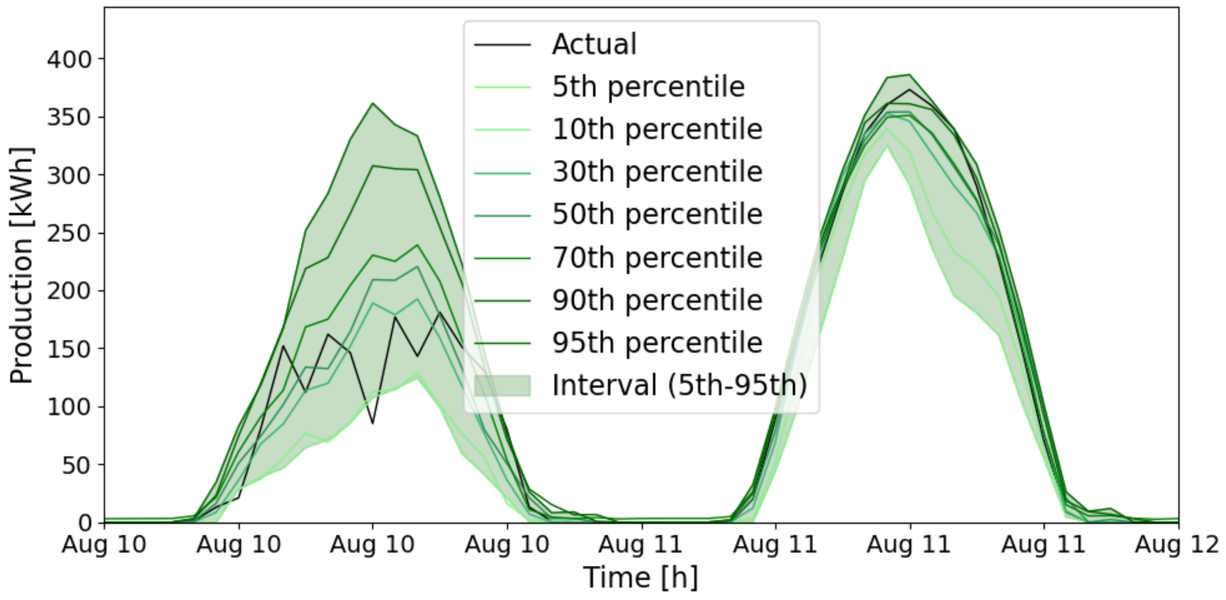


Figure 4.28: A plot of the predicted power production by the QRNN model and the actual production from August 10 to August 12, 2021, with its percentiles and prediction interval. The 50th percentile represents the point prediction from the model.

The same illustration of its ability, but on the random test data is shown in Figure 4.29. This illustration shows its ability only on some of the hours of the random test data because for all hours, all lines and intervals lie so close to each other it became hard to see the ability of the model. All hours included, are however shown in appendix A.3 Figure A.1.

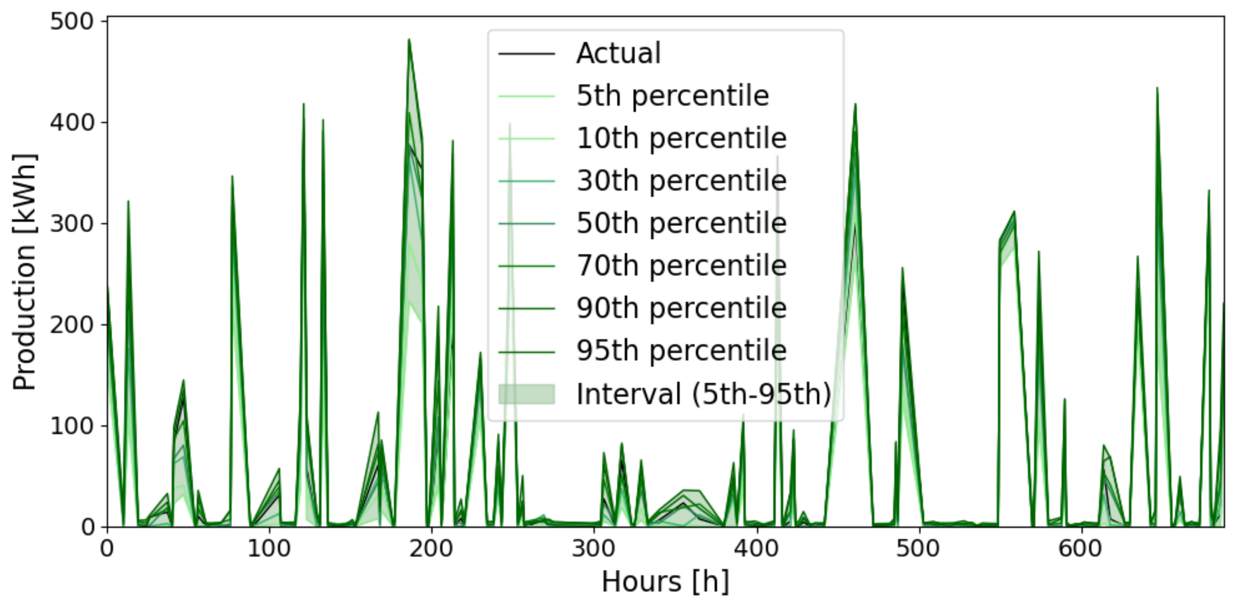


Figure 4.29: A plot of the predicted power production by the QRNN model on random test data and the actual production, with its percentiles and prediction interval. The 50th percentile represents the point prediction from the model.

From the illustrations, it seems the actual production lies within the interval at most times. It misses some of the peaks and where the production changed a lot in one hour. The model also seems to predict a better interval on random test data, as more actual production lies within.

4.7 Prediction Performance

The performance was tested on both test data and the training data as explained in chapter 3 section 3.7. Tables of the deterministic performance of the models are presented in this section. The difference between the two testing data is of the largest importance, and the variation in ANN performance for different fits, but with the same settings and parameters.

To show all performances the performances on the August test will be shown first, followed by the performance on the random test data and train data. Table 4.4 shows the deterministic performance on the August test data for all models, and Table 4.5 shows only the RMSE and MAE of 30 ANN model fits on August test data. This decision was made as these are the values that will be used for comparison. The full performance is included in Appendix A.3 Table A.1.

Table 4.4: Performance August Test. The RMSE, nRMSE, MAE and R^2 of all models in August, 2021. Ens is short for ensemble.

	SVR	ANN	Ens SVR ANN	Ens ANN	ANN NWP	QRNN	ARIMA
RMSE[kWh]	34.742	36.276	34.622	34.394	39.277	35.174	41.213
nRMSE[%]	0.057	0.059	0.056	0.056	0.064	0.057	0.067
MAE[kWh]	16.935	19.320	16.996	17.273	22.394	17.317	23.729
$\frac{\text{MAE}}{\text{Wp}}$[%]	0.028	0.028	0.027	0.028	0.037	0.028	0.039
R^2	0.912	0.904	0.913	0.914	0.888	0.910	0.876

Table 4.5: Performance August Test 30 fits. The RMSE, nRMSE, MAE and R^2 of all 30 fits of the ANN model in August, 2021.

	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	34.946	34.842	34.183	35.383	36.185	34.987
MAE[kWh]	18.184	18.101	17.068	18.945	18.700	17.955
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	34.969	35.586	34.780	34.528	35.510	34.997
MAE[kWh]	17.880	18.005	17.840	17.788	17.714	17.374
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	36.059	36.789	35.185	35.747	34.731	34.624
MAE[kWh]	20.311	21.544	18.89	18.286	18.406	17.737
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	36.673	37.034	36.779	35.341	36.930	35.216
MAE[kWh]	22.082	21.608	20.389	17.880	20.596	17.750
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	36.155	34.932	35.133	34.689	34.895	36.582
MAE[kWh]	18.360	17.459	18.335	17.846	17.880	21.361

Table 4.6 shows the performance on the random test data for all models except the ARIMA model, as it only predicted August, while Table 4.7 shows only the RMSE and MAE of 30 ANN model fits on the random test data. This decision was also as previously mentioned made as these are the only values that will be used for comparison. The full performance is included in Appendix A.3 Table A.2.

Table 4.6: Performance Random Test. The RMSE, nRMSE, MAE and R^2 of all models

	SVR	ANN	Ens SVR ANN	Ens ANN	ANN NWP	QRNN
RMSE[kWh]	21.858	21.878	21.689	21.403	28.097	21.913
nRMSE[%]	0.036	0.035	0.035	0.035	0.046	0.036
MAE[kWh]	8.835	10.549	8.897	8.918	14.725	10.446
$\frac{\text{MAE}}{\text{Wp}}$[%]	0.015	0.017	0.014	0.015	0.024	0.017
R^2	0.951	0.951	0.952	0.952	0.920	0.951

Table 4.8 shows the deterministic performance on the random train data for all models except ARIMA because predictions were not done on the train data either. These two decisions were made as the model is only used as a benchmark and it was deemed not necessary to include predictions on the random test and train when all other models did.

Table 4.7: Performance Random Test 30 fits. The RMSE, nRMSE, MAE and R^2 of all 30 fits of the ANN model.

	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	21.738	21.879	22.174	21.931	24.045	22.068
MAE[kWh]	9.240	9.525	9.350	10.107	10.612	9.909
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	21.854	21.927	22.819	22.349	22.385	22.639
MAE[kWh]	9.082	9.919	9.765	9.490	9.208	9.246
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	22.716	23.721	22.851	23.291	21.688	22.318
MAE[kWh]	11.622	13.834	11.797	9.870	9.634	9.321
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	23.954	23.588	22.761	22.02	23.535	22.743
MAE[kWh]	14.277	12.947	11.172	9.464	10.589	9.695
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	22.752	21.485	21.802	21.852	22.438	22.983
MAE[kWh]	9.603	9.058	10.126	9.707	9.572	12.550

Table 4.8: Performance Random Train. The RMSE, nRMSE, MAE and R^2 of all models.

	SVR	ANN	Ens SVR ANN	Ens ANN	ANN NWP	QRNN
RMSE[kWh]	22.788	22.912	22.445	22.057	28.472	23.045
nRMSE[%]	0.037	0.037	0.037	0.036	0.046	0.038
MAE[kWh]	8.930	10.910	9.012	9.024	14.591	9.094
$\frac{\text{MAE}}{\text{Wp}}$[%]	0.015	0.018	0.015	0.015	0.024	0.015
R^2	0.948	0.947	0.949	0.951	0.918	0.946

The performance of the 30 ANN model fits on random train data is shown in Appendix A.3 Table A.3, as it was assumed less important to compare train performances after comparing test performances.

For the ANN model, the overfitting indicator is 1.047 for the random test, meaning it indicates a slight over-fit. For the August test, the indicator is 0.628, meaning it indicates an under-fit. For the SVR model, the overfitting indicators for the random and the August test are 1.042, and 0.656, respectively. For the ANN model using only NWPs as inputs, the ANN ensemble model, the ANN, and SVR ensemble model, and the QRNN model, these indicators for the random test are 1.013, 1.030, 1.035, and 1.052, and for the August test is 0.725, 0.641, 0.648 and 0.655, respectively. The low numbers for August are probably because the statistical difference between the August test and the training data is higher than between the random test and the training data.

The probabilistic performance of the QRNN model, measured by PICP, PINAW, and CWC is shown in Table 4.9.

Table 4.9: Probabilistic performance of the QRNN model. The PICP, PINAW and CWC for both random test and August, 2021.

	Random			August		
Confidence Level	PICP	PINAW	CWC	PICP	PINAW	CWC
80%	0.884	0.066	0.067	0.800	0.110	0.110
90%	0.958	0.094	0.099	0.910	0.155	0.249

The PICP in the case of a 90% prediction interval is bigger than the PICP of 80%. The PINAW is wider for a 90% prediction interval, meaning larger prediction intervals have a higher probability of covering the actual values. Compared to other studies [2], [27], [28] this performance is acceptable. The QRNN model provides both deterministic and probabilistic forecasts with good accuracy. This model may further improve unit commitment and load dispatch even more than the deterministic models, as it provides a probability with given confidence that the power production will be within the given interval.

The largest errors for all deterministic models are shown in Table 4.10, to show the largest failure of all deterministic models. These errors may refer to the differences between true and predicted values found at the peaks or the sudden changes/drops in the earlier displayed figures.

Table 4.10: The largest error for all deterministic models for both random test and August, 2021.

Model, test	Error [kWh]	Error [%]
ANN, Random	193.416	31.55
ANN, August	242.747	39.60
SVR, Random	217.876	35.54
SVR, August	221.277	36.10
Ens SVR ANN, Random	211.397	34.49
Ens SVR ANN, August	218.963	35.72
Ens ANN, Random	204.874	33.42
Ens ANN, August	219.765	35.85
ANN NWP, Random	195.059	31.82
ANN NWP, August	243.525	39.73
ARIMA, August	239.418	39.06

4.8 Comparison of deterministic models

All models predict the production quite well, but at hours with a peak in production, they tend to miss some of the peaks. The power peaks may be hard to predict, especially since the correlation between the irradiance and the production is not at 100%, and because the irradiance is a forecasted value and not a measured one. Other reasons might be because of unknown failures, maintenance, etc.

Since the ANN models give slightly different results on each fit, it was noticed that for some fits the ANN model outperformed the SVR model, and for other fits, it was the opposite. From Table A.2, out of the 30 fits on the random test, most fits actually gave a worse performance than the SVR model, and from Table A.1, most fits on the August test also gave a worse performance.

As expected the ANN model using only NWPs as inputs has higher errors and lower R^2. Comparing the ANN model with the median of the QRNN model, their results are almost the same, which was expected as the quantile regression loss function gives the mean absolute error for the 50th percentile. Compared to the ARIMA model, all other models outperformed it. This was expected as the model was made as a benchmark where not many different orders were tested to see if the results improved, and because the ANN model in [22] also outperformed the ARIMA models.

With the random test, the ANN ensemble model got the lowest RMSE value of 21.403 kWh and nRMSE value of 0.035, and the highest R^2 score of 0.953, while the ANN and SVR ensemble model got the lowest MAE value of 8.764 kWh. As expected, the ensemble models performed slightly better than the standalone models for the random test, therefore the results are satisfactory. For the August test, the ANN ensemble model performed best as well in terms of lowest, RMSE, nRMSE, and highest R^2 score. In terms of MAE however, the SVR model performed best with an MAE of 16.935. For both test data, the SVR model had low MAE, meaning the SVR model is penalized for having larger errors on both tests while having lower absolute errors. The observation of SVR having larger errors than the ensemble models because of higher RMSE is confirmed in Table 4.10. For different fits, these results would probably change a little, but it is expected that the ensemble models will perform best overall.

Comparing the performance of the two different testing data, it is clear that the models can predict the random testing data much better. This is probably because the statistical difference between the random test and train is so small. This and the difference in overfitting indicators show that the statistical difference between test and train is of high importance. This was also expected because if one train on only winter months it would make the model learn a pattern in production that is low and thereby making it hard to predict high production in a summer month.

As mentioned in theory subsection 2.8.1, Mean Relative Error (MRE) and MAE/W_p are the same measurement, the MRE in literature can be compared with MAE/Wp in this thesis. Wang et al. measured MRE values in the range between 8.03% and 12.32% without time interval, but in the range of 1.07% and 7.71% with time interval and weather type classification [12], and in this thesis, the range is between 1.4% and 2.4% for the random test and between 2.7% and 3.9%. This means all models in this thesis got low MAE values and might be very accurate. Compared with an ensemble model and some standalone models used by Massaoudi et al. in [14], where the RMSE range is between 3.88 kW and 27.37 kW, and in this thesis is between 21.403 kW and 28.097 kW for the random test and between 34.394 kW and 41.213 kW for August test, the models get a little higher RMSE, indicating less accuracy, especially for August test. However, the RMSE is not a relative error metric, meaning an RMSE of 30 kW might be the result of very bad predictions where the installed capacity of the system is quite low, but very good RMSE where the capacity is very high. As Massaoudi et al. do not state the capacity of the system they used data from, it is hard to tell whether the models in this thesis are more accurate.

A hybrid CNN-LSTM model was made by Zhang et al. and compared to an MLP- and an LSTM model. For all PV facilities, the average RMSE of the models' predictions is 0.0778 for the MLP model, 0.0714 for the LSTM model, and 0.0689 for the CNN-LSTM model [5], which seems very low. It is assumed these values are in kW as the installed nominal power of the facilities they use data from ranges between 100 kW and 8500 kW. Compared to the RMSE values of this thesis, it would seem the models used here are not very accurate. The normalized root-mean-square error nRMSE% (based on the maximum observed power output) of the ANN-model used by Leva et al. in [20] is 12.5%, 24%, and 36.9% for a sunny day, partially cloudy day and cloudy day, respectively, and the nMAE%, based on the rated power of the system is 5.19, 13.2 and 11, respectively. Compared to the nRMSE and MAE/Wp given by the models in this thesis, ranging between 0.035 and 0.067 and between 0.014 and 0.039, respectively, it would seem that these models predict quite precisely.

For sunny, partly cloudy, and overcast days the nRMSE of the ANN models used in [21] is 0.0174%, 0.02% and 0.0108%, respectively. Since these errors are much lower than the ones in this study it would seem their ANN models predict more correctly. Compared to the best performing model used by Fernandez-Jimenez et al., the MLP model performed with an nRMSE (RMSE/rated power) of 11.79% and to their ARIMA models performing with nRMSE values of 21.14% and 17.36%, the models in this work perform slightly better. Kardakos et al. used a persistence model, two ANN models, and two seasonal ARIMA (SARIMA) models and they gave yearly average nRMSE (normalized concerning the PV installed capacity) values ranging from 11.12% to 12.89% [23]. These nRMSE values are higher than those measured in this research indicating that the models forecast accurately.

In [24] De Felice et al. the SVR model gave nRMSE (RMSE/maximum power output) values below 0.08 when they observed meteorological data as predictors and 0.18 when they use forecasted predictors on the entire prediction range. Rana et al. used an ensemble of ANNs and an SVR algorithm and got MRE (MAE normalized by the range of the target values) values ranging from 3.92% to 11.09% [25]. Different machine learning models, including SVR and MLP, were used by Mahmud et al. in [26]. The measured MAE values for SVR and MLP are 0.0157 and 0.1492, but the unit is not specified. It seems that power is measured for just one PV array with a rated power of 5.6 kW, meaning if the unit is kW, the MRE (MAE/rated power) values are 0.0028 and 0.0266 for the SVR- and MLP model, respectively. The convolutional LSTM network in [27] got MAREs (MAE divided by rated power) of 0.20%, 0.42%, 0.65%, and 0.89% for 1-step, 2-step, 3-step, and 4-step, respectively. The first two of the above papers suggest that the models in this work, forecast slightly more exactly, and the last two suggests otherwise.

Comparing the models to the results of the pre-project, the performance is very similar. For SVR the RMSE was 21.399 kWh and 30.138 kWh on the random and the August test, respectively. The difference between the SVR model in the pre-project and this work, is the training data, as more data was available this time because the PV system had been operating longer. The pre-project did also use a kNN model, producing RMSE values of 24.504 kWh and 34.529 kWh for the random and the August test, respectively. Compared to the kNN model, all models, except the ARIMA model and the ANN model trained on only NWPs, performed better.

5 Conclusions

This thesis presented forecasts of future power generation based on machine learning, artificial neural networks, and auto-regression. The models used historical power production and/or numerical weather predictions (NWPs), as well as future NWPs for their forecasts. Insignificant amounts of missing data were detected, so no processing tools were used. The data was split into two test datasets for most models, one random and the other the August month. The rest of the dataset was split into train and validation sets for most models. The ARIMA model only used august test data, while the rest was train data. After splitting, correlation techniques and principal component analysis were used for feature reduction for all models except ARIMA.

The ANN ensemble got the lowest RMSE, nRMSE, and the highest R^2 score on the random test, while the ANN and SVR ensemble model got the lowest MAE. This was expected, so on the random test, the results are satisfactory. For the August test, however, the ANN ensemble model performed best as well in terms of lowest RMSE, nRMSE, and highest R^2 score, but the lowest MAE was produced by the SVR model. For both test data, the SVR model had low MAE, meaning the SVR model is penalized for having larger errors on both tests while having lower absolute errors. For different fits, these results could perhaps slightly change, but it is expected that the ensemble models will perform best overall. All models can predict both testing data very well. However, the models can predict the random testing data much better, which is probably because the statistical difference between the random test and train is so small. The QRNN model provides both deterministic and probabilistic forecasts with good accuracy. For a 90% confidence level, PICP is 0.958 and PINAW is 0.094 for the random test and 0.910 and 0.155 for the August test, respectively. Using only NWP data as inputs, the ANN model provides good predictions, with just slightly more error than when past production is used. The difference in RMSE is 6.219 kWh for the random test, and 3.001 kWh for the August test.

Both ensemble models performed slightly better than the standalone models for the random test. However, for the August test SVR performed better than the ANN ensemble. There was still an ensemble model that performed best for both testing data in terms of RMSE, nRMSE, and R^2. In terms of MAE, SVR performed the best on the August test. All models can predict both testing data very well. However, the models can predict the random testing data much better, which is probably because the statistical difference between the random test and train is so small. The QRNN model provides both deterministic and probabilistic forecasts with good accuracy. For a 90% confidence level, PICP is 0.093 and PINAW is 0.093 for the random test and 0.909 and 0.161 for the August test, respectively. Using only NWP data as inputs, the ANN model provides good predictions, with just a little more error, than when past production is used. The difference in RMSE is 6.974 kWh for the random test, and 5.249 kWh for the August test.

Overall, all models can predict future production quite well, with a few misses at peak production or a rapid change in production.

6 Recommendations

In this chapter, suggested further work is presented, related to further advancing the field of research within photovoltaic power forecasting.

More years of historical PV power, irradiance, and temperature predictions should be available to possibly make even more accurate predictions, especially when testing on different months. However, over time degradation of PV cells may occur, and as the models do not consider this, the degradation may influence the prediction results. This degradation will probably have a limited effect, where the effect will probably be larger for models based on longer time periods. In addition, the models do not take into account hot spots, dust accumulation, soiling effects, light-induced effects, and mismatches, meaning, when doing long-term forecasting it could be interesting to see if one can build models that consider these effects.

One variable that has not been considered in this work and many other works is snow cover. This is probably because many PV plants do not get a lot of snow on them during winter. However, in Norway, this could affect the performance and maybe make the predictions for the winter months more precise.

The method of finding the optimal settings and parameters of the ANN models were limited to be within the range found in theoretical and informational documents, but it was observed in the literature that some used a lot more hidden neurons and layer, and other initializers and activation functions. Including a bigger grid to search would make the search take a lot more time and would make the models more complex, but should be considered as a part of further investigation to improve the forecasts. Testing more order of the ARIMA models, and including control for seasonality would make the model more complex and perhaps more accurate, being able to compete with the other models in this work.

The ensemble models in this work consisted of either two different models or the same model but with different settings and parameters, and it could be interesting to look further into including more models in an ensemble model, perhaps combining both artificial neural networks, machine learning models and statistical models, in addition, to use more different parameters and settings for the same models. If there are models that have different strengths and weaknesses, predicting better were others do not, an ensemble of them would perhaps get even more accurate predictions.

Lastly, as only one method of making probabilistic forecasts was used in this thesis, different methods and further investigations into this part of the field could improve the forecasts, as the quantile regression neural network model in this work has not been compared to another probabilistic model.

With all these suggestions in mind, the following suggestions to further improve the models used for solar power production forecasting based on the PV models and weather predictions located in Lillesand are:

- Use multiple years of historical data and weather forecasts
- Consider snow cover, especially for predicting the winter months
- Include models taking degradation, hot spots, dust accumulation, soiling effect, light-induced effects, and mismatches into account
- Grid searching a larger grid
- Expand the use of models in an ensemble, preferably ones that predict better at times others do not
- Look further into probabilistic models

Bibliography

[1] J. Antonanzas, N. Osorio, R. Escobar, R. Urraca, F. Martinez-de-Pison, and F. Antonanzas-Torres, "Review of photovoltaic power forecasting," *Solar Energy*, vol. 136, pp. 78–111, 2016, ISSN: 0038-092X. DOI: `https://doi.org/10.1016/j.solener.2016.06.069`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0038092X1630250X`.

[2] W.-H. Chen, L.-S. Cheng, Z.-P. Chang, *et al.*, "Interval prediction of photovoltaic power using improved narx network and density peak clustering based on kernel mahalanobis distance," *Complexity*, vol. 2022, 2022. DOI: `https://doi.org/10.1155/2022/8169510`.

[3] M. Xiang, W. Cui, C. Wan, and C. Zhao, "A sky image-based hybrid deep learning model for nonparametric probabilistic forecasting of solar irradiance," in *2021 International Conference on Power System Technology (POWERCON)*, 2021, pp. 946–952. DOI: `10.1109/POWERCON53785.2021.9697876`.

[4] T. Takamatsu, H. Ohtake, and T. Oozeki, "Support vector quantile regression for the post-processing of meso-scale ensemble prediction system data in the kanto region: Solar power forecast reducing overestimation," *Energies*, vol. 15, no. 4, 2022, ISSN: 1996-1073. DOI: `10.3390/en15041330`. [Online]. Available: `https://www.mdpi.com/1996-1073/15/4/1330`.

[5] W. Zhang, X. Chen, K. He, *et al.*, "Semi-asynchronous personalized federated learning for short-term photovoltaic power forecasting," *Digital Communications and Networks*, 2022, ISSN: 2352-8648. DOI: `https://doi.org/10.1016/j.dcan.2022.03.022`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S2352864822000438`.

[6] L. Guohai, S. Wenqing, W. Zhenfei, C. Zhaoling, and Z. Zhiyuan, "Short-term photovoltaic power forecasting based on attention-gru model," *Acta Energiae Solaris Sinica*, vol. 43, no. 2, p. 226, 2022. DOI: `10.19912/j.0254-0096.tynxb.2020-1202`.

[7] Q. Lia, X. Zhanga, T. Mab, D. Liud, H. Wanga, and W. Hua, "Multi-step ahead photovoltaic power forecasting model based on timegan, soft dtw-based k-medoids clustering, and a cnn-gru hybrid neural network," [Online]. Available: `http://dx.doi.org/10.2139/ssrn.4017353`.

[8] Q. Li, Y. Xu, B. Chew, H. Ding, and L. Zhao, "An integrated missing-data tolerant model for probabilistic pv power generation forecasting," *IEEE Transactions on Power Systems*, pp. 1–1, 2022. DOI: `10.1109/TPWRS.2022.3146982`.

[9] M. Sabri and M. El Hassouni, "A novel deep learning approach for short term photovoltaic power forecasting based on gru-cnn model," in *E3S Web of Conferences*, EDP Sciences, vol. 336, 2022, p. 00 064. DOI: `https://doi.org/10.1051/e3sconf/202233600064`.

[10] Z. Wang, D. Pan, F. Gao, H. Zhou, L. Dong, and G. He, "Ultra-short-term distributed photovoltaic power forecasting based on cloud image feature extraction," in *2021 International Conference on Power System Technology (POWERCON)*, 2021, pp. 920–924. DOI: `10.1109/POWERCON53785.2021.9697462`.

[11] S. Park, Y. Noh, S. Jung, and E. Hwang, "Shap-based explainable photovoltaic power forecasting scheme using lstm," in *Proceedings of the Korea Information Processing Society Conference*, Korea Information Processing Society, 2021, pp. 845–848. DOI: 10.3745/PKIPS.Y2021M11A.845.

[12] X. Wang, Y. Sun, D. Luo, and J. Peng, "Comparative study of machine learning approaches for predicting short-term photovoltaic power output based on weather type classification," *Energy*, vol. 240, p. 122 733, 2022, ISSN: 0360-5442. DOI: https://doi.org/10.1016/j.energy.2021.122733. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360544221029820.

[13] M. Sabri and M. El Hassouni, "A comparative study of lstm and rnn for photovoltaic power forecasting," in *International Conference on Advanced Technologies for Humanity*, Springer, 2021, pp. 265–274. DOI: https://doi.org/10.1007/978-3-030-94188-8_25.

[14] M. Massaoudi, H. Abu-Rub, S. S. Refaat, M. Trabelsi, I. Chihi, and F. S. Oueslati, "Enhanced deep belief network based on ensemble learning and tree-structured of parzen estimators: An optimal photovoltaic power forecasting method," *IEEE Access*, vol. 9, pp. 150 330–150 344, 2021. DOI: 10.1109/ACCESS.2021.3125895.

[15] G. O. Micha and C.-H. Kim, "An intelligent photovoltaic power forecasting model based on bagged-boosted stack support vector regression with kernel linear," *Journal of Electrical Society*, vol. 70, no. 11, pp. 1633–1639, 2021. DOI: 10.5370/KIEE.2021.70.11.1633.

[16] M. AlShafeey and C. Csáki, "Evaluating neural network and linear regression photovoltaic power forecasting models based on different input methods," *Energy Reports*, vol. 7, pp. 7601–7614, 2021, ISSN: 2352-4847. DOI: https://doi.org/10.1016/j.egyr.2021.10.125. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352484721011446.

[17] R. Nguyen, Y. Yang, A. Tohmeh, and H.-G. Yeh, "Predicting pv power generation using svm regression," in *2021 IEEE Green Energy and Smart Systems Conference (IGESSC)*, 2021, pp. 1–5. DOI: 10.1109/IGESSC53124.2021.9618677.

[18] B. Zazoum, "Solar photovoltaic power prediction using different machine learning methods," *Energy Reports*, vol. 8, pp. 19–25, 2022, 2021 The 8th International Conference on Power and Energy Systems Engineering, ISSN: 2352-4847. DOI: https://doi.org/10.1016/j.egyr.2021.11.183. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352484721013287.

[19] M. A. F. B. Lima, L. M. Fernández Ramírez, P. C. M. Carvalho, J. G. Batista, and D. M. Freitas, "A Comparison Between Deep Learning and Support Vector Regression Techniques Applied to Solar Forecast in Spain," *Journal of Solar Energy Engineering*, vol. 144, no. 1, Aug. 2021, 010802, ISSN: 0199-6231. DOI: 10.1115/1.4051949. eprint: https://asmedigitalcollection.asme.org/solarenergyengineering/article-pdf/144/1/010802/6739905/sol_144_1_010802.pdf. [Online]. Available: https://doi.org/10.1115/1.4051949.

[20] S. Leva, A. Dolara, F. Grimaccia, M. Mussetta, and E. Ogliari, "Analysis and validation of 24 hours ahead neural network forecasting of photovoltaic output power," *Mathematics and Computers in Simulation*, vol. 131, pp. 88–100, 2017, 11th International Conference on Modeling and Simulation of Electric Machines, Converters and Systems, ISSN: 0378-4754. DOI: https://doi.org/10.1016/j.matcom.2015.05.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378475415001238.

[21] A. Mellit, A. Massi Pavan, and V. Lughi, "Short-term forecasting of power production in a large-scale photovoltaic plant," *Solar Energy*, vol. 105, pp. 401–413, 2014, ISSN: 0038-092X. DOI: https://doi.org/10.1016/j.solener.2014.03.018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0038092X14001522.

[22] L. A. Fernandez-Jimenez, A. Muñoz-Jimenez, A. Falces, *et al.*, "Short-term power forecasting system for photovoltaic plants," *Renewable Energy*, vol. 44, pp. 311–317, 2012, ISSN: 0960-1481. DOI: https://doi.org/10.1016/j.renene.2012.01.108. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0960148112001516.

[23] E. G. Kardakos, M. C. Alexiadis, S. I. Vagropoulos, C. K. Simoglou, P. N. Biskas, and A. G. Bakirtzis, "Application of time series and artificial neural network models in short-term forecasting of pv power generation," in *2013 48th International Universities' Power Engineering Conference (UPEC)*, 2013, pp. 1–6. DOI: 10.1109/UPEC.2013.6714975.

[24] M. De Felice, M. Petitta, and P. M. Ruti, "Short-term predictability of photovoltaic production over italy," *Renewable Energy*, vol. 80, pp. 197–204, 2015, ISSN: 0960-1481. DOI: https://doi.org/10.1016/j.renene.2015.02.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0960148115001007.

[25] M. Rana, I. Koprinska, and V. G. Agelidis, "Univariate and multivariate methods for very short-term solar photovoltaic power forecasting," *Energy Conversion and Management*, vol. 121, pp. 380–390, 2016, ISSN: 0196-8904. DOI: https://doi.org/10.1016/j.enconman.2016.05.025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0196890416303934.

[26] K. Mahmud, S. Azam, A. Karim, S. Zobaed, B. Shanmugam, and D. Mathur, "Machine learning based pv power generation forecasting in alice springs," *IEEE Access*, vol. 9, pp. 46 117–46 128, 2021. DOI: 10.1109/ACCESS.2021.3066494.

[27] M. Bai, X. Zhao, Z. Long, J. Liu, and D. Yu, *Short-term probabilistic photovoltaic power forecast based on deep convolutional long short-term memory network and kernel density estimation*, 2021. DOI: 10.48550/ARXIV.2107.01343. [Online]. Available: https://arxiv.org/abs/2107.01343.

[28] Z. Cheng, W. Zhang, and C. Liu, "Photovoltaic power generation probabilistic prediction based on a new dynamic weighting method and quantile regression neural network," in *2019 Chinese Control Conference (CCC)*, 2019, pp. 6445–6451. DOI: 10.23919/ChiCC.2019.8866208.

[29] United Nations. "Take action for the sustainable development goals." (Sep. 2015), [Online]. Available: https://www.un.org/sustainabledevelopment/sustainable-development-goals/ (visited on 11/07/2021).

[30] UNFCCC. "The paris agreement | unfccc." Paris Climate Change Conference - November 2015. (Dec. 2015), [Online]. Available: https://unfccc.int/process-and-meetings/the-paris-agreement/the-paris-agreement (visited on 11/07/2021).

[31] Lovdata. "Lov om klimamål (klimaloven)." Klima- og miljødepartementet. (Jun. 2017), [Online]. Available: https://lovdata.no/lov/2017-06-16-60 (visited on 11/07/2021).

[32] M. P. Almeida, O. Perpiñán, and L. Narvarte, "Pv power forecast using a nonparametric pv model," *Solar Energy*, vol. 115, pp. 354–368, 2015, ISSN: 0038-092X. DOI: https://doi.org/10.1016/j.solener.2015.03.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0038092X15001218.

[33] M. P. Almeida, M. Muñoz, I. de la Parra, and O. Perpiñán, "Comparative study of pv power forecast using parametric and nonparametric pv models," *Solar Energy*, vol. 155, pp. 854–866, 2017, ISSN: 0038-092X. DOI: https://doi.org/10.1016/j.solener.2017.07.032. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0038092X17306175.

[34] P. Bacher, H. Madsen, and H. A. Nielsen, "Online short-term solar power forecasting," *Solar Energy*, vol. 83, no. 10, pp. 1772–1783, 2009, ISSN: 0038-092X. DOI: https://doi.org/10.1016/j.solener.2009.05.016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0038092X09001364.

[35] S. Alessandrini, L. Delle Monache, S. Sperati, and G. Cervone, "An analog ensemble for short-term probabilistic solar power forecast," *Applied Energy*, vol. 157, pp. 95–110, 2015, ISSN: 0306-2619. DOI: https://doi.org/10.1016/j.apenergy.2015.08.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306261915009368.

[36] L. Massidda and M. Marrocu, "Quantile regression post-processing of weather forecast for short-term solar power probabilistic forecasting," *Energies*, vol. 11, no. 7, 2018, ISSN: 1996-1073. DOI: 10.3390/en11071763. [Online]. Available: https://www.mdpi.com/1996-1073/11/7/1763.

[37] F. Almonacid, P. Pérez-Higueras, E. F. Fernández, and L. Hontoria, "A methodology based on dynamic artificial neural network for short-term forecasting of the power output of a pv generator," *Energy Conversion and Management*, vol. 85, pp. 389–398, 2014, ISSN: 0196-8904. DOI: https://doi.org/10.1016/j.enconman.2014.05.090. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0196890414005093.

[38] U. K. Das, K. S. Tey, M. Seyedmahmoudian, *et al.*, "Forecasting of photovoltaic power generation and model optimization: A review," *Renewable and Sustainable Energy Reviews*, vol. 81, pp. 912–928, 2018, ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2017.08.017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032117311620.

[39] 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.

[40] R. Ahmed, V. Sreeram, Y. Mishra, and M. Arif, "A review and evaluation of the state-of-the-art in pv solar power forecasting: Techniques and optimization," *Renewable and Sustainable Energy Reviews*, vol. 124, p. 109 792, 2020, ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2020.109792. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032120300885.

[41] Y. He, Y. Yan, and Q. Xu, "Wind and solar power probability density prediction via fuzzy information granulation and support vector quantile regression," *International Journal of Electrical Power & Energy Systems*, vol. 113, pp. 515–527, 2019, ISSN: 0142-0615. DOI: https://doi.org/10.1016/j.ijepes.2019.05.075. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0142061518332356.

[42] D. van der Meer, J. Widén, and J. Munkhammar, "Review on probabilistic forecasting of photovoltaic power production and electricity consumption," *Renewable and Sustainable Energy Reviews*, vol. 81, pp. 1484–1512, 2018, ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2017.05.212. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032117308523.

[43] Y. Han, L. Mi, L. Shen, C. Cai, Y. Liu, and K. Li, "A short-term wind speed interval prediction method based on wrf simulation and multivariate line regression for deep learning algorithms," *Energy Conversion and Management*, vol. 258, p. 115 540, 2022, ISSN: 0196-8904. DOI: https://doi.org/10.1016/j.enconman.2022.115540. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0196890422003363.

[44] H. Cui, F. Li, X. Fang, H. Chen, and H. Wang, "Bi-level arbitrage potential evaluation for grid-scale energy storage considering wind power and lmp smoothing effect," *IEEE Transactions on Sustainable Energy*, vol. PP, Sep. 2017. DOI: 10.1109/TSTE.2017.2758378.

[45] A. S. B. Mohd Shah, H. Yokoyama, and N. Kakimoto, "High-precision forecasting model of solar irradiance based on grid point value data analysis for an efficient photovoltaic system," *IEEE Transactions on Sustainable Energy*, vol. 6, no. 2, pp. 474–481, 2015. DOI: 10.1109/TSTE.2014.2383398.

[46] L. Ramírez and J. Vindel, "13 - forecasting and nowcasting of dni for concentrating solar thermal systems," in *Advances in Concentrating Solar Thermal Research and Technology*, ser. Woodhead Publishing Series in Energy, M. J. Blanco and L. R. Santigosa, Eds., Woodhead Publishing, 2017, pp. 293–310, ISBN: 978-0-08-100516-3. DOI: https://doi.org/10.1016/B978-0-08-100516-3.00013-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780081005163000137.

[47] Statistics Solutions. "Correlation (pearson, kendall, spearman) - statistics solutions." (Jan. 2013), [Online]. Available: https://www.statisticssolutions.com/%20free-resources/directory-of-statistical-analyses/%20correlation-pearson-kendall-spearman/ (visited on 11/10/2021).

[48] S. Mukhopadhyay, *Advanced Data Analytics Using Python: With Machine Learning, Deep Learning and NLP Examples*, eng. Berkeley, CA: Apress L. P, 2018, ISBN: 9781484234495.

[49] S.-F. Wu, C.-Y. Chang, and S.-J. Lee, "Time series forecasting with missing values," vol. 1, Nov. 2015. DOI: 10.4108/icst.iniscom.2015.258269.

[50] G. E. A. P. A. Batista and M. C. Monard, "An analysis of four missing data treatment methods for supervised learning," *Applied Artificial Intelligence*, vol. 17, no. 5-6, pp. 519–533, 2003. DOI: 10.1080/713827181.

[51] T. Kim, W. Ko, and J. Kim, "Analysis and impact evaluation of missing data imputation in day-ahead pv generation forecasting," *Applied Sciences*, vol. 9, no. 1, 2019, ISSN: 2076-3417. DOI: 10.3390/app9010204. [Online]. Available: https://www.mdpi.com/2076-3417/9/1/204.

[52] I. P. Panapakidis, A. S. Bouhouras, and G. C. Christoforidis, "A missing data treatment method for photovoltaic installations," in *2018 IEEE International Energy Conference (ENERGYCON)*, 2018, pp. 1–6. DOI: 10.1109/ENERGYCON.2018.8398780.

[53] C. Khanna. "What and why behind fit_transform() and transform() | towards data science." (Aug. 2020), [Online]. Available: https://towardsdatascience.com/what-and-why-behind-fit-transform-vs-transform-in-scikit-learn-78f915cf96fe.

[54] Deepchecks. "What is validation set in machine learning - deepchecks." (May 2021), [Online]. Available: https://deepchecks.com/glossary/validation-set-in-machine-learning/ (visited on 05/07/2022).

[55] T. V. Stefansson, "Forecasting of photovoltaic power production," Unpublished, Energy Research Project, University of Agder, Grimstad, Norway, 2021.

[56] S. Qijun, L. Fen, Q. Jialin, Z. Jinbin, and C. Zhenghong, "Photovoltaic power prediction based on principal component analysis and support vector machine," in *2016 IEEE Innovative Smart Grid Technologies - Asia (ISGT-Asia)*, 2016, pp. 815–820. DOI: 10.1109/ISGT-Asia.2016.7796490.

[57] N. J. Johannesen, M. L. Kolhe, and M. Goodwin, "Comparing recurrent neural networks using principal component analysis for electrical load predictions," in *2021 6th International Conference on Smart and Sustainable Technologies (SpliTech)*, 2021, pp. 1–6. DOI: 10.23919/SpliTech52315.2021.9566357.

[58] G. Zhang, B. Eddy Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks:: The state of the art," *International Journal of Forecasting*, vol. 14, no. 1, pp. 35–62, 1998, ISSN: 0169-2070. DOI: https://doi.org/10.1016/S0169-2070(97)00044-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207097000447.

[59] M. M. Mijwil. "Artificial neural networks advantages and disadvantages | linkedin." (Jan. 2018), [Online]. Available: https://www.linkedin.com/pulse/artificial-neural-networks-advantages-disadvantages-maad-m-mijwel/.

[60] N. Donges. "4 disadvantages of neural networks | built in." (Dec. 2021), [Online]. Available: https://builtin.com/data-science/disadvantages-neural-networks.

[61] H. Maier, A. Jain, G. Dandy, and K. Sudheer, "Methods used for the development of neural networks for the prediction of water resource variables in river systems: Current status and future directions," *Environmental Modelling & Software*, vol. 25, pp. 891–909, Aug. 2010. DOI: 10.1016/j.envsoft.2010.02.003.

[62] S. Ghorbani, M. Barari, and M. Hoseini, "Presenting a new method to improve the detection of micro-seismic events," *Environmental Monitoring and Assessment*, vol. 190, Jul. 2018. DOI: 10.1007/s10661-018-6837-6.

[63] N. Kimura, I. Yoshinaga, K. Sekijima, I. Azechi, and D. Baba, "Convolutional neural network coupled with a transfer-learning approach for time-series flood predictions," *Water*, vol. 12, p. 96, Dec. 2019. DOI: 10.3390/w12010096.

[64] S. Chandrasekaran and S. K. Govindarajan, "Sridharan c., and g. suresh kumar. (2019). "optimization of rate of penetration with real time measurements using machine learning and meta-heuristic algorithm". international journal of scientific & technology research (elsevier publications: (issn 2277-8616), vol. 8(9)., pp. 1427-1432.," *International Journal of Scientific & Technology Research*, vol. 8, pp. 1427–1432, Oct. 2019.

[65] S. Haykin, *Neural networks and learning machines*, eng, Upper Saddle River, N.J, 2009.

[66] D. Johnson. "Back propagation neural network: What is backpropagation algorithm in machine learning?" (Feb. 2022), [Online]. Available: https://www.guru99.com/backpropogation-neural-network.html (visited on 02/19/2022).

[67] N. Kimura, I. Yoshinaga, K. Sekijima, I. Azechi, and D. Baba, "Convolutional neural network coupled with a transfer-learning approach for time-series flood predictions," *Water*, vol. 12, no. 1, 2020, ISSN: 2073-4441. DOI: 10.3390/w12010096. [Online]. Available: https://www.mdpi.com/2073-4441/12/1/96.

[68] Z. Mohamed, "Using the artificial neural networks for prediction and validating solar radiation," *Journal of the Egyptian Mathematical Society*, vol. 27, p. 47, Nov. 2019. DOI: 10.1186/s42787-019-0043-8.

[69] A. J. Thomas, M. Petridis, S. D. Walters, S. M. Gheytassi, and R. E. Morgan, "Two hidden layers are usually better than one," in *Engineering Applications of Neural Networks*, G. Boracchi, L. Iliadis, C. Jayne, and A. Likas, Eds., Cham: Springer International Publishing, 2017, pp. 279–290, ISBN: 978-3-319-65172-9.

[70] R. Azka, W. Soefian, D. R. Aryani, F. H. Jufri, and A. R. Utomo, "Modelling of photovoltaic system power prediction based on environmental conditions using neural network single and multiple hidden layers," *IOP Conference Series: Earth and Environmental Science*, vol. 599, no. 1, p. 012 032, Nov. 2020. DOI: `10.1088/1755-1315/599/1/012032`. [Online]. Available: `https://doi.org/10.1088/1755-1315/599/1/012032`.

[71] D. W. Patterson, *Artificial neural networks : Theory and applications*, eng, Singapore, 1996.

[72] 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, ISSN: 0038-092X. DOI: `https://doi.org/10.1016/j.solener.2011.08.027`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0038092X11003008`.

[73] M. Ding, L. Wang, and R. Bi, "An ann-based approach for forecasting the power output of photovoltaic system," *Procedia Environmental Sciences*, vol. 11, pp. 1308–1315, 2011, 2011 2nd International Conference on Challenges in Environmental Science and Computer Engineering (CESCE 2011), ISSN: 1878-0296. DOI: `https://doi.org/10.1016/j.proenv.2011.12.196`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S187802961101019X`.

[74] B. Kermanshahi, "Recurrent neural network for forecasting next 10 years loads of nine japanese utilities," *Neurocomputing*, vol. 23, no. 1, pp. 125–133, 1998, ISSN: 0925-2312. DOI: `https://doi.org/10.1016/S0925-2312(98)00073-3`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0925231298000733`.

[75] G. Panchal, A. Ganatra, Y. Kosta, and D. Panchal, "Behaviour analysis of multilayer perceptrons with multiple hidden neurons and hidden layers," *International Journal of Computer Theory and Engineering*, vol. 3, no. 2, pp. 332–337, 2011.

[76] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *towards data science*, vol. 6, no. 12, pp. 310–316, 2017.

[77] Keras Team. "Layer activation functions." (Apr. 2015), [Online]. Available: `https://keras.io/api/layers/activations/#layer-activation-functions`.

[78] A. Shah, E. Kadam, H. Shah, S. Shinde, and S. Shingade, "Deep residual networks with exponential linear unit," in *Proceedings of the Third International Symposium on Computer Vision and the Internet*, ser. VisionNet'16, Jaipur, India: Association for Computing Machinery, 2016, pp. 59–65, ISBN: 9781450343015. DOI: `10.1145/2983402.2983406`. [Online]. Available: `https://doi.org/10.1145/2983402.2983406`.

[79] L. SHUKLA. "Fundamentals of neural networks on weights & biases." (Aug. 2019), [Online]. Available: `https://wandb.ai/site/articles/fundamentals-of-neural-networks`.

[80] S. Krishna Kumar, "On weight initialization in deep neural networks," *arXiv e-prints*, arXiv–1704, 2017.

[81] S. Arora. "Weight initialization techniques for deep neural networks - geeksforgeeks." (Jan. 2022), [Online]. Available: `https://www.geeksforgeeks.org/weight-initialization-techniques-for-deep-neural-networks/`.

[82] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, “Self-normalizing neural networks,” *ArXiv*, vol. abs/1706.02515, 2017.

[83] D. Kumar. “Vanishing gradient problem | what is vanishing gradient problem?” (Sep. 2020), [Online]. Available: `https://www.mygreatlearning.com/blog/the-vanishing-gradient-problem/`.

[84] DeepAI. “Exploding gradient problem definition | deepai.” (Jan. 2018), [Online]. Available: `https://deepai.org/machine-learning-glossary-and-terms/exploding-gradient-problem`.

[85] B. Or. “The exploding and vanishing gradients problem in time series | by barak or | towards data science.” (Oct. 2020), [Online]. Available: `https://towardsdatascience.com/the-exploding-and-vanishing-gradients-problem-in-time-series-6b87d558d22`.

[86] Y. Bengio, I. Goodfellow, and A. Courville, *Deep learning*. MIT press Cambridge, MA, USA, 2017, vol. 1.

[87] T. Quddoos. “Neural network basics: Loss and cost functions | by talha quddoos | artificialis | medium.” (Nov. 2021), [Online]. Available: `https://medium.com/artificialis/neural-network-basics-loss-and-cost-functions-9d089e9de5f8`.

[88] J. Brownlee. “Loss and loss functions for training deep learning neural networks.” (Oct. 2019), [Online]. Available: `https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/`.

[89] D. Lee and K. Kim, “Recurrent neural network-based hourly prediction of photovoltaic power output using meteorological information,” *Energies*, vol. 12, no. 2, 2019, ISSN: 1996-1073. DOI: `10.3390/en12020215`. [Online]. Available: `https://www.mdpi.com/1996-1073/12/2/215`.

[90] A. M. Karimi, Y. Wu, M. Koyuturk, and R. H. French, “Spatiotemporal graph neural network for performance prediction of photovoltaic power systems,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 15 323–15 330.

[91] Deeplizard. “Loss in a neural network explained - deeplizard.” (Nov. 2027), [Online]. Available: `https://deeplizard.com/learn/video/Skc8nqJirJg`.

[92] N. S. Chauhan. “Loss functions in neural networks.” (Aug. 2021), [Online]. Available: `https://www.theaidream.com/post/loss-functions-in-neural-networks`.

[93] J. Brownlee. “Loss and loss functions for training deep learning neural networks.” (Jan. 2019), [Online]. Available: `https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/`.

[94] S. Doshi. “Various optimization algorithms for training neural network | by sanket doshi | towards data science.” (Jan. 2019), [Online]. Available: `https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6`.

[95] R. Khandelwal. “Overview of different optimizers for neural networks | by renu khandelwal | datadriveninvestor.” (Feb. 2019), [Online]. Available: `https://medium.datadriveninvestor.com/overview-of-different-optimizers-for-neural-networks-e0ed119440c3`.

[96] G. Rajpal. “Optimizer & loss functions in neural network | by gaurav rajpal | analytics vidhya | medium.” (Oct. 2020), [Online]. Available: `https://medium.com/analytics-vidhya/optimizer-loss-functions-in-neural-network-2520c244cc22`.

[97] E. Alpaydin, *Introduction to machine learning*, eng, Cambridge, Massachusetts, 2020.

[98] A. Géron, *Hands-on machine learning with scikit-learn, keras, and tensorflow : Concepts, tools, and techniques to build intelligent systems*, eng, Sebastopol, CA, 2019.

[99] S. Sobri, S. Koohi-Kamali, and N. A. Rahim, "Solar photovoltaic generation forecasting methods: A review," *Energy Conversion and Management*, vol. 156, pp. 459–497, 2018, ISSN: 0196-8904. DOI: `https://doi.org/10.1016/j.enconman.2017.11.019`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0196890417310622`.

[100] M. Awad and R. Khanna, "Support vector regression," in *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*. Berkeley, CA: Apress, 2015, pp. 67–80, ISBN: 978-1-4302-5990-9. DOI: `10.1007/978-1-4302-5990-9_4`. [Online]. Available: `https://doi.org/10.1007/978-1-4302-5990-9_4`.

[101] M. Abuella and B. Chowdhury, "Solar power forecasting using support vector regression," Mar. 2017. DOI: `https://doi.org/10.48550/arxiv.1703.09851`.

[102] T. M. Bafitlhile and Z. Li, "Applicability of ε-support vector machine and artificial neural network for flood forecasting in humid, semi-humid and semi-arid basins in china," *Water*, vol. 11, no. 1, 2019, ISSN: 2073-4441. DOI: `10.3390/w11010085`. [Online]. Available: `https://www.mdpi.com/2073-4441/11/1/85`.

[103] R. Nau. "Introduction to arima models." (Aug. 2020), [Online]. Available: `https://people.duke.edu/~rnau/411arim.htm`.

[104] K. Basaran, A. Özçift, and D. Kılınç, "A new approach for prediction of solar radiation with using ensemble learning algorithm," *Arabian Journal for Science and Engineering*, vol. 44, no. 8, pp. 7159–7171, 2019.

[105] K. Li, R. Wang, H. Lei, T. Zhang, Y. Liu, and X. Zheng, "Interval prediction of solar power using an improved bootstrap method," *Solar Energy*, vol. 159, pp. 97–112, 2018, ISSN: 0038-092X. DOI: `https://doi.org/10.1016/j.solener.2017.10.051`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0038092X17309313`.

[106] C. Chatfield, "Prediction intervals for time-series forecasting," in *Principles of Forecasting: A Handbook for Researchers and Practitioners*, J. S. Armstrong, Ed. Boston, MA: Springer US, 2001, pp. 475–494, ISBN: 978-0-306-47630-3. DOI: `10.1007/978-0-306-47630-3_21`. [Online]. Available: `https://doi.org/10.1007/978-0-306-47630-3_21`.

[107] Y. He and H. Li, "Probability density forecasting of wind power using quantile regression neural network and kernel density estimation," *Energy Conversion and Management*, vol. 164, pp. 374–384, 2018, ISSN: 0196-8904. DOI: `https://doi.org/10.1016/j.enconman.2018.03.010`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0196890418302255`.

[108] J. W. Taylor, "A quantile regression neural network approach to estimating the conditional density of multiperiod returns," *Journal of Forecasting*, vol. 19, no. 4, pp. 299–311, 2000. DOI: `https://doi.org/10.1002/1099-131X(200007)19:4<299::AID-FOR775>3.0.CO;2-V`.

[109] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[110] D. Berrar, "Cross-validation," in *Encyclopedia of Bioinformatics and Computational Biology*, S. Ranganathan, M. Gribskov, K. Nakai, and C. Schönbach, Eds., Oxford: Academic Press, 2019, pp. 542–545, ISBN: 978-0-12-811432-2. DOI: `https://doi.org/10.1016/B978-0-12-809633-8.20349-X`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/B978012809633820349X`.

[111] Great Learning Team. "An introduction to grid search cv | what is grid search." (Sep. 2020), [Online]. Available: https://www.mygreatlearning.com/blog/gridsearchcv/.

[112] Javatpoint. "Bias and variance in machine learning - javatpoint." (Nov. 2021), [Online]. Available: https://www.javatpoint.com/bias-and-variance-in-machine-learning.

[113] Simplilearn. "Bias and variance in machine learning: An in depth explanation." (Feb. 2022), [Online]. Available: https://www.simplilearn.com/tutorials/machine-learning-tutorial/bias-and-variance (visited on 03/18/2022).

[114] JJ. "Mae and rmse — which metric is better? | by jj | human in a machine world | medium." (Mar. 2016), [Online]. Available: https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d.

[115] Zach. "Mae vs. rmse: Which metric should you use? - statology." (Oct. 2021), [Online]. Available: https://www.statology.org/mae-vs-rmse/.

[116] W. Ding and F. Meng, "Point and interval forecasting for wind speed based on linear component extraction," *Applied Soft Computing*, vol. 93, p. 106 350, 2020, ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2020.106350. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494620302908.

[117] A. Khosravi, S. Nahavandi, D. Creighton, and A. F. Atiya, "Lower upper bound estimation method for construction of neural network-based prediction intervals," *IEEE Transactions on Neural Networks*, vol. 22, no. 3, pp. 337–346, 2011. DOI: 10.1109/TNN.2010.2096824.

[118] "Table convert online - make it easier to work with tables." (Nov. 2021), [Online]. Available: https://tableconvert.com/.

[119] H. T. D. Owens R. G. "2.1 global atmospheric model - forecast user guide - ecmwf confluence wiki." (2018), [Online]. Available: https://confluence.ecmwf.int/display/FUG/2.1+Global+Atmospheric+Model.

[120] F. Chollet. "The sequential model." (Apr. 2020), [Online]. Available: https://keras.io/guides/sequential_model/.

[121] H. Zang, L. Cheng, T. Ding, K. W. Cheung, Z. Wei, and G. Sun, "Day-ahead photovoltaic power forecasting approach based on deep convolutional neural networks and meta learning," *International Journal of Electrical Power & Energy Systems*, vol. 118, p. 105 790, 2020, ISSN: 0142-0615. DOI: https://doi.org/10.1016/j.ijepes.2019.105790. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0142061519307409.

[122] A. Mellit, A. M. Pavan, and V. Lughi, "Deep learning neural networks for short-term photovoltaic power forecasting," *Renewable Energy*, vol. 172, pp. 276–288, 2021, ISSN: 0960-1481. DOI: https://doi.org/10.1016/j.renene.2021.02.166. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0960148121003475.

[123] J. Wang, Z. Qian, J. Wang, and Y. Pei, "Hour-ahead photovoltaic power forecasting using an analog plus neural network ensemble method," *Energies*, vol. 13, no. 12, 2020, ISSN: 1996-1073. DOI: 10.3390/en13123259. [Online]. Available: https://www.mdpi.com/1996-1073/13/12/3259.

[124] J. Brownlee. "A gentle introduction to mini-batch gradient descent and how to configure batch size." (Aug. 2019), [Online]. Available: https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/.

[125] knowledge Transfer. “What is batch size, steps, iteration, and epoch in the neural network? - knowledge transfer.” (Aug. 2021), [Online]. Available: https://androidkt.com/batch-size-step-iteration-epoch-neural-network/.

A Appendices

The appendix is divided into three main parts, being the abstract of the research project, the script in python and other python plots and tables A.3.

A.1 Abstract Research Project

Photovoltaic (PV) power generation mainly depends on the amount of solar irradiance, which in turn depends on other weather parameters. Therefore, for solar farm owners, small scale solar system owners and grid operators, predicting future power production may help in integrating this renewable energy source on a larger scale. In this research project, models based on support vector regression (SVR) and k-nearest neighbor (kNN) is proposed to forecast PV power production at time t, using both hourly measured historical observed power production and hourly measured numerical weather predictions (NWP). The strongly correlated NWP variables were used in feature selection for both models, and the models were developed on both all hours of the day and on only the hours between sunrise and sunset. The research question is:

"How will the models perform when trained and predicted on only the hours between sunrise and sunset, compared to on all hours of the day?"

The objective of this research project work is to find an answer to the question and to build good forecasting models, the company Gorines can use to predict their system's future power production. The proposed models were found to quite accurately predict the PV power production in both cases, where the SVR model predicted the output a little better than the kNN model. The measured RMSE for all hours, on random test data and august test data for SVR is 21.399 kWh and 30.138 kWh, respectively, and for kNN, this is 24.504 and 24.529, respectively. On daylight hours this is 31.026 kWh and 38.488 kWh for SVR and 35.772 kWh and 46.029 kWh for kNN. The differences between the two cases are quite small, when the night hours are disregarded. Based on the quite accurate predictions, these models could improve standalone operations and grid integration of PV power generation. The results could possibly have been improved with more measured data, meaning having more months/years of hourly measured data.

A.2 Python code

Here is all the imported libraries and further follows all python codes separated into subsections.

```
#%% Importing libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from os import path
```

```
import matplotlib.dates as dates
from sklearn import preprocessing as pre
from sklearn.decomposition import PCA
import keras
from keras.models import Sequential
from keras.layers import Dense
import tensorflow as tf
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
from scikeras.wrappers import  KerasRegressor
from keras.wrappers.scikit_learn import KerasRegressor as KR
from numpy import argsort
from sklearn.metrics import mean_absolute_error
from sklearn.ensemble import VotingRegressor
from sklearn.metrics import make_scorer
import time
from statsmodels.tsa.arima.model import ARIMA
```

A.2.1 Importing and cleaning data

```
#%% Importing and cleaning data
#Importing data
pv = pd.read_csv('Gorines data.csv')

#Changing time column into UTC time
pv['Timestamp']=pd.to_datetime(pv['Timestamp'],dayfirst=True,utc=False)
pv['local_time'] = pv['Timestamp'].dt.tz_localize('CET',ambiguous='NaT')
pv['UTC']=pv['local_time'].dt.tz_convert('UTC')
pv['UTC']=pd.to_datetime(pv['UTC'])
pv['UTC']=pv['UTC'].dt.tz_localize(None)
pv=pv.set_index('UTC')
pv=pv.drop(columns=['Timestamp','local_time'])
pv=pv.replace('--' ,np.nan)
pv_0=pv[pv.index.minute==00]
Power_data=pd.DataFrame()
Power_data=pd.DatetimeIndex(np.unique(np.hstack([pd.date_range(
    ('2021-01-01 00:00:00'),('2021-12-31 23:00:00' ),freq='H') ])))
Power_data=pd.DataFrame(Power_data,columns=['UTC'])
Power_data=Power_data.set_index('UTC')
Power_data=pd.concat([Power_data,pv_0],axis=1)

#Normalizing
def nm(df):
    df = df.copy()
    df['UTC'] = df.index.strftime('%Y-%m-%d %H:00:00')
    df=df.set_index('UTC')
    return df
```

```
Power_data=nm(Power_data)
pv_1=nm(pv[pv.index.minute==1])
pv_8=nm(pv[pv.index.minute==8])
pv_14=nm(pv[pv.index.minute==14])
pv_15=nm(pv[pv.index.minute==15])
pv_27=nm(pv[pv.index.minute==27])
pv_34=nm(pv[pv.index.minute==34])
pv_35=nm(pv[pv.index.minute==35])
pv_42=nm(pv[pv.index.minute==42])
pv_52=nm(pv[pv.index.minute==52])

for col in Power_data.columns:
    for idx in Power_data.index:
        if  pd.isna(Power_data.loc[idx,col])==True :
            if idx  in pv_1.index:
                Power_data.loc[idx,col]=pv_1.loc[idx,col]
            elif idx  in pv_8.index:
                Power_data.loc[idx,col]=pv_8.loc[idx,col]
            elif idx  in pv_14.index:
                Power_data.loc[idx,col]=pv_14.loc[idx,col]
            elif idx  in pv_15.index:
                Power_data.loc[idx,col]=pv_15.loc[idx,col]
            elif idx  in pv_27.index:
                Power_data.loc[idx,col]=pv_27.loc[idx,col]
            elif idx  in pv_34.index:
                Power_data.loc[idx,col]=pv_34.loc[idx,col]
            elif idx  in pv_35.index:
                Power_data.loc[idx,col]=pv_35.loc[idx,col]
            elif idx  in pv_42.index:
                Power_data.loc[idx,col]=pv_42.loc[idx,col]
            elif idx  in pv_52.index:
                Power_data.loc[idx,col]=pv_52.loc[idx,col]

#NaNs
nr_nans=Power_data.isna().sum()
print('NaNs:')
print(nr_nans)
print('How much of the total data is missing?')
print(nr_nans.max()/len(Power_data)*100,'%')
nan_rows = Power_data[Power_data.isna().any(axis=1)]
Power_data=Power_data.dropna()

#Combining the power from M67, M68 and M69
Power_data['Production'] = Power_data[['M67', 'M68',
                                      'M69']].astype('float64').sum(axis=1)
Power_data = Power_data.drop(columns = ['M67', 'M68', 'M69'])
NWP_data=pd.read_csv('Gorines_ECMWF_IFS.csv') # NWP data
```

```
NWP_data['UTC']=pd.to_datetime(NWP_data['Date'],yearfirst=True)
NWP_data=NWP_data.drop(columns=['Date'])
NWP_data['UTC']=NWP_data['UTC'].dt.tz_localize(None)
NWP_data['UTC'] = NWP_data['UTC'].dt.strftime('%Y-%m-%d %H:00:00')
NWP_data=NWP_data.set_index('UTC')
All_data=pd.concat((NWP_data,Power_data),axis=1)
nans_all = All_data[All_data.isna().any(axis=1)]

#Giving the NWP columns new names
All_data = All_data.set_axis(["Temp", "Precipitation", "WS", "WD",
                          "Global rad", "Direct rad", "Diffuse rad", "CC",
                          "Production"], axis=1)
All_data.index = pd.to_datetime(All_data.index,yearfirst=True)

#Filling nans in production with zero where Global rad = 0 and nighthour
All_data['Production'] = np.where((All_data['Global rad'] == 0) &
                                  (All_data['Production'].isna()), 0,
                                  All_data['Production'])
nans = All_data[All_data.isna().any(axis=1)]
All_data = All_data.dropna()

print('When is hours missing now?')
print('Jan 2021')
print(744-len(All_data.loc['2021-01-01 00:00:00':'2021-01-31 23:00:00']),
      'hours')
print('Feb 2021')
print(672-len(All_data.loc['2021-02-01 00:00:00':'2021-02-28 23:00:00']),
      'hours')
print('March 2021')
print(744-len(All_data.loc['2021-03-01 00:00:00':'2021-03-31 23:00:00']),
      'hours')
print('April 2021')
print(720-len(All_data.loc['2021-04-01 00:00:00':'2021-04-30 23:00:00']),
      'hours')
print('May 2021')
print(744-len(All_data.loc['2021-05-01 00:00:00':'2021-05-31 23:00:00']),
      'hours')
print('June 2021')
print(720-len(All_data.loc['2021-06-01 00:00:00':'2021-06-30 23:00:00']),
      'hours')
print('July 2021')
print(744-len(All_data.loc['2021-07-01 00:00:00':'2021-07-31 23:00:00']),
      'hours')
print('Aug 2021')
print(744-len(All_data.loc['2021-08-01 00:00:00':'2021-08-31 23:00:00']),
      'hours')
print('Sept 2021')
print(720-len(All_data.loc['2021-09-01 00:00:00':'2021-09-30 23:00:00']),
      'hours')
print('Okt 2021')
print(744-len(All_data.loc['2021-10-01 00:00:00':'2021-10-31 23:00:00']),
      'hours')
print('Nov 2021')
print(720-len(All_data.loc['2021-11-01 00:00:00':'2021-11-30 23:00:00']),
      'hours')
print('Des 2021')
print(744-len(All_data.loc['2021-12-01 00:00:00':'2021-12-31 23:00:00']),
      'hours')
```

A.2.2 Data visualization

```
#%% Data visualization
outpath = "C:/Users/thale/OneDrive/Documents/UiA/Master/All hours"
# To show the first day of the data frame in report
Firstday = All_data.loc['2021-01-01 00:00:00':'2021-01-01 23:00:00']
Firstday = Firstday.loc[:,Firstday.columns != 'Hour of day']
Firstday = Firstday.loc[:,Firstday.columns != 'Hour of year']
Firstday.to_csv('Firstday.csv')
# plotting hourly production for various months
""" Trying to show days with no missing hours"""
day_jan = All_data.loc['2021-01-15 00:00:00':'2021-01-15 23:00:00']
day_feb = All_data.loc['2021-02-28 00:00:00':'2021-02-28 23:00:00']
day_march = All_data.loc['2021-03-15 00:00:00':'2021-03-15 23:00:00']
day_april = All_data.loc['2021-04-19 00:00:00':'2021-04-19 23:00:00']
day_may = All_data.loc['2021-05-18 00:00:00':'2021-05-18 23:00:00']
day_june = All_data.loc['2021-06-23 00:00:00':'2021-06-23 23:00:00']
day_july = All_data.loc['2021-07-12 00:00:00':'2021-07-12 23:00:00']
day_aug = All_data.loc['2021-08-11 00:00:00':'2021-08-11 23:00:00']
day_sept= All_data.loc['2021-09-14 00:00:00':'2021-09-14 23:00:00']
day_okt = All_data.loc['2021-10-12 00:00:00':'2021-10-12 23:00:00']
day_nov = All_data.loc['2021-11-21 00:00:00':'2021-11-21 23:00:00']
day_des = All_data.loc['2021-12-21 00:00:00':'2021-12-21 23:00:00']

plt.figure(figsize=(10,5))
plt.plot(np.array(day_jan['Production']), label = 'January')
plt.plot(np.array(day_feb['Production']), label = 'February')
plt.plot(np.array(day_march['Production']), label = 'March')
plt.plot(np.array(day_april['Production']), label = 'April')
plt.plot(np.array(day_may['Production']), label = 'May')
plt.plot(np.array(day_june['Production']), label = 'June')
plt.plot(np.array(day_july['Production']), label = 'July')
plt.plot(np.array(day_aug['Production']), label = 'August', color = 'k')
plt.plot(np.array(day_sept['Production']), label = 'September')
plt.plot(np.array(day_okt['Production']), label = 'October')
plt.plot(np.array(day_nov['Production']), label = 'November')
plt.plot(np.array(day_des['Production']), label = 'December')
plt.xticks(np.arange(0, 24, 1), fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylim(0)
plt.xlim(0,23)
plt.xlabel('Time of Day [h]', fontsize = 16)
plt.ylabel('Production [kWh]', fontsize = 16)
plt.legend(fontsize = 13)
plt.tight_layout()
plt.savefig(path.join(
    outpath, "Hourly production a clear sky day for various months"))
plt.show()

plt.figure(figsize=(10,5))
plt.plot(np.array(day_jan['Global rad']), label = 'January')
plt.plot(np.array(day_feb['Global rad']), label = 'February')
plt.plot(np.array(day_march['Global rad']), label = 'March')
plt.plot(np.array(day_april['Global rad']), label = 'April')
plt.plot(np.array(day_may['Global rad']), label = 'May')
plt.plot(np.array(day_june['Global rad']), label = 'June')
plt.plot(np.array(day_july['Global rad']), label = 'July')
plt.plot(np.array(day_aug['Global rad']), label = 'August', color = 'k')
plt.plot(np.array(day_sept['Global rad']), label = 'September')
plt.plot(np.array(day_okt['Global rad']), label = 'October')
plt.plot(np.array(day_nov['Global rad']), label = 'November')
plt.plot(np.array(day_des['Global rad']), label = 'December')
plt.xticks(np.arange(0, 24, 1), fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylim(0)
plt.xlim(0,23)
plt.xlabel('Time of Day [h]', fontsize = 16)
plt.ylabel('Global Radiation [W/m$^2$]', fontsize = 16)
plt.legend(fontsize = 13)
plt.tight_layout()
plt.savefig(path.join(
    outpath, "Hourly Global radiation a clear sky day for various months"))
plt.show()

# show hourly production [kWh] in july
july = All_data.loc['2021-07-01 00:00:00':'2021-07-31 23:00:00']
fig, ax = plt.subplots(figsize=(10,5))
plt.plot(july['Production'])
plt.xlabel('Date', fontsize=16)
plt.ylabel('Production [kWh]', fontsize=16)
plt.ylim(0)
plt.xlim(july.index[0],july.index[743])
plt.xticks( fontsize = 14)
plt.yticks(fontsize = 14)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(outpath,"Hourly production in july"))
plt.show()

# show hourly global radiation [W/m^2] in july
july = All_data.loc['2021-07-01 00:00:00':'2021-07-31 23:00:00']
fig, ax = plt.subplots(figsize=(10,5))
plt.plot(july['Global rad'],label = 'Global radiation')
plt.plot(july['Direct rad'],label = 'Direct radiation')
plt.plot(july['Diffuse rad'],label = 'Diffuse radiation')
plt.xlabel('Date', fontsize=16)
plt.ylabel('Radiation [W/m$^2$]', fontsize=16)
plt.ylim(0)
plt.xlim(july.index[0],july.index[743])
plt.xticks( fontsize = 14)
plt.yticks(fontsize = 14)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.legend()
plt.tight_layout()
plt.savefig(path.join(outpath,"Hourly irradiation in july"))
plt.show()

# show hourly temperature [*C] in july
july = All_data.loc['2021-07-01 00:00:00':'2021-07-31 23:00:00']
fig, ax = plt.subplots(figsize=(10,5))
plt.plot(july['Temp'])
plt.xlabel('Date', fontsize=16)
plt.ylabel('Temperature [$^\circ$C]', fontsize=16)
plt.ylim(7)
plt.xlim(july.index[0],july.index[743])
plt.xticks( fontsize = 14)
plt.yticks(fontsize = 14)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(outpath,"Hourly temperature in july"))
plt.show()
```

A.2.3 Time lags and correlation

```
#%% Time lags and correlation
A_Data = All_data[["Temp", "Precipitation", "WS", "WD", "Global rad",
                   "Direct rad", "Diffuse rad", "CC", "Production"]]

# define function for create N lags
def create_lags(Data, N):
    Data = Data.copy()
    for i in range(N):
        Data['lag' + str(i+1)] = Data['Production'].shift(i+1)
        Data['Temp lag' + str(i+1)] = Data['Temp'].shift(i+1)
        Data['WS lag' + str(i+1)] = Data['WS'].shift(i+1)
        Data['WD lag' + str(i+1)] = Data['WD'].shift(i+1)
        Data['CC lag' + str(i+1)] = Data['CC'].shift(i+1)
        Data['Precipitation lag' + str(i+1)] = Data[
            'Precipitation'].shift(i+1)
        Data['Global rad lag' + str(i+1)] = Data['Global rad'].shift(i+1)
        Data['Direct rad lag' + str(i+1)] = Data['Direct rad'].shift(i+1)
        Data['Diffuse rad lag' + str(i+1)] = Data['Diffuse rad'].shift(i+1)
    return Data

Data_R = create_lags(A_Data,6)

# Drop the first rows bacuse the values are NaNs now
Data_R = Data_R.dropna()

# Show the production at t, t-1, t-2, t-3 and t-4 for the first day
Production_lags = Data_R[['Production', 'lag1', 'lag2', 'lag3', 'lag4']]
Production_lags_day1 = Production_lags.loc[
    '2021-01-01 00:00:00':'2021-01-01 23:00:00']
Production_lags_day1.to_csv('Production_lags_day1.csv')

corr_p = Data_R.corr(method='pearson')
corr_k = Data_R.corr(method='kendall')
corr_s = Data_R.corr(method='spearman')

corr_pearson = corr_p.iloc[:,8]
corr_kendall = corr_k.iloc[:,8]
corr_spearman = corr_s.iloc[:,8]

corr_pearson = corr_pearson[corr_pearson > 0.5]
corr_kendall = corr_kendall[corr_kendall > 0.5]
corr_spearman = corr_spearman[corr_spearman > 0.5]

corr_pearson = pd.DataFrame(corr_pearson)
corr_pearson.rename(columns = {'Production' : 'Production P'}, inplace = True)
corr_pearson = corr_pearson['Production P'].round(decimals = 3)

corr_kendall = pd.DataFrame(corr_kendall)
corr_kendall.rename(columns = {'Production' : 'Production K'}, inplace = True)
corr_kendall = corr_kendall['Production K'].round(decimals = 3)

corr_spearman = pd.DataFrame(corr_spearman)
corr_spearman.rename(columns = {'Production' : 'Production S'},
                     inplace = True)
```

```
corr_spearman = corr_spearman['Production S'].round(decimals = 3)
corr = pd.concat((corr_pearson,corr_kendall,corr_spearman),axis=1)

corr.to_csv('corr.csv')

corr_P = corr_p.iloc[:,8]
corr_P = corr_P.sort_index()
cc_P = np.array(corr_P.iloc[0:7])
diff_P = np.array(corr_P.iloc[7:14])
dir_P = np.array(corr_P.iloc[14:21])
gi_P = np.array(corr_P.iloc[21:28])
pres_P = np.array(corr_P.iloc[28:35])
temp_P = np.array(corr_P.iloc[36:43])
wd_P = np.array(corr_P.iloc[43:50])
ws_P = np.array(corr_P.iloc[50:57])
prod_P = corr_P.iloc[[35,57,58,59,60,61,62]]
prod_P = pd.DataFrame(prod_P)
prod_P = prod_P.transpose()
prod_P.rename(columns = {'Production' : '0h', 'lag1' : '1h', 'lag2' : '2h',
                       'lag3' : '3h', 'lag4' : '4h', 'lag5' : '5h',
                       'lag6' : '6h'}, inplace = True)
prod_P = prod_P.transpose()

corr_K = corr_k.iloc[:,8]
corr_K = corr_K.sort_index()
cc_K = np.array(corr_K.iloc[0:7])
diff_K = np.array(corr_K.iloc[7:14])
dir_K = np.array(corr_K.iloc[14:21])
gi_K = np.array(corr_K.iloc[21:28])
pres_K = np.array(corr_K.iloc[28:35])
temp_K = np.array(corr_K.iloc[36:43])
wd_K = np.array(corr_K.iloc[43:50])
ws_K = np.array(corr_K.iloc[50:57])
prod_K = corr_K.iloc[[35,57,58,59,60,61,62]]
prod_K = pd.DataFrame(prod_K)
prod_K = prod_K.transpose()
prod_K.rename(columns = {'Production' : '0h', 'lag1' : '1h', 'lag2' : '2h',
                       'lag3' : '3h', 'lag4' : '4h', 'lag5' : '5h',
                       'lag6' : '6h'}, inplace = True)
prod_K = prod_K.transpose()

corr_S = corr_s.iloc[:,8]
corr_S = corr_S.sort_index()
cc_S = np.array(corr_S.iloc[0:7])
diff_S = np.array(corr_S.iloc[7:14])
dir_S = np.array(corr_S.iloc[14:21])
gi_S = np.array(corr_S.iloc[21:28])
```

```
pres_S = np.array(corr_S.iloc[28:35])
temp_S = np.array(corr_S.iloc[36:43])
wd_S = np.array(corr_S.iloc[43:50])
ws_S = np.array(corr_S.iloc[50:57])
prod_S = corr_K.iloc[[35,57,58,59,60,61,62]]
prod_S = pd.DataFrame(prod_S)
prod_S = prod_S.transpose()
prod_S.rename(columns = {'Production' : '0h', 'lag1' : '1h', 'lag2' : '2h',
                       'lag3' : '3h', 'lag4' : '4h', 'lag5' : '5h',
                       'lag6' : '6h'}, inplace = True)
prod_S = prod_S.transpose()

Y = [0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]

fig, axs = plt.subplots(3, 1, figsize=(8, 12))
axs[0].plot(prod_P, c = 'k', label = 'Production')
axs[0].plot(gi_P, c = 'deepskyblue', label = 'Global Irradiance')
axs[0].plot(diff_P, c = 'yellowgreen', label = 'Diffuse Irradiance')
axs[0].plot(dir_P, c = 'darkgreen', label = 'Direct Irradiance')
axs[0].plot(temp_P, c = 'red', label = 'Temperature')
axs[0].plot(wd_P, c = 'b', label = 'Wind Direction')
axs[0].plot(ws_P, c = 'indigo', label = 'Wind Speed')
axs[0].plot(pres_P, c = 'orange', label = 'Precipitation')
axs[0].plot(cc_P, c = 'pink', label = 'Cloud Cover')
axs[0].set_title('Pearson', fontsize=20)

axs[1].plot(prod_K, c = 'k', label = 'Production')
axs[1].plot(gi_K, c = 'deepskyblue', label = 'Global Irradiance')
axs[1].plot(diff_K, c = 'yellowgreen', label = 'Diffuse Irradiance')
axs[1].plot(dir_K, c = 'darkgreen', label = 'Direct Irradiance')
axs[1].plot(temp_K, c = 'red', label = 'Temperature')
axs[1].plot(wd_K, c = 'b', label = 'Wind Direction')
axs[1].plot(ws_K, c = 'indigo', label = 'Wind Speed')
axs[1].plot(pres_K, c = 'orange', label = 'Precipitation')
axs[1].plot(cc_K, c = 'pink', label = 'Cloud Cover')
axs[1].set_title('Kendall', fontsize=20)
axs[1].set_ylabel('Correlation coefficient', fontsize=18)

axs[2].plot(prod_S, c = 'k', label = 'Production')
axs[2].plot(gi_S, c = 'deepskyblue', label = 'Global Irradiance')
axs[2].plot(diff_S, c = 'yellowgreen', label = 'Diffuse Irradiance')
axs[2].plot(dir_S, c = 'darkgreen', label = 'Direct Irradiance')
axs[2].plot(temp_S, c = 'red', label = 'Temperature')
axs[2].plot(wd_S, c = 'b', label = 'Wind Direction')
axs[2].plot(ws_S, c = 'indigo', label = 'Wind Speed')
axs[2].plot(pres_S, c = 'orange', label = 'Precipitation')
axs[2].plot(cc_S, c = 'pink', label = 'Cloud Cover')
```

```
axs[2].set_title('Spearman', fontsize=20)

axs[2].set_xlabel('Time lag', fontsize=18)
for ax in axs:
    ax.legend(loc='upper right')
    ax.tick_params(axis='x', labelsize=14)
    ax.tick_params(axis='y', labelsize=14)
    ax.plot(prod_P.index,Y,'--')
    ax.set_xlim(0,6)
    ax.set_ylim(-0.2,1)
    ax.set_xticks(prod_P.index)
plt.tight_layout()
plt.savefig(path.join(outpath,"Correlation3"))
plt.show()
```

A.2.4 Data Division, scaling, fitting and transformation

```
#%% Data Division, scaling, fitting and transformation
corr = corr.transpose()
corr.columns
Data_reg = Data_R[corr.columns]

test_1= Data_reg.loc['2021-08-01 00:00:00':'2021-08-31 23:00:00']

X0 = Data_reg.loc['2020-12-31 23:00:00':'2021-07-31 23:00:00']
X1 = Data_reg.loc['2021-09-01 00:00:00':'2021-12-31 22:00:00']
X = pd.concat((X0,X1))



X_ran=X.copy().reset_index().drop(columns='UTC')
"""
def randomization(dataset,percentage):
    dataset=pd.DataFrame(dataset)
    index=int(np.ceil(percentage*len(dataset)))
    for i in range(1000000):
        print(i)
        shuffled=dataset.iloc[0:len(dataset) ,:]
        shuffled=shuffled.sample(frac=1)
        train = shuffled.iloc[0:index , :].values
        test=shuffled.iloc[index:len(dataset), :].values
        AV_train=train.mean(0)
        AV_train=AV_train.reshape(1,train.shape[1])
        STD_train=train.std(0)
        STD_train=STD_train.reshape(1,train.shape[1])
        AV_test=test.mean(0)
        AV_test=AV_test.reshape(1,train.shape[1])
        STD_test=test.std(0)
        STD_test=STD_test.reshape(1,train.shape[1])
        AV=np.concatenate((AV_train,AV_test),axis=0)
        STD=np.concatenate((STD_train, STD_test),axis=0)
        CV=STD/AV
        C1=CV[0,:].reshape(1,train.shape[1])
        C2=CV[1,:].reshape(1,train.shape[1])
        C12 = np.vstack([C1, C2])
        MaxC12=C12.max(0).reshape(1,train.shape[1])
        ERR=np.vstack([(abs((C1-C2)/MaxC12))])
        if np.all(ERR <=0.03):
            print("result"+str(i))
            result=shuffled
            break
    return result.iloc[0:index , :],result.iloc[index:len(dataset), :]


train_df,test_df=randomization(X_ran,percentage=0.8)

# To export the above result to be able change the plots and so on,
# on different days when Spyder gets closed at bedtime each day
train_df.to_csv('train_df.csv')
test_df.to_csv('test_df.csv')
"""
train_df=pd.read_csv('train_df.csv')
test_df=pd.read_csv('test_df.csv')

new_columns = train_df.columns.values
new_columns[0] = 'Index'
train_df.columns = new_columns
train_df = train_df.set_index('Index')
new_columns_1 = test_df.columns.values
new_columns_1[0] = 'Index'
test_df.columns = new_columns_1
test_df = test_df.set_index('Index')

train_des=train_df.describe()
test_des=test_df.describe()

x_train = train_df.loc[:,corr.columns != 'Production'].values

y_train = train_df['Production'].values

y_train_df = train_df['Production']

x_test_df = test_df.loc[:,corr.columns != 'Production']

x_test = np.array(x_test_df)
```

```
y_test = test_df['Production'].values

y_test_df = test_df['Production']


print("x_train shape: {}".format(x_train.shape))
print("x_test shape: {}".format(x_test.shape))
print("y_train shape: {}".format(y_train.shape))
print("y_test shape: {}".format(y_test.shape))

scaler = pre.StandardScaler()

X_train_scaled = scaler.fit_transform(x_train)
X_test_scaled = scaler.transform(x_test)

x_test_1 = test_1.loc[:,corr.columns != 'Production'].values
x_test_1_df = test_1.loc[:,corr.columns != 'Production']

y_test_1 = test_1['Production'].values

X_test_scaled_1 = scaler.transform(x_test_1)

scaler_y = pre.StandardScaler()
y_train_scaled = scaler_y.fit_transform(y_train.reshape(-1,1))
```

A.2.5 PCA

```
#%% PCA
pca = PCA(n_components = 11)
pca = PCA(.99)
X_train_scaled = pca.fit_transform(X_train_scaled)
X_test_scaled = pca.transform(X_test_scaled)
X_test_scaled_1 = pca.transform(X_test_scaled_1)
pca.explained_variance_ratio_.cumsum()
N_inputs = pca.n_components_
```

A.2.6 Performance

```
#%% Performance function to compare performances
def performance (true,pred, title):
    true=np.array(true).reshape(-1,1)
    pred=np.array(pred).reshape(-1,1)
    #Root Mean Square Error
    title=title
    error= true-pred
    error_sq=np.square (error)
    RMSE=np.sqrt(sum(error_sq)/error.shape[0])
    RMSE = np.round_((np.float64(RMSE)), decimals = 3)
    NRMSE=RMSE/613
    NRMSE = np.round_((np.float64(NRMSE)), decimals = 3)
    MAE = sum(abs(error))/error.shape[0]
    MAE = np.round_((np.float64(MAE)), decimals = 3)
    MAE_WP = MAE/613
    MAE_WP = np.round_((np.float64(MAE_WP)), decimals = 3)
    #Efficiency index
    R2=  1-(sum(error_sq)/sum(np.square(true-np.mean(true))))
    R2=np.round_((np.float64(R2)), decimals = 3)
    Performance_parameters=([['.',str(title)],
                             ['RMSE', RMSE],
                             ['NRMSE', NRMSE],
                             ['MAE',MAE],
                             ['MAE/Wp',MAE_WP],
                             ['R^2',R2]])
    Performance_parameters=np.array(Performance_parameters,dtype=object)
    return Performance_parameters
```

A.2.7 ANN Grid search

```
#%% ANN Grid search
def make_ANN_model(optimizer, initializer, batch_size):
# neurons, activations, hidden_layers
# optimizer, initializer, batch_size
# Making a linear pipeline (a stack) of neural networks layers.
    ANN_model = Sequential()
    # initializer = tf.keras.initializers.HeUniform()
# A single layer with x artificial neurons, and it expects y input variables
    ANN_model.add(Dense(12, input_dim = N_inputs,
                        kernel_initializer = initializer,
                        activation = 'selu'))
# Defining the Second hidden layer of the model
    ANN_model.add(Dense(12, kernel_initializer = initializer,
                        activation = 'selu'))
    # for i in range(hidden_layers):
    #     # Add one hidden layer
    #     ANN_model.add(Dense(neurons, kernel_initializer = initializer,
    #                         activation = activations))
# The output neuron is a single fully connected node since I will be
# predicting a single number (Power production)
    ANN_model.add(Dense(1, kernel_initializer = initializer,
                        activation = 'linear'))
# Compiling the model
    ANN_model.compile(loss = 'mean_squared_error', optimizer = optimizer)

    return ANN_model

# Listing all the parameters to try
```

```
Parameters = {
    # 'neurons': [6,8,10,12,14],
    # 'activations': ['relu', 'selu', 'elu'],
    # 'hidden_layers':[1,2,3],
    'batch_size': [32,64,128,256], 'optimizer': ['adam', 'rmsprop', 'sdg'],
    'initializer': ['lecun_normal', 'lecun_uniform']
              }

# Creating the ANN model
ANN_Model = KR(make_ANN_model, verbose=0)

# Defining a custom function to calculate accuracy
def Accuracy_Score(true,pred):
    MAPE = np.mean(100 * (np.abs(true-pred)/true))
    print('#'*70,'Accuracy:', 100-MAPE)
    return(100-MAPE)

custom_Scoring  = make_scorer(Accuracy_Score, greater_is_better=True)
# Creating the Grid search space
grid_search = GridSearchCV(estimator = ANN_Model,
                          param_grid = Parameters,
                          scoring = custom_Scoring,
                          cv = 5)

# Measuring how much time it took to find the best params
StartTime = time.time()

# Running Grid Search for different paramenters
grid_search.fit(X_train_scaled,y_train_scaled, verbose=1, epochs = 100)

EndTime=time.time()
print("########## Total Time Taken: ", round((EndTime-StartTime)/60), 'Minutes')

print('### Printing Best parameters ###')
grid_search.best_params_
```

A.2.8 ANN

```
#%% ANN
# Performance_30 = []
# Performance_train_30 = []
# Performance_1_30 = []
Results_test = []
Results_test_1 = []
Results_train = []
# Change range to how many runs you want
for _ in range (30):
# Making a linear pipeline (a stack) of neural networks layers.
```

```
    ANN_model = Sequential()
# A single layer with x artificial neurons, and it expects y input variables
    ANN_model.add(Dense(14, input_dim = N_inputs,
                        kernel_initializer = 'lecun_uniform',
                        activation = 'selu'))
# Defining the Second hidden layer of the model
    ANN_model.add(Dense(14, kernel_initializer = 'lecun_uniform',
                        activation = 'selu'))
# The output neuron is a single fully connected node since I will be
# predicting a single number (Power production)
    ANN_model.add(Dense(1, kernel_initializer = 'lecun_uniform',
                        activation = 'linear'))
# Compiling the model
    ANN_model.compile(loss = 'mean_squared_error', optimizer = 'rmsprop')
# Early stopping
    callback = tf.keras.callbacks.EarlyStopping(monitor = 'loss',
                                                patience = 10,
                                                restore_best_weights = True)
# Fitting the model
    ANN_model.fit(X_train_scaled, y_train_scaled,
                  epochs = 100, batch_size = 64, validation_split = 0.2,
                  callbacks = [callback], verbose = 1)
# Predicting on random test, August test and train
    Pred_test = ANN_model.predict(X_test_scaled)
    Pred_test = scaler_y.inverse_transform(Pred_test)
    Pred_test.min()
    Pred_test[Pred_test < 0] = 0

    Pred_test1 = ANN_model.predict(X_test_scaled_1)
    Pred_test1 = scaler_y.inverse_transform(Pred_test1)
    Pred_test1.min()
    Pred_test1[Pred_test1 < 0] = 0

    Pred_train = ANN_model.predict(X_train_scaled)
    Pred_train = scaler_y.inverse_transform(Pred_train)
    Pred_train.min()
    Pred_train[Pred_train < 0] = 0

# Performance
    print('     ')
    print('ANN:')
    ANN_performance= performance(y_test,Pred_test,'ANN')
    ANN_performance_train= performance(y_train,Pred_train,'ANN')
    print('overfitting',float(ANN_performance_train[1,1])/float(
        ANN_performance[1,1]))
    print("ANN performance test:")
    print(ANN_performance)
    print('     ')
    ANN_performance_1= performance(y_test_1,Pred_test1,'ANN')
    ANN_performance_train= performance(y_train,Pred_train,'ANN')
    print('overfitting',float(ANN_performance_train[1,1])/float(
        ANN_performance_1[1,1]))
    print("ANN performance august:")
    print(ANN_performance_1)

    Results_test.append(Pred_test)
    Results_test_1.append(Pred_test1)
    Results_train.append(Pred_train)
    # Performance_30.append(ANN_performance)
    # Performance_train_30.append(ANN_performance_train)
    # Performance_1_30.append(ANN_performance_1)


# Plotting
y_test_1_df = test_1['Production']
fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(y_test_1_df.index,y_test_1,color='k',linewidth=2)
for i in range(len(Results_test_1)):
    plt.plot(y_test_1_df.index,Results_test_1[i], color = 'rosybrown',linewidth=2)
plt.ylim(0)
plt.xlim(y_test_1_df.index[0],y_test_1_df.index[743])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Time [h]', fontsize=16)
plt.legend(['Actual','Predicted'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(
    outpath,"Predicted- and Actual Power Production ANN, august test"))

fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(y_test_1_df.index,y_test_1,color='k',linewidth=2)
for i in range(len(Results_test_1)):
    plt.plot(y_test_1_df.index,Results_test_1[i], color = 'rosybrown', linewidth=2)
plt.ylim(0)
plt.xlim(y_test_1_df.index[120],y_test_1_df.index[360])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Time [h]', fontsize=16)
plt.legend(['Actual','Predicted'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(
    outpath,"Predicted- and Actual Power Production ANN, august test, zoom"))

"""
# 30 ANN

Perf_30 = pd.DataFrame(np.hstack(Performance_30))
Perf_30 = np.hsplit(Perf_30,5)
Perf_30 = pd.DataFrame(np.vstack(Perf_30))
Perf_30 = Perf_30.iloc[:,[0,1,3,5,7,9,11]]
Perf_30.to_csv('Performance_30_runs.csv')

Perf_train_30 = pd.DataFrame(np.hstack(Performance_train_30))
Perf_train_30 = np.hsplit(Perf_train_30,5)
Perf_train_30 = pd.DataFrame(np.vstack(Perf_train_30))
Perf_train_30 = Perf_train_30.iloc[:,[0,1,3,5,7,9,11]]
Perf_train_30.to_csv('Performance_train_30_runs.csv')

Perf_1_30 = pd.DataFrame(np.hstack(Performance_1_30))
Perf_1_30 = np.hsplit(Perf_1_30,5)
Perf_1_30 = pd.DataFrame(np.vstack(Perf_1_30))
Perf_1_30 = Perf_1_30.iloc[:,[0,1,3,5,7,9,11]]
Perf_1_30.to_csv('Performance_August_30_runs.csv')

y_test_1_df = test_1['Production']
fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(y_test_1_df.index,y_test_1,color='k',linewidth=2)
for i in range(len(Results_test_1)):
    plt.plot(y_test_1_df.index,Results_test_1[i],linewidth=1)
plt.ylim(0)
plt.xlim(y_test_1_df.index[120],y_test_1_df.index[360])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Time [h]', fontsize=16)
plt.legend(['Actual','Predicted'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(
    outpath,
    "Predicted- and Actual Power Production 30 ANN, august test, zoom"))

fig = plt.figure(figsize=[10,5])
```

```
ax = fig.add_subplot(111)
plt.plot(y_test_1_df.index,y_test_1,color='k',linewidth=2)
for i in range(len(Results_test_1)):
    plt.plot(y_test_1_df.index,Results_test_1[i],linewidth=1)
plt.ylim(0)
plt.xlim(y_test_1_df.index[240],y_test_1_df.index[264])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Time [h]', fontsize=16)
plt.legend(['Actual','Predicted'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(
    outpath,
    "Predicted- and Actual Power Production 30 ANN, august test, megazoom"))
"""

def create_error(true, pred):
    all_errors = []
    for i in range(len(pred)):
        error= true[i]-pred[i]
        all_errors.append(error)
    return all_errors

all_errors_ANN = create_error(y_test,Results_test[0])
all_errors_ANN_1 = create_error(y_test_1,Results_test_1[0])

fig = plt.figure(figsize=[8, 4])
ax = fig.add_subplot(111)
plt.plot(y_test_df.index, all_errors_ANN,color='red',marker='.',
            linewidth=0,markersize=8,alpha=.4)
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Error (Actual-Predicted) [kWh]', fontsize=14)
plt.xlabel('Time [h]', fontsize=14)
plt.legend(['Error'], loc='best', fontsize=16)
plt.tight_layout()
plt.savefig(path.join(outpath,"Error random test, ANN"))
plt.show()

fig = plt.figure(figsize=[8, 4])
ax = fig.add_subplot(111)
plt.plot(y_test_1_df.index, all_errors_ANN_1,color='red',marker='.',
            linewidth=0,markersize=8,alpha=.4)
plt.xlim(y_test_1_df.index[0],y_test_1_df.index[743])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Error (Actual-Predicted) [kWh]', fontsize=14)
plt.xlabel('Time [h]', fontsize=14)
plt.legend(['Error'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(outpath,"Error august test, ANN"))
plt.show()
```

A.2.9 ANN QR

```
#%% ANN QR
def QR_loss(q, y, f):
    e = (y - f)
    return keras.backend.mean(keras.backend.maximum(q * e, (q - 1) * e),
                              axis=-1)
QUANTILES = [0.05, 0.1, 0.3, 0.5, 0.7, 0.9, 0.95]

initializer = tf.keras.initializers.HeNormal()
# Early stopping
callback = tf.keras.callbacks.EarlyStopping(monitor = 'loss',
                                            patience = 10,
                                            restore_best_weights = True)
def pred(X_train_scaled, y_train_scaled, q):
# Making a linear pipeline (a stack) of neural networks layers.
    ANN_model = Sequential()
# A single layer with x artificial neurons, and it expects y input variables
    ANN_model.add(Dense(14, input_dim = N_inputs,
                        kernel_initializer = 'lecun_uniform',
                        activation = 'selu'))
# Defining the Second hidden layer of the model
    ANN_model.add(Dense(14, kernel_initializer = 'lecun_uniform',
                        activation = 'selu'))
# The output neuron is a single fully connected node since I will be
# predicting a single number (Power production)
    ANN_model.add(Dense(1, kernel_initializer = 'lecun_uniform',
                        activation = 'linear'))
# Compiling the model
    ANN_model.compile(loss=lambda y, f: QR_loss(q, y, f), optimizer='rmsprop')
# Fitting the model
    ANN_model.fit(X_train_scaled, y_train_scaled,
                  epochs = 100, batch_size = 64, validation_split = 0.2,
                  callbacks = [callback], verbose = 1)
# Predicting on August test
    return ANN_model.predict(X_test_scaled_1)

pred = np.concatenate([pred(X_train_scaled, y_train_scaled, q)
     for q in QUANTILES], axis=1)
```

```
pred = scaler_y.inverse_transform(pred)
pred.min()
pred[pred < 0] = 0
pred = pd.DataFrame(pred, columns = ['5th', '10th', '30th', '50th', '70th',
                                     '90th', '95th'])
################### RANDOM #####################
def preds(X_train_scaled, y_train_scaled, q):
# Making a linear pipeline (a stack) of neural networks layers.
    ANN_model = Sequential()
# A single layer with x artificial neurons, and it expects y input variables
    ANN_model.add(Dense(14, input_dim = N_inputs,
                        kernel_initializer = 'lecun_uniform',
                        activation = 'selu'))
# Defining the Second hidden layer of the model
    ANN_model.add(Dense(14, kernel_initializer = 'lecun_uniform',
                        activation = 'selu'))
# The output neuron is a single fully connected node since I will be
# predicting a single number (Power production)
    ANN_model.add(Dense(1, kernel_initializer = 'lecun_uniform',
                        activation = 'linear'))
# Compiling the model
    ANN_model.compile(loss=lambda y, f: QR_loss(q, y, f), optimizer='rmsprop')
# Fitting the model
    ANN_model.fit(X_train_scaled, y_train_scaled,
                  epochs = 100, batch_size = 64, validation_split = 0.2,
                  callbacks = [callback], verbose = 1)
# Predicting on Random test
    return ANN_model.predict(X_test_scaled)

preds = np.concatenate([preds(X_train_scaled, y_train_scaled, q)
     for q in QUANTILES], axis=1)
preds = scaler_y.inverse_transform(preds)
preds.min()
preds[preds < 0] = 0
preds = pd.DataFrame(preds, columns= ['5th', '10th', '30th', '50th', '70th',
                                      '90th', '95th'])

################### TRAIN #####################
def preds_train(X_train_scaled, y_train_scaled, q):
# Making a linear pipeline (a stack) of neural networks layers.
    ANN_model = Sequential()
# A single layer with x artificial neurons, and it expects y input variables
    ANN_model.add(Dense(14, input_dim = N_inputs,
                        kernel_initializer = 'lecun_uniform',
                        activation = 'selu'))
# Defining the Second hidden layer of the model
    ANN_model.add(Dense(14, kernel_initializer = 'lecun_uniform',
                        activation = 'selu'))
# The output neuron is a single fully connected node since I will be
# predicting a single number (Power production)
    ANN_model.add(Dense(1, kernel_initializer = 'lecun_uniform',
                        activation = 'linear'))
# Compiling the model
    ANN_model.compile(loss=lambda y, f: QR_loss(q, y, f), optimizer='rmsprop')
# Fitting the model
    ANN_model.fit(X_train_scaled, y_train_scaled,
                  epochs = 100, batch_size = 64, validation_split = 0.2,
                  callbacks = [callback], verbose = 1)
# Predicting on train
    return ANN_model.predict(X_train_scaled)

Preds_train = np.concatenate([preds_train(X_train_scaled, y_train_scaled, q)
     for q in QUANTILES], axis=1)
Preds_train = scaler_y.inverse_transform(Preds_train)
Preds_train.min()
Preds_train[Preds_train < 0] = 0
Preds_train = pd.DataFrame(Preds_train, columns= ['5th', '10th', '30th',
                                                  '50th', '70th','90th',
                                                  '95th'])
""""error metrics August"""
L = pred[['5th']].values
U = pred[['95th']].values

L = pred[['10th']].values
U = pred[['90th']].values

mu = 0.90
mu = 0.80
eta = 50
epsilon = []
pi = []
for i in range(len(y_test_1)):
    if y_test_1[i] >= L[i] and y_test_1[i] <= U[i]:
        c = 1
    else:
        c = 0
    epsilon.append(c)
    a =np.float64((U[i]-L[i])/(y_test_1.max()-y_test_1.min()))
    pi.append(a)
Picp = sum(epsilon)/y_test_1.shape[0]
Pinaw = sum(pi)/y_test_1.shape[0]

if Picp >= mu:
    gamma = 1
```

```
else:
    gamma = 0
Cwc = Pinaw*(1+gamma*np.exp(-eta*(Picp-mu)))

""""error metrices random"""
Lower = preds[['5th']].values
Upper = preds[['95th']].values

Lower = preds[['10th']].values
Upper = preds[['90th']].values

mu = 0.90
mu = 0.80
eta = 50
epsilons = []
pis = []
for i in range(len(y_test)):
    if y_test[i] >= Lower[i] and y_test[i] <= Upper[i]:
        c = 1
    else:
        c = 0
    epsilons.append(c)
    b =np.float64((Upper[i]-Lower[i])/(y_test.max()-y_test.min()))
    pis.append(b)
picp = sum(epsilons)/y_test.shape[0]
pinaw = sum(pis)/y_test.shape[0]

if picp >= mu:
    gamma = 1
else:
    gamma = 0
cwc = pinaw*(1+gamma*np.exp(-eta*(picp-mu)))

# Performance
print('     ')
print('ANN QR:')
QR_performance= performance(y_test,preds['50th'].values,'QR')
QR_performance_train= performance(y_train,Preds_train['50th'].values,'QR')
print('overfitting',float(QR_performance_train[1,1])/float(
    QR_performance[1,1]))
print("QR performance test:")
print(QR_performance)
print(round(picp,3))
print(round(pinaw,3))
print(round(cwc,3))

print('     ')
```

```
QR_performance_1= performance(y_test_1,pred['50th'].values,'QR')
QR_performance_train= performance(y_train,Preds_train['50th'].values,'QR')
print('overfitting',float(QR_performance_train[1,1])/float(
    QR_performance_1[1,1]))
print("QR performance august:")
print(QR_performance_1)
print(round(Picp,3))
print(round(Pinaw,3))
print(round(Cwc,3))


Performance_QR = [round(picp,3), round(pinaw,3), round(cwc,3),
                round(Picp,3), round(Pinaw,3), round(Cwc,3)]
Performance_QR = pd.DataFrame(Performance_QR)
columns = ['Random PICP', 'Random PINAW',
                                        'Random CWC', 'August PICP',
                                        'August PINAW', 'August CWC']
columns = pd.DataFrame(columns)
Performance_QR = pd.concat([columns,Performance_QR], axis = 1).transpose()
Performance_QR.to_csv('Performance QR.csv')

# Plotting
# August
y_test_1_df = test_1['Production']
fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(y_test_1_df.index,y_test_1,color='k',linewidth=1, label='Actual')
plt.plot(y_test_1_df.index,pred['5th'],color='lightgreen',linewidth=1,
         label='5th percentile')
plt.plot(y_test_1_df.index,pred['10th'],color='lightgreen',linewidth=1,
         label='10th percentile')
plt.plot(y_test_1_df.index,pred['30th'],color='mediumseagreen',linewidth=1,
         label='30th percentile')
plt.plot(y_test_1_df.index,pred['50th'],color='seagreen',linewidth=1,
         label='50th percentile')
plt.plot(y_test_1_df.index,pred['70th'],color='green',linewidth=1,
         label='70th percentile')
plt.plot(y_test_1_df.index,pred['90th'],color='darkgreen',linewidth=1,
         label='90th percentile')
plt.plot(y_test_1_df.index,pred['95th'],color='darkgreen',linewidth=1,
         label='95th percentile')
plt.fill_between(y_test_1_df.index,pred['5th'], pred['95th'],
                 alpha=0.5,color='darkseagreen',
                 label='Interval (5th-95th)')
plt.ylim(0)
plt.xlim(y_test_1_df.index[0],y_test_1_df.index[743])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Time [h]', fontsize=16)
plt.legend(loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(
    outpath,"Predicted- and Actual Power Production QRNN"))

fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(y_test_1_df.index,y_test_1,color='k',linewidth=1, label='Actual')
plt.plot(y_test_1_df.index,pred['5th'],color='lightgreen',linewidth=1,
         label='5th percentile')
plt.plot(y_test_1_df.index,pred['10th'],color='lightgreen',linewidth=1,
         label='10th percentile')
plt.plot(y_test_1_df.index,pred['30th'],color='mediumseagreen',linewidth=1,
         label='30th percentile')
plt.plot(y_test_1_df.index,pred['50th'],color='seagreen',linewidth=1,
         label='50th percentile')
plt.plot(y_test_1_df.index,pred['70th'],color='green',linewidth=1,
         label='70th percentile')
plt.plot(y_test_1_df.index,pred['90th'],color='darkgreen',linewidth=1,
         label='90th percentile')
plt.plot(y_test_1_df.index,pred['95th'],color='darkgreen',linewidth=1,
         label='95th percentile')
plt.fill_between(y_test_1_df.index,pred['5th'], pred['95th'],
                 alpha=0.5,color='darkseagreen',
                 label='Interval (5th-95th)')
plt.ylim(0)
plt.xlim(y_test_1_df.index[120],y_test_1_df.index[360])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Time [h]', fontsize=16)
plt.legend(loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(
    outpath,"Predicted- and Actual Power Production QRNN, zoomed"))

fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(y_test_1_df.index,y_test_1,color='k',linewidth=1, label='Actual')
plt.plot(y_test_1_df.index,pred['5th'],color='lightgreen',linewidth=1,
         label='5th percentile')
plt.plot(y_test_1_df.index,pred['10th'],color='lightgreen',linewidth=1,
         label='10th percentile')
plt.plot(y_test_1_df.index,pred['30th'],color='mediumseagreen',linewidth=1,
         label='30th percentile')
plt.plot(y_test_1_df.index,pred['50th'],color='seagreen',linewidth=1,
         label='50th percentile')
plt.plot(y_test_1_df.index,pred['70th'],color='green',linewidth=1,
         label='70th percentile')
plt.plot(y_test_1_df.index,pred['90th'],color='darkgreen',linewidth=1,
         label='90th percentile')
plt.plot(y_test_1_df.index,pred['95th'],color='darkgreen',linewidth=1,
         label='95th percentile')
plt.fill_between(y_test_1_df.index,pred['5th'], pred['95th'],
                 alpha=0.5,color='darkseagreen',
                 label='Interval (5th-95th)')
plt.ylim(0)
plt.xlim(y_test_1_df.index[216],y_test_1_df.index[264])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Time [h]', fontsize=16)
plt.legend(loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(
    outpath,"Predicted- and Actual Power Production QRNN, megazoomed"))

# Random
y_test_df = pd.DataFrame(y_test_df,columns=['Production'])
y_test_df = y_test_df.sort_index()
fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(y_test_df.index,y_test,color='k',linewidth=1, label='Actual')
plt.plot(y_test_df.index,preds['5th'],color='lightgreen',linewidth=1,
         label='5th percentile')
plt.plot(y_test_df.index,preds['10th'],color='lightgreen',linewidth=1,
         label='10th percentile')
plt.plot(y_test_df.index,preds['30th'],color='mediumseagreen',linewidth=1,
         label='30th percentile')
plt.plot(y_test_df.index,preds['50th'],color='seagreen',linewidth=1,
         label='50th percentile')
plt.plot(y_test_df.index,preds['70th'],color='green',linewidth=1,
         label='70th percentile')
plt.plot(y_test_df.index,preds['90th'],color='darkgreen',linewidth=1,
         label='90th percentile')
plt.plot(y_test_df.index,preds['95th'],color='darkgreen',linewidth=1,
         label='95th percentile')
plt.fill_between(y_test_df.index,preds['5th'], preds['95th'],
                  alpha=0.5,color='darkseagreen',
                  label='Interval (5th-95th)')
plt.ylim(0)
plt.xlim(y_test_df.index[0],y_test_df.index[743])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Hours [h]', fontsize=16)
plt.legend(loc='best', fontsize=16)
plt.tight_layout()
plt.savefig(path.join(
    outpath,"Predicted- and Actual Power Production QRNN, random"))

fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(y_test_df.index,y_test,color='k',linewidth=1, label='Actual')
plt.plot(y_test_df.index,preds['5th'],color='lightgreen',linewidth=1,
         label='5th percentile')
plt.plot(y_test_df.index,preds['10th'],color='lightgreen',linewidth=1,
         label='10th percentile')
plt.plot(y_test_df.index,preds['30th'],color='mediumseagreen',linewidth=1,
         label='30th percentile')
plt.plot(y_test_df.index,preds['50th'],color='seagreen',linewidth=1,
         label='50th percentile')
plt.plot(y_test_df.index,preds['70th'],color='green',linewidth=1,
         label='70th percentile')
plt.plot(y_test_df.index,preds['90th'],color='darkgreen',linewidth=1,
         label='90th percentile')
plt.plot(y_test_df.index,preds['95th'],color='darkgreen',linewidth=1,
         label='95th percentile')
plt.fill_between(y_test_df.index,preds['5th'], preds['95th'],
                  alpha=0.5,color='darkseagreen',
                  label='Interval (5th-95th)')
plt.ylim(0)
plt.xlim(y_test_df.index[0],y_test_df.index[150])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Hours [h]', fontsize=16)
plt.legend(loc='best', fontsize=16)
plt.tight_layout()
plt.savefig(path.join(
    outpath,"Predicted- and Actual Power Production QRNN, random, zoomed"))
```

A.2.10 SVR

```
#%% SVR
#Function to calculate the best parameters for SVR
```

```
def SVR_best_parameters(X_train, y_train, n_fold):
    parameters={'C':[8,9,10,11,12], 'gamma':[1, 0.1, 0.01, 0.001],
                'epsilon':[0.01, 0.05, 0.08]}
    grid_search=GridSearchCV(estimator=SVR(),
                             param_grid=parameters,
                             cv=n_fold,
                             n_jobs = -1, verbose=1)
    grid_search=grid_search.fit(X_train, y_train)
    best_score=grid_search.best_score_
    print(best_score)
    best_model=grid_search.best_params_
    return best_model

parameters = SVR_best_parameters(X_train_scaled, y_train_scaled.ravel(), 5)
print(parameters)

"""
parameters={'C':[1,2,3,7,10], 'gamma':[1, 0.1, 0.01, 0.001],
                'epsilon':[0.01, 0.05, 0.08]}
    {'C': 10, 'epsilon': 0.05, 'gamma': 0.01}

parameters={'C':[8,9,10,11,12], 'gamma':[1, 0.1, 0.01, 0.001],
                'epsilon':[0.01, 0.05, 0.08]}
    {'C': 9, 'epsilon': 0.05, 'gamma': 0.01}

"""

SVR_model = SVR(kernel='rbf',C=9,gamma=0.01,
                epsilon=0.05).fit(X_train_scaled,y_train_scaled.ravel())

predict_y_array = SVR_model.predict(X_test_scaled)
predict_y_array_1 = SVR_model.predict(X_test_scaled_1)
predict_y_train_array = SVR_model.predict(X_train_scaled)

predict_y_array=scaler_y.inverse_transform(predict_y_array.reshape(-1, 1))
predict_y_array[predict_y_array < 0] = 0

predict_y_array_1=scaler_y.inverse_transform(predict_y_array_1.reshape(-1, 1))
predict_y_array_1[predict_y_array_1 < 0] = 0

predict_y_train_array=scaler_y.inverse_transform(
    predict_y_train_array.reshape(-1, 1))
predict_y_train_array[predict_y_train_array < 0] = 0

print('     ')
print('SVR:')
SVR_performance= performance(y_test,predict_y_array,'SVR')
```

```
SVR_performance_train= performance(y_train,predict_y_train_array,'SVR')
print('overfitting',float(SVR_performance_train[1,1])/float(
    SVR_performance[1,1]))
print("SVR performance test:")
print(SVR_performance)

print('     ')
SVR_performance_1= performance(y_test_1,predict_y_array_1,'SVR')
SVR_performance_train= performance(y_train,predict_y_train_array,'SVR')
print('overfitting',float(SVR_performance_train[1,1])/float(
    SVR_performance_1[1,1]))
print("SVR performance august:")
print(SVR_performance_1)

y_test_1_df = test_1['Production']
fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(y_test_1_df.index,y_test_1,color='k',linewidth=2)
plt.plot(y_test_1_df.index,predict_y_array_1,color='g',linewidth=2)
plt.ylim(0)
plt.xlim(y_test_1_df.index[0],y_test_1_df.index[743])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Time [h]', fontsize=16)
plt.legend(['Actual','Predicted'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(
    outpath,"Predicted- and Actual Power Production SVR, august test"))

fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(y_test_1_df.index,y_test_1,color='k',linewidth=2)
plt.plot(y_test_1_df.index,predict_y_array_1,color='g',linewidth=2)
plt.ylim(0)
plt.xlim(y_test_1_df.index[120],y_test_1_df.index[360])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Time [h]', fontsize=16)
plt.legend(['Actual','Predicted'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(
    outpath,"Predicted- and Actual Power Production SVR, august test, zoom"))
```

```
# Plot the predicted- vs actual Production SVR august
fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(y_test_1_df,predict_y_array_1,color='b',marker='.',linewidth=0,
            markersize=12,alpha=.4)
plt.plot([0,predict_y_array_1.max()],[0,predict_y_array_1.max()],'k')
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Predicted Hourly Production [kWh]', fontsize=16)
plt.xlabel('Actual Hourly Production [kWh]', fontsize=16)
plt.tight_layout()
plt.savefig(path.join(outpath,"Predicted vs Actual Power Production SVR"))
plt.show()

def create_error(true, pred):
    all_errors = []
    for i in range(len(pred)):
        error= true[i]-pred[i]
        all_errors.append(error)
    return all_errors

all_errors_SVR = create_error(y_test,predict_y_array)
all_errors_SVR_1 = create_error(y_test_1,predict_y_array_1)

fig = plt.figure(figsize=[8, 4])
ax = fig.add_subplot(111)
plt.plot(y_test_df.index, all_errors_SVR,color='red',marker='.',
            linewidth=0,markersize=8,alpha=.4)
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Error (Actual-Predicted) [kWh]', fontsize=14)
plt.xlabel('Time [h]', fontsize=14)
plt.legend(['Error'], loc='best', fontsize=16)
plt.tight_layout()
plt.savefig(path.join(outpath,"Error random test"))
plt.show()

fig = plt.figure(figsize=[8, 4])
ax = fig.add_subplot(111)
plt.plot(y_test_1_df.index, all_errors_SVR_1,color='red',marker='.',
            linewidth=0,markersize=8,alpha=.4)
plt.xlim(y_test_1_df.index[0],y_test_1_df.index[743])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Error (Actual-Predicted) [kWh]', fontsize=14)
plt.xlabel('Time [h]', fontsize=14)
plt.legend(['Error'], loc='best', fontsize=16)
```

```
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(outpath,"Error august test"))
plt.show()
```

A.2.11 ARIMA

```
#%% ARIMA
#New train and test
X = All_data[['Production']]
test = X.loc['2021-08-01 00:00:00':'2021-08-31 23:00:00'].values
train1 = X.loc['2020-12-31 23:00:00':'2021-07-31 23:00:00']
train2 = X.loc['2021-09-01 00:00:00':'2021-12-31 22:00:00']
train = pd.concat((train1,train2)).values
history = [x for x in train]
Y = All_data[['Global rad']]
ytest = Y.loc['2021-08-01 00:00:00':'2021-08-31 23:00:00'].values
ytrain1 = Y.loc['2020-12-31 23:00:00':'2021-07-31 23:00:00']
ytrain2 = Y.loc['2021-09-01 00:00:00':'2021-12-31 22:00:00']
ytrain = pd.concat((ytrain1,ytrain2)).values
exog = [x for x in ytrain]
xexog = [x for x in ytest]
predictions = list()
for t in range(len(test)):
    model = ARIMA(history, exog = exog, order=(1, 0, 1))
    model_fit = model.fit()
    start = len(train)
    end = len(train) + len(test) - 1
    predictions = model_fit.predict(start, end, exog = xexog)
print('     ')
#printing performance
ARIMA_performance = performance(test, predictions,'ARIMA')
print("ARIMA performance:")
print(ARIMA_performance)

ARIMA_test_df = X.loc['2021-08-01 00:00:00':'2021-08-31 23:00:00']
fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(ARIMA_test_df.index,test,color='k',linewidth=2)
plt.plot(ARIMA_test_df.index,predictions,color='saddlebrown',linewidth=2)
plt.ylim(0)
plt.xlim(ARIMA_test_df.index[0],ARIMA_test_df.index[743])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Time [h]', fontsize=16)
plt.legend(['Actual','Predicted'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
```

```
plt.tight_layout()
plt.savefig(path.join(
    outpath,"Predicted- and Actual Power Production ARIMA, august test"))

fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(ARIMA_test_df.index,test,color='k',linewidth=2)
plt.plot(ARIMA_test_df.index,predictions,color='saddlebrown',linewidth=2)
plt.ylim(0)
plt.xlim(ARIMA_test_df.index[120],ARIMA_test_df.index[360])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Time [h]', fontsize=16)
plt.legend(['Actual','Predicted'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(
    outpath,"Predicted- and Actual Power Production ARIMA, august test, zoom"))

def create_error(true, pred):
    all_errors = []
    for i in range(len(pred)):
        error= true[i]-pred[i]
        all_errors.append(error)
    return all_errors

all_errors_ARIMA = create_error(test,predictions)

fig = plt.figure(figsize=[8, 4])
ax = fig.add_subplot(111)
plt.plot(ARIMA_test_df.index, all_errors_ARIMA,color='red',marker='.',
            linewidth=0,markersize=8,alpha=.4)
plt.xlim(ARIMA_test_df.index[0],ARIMA_test_df.index[743])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Error (Actual-Predicted) [kWh]', fontsize=14)
plt.xlabel('Time [h]', fontsize=14)
plt.legend(['Error'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(outpath,"Error august test, ARIMA"))
plt.show()
```

A.2.12 Ensemble ANN & SVR

```
#%% Ensemble ANN, SVR
def ann_model():
```

```
# Making a linear pipeline (a stack) of neural networks layers.
    ANN_model = Sequential()
# A single layer with x artificial neurons, and it expects y input variables
    ANN_model.add(Dense(14, input_dim = N_inputs,
                        kernel_initializer = 'lecun_uniform',
                        activation = 'selu'))
# Defining the Second hidden layer of the model
    ANN_model.add(Dense(14, kernel_initializer = 'lecun_uniform',
                        activation = 'selu'))
# The output neuron is a single fully connected node since I will be
# predicting a single number (Power production)
    ANN_model.add(Dense(1, kernel_initializer = 'lecun_uniform',
                        activation = 'linear'))
# Compiling the model
    ANN_model.compile(loss = 'mean_squared_error', optimizer = 'rmsprop')
    return ANN_model
# Early stopping
callback = tf.keras.callbacks.EarlyStopping(monitor = 'loss', patience = 10,
                                            restore_best_weights = True)
learner_1 = KerasRegressor(build_fn=ann_model, epochs=100, batch_size=64,
                           validation_split=0.2, callbacks=[callback],
                           verbose=1)
learner_2 = SVR(kernel='rbf',C=9,gamma=0.01,
                epsilon=0.05)

# evaluate a weighted average ensemble for regression with rankings for model
# weights
def get_models():
models = list()
models.append(('ann', learner_1))
models.append(('svr', learner_2))
return models

models = get_models()

# evaluate each base model
def evaluate_models(models, X_train, X_val, y_train, y_val):
    # fit and evaluate the models
    scores = list()
    for name, model in models:
        # fit the models
        model.fit(X_train_scaled, y_train_scaled.ravel())
        # evaluate the model
        yhat = model.predict(X_test_scaled)
        yhat = scaler_y.inverse_transform(yhat.reshape(-1, 1))
        yhat.min()
        yhat[yhat < 0] = 0
        mae = mean_absolute_error(y_test, yhat)
        # store the performance
        scores.append(-mae)
        # report model performance
    return scores

# fit and evaluate each model
scores = evaluate_models(models, X_train_scaled, X_test_scaled,
                         y_train_scaled.ravel(), y_test)
ranking = 1 + argsort(argsort(scores))
for i in range(len(models)):
print('>%s: %.3f' % (models[i][0], scores[i]))
######################################
voting = VotingRegressor([('ANN', learner_1), ('SVR', learner_2)],
                         weights=ranking)

voting.fit(X_train_scaled,y_train_scaled.ravel())

ens_preds = voting.predict(X_test_scaled)
ens_preds = scaler_y.inverse_transform(ens_preds.reshape(-1, 1))
ens_preds.min()
ens_preds[ens_preds < 0] = 0

preds_train = voting.predict(X_train_scaled)
preds_train = scaler_y.inverse_transform(preds_train.reshape(-1, 1))
preds_train.min()
preds_train[preds_train < 0] = 0

print('     ')
print('Ensemble:')
Ensemble_performance= performance(y_test,ens_preds,'ANN')
Ensemble_performance_train= performance(y_train,preds_train,'ANN')
print('overfitting',float(Ensemble_performance_train[1,1])/float(
    Ensemble_performance[1,1]))
print("Ensemble performance test:")
print(Ensemble_performance)

preds_1 = voting.predict(X_test_scaled_1)
preds_1 = scaler_y.inverse_transform(preds_1.reshape(-1, 1))
preds_1.min()
preds_1[preds_1 < 0] = 0

print('     ')
Ensemble_performance_1= performance(y_test_1,preds_1,'ANN')
Ensemble_performance_train= performance(y_train,preds_train,'ANN')
print('overfitting',float(Ensemble_performance_train[1,1])/float(
    Ensemble_performance_1[1,1]))
```

```
print("Ensemble performance august:")
print(Ensemble_performance_1)

y_test_1_df = test_1['Production']
fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(y_test_1_df.index,y_test_1,color='k',linewidth=2)
plt.plot(y_test_1_df.index,preds_1,color='yellowgreen',linewidth=2)
plt.ylim(0)
plt.xlim(y_test_1_df.index[0],y_test_1_df.index[743])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Time [h]', fontsize=16)
plt.legend(['Actual','Predicted'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(
    outpath,"Predicted- and Actual Power Production ensemble SVR and ANN"))

fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(y_test_1_df.index,y_test_1,color='k',linewidth=2)
plt.plot(y_test_1_df.index,preds_1,color='yellowgreen',linewidth=2)
plt.ylim(0)
plt.xlim(y_test_1_df.index[120],y_test_1_df.index[360])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Time [h]', fontsize=16)
plt.legend(['Actual','Predicted'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(
    outpath,
    "Predicted- and Actual Power Production ensemble SVR and ANN, zoomed"))

def create_error(true, pred):
    all_errors = []
    for i in range(len(pred)):
        error= true[i]-pred[i]
        all_errors.append(error)
    return all_errors

errors_ens_SVRANN = create_error(y_test,ens_preds)
errors_ens_SVRANN_1 = create_error(y_test_1,preds_1)
```

```
fig = plt.figure(figsize=[8, 4])
ax = fig.add_subplot(111)
plt.plot(y_test_df.index, errors_ens_SVRANN,color='red',marker='.',
            linewidth=0,markersize=8,alpha=.4)
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Error (Actual-Predicted) [kWh]', fontsize=14)
plt.xlabel('Time [h]', fontsize=14)
plt.legend(['Error'], loc='best', fontsize=16)
plt.tight_layout()
plt.savefig(path.join(outpath,"Error random test ens SVRANN"))
plt.show()

fig = plt.figure(figsize=[8, 4])
ax = fig.add_subplot(111)
plt.plot(y_test_1_df.index, errors_ens_SVRANN_1,color='red',marker='.',
            linewidth=0,markersize=8,alpha=.4)
plt.xlim(y_test_1_df.index[0],y_test_1_df.index[743])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Error (Actual-Predicted) [kWh]', fontsize=14)
plt.xlabel('Time [h]', fontsize=14)
plt.legend(['Error'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(outpath,"Error august test ens SVRANN"))
plt.show()
```

A.2.13 ANN ensemble

```
#%% ANN ensemble
def ann_model1():
# Making a linear pipeline (a stack) of neural networks layers.
    ANN_model = Sequential()
# A single layer with x artificial neurons, and it expects y input variables
    ANN_model.add(Dense(14, input_dim = N_inputs,
                        kernel_initializer = 'lecun_uniform',
                        activation = 'selu'))
# Defining the Second hidden layer of the model
    ANN_model.add(Dense(14, kernel_initializer = 'lecun_uniform',
                        activation = 'selu'))
# The output neuron is a single fully connected node since I will be
# predicting a single number (Power production)
    ANN_model.add(Dense(1, kernel_initializer = 'lecun_uniform',
                        activation = 'linear'))
# Compiling the model
    ANN_model.compile(loss = 'mean_squared_error', optimizer = 'rmsprop')
    return ANN_model

# Early stopping
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=10,
                                            restore_best_weights=True)

def ann_model2():
# Making a linear pipeline (a stack) of neural networks layers.
    ANN_model = Sequential()
    initializer = tf.keras.initializers.HeUniform()
# A single layer with x artificial neurons, and it expects y input variables
    ANN_model.add(Dense(12, input_dim = N_inputs,
                        kernel_initializer = initializer,
                        activation = 'relu'))
# Defining the Second hidden layer of the model
    ANN_model.add(Dense(12, kernel_initializer = initializer,
                        activation = 'relu'))
# The output neuron is a single fully connected node since I will be
# predicting a single number (Power production)
    ANN_model.add(Dense(1, kernel_initializer = initializer,
                        activation = 'linear'))
# Compiling the model
    ANN_model.compile(loss = 'mean_squared_error', optimizer = 'adam')
    return ANN_model

learners = []
for _ in range (15):
    learner1 = KerasRegressor(build_fn=ann_model1, epochs=100, batch_size=64,
                             validation_split=0.2,
                  callbacks=[callback], verbose=1)
    learner2 = KerasRegressor(build_fn=ann_model2, epochs=100, batch_size=64,
                         validation_split=0.2,
              callbacks=[callback], verbose=1)
    learners.append(learner1)
    learners.append(learner2)

# evaluate a weighted average ensemble for regression with rankings for model
# weights
def get_models():
    models = list()
    models.append(('ann', learners[0]))
    models.append(('ann1', learners[1]))
    models.append(('ann2', learners[2]))
    models.append(('ann3', learners[3]))
    models.append(('ann4', learners[4]))
    models.append(('ann5', learners[5]))
    models.append(('ann6', learners[6]))
    models.append(('ann7', learners[7]))
    models.append(('ann8', learners[8]))
    models.append(('ann9', learners[9]))
    models.append(('ann10', learners[10]))
    models.append(('ann11', learners[11]))
    models.append(('ann12', learners[12]))
    models.append(('ann13', learners[13]))
    models.append(('ann14', learners[14]))
    models.append(('ann15', learners[15]))
    models.append(('ann16', learners[16]))
    models.append(('ann17', learners[17]))
    models.append(('ann18', learners[18]))
    models.append(('ann19', learners[19]))
    models.append(('ann20', learners[20]))
    models.append(('ann21', learners[21]))
    models.append(('ann22', learners[22]))
    models.append(('ann23', learners[23]))
    models.append(('ann24', learners[24]))
    models.append(('ann25', learners[25]))
    models.append(('ann26', learners[26]))
    models.append(('ann27', learners[27]))
    models.append(('ann28', learners[28]))
    models.append(('ann29', learners[29]))
    return models

models = get_models()

# evaluate each base model
def evaluate_models(models, X_train, X_val, y_train, y_val):
    # fit and evaluate the models
    scores = list()
    for name, model in models:
        # fit the models
        model.fit(X_train_scaled, y_train_scaled.ravel())
        # evaluate the model
        yhat = model.predict(X_test_scaled)
        yhat = scaler_y.inverse_transform(yhat.reshape(-1, 1))
        yhat.min()
        yhat[yhat < 0] = 0
        mae = mean_absolute_error(y_test, yhat)
        # store the performance
        scores.append(-mae)
        # report model performance
    return scores

# fit and evaluate each model
scores = evaluate_models(models, X_train_scaled, X_test_scaled,
                         y_train_scaled.ravel(), y_test)
ranking = 1 + argsort(argsort(scores))
for i in range(len(models)):
print('>%s: %.3f' % (models[i][0], scores[i]))

voting = VotingRegressor([('ANN0',learners[0]),('ANN1',learners[1]),
                          ('ANN2',learners[2]),('ANN3',learners[3]),
                          ('ANN4',learners[4]),('ANN5',learners[5]),
                          ('ANN6',learners[6]),('ANN7',learners[7]),
                          ('ANN8',learners[8]),('ANN9',learners[9]),
                          ('ANN10',learners[10]),('ANN11',learners[11]),
                          ('ANN12',learners[12]),('ANN13',learners[13]),
                          ('ANN14',learners[14]),('ANN15',learners[15]),
                          ('ANN16',learners[16]),('ANN17',learners[17]),
                          ('ANN18',learners[18]),('ANN19',learners[19]),
                          ('ANN20',learners[20]),('ANN21',learners[21]),
                          ('ANN22',learners[22]),('ANN23',learners[23]),
                          ('ANN24',learners[24]),('ANN25',learners[25]),
                          ('ANN26',learners[26]),('ANN27',learners[27]),
                          ('ANN28',learners[28]),('ANN29',learners[29]),
                          ], weights=ranking).fit(X_train_scaled,
                                                  y_train_scaled.ravel())

voting = VotingRegressor(models, weights=ranking).fit(X_train_scaled,
                                                      y_train_scaled.ravel())

ann_preds = voting.predict(X_test_scaled)
ann_preds = scaler_y.inverse_transform(ann_preds.reshape(-1, 1))
ann_preds.min()
ann_preds[ann_preds < 0] = 0

ann_preds_train = voting.predict(X_train_scaled)
ann_preds_train = scaler_y.inverse_transform(ann_preds_train.reshape(-1, 1))
ann_preds_train.min()
ann_preds_train[ann_preds_train < 0] = 0

print('     ')
print('Ensemble:')
ANN_Ensemble_performance= performance(y_test,ann_preds,'ANN')
ANN_Ensemble_performance_train= performance(y_train,ann_preds_train,'ANN')
print('overfitting',float(ANN_Ensemble_performance_train[1,1])/float(
    ANN_Ensemble_performance[1,1]))
print("Ensemble performance test:")
print(ANN_Ensemble_performance)

ann_preds_1 = voting.predict(X_test_scaled_1)
ann_preds_1 = scaler_y.inverse_transform(ann_preds_1.reshape(-1, 1))
```

```
ann_preds_1.min()
ann_preds_1[ann_preds_1 < 0] = 0

print('     ')
ANN_Ensemble_performance_1= performance(y_test_1,ann_preds_1,'ANN')
ANN_Ensemble_performance_train= performance(y_train,ann_preds_train,'ANN')
print('overfitting',float(ANN_Ensemble_performance_train[1,1])/float(
    ANN_Ensemble_performance_1[1,1]))
print("Ensemble performance august:")
print(ANN_Ensemble_performance_1)

y_test_1_df = test_1['Production']
fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(y_test_1_df.index,y_test_1,color='k',linewidth=2)
plt.plot(y_test_1_df.index,ann_preds_1,color='cornflowerblue',linewidth=2)
plt.ylim(0)
plt.xlim(y_test_1_df.index[0],y_test_1_df.index[743])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Time [h]', fontsize=16)
plt.legend(['Actual','Predicted'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(
    outpath,"Predicted- and Actual Power Production ensemble two ANN"))

fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(y_test_1_df.index,y_test_1,color='k',linewidth=2)
plt.plot(y_test_1_df.index,ann_preds_1,color='cornflowerblue',linewidth=2)
plt.ylim(0)
plt.xlim(y_test_1_df.index[120],y_test_1_df.index[360])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Time [h]', fontsize=16)
plt.legend(['Actual','Predicted'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(
    outpath,"Predicted- and Actual Power Production ensemble two ANN, zoomed"))

def create_error(true, pred):
    all_errors = []
    for i in range(len(pred)):
        error= true[i]-pred[i]
        all_errors.append(error)
    return all_errors

errors_ens_ANN = create_error(y_test,ann_preds)
errors_ens_ANN_1 = create_error(y_test_1,ann_preds_1)

fig = plt.figure(figsize=[8, 4])
ax = fig.add_subplot(111)
plt.plot(y_test_df.index, errors_ens_ANN,color='red',marker='.',
            linewidth=0,markersize=8,alpha=.4)
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Error (Actual-Predicted) [kWh]', fontsize=14)
plt.xlabel('Time [h]', fontsize=14)
plt.legend(['Error'], loc='best', fontsize=16)
plt.tight_layout()
plt.savefig(path.join(outpath,"Error random test ens ANN"))
plt.show()

fig = plt.figure(figsize=[8, 4])
ax = fig.add_subplot(111)
plt.plot(y_test_1_df.index, errors_ens_ANN_1,color='red',marker='.',
            linewidth=0,markersize=8,alpha=.4)
plt.xlim(y_test_1_df.index[0],y_test_1_df.index[743])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Error (Actual-Predicted) [kWh]', fontsize=14)
plt.xlabel('Time [h]', fontsize=14)
plt.legend(['Error'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(outpath,"Error august test ens ANN"))
plt.show()
```

A.2.14 ANN NWP

```
#%% ANN NWP
A_Data_NWP = All_data[["Temp", "Precipitation", "WS", "WD", "Global rad",
                   "Direct rad", "Diffuse rad", "CC", "Production"]]

Data_NWP = A_Data_NWP.copy()

NWP_test_1= Data_NWP.loc['2021-08-01 00:00:00':'2021-08-31 23:00:00']

NWP_x0 = Data_NWP.loc['2020-12-31 23:00:00':'2021-07-31 23:00:00']
NWP_x1 = Data_NWP.loc['2021-09-01 00:00:00':'2021-12-31 22:00:00']
NWP_x = pd.concat((NWP_x0,NWP_x1))
```

```
NWP_X_ran=NWP_x.copy().reset_index().drop(columns='UTC')
"""
def randomization(dataset,percentage):
    dataset=pd.DataFrame(dataset)
    index=int(np.ceil(percentage*len(dataset)))
    for i in range(1000000):
        print(i)
        shuffled=dataset.iloc[0:len(dataset) ,:]
        shuffled=shuffled.sample(frac=1)
        train = shuffled.iloc[0:index , :].values
        test=shuffled.iloc[index:len(dataset), :].values
        AV_train=train.mean(0)
        AV_train=AV_train.reshape(1,train.shape[1])
        STD_train=train.std(0)
        STD_train=STD_train.reshape(1,train.shape[1])
        AV_test=test.mean(0)
        AV_test=AV_test.reshape(1,train.shape[1])
        STD_test=test.std(0)
        STD_test=STD_test.reshape(1,train.shape[1])

        AV=np.concatenate((AV_train,AV_test),axis=0)
        STD=np.concatenate((STD_train, STD_test),axis=0)
        CV=STD/AV
        C1=CV[0,:].reshape(1,train.shape[1])
        C2=CV[1,:].reshape(1,train.shape[1])
        C12 = np.vstack([C1, C2])
        MaxC12=C12.max(0).reshape(1,train.shape[1])
        ERR=np.vstack([(abs((C1-C2)/MaxC12))])
        if np.all(ERR <=0.03):
            print("result"+str(i))
            result=shuffled
            break
    return result.iloc[0:index , :],result.iloc[index:len(dataset), :]


NWP_train_df,NWP_test_df=randomization(NWP_X_ran,percentage=0.8)
# To export the above result to be able change the plots and so on,
# on different days when Spyder gets closed at bedtime each day
NWP_train_df.to_csv('NWP_train_df.csv')
NWP_test_df.to_csv('NWP_test_df.csv')
"""
NWP_train_df=pd.read_csv('NWP_train_df.csv')
NWP_test_df=pd.read_csv('NWP_test_df.csv')

NWP_new_columns = NWP_train_df.columns.values
NWP_new_columns[0] = 'Index'
```

```
NWP_train_df.columns = NWP_new_columns
NWP_train_df = NWP_train_df.set_index('Index')
NWP_new_columns_1 = NWP_test_df.columns.values
NWP_new_columns_1[0] = 'Index'
NWP_test_df.columns = NWP_new_columns_1
NWP_test_df = NWP_test_df.set_index('Index')

NWP_x_train = NWP_train_df.loc[:,NWP_train_df.columns != 'Production'].values
NWP_y_train = NWP_train_df['Production'].values
NWP_x_test_df = NWP_test_df.loc[:,NWP_test_df.columns != 'Production']
NWP_x_test = np.array(NWP_x_test_df)
NWP_y_test = NWP_test_df['Production'].values
NWP_y_test_df = NWP_test_df['Production']

print("NWP_x_train shape: {}".format(NWP_x_train.shape))
print("NWP_x_test shape: {}".format(NWP_x_test.shape))
print("NWP_y_train shape: {}".format(NWP_y_train.shape))
print("NWP_y_test shape: {}".format(NWP_y_test.shape))

scaler = pre.StandardScaler()
NWP_X_train_scaled = scaler.fit_transform(NWP_x_train)
NWP_X_test_scaled = scaler.transform(NWP_x_test)
NWP_x_test_1 = NWP_test_1.loc[:,NWP_test_1.columns != 'Production'].values
NWP_y_test_1 = NWP_test_1['Production'].values
NWP_X_test_scaled_1 = scaler.transform(NWP_x_test_1)
scaler_y = pre.StandardScaler()
NWP_y_train_scaled = scaler_y.fit_transform(NWP_y_train.reshape(-1,1))

NWP_pca = PCA(.99)
NWP_X_train_scaled = NWP_pca.fit_transform(NWP_X_train_scaled)
NWP_X_test_scaled = NWP_pca.transform(NWP_X_test_scaled)
NWP_X_test_scaled_1 = NWP_pca.transform(NWP_X_test_scaled_1)
NWP_pca.explained_variance_ratio_.cumsum()
NWP_N_inputs = NWP_pca.n_components_

# NWP Grid search
def make_ANN_model(optimizer, initializer, batch_size):
# neurons, activations, hidden_layers
# optimizer, initializer, batch_size
# Making a linear pipeline (a stack) of neural networks layers.
    ANN_model = Sequential()
    # initializer = tf.keras.initializers.HeUniform()
# A single layer with x artificial neurons, and it expects y input variables
    ANN_model.add(Dense(14, input_dim = N_inputs,
                        kernel_initializer = initializer,
                        activation = 'selu'))
# Defining the Second hidden layer of the model
```

```
    ANN_model.add(Dense(14, kernel_initializer = initializer,
                        activation = 'selu'))
    # for i in range(hidden_layers):
    #     # Add one hidden layer
    #     ANN_model.add(Dense(neurons, kernel_initializer = initializer,
    #                         activation = activations))
# The output neuron is a single fully connected node since I will be
# predicting a single number (Power production)
    ANN_model.add(Dense(1, kernel_initializer = initializer,
                        activation = 'linear'))
# Compiling the model
    ANN_model.compile(loss = 'mean_squared_error', optimizer = optimizer)

    return ANN_model

# Listing all the parameters to try
Parameters = {
    # 'neurons': [6,8,10,12,14],
    # 'activations': ['relu', 'selu', 'elu'],
    # 'hidden_layers':[1,2,3],
    'batch_size': [32,64,128,256], 'optimizer': ['adam', 'rmsprop', 'sdg'],
    'initializer': ['lecun_normal', 'lecun_uniform']
              }


# Creating the ANN model
ANN_Model = KR(make_ANN_model, verbose=0)

# Defining a custom function to calculate accuracy
def Accuracy_Score(true,pred):
    MAPE = np.mean(100 * (np.abs(true-pred)/true))
    print('#'*70,'Accuracy:', 100-MAPE)
    return(100-MAPE)

custom_Scoring  = make_scorer(Accuracy_Score, greater_is_better=True)
# Creating the Grid search space
grid_search = GridSearchCV(estimator = ANN_Model,
                          param_grid = Parameters,
                          scoring = custom_Scoring,
                          cv = 5)

# Measuring how much time it took to find the best params
StartTime = time.time()

# Running Grid Search for different paramenters
grid_search.fit(NWP_X_train_scaled,NWP_y_train_scaled, verbose=1, epochs = 100)
```

```
EndTime=time.time()
print("########## Total Time Taken: ", round((EndTime-StartTime)/60),
      'Minutes')

print('### Printing Best parameters ###')
grid_search.best_params_

ANN_model = Sequential()
ANN_model.add(Dense(14, input_dim = NWP_N_inputs,
                    kernel_initializer='lecun_uniform', activation='selu'))

ANN_model.add(Dense(14, kernel_initializer='lecun_uniform',
                    activation='selu'))

ANN_model.add(Dense(1, kernel_initializer='lecun_uniform',
                    activation='linear'))

# Compiling the model
ANN_model.compile(loss='mean_squared_error', optimizer='rmsprop')

# Early stopping
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=10)

# Fitting the model
ANN_model.fit(NWP_X_train_scaled, NWP_y_train_scaled, epochs = 100,
              batch_size = 32, validation_split=0.2,
              callbacks=[callback], verbose=1)

NWP_Pred_test = ANN_model.predict(NWP_X_test_scaled)
NWP_Pred_test = scaler_y.inverse_transform(NWP_Pred_test)
NWP_Pred_test.min()
NWP_Pred_test[NWP_Pred_test < 0] = 0

NWP_Pred_test1 = ANN_model.predict(NWP_X_test_scaled_1)
NWP_Pred_test1 = scaler_y.inverse_transform(NWP_Pred_test1)
NWP_Pred_test1.min()
NWP_Pred_test1[NWP_Pred_test1 < 0] = 0

NWP_Pred_train = ANN_model.predict(NWP_X_train_scaled)
NWP_Pred_train = scaler_y.inverse_transform(NWP_Pred_train)
NWP_Pred_train.min()
NWP_Pred_train[NWP_Pred_train < 0] = 0

print('     ')
print('NWP ANN:')
NWP_ANN_performance= performance(NWP_y_test,NWP_Pred_test,'ANN')
NWP_ANN_performance_train= performance(NWP_y_train,NWP_Pred_train,'ANN')
```

```
print('overfitting',float(NWP_ANN_performance_train[1,1])/float(
    NWP_ANN_performance[1,1]))
print("NWP ANN performance test:")
print(NWP_ANN_performance)

print('     ')
NWP_ANN_performance_1= performance(NWP_y_test_1,NWP_Pred_test1,'ANN')
NWP_ANN_performance_train= performance(NWP_y_train,NWP_Pred_train,'ANN')
print('overfitting',float(NWP_ANN_performance_train[1,1])/float(
    NWP_ANN_performance_1[1,1]))
print("NWP ANN performance august:")
print(NWP_ANN_performance_1)

NWP_y_test_1_df = NWP_test_1['Production']
fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(NWP_y_test_1_df.index,NWP_y_test_1,color='k',linewidth=2)
plt.plot(NWP_y_test_1_df.index,NWP_Pred_test1,linewidth=2)
plt.ylim(0)
plt.xlim(NWP_y_test_1_df.index[0],NWP_y_test_1_df.index[743])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Time [h]', fontsize=16)
plt.legend(['Actual','Predicted'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(
    outpath,"Predicted- and Actual Power Production NWP ANN"))

fig = plt.figure(figsize=[10,5])
ax = fig.add_subplot(111)
plt.plot(NWP_y_test_1_df.index,NWP_y_test_1,color='k',linewidth=2)
plt.plot(NWP_y_test_1_df.index,NWP_Pred_test1,linewidth=2)
plt.ylim(0)
plt.xlim(NWP_y_test_1_df.index[120],NWP_y_test_1_df.index[360])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Production [kWh]', fontsize=16)
plt.xlabel('Time [h]', fontsize=16)
plt.legend(['Actual','Predicted'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(
    outpath,"Predicted- and Actual Power Production NWP ANN, zoomed"))

def create_error(true, pred):
    all_errors = []
    for i in range(len(pred)):
        error= true[i]-pred[i]
        all_errors.append(error)
    return all_errors

NWP_errors_ANN = create_error(NWP_y_test,NWP_Pred_test)
NWP_errors_ANN_1 = create_error(NWP_y_test_1,NWP_Pred_test1)

fig = plt.figure(figsize=[8, 4])
ax = fig.add_subplot(111)
plt.plot(NWP_y_test_df.index, NWP_errors_ANN,color='red',marker='.',
            linewidth=0,markersize=8,alpha=.4)
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Error (Actual-Predicted) [kWh]', fontsize=14)
plt.xlabel('Time [h]', fontsize=14)
plt.legend(['Error'], loc='best', fontsize=16)
plt.tight_layout()
plt.savefig(path.join(outpath,"Error random test NWP ANN"))
plt.show()

fig = plt.figure(figsize=[8, 4])
ax = fig.add_subplot(111)
plt.plot(NWP_y_test_1_df.index, NWP_errors_ANN_1,color='red',marker='.',
            linewidth=0,markersize=8,alpha=.4)
plt.xlim(NWP_y_test_1_df.index[0],NWP_y_test_1_df.index[743])
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.ylabel('Error (Actual-Predicted) [kWh]', fontsize=14)
plt.xlabel('Time [h]', fontsize=14)
plt.legend(['Error'], loc='best', fontsize=16)
ax.xaxis.set_major_formatter(dates.DateFormatter('%b %d'))
plt.tight_layout()
plt.savefig(path.join(outpath,"Error august test NWP ANN"))
plt.show()
```

A.2.15 Performances

```
#%% Performances
print('     ')
print('ANN:')
ANN_performance= performance(y_test,Pred_test,'ANN')
ANN_performance_train= performance(y_train,Pred_train,'ANN')
print('overfitting',float(ANN_performance_train[1,1])/float(
    ANN_performance[1,1]))
print("ANN performance test:")
print(ANN_performance)
print('     ')
ANN_performance_1= performance(y_test_1,Pred_test1,'ANN')
ANN_performance_train= performance(y_train,Pred_train,'ANN')
print('overfitting',float(ANN_performance_train[1,1])/float(
    ANN_performance_1[1,1]))
print("ANN performance august:")
print(ANN_performance_1)

print('     ')
print('ANN QR:')
QR_performance= performance(y_test,preds['50th'].values,'QR')
QR_performance_train= performance(y_train,Preds_train['50th'].values,'QR')
print('overfitting',float(QR_performance_train[1,1])/float(
    QR_performance[1,1]))
print("QR performance test:")
print(QR_performance)

print('     ')
QR_performance_1= performance(y_test_1,pred['50th'].values,'QR')
QR_performance_train= performance(y_train,Preds_train['50th'].values,'QR')
print('overfitting',float(QR_performance_train[1,1])/float(
    QR_performance_1[1,1]))
print("QR performance august:")
print(QR_performance_1)

print('     ')
print('SVR:')
SVR_performance= performance(y_test,predict_y_array,'SVR')
SVR_performance_train= performance(y_train,predict_y_train_array,'SVR')
print('overfitting',float(SVR_performance_train[1,1])/float(
    SVR_performance[1,1]))
print("SVR performance test:")
print(SVR_performance)

print('     ')
SVR_performance_1= performance(y_test_1,predict_y_array_1,'SVR')
SVR_performance_train= performance(y_train,predict_y_train_array,'SVR')
print('overfitting',float(SVR_performance_train[1,1])/float(
    SVR_performance_1[1,1]))
print("SVR performance august:")
print(SVR_performance_1)

print('     ')
ARIMA_performance = performance(test, predictions,'ARIMA')
print("ARIMA performance:")
print(ARIMA_performance)
print('     ')
print('Ensemble:')
Ensemble_performance= performance(y_test,ens_preds,'ANN')
Ensemble_performance_train= performance(y_train,preds_train,'ANN')
print('overfitting',float(Ensemble_performance_train[1,1])/float(
    Ensemble_performance[1,1]))
print("Ensemble performance test:")
print(Ensemble_performance)

print('     ')
Ensemble_performance_1= performance(y_test_1,preds_1,'ANN')
Ensemble_performance_train= performance(y_train,preds_train,'ANN')
print('overfitting',float(Ensemble_performance_train[1,1])/float(
    Ensemble_performance_1[1,1]))
print("Ensemble performance august:")
print(Ensemble_performance_1)

print('     ')
print('Ensemble:')
ANN_Ensemble_performance= performance(y_test,ann_preds,'ANN')
ANN_Ensemble_performance_train= performance(y_train,ann_preds_train,'ANN')
print('overfitting',float(ANN_Ensemble_performance_train[1,1])/float(
    ANN_Ensemble_performance[1,1]))
print("Ensemble performance test:")
print(ANN_Ensemble_performance)

print('     ')
ANN_Ensemble_performance_1= performance(y_test_1,ann_preds_1,'ANN')
ANN_Ensemble_performance_train= performance(y_train,ann_preds_train,'ANN')
print('overfitting',float(ANN_Ensemble_performance_train[1,1])/float(
    ANN_Ensemble_performance_1[1,1]))
print("Ensemble performance august:")
print(ANN_Ensemble_performance_1)

print('     ')
print('NWP ANN:')
NWP_ANN_performance= performance(NWP_y_test,NWP_Pred_test,'ANN')
NWP_ANN_performance_train= performance(NWP_y_train,NWP_Pred_train,'ANN')
print('overfitting',float(NWP_ANN_performance_train[1,1])/float(
    NWP_ANN_performance[1,1]))
print("NWP ANN performance test:")
print(NWP_ANN_performance)

print('     ')
NWP_ANN_performance_1= performance(NWP_y_test_1,NWP_Pred_test1,'ANN')
NWP_ANN_performance_train= performance(NWP_y_train,NWP_Pred_train,'ANN')
```

```
print('overfitting',float(NWP_ANN_performance_train[1,1])/float(
    NWP_ANN_performance_1[1,1]))
print("NWP ANN performance august:")
print(NWP_ANN_performance_1)
```

A.2.16 Performance tables in report

```
#%% Performance tables in report
SVR_performance = pd.DataFrame(SVR_performance)
ANN_performance = pd.DataFrame(ANN_performance)
Ensemble_performance = pd.DataFrame(Ensemble_performance)
ANN_Ensemble_performance = pd.DataFrame(ANN_Ensemble_performance)
NWP_ANN_performance = pd.DataFrame(NWP_ANN_performance)
QR_performance = pd.DataFrame(QR_performance)
Performance = [SVR_performance, ANN_performance, Ensemble_performance,
                ANN_Ensemble_performance, NWP_ANN_performance, QR_performance]
Performance = pd.concat(Performance, axis = 1)
Performance = Performance.iloc[1:6,[0,1,3,5,7,9,11]]
Performance.columns = [' ','SVR','ANN', 'Ensemble SVR ANN', 'Ensemble ANN',
                        'ANN NWP', 'QR']
Performance = Performance.set_index(' ')

SVR_performance_1 = pd.DataFrame(SVR_performance_1)
ANN_performance_1 = pd.DataFrame(ANN_performance_1)
Ensemble_performance_1 = pd.DataFrame(Ensemble_performance_1)
ANN_Ensemble_performance_1 = pd.DataFrame(ANN_Ensemble_performance_1)
NWP_ANN_performance_1 = pd.DataFrame(NWP_ANN_performance_1)
QR_performance_1 = pd.DataFrame(QR_performance_1)
ARIMA_performance = pd.DataFrame(ARIMA_performance)
Performance_1 = [SVR_performance_1, ANN_performance_1,
                  Ensemble_performance_1, ANN_Ensemble_performance_1,
                  NWP_ANN_performance_1, QR_performance_1, ARIMA_performance]
Performance_1 = pd.concat(Performance_1, axis = 1)
Performance_1 = Performance_1.iloc[1:6,[0,1,3,5,7,9,11,13]]
Performance_1.columns = [' ','SVR','ANN', 'Ensemble SVR ANN',
                          'Ensemble ANN', 'ANN NWP', 'QR','ARIMA']
Performance_1 = Performance_1.set_index(' ')

SVR_performance_train = pd.DataFrame(SVR_performance_train)
ANN_performance_train = pd.DataFrame(ANN_performance_train)
Ensemble_performance_train = pd.DataFrame(Ensemble_performance_train)
ANN_Ensemble_performance_train = pd.DataFrame(ANN_Ensemble_performance_train)
NWP_ANN_performance_train = pd.DataFrame(NWP_ANN_performance_train)
QR_performance_train = pd.DataFrame(QR_performance_train)
Performance_train = [SVR_performance_train, ANN_performance_train,
                      Ensemble_performance_train,
                      ANN_Ensemble_performance_train,
                      NWP_ANN_performance_train, QR_performance_train]
```

```
Performance_train = pd.concat(Performance_train, axis = 1)
Performance_train = Performance_train.iloc[1:6,[0,1,3,5,7,9,11]]
Performance_train.columns = [' ','SVR','ANN', 'Ensemble SVR ANN',
                              'Ensemble ANN', 'ANN NWP', 'QR']
Performance_train = Performance_train.set_index(' ')

Performances = [Performance, Performance_1, Performance_train]
Performances = pd.concat(Performances, axis = 1)

Performances.to_csv('Performances.csv')
Performance.to_csv('Performance random.csv')
Performance_1.to_csv('Performance august.csv')
Performance_train.to_csv('Performance train.csv')

all_errors_SVR_df = pd.DataFrame(all_errors_SVR)
all_errors_SVR_df.max()
all_errors_SVR_df.min() #largest error

all_errors_SVR_1_df = pd.DataFrame(all_errors_SVR_1)
all_errors_SVR_1_df.max() #largest error
all_errors_SVR_1_df.min()

all_errors_ANN_df = pd.DataFrame(all_errors_ANN)
all_errors_ANN_df.max()
all_errors_ANN_df.min() #largest error for my fit

all_errors_ANN_1_df = pd.DataFrame(all_errors_ANN_1)
all_errors_ANN_1_df.max() #largest error for my fit
all_errors_ANN_1_df.min()

all_errors_ARIMA_df = pd.DataFrame(all_errors_ARIMA)
all_errors_ARIMA_df.max()
all_errors_ARIMA_df.min() #largest error

errors_ens_SVRANN_df = pd.DataFrame(errors_ens_SVRANN)
errors_ens_SVRANN_df.max()
errors_ens_SVRANN_df.min() #largest error for my fit

errors_ens_SVRANN_1_df = pd.DataFrame(errors_ens_SVRANN_1)
errors_ens_SVRANN_1_df.max() #largest error for my fit
errors_ens_SVRANN_1_df.min()

errors_ens_ANN_df = pd.DataFrame(errors_ens_ANN)
errors_ens_ANN_df.max()
errors_ens_ANN_df.min() #largest error for my fit

errors_ens_ANN_1_df = pd.DataFrame(errors_ens_ANN_1)
```

```
errors_ens_ANN_1_df.max() #largest error for my fit
errors_ens_ANN_1_df.min()

NWP_errors_ANN_df = pd.DataFrame(NWP_errors_ANN)
NWP_errors_ANN_df.max() #largest error for my fit
NWP_errors_ANN_df.min()

NWP_errors_ANN_1_df = pd.DataFrame(NWP_errors_ANN_1)
NWP_errors_ANN_1_df.max()
NWP_errors_ANN_1_df.min() #largest error for my fit
```

A.3 Other python plots and tables

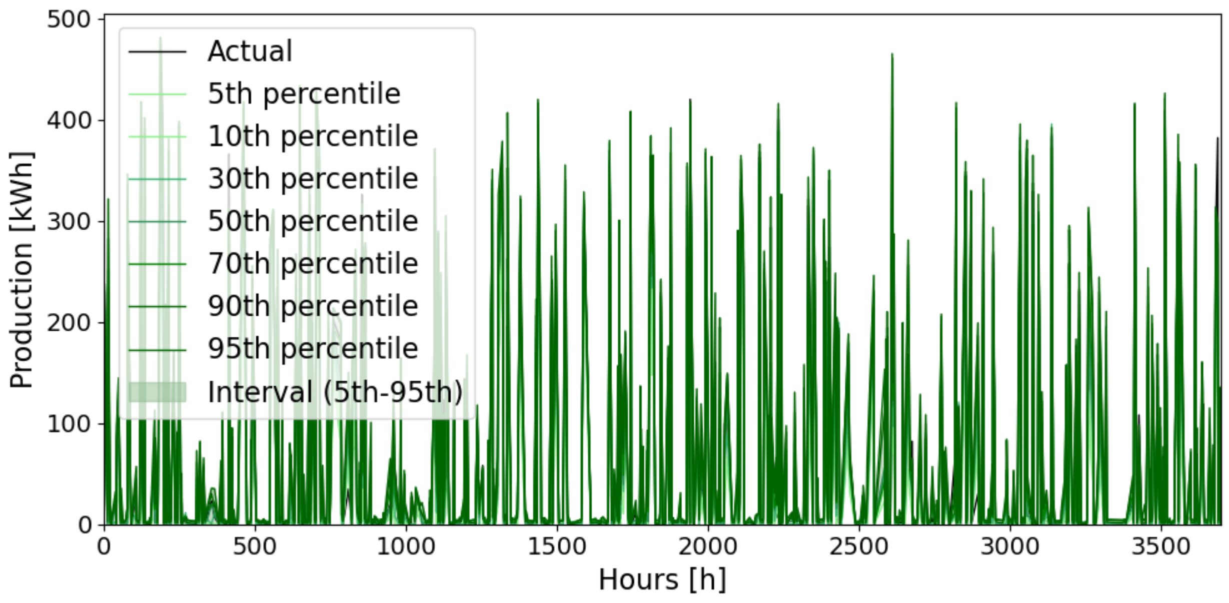


Figure A.1: The predicted power production by the QRNN model on random test data and the actual production.

Table A.1: Performance August Test 30 fits. The RMSE, nRMSE, MAE and R^2 of all 30 fits of the ANN model in August, 2021.

	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	34.946	34.842	34.183	35.383	36.185	34.987
nRMSE[%]	0.057	0.057	0.056	0.058	0.059	0.057
MAE[kWh]	18.184	18.101	17.068	18.945	18.700	17.955
$\frac{\text{MAE}}{\text{Wp}}$[%]	0.030	0.030	0.028	0.031	0.031	0.029
R^2	0.911	0.912	0.915	0.909	0.905	0.911
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	34.969	35.586	34.780	34.528	35.510	34.997
nRMSE[%]	0.057	0.058	0.057	0.056	0.058	0.057
MAE[kWh]	17.880	18.005	17.840	17.788	17.714	17.374
$\frac{\text{MAE}}{\text{Wp}}$[%]	0.029	0.029	0.029	0.029	0.029	0.028
R^2	0.911	0.908	0.912	0.913	0.908	0.911
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	36.059	36.789	35.185	35.747	34.731	34.624
nRMSE[%]	0.059	0.060	0.057	0.058	0.057	0.056
MAE[kWh]	20.311	21.544	18.89	18.286	18.406	17.737
$\frac{\text{MAE}}{\text{Wp}}$[%]	0.033	0.035	0.031	0.030	0.030	0.029
R^2	0.905	0.901	0.910	0.907	0.912	0.913
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	36.673	37.034	36.779	35.341	36.930	35.216
nRMSE[%]	0.060	0.060	0.060	0.058	0.060	0.057
MAE[kWh]	22.082	21.608	20.389	17.880	20.596	17.750
$\frac{\text{MAE}}{\text{Wp}}$[%]	0.036	0.035	0.033	0.029	0.034	0.029
R^2	0.902	0.900	0.901	0.909	0.901	0.910
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	36.155	34.932	35.133	34.689	34.895	36.582
nRMSE[%]	0.059	0.057	0.057	0.057	0.057	0.060
MAE[kWh]	18.360	17.459	18.335	17.846	17.880	21.361
$\frac{\text{MAE}}{\text{Wp}}$[%]	0.030	0.028	0.030	0.029	0.029	0.035
R^2	0.905	0.911	0.91	0.912	0.911	0.903

Table A.2: Performance Random Test 30 fits. The RMSE, nRMSE, MAE and R^2 of all 30 fits of the ANN model.

	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	21.738	21.879	22.174	21.931	24.045	22.068
nRMSE[%]	0.035	0.036	0.036	0.036	0.039	0.036
MAE[kWh]	9.240	9.525	9.350	10.107	10.612	9.909
$\frac{\text{MAE}}{\text{Wp}}$[%]	0.015	0.016	0.015	0.016	0.017	0.016
R^2	0.952	0.951	0.950	0.951	0.941	0.950
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	21.854	21.927	22.819	22.349	22.385	22.639
nRMSE[%]	0.036	0.036	0.037	0.036	0.037	0.037
MAE[kWh]	9.082	9.919	9.765	9.490	9.208	9.246
$\frac{\text{MAE}}{\text{Wp}}$[%]	0.015	0.016	0.016	0.015	0.015	0.015
R^2	0.951	0.951	0.947	0.949	0.949	0.947
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	22.716	23.721	22.851	23.291	21.688	22.318
nRMSE[%]	0.037	0.039	0.037	0.038	0.035	0.036
MAE[kWh]	11.622	13.834	11.797	9.870	9.634	9.321
$\frac{\text{MAE}}{\text{Wp}}$[%]	0.019	0.023	0.019	0.016	0.016	0.015
R^2	0.947	0.942	0.947	0.944	0.952	0.949
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	23.954	23.588	22.761	22.02	23.535	22.743
nRMSE[%]	0.039	0.038	0.037	0.036	0.038	0.037
MAE[kWh]	14.277	12.947	11.172	9.464	10.589	9.695
$\frac{\text{MAE}}{\text{Wp}}$[%]	0.023	0.021	0.018	0.015	0.017	0.016
R^2	0.941	0.943	0.947	0.950	0.943	0.947
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	22.752	21.485	21.802	21.852	22.438	22.983
nRMSE[%]	0.037	0.035	0.036	0.036	0.037	0.037
MAE[kWh]	9.603	9.058	10.126	9.707	9.572	12.550
$\frac{\text{MAE}}{\text{Wp}}$[%]	0.016	0.015	0.017	0.016	0.016	0.020
R^2	0.947	0.953	0.951	0.951	0.948	0.946

Table A.3: Performance Random Train 30 fits

	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	22.904	23.130	22.956	22.927	24.847	22.633
nRMSE[%]	0.037	0.038	0.037	0.037	0.041	0.037
MAE[kWh]	9.569	9.838	9.485	10.301	10.603	10.221
$\frac{\mathrm{MAE}}{\mathrm{Wp}}$[%]	0.016	0.016	0.015	0.017	0.017	0.017
R^2	0.947	0.946	0.947	0.947	0.938	0.948
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	22.553	22.560	23.290	22.505	22.841	22.761
nRMSE[%]	0.037	0.037	0.038	0.037	0.037	0.037
MAE[kWh]	9.232	10.062	9.748	9.442	9.327	9.206
$\frac{\mathrm{MAE}}{\mathrm{Wp}}$[%]	0.015	0.016	0.016	0.015	0.015	0.015
R^2	0.949	0.949	0.945	0.949	0.947	0.948
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	23.404	24.654	23.272	23.861	22.678	22.737
nRMSE[%]	0.038	0.040	0.038	0.039	0.037	0.037
MAE[kWh]	11.726	14.032	11.846	9.977	9.859	9.361
$\frac{\mathrm{MAE}}{\mathrm{Wp}}$[%]	0.019	0.023	0.019	0.016	0.016	0.015
R^2	0.945	0.939	0.945	0.942	0.948	0.948
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	24.107	24.445	23.098	22.831	23.493	23.994
nRMSE[%]	0.039	0.040	0.038	0.037	0.038	0.039
MAE[kWh]	14.377	13.205	11.291	9.588	10.504	9.955
$\frac{\mathrm{MAE}}{\mathrm{Wp}}$[%]	0.023	0.022	0.018	0.016	0.017	0.016
R^2	0.941	0.940	0.946	0.947	0.944	0.942
	ANN	ANN	ANN	ANN	ANN	ANN
RMSE[kWh]	22.904	22.689	22.443	22.529	23.079	23.864
nRMSE[%]	0.037	0.037	0.037	0.037	0.038	0.039
MAE[kWh]	9.508	9.359	10.280	9.800	9.590	12.721
$\frac{\mathrm{MAE}}{\mathrm{Wp}}$[%]	0.016	0.015	0.017	0.016	0.016	0.021
R^2	0.947	0.948	0.949	0.949	0.946	0.942