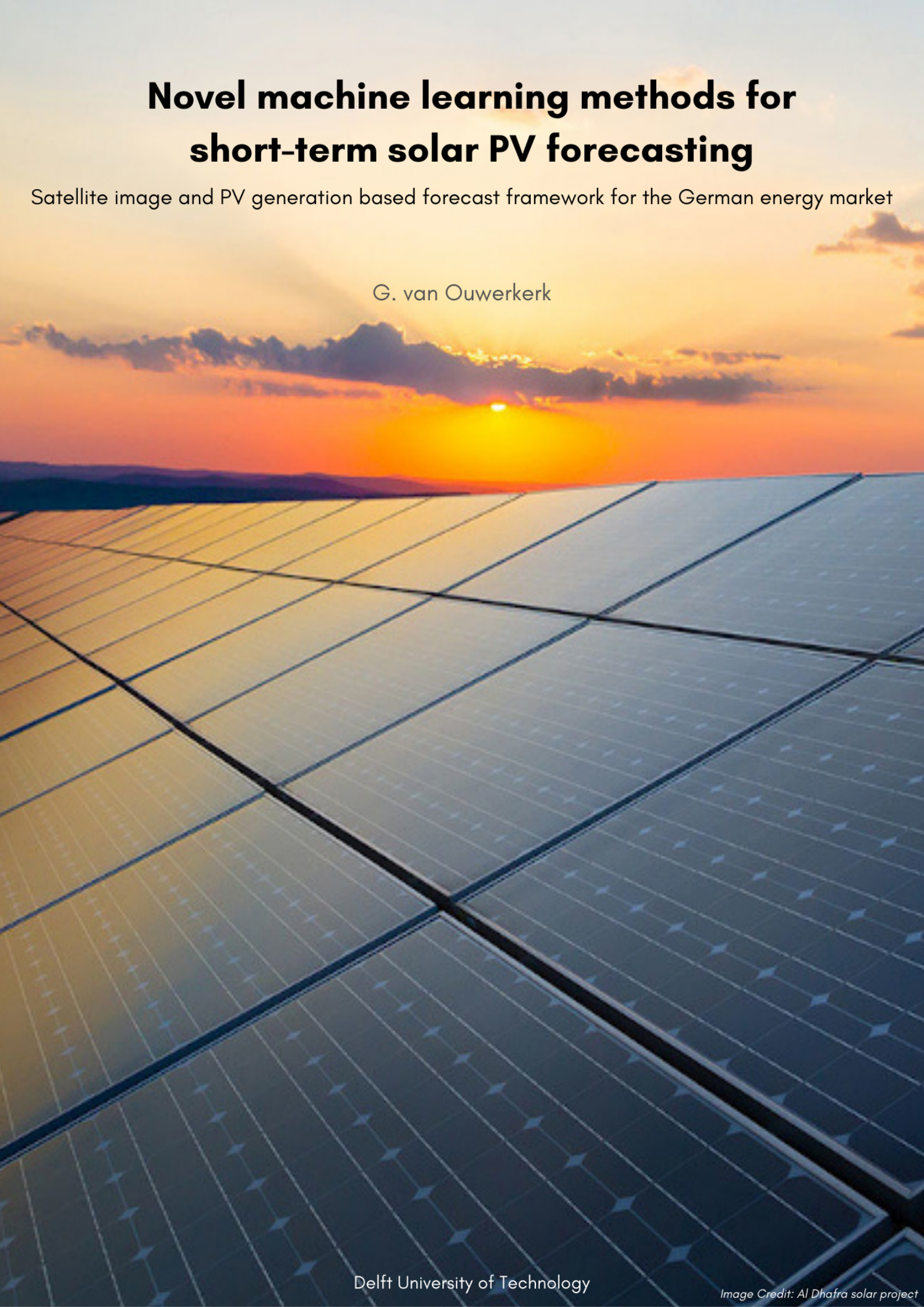
Novel machine learning methods for short-term solar PV forecasting

Satellite image and PV generation based forecast framework for the German energy market

G. van Ouwerkerk



Delft University of Technology

Image Credit: Al Dhafra solar project

Novel machine learning methods for short-term solar PV forecasting

Satellite image and PV generation based forecast framework for the German energy market

by

Gijs van Ouwerkerk

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday June 17, 2021 at 13:00 AM.

Student number:	4474767	
Project duration:	October 5, 2020 – June 17, 2021	
Thesis committee:	Dr. S. Basu,	TU Delft, supervisor
	Dr. ir. R. A. Verzijlbergh,	TU Delft
	Dr. H. Ziar,	TU Delft

This thesis is confidential and cannot be made public until June 1, 2023.

TU Delft

Preface

This report is written as part of my master's degree in Sustainable Energy Technology at Delft University of Technology. I carried out this research project in collaboration with Northpool, a company active on the European energy market. This report describes the work done on the development of a novel machine learning based solar PV power forecasting framework. Increasingly more accurate solar PV power forecast support the continues large scale adoption of solar PV in our energy mix and I sincerely hope the results of this project supports the steps to be taken in that direction.

At the start of this project I was already extremely interested in renewable energy and learning about it for several years, however my knowledge on machine learning was close to none. In the course of this project I learned a lot about, and especially experimented a lot with, machine learning. For me this has been the most valuable part of this research project, as I have been inspired by the vast possibilities of artificial intelligence. I am certain that following this project I will keep on expanding my knowledge on machine learning and find new and interesting applications for it in relation to renewable energy technologies. I hope that while reading this report you will get as inspired about solar PV power, energy markets and artificial intelligence as I am now.

I would like to thank my TU Delft supervisors Sukanta Basu and Remco Verzijlbergh for their support, advice and discussions throughout my thesis project. I also would like to thank Hesan Ziar from the PVDM group for his support in evaluating the results presented in this report. Moreover, I am very grateful for the opportunities that Northpool provided to me throughout my masters, with many interesting projects related to the energy market and now also solar PV power forecasting. I am certain this sparked an everlasting personal interest in the relation between renewable energy and their impact on the energy markets. Finally, I especially want to thank Stefan de Weger for his supervision, ideas and interesting discussions throughout my thesis project.

G. van Ouwerkerk
The Hague, May 2021

Summary

With the growing global drive to act up on climate change, the adoption of renewable energy sources such as solar photovoltaic (PV) and wind is continuously increasing. This crucial shift poses many economic and environmental benefits, however the variability in solar PV generation may also threaten the stability of our grid and energy supply. The reliable prediction of this fluctuating power resource on various time scales has been identified as a crucial technology for the continuous massive adoption of solar PV and the next-generation energy system. Moreover, a knowledge gab has been identified with respect to the application of satellite images for solar PV power forecast in a straightforward, yet accurate way. To this end, the research objective of this project was to investigate and leverage on the recent developments in the field of deep learning, with respect to convolutional neural networks (CNN) and Long Short Term Memory (LSTM), in order to propose new models for deterministic intra-day solar PV power forecasting.

The study focuses on direct solar PV power prediction models with a forecast horizon of 3 hours and an interval of 15-minutes, to be applied in Germany. Additionally, the models are designed to be applicable in a real-time operational setting with a short forecast lag, which required for energy market trading purposes. The methodology followed consists of the development of two individual forecast models: (1) A LSTM network that leverages on the latest solar PV generation data and a NWP based day-ahead power forecast, and (2) a CNN-LSTM network designed to utilize the latest satellite images and a NWP based day-ahead power forecast. The difference in input feature selection results in a forecast lag of maximum 60-minutes for model 1 and only 5-minutes for model 2.

The accuracy of the two proposed forecast models are evaluated using one year of solar PV power generation data in Germany (January 2020 through December 2020), and are compared to a persistence model and a NWP based day-ahead and intra-day power forecast provided by the German transmission system operators (TSOs). The results show that the two proposed models perform equal or better than the benchmark models, obtaining a Mean Absolute Error (MAE) of 316 MW for model 1 and 547 MW for model 2, compared to 1333 MW, 672 MW and 588 MW for the benchmark persistence, day-ahead and intra-day model, respectively. This showcases the potential of deep learning models for short-term solar PV power forecasting, in the first place based on the latest actual solar PV generation and alternatively based on satellite images. In particular, this study shows the potential of satellite images in being a valuable proxy for the latest actual solar PV generation and subsequent power predictions; especially when there is no real-time actual generation data available. It is also found that the two deep learning models outperform the benchmark model accuracy most significantly around the peak of solar generation and less during the daily ramp up of solar PV power output. This result stresses the weakness of statistical models in its dependence on complex input-output mapping, and hence its reliance on representative past input data. Moreover, it is found that the statistical deep learning models outperform the NWP based intra-day benchmark forecast up to a forecast horizon of 2-3 hours. Finally, an extensive analysis, including several case studies, show that both proposed models experience prediction accuracy dependency on weather conditions. The lowest accuracy occurs under conditions featuring strong spatial and temporal variation in the cloud cover, such as with broken cloud cover or under convective cloud formation. It is also shown that this effect is stronger, resulting in a lower overall prediction accuracy, for a regional forecast compared to a country-aggregated forecast. This outcome is explained by the averaging effect of local inaccuracies in aggregated forecasts.

In light of the encouraging results observed in this research project, it is concluded that the proposed deep learning models 1 and 2 are a powerful tool for short-term solar PV power forecasting. Although, taking into considerations its limitations under certain weather conditions, input feature dependence, forecast delay, forecast horizon and the black-box characteristic of deep learning models; it seems apparent that deep learning based solar PV power forecasting models are rather an addition to a comprehensive forecast toolbox than a replacement. Future research recommendations to further improve the proposed forecast models or to pursue different strategies include: the investigation of convLSTM architectures, the use of different satellite derived data products, the further tuning of model hyperparameters and input data pre-processing, and the development of statistical-physical hybrid models using the CNN-LSTM architecture of model 2.

Contents

Preface ii

Summary iv

List of Figures ix

List of Tables x

List of Abbreviations xi

1 Introduction 1

2 Solar PV power forecasting techniques 3
2.1 Forecast techniques . . . 3
2.1.1 Physical models . . . 3
2.1.2 Statistical models . . . 4
2.2 Sky imagery and satellite based models . . . 6
2.3 Spatial and temporal aspects of forecast . . . 7
2.4 Forecasts and energy markets . . . 8

3 Machine learning techniques 10
3.1 Artificial Neural Networks (ANN) . . . 10
3.1.1 Training algorithm . . . 12
3.1.2 Hyperparameters . . . 13
3.1.3 Generalization . . . 15
3.2 Convolutional Neural Networks (CNN) . . . 15
3.3 Long-Short Term Memory Neural Networks (LSTM) . . . 17

4 Methodology 21
4.1 Dataset description . . . 21
4.1.1 Actual solar PV power feed-in data . . . 21
4.1.2 TSO solar PV power feed-in prediction data . . . 23
4.1.3 Satellite image data . . . 23
4.1.4 Day-of-year and Quarter-of-day . . . 25
4.1.5 Data split . . . 25
4.2 Clear Sky Model . . . 26
4.2.1 Clear sky model methodology . . . 26
4.2.2 Regional clear sky power model . . . 27
4.3 Model framework . . . 28
4.3.1 Model 1 - LSTM . . . 29
4.3.2 Model 2 - CNN-LSTM . . . 31
4.4 Forecast performance evaluation metrics . . . 35
4.5 Experimental setup . . . 37

5 Results & Discussion 38
5.1 Global statistics . . . 38
5.1.1 Overall performance . . . 38
5.1.2 Performance over the solar ramp . . . 38
5.1.3 Performance over the forecast horizon . . . 39
5.1.4 Forecast bias . . . 41
5.1.5 Performance dependency on weather conditions . . . 42
5.1.6 Seasonal effects . . . 43
5.2 Case studies . . . 43
5.3 Regional forecast . . . 49

6 Conclusion & recommendations 55

Appendices **62**

A Transfer learning **63**

B Model 1 script **66**

C Model 2 script **72**

D Model 2A script **79**

E Data generator script **84**

List of Figures

1.1 EU27 cumulative installed solar PV capacity, scenarios for 2021-2024. 1

2.1 Illustration of a general physical solar PV power forecasting scheme. 4
2.2 Broad classification of various forecasting methods based on the spatio-temporal horizon of its application. The highlighted field of interest is discussed in section 2.4 and related to short-term solar PV power forecasts. 8

3.1 Architecture of a two-layered MLP used for a regression task. 11
3.2 Simplified schematic representation of model training process for a one perceptron network. . . . 12
3.3 Illustration of a general CNN architecture with a fully connected layer on top. Overall, this network contains two convolutional layers and two pooling layers, with attached a flattening layer and three fully connected layers. 16
3.4 An illustration of the operations in a convolutional layer. Here the red block represents the sliding of the filter over the input array resulting in the step-by-step generation of the feature map. . . . 16
3.5 An illustration of the operations in a pooling layer. Here the red block represents the sliding of the pooling kernel over the input array resulting in the step-by-step generation of the down-sampled feature map. 17
3.6 A single localized LSTM unit cell in the first layer for time step t, inspired by. The red and green arrows depict the information flows of the input data (x_t) and (h_{t-1}) respectively. The symbols $\odot$ and $\oplus$ represent element-wise multiplication and summation operations respectively. 19
3.7 Schematic representation of an encoder-decoder framework for sequence-to-sequence prediction models. 20

4.1 Map of TSO control regions in Germany and installed solar PV capacity as of December 2020. . 22
4.2 Actual solar PV feed-in power, maximum solar PV power and two standard solar PV forecast time series on a summer day (a) and winter day (b). 23
4.3 Two dataset samples of high resolution visible satellite images over Germany. 24
4.4 Schematic representation of the regional clear sky irradiance and maximum solar PV power forecasting scheme. 28
4.5 Schematic representation of the length and temporal dependence of the model input and output features. 29
4.6 Overview of the model 1 LSTM Encoder-Decoder architecture. The values in brackets represent the output shape of each layer, which illustrates the data-flow in one forecast run. 30
4.7 Training and validation MSE loss for model 1. 31
4.8 Overview of the model 2A, designed for the processing of satellite images to corresponding solar PV power output. 32
4.9 Training and validation MSE loss for model 2A. 33
4.10 Overview of the model 2 with the integration of the pre-trained model 2A. 34
4.11 Training and validation MSE loss for model 2. 34
4.12 Illustration of the grouping mechanism used to separate the solar ramp in three events: ramp up, peak and ramp down. 36

5.1 Comparison of the prediction accuracy of the models across the 3-hour forecast horizon, with a split for the full day, ramp up, peak and ramp down solar ramp events. 40
5.2 Scatter plot of 1 hour, 2 hour and 3 hour forecast predictions of models 1 and 2, split by solar ramp event, compared to the actual solar PV generation. 41
5.3 Dependency of the two proposed models and Entsoe intra-day and day-ahead models MAE on the normalized actual solar PV feed-in power, presented as a proxy for the clear sky index. . . . 42
5.4 Time series of the forecasts made for case study 1 with accompanying satellite images. The first forecast subfigure is presented top left with the arrows indicating the chronological order. 45
5.5 Time series of the forecasts made for case study 2 with accompanying satellite images. The first forecast subfigure is presented top left with the arrows indicating the chronological order. 47
5.6 Time series of the forecasts made for case study 3 with accompanying satellite images. The first forecast subfigure is presented top left with the arrows indicating the chronological order. 48
5.7 Comparison of the prediction accuracy of the models for the 50Hertz grid region across the 3-hour forecast horizon, with a split for the full day, ramp up, peak and ramp down solar ramp events. . 50

5.8 Time series of the forecasts made for regional 50Hertz case study 1, with accompanying satellite images over the grid control region. The first forecast subfigure is presented top left with the arrows indicating the chronological order. 52
5.9 Four forecast runs for Germany on the same day as the regional case study 1, 10 February 2020. 53
5.10 Time series of the forecasts made for regional 50Hertz case study 2, with accompanying satellite images over the grid control region. The first forecast subfigure is presented top left with the arrows indicating the chronological order. 54

A.1 Overview of three experimental CNN-LSTM architectures, with different CNN configurations, called model A, B and C. The value in angle brackets represents the kernel size of the filter, with the value in front the number of filters. 64
A.2 Training and validation RMSE loss of experimental models A, B and C. 64

List of Tables

2.1 The correlation between meteorological parameters and solar PV power output at a site in the USA. . . . 3
2.2 Overview of several statistical linear solar PV power forecasting models. . . . 5
2.3 Overview of several statistical non-linear solar PV power forecasting models. . . . 6

3.1 Overview of commonly used activation functions in literature. . . . 14

4.1 Average delay time in publication of solar PV feed-in per grid control area. . . . 22
4.2 Overview of dataset split. . . . 25

5.1 Comparison of the forecast accuracy of the benchmarks and proposed models using the performance metrics MAE, RMSE, sMAPE and tMAPE for the year 2020. The best performance is marked with bold font. . . . 38
5.2 Comparison of the proposed models and benchmark forecast accuracy split in the three characteristic solar ramp events. The best performance is marked with bold font and the RMESE, MAE and sMAPE in (MW), (MW) and (%), respectively. . . . 39
5.3 Comparison of the proposed models and benchmark forecast mean bias error in MW. The MBE score is split in the three characteristic solar ramp events, plus a further split for a 0-1 hour, 1-2 hour and 2-3 hour forecast horizon. . . . 42
5.4 Comparison of the seasonal performance of the benchmark and two proposed models for the year 2020. The best performance is marked with bold font. . . . 43
5.5 Comparison of the forecast accuracy of the benchmark and proposed models for the 50Hertz grid region using the performance metrics MAE, RMSE, sMAPE and tMAPE for the year 2020. The best performance is marked with bold font. . . . 49
5.6 Comparison of the proposed models and benchmark forecast accuracy for the 50Hertz grid region and split in the three characteristic solar ramp events. The best performance is marked with bold font and the RMESE, MAE and sMAPE in (MW), (MW) and (%), respectively. . . . 50

List of Abbreviations

AE	Auto-Encoder
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
ARIMA	Auto-Regressive Integrated Moving Average
ARMA	Auto-Regressive Moving Average
ARMAX	Auto-Regressive Moving Average eXogenous
ARX	Auto-Regressive eXogenous
CMV	Cloud Motion Vector
CNN	Convolutional Neural Network
DFFNN	Deep FeedForward Neural Network
DHI	Diffuse Horizontal Irradiance
DNI	Direct Normal Irradiance
DNN	Deep Neural Network
DOY	Day-Of-Year
DWD	Deutscher Wetterdienst
ECMWF	European Centre for Medium-Range Weather Forecasts
EU	European Union
EUMETSAT	European Organization for the Exploitation of Meteorological Satellites
EWMA	Exponential Weighted Moving Average
FFNN	FeedForward Neural Network
GFS	Global Forecast System
GHI	Global Horizontal Irradiance
GRU	Gated Recurrent Unit
IR	InfraRed
k-NN	k-Nearest Neighbors
LSTM	Long Short Term Memory
MAE	Mean Absolute Error
MBE	Mean Bias Error
MLP	Multi-Layer Perceptron
MOS	Model Output Statistics
MSE	Mean Squared Error
MSG	Meteosat Second Generation
NWP	Numerical Weather Predictions
PReLU	Parametric Rectified Linear Unit
PV	Photovoltaics
QOD	Quarter-Of-Day
ReLU	Rectified Linear Unit
RES	Renewable Energy Sources
RF	Random Forest
RGB	Red Green Blue
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
SARIMA	Seasonal Auto-Regressive Integrated Moving Average
SEVIRI	Spinning Enhanced Visible and InfraRed Imager
SGD	Stochastic Gradient Descent
SICCS	Surface Insolation under Clear and Cloudy skies derived from SEVIRI
sMAPE	solar Mean Absolute Percentage Error
SMV	Support Vector Machine
SVF	Sky View Factor
tMAPE	total Mean Absolute Percentage Error
TSO	Transmission Systems Operator

Chapter 1: Introduction

The renewable energy sector has shown remarkable strength in 2020, despite the Covid-19 pandemic negatively impacting many industries and everybody's life in numerous ways. The demand for power from renewable energy sources (RES) has increased notably in the European Union (EU) over the past year, with a leading role for solar photovoltaics (PV) and wind energy technologies [1]. Despite a slowdown of new additions in the first half of 2020, due to lock-downs and global supply chain disruptions, the installation pace accelerated over the second half driven by a push for a 'green' and sustainable economic recovery [1]. The installed solar PV capacity grew by 18.2 GW in 2020, which is an 11% improvement over 2019 and marking the second-best year ever in the EU solar history [2]. Germany alone already installed 4.8 GW in 2020, again marking its place as the largest solar market in Europe [2].

Due to the broadening acknowledgment of the benefits of solar PV, market analysis suggest that in the next 4 year the the European solar sector will see continued expansion. Figure 1.1 shows the expected growth in installed solar PV capacity in Europe for several scenarios, as projected by SolarPower Europe [2]. In a positive scenario the installed capacity could more than doubled by 2024, with 292.8 GW installed across Europe. The same scenario projects a PV capacity growth of 32.2 GW in Germany by 2024, which is a significant increase over the total installed capacity of 57.5 GW in December 2020 [2]. The numerous positive developments leading to this optimistic outlook, such as improving cost leadership and government support policies, will most likely continue to reshape Europe's energy markets.

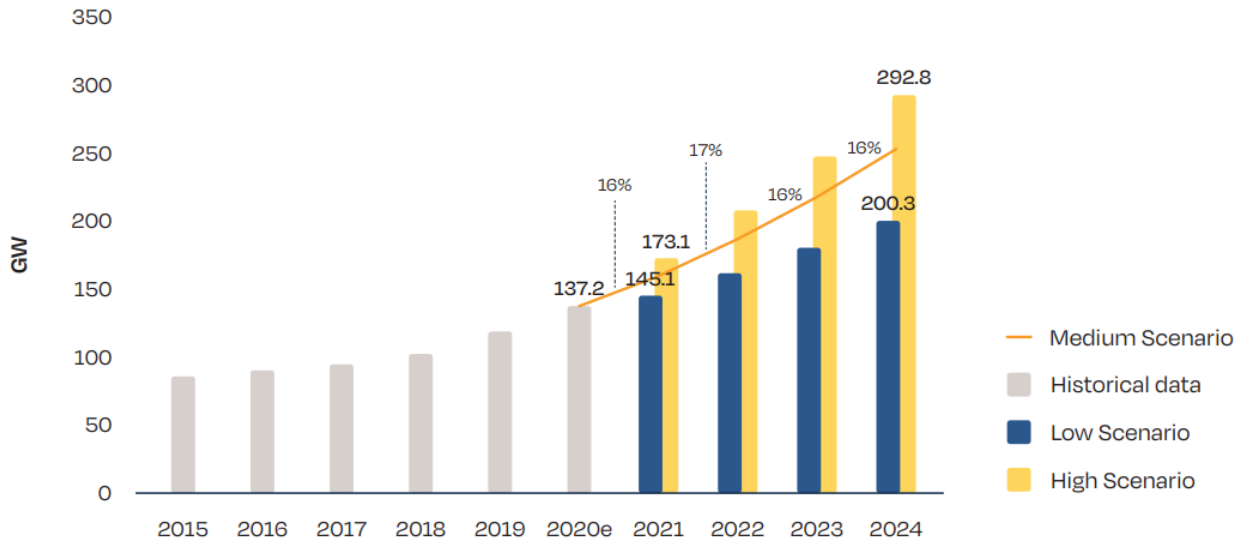


Figure 1.1: EU27 cumulative installed solar PV capacity, scenarios for 2021-2024 [2].

The continued adoption of solar PV generation poses many environmental and economic benefits, however the variability in its power output may also threaten the stability of the grid and our electricity supply [3]. A number of challenges remain for the continued efficient and reliable integration of solar PV. One of those key challenges will be the development of practices and tools that can mitigate the variability and uncertainty of solar power generation [4]. In short, these variations are related to ever-changing meteorological states and cloud formations [5]. The short-term power uncertainty can be handled with a range of solutions, such as the balancing of power between allocation areas, demand-side response, and the deployment of various energy storage solutions. Yet, one of the most economical and efficient solutions is the employment of forecasts for the expected solar PV power generation [6]. Hence, the ability to more accurately forecast the power produced by solar PV systems on various timescales has been identified as one of the main steps to overcome for continued massive solar PV adoption in the grid [6].

Solar PV power forecast are used by a variety of participants in the electricity market. Grid operators use forecast to continuously balance the energy supply and demand in real time. Asset owners and PV systems operators use the forecast to schedule generation and balance positions during the day to manages changes in earlier predicted power output. On sunny days, solar PV generation can temporarily supply up to 50% of the electricity demand in Germany, which is an amount that can strongly impact market prices [7]. As a result, also energy traders are actively making use of solar power forecast to take up positions in the energy market.

The field of solar PV power forecast involves a variety of methods, mainly based on requirements regarding the spatial domain, the time horizon, the availability of data and user application. These methods include numerical weather prediction models (NWP), satellite imagery, sky imagers, statistical learning methods and ensemble techniques which blend various aspects of different methods [4]. In general, for forecast with longer horizons the chance and magnitude of forecast errors increases. In contrast, very short-term forecasts are negatively influenced by unexpected natural events, such as convective clouds [8]. As such, a wide range of techniques and model variants have been proposed in literature depending on the forecast task, with each having its own strengths and weaknesses. Intra-day solar PV power forecast cover a temporal horizon of 1 to 6 hours, these forecasts are mainly used by participant on the intra-day electricity market and employ statistical learning methods, satellite imagery and NWP models [9].

In recent years the field of solar PV power forecasts based on statistical learning methods have seen an increased use of machine learning, a modeling method that relies on artificial intelligence (AI) to learn from experience with data in order to perform certain tasks [8]. Machine learning models can interpret highly non-linear relations and features in data, without any preordained physical equations. Hence, this minimizes the effort needed to manually engineer features in a model, such as physical relations between irradiance and solar power output, or statistical relations between past observations and future predictions. Furthermore, the development of simple yet powerful front-end programming packages have significantly improved the access to realm of deep learning models for practitioners in many fields.

As the development of precise forecast remains challenging, current research and industries are continuously investigating established methods, ensemble techniques and newly available approaches such as machine learning [8]. To the best of our knowledge, the current statistical and machine learning methods proposed have considered measurements on solar PV power output, NWP model irradiance data and other atmospheric parameters as model inputs. While these input variables are a highly sensible selection to build time series models for solar PV power forecasting, they lack direct information on the most recent cloud development, which is the main factor influencing solar irradiance at the Earth's surface [10]. A forecasting techniques based on satellite images in combination with cloud motion vectors aims to tackle this information gap. However, this is a highly complex forecasting technique that is not easily deployed and involves numerous processing steps [10]. This presents a knowledge gab on the application of satellite images for solar PV power forecast in a straightforward, yet accurate way.

To address the continuous need for accurate solar PV power prediction methods and bridge the current knowledge gab with respect to the use of satellite image data for solar PV power forecasting, this project has the following research objective: investigate and leverage on the recent developments in the field of deep learning, with respect to convolutional neural networks (CNN) and Long Short Term Memory (LSTM), to process real-time satellite images and solar PV power generation data, in order to make accurate deterministic intra-day solar PV power forecasts. This study concentrates on the proposal of two forecasting frameworks that can be applied in a real-time operational setting and generate an aggregated solar PV power forecast over Germany on a 15-minute interval and with a forecast horizon of 3 hours. The performance of the novel forecast frameworks are compared to local energy market standard solar PV power forecasts and a persistence model based forecast.

In chapter 2 of this report, an literature review is provided on the main approaches to solar PV power forecasting. An introduction in machine learning techniques and a discussion on several required consideration in deep learning model setup is presented in chapter 3. In chapter 4 the data pre-processing techniques and the frameworks of the two proposed solar PV power forecasting frameworks are outlined. The overall model performance result and several forecast case studies for Germany are discussed and compared to the benchmark models in chapter 5. Finally, in chapter 6, the report is concluded and recommendations are given on further development and future research.

Chapter 2: Solar PV power forecasting techniques

In this chapter an overview of solar PV power forecasting is presented. It aims to provide the reader with the necessary background needed to understand the terminology and forecasting concepts considered in this research. First, in section 2.1, the two main classes of forecasting techniques are introduced. Next, in section 2.2, a further explanation is given on sky and satellite imagery based models. In section 2.3 the spatio-temporal aspects of solar PV power forecasting are discussed. Finally, in section 2.4, the relation between energy market characteristics and solar PV power forecasts is discussed.

2.1 Forecast techniques

Many types of solar irradiance and subsequently solar PV power forecasting models have been developed [9]. These models can be divided in two main groups based on the used approach. The first technique consists of using physical equations to model the power to be produced by PV systems. In this method most efforts are committed to obtain accurate irradiance forecasts, as this is the main factor influencing the power generation [8]. This approach is known as physical or 'white box' method. In contrast, the second approach is based on directly forecasting PV power output using statistical or machine learning methods. These schemes are sometimes called direct or 'black box' methods, as it is not required to manually engineer preordained physical equations in the forecast model. Additionally, an ensemble of both approaches can be used in an effort to incorporate the positive aspects of one and the other. These are often denoted as hybrid models or 'grey box' models, not to be mistaken with an hybrid of several statistical or machine learning methods.

2.1.1 Physical models

Most of the efforts of a physical solar forecasting method is made in the prediction of solar irradiance and additional meteorological parameters, such as atmospheric temperature, humidity, wind velocity, barometric pressure and aerosol changes [8]. All these parameters are dependent on geographical locations and climatic conditions. Solar irradiance correlates the strongest with PV power output compared to other meteorological parameters [8]. As an indication of the dependencies of several parameters, table 2.1 lists the correlation coefficient between solar PV power output and meteorological factors found in a correlation analysis at a PV power plant site in Ashland, USA [11]. By implementing many relevant input variables that show a high correlation for the specific local weather conditions, the performance of a physical solar forecasting method can be enhanced. However, trying to impose every single input vector on a model is not a feasible nor computationally efficient task [11]. Hence, one of the challenges of designing a good physical model lies in the selection of an optimum number of input variables that are highly correlated with solar PV power.

Table 2.1: The correlation between meteorological parameters and solar PV power output at a site in the USA [11].

Meteorological factor	**Correlation coefficient**
Solar irradiance	0.9840
Air temperature	0.7615
Cloud type	-0.4847
Dew point	0.6386
Relative humidity	-0.4918
Precipitable water	0.3409
Wind direction	0.1263
Wind speed	0.1970
Air pressure	0.0815

Predictions of meteorological parameters are obtained from NWP models. These models forecast the temporal development of the state of the atmosphere using a set of coupled partial differential equations which represent the physical laws determining the weather [10]. Global NWP models are initialized using a set of conditions obtained from worldwide observations, after which the future state of the atmosphere is modeled per time

step [10]. Examples of global NWP models are the European Centre for Medium-Range Weather Forecasts (ECMWF) and the Global Forecast System (GFS). These models can forecast the state of the atmosphere more than 15-days ahead, but have a coarse resolution in the range of 9-50 km [12] [13]. In order to tune NWP models for more local climate effects and a higher spatial and temporal resolution, global models are scaled down to mesoscale or regional models, often run by national weather services or companies [10]. An example of a regional NWP model covering Germany is the Deutscher Wetterdienst (DWD) ICON-D2 model, which has a spatial resolution of 2.2 km and in the vertical defines 65 atmosphere levels. The ICON-D2 model is operated for a relative short forecast horizons of 27 hours (45 hours only for 03 UTC run) and update frequency of 3 hours.

When the technical specifications of a solar PV power plants are known, together with the irradiance forecast and other meteorological parameters from NWP models, a local or regional power forecast can be made as visualized in fig. 2.1. As such, the physical conversion of input parameters into power output is strictly speaking not the forecasting technique on its own. The main forecasting effort lies in the prediction of relevant meteorological parameters in a NWP model. This characteristic gives rise to the main advantage and disadvantage of a physical forecasting model. The major advantage of a physical model over a statistical model is that for local forecast there is not necessary the need for historic data [14]. Although, strictly speaking historic data is often used for statistical post-processing of NWP data to corrected for systematic biases, called Model Output Statistics (MOS). With solar PV plant technical specifications and NWP data a power forecast can be made, even before plant construction. On the contrary, the high dependency on the input parameters obtained from NWP models is also the main disadvantage. The errors in NWP model parameters persisting in the power forecast, together with inadequate temporal and spatial resolution of NWP for local solar PV systems, are considered to be the lead source of inaccuracy for this approach [9]. An other method to derive irradiance predictions is with the use of satellite images as also visualized in fig. 2.1. This method is also classified as a physical forecast and described in further detail in section 2.2.

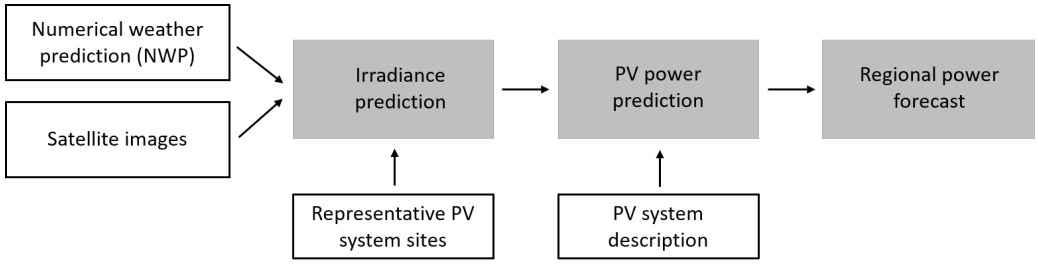


Figure 2.1: Illustration of a general physical solar PV power forecasting scheme.

2.1.2 Statistical models

Statistical models aim to forecast solar PV power output directly using conventional statistical or new-generation machine learning methods. This is a data-driven approach which tries to extract relations in past observations, for example irradiance or solar PV power output observations, in order to forecast the future power output of a solar PV power plant. In these models it is not required to manually engineer preordained physical equations in the forecast and does not necessarily require site specific system information. For this approach the quantity and quality of historic datasets is essential for the development of an accurate model [9]. Historical datasets can be obtained from meteorological and power measurements, and from archived NWP forecasts. Similar to a physical model, a challenge of a statistical model design lies in the selection of appropriate and an optimal number of input parameters which yield the best results, while at the same time balances the trade-off between accuracy and complexity. Contrary to physical models, not only meteorological parameters can be available, but also measurements of PV system power output, current, module temperature and other parameters. Data obtained from time series records of PV power plant output are called endogenous inputs, and meteorological parameter data obtained from measurements or NWP models are known as exogenous inputs.

Statistical techniques can be divided in two main groups: time series based models (often linear) and machine learning models (often non-linear). In time series based models a statistical approach is used to map relations between input variables, the predictors, and the variable to be predicted. The general idea is that by evaluation

patterns in past time series based data, and represent this in a mathematical expression, a prediction of future values can be made. Some well established and applied techniques include: exponential weighted moving average (EWMA), auto-regressive moving average (ARMA), auto-regressive integrated moving average (ARIMA) and seasonal auto-regressive integrated moving average (SARIMA) [8]. Models where additional predictions of exogenous variables are introduced include: auto-regressive exogenous (ARX) and auto-regressive moving average exogenous (ARMAX) models [15]. In table 2.2 an overview is given of several examples where linear time series based models are applied for solar power forecasting. Several of these studies provide a comparison to machine learning techniques, such as artificial neural networks (ANN) and k-nearest neighbors (k-NN), or hybrid models.

Table 2.2: Overview of several statistical linear solar PV power forecasting models.

Authors	Horizon	Resolution	Methods	Description
Bacher et al.[16]	1-36 h	1 h	AR, ARX	Forecast of normalized solar PV power are made using an AR model with 15-min power observations and an ARX model with additional exogenous NWP irradiance forecast data. Results indicate that up to 2 h ahead endogenous power input is most important.
Pedro et al. [17]	1-2 h	1 h	Persistence, ARIMA, k-NN, ANN	Comparison of several statistical methods using only exogenous 1h resolution inputs of a 1 MWp solar PV plant. Findings show that ANN based model outperforms on both 1-2 h forecast horizon, ARIMA outperforms persistence on 2 h horizon.
Li et al. [15]	Day-ahead	1 h	ARIMA, ARMAX	Comparison of ARIMA model with endogeneous inputs and ARMAX model with additional NWP forecast exogenous inputs (not irradiance) for a 2.1kW PV system. Forecast are made for day-ahead daily power generation, here ARMAX outperforms ARIMA due to the availability of NWP forecasted atmospheric parameters.
Reikard et al. [18]	5 min - 4 h	5, 15, 30, 60 min	ARIMA, ANN	Comparison of ARIMA and ANN for the forecasting at various resolutions of global horizontal irradiance at 6 sites in the USA using past irradiance data. ARIMA outperforms at most resolutions and forecast horizons as it is better in capturing the diurnal cycle.
Bouzerdoum et al. [19]	1 h	-	SARIMA, SVM, Hybrid	Evaluation of SARIMA, non-linear support vector machine (SVM) model and hybrid model for 1 h horizon forecast of solar PV power generation by a 20KWp plant in Italy. Results show that hybrid model outperforms, as it combines SARIMA capturing the linear components in the power while SVM finds non-linear patterns.

In the group of non-linear models machine learning techniques are used, an approach that is build on artificial intelligence. This method depends on a machine learning model to gain predictive capabilities based on experience with historical data. In these models computers run many iterations with historic data before a model is experienced enough to make accurate predictions using newly available data. These models can learn complex representations of the input data without needing preordained physical or statistical equations, as used in physical or linear statistical forecast models [20]. Some well established machine learning techniques in the field of solar PV power forecasting include: artificial neural networks (ANN), k-nearest neighbors (k-NN), support vector machines (SVM) and random forest (RF) [9]. Deep learning is the current state-of-the-art class of machine learning and is increasingly being applied in solar PV power forecasting [8]. Deep learning models are able to learn even more complex and non-linear representations for input-output mapping, while processing input data in its raw form, something that is not possible with conventional machine learning techniques [21]. Some deep learning techniques used in literature for forecasting include: deep neural networks (DNN), convolutional neural networks (CNN), auto-encoder (AE), recurrent neural networks (RNN) and Long Short Term Memory (LSTM). All these techniques have their own characteristics and are used for specific applications. A detailed description of the machine learning techniques used in this report is presented in chapter 3. Similar to linear models, endogenous and exogenous data can be used as input variables for machine learning models. In table 2.3 an overview is given of several examples where machine learning models are applied in the field of solar power and irradiance forecasting.

Many variants of physical and statistical solar power forecasting methods have been proposed and applied. However, there is no unique technique capable of making accurate predictions in all weather situations, as each model may omit some crucial information for the prediction of a particular weather situation [22]. Hence it is a common practice to combine several techniques to capture the individual strengths of different models [22]. This technique is often denoted as ensemble, combined, blended or hybrid models. Models can be combined in several ways, such as by stacking, bagging, boosting or voting [9]. Hybrid models can be made by combining a number of statistical techniques (hybrid-statistical) or by blending a statistical and physical model technique.

Table 2.3: Overview of several statistical non-linear solar PV power forecasting models.

Authors	Horizon	Resolution	Methods	Description
Pedro et al. [16]	15 min - 2 h	-	ANN, k-NN	Comparison of ANN and k-NN for the forecasting of global horizontal irradiance on various temporal horizons at 5 different micro-climates in the USA making use of irradiance measurements. Models outperform simple persistence models on all timescales and conclude that relevant input feature vary for different micro-climates.
Jie et al. [23]	Day-ahead	15 min	SVM	A SVM model is proposed for the forecasting of day-ahead PV power for a 20 KWp system, using 4 models based on weather classification: sunny, rainy, foggy, cloudy. Input is PV power measurement of nearest day with the same weather type and day-ahead NWP temperature.
Lee et al. [23]	Day-ahead	-	CNN+LSTM, AE+LSTM	Deploys a combination of a CNN or AE to extract relevant features from a 10-min interval power and atmospheric parameter observations matrix, and subsequently uses an LSTM architecture to predict day-ahead solar PV power output. Results show CNN+LSTM outperforms AE+LSTM and other conventional statistical methods. Additional of weather inputs increases forecast accuracy.
Do et al. [24]	1 h	-	ANN	Study analyzing the training period required to forecast PV power at two plants using instantaneous PV power measurements and exogenous parameters: cloud cover and temperature. ANN model outperforms persistence and AR benchmark and shows that a longer training period is required for locations with seasonal effects (min 6 months).
Li et al. [25]	15 - 90 min	15 min	RNN, LSTM, ANN, SVM, RBF	Uses intraday and adjacent day power measurements of 2.1 GWp net region in Belgium as input for RNN to forecast power on 15-min resolution, benchmarked against LSTM, ANN and other methods. RNN outperforms on all forecast horizons with LSTM results inline, but has increasing errors for 75 and 90 min forecast. Forecast error larger in winter compared to summer.
Lago et al. [26]	1 - 6 h	1 h	ANN	Deploys an ANN to forecast irradiance at 25 locations in the Netherlands using forecasted NWP and clear sky irradiance, plus past irradiance values derived from satellite images using a SICCS algorithm. Method shows potential of regression of various input sources of past and future irradiance for local forecasts.
Abdel-Nasser et al. [27]	1 h	1 h	LSTM	Analyses of different LSTM architecture for the forecasting of PV power for systems in Egypt, based only on power measurements. Outperforms linear benchmark models and suggest multiple previous time-steps as input yields best result.

2.2 Sky imagery and satellite based models

In physical forecasting models solar irradiance is the most important parameter, as it correlates the strongest with solar PV power output, as described in section 2.1.1. The presence of clouds is the main factor influencing solar irradiance at the surface, of which cloud cover and cloud optical depth are of main consideration [10]. Hence, next to obtaining irradiance forecast from NWP models, efforts have been made on the prediction and tracking of cloud movement and subsequently derive irradiance. Cloud formation and movement follow certain physical rules, however turbulent and convective processes make it a very difficult task to model the behavior [9]. Yet, for forecast horizons up to several hours, the change of cloud structures and position through time is strongly influenced by cloud motion as a result of horizontal advection [10]. Ground-based sky imagers and satellite images are used for the derivation and forecasting of solar irradiance from cloud cover. A clear sky model is often used in these methods to derive the irradiance at a location and point in time under cloudless conditions, in section 4.2 a detailed description of such a model is given.

Sky imagers produce high quality images of the sky, which are used for cloud detection, cloud classification, cloud height estimation and calculating cloud motion [28]. Past consecutive sky images are used to estimate cloud velocity, after which extrapolation of the motion in time is used to derive future cloud positions and subsequently forecast the future irradiance. The maximum forecast horizon of this technique is strongly influenced by the cloud velocity and the spatial extension of the images [28]. In contrast to satellite images, the sky images have a much higher temporal and spatial resolution, and can thus capture sudden changes in irradiance over a certain site. Sky imagers are mainly used for local solar PV power forecast and have a maximum temporal horizon of approximately 30-minutes [28].

Images from geostationary satellites such as NOAA and Meteosat can be used in a manner similar to the observations from sky imagers. In a technique based on the satellite data, consecutive images are combined to create cloud motion vector (CMV) fields, which are used to extrapolate cloud motion and derive future cloud position [29]. Next, a physics-based and empirically adjusted algorithm can be used, such as Surface Insolation under Clear and Cloudy skies derived from SEVIRI imagery (SICCS), to model for light attenuation by clouds in future positions in order to forecast irradiance [30]. Subsequently, a power forecast can be made using the irradiance forecast and PV system characteristics. Regional solar PV power forecasting using cloud motion vector fields is a complex process involving many steps and requires knowledge on all PV system technical parameters. It has however shown to be effective for irradiance forecasting on a temporal horizon of up to 4 hours [29]. This method shows less accuracy under convective and marine layer cloud regimes, as under these conditions clouds can rapidly from or disappear [28].

2.3 Spatial and temporal aspects of forecast

For the temporal aspect of solar PV power forecast 3 concepts are important: forecast resolution, forecast interval and forecast horizon [9]. The forecast interval describes the frequency at which a new forecasts is issued by a model. The forecast resolution expresses the time range between subsequent point predictions in the forecast. Finally, the forecast horizon is the time period between the effective time of the forecast and the actual time for which the forecast is made. The forecast horizon can be broadly categorized in four groups, although no universal classification criteria exist [8]:

1. **Very short-term** forecast models have a forecast horizons from seconds to 30 minutes. These forecasts find their application in smart grids, power smoothing processes, real-time power system dispatch and energy storage control.
2. **Short-term** forecast models have a temporal horizon between 30 minutes and 6 hours. These forecast are often used in intraday electricity markets, renewable power management systems and economic dispatch models.
3. **Medium-term** forecast models span 6 to 24 hours. These forecast models find applications in day-ahead economic dispatch and electricity markets, reserve planning and maintenance scheduling.
4. **Long-term** forecast models consider time periods longer than 24 hours. These prediction horizons are suitable for the long term planning of transmission, distribution and power generation.

Solar PV power forecast can be made for a range of spatial horizons, from a single local PV system to a regional power forecast. Different forecasting techniques are applied based on the spatial coverage required for the application. Sky imagers are only used for single plant or very local forecast, whereas satellite images and NWP based physical models are often applied for regional aggregated power forecast [28]. Statistical models find their applications in both local and regional forecast. The main difference between local and regional forecast is found in the short term power output variability [31]. Regional forecast, including a number of dispersed solar PV systems, benefit form forecast error reduction due to spatial averaging and smoothing effects [31].

Many different solar forecasting approaches have been proposed and studied in literature, which vary based on the forecast horizon, forecast resolution, spatial application, data availability and types of input variables. Comparing different forecast approaches and determining the optimal forecast technique for a certain application is a challenging task. Nonetheless several authors have made overviews of forecast approaches and application using a classification based on the spatial and the temporal resolution, all with slightly varying results [9] [10]. Based on these two aspects fig. 2.2 presents a broad classification and the following summary of the discussed forecasting methods can be made:

1. Sky images based models make use of local data with a small spatial horizon and are suitable for forecast with a temporal horizon up to 30-minutes [28].
2. Satellite image based cloud motion vector models have a rather large spatial resolution and horizon, and are therefore applied in regional irradiance and subsequent power forecast with a temporal horizon up to 4 hours [29].

3. Physical models which are based on atmospheric parameters from NWP models inherit the coarse spatial and temporal resolution of these input parameters. Hence these models are most often successful applied in forecast with a temporal longer than 2-4 hours and for regional forecast [9].

4. Both statistical time series based models and machine learning models are used for local and regional forecast, and depending on model technique are used from 5-minute to day-ahead forecast horizons [9].

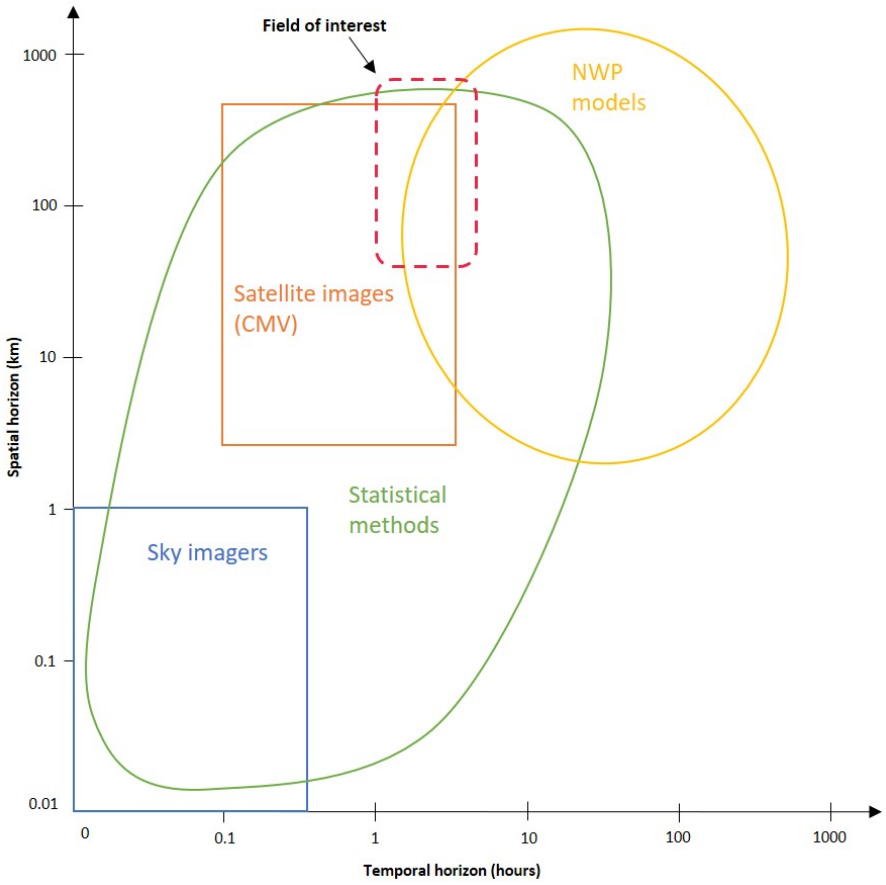


Figure 2.2: Broad classification of various forecasting methods based on the spatio-temporal horizon of its application. The highlighted field of interest is discussed in section 2.4 and related to short-term solar PV power forecasts.

2.4 Forecasts and energy markets

Solar PV power forecast are becoming increasingly important to energy markets because of the continuously growing installed capacity, which results in a major contribution of solar PV power to the energy supply. The positive forecast scenario of SolarPower Europe projects a PV capacity growth of 32.2 GW in Germany by 2024, which is a significant increase over the total installed capacity of 57.5 GW in December 2020 [2]. Currently, on sunny days, solar PV generation can already temporarily supply up to 50% of the electricity demand in Germany. In general, the greatest contribution of solar PV generation coincides with the peak power demand around noon.

Electricity, including solar PV, is traded on various marketplaces. The duration of the contracts, delivery time-frame and form of the transaction define these marketplaces. Currently, there is no technology available to store electricity economically in large quantities, therefore power is traded using long-term and short-term contracts in order to optimize the match of power demand and supply [32]. The contracts imply an obligation to deliver or consume a certain quantity of power for a certain period at a agreed-upon price, i.e., a futures contract.

In the EU short-term power contracts are traded via exchanges, with the largest being EPEX Spot and Nordpool. Typical short-term power markets are the day-ahead market for the upcoming day and intraday market for the current day. On the day-ahead market power is traded in dedicated hour, half-hour or quarter-hour blocks, and additionally customized time intervals can be traded. Trading periods vary between exchanges, on EPEX Spot the deadline for the day-ahead auction is at noon of the day before the delivery day [33]. On the intraday market power is most commonly traded in dedicated hour, half-hour and quarter-hour intervals, although custom time intervals are also possible. These contracts can be traded from the previous day up to 5 minutes before delivery (lead time) in some countries, including Germany [33]. An important difference between the intraday and day-ahead market is the pricing mechanism. On the day-ahead market the product price is determined by a market clearing price principle, in which the last accepted bid sets the price for all transactions in that contract. On the intraday market the contract price is continuously based on the price of each transaction, this is known as the pay-as-bid principle and similar to stock markets.

A significant deviation of the actual solar PV generation from the forecasted power output requires actions by solar PV asset owners. In order to avoid a costly position with a shortage or surplus of power in the imbalance, asset owners need to offset these difference by buying or selling power on short-term power markets. The resulting increase in supply or demand can cause strong market price movements. The considerable share of variable solar PV power in the German grid and the resulting impact on market prices, lead to a active economic interest in cutting edge solar PV power forecast methods by many parties involved in the energy market.

The various different power contracts require dedicated solar PV power forecast tools designed specifically for their unique characteristics. On the day-ahead market, power is sold one day in advance, requiring forecast with a 1 day horizon and typically a resolution of 1 hour. On the intra-day market an update is required for the forecasted generation of that specific day. Here, two forecast types are of importance. One forecast is required for the remaining period of the day, usually made in the morning based on the latest NWP model data and having a forecast resolution of 1 hour. Next, an additional short-term forecast is required, used to anticipate sudden changes in the solar PV generation based on the latest weather conditions. These forecast require a temporal horizon of minimal 30 minutes to maximal 6 hours and a resolution of 15-minutes equal to the quarter-hour contracts. Moreover, solar PV power forecast used for trading on the energy market need to be provided on a regional level, as the power contracts are marketed for entire grid control regions, representing an area of several hundred kilometers. The specific spatio-temporal characteristics required for short-term intra-day forecast, is highlighted in fig. 2.2 as a field of interest. This area, with overlapping forecast methodologies based on spatio-temporal horizon, is of special interest for trading companies active on the energy market, and hence the focus of this research project.

Chapter 3: Machine learning techniques

Machine learning is a modeling method that relies on computing and artificial intelligence to learn from experience with data in order to perform certain tasks. Machine learning is at the core of modern day society, as it finds its application in web searches, voice recognition, e-commerce advertisements and is increasingly present in our day-to-day consumer products such as phones and cars [34]. In all these applications the deep learning class of the machine learning techniques has become the dominant go-to approach [34]. Deep learning allows models with multiple layers to learn complex and non-linear representations of input data, while processing the data in its raw form, something that is not possible with conventional machine learning techniques [21]. Furthermore deep learning allows models to automatically extract, analyze and understand complex relations between input and output, without the need of manually engineering the model [21].

Both deep learning and conventional machine learning can be classified based on two types of learning methodologies, unsupervised and supervised learning [21]. In unsupervised learning the target variable is not provided, that is, only input data is available with no corresponding output variable. The goal of an unsupervised learning model is to learn an underlying structure or distribution of the input data, for example a certain grouping. In supervised learning the input data and the target variable are provided. The goal of an supervised learning model is to learn a mapping function from the input variables X to the output variable y, that is: $y = f(X)$. The learned mapping function can be used to predict the output variables for new and unseen input data. Supervised learning can be further grouped into classification and regression problems. In a classification problem the output variable is a category, where in an regression problem the output variable is a real value.

In the field of deep learning, algorithm such as Convolutional Neural Network (CNN) have presented interesting results in the processing of images and videos [35]. Other deep networks like Long Short Term Memory (LSTM) have yielded better results with sequential data, such as speech, text and time series forecasting [35]. Although these deep learning algorithms form the foundation of the forecast models developed in this research project, the basic concepts of deep learning are best introduced via a discussion of Artificial Neural Networks (ANN) as presented in section 3.1. Next, in section 3.2 the working principles of a CNN is discussed. Finally, in section 3.3 the functionality and operational principles of a LSTM network is described in detail.

3.1 Artificial Neural Networks (ANN)

The Artificial neural network model is originally inspired by the biological theory of neurons [35]. The working of an ANN can be compared to the functioning of the human brain, as it consists of many complexes of interconnected ”neurons” working together to solve a specific problem and learn by example like people do [35]. ANNs are very versatile, with a wide range of architectures and specialized designs for specific applications [35]. A class of ANN is the feedforward neural network (FFNN), which can be used for classification and regression tasks. The architecture of a FFNN is composed of several connected layers, which subsequently consist of unit cells or nodes. An example of FFNNs is a multi-layer perceptron (MLP), of which an exemplary three layered network architecture is shown in fig. 3.1. This example MLP shows a regression task, where the model maps the input variables irradiance, temperature and cloud cover to the output variable solar PV power, just as a physical model could do.

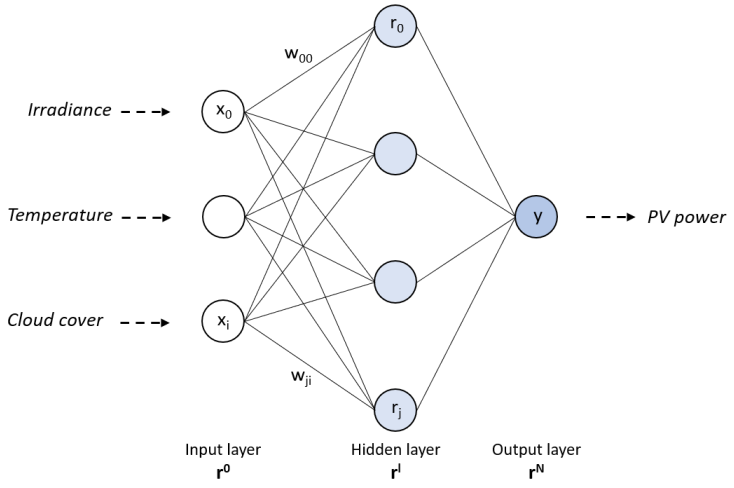


Figure 3.1: Architecture of a two-layered MLP used for a regression task.

The architecture of the MLP in fig. 3.1 can be broken down in the following components:

1. The first layer is the input layer, which gathers the model input vector x. Each unit corresponds to one feature from the input data stream, in fig. 3.1 these are irradiance, temperature and cloud cover.
2. The layer in between the input and output layer is called the hidden layer. In this case there is only 1 hidden layer, however multiple hidden layers can be added which pushes the model to the deep feedforward neural network (DFFNN) class. The hidden layers are responsible for the complex non-linear input-output mapping.
3. The last layer is the output layer, this yields the model output vector y. The output layer provides the target variable, which is a known quantity in the training process of a supervised learning model. In fig. 3.1 the output 'vector' consists only of 1 variable, namely solar PV power.

The information processing in the network involves the flow of input data through the hidden layers until it reaches the output layer. Each unit in layer (l) is connected to all other units in the previous layer $(l-1)$ via a certain weight w_{ji} represented by the lines in fig. 3.1. For the mapping of the input vector $\boldsymbol{x}$ to the output vector $\boldsymbol{y}$ an expression can be given for the activity of all units in the input layer, hidden layer and output layer by eq. (3.1), eq. (3.2), eq. (3.3) respectively. In these equations the vector $\boldsymbol{r}^{(l)}$ represents the unit activity of all units in the (l)-th layer. The $\boldsymbol{W}^{(l)}$ is the connection matrix form the $(l-1)$ to the (l)-th layer consisting of all weight w_{ji} connecting the nodes of the two respective layers. The $f(\cdot)$ and $g(\cdot)$ are the activation function of the model neurons. The vector parameters $\boldsymbol{b}^{(l)}$ are the biases for the model neurons. Note that in these equations $l = 0$ is the input layer, and $l = N$ is the output layer.

$$\boldsymbol{r}^{(0)} = \boldsymbol{x} \tag{3.1}$$

$$\boldsymbol{r}^{(l)} = f\left(\boldsymbol{W}^{(l)}\,\boldsymbol{r}^{(l-1)} + \boldsymbol{b}^{(l)}\right), \quad 0 < l < N \tag{3.2}$$

$$\boldsymbol{r}^{(N)} = \boldsymbol{y} = g\left(\boldsymbol{W}^{(N)}\,\boldsymbol{r}^{(N-1)} + \boldsymbol{b}^{(N)}\right) \tag{3.3}$$

In the hidden layers an activation function $f(\cdot)$ is applied to the multiplication of the connection matrix and the activity vector of the neurons in the previous layer, plus the bias $(\boldsymbol{W}^{(l)}\,\boldsymbol{r}^{(l-1)} + \boldsymbol{b}^{(l)})$. In a similar manner an activation function $g(\cdot)$ is applied to the connection from the last hidden layer to the output layer. The choice of activation function has significant impact on the operation and capabilities of a ANN. The choice for a non-linear activation function, introduces non-linearity to the model which is required for an ANN to process data, learn and map inputs to outputs with any arbitrary complex function [36]. If no activation was used, the ANN would act as a linear regression model and the input-output relation would simply be a one degree polynomial function. The use of non-linear activation functions make ANN universal function approximators, hence in theory it can approximate any arbitrary function [37]. This characteristic makes the performance of ANNs standout compared to conventional statistical models. Typically all hidden layers have the same

activation function, however the output layer often has a different activation function $g(\cdot)$ depending on the required model task [36]. The choice of an activation function is not a straightforward process and is often context and task dependent [36]. This makes the activation function a model hyperparameter, which is further discussed in section 3.1.2. However, first the concepts involved in model training are discussed in section 3.1.1. Finally, in section 3.1.3 the concept of generalization is introduced.

3.1.1 Training algorithm

In an ANN, it is the aim to find an optimal mapping function to relate input data to output data. This is done by searching for a set of trainable model parameters that does this best for all data. It is not possible to directly compute the optimal set of parameters nor can it be guaranteed that a model converges to an optimal set. In a MLP, the trainable parameters are the connection matrix $\boldsymbol{W}$ and the bias vector $\boldsymbol{b}$. Training a MLP consist of updating the trainable model parameters subject to an objective function, also known as a cost or loss function. Gradient descent is a technique to minimize this loss function by updating the model parameters in the direction opposite to the gradient of the objective function [38].

There are three variants of gradient descent: batch gradient descent, stochastic gradient descent (SGD) and mini-batch gradient descent [38]. These variants differ only in the quantity of data that is used at once to compute the gradient of the objective function, which governs a trade-off between accuracy of the parameter update and the time required to execute an update. For deep neural networks mini-batch gradient descent is the typical used algorithm [38]. The quantity of data used in the gradient descent technique is described by the batch size, which is a model hyperparameter and further discussed in section 3.1.2. The processes of model training using SGD is visualized in fig. 3.2 and can be summarized in the following steps [37]:

1. In the first phase forward propagation occurs. In this process the input data $\boldsymbol{x}$ is passed into the network to produce an output $\boldsymbol{y}$, following equations eq. (3.1) - eq. (3.3). In the very first pass the trainable parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ are initialized randomly.
2. In the next phase a loss function is used to quantify the error between the model output ($\boldsymbol{y}$) and the target variable ($\hat{\boldsymbol{y}}$) provided in a supervised learning task.
3. In the final phase backward propagation occurs. In this process the calculated loss is used to update the trainable model parameters $\boldsymbol{W}$ and $\boldsymbol{b}$. Let L be the loss function and let $\boldsymbol{\theta}$ collectively denote all trainable parameters, then the gradient of the loss function is $\partial L/\partial \boldsymbol{\theta}$. In order to decrease the error, trainable parameters are updated in the direction opposite to the gradient, with a magnitude proportional to the learning rate η as in eq. (3.4). Starting from the output layer, the loss information updates the trainable parameters of the units in the hidden layers based on the relative contribution they made to the output. When all trainable parameters are updated, the process is repeated for a new batch of data.

$$\Delta\theta = -\eta \frac{\partial L}{\partial \theta} \tag{3.4}$$

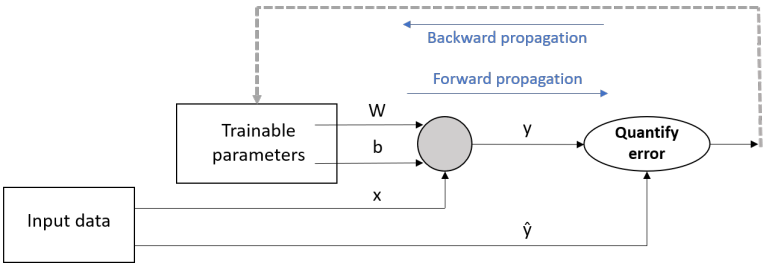


Figure 3.2: Simplified schematic representation of model training process for a one perceptron network.

For an ANN with many layers, back-propagation can be a slow and challenging task. Here the loss function is represented by a cost surface, which is typically a non-convex, non-quadratic and a high dimensional structure with numerous local minima [39]. There is no guarantee that a network will converge to an adequate solution, will converge fast or will converge at all [39]. As a consequence there are some challenges when setting up the gradient descent algorithm [38]:

- Selecting an adequate learning rate. A learning rate which is too large can hinder convergence or can even result in divergence, and a learning rate that is too small will have extremely slow convergence.
- Setting up a good learning rate schedule. This can solve the problem of choosing a single learning rate for the entire training process, as the schedule adjusts the learning rate according to the amount of model convergence, i.e., lowering the learning rate when the change in the loss function falls below a certain threshold. However a problem is that the learning rate needs to be set in advance, which makes it incapable of to adapting to specific dataset characteristics.
- All parameters are updated by the same learning rate. Depending on the characteristics of the features in a models input data, it might be effective not to update all features at the same rate or to the same extend. For example, one might want to perform large updates, with high learning rates, for infrequent features.
- Setting adequate learning rates in order to prevent against getting trapped in not only local minima, but also saddle points in the cost surface characterized by a plateau with an error of the same magnitude. This is particularly important for SGD, as for a saddle point the gradient will be nearly zero in all directions and subsequently the algorithm will not converge further.

To overcome these challenges several algorithms have been introduced which add elements to SGD in order to improve its robustness [40]. These algorithms include: Momentum, Adagrad, Adadelta, Adamax, RMSprop and Adam. One of the most widely used optimization algorithm in deep learning is the Adam, short for Adaptive Moment Estimation [40]. Adam computes adaptive learning rates for each parameter and in addition combines the advantages of RMSprop and AdaGrad, related to sparse gradients and non-stationary settings. Adam has several other advantages: it is easy to implement, computationally and memory efficient and requires little tuning for the learning rate when desired [40]. The choice of optimization algorithm is a model hyperparameter, for which Adam is the common go to approach in deep learning [40].

3.1.2 Hyperparameters

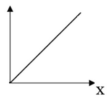
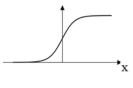
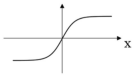
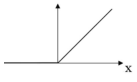
When training a model using an optimization algorithm, a number of other decisions must be made on variables in the training process, called hyperparameters. The determination of hyperparameter settings is often guided by a combination of empirical evidence (trail-and-error), theory and hardware constraints [37]. Several important hyperparameters are discussed in this section and acts as a guidance for proposed model setup in section 4.3

Loss function
To quantify the error between a model output ($\boldsymbol{y}$) and target output ($\hat{\boldsymbol{y}}$) an objective function must by chosen. The selection of the loss function depends on data characteristics and the model task. For regression tasks the most common loss functions are Mean Absolute Error (MAE, or L1-loss), Mean Squared Error (MSE, or L2-loss), log cosh loss or Huber loss.

Activation function
The choice of activation function has a significant impact on the operation and capabilities of an ANN [36]. A list of several activation functions which are often applied in the field of machine learning is provided in table 3.1. These activation functions have different characteristics based on their mathematical function. For example, the sigmoid function transforms any input in the domain $(-\infty, \infty)$ to outputs in the range $(0, 1)$; whereas the rectified linear unit (ReLU) function sets all values in the domain $(-\infty, 0)$ to zero, while passing values through for the domain $(0, \infty)$. The choice of an activation function is not a straightforward process, as each has its own advantages and disadvantages depending on the system design and context of the model task. However, the ReLU activation function is currently the most common choice for hidden layers [36]. ReLU has shown to outperform other activation functions, as it reduces the vanishing gradient problem and accelerates convergence [36].

Weight initialization
At the beginning of a training process a starting point is required, this is defined by the initial model trainable parameters $\boldsymbol{W}$ and $\boldsymbol{b}$. As in model training the loss surface is non-convex, the optimization algorithm is sensitive to the chosen initial starting point parameters [41]. The initial weight values are small random numbers generated by a weight initialization method, which governs the scale and distribution of these values. The most widely used initialization methods are Xavier uniform initialization and He normal initialization [42]. In the Xavier uniform initialization method the initial values are selected from a bounded random uniform

Table 3.1: Overview of commonly used activation functions in literature.

Name	Equation	Plot
Linear	$f(x) = ax$	
Sigmoid	$f(x) = \frac{1}{e^{-x}}$	
tanh	$f(x) = \frac{2}{1+e^{-2x}} - 1$	
ReLU	$f(x) = max(0, x)$	

distribution as given in eq. (3.5), in which (n_i) and (n_j) are the number of incoming and outgoing network connections respectively. This method solves the problem of the saturation of initial layers in early stages of training. This problem occurs when using the sigmoid activation functions, which is relevant for LSTM networks [42]. In the He normal initialization method parameters are selected from a bounded normal distribution as given in eq. (3.6), in which (a) is a parameter of the Relu activation function class (ReLU $a = 0$, PReLU $a > 0$) [43]. This method works better with ReLU activation functions as it helps to attain global minima of the loss function more efficiently, which is particularly relevant for fully connected layers [42].

$$U[(-\frac{\sqrt{6}}{\sqrt{n_i + n_j}}, \frac{\sqrt{6}}{\sqrt{n_i + n_j}})] \tag{3.5}$$

$$N[(-\frac{\sqrt{6}}{\sqrt{n_i(1 + a^2)}}, \frac{\sqrt{6}}{\sqrt{n_i(1 + a^2)}}) \tag{3.6}$$

Batch size
As discussed earlier in section 3.1.1 mini-batch gradient descent is the typical used algorithm in model training. Here the loss and subsequent gradients are calculated in batches, which are a small defined number of samples from the training dataset. The batch size has an effect on the convergence speed and computational efficiency of a training process. Compared to large batch sizes, small batches have higher convergence speed but are less computationally efficient and have a more noisy training process. Common mini-batch sizes range between 2 and 256, but can vary for different applications and various different results have been shown in literature [44] [38].

Epochs
The model training process must be repeated many times with the same training data for the model trainable parameters to reach a minima in the objective function. The training epochs are the total number of iterations through the complete training dataset, after which the training process is stopped. Too little epochs will lead to an oversimplified mapping function approximation, while too many epochs will lead to increased generalization error and overfitting. The proper amount of epochs is model and task dependent, and often determined using a trail-and-error method or using a callback algorithm such as early stopping.

Learning rate
As introduced in section 3.1.1 the learning rate is an important parameter in the model training algorithm. A learning rate that is too large can hinder convergence or can even result in diverge, and a learning rate that is too small will have extremely slow convergence. Specialized optimization algorithms, such as Adam, introduce adaptive learning rates for each parameter. However, the setting of an adequate global learning rate or learning rate schedule in combination with Adam can still aid the model training performance. In a learning

rate schedule the learning rate is lowered during the process of model convergence. This can be done with a pre-set schedule, a constant or exponential rate decay, or based on a certain threshold value of the loss function. The effects of the learn rate is often investigated using a trail-and-error method.

3.1.3 Generalization

Machine learning is basically a "generalization" process which learns a mapping function from the input variables X to the output variable y based on training data. The main goal in the development of an ANN is to create a model which performs well on both the training dataset and new data not seen during training (e.g. the test dataset). The ability of a model to perform well on new data is known as generalization [20]. Methods used to monitor the generalization ability of a model are train/test split or k-fold cross-validation, which involve the splitting of the available dataset in seen and unseen data [20]. The dataset used for training is often known as the training data, and the dataset used to test a model's generalization ability is known as the test data. Adequately training a model, while at the same time support a good generalization ability, is a difficult process. The following two concepts are important in the assessment of a machine learning model:

- **Overfitting** occurs when a model learns the training dataset too well, resulting in a good performance on the training dataset but a poor performance on the test dataset. The gap between the training error and the test error is too large.
- **Underfitting** occurs when a model fails to learn a good mapping function for the training dataset, and subsequently has a poor performance on the test dataset.

Overfitting or underfitting can be prevented by altering the capacity of a model, which involves the ability of a model to learn a variety of functions [20]. Shallow models with a low number of neurons have a lower capacity and are prone to underfitting as it has difficulties learning the training dataset. Deep models with a high number of neurons have a large capacity and are thus prone to overfitting as the model can memorize properties that are specific only to the training dataset. In general, models with a large capacity are able to solve more complex task. As increasing the capacity of a model is relatively easy to achieve, by adding more layers and neurons, it is more common for overfitting to occur than underfitting when designing a model for a complex task. A set of strategies used to reduce overfitting is collectively known as regularization [20]. Some well known regularization techniques applied to deep learning models include among many others: dropout, batch normalization and early stopping [45].

3.2 Convolutional Neural Networks (CNN)

The Convolutional Neural Network (CNN) is a deep learning model architecture that has presented interesting results in various computer visual recognition tasks, such as in the processing and classification of images and videos [35]. CNNs are specifically designed to process data that is structured and consists out of multiple arrays: 1D for sequences and signals, 2D for images, and 3D for sequential images or videos [34]. As an example, a color image consists of three 2D arrays, composed of the pixel intensities in the red, green and blue (RGB) color channels. The basic principles of CNNs are the same as for other ANNs, such as the FFNN discussed in section 3.1; as the network consist of connected cells that are arranged in layers and utilize repetitive bias and weight updates to learn a certain mapping function. The key differences of CNNs compared to other ANNs are found in the architecture and performed operations, which gives the network the exceptional ability to extract and capture spatial patterns in 2D or 3D structured data [46]. The typical architecture of a CNN is organized as a series of stages. The first number of stages are composed of two types of layers performing specific operations: the convolutional layers and the pooling layers. The final stage is composed of a flattening layer and fully connected layers. An example of the structure of a CNN with all its stages and process from input to output is illustrated in fig. 3.3. The operation and working principle of each layer are discussed in the following subsections.

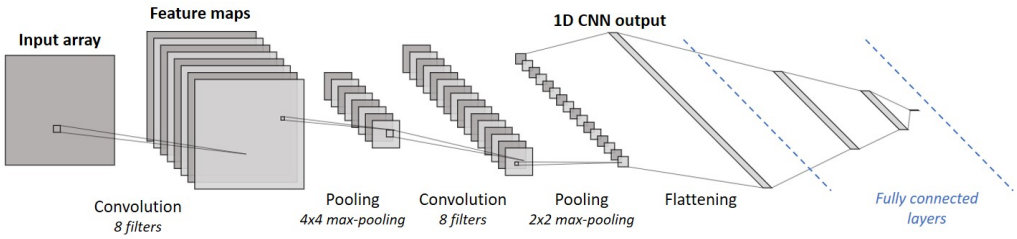


Figure 3.3: Illustration of a general CNN architecture with a fully connected layer on top. Overall, this network contains two convolutional layers and two pooling layers, with attached a flattening layer and three fully connected layers.

Convolutional layers

In the convolutional layers a multiplication is performed between a 2D array of weights and an array of input data, followed by summation. This operation is performed in order to detect features in the input data and capture these in so called feature maps [46]. The 2D arrays of weights are known as kernels or filters. The filter is often small in size compared to the input data and is moved with a sliding window over the input data, as visualized in fig. 3.4. This operation where the filter moves over the input allows for the detection of a feature, recognizable by the filter, anywhere in the input data and is commonly known as translation invariance [20]. The operation between the input data and the filter is a dot product, which is a element-wise multiplication followed by a summation. A feature map is the resulting output of the application of a filter after it has moved completely over an input array. Each single feature map in a layer is a result of a different filter, however all elements in a feature map share the same filter.

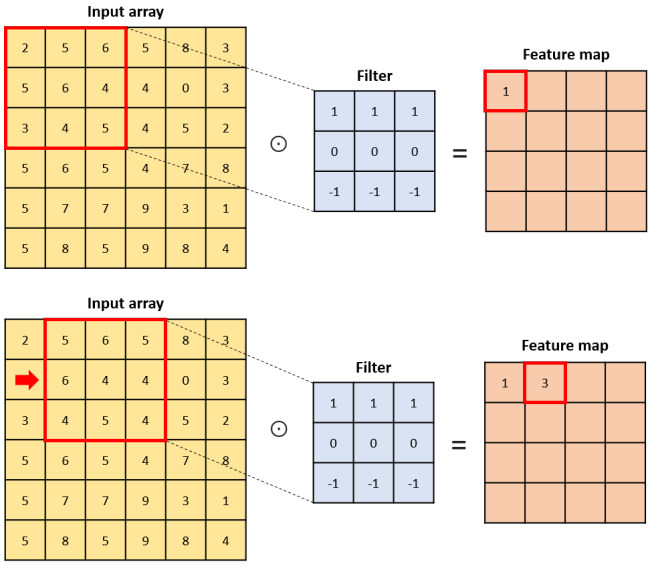


Figure 3.4: An illustration of the operations in a convolutional layer. Here the red block represents the sliding of the filter over the input array resulting in the step-by-step generation of the feature map.

There are several hyperparameters in the design of a convolutional layer, such as the number of filters, the size of the filter, the stride with which the filter moves over the input array, the weight initialization of the filters, the padding type, and the type of activation function applied to the feature maps. In the complete structure of the CNN several convolutional layers can be added, this is often done with the aim to extract higher level feature representations [20]. The model training processes for CNNs are based on the same concepts as seen for FFNN in section 3.1, however the formulas and algorithms are modified to accommodate the use of convolutions and the required neuron arrangements [46]. In the training process the filters of the convolutional layers, representing the model weights, are optimized to detect certain features required for the task of the model.

Pooling layers
In the pooling layers feature maps are down-sampled, reducing their dimensions, which aids a faster model training process and reduces the tendency of overfitting [46]. In the pooling layer a pooling unit typically computes the maximum or average of a small patch of the input feature map [47]. The pooling unit slides with a moving window over the feature map as illustrated in fig. 3.5, often avoiding overlap by matching the pooling unit size to the stride with which it moves. For example, in fig. 3.5 the spatial windows of the pooling unit has a size of 2 x 2 and a stride of 2. Several pooling layers can be added to the structure of the CNN, often repetitively alternated with a number of convolutional layers.

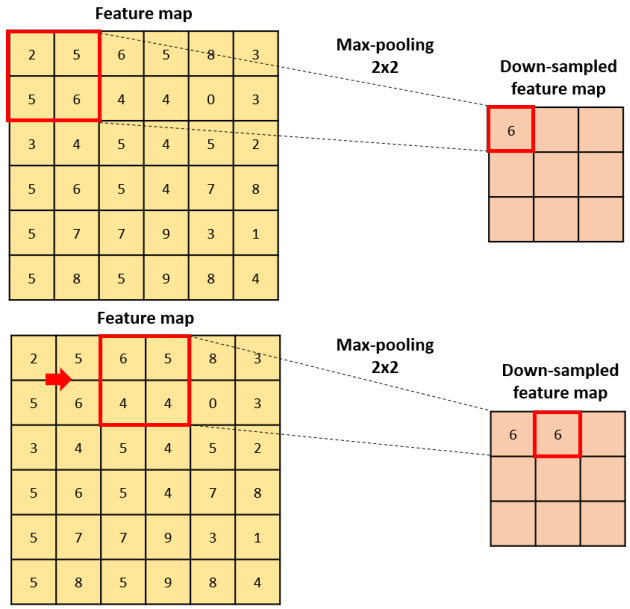


Figure 3.5: An illustration of the operations in a pooling layer. Here the red block represents the sliding of the pooling kernel over the input array resulting in the step-by-step generation of the down-sampled feature map.

Flattening and fully connected layers
The repetition of several convolution and pooling layers results in a 3D arrangement of neurons represented by a number of feature maps, as visible in fig. 3.3 to the right of the final pooling layer. This data needs to be flattened to produce a 1D vector output from the convolutional neural network. Subsequently, this output vector can act as input for a fully connected layer used in classification or regression tasks, or act as input for some other type of ANN.

3.3 Long-Short Term Memory Neural Networks (LSTM)

The LSTM neural network is part of the class of recurrent neural networks (RNN), a class often used to capture temporal dependency patterns and are applied for time series prediction tasks [35]. The RNN is an extension to the FFNN discussed in section 3.1 as it has the same structure of connected neurons, but also incorporates feedback loops resulting in the sharing of parameters and transformations over time. Generally, a FFNN maps from a fixed size input vector to fixed size output, whereas the RNN naturally operates on input sequences of variable length and map these to output sequences of variable length [48]. Hidden units in the architecture of RNN allow for the sharing of parameters and transformations over time, which is the mechanism that facilitates the building of a memory of long sequences and enable the network to learn progressively [48]. If one neuron is considered to be an information processing unit, then the feedback loop to that neuron provides additional information on the previous state of the neuron and thus information from the previous time step. Similar to FFNN, a RNN is commonly trained using a gradient-based optimization algorithm, in which the gradient of the loss function is obtained using backpropagation [48]. Applying the backpropagation method to RNN involves repeated multiplication of the connection matrix $\boldsymbol{W}$ and tanh activation function of the network, which leads to extreme vanishing and exploding of gradients [49]. More intuitively, it arises from the propagation of local errors in each time step, where similarly for our brain, it is simply impossible to remember every single event.

The LSTM network was introduced to (partially) mitigate the vanishing gradient problem occurring with RNN, and due to its success has become one of the most popular architectures for sequence prediction tasks [48]. Similar to a FFNN, the LSTM network consist of an input layer, an output layer and one or more hidden layers. The main feature of a LSTM network is contained in the hidden layers, which are composed of so called memory cells [50]. Four self-connected gates are introduced in the memory cells, which allows the cells to read, write and remove information from memory. This feature enables the memory cell to selectively hold or forget information, resulting in a more constant error flow [50]. Again in comparison to our human brain, the memory cell selectively remembers details and special events from the past, while not bothering to remember everything.

In order to explain the functioning of an LSTM memory cell, a single localized cell for time step t is depicted in fig. 3.6. Every gate in the memory cell has a different role; in fig. 3.6 the input (i_t), output (o_t), forget (f_t) and update (g_t) symbols represent the output values of each of the four gates. The gates receive input data of the current time step (x_t) and an input (h_{t-1}) obtained from the same LSTM memory cell's output in the previous time step. In fig. 3.6 these information flows are depicted by the red and green arrows for the input data (x_t) and (h_{t-1}) respectively. For each new information time step t, the new cell state (C_t) and output (h_t) are calculated for every memory cell using the following steps [46] [51]:

1. The output of the forget gate (f_t) is calculated using eq. (3.7), which quantifies what information is discarded from the the previous cell state (C_{t-1}). Here the (W_{xf}) and (W_{hf}) are the weight matrices of the input data and the previous cell state output data, whereas (b_f) represents the bias vector of the gate. The gate operates through a sigmoid activation function (σ), which scales all values in the range 0 (forget completely) to 1 (remember completely).

$$f_t = \sigma(W_{xf}\, x_t + W_{hf}\, h_{t-1} + b_f) \tag{3.7}$$

2. The input gate and update gate governs what information will be written to the new cell state (C_t), and are calculated using eq. (3.8) and eq. (3.9). The input gate operates through a sigmoid activation function (σ) and the update gate through a hyperbolic tangent ($tanh$). Here the (W_{xi}) and (W_{xg}) are the weight matrices of the input data and the (W_{hi}) and (W_{hg}) are the weight matrices of the previous cell state output data. The (b_i) and (b_g) represent the bias vector of the input gate and update gate.

$$i_t = \sigma(W_{xi}\, x_t + W_{hi}\, h_{t-1} + b_i) \tag{3.8}$$

$$g_t = tanh(W_{xg}\, x_t + W_{hg}\, h_{t-1} + b_g) \tag{3.9}$$

3. The previous cell state (C_{t-1}) is updated to the new cell state (C_t) using the forget, update and input gate as given in eq. (3.10). Here the output of the input gate and update gate interact through element-wise multiplication ($\odot$). Similarly, the output of the forget gate and the previous cell state interact through element-wise multiplication, after which both resulting vectors are added to produce the new cell state.

$$c_t = i_t \odot g_t + f_t \odot c_{t-1} \tag{3.10}$$

4. The output of the current memory cell (h_t) is calculated using element-wise multiplication of the new cell state and the output gate, as given in eq. (3.12). Here the new cell state is activated through the *tanh* activation function which bounds the output values, alternatively the ReLU activation function can used. The output gate vector is calculated using eq. (3.11), which implements a sigmoid activation function and governs how much information is passed through to the next layer or time step. Here the (W_{xo}) and (W_{ho}) are the weight matrices of the input data and the previous cell state output data, whereas (b_o) represents the bias vector of the gate.

$$o_t = \sigma(W_{xo}\, x_t + W_{ho}\, h_{t-1} + b_o) \tag{3.11}$$

$$h_t = o_t \odot tanh(c_t) \tag{3.12}$$

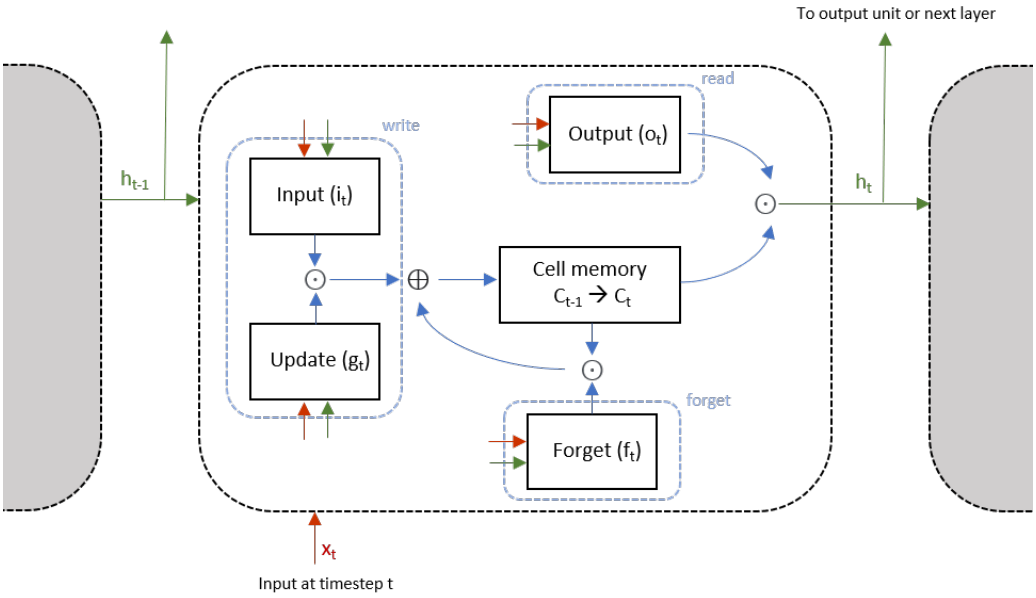


Figure 3.6: A single localized LSTM unit cell in the first layer for time step t, inspired by [52]. The red and green arrows depict the information flows of the input data (x_t) and (h_{t-1}) respectively. The symbols $\odot$ and $\oplus$ represent element-wise multiplication and summation operations respectively.

Similar to the way in which layers in FFNN are stacked in conventional deep neural networks, layers of LSTM networks can also be stacked. Stacked LSTM networks have multiple hidden LSTM layers, which are each comprised of numerous memory cells. In this way the LSTM network is both inherently deep in time due to its memory cell structure and has depth in space as a result of hierarchical processing due to the stacking of layers [53]. The additional hidden layers recombine the learned representation of previous layers and form new representations at higher levels of abstraction [53]. This can be more intuitively conceptualized as a processing pipeline, where each layer processes a small part of the problem and passes it on to the next layer, which in turn processes a different part of the problem while retaining the previous learned information. The outputs of the memory cells of the lower hidden layers acts as inputs for the upper hidden layers corresponding the same time step. A function description of stacked LSTM networks can be given as follows [51]:

$$\begin{aligned} h_t^1 &= f\left(h_{t-1}^1, x_t\right) \\ h_t^2 &= f\left(h_{t-1}^2, h_t^1\right) \\ &\dots \\ h_t^N &= f\left(h_{t-1}^N, h^{N-1}{}_t\right) \end{aligned} \tag{3.13}$$

In eq. (3.13) the outputs of layer 1, layer 2, till layer N are given by h_t^1, h_t^2, h_t^N at time step t respectively, and are the first part of the input stream of each respective subsequent layer. At the same time the second input stream is represented by h_{t-1}^1, h_{t-1}^2, h_{t-1}^N, which donates the outputs of layer 1, layer 2, till layer N at the previous time step $t-1$ respectively. The relation between the two input streams of the memory cell are described by function f, which represents the hidden layer functions of a memory cell as described by eq. (3.7) till eq. (3.12).

LSTM networks are specifically designed to work, learn and predict from sequence input data [48]. Its design supports both single-step and multi-step time series forecasts, of which the latter is more generally referred to as sequence-to-sequence predictions. Sequence-to-sequence models lies behind a multitude of applications that many people use on a daily basis, such as voice-enabled devices, Google Translate and online customer service chat-robots [54]. An LSTM models is able to map sequence input data directly to an output vector representing multiple output time steps or a sequence. Moreover, specialized architectures have been developed to more effectively use LSTM networks for advanced sequence-to-sequence prediction problems, such as a specialized encoder-decoder framework used for machine translation [55]. This specialized framework consist of two main

components: the encoder part reads the input sequences and produces a fixed-length vector which captures the temporal representation of the input sequence, and the decoder part which interprets the temporal representation and uses it to predict the output sequence [54]. A schematic representation of the encoder-decoder framework is given in fig. 3.7. The decoder part can be combined with a fully connected layer on top to predict a sequence target directly, or alternatively repeat the temporal representation and make a one-step prediction for each element in the targeted output sequence separately [56]. The use of an LSTM network in the decoder is an important feature of the encoder-decoder framework, as it allows the model to know what was predicted for the previous step in the sequence and accumulate some internal state on it while generating the complete output sequence [56]. In both the encoder and decoder several layers of LSTM cells can be stacked to form a deep network.

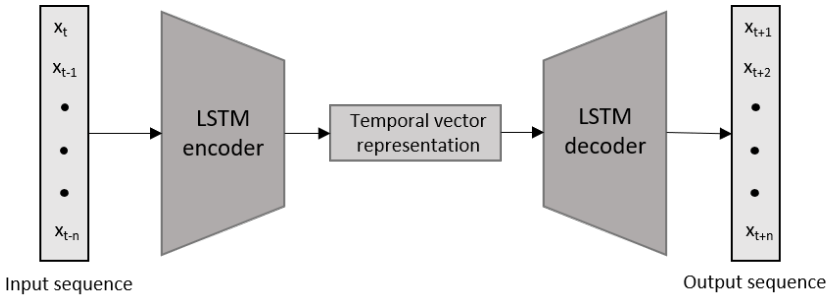


Figure 3.7: Schematic representation of an encoder-decoder framework for sequence-to-sequence prediction models.

Chapter 4: Methodology

In this chapter the methodology followed in the course of this research project is explained. Based on the problem statement and knowledge gab detailed in chapter 1, this study concentrates on the development of a novel forecasting framework for an aggregated solar PV power forecast over Germany with a temporal horizon of 3 hours. This goal is further motivated by the advances made in the field of solar PV power forecasting as elaborated upon in chapter 2, especially more recently with the application of (deep) machine learning methods. Based on the study of deep learning methods discussed in chapter 3, two separate forecasting models are investigated which leverage on convolutional neural networks (CNN) and Long Short Term Memory (LSTM) networks to process various types of data input streams and subsequently generate a solar PV power forecast. In the first model, a LSTM network is developed which processes multivariate time series data consisting of the latest available solar PV power feed-in data and solar PV power prediction data from a standard forecast employed in the energy market. In the second model, a CNN-LSTM network is developed which takes as input stream the latest satellite image data and solar PV power prediction data from a standard forecast.

This chapter starts with an introduction of the available dataset used for the training, testing and validation of the models and the applied data pre-processing steps. Next, in section 4.2 the design considerations of a clear sky model are discussed, which is a critical component of the solar PV power forecast methodology and data pre-processing steps. In section 4.3 the architecture and design considerations of the two proposed models are explained in detail. Next, section 4.4 presents the methodology applied to evaluate the model prediction performance. Finally, in section 4.5 the hardware and software environment used for model development is described.

4.1 Dataset description

This study concentrates on the development of an aggregated solar PV power forecast over Germany, with more specifically a temporal horizon of 3 hours, a 15-minute resolution and which has a forecast frequency of 15-minutes. These characteristics are most suitable for short-term forecast applied in the intra-day energy market, as identified in section 2.4. As a consequence, all model input variables, otherwise known as features, must be available on this resolution and with a near-live data stream to employ the models in a real-time operational setting. The selection of input features vary between the two proposed models, with one leveraging on the latest available solar PV power feed-in data whereas the other utilizes the latest available satellite images. On the other hand, some input features are shared between the two proposed models, such as solar PV power prediction data from a standard forecast employed in the energy market. The preparation of the datasets and pre-processing steps used to generating all available model features are identical for the two proposed models. For this reason, each available model input feature, its underlying dataset and required pre-processing steps are first discussed in the following subsection.

4.1.1 Actual solar PV power feed-in data

In Germany four regional transmission systems operators (TSOs) are responsible for uninterrupted exchange of power in their governing region. Moreover, they ensure that the consumption and generation of all power production types, including solar PV, are well balanced at all times. The four TSOs that each control a region of the German grid are: 50Hertz, Tennet, Amprion and TransnetBW, as illustrated in fig. 4.1. Each TSO publishes a near-live projection of the total solar PV feed-in power in their responsible control region. As each control area includes many thousands of PV systems, it is practically impossible to monitor the live solar PV power output of each individual system with measuring instruments. Instead, the TSOs use extrapolation methods to make projections of the actual total solar PV feed-in power based on a number of reference systems. This method takes into account measured values of reference systems, geographic information and individual PV system orientation and properties [57]. The projections of the actual solar PV feed-in power are available with a 15-minute interval and are collectively published for the four control regions on the Entsoe Transparency Platform [58]. The average delay between the closing time of a 15-minute interval and the publication time of the PV feed-in projection of that interval on the Entsoe Transparency Platform is provided in table 4.1. The publication delay varies significantly per grid control area and an aggregated publication delay over Germany is governed by the slowest provider. In other words, the data for the total solar PV feed-in power in Germany

is available for live model usage with a 45-60 minute delay.

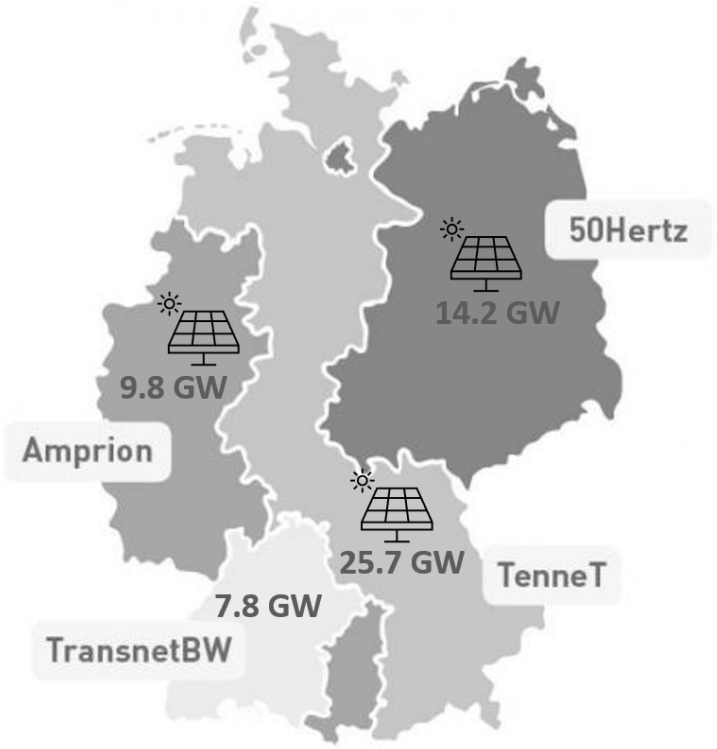
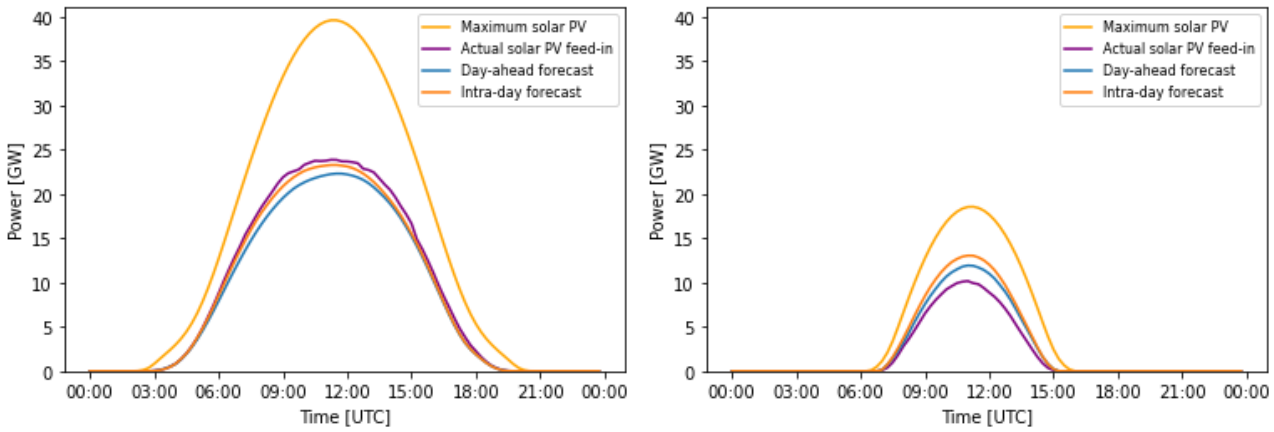


Figure 4.1: Map of TSO control regions in Germany and installed solar PV capacity as of December 2020 [59] [60].

Table 4.1: Average delay time in publication of solar PV feed-in per grid control area.

TSO	**Publication delay**
50Hertz	15 - 30 min
Amprion	45 - 60 min
TenneT	15 - 30 min
TransnetBW	0 - 15 min

A historic dataset of the solar PV feed-in power per grid control region and aggregated for Germany is available starting from 01-01-2015. As a pre-processing step, the actual solar PV feed-in is normalized using the maximum possible solar PV power generation at every instance, calculated by a clear sky model as described in section 4.2. The normalized actual solar PV feed-in power aggregated for Germany will be denoted as P_{actual}. The normalization is motivated by the non-stationarity of the solar PV feed-in data, whereas the normalized power is more stationary resulting in a lower effect of change in power output over the day [16]. Additionally, the use of normalized solar PV feed-in accommodates for the increasing installed solar PV capacity over the time range of the dataset. Finally, the use of normalized data limits the regression error during model training and increases model computational speed [8]. The characteristic daily time series for the solar PV feed-in power en maximum solar PV power from the clear sky model are shown for a summer and winter day in fig. 4.2a and fig. 4.2b respectively. These figures exemplify the significant changes in solar PV generation throughout the day and the seasons.



(a) Summer day (02-07-2019)

(b) Winter day (05-12-2019)

Figure 4.2: Actual solar PV feed-in power, maximum solar PV power and two standard solar PV forecast time series on a summer day (a) and winter day (b).

4.1.2 TSO solar PV power feed-in prediction data

A day-ahead and intra-day solar PV generation forecast is published by each of the four German TSOs, which themselves use for generation estimation and scheduling purposes [61]. Moreover, it can be seen as a standard for solar PV power forecast employed in the energy, as it is freely available to all market participants. The solar PV generation predictions in each grid control area are based on forecast services provided by private corporations, hence detailed information on the forecast methodology are unknown. In general, these are physical forecast models based PV system characteristics of individual installations and NWP model data (e.g. ECMWF, GFS) [62]. The forecasts are available with a 15-minute interval and are collectively published on the Entsoe Transparency Platform [58]. The day-ahead forecast is published before 18:00 Brussels time the day before actual delivery. The intra-day forecast is published before 08:00 Brussels time on the day of actual delivery. As a result of the publication time and the required initiation time of a live version of the models developed in this research, only the day-ahead forecast is available as a model input feature. As a pre-processing step, the day-ahead power forecast is normalized using the maximum solar PV generation calculated by the same clear sky model as applied to P_{actual}. The day-ahead solar PV power forecast data aggregated for Germany will be denoted as $P_{dayahead}$. The characteristic daily time series for the day-ahead and intra-day solar PV generation forecast are illustrated for a summer and winter day in fig. 4.2.

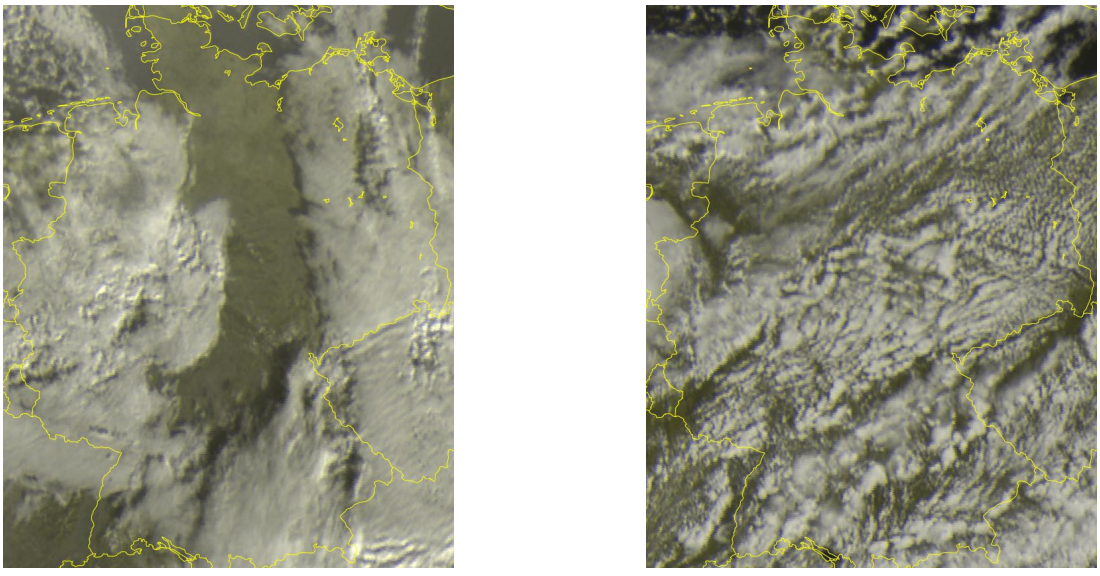
4.1.3 Satellite image data

The European geostationary meteorological satellites, Meteosat Second Generation (MSG), operated by the European Organization for the Exploitation of Meteorological Satellites (EUMETSAT) provide images of the hemisphere covering Europe, Africa and the Atlantic. The 12-channel imager, named Spinning Enhanced Visible and InfraRed Imager (SEVIRI), on-board MSG provides the data for several earth observation products with a repeat cycle of 15 minutes. The SEVIRI instrument has three channels in the solar spectrum at 0.6, 0.8 and 1.6 µm; and eight channels in the solar thermal infrared (IR); and a broadband high-resolution visible channel [63]. Several visualized and RGB composites products are based upon the observations from the SEVIRI instrument. An example is the Natural Color Enhance product using the three channels in the solar spectrum which provides a natural perception of the cloud cover. Although historic datasets for the visualized products and separate observation channels from SEVIRI are available via the EUMETSAT data portal, they lack a simple access method to commercially available live data. The access to live data is crucial to the deployment of any proposed model in an online production environment. Due to this requirement, high resolution visible satellite images are used which are published online by Weerslag [64]. These images are near live available (5-min delay), with a 15-minute interval and are based on the data products of the MSG satellite operated by EUMETSAT. A historic dataset of the satellite images published online by Weerslag is made available by Northpool.

The satellite images in the dataset have a size of 2000x1450 pixels, covering the Northwest region of Europe with a one-kilometer spatial resolution. The dataset provides images with a 15-minute interval observed since 1 January 2015, however, images are sometimes missing for various reasons. In order to make the satellite images applicable for the goal of this research, several pre-processing steps are performed as explained in the following subsections.

Dataset cleansing

The high resolution visible satellite images making up the available dataset are published online by Weerslag and have been extracted using a scraper since 1 January 2015. For various reasons this operation has been imperfect resulting in missing data. The missing satellite images have a non-trivial impact on the model operation and prediction performance, as for every 15-minute time step an image is expected by the model. Moreover, it is important that the model receives satellite images that are in accordance with the previous and next image, instead of an arbitrary data fill. If all missing images were interpolated without any restrictions, the model would learn false information on intervals where a simple interpolation would not be adequate, such as with large data gaps or during the transition from night to day and vice versa. A 3-step heuristic method is applied to build a complete and representative dataset. As a first step, missing images before sunrise and after sunset were filled with blank images, similar to all other night time images. As a second step, a nearest neighbor interpolation method in time is applied when six or less consecutive images were missing. For example, when the satellite images at 13:15 and 13:30 are missing, they are filled by the images from timestamp 13:00 and 13:45 of the same day respectively. It is assumed that the relative change from one image to the next with a data gap up to six time steps is sufficiently small that this will not lead to training inconsistencies. As a final step, when images are missing within 30-minutes from sunrise or the data gap is larger than six time steps, new images are manually added from the online Sat24 historic satellite image database [65]. This database contains similar high resolution visible satellite images based on the same data products of the MSG satellite as published by Weerslag. The satellite images from the Sat24 historic database are available with a 60-minute interval. First the missing hourly timestamps are filled with the images from Sat24, subsequently the intra-hour data gaps are filled using nearest neighbor interpolation in time. For example, when images are missing from 11:45 till 14:15, satellite images from Sat24 are copied for timestamps 12:00, 13:00 and 14:00 after which the intra-hour 15-minute gaps are filled using the previous described nearest neighbor interpolation method.



(a) 03-01-2019 08:45 UTC *(b) 06-05-2019 09:00 UTC*

Figure 4.3: Two dataset samples of high resolution visible satellite images over Germany.

Cropping and re-sampling

The original satellite images in the dataset have a size of 2000x1450 pixels, covering the Northwest region of Europe. As a pre-processing step the images are cropped over the maximum geographical width and length of Germany, as visible in fig. 4.3b where the yellow lines represent country borders. The resulting images have a size of 680 x 850 pixels with a one-kilometer resolution. This pre-processing step is motivated by the aim to extract as much relevant cloud information over Germany, while limiting the number of pixels outside the

borders of Germany. As a second step, the images are down-sampled to reduce the total size of the dataset, required by limited DRAM capacity during model training. A bilinear re-sampling method is used to reduce the image size to 32 x 40 pixels, maintaining the original aspect ratio of the cropped image over Germany. The resulting satellite images have a spatial resolution of 21 kilometer per pixel. Several other re-sampling resolutions were evaluated during model training, such as: 32x32, 64x64, 64x80 and 128x128, however these resulted in a lower model prediction accuracy or similar accuracy with significantly increased training time. As an additional pre-processing step the three RGB channels making up the images are normalized using the maximum value 255 of each channel. As an alternative pre-processing step the images were converted to grey-scale; however during the development of the model it was found to have reduced prediction accuracy. Furthermore, no data augmentation such as image rotation or translation has been applied. Although this is a common pre-processing step to increase the size of the available dataset, it is not justified for this case as the distribution of PV systems over Germany is not uniform.

4.1.4 Day-of-year and Quarter-of-day

The Earth's tilted axis causes the times of sunrise and sunset to vary over the year, subsequently influencing the length of the period in which solar PV power is generated. In order to add context to the model on seasonal and day-to-day patterns, two additional input features are introduced: the day-of-year (DOY) and the quarter-of-day (QOD). The latter refers to 96 periods of 15-minutes in a day, where the period 00:00-00:15 is quarter 1 and period 23:45-00:00 is quarter 96. As a pre-processing step the DOY and QOD are normalized using a cosine function, as described in eq. (4.1) and eq. (4.2) respectively. The use of this normalization technique aids model training and use of the cosine function specifically results in a smooth numerical transition from day-to-day and year-to-year.

$$DOY_{norm} = \frac{\cos\left(2\pi \, \frac{DOY}{365}\right) + 1}{2} \tag{4.1}$$

$$QOD_{norm} = \frac{\cos\left(2\pi \, \frac{QOD}{96}\right) + 1}{2} \tag{4.2}$$

4.1.5 Data split

The complete dataset is split in a train, test and validation dataset with a 66%, 17% and 17% distribution respectively. The years 2015-2018 are used for training, the complete year 2019 is used for validation and the year 2020 is used as final test set. The number of samples and range of timestamps per dataset are presented in table 4.2. The validation dataset is held back from training and is used to quantify the prediction accuracy of the model while tuning its architecture and hyperparameters. The final test set is held back completely from training and model tuning, in order to present an unbiased estimate of the prediction accuracy of the final model. Data splits with different time ranges were tested, however the split by calendar year shows the best train-validation set representation and overcomes seasonal patterns.

Table 4.2: Overview of dataset split.

Dataset	**Timestamp range**	**Samples**
Train	01-01-2015 00:00:00 - 31-12-2018 23:45:00	140256
Validation	01-01-2019 00:00:00 - 31-12-2019 23:45:00	35040
Test	01-01-2020 00:00:00 - 31-12-2020 23:45:00	35136

4.2 Clear Sky Model

An important data pre-processing step, as described in section 4.1, is the normalization of the actual solar PV feed-in power and the TSO day-ahead solar PV power forecast using a clear sky model. This model provides the maximum possible solar PV power generation at any given time. The choice of normalization via a clear sky model is motivated among other things by: increasing installed solar PV capacity, non-stationarity of solar PV power data and beneficial effects on the training process of deep learning models. The following subsections describe the background of clear sky models and the adopted methodology to calculate the maximum solar PV power generation in Germany at any given time.

4.2.1 Clear sky model methodology

As introduced in chapter 2, the presence of clouds is of influence on the solar irradiance and presents difficulties in irradiance forecasts. It is however possible to approximate the solar irradiance under the absence of clouds, that is, under clear sky conditions. The approximation of the amount of terrestrial solar irradiance on a clear sky day is a function of site location, altitude, solar elevation angle, water vapor, aerosol concentration and multiple other atmospheric conditions [9]. Clear sky models are generally classified into physical and empirical models [9]. Physical clear sky models are based on radiative transfer models which simulate the attenuation of solar irradiance in various layers of the atmosphere. These models offer a detailed analysis of the atmospheric processes, however they come at a high computational cost and a large number of input requirements [14]. On the contrary, empirical models are based on simplified parameterizations of a selection of atmospheric attenuation processes, resulting in an analytical expression to approximate the clear sky irradiance using a set of atmospheric parameters as input. Empirical models are most often used for general applications, due to the simplicity of the analytical equations, low computational cost and lower amount of input variables [9]. A large number of clear sky models are described in literature, which differ mainly from one another in the required model inputs. The most simple models are based only on the solar zenith angle. The more complicated models use many more additional atmospheric parameters, such as precipitable water and aerosols concentration to more accurately model the atmospheric attenuation processes.

Atmospheric effects have several impacts on the solar irradiance before it reaches the Earth's surface. On a clear sky day at noon, approximately 25% of the extraterrestrial solar irradiance from the sun is absorbed, scattered or reflected by the atmosphere [66]. When the sun is lower on the horizon, the attenuation of the solar irradiance by the atmosphere increases due to the longer traveling path. The solar radiation which comes directly from the sun is known as direct irradiance or the direct beam component. This direct normal irradiance (DNI) is measured by the flux of this component through a plane perpendicular to the direction of the sun. As part of the attenuation by the atmosphere, sunlight is scattered in all directions of which a part is redirected towards the Earth's surface as diffuse horizontal irradiance (DHI). The total solar irradiation is the sum of the direct beam and diffuse component received by a horizontal surface and is called the global horizontal irradiance (GHI). The GHI is a measure of the power of received sunlight (W/m^2), hence at any given location and time it can be utilized to model the maximum power output of a solar PV system under clear sky conditions.

The selection of a clear sky model for a specific location is driven by the quality and availability of input data, which is the primary limiting factor [67]. The data limiting the selection of a clear sky model for this project is the lack and quality of complete historic installed capacity and location data on all individual solar PV systems in Germany. However, the German Federal Network Agency [60] publishes monthly installed solar PV capacity data per postal code since 2015. Still this data per postal code is very limited, as no information is available on the location, elevation, solar panel orientation of all systems. Nonetheless, a representative average location of all solar PV systems per postal code can be estimated using the geographic midpoint of that code's area. By reason of the limiting quality of the data, a simple clear sky model is selected which is only a function of the solar zenith angle. The solar zenith angle is in it self only a function of location, so no atmospheric input parameters are required.

The solar irradiance is highly dependent on the position of the sun in the sky relative to the receiving area on the Earth's surface. Hence the clear sky model requires geometric inputs expressing the solar zenith angle throughout the year. At any day of the year (DOY) the solar zenith angle (z) can be calculated by subtracting the declination angle (δ) from site's latitude (ϕ):

$$z = \phi - \delta \tag{4.3}$$

$$\text{where } \delta = 23.45 \sin(x) \tag{4.4}$$

$$\text{with } x = \frac{360^\circ}{365^\circ}(DOY - 81) \tag{4.5}$$

The solar time is then calculated using the difference between the longitude of the site, the meridian of its timezone and the annual perturbations in the Earth's rotation rate around the sun:

$$Solar\,Time = Local\,Time + 4\,(standard\,Meridian - Local\,Meridian) + EoT \tag{4.6}$$

$$\text{where } EoT = 9.87 \sin(2x) - 7.53 \cos(x) - 1.5 \sin(x) \tag{4.7}$$

The angle between a line pointing directly towards the sun at solar noon and the line pointing directly to the sun at an arbitrary time is called the hour angle (ω). This is nothing more than an angular representation of the solar time, where one hour is represented by 15 degrees:

$$\omega = 15\,(Solar\,Time - 12) \tag{4.8}$$

Finally, the solar zenith angle can be calculated for any location, date and time using the site latitude, declination angle and hour angle:

$$\cos(z) = \cos(\phi)\,\cos(\delta)\,\cos(\omega) + \sin(\phi)\,\sin(\delta) \tag{4.9}$$

4.2.2 Regional clear sky power model

Motivated by the limited availability of PV system and atmospheric input variables the Meinel (1976) [68] clear sky model is selected to compute the direct solar radiation component. The Meinel model is presented in eq. (4.10). Here the attenuation of the extraterrestrial normal incident irradiance (I_0) through the atmosphere is described as a function of the relative airmass (AM). The Kasten and Young (1989) [69] model for relative airmass is selected for this purpose, which is only a function of the solar zenith angle (z). The extraterrestrial radiation is the radiation that reaches the Earth's outer atmosphere. The intensity of the radiation varies slightly throughout the year due to the eccentricity of the Earth's orbit around the sun, however for simplicity radiation can be considered constant at 1361 W/m^2 [70]. The DNI component of the clear sky model is calculated as follows:

$$DNI = I_0 \cdot 0.7^{\,AM^{\,0.678}} \tag{4.10}$$

$$\text{with } AM = \frac{1}{\cos(z) + 0.50572\,(96.07995 - z)^{\,-1.6354}} \tag{4.11}$$

$$\text{where } DNI = \begin{cases} 0 & (90 - z) \leq 0 \\ DNI & (90 - z) > 0 \end{cases} \tag{4.12}$$

The diffuse irradiance component is good for approximately 10% of the direct irradiance under clear sky conditions [70]. The clear sky models, including the DHI component, in literature are essentially empirical correlations based on measurements for a specific location. An available clear sky model from Northpool uses a modified version of the Meinel (1976) [68] DNI model for the DHI component, which is calibrated to fit clear sky days over Germany. During the tuning of the clear sky model it was found that the actual solar PV feed-in data contained values before and after the time of sunrise and sunset of the clear sky model on a quarterly interval. This is possibly caused by imperfections in the used representative average location of all solar PV systems per postal code, or caused by illumination of the Earth's surface even when the sun is below the horizon [71]. To overcome this problem a correction parameter (fTW) is added to the clear sky model near sunrise and sun set. In the early stages of model development it was found that the addition of this correction parameter improved model performance, even though it reduced the stationarity of the normalized actual solar PV feed-in data. The DHI component of the clear sky model is calculated as follows:

$$DHI = 0.14285\ I_0 \cdot 0.7^{\ AM_{tw}\ ^{0.678}} \tag{4.13}$$

$$\text{with}\ \ AM_{tw} = \frac{1}{\cos\left(z + fTW\right) + 0.50572\ \left(96.07995 - \left(z + fTW\right)\right)^{\ -1.6354}} \tag{4.14}$$

$$\text{where}\ \ DHI = \begin{cases} 0 & (90 - z) \leq fTW \\ DHI & (90 - z) > fTW \end{cases} \tag{4.15}$$

The mounting orientation of a solar module has implications on the amount of direct irradiance perceived by that module. The orientation of a module when mounted on a horizontal plane is described by the altitude (a_M) and the azimuth (A_M) of the module surface normal. Here the altitude of the module normal is related to the tilt angle by $a_M = 90° - \theta_M$. The position of the direct beam component of the sun is described by the solar azimuth (A_s) and the solar altitude (a_s). The direct irradiance on a module (G_m^{dir}) is related to the DNI via the angle between the incident direction of the sunlight and the module surface normal, as expressed in eq. (4.16). Unfortunately, no information is available on the orientation of each solar PV system per postal code in Germany. Therefore, a normal distribution is assumed for the module azimuth and tilt angle to represent the average orientation of all solar modules in Germany. For the module azimuth a normal distribution is used with a mean of 180° and a standard deviation of 30°, where $90 \leq A_M \leq 270$. The tilt angle is described by a normal distribution with a mean of 25° and a standard deviation of 5°, where $0 \leq \theta_M \leq 90$. The diffuse irradiance received on a module (G_m^{dif}) is proportional to the module tilt angle via the sky view factor (SVF). For simplicity the SVF is considered to be equal to one, resulting in $G_m^{dif} = DHI$. As a final step, the G_m^{dir} and G_m^{dif} are added to obtain the average irradiance on a PV module per time step and postal code, as in eq. (4.17). Together with linearly interpolated installed capacity data per postal code and the G_m from the clear sky irradiance model, a aggregate clear sky power model is created for Germany with a 15-minute interval. The complete clear sky power forecasting scheme as described in this section is shown in fig. 4.4.

$$G_m^{dir} = DNI\ \left[\sin\left(\theta_M\right)\ \cos\left(a_s\right)\ \cos\left(A_M - A_S\right) + \cos\left(\theta_M\right)\ \sin\left(a_s\right)\right] \tag{4.16}$$

$$G_m = G_m^{dir} + G_m^{diff} \tag{4.17}$$

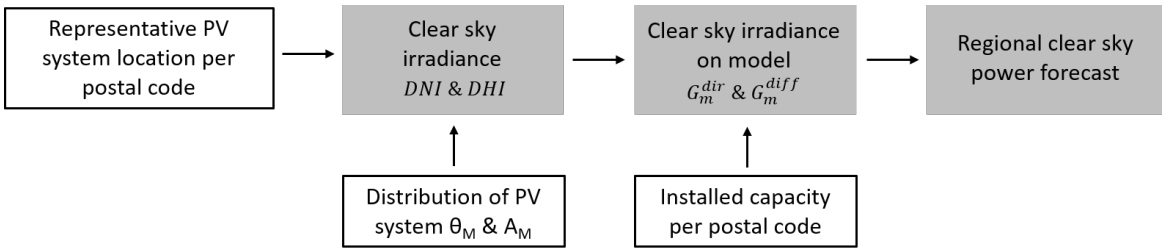


Figure 4.4: Schematic representation of the regional clear sky irradiance and maximum solar PV power forecasting scheme.

4.3 Model framework

As introduced at the beginning of this chapter, in this research project two individual solar PV forecasting models are developed. The models leverage convolutional neural networks (CNN) and Long Short Term Memory (LSTM) networks to process the various input data streams introduced in section 4.1. In the first model a LSTM network is developed which utilizes as input stream the P_{actual} and $P_{dayahead}$ data. In the second model a CNN-LSTM network is developed which takes as input stream the latest satellite image data and $P_{dayahead}$ data.

The development of two separate models models is motivated by the real-time availability of the data streams. The actual solar PV feed-in power for Germany is published with a 45-60 minute delay, whereas the satellite

images are available with only a 5-minute delay. On the target forecast horizon of 3 hours, the difference in data delay has a significant impact on the real-time usage of the forecast. The latest changes in cloud cover, impacting the total solar PV power generation, are picked up faster and translated into a forecast by a model with a shorter data delay. Moreover, the development of two separate models with the P_{actual} and satellite images as distinctive data streams, allows for a comparison between the two input features and their relevance to solar PV power forecast. In the following subsections the two proposed models and their design considerations are discussed in detail.

4.3.1 Model 1 - LSTM

The first model is designed while considering two data-streams: the latest available actual solar PV power feed-in data (P_{actual}) and the day-ahead solar PV power prediction data ($P_{dayahead}$) from Entsoe. The model is designed to generate a forecast with a temporal horizon of 3 hours, a 15-minute resolution and which has a forecast frequency of 15-minutes. The time t is considered to be the initiation time of a forecast. The input data matrix consisting of 12 time steps and 4 input features: P_{actual}, $P_{dayahead}$, DOY_{norm} and QOD_{norm} as illustrated in fig. 4.5. The P_{actual}, DOY_{norm} and QOD_{norm} are lagged values with respect to the initiation time t of the forecast; i.e. $(P_{actual,t-11}, \ldots, P_{actual,t})$. Whereas the input feature $P_{dayahead}$ consist of future values relative to initiation time t; i.e. $(P_{dayahead,t+1}, \ldots, P_{dayahead,t+12})$. In order to support the translation of future time steps of the $P_{dayahead}$ to the input sequence, as visualized in fig. 4.5, a symmetrical input-output sequence length is required. This however limits the optimization flexibility of the number of input feature time steps, as it is now dictated by the output sequence length. The output sequence consist of future values of the solar PV generation, that is during model training $P_{actual,t+1}, \ldots, P_{actual,t+12}$. Hence, both the output and input sequence length is equal to 12 time steps. For every new forecast the models moves with a sliding window of 15-minutes or 1 step over the dataset.

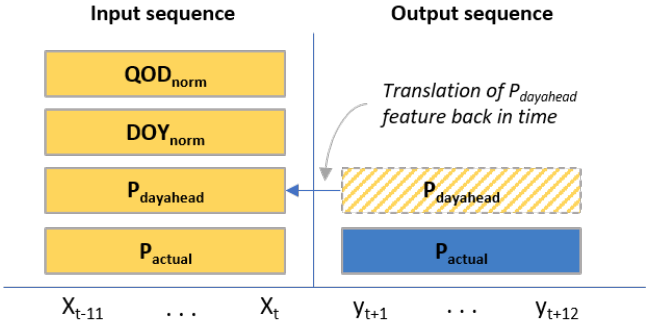


Figure 4.5: Schematic representation of the length and temporal dependence of the model input and output features.

The architecture of model 1 consists out of an encoder-decoder LSTM network, as visualized in fig. 4.6. The encoder part compresses the information from the input matrix into a temporal vector representation, after which the decoder translates this to the desired target sequence. The specialized encoder-decoder has shown to be very effective for sequence-to-sequence prediction as explained in detail in section 3.3. The encoder consists of three stacked LSTM layers, with each 200 memory cells. In early model training experiments it is observed that deep LSTM networks, with multiple stacked layers and memory cells, perform better than shallow LSTM networks. The temporal vector representation is duplicated 12 times in a repeater layer, once for each output sequence time step. The copies of the temporal vector representation act as a two-dimensional input for the decoder part of the model. The decoder consists of three stacked LSTM layers, with each 200 memory cells. The last LSTM layer is connected to three time distributed fully-connected layers (FC-layers), which transforms the two-dimensional output of the last LSTM decoder layer in a 12 time step output sequence.

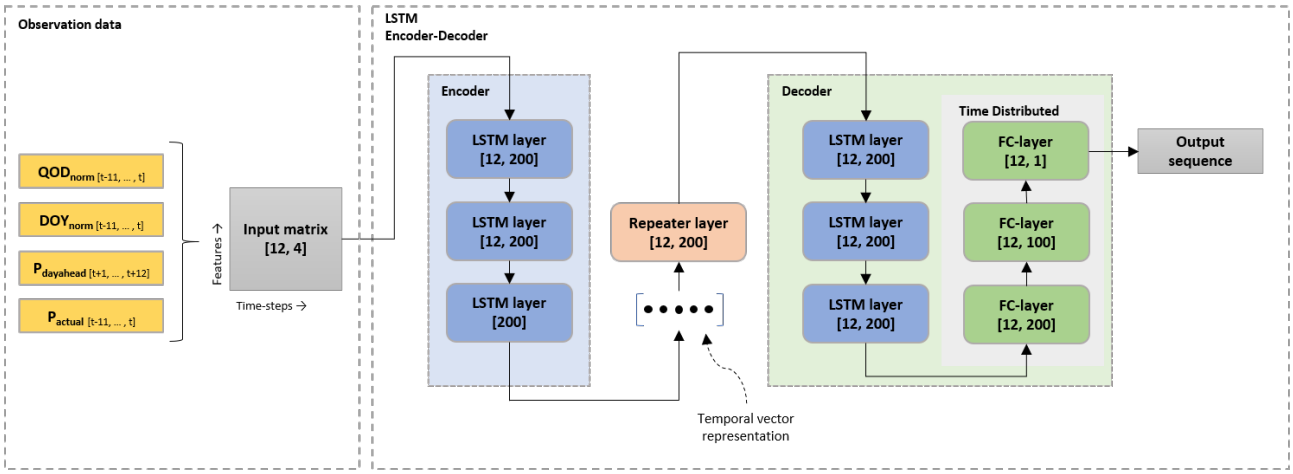


Figure 4.6: Overview of the model 1 LSTM Encoder-Decoder architecture. The values in brackets represent the output shape of each layer, which illustrates the data-flow in one forecast run.

During model development a number of decisions must be made on hyperparameters related to the model architecture and variables in the training process. The choice of hyperparameters can have a substantial effect on the performance of a model, however the setup of appropriate values is a complex task. For example, numerous configurations of cells and layers in the LSTM stack can be tested to find an optimal model performance. Underestimating the number of stacks and cells will give rise to an over simplified approximation, while overestimation will lead to increased generalization error and overfitting. The selection process of hyperparameter settings is often guided by a combination of trail-and-error, theory and hardware constraints [37]. Due to limited computational resources a disciplined grid search of the optimum set of hyperparameters is beyond the scope of this research project. However a large number of experiments throughout the model development phase have been performed to find appropriated values for a selection of hyperparameters. Based on these experiments and literature regarding hyperparameter selection, as discussed in section 3.1.2, the following decisions on hyperparameter selection are made:

- MSE is selected as loss function to quantify the training error in the model optimization process.
- Adam is used as optimization algorithm for model training.
- A learning rate schedule is adopted which lowers the learning rate by 50% for every 5th epoch and starts with an initial learning rate of 0.002.
- The mini-batch size is set to 96 samples.
- The ReLU activation function in combination with He normal initialization is applied to the fully-connected layers.
- The tanh activation function is used for the cell state and hidden state of the LSTM cells.
- The sigmoid activation function is used for the input, output and forget gate of the LSTM cells.
- The time distributed fully connected stack consists of 3 layers, in which the first layer consist of 200 neurons. The last fully-connected layer has 1 neuron, corresponding to one time step output. The layer in between consists of 100 neurons, equal to half the number of neurons of the first fully-connected layer.
- Both the model encoder and decoder LSTM stack consists out of 3 layers with 200 neurons each.
- The model is trained for 50 epochs.
- No dropout layers are applied as regularization technique, as it reduced model performance.

The model learning performance throughout the experiments are evaluated using model training and validation loss curves over experience in terms of epochs. These so called optimization learning curves are based on the MSE metric by which the model parameters in the training process are optimized, that is, are minimized. The learning curves can be used to diagnose if a model is overfitting, underfitting or is well-fit. Moreover, they can be used to diagnose if the validation or training datasets are a good representation of the model task domain.

The training learning curve is computed based on the training dataset and thus gives an impression on how well a model is learning. The validation learning curve is computed based on the validation dataset, so it is used to evaluate how well the model is generalizing. The training and validation learning curves for model 1 are presented in fig. 4.7, with on the y-axis the MSE loss on a logarithmic scale. It is observed that the model is well-fit as the training and validation loss decrease to a stable minimum with a only a small generalization gab between the loss values at the final epoch.

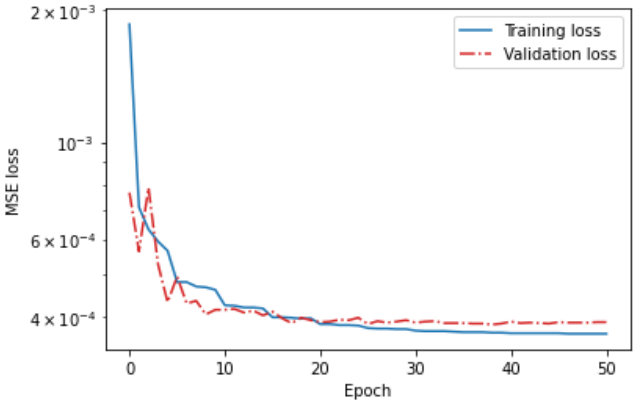


Figure 4.7: Training and validation MSE loss for model 1.

4.3.2 Model 2 - CNN-LSTM

The second model is designed while considering two data-streams: the latest available satellite images and the day-ahead solar PV power prediction data ($P_{dayahead}$). The model is designed for exactly the same purpose as model 1: a forecast with a temporal horizon of 3 hours, a 15-minute resolution and which has a forecast frequency of 15-minutes. The input data matrix consisting of 12 time steps and 4 input features: $P_{dayahead}$, DOY_{norm}, QOD_{norm} and the satellite images, of which the latter is a 3-dimensional data source. The proposed model is designed in a two step approach. First a CNN-LSTM model (model 2A) is designed and trained to process the satellite images into a power output ($P_{satellite}$) corresponding to the images in the input sequence. Next a LSTM model (model 2) is designed which generates a solar PV power forecast from the $P_{satellite}$ in combination with the three remaining input features. In several model design experiments this approach shows to outperform a model framework in which the model is not split and trained separately to process satellite images to a solar PV power output. In the following two subsections the design process of model 2A and its integration in model 2 are discussed separately.

Model 2A

The architecture of model 2A consists out of a CNN in combination with an encoder-decoder LSTM network, as visualized in fig. 4.8. The CNN is employed to capture spatial information on clouds and shaded areas from the pre-processed satellite images and encode the captured features in representation vectors, one for each image. An encoder-decoder LSTM is subsequently employed to capture the temporal relation between the representation vectors of the satellite images and provide a mapping function to the corresponding solar PV power generation output sequence. The CNN consist out of a repetition of four convolutional and max-pooling layers followed by a flatten layer. The CNN is employed in a time-distributed wrapper, which allows for the utilization and training of the same CNN for each satellite image in the 12 time step input sequence. The LSTM encoder and decoder consist both out of a three stacked LSTM layers with 200 memory cells each, which follows the same structure that has shown to be effective in design experiments for model 1. Several experiments have been performed to tune the hyperparameters of model 2A, with a special focus on the CNN architecture. Based on these experiments and literature regarding hyperparameter selection, the following decisions on hyperparameter selection are made:

- MSE is selected as loss function to quantify the training error in the model optimization process.
- Adam is used as optimization algorithm for model training.

- A learning rate schedule is adopted which lowers the learning rate by 50% for every 15th epoch and starts with an initial learning rate of 0.001.
- The mini-batch size is set to 96 samples.
- The ReLU activation function in combination with He normal initialization is applied to the fully-connected layers.
- The tanh activation function is used for the cell state and hidden state of the LSTM cells.
- The sigmoid activation function is used for the input, output and forget gate of the LSTM cells.
- The time distributed fully connected stack consists of 3 layers with 200, 100 and 1 neuron(s).
- The model encoder and decoder LSTM stack consists out of 3 layers with 200 neurons each.
- The CNN has four convolutional layers with 32 filters each, a kernel size of 3x3, zero padding, a ReLU activation function and He uniform kernel initiation.
- The CNN has four max-pooling layers with a kernel size of 2x2 with no (valid) padding.
- Batch normalization is not applied as regularization technique after each convolutional layer, as it showed to increase the validation error.
- The model is trained for 50 epochs.

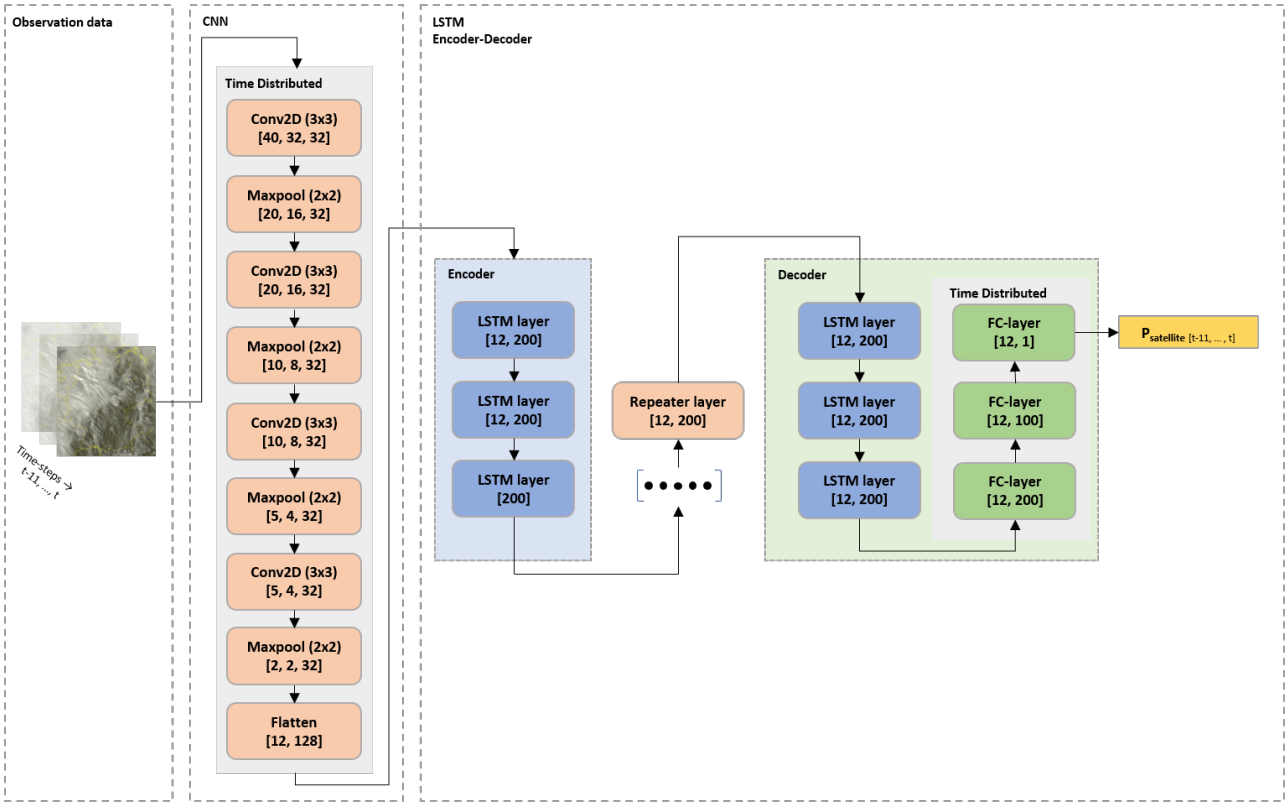


Figure 4.8: Overview of the model 2A, designed for the processing of satellite images to corresponding solar PV power output.

The learning performance throughout the numerous CNN-LSTM architecture experiments for model 2A are evaluated using model training and validation optimization learning curves. The training and validation learning curves for model 2A are presented in fig. 4.7, with on the y-axis the MSE loss on a logarithmic scale. It is observed that the training and validation loss significantly decreases over the number of epochs, indicating that the model is learning the mapping task of satellite images to solar PV power output ($P_{satellite}$). The validation loss goes to a stable minimum after about 35 epochs, while the training loss continuous to decrease. As the validation loss is not increasing and does not have an inflection point after 35 epochs, the plot of the

optimization curves does not shows the dynamics of overfitting. In this case the training loss continuous to decrease while the validation loss is stable, this situation can indicate an under-representative training dataset. This means that the training dataset does not provide sufficient information to completely learn the model task. This can occur when the training dataset has too little samples compared to the validation dataset. Due to the highly dynamic nature of clouds and weather condition, a large number of cloud cover settings can occur in the satellite images over Germany. Hence, it is possible that several characteristic cloud cover settings and their corresponding power output are unrepresented in the training dataset, while occurring in the validation dataset. However, it must be noted that the MSE loss is plotted on a logarithmic scale in fig. 4.7 and the gab between the training and validation loss is relatively small compared to the overall reduction in MSE loss over the 50 epochs.

Due to the possible under-representation of the training dataset a method called transfer learning is investigated in the course of this research project. Transfer learning is a method where an earlier developed and trained model is reused as a starting point for a new model aiming to tackle a related task [20]. This approach is often applied as the development and training of new models requires vast amount of computational and time resources, plus an extensive training dataset. A summary of the transfer learning technique and performed experiments is provided in appendix A. The experiments indicate that a self-trained model is superior to models based on transfer learning for this particular task.

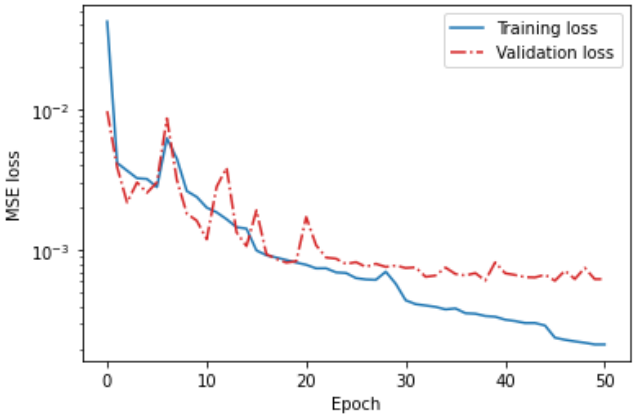


Figure 4.9: Training and validation MSE loss for model 2A.

Model 2

The architecture of model 2 integrates the pre-trained CNN-LSTM architecture of model 2A for the processing of satellite images to their corresponding solar PV power ($P_{satellite}$). Additionally an encoder-decoder LSTM network is added to generate the 12-step solar PV power forecast sequence, as visualized in fig. 4.10. In this model only the encoder-decoder LSTM architecture is trained with the purpose of mapping four input features $P_{satellite}$, $P_{dayahead}$, DOY_{norm} and QOD_{norm} to a solar PV power forecast. In this way, the architecture and input matrix of model 2 is the same as model 1, except that the actual solar PV power (P_{actual}) is replaced by ($P_{satellite}$) provided by model 2A. As a result the data and model delay for an online version is reduced from 45-60 minutes to 5 minutes. Similar to model 1, the LSTM encoder and decoder consist both out of a three stacked LSTM layers with 200 memory cells each. The hyperparameters of model 2 are kept similar to those of model 1. This is motivated by the use of the same model architecture and similar input features in relation to the hyperparameter optimization experiments performed for model 1. Moreover the similar setup allows for a like-for-like comparison to model 1 regarding the input feature P_{actual} versus $P_{satellite}$.

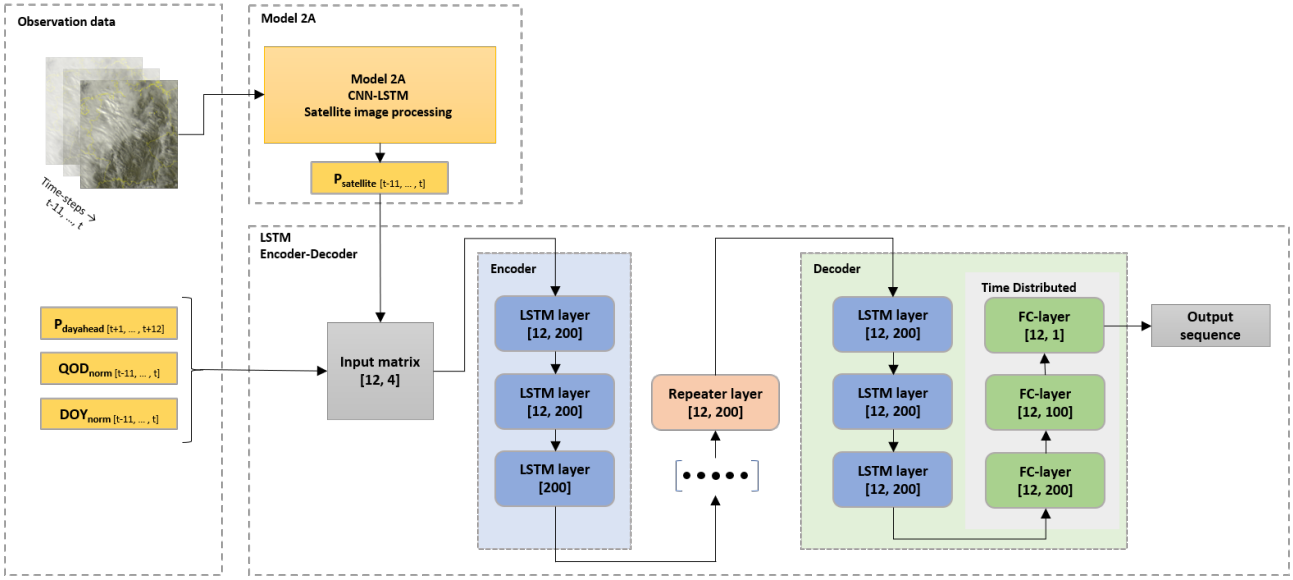


Figure 4.10: Overview of the model 2 with the integration of the pre-trained model 2A.

The training and validation learning curves for model 2A are presented in fig. 4.11, with on the y-axis the MSE loss on a logarithmic scale. It is observed that the training and validation loss significantly decreases over the number of epochs, indicating that the model is learning the mapping task of the input features, including $P_{satellite}$, to solar PV power forecast sequence. Moreover, it can be seen that the training and validation loss go to a stable minimum, with a relatively small generalization gab between the loss values at the final epoch.

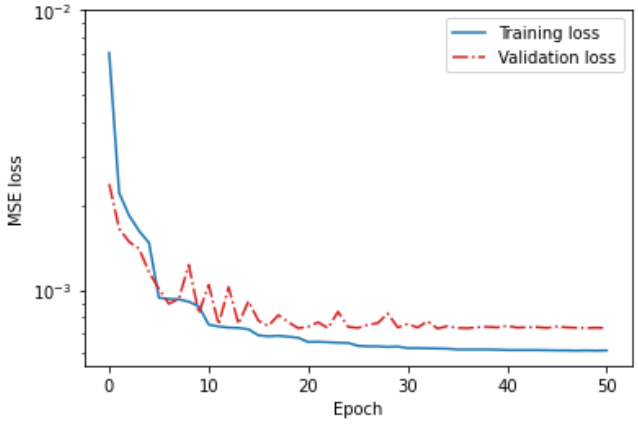


Figure 4.11: Training and validation MSE loss for model 2.

Due to the complex and ensemble architecture of model 2 several problems arise with providing input data to the model training process. The 4 input data sources required by model 2 consist of 1-dimensional sequence data and 3-dimensional image sequence data. Moreover, these data streams are required as input at different stages in the complete model. Finally, the satellite image training dataset consists out of 140256 sequences with 12 images each. If loaded and provided to the training phase directly, this would require much more DRAM memory than available. To overcome these problems a dedicated data generator is build to automatically transform the multivariate and multidimensional time series data sources into samples. These samples are subsequently provided in batches and separate data streams to the model in the training phase. The script of the custom built multivariate time series generator is available in appendix E.

4.4 Forecast performance evaluation metrics

The accuracy and performance of the proposed models are assessed using four metrics: mean absolute error (MAE), root mean square error (RMSE), mean bias error (MBE) and mean absolute percentage error (MAPE). The metrics enable the comparison between the two proposed models predictions and benchmark models predictions relative to the actual solar PV power generation. Each metric adds some information on a certain aspect of the accuracy of a model. In the MAPE metric two denominators are considered, the total solar PV power generation for the sMAPE and the total power generation of all production types for the tMAPE. The evaluation metrics are calculated as follows:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|P_{pred} - P_{true}| \tag{4.18}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(P_{pred} - P_{true})^2} \tag{4.19}$$

$$MBE = \frac{1}{N}\sum_{i=1}^{N}(P_{pred} - P_{true}) \tag{4.20}$$

$$sMAPE = \frac{100}{N}\sum_{i=1}^{N}\left|\frac{P_{pred} - P_{true}}{P_{true}}\right| \tag{4.21}$$

$$tMAPE = \frac{100}{N}\sum_{i=1}^{N}\left|\frac{P_{pred} - P_{true}}{P_{total}}\right| \tag{4.22}$$

where N is the number of data points in the evaluation set, P_{pred} is the predicted power, P_{true} is the actual solar PV feed-in power and P_{total} is the total power of all generation types. The MAE shows the average error between model predictions and actual values, making it suitable for the evaluation of uniform forecast errors [9]. The RMSE penalizes large errors more significantly. The MBE metric tells if a model overestimates or underestimates on average. The sMAPE and tMAPE are suitable to evaluate uniform forecast errors and is a useful tool for the comparison of model performance on different datasets, such as a regional forecast in comparison to an aggregated forecast for Germany. Moreover, the sMAPE and tMAPE provide a more intuitive indication of the impact of the forecast error on energy market, by showing the relative error percentage to the solar PV and total power generation.

Independent of the metrics applied to assess model performance, there are several factors influencing the comparison between models and the accuracy of a single model. The following factors and their appropriate solution are considered:

- **Day/night values:** In order to make fair comparison between the proposed models, benchmark models and other models from literature, it is important to clearly state the time range over which the model performance is evaluated. Two options can be considered, whether to evaluate both day and night values or only time steps in which the actual solar PV generation (P_{actual}) is larger than zero. Although the proposed models make forecast for all quarters of the day, the performance evaluation over forecast values where $P_{actual} > 0$ results in a better representation of the average error on the interval for which the forecast is important. The inclusion of forecast where $P_{actual} = 0$ would increase the model performance since the estimator at night is also very close to zero. Hence, in the model evaluation all forecast timestamps where P_{actual} is equal to zero are discarded. This approach is most often followed in literature when new solar PV power forecast methods are proposed [9].
- **Solar ramp:** Ramp events are caused by the variability of sunlight throughout the day, resulting in the characteristic bell shape solar PV power time series as visible in fig. 4.12. Due to the statistical nature of the proposed models, relying on past input data, the model performance may strongly vary throughout the

day. Moreover, the impact of the solar PV generation and the model forecast error at noon is larger and more important to energy trading than the generation just around sunrise and sunset. In order to better assess the model performance based on time-of-day and relative impact of the forecast error, the solar PV prediction errors are grouped in three events. A separation based on timestamp is not possible, as the time of sunrise and sunset varies throughout the seasons. Instead a separation is made using the maximum solar PV generation throughout the day. The first group is called the 'ramp up event' and contains all forecast timestamps for which the maximum solar PV generation is increasing and below 75% of its daily maximum. The second group is called the 'ramp down event' and contains all forecast timestamps for which the maximum solar PV generation is decreasing and below 75% of its daily maximum. The group containing all forecast timestamps for which the maximum solar PV generation is above 75% is called the 'peak event'. An illustration of this grouping mechanism is presented in fig. 4.12 for summer day on 02-07-2019. In the forecast results of the model test year 2020 the ramp up, peak and ramp down event represent 19.1%, 20.9% and 20.0% of the samples, respectively. The remaining 40% of the samples are forecast where the $P_{actual} = 0$.

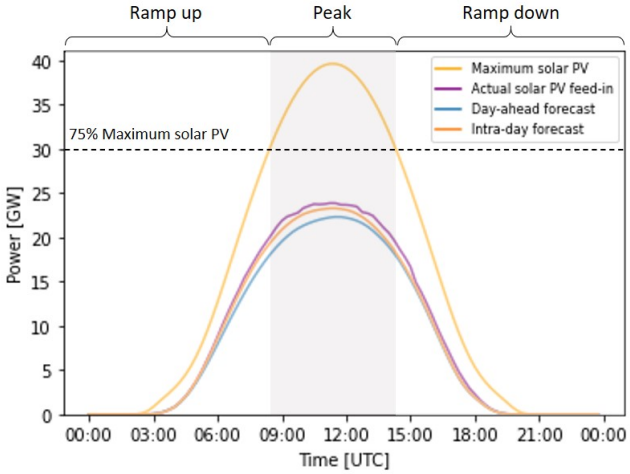


Figure 4.12: Illustration of the grouping mechanism used to separate the solar ramp in three events: ramp up, peak and ramp down.

In order to evaluate the performance of the two proposed forecast models, three benchmark models are considered: a persistence model, the Entsoe intra-day solar PV forecast and the Entsoe day-ahead forecast. A persistence model is an often used benchmark in literature on solar PV power forecasting. It is useful to compare the prediction accuracy of the proposed forecast model to the accuracy of any trivial model, such as a persistence model [10]. The underlying fundamentals of a persistence model makes it popular for very short-term to short-term forecast, similar to the temporal forecast domain of the proposed models in this research. A persistence model assumes no change in conditions from the instance the forecast is made (t) to the forecast timestamp (f_t). The designed persistence model for this research project assumes that the future normalized solar PV power at f_t will be the same as the normalized solar PV feed-in power at instance t, as described in eq. (4.23). The forecasted normalized power is subsequently scaled to the true solar PV power output using the maximum solar PV power from the clear sky model at f_t.

$$P_{norm}(f_t) = P_{norm}(t) \tag{4.23}$$

An intra-day and a day-ahead solar PV power forecast are published by the four Germany TSOs and are publicly available on the Entsoe transparency platform, as introduced in section 4.1.2. These forecasts act as a market standard and can significantly influence energy market trading when the projected solar PV feed-in deviate from the actual generation. As a consequence, these forecast are important benchmarks to evaluate the performance of the proposed models. An important difference of these forecast compared to the proposed models lies in the forecast methodology, update frequency and forecast horizon. The intra-day and day-ahead forecast are

physical forecast models based on PV system characteristics and NWP model data. The day-ahead forecast is published once every 24 hours, no later than 18:00 (Brussels time) the day before actual power delivery, and have a forecast horizon from 00:00 to 23:45 for the day of actual power delivery. The intra-day forecast is published once every 24 hours, before 08:00 (Brussels time) on the day of actual power deliver, and have a forecast horizon from the moment of publication to 23:45. In contrast, the proposed statistics based models, have an update frequency of 15-minutes and a forecast horizon of 3 hours.

In order to compare the forecasts of the proposed models and the benchmark models, which have different forecast frequency and horizon, an element wise comparison mechanism is adopted. A forecast result matrix is generated which stores all 12 time step sequences of the two proposed models. Next the actual solar PV feed-in and power forecast of the persistence, intra-day and day-ahead models are added for each individual time step in the forecast. In the computation of the performance metrics the errors are first element wise calculated and then averaged.

4.5 Experimental setup

An extensive model development phase and model comparison studies have been performed in this research project. The model code and analysis tools are written in the Python 3.8.5 programming language. Keras 2.4.3 with Tensorflow 2.4.0 backend open source deep learning libraries are used to build all the experimental deep learning models. Keras is an easy to use application programming interface (API) which supports the development and evaluation of models with a limited amount of programming. The Keras sequential API supports layer-by-layer model creation, which is sufficient for most problems but has a limited flexibility for models with shared layers or multiple input streams. The Keras functional API allows for the creation of more complex model structures and is therefor used for the development of model 2. Other packages used for comparative studies, pre-processing and visualization include: Numpy 1.19.2, Pandas 1.1.3, Sklearn 0.23.2 and Matplotlib 3.3.2. Neptune 0.9.2 is used as experiment management tool as it allows for systematical logging, tracking and comparing of experiments. All experiments are conducted on a virtual machine, whose configuration is Intel(R) Xeon(R) Gold 6152 CPU 2.10 Ghz with 12 processors and 48GB of memory. The main python scripts developed for model 1 and model 2 are available in appendix B and appendix C respectively.

The training time required by a model is dependent on hyperparameter settings and the size of the training dataset. However, it is possible to give an indication of the training time range experienced during model development. Model 1 has a training time of about 5 minutes per epoch on the available configuration. Model 2A and 2 are mainly governed by the time it takes to perform the convolutions in the CNN and has a training time of 20-30 minutes per epoch. More important to note is that the prediction process time of one 3-hour forecast takes less than a second. This makes the proposed models applicable for a real-time service, in which the forecast is run every 15-minutes using the latest available data.

Chapter 5: Results & Discussion

This chapter presents the results and evaluates the performance of the two proposed solar PV power forecasting models. Through a comparison of the proposed models to three benchmark forecasts, the model performance, effectiveness and characteristics are analyzed. In order to simplify the comparability between the two proposed models, it is assumed that all data is available in real time, and both models can be initiated at the same time for a 3-hour forecast horizon. The analysis is broken down in three sections in order to get a complete understanding of the performance and highlight important model characteristics. First, the global statistics of the two proposed are presented in section 5.1, in particular highlighting 5 aspects of the prediction results. Next, in section 5.2 three characteristic case studies are presented. Finally, in section 5.3 a regional forecast is presented to analyse the effect of spatial aggregation and geographic invariance.

5.1 Global statistics

After defining the model architecture, hyperparameter tuning and model training using the training and validation dataset as described in chapter 4, in this chapter the performance of the proposed model 1 and model 2 is evaluated against a year-long unseen dataset. The unseen test dataset ranges from 01-01-2020 00:00:00 to 31-12-2020 23:45:00 and contains 35136 multivariate sample sequences of 12 time steps. The average global accuracy of the two proposed models is evaluated using the performance metrics MAE, RMSE, sMAPE and tMape, which are used to compare the accuracy of the proposed models against three benchmark models as described in section 4.4. The analysis of accuracy results is divided in five subsections, in order to highlight important aspects of the model performance.

5.1.1 Overall performance

The overall performance results are presented in table 5.1. In terms of squared and absolute errors model 1 shows a substantial forecast performance improvement over the Entsoe day-ahead and intra-day forecast, with a relative MAE improvement of 53% and 43%, respectively. Moreover, the mean absolute percentage error of the model 1 forecast relative to the actual solar generation and total generation is significantly reduced to 8.4% and 0.3% respectively. Additionally the non-linear LSTM network of model 1 shows absolute superiority to a trivial persistence model. The predictive accuracy of the satellite image based model 2 is similar to that of the Entsoe intra-day forecast, and slightly better than day-ahead forecast.

Table 5.1: Comparison of the forecast accuracy of the benchmarks and proposed models using the performance metrics MAE, RMSE, sMAPE and tMAPE for the year 2020. The best performance is marked with bold font.

Model	**RMSE (MW)**	**MAE (MW)**	**sMAPE (%)**	**tMAPE (%)**
Persistence	2159	1333	59.6	1.5
Entsoe day-ahead	1008	672	15.7	0.7
Entsoe intra-day	840	558	13.2	0.6
Model 1	**507**	**316**	**8.4**	**0.3**
Model 2	812	547	11.6	0.6

5.1.2 Performance over the solar ramp

Next, the performance is evaluated while considering the period of the forecast horizon relative to the characteristic daily solar ramp. In table 5.2 the solar PV power prediction errors are grouped in three solar ramp events: ramp up, peak, ramp down. Model 1 shows superior prediction accuracy for all three groups and on all evaluation metrics. The performance of model 2 is slightly better than the Entsoe intra-day forecast only for the peak and ramp down event based on the RMSE and MAE, but better for all three groups based on the sMAPE. This observation can be explained by a different distribution in the error of model 2 relative to its location on the solar ramp, compared to the location of the error of the intra-day forecast. It can be observed

that the predictive accuracy of model 1 is particularly better for the peak and ramp down event compared to the Entsoe benchmark forecasts, whereas the accuracy in the ramp up event is more in accordance with the Entsoe intra-day forecast. This observation stresses the underlying weakness of a statistical forecast method, such as employed in model 1 and model 2, where the predictions are based on past observations. A large number of the predictions made in the ramp up event contain input feature values obtained before sunrise. For model 1 this means that the normalized actual solar PV feed-in input feature (P_{actual}) consist of zero values and is thus not a good proxy for the weather conditions and cloud-cover that it otherwise represents. Similarly for model 2, the satellite images before sunrise do not contain information in the visible band and thus do not provide a good estimator for the impact of cloud cover on the future solar PV generation. In order to boost the performance of models 1 and 2 in the solar ramp up event, the Entsoe day-ahead forecast ($P_{dayahead}$) was added to both models as input feature. This input feature improves both model's accuracy in the ramp up event to the values shown in table 5.2. However, this design decision might also negatively influence the model accuracy during other periods of the solar ramp, as the model's accuracy is now partly dictated by the inherited accuracy of the day-ahead forecast. Moreover, if the day-ahead forecast is inaccurate for a specific day during the solar ramp up, the models 1 and 2 also show a poor performance for the same period.

Table 5.2: Comparison of the proposed models and benchmark forecast accuracy split in the three characteristic solar ramp events. The best performance is marked with bold font and the RMESE, MAE and sMAPE in (MW), (MW) and (%), respectively.

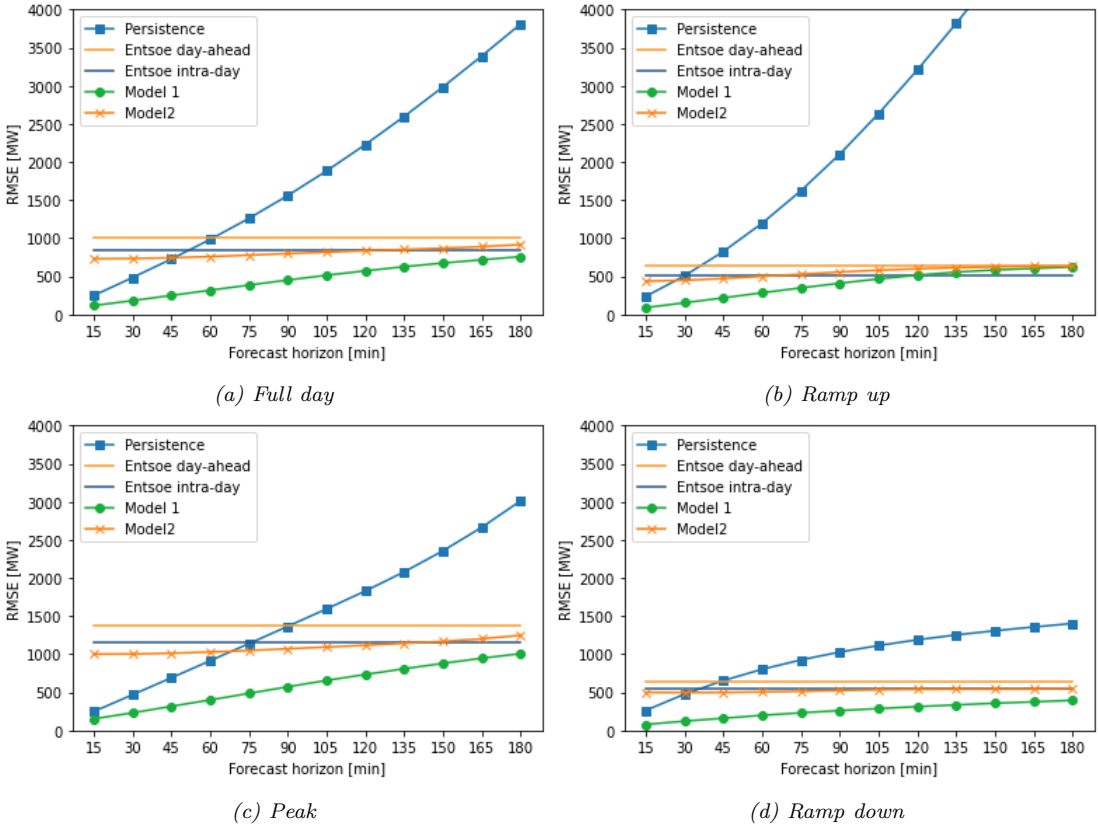
	Ramp up			**Peak**			**Ramp down**		
Model	**RMSE**	**MAE**	**sMAPE**	**RMSE**	**MAE**	**sMAPE**	**RMSE**	**MAE**	**sMAPE**
Persistence	3162	1986	58.9	1748	1175	9.5	1042	842	130.9
Entsoe day-ahead	645	418	16.6	1384	1056	9.2	636	408	24.0
Entsoe intra-day	507	321	13.0	1159	886	7.9	546	357	20.9
Model 1	**441**	**267**	**11.9**	**659**	**452**	**3.8**	**277**	**177**	**11.3**
Model 2	556	360	12.6	1098	829	7.0	524	354	17.2

5.1.3 Performance over the forecast horizon

A third important aspect to analyse is the performance of the proposed models over the forecast horizon and whether the performance is satisfactory across all prediction times. In particular, it is important to check from which forecast horizon the proposed statistical models equal the performance of the day-ahead and intra-day forecast of the TSOs. Figure 5.1 shows the squared prediction errors for each time step across the 3 hour forecast horizon. Here the RMSE accuracy of the Entsoe intra-day and day-ahead forecast are flat over the forecast horizon. This is because the forecasts are available on a daily interval and forecast horizon, resulting in a uniform error for every 15-minute initiation and 3-hour forecast horizon of the two proposed models in this research project. Model 1 and the persistence model show the best performance for the shortest forecast horizon of 15-minutes and degrade over longer horizons. The performance of the persistence model degrades sharply with increasing prediction horizon. These results are in accordance with the literature study, which indicated that persistence models prove only good results for forecast horizons shorter than 1 hour [10]. The prediction accuracy of the non-linear LSTM based model 1 degrades less sharp over the forecast horizon, and outperforms the Entsoe benchmarks on the 3-hour target horizon for both the solar peak and ramp down events. Due to the aforementioned statistical methodology it is observed that the performance of model 1 degrades faster for forecast in the solar ramp up and performs in line with the Entsoe intra-day forecast after 120-minutes.

As can be seen from table 5.2 and fig. 5.1 the prediction accuracy of model 2 is more in line with the Entsoe intra-day forecast and has a higher prediction error for short forecast horizons compared to model 1. The development of two individual solar PV forecasting models in this research project was motivated by the live availability of their respective data input streams. The actual solar PV feed-in power input feature of model 1 is available with a 45-60 minute delay, whereas the satellite images for model 2 are available with only a 5-minute delay. Due to this delay difference the first still relevant forecast point is 60-minutes after forecast initiation for model 1, compared to 5-minutes for model 2. Even if the prediction accuracy across the forecast horizon of model 1 is shifted backward by 60-minutes, the squared prediction errors of model 1 are still lower than those of model 2. Therefore, as an initial observation when comparing the average performance of the two proposed model with in addition the benchmark forecasts, a model based on actual generation data seems to be be the

superior choice for live solar PV power forecasting with a 3-hour horizon.

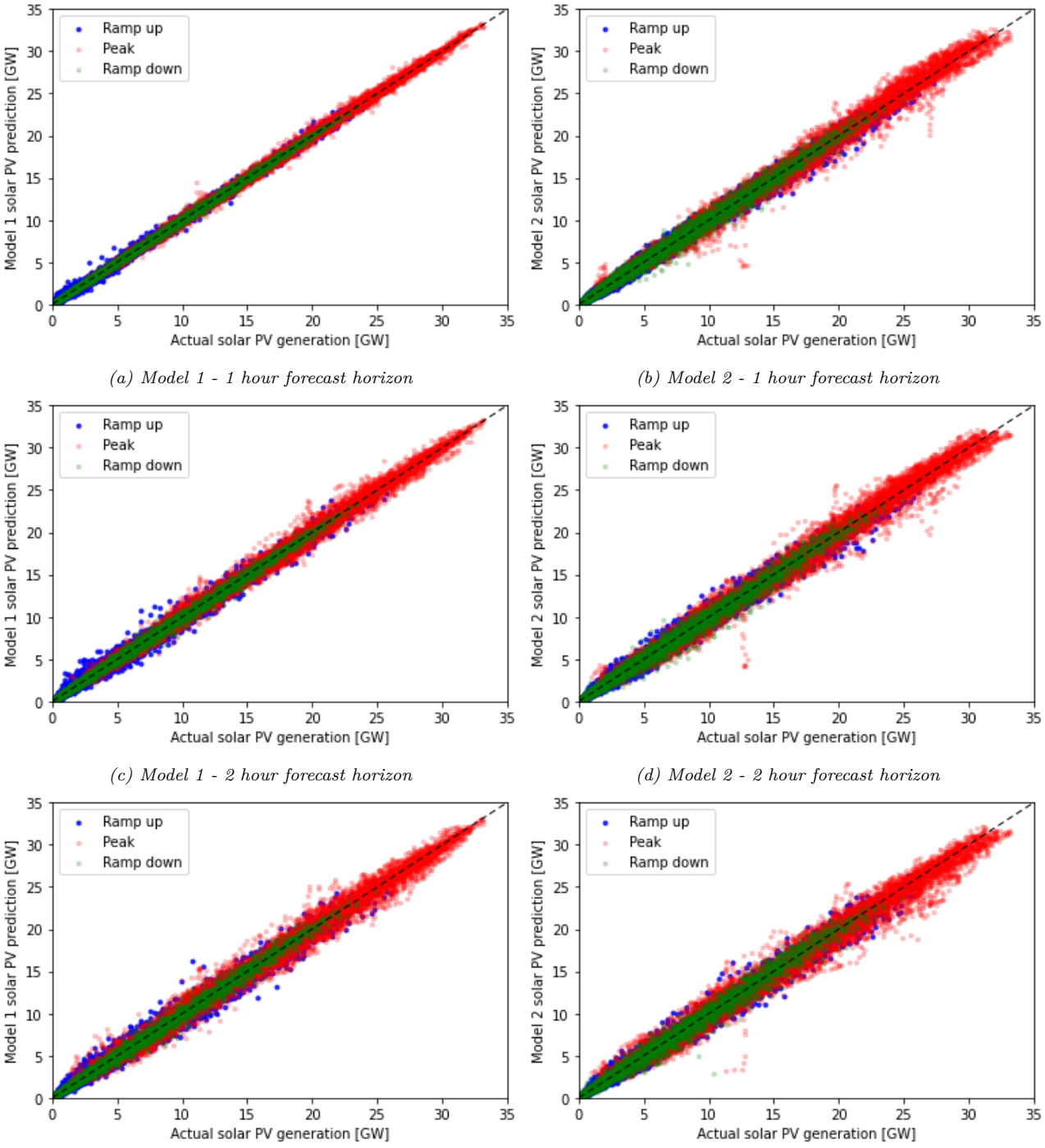


(a) Full day *(b) Ramp up*

(c) Peak *(d) Ramp down*

Figure 5.1: Comparison of the prediction accuracy of the models across the 3-hour forecast horizon, with a split for the full day, ramp up, peak and ramp down solar ramp events.

Figure 5.2 shows scatter plots for the solar PV power predictions of models 1 and 2 on a 1 hour, 2 hour and 3 hour forecast horizon versus the actual solar PV generation. It can be observed in these plots that the performance of model 1 is better than model 2 on all three forecast horizons, as it features a smaller spread. It is also observed that the predictions of model 2 show several extreme deviations, especially to an under estimation in the peak of the solar ramp. Most of the extreme outliers are traced back to 28 February 2020, for which the satellite image at 08:45 is blank and thus incorrect. This leads to a extreme reduction in the forecasted solar PV power output for 12 forecast runs, in which the incorrect image is in the input sequence. This demonstrates that it is crucial for the performance of model 2 that the provided dataset for training and prediction operations is errorless and consistent. Further investigation of the predictions in the outer band of the scatter plots spread of model 2 shows that these occur often under conditions of rapid cloud formation or under conditions featuring high spatio-temporal variation in the cloud cover. Several of these events are presented in section 5.2, where the accuracy of models 1 and 2 under those weather conditions are further analysed. It can also be observed from fig. 5.2 that the error spread is more uniform from low to high solar PV power generation for model 2, compared to model 1, where the spread decreases with higher solar PV generation. This is in accordance with the results of table 5.2, which shows that the sMAPE is lowest for solar PV power predictions in the peak of the solar ramp compared to the ramp up and down, and more so for model 1 compared to model 2.



(a) Model 1 - 1 hour forecast horizon

(b) Model 2 - 1 hour forecast horizon

(c) Model 1 - 2 hour forecast horizon

(d) Model 2 - 2 hour forecast horizon

(e) Model 1 - 3 hour forecast horizon

(f) Model 2 - 3 hour forecast horizon

Figure 5.2: Scatter plot of 1 hour, 2 hour and 3 hour forecast predictions of models 1 and 2, split by solar ramp event, compared to the actual solar PV generation.

5.1.4 Forecast bias

A further step to analyse is if the proposed models have a tendency to overestimate or underestimate the solar PV power generation. The spread of the scatter plots in fig. 5.2 indicate a possible negative bias, especially for model 2. In table 5.3 the mean bias error is presented for the solar ramp up, peak and ramp down event, plus a further split for a 0-1 hour, 1-2 hour and 2-3 hour forecast horizon. It can be observed that for both models 1 and 2 the bias turns from an underestimation for short forecast horizons to an overestimation for longer forecast horizons in the ramp up. Furthermore, both models 1 and 2 have a tendency to underestimate the solar PV

power generation in both the peak and ramp down of the solar ramp, and more so for shorter forecast horizons. However, it must be noted that these bias errors are relatively small compared to the RMSE, MAE and total solar generation.

Table 5.3: Comparison of the proposed models and benchmark forecast mean bias error in MW. The MBE score is split in the three characteristic solar ramp events, plus a further split for a 0-1 hour, 1-2 hour and 2-3 hour forecast horizon.

	Ramp up			Peak			Ramp down		
Model	**0-1h**	**1-2h**	**2-3h**	**0-1h**	**1-2h**	**2-3h**	**0-1h**	**1-2h**	**2-3h**
Persistence	481	1760	3543	-214	-441	-313	-446	-840	-1015
Entsoe day-ahead	-21	-21	-21	8	8	8	-34	-34	-34
Entsoe intra-day	-9	-9	-9	-34	-34	-34	-38	-38	-38
Model 1	-15	3	17	-49	-65	-61	-34	-46	-56
Model 2	-22	20	57	-104	-84	52	-118	-64	-50

5.1.5 Performance dependency on weather conditions

A next step in analysing the global statistics is to check for forecast dependencies on weather conditions. There are two commonly used parameters to classify weather conditions, the clearness index (k_t) and the clear sky index (k_{cs}) [9]. These parameters are obtained from the ratio of measured irradiance to modeled clear sky irradiance, with a difference in the latter used as normalization factor. As this research project is dedicated to direct power forecasting, no irradiance observation data is available. However, the normalized P_{actual}, obtained as a ratio of the actual solar PV feed-in power and the clear sky model solar PV power, can act as a proxy for the clear sky index. Figure 5.3 shows the MAE for the two proposed models and the Entsoe benchmark models over the P_{actual}, where the MAE is calculated for 10 equally sized bins. It is observed that all forecast show a prediction accuracy dependency on the P_{actual}. Clear sky conditions (P_{actual} near 1) and overcast cloud cover (P_{actual} near 0) show a better prediction accuracy compared to broken cloud cover (P_{actual} near 0.5). The weather conditions where P_{actual} is near 0.5 can feature strong spatial and temporal variations in the cloud cover. Here model 1 significantly outperforms the satellite image based model 2, which is less the case for overcast conditions. Interestingly, the predictions accuracy of model 2 is the lowest under clear sky conditions compared to the other models. A possible explanation for this is the overestimation of impact of small cloud features or scattered clouds on the total solar PV generation. A critical note must be made on the assumption of the P_{actual} being a good proxy for the clear sky index. Under this assumption the P_{actual} data should be completely stationary, which is not the case near sunrise and sunset. Here the P_{actual} is near zero, hence resulting in an overly small MAE in the lower P_{actual} domain of fig. 5.3. This characteristic complicates the analysis of the performance based on weather conditions using the normalized solar PV feed-in power. A further step in analysing the forecast dependencies on weather conditions is presented in the seasonal forecast analysis of section 5.1.6.

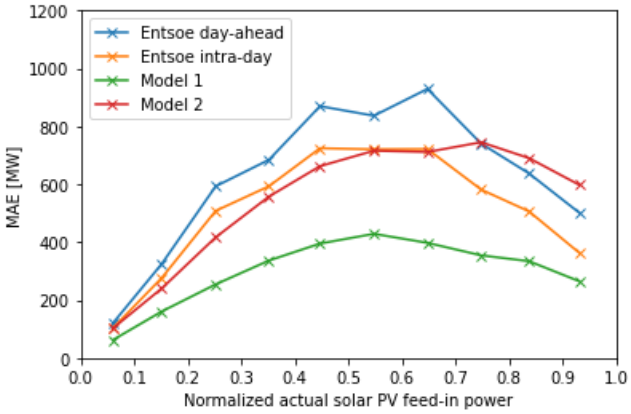


Figure 5.3: Dependency of the two proposed models and Entsoe intra-day and day-ahead models MAE on the normalized actual solar PV feed-in power, presented as a proxy for the clear sky index.

5.1.6 Seasonal effects

The final step in analyzing the proposed models performance is to validate whether the prediction accuracy is constant or dependent on seasonality. Since weather conditions and sun elevation change by seasons, these dependencies might influence the quality of the models. In order to analyze the seasonal performance, the test dataset is divided in four subsets by meteorological seasons. Specifically, the spring test set ranges from 1 March 2020 to 31 May 2020, the summer test set from 1 June 2020 to 31 August 2020, the autumn test set from 1 September 2020 to 30 November 2020, and the winter test set from 1 December 2020 to 31 December 2020 plus 1 January 2020 to 29 February 2020. The results of this analysis are summarized in table 5.4. In particular, analysing these results, it is clear that model 1 maintains its superior performance over all four seasons and all evaluation metrics relative to model 2 and the benchmark forecasts. The model performance in autumn and especially winter shows a higher relative error to the total amount of solar generation, expressed by the sMAPE. However, according to the tMAPE, these errors have a lower impact on the total power generation compared to summer. This can be easily expected, as the total daily solar generation is less in winter. Based on the RMSE and MAE there is no strong impact of seasonal variability visible. Though, the squared and absolute errors of both models 1 and 2 are slightly higher in summer and spring compared to autumn and winter. A possible explanation for this phenomena is a more frequent occurrence of convective cloud formation in summer and spring. In the analysis of fig. 5.2 this is already identified as a possible source of larger errors, especially for model 2. Also interesting to observe from table 5.4 is that model 2 outperforms the Entsoe intra-day forecast more significantly in spring and winter, compared to summer and autumn.

Table 5.4: Comparison of the seasonal performance of the benchmark and two proposed models for the year 2020. The best performance is marked with bold font.

Season	Model	RMSE	MAE (MW)	sMAPE (%)	tMAPE (%)
Spring	Persistence	2560	1564	57.5	2.9
	Entsoe day-ahead	1137	758	10.9	1.3
	Entsoe intra-day	941	615	9.0	1.0
	Model 1	**552**	**340**	**5.7**	**0.6**
	Model 2	879	581	8.0	1.0
Summer	Persistence	2082	1306	53.2	2.4
	Entsoe day-ahead	975	644	12.5	1.1
	Entsoe intra-day	768	514	9.8	0.9
	Model 1	**512**	**328**	**7.3**	**0.6**
	Model 2	787	549	9.9	0.9
Autumn	Persistence	2097	1321	60.5	2.1
	Entsoe day-ahead	965	660	19.5	1.0
	Entsoe intra-day	799	548	16.1	0.8
	Model 1	**485**	**300**	**9.7**	**0.5**
	Model 2	799	559	13.1	0.9
Winter	Persistence	1617	1037	73.0	1.5
	Entsoe day-ahead	896	599	24.1	0.9
	Entsoe intra-day	845	562	22.1	0.8
	Model 1	**449**	**277**	**13.2**	**0.4**
	Model 2	761	475	18.7	0.7

5.2 Case studies

To further analyze the two proposed models and identify characteristic model behavior under certain weather events three case studies are presented. In the first case study a winter day with overcast cloud cover is presented. Next, in the second study an autumn day with rapidly changing broken cloud cover is discussed. Then, in the third study a day with convective cloud formation in spring is examined.

Case 1: Overcast winter day

The first case study presents a winter day with overcast cloud condition on 9 January 2020. A time series of models 1 and 2 forecast runs and corresponding satellite images are presented in section 5.2. The forecast initiations range from 07:00 to 11:30 and are presented for every half and full hour. It can be observed that the forecast of models 1 and 2 for 07:00 are in line with the Entsoe day-ahead forecast, as for this initiation the input variable P_{actual} and the satellite images provide no context on the latest cloud conditions. This underlines the observation that the accuracy of models 1 and 2 in the solar ramp up are strongly dependent on the correctness of the day-ahead forecast, as previously discussed in section 5.1. Starting from the 07:30 forecast and onward the P_{actual} and the satellite images provide information on the latest weather conditions, which is translated in a power forecast that is higher than the Entsoe day-ahead and intra-day forecast. Both models 1 and 2 seem to pick up the clearing of clouds in the southern part of Germany, visible in the satellite image from 08:00 and onward. In this case, model 2 provides a more accurate forecast of the solar PV power output throughout the day. The power forecast of model 1 is adjusted downward from the 10:00 run to the 10:30 run, possibly influenced by the slight dip in the P_{actual} visible in the solar ramp for this time range. After this dip in the solar ramp, the actual solar PV feed-in increases again more strongly, for which the forecast of model 1 adjusts accordingly. This observation stresses the dependency of model 1 on past input data. If changes in an input feature are only momentarily and not continuous, such as observed in this case study, it can negatively impact the forecast accuracy on the full 3-hour forecast horizon.

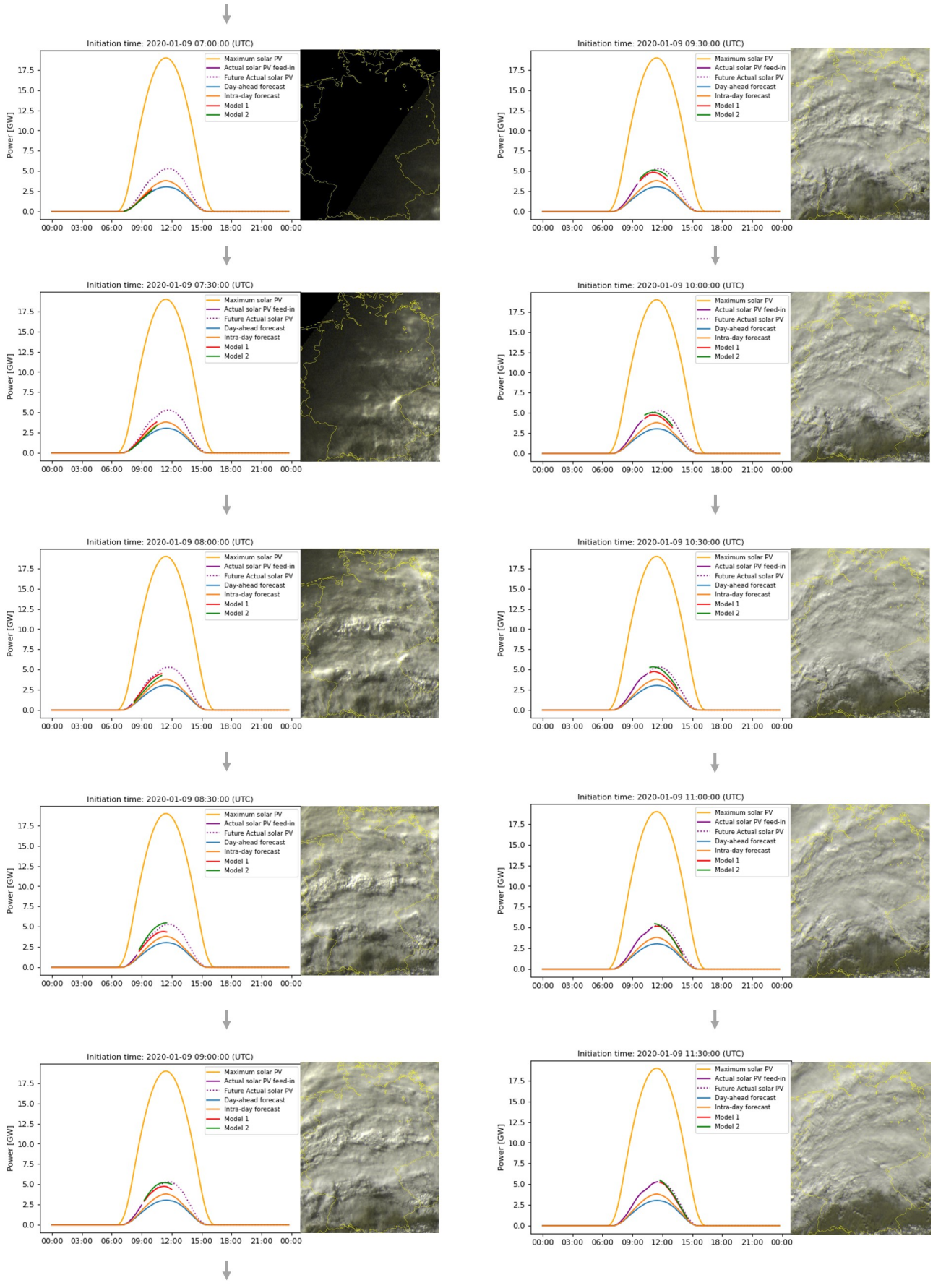


Figure 5.4: Time series of the forecasts made for case study 1 with accompanying satellite images. The first forecast subfigure is presented top left with the arrows indicating the chronological order.

Case 2: Autumn day with cloud formation
This case study presents a day with rapidly changing broken cloud cover on 6 September 2020. A forecast time series is presented in section 5.2, starting from forecast initiation 06:30 to 09:00 with a 30-minute temporal resolution and from 09:15 to 10:00 with a 15-minute resolution. The day starts with the break up of clouds in the north of Germany, resulting in a higher solar PV power output forecast by model 1 and model 2 compared to the Entsoe day-ahead and intraday forecast. Both models quite accurately predict the peak power generation at the 07:30 forecast, but overestimate the time for which this holds, as both models have no indication that conditions will change. In the satellite image of 09:00 in section 5.2 it can be observed that the cloud condition are changing. An increase in cloud cover starts over the northern and central part of Germany, possibly due to turbulent and convective processes. This results in a big update for the satellite image based model 2 at 09:15, strongly reducing the expected solar PV power output over the entire 3-hour horizon. Although this forecast is inaccurate, it indicates that model 2 picks up rapidly changing cloud conditions and translate this in the power forecast. The forecast of model 1 is decreasingly overestimating the solar PV power output for the runs from 09:00 to 09:45. This indicates that the adjustment to the rapidly changing conditions takes longer and is more gradual for this model. Still, also model 1 is able to pick up the changing conditions and adjust the forecast accordingly. This shows that in this case the P_{actual} is also good proxy for the latest cloud conditions and results in a more accurate forecast.

Case 3: Spring day with cloud formation
This case presents spring day with strong cloud formation over the central and eastern par of Germany on 3 May 2020. The forecast time series of this day is presented in section 5.2 starting from forecast initiation 07:30 to 10:00 with a 15-minute temporal resolution. Both model 1 and model 2 point to a slightly lower solar PV power output for the first four presented forecast runs compared to the Entsoe day-ahead and intraday forecast. In the satellite images starting from 08:00 the first increase in cloud cover can be observed, with a stronger increase from 09:00 and onward. The forecast of model 1 points to an increasingly lower power feed-in starting from the 8:30 run. The satellite image based model 2 forecast points to higher solar PV power feed-in and less decreasing compared to model 1. Furthermore it is observed that in the forecasts of 08:45 to 9:15 both models 1 and 2 have a lower power offset point (15-minutes after model initiation), but still expect an increase in solar PV power following the characteristic solar ramp shape. This expectation to have an increasing solar PV power output up to the time of peak maximum solar power is even stronger for model 2 compared to model 1. From this observation it can be questioned if normalized P_{actual} used for the training of both models is stationary enough and if this reduces the flexibility of the model to deviate from the characteristic solar ramp shape. Analyzing the results of this case study together with case 2, indicates that both models 1 and 2 have the ability to identify events with convective cloud formation and translate this in a forecast. However it also shows that both models do not have the ability to forecast these events before they onset.

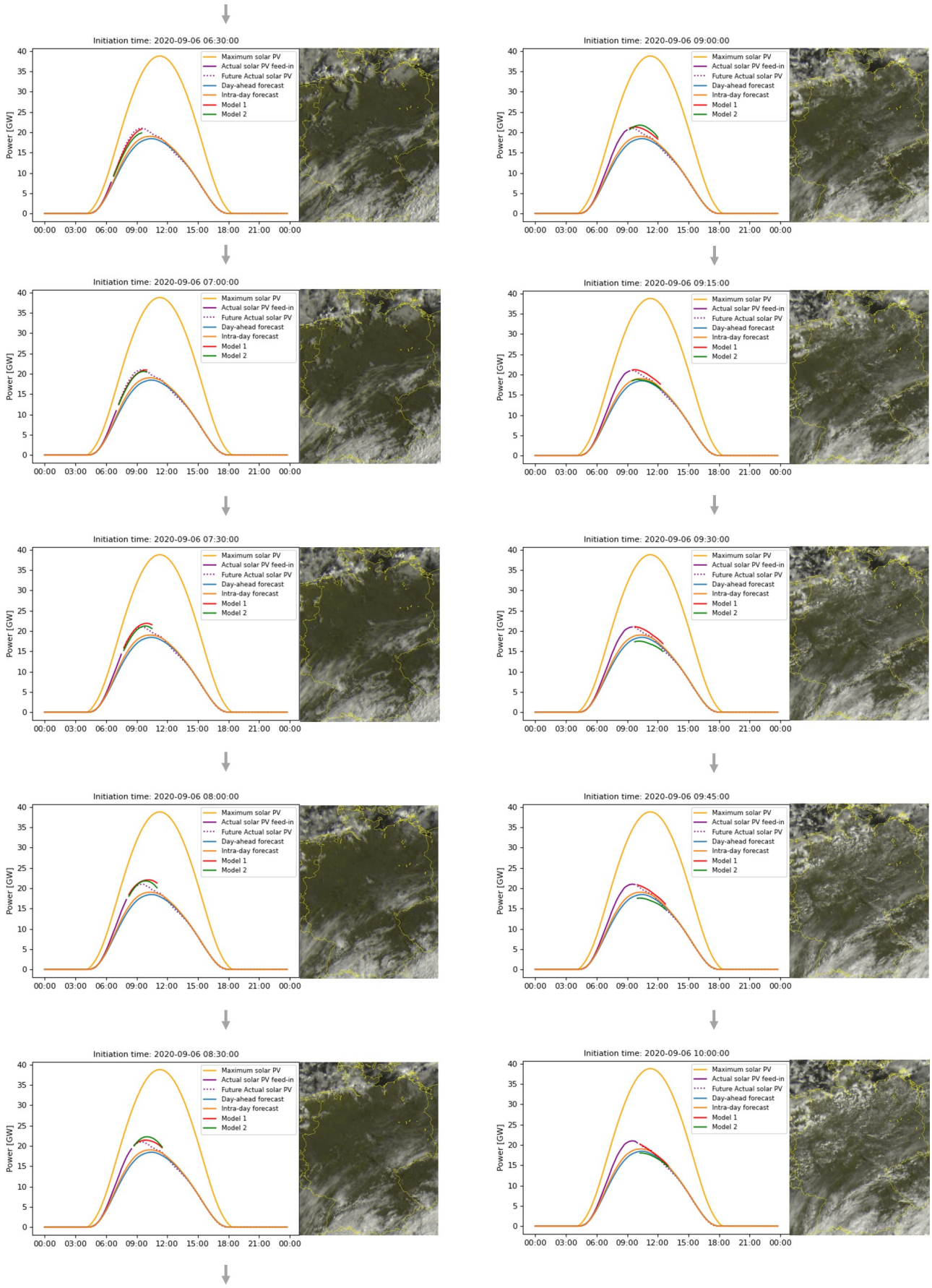


Figure 5.5: Time series of the forecasts made for case study 2 with accompanying satellite images. The first forecast subfigure is presented top left with the arrows indicating the chronological order.

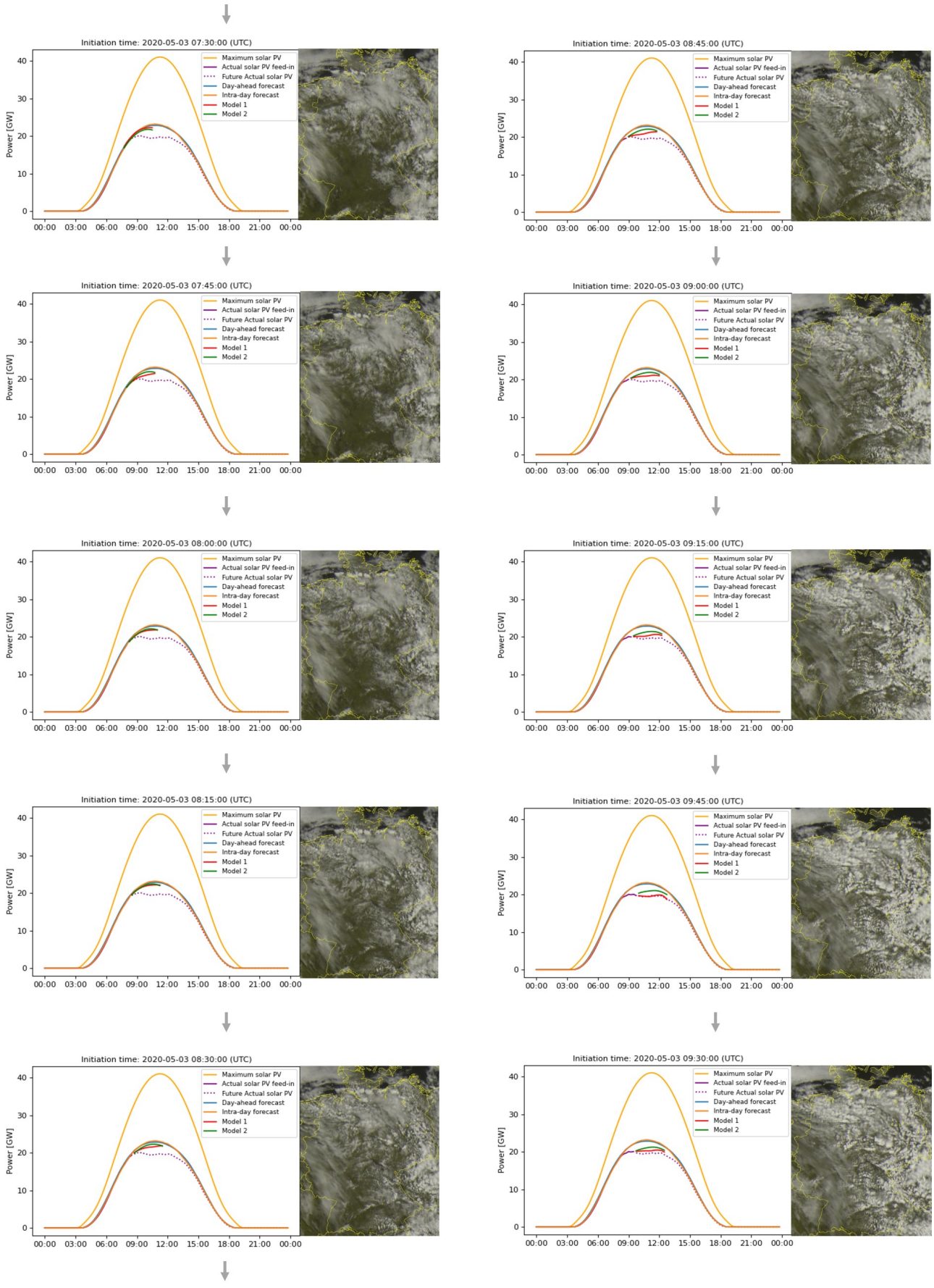


Figure 5.6: Time series of the forecasts made for case study 3 with accompanying satellite images. The first forecast subfigure is presented top left with the arrows indicating the chronological order.

5.3 Regional forecast

A forecast over a large number of individual solar PV power plants reduces the variability and fluctuation of the total solar PV power output, resulting in a more reliable forecast for aggregated regions compared to individual sites or small regions [9]. Although the initial application of the proposed model framework is for Germany, representing a large aggregated region, it is useful to evaluate if the forecasting framework is easily applicable and transferable to smaller or different regions. Moreover, regional forecasts are of major interest to energy markets, as power can also be traded per dedicated TSO control region in Germany and with a shorter lead time to contract delivery. For this reason, the proposed forecast models 1 and 2 are additionally trained and evaluated on one local TSO control area.

The grid area controlled by 50Hertz is selected for this purpose, located in the North-Eastern region of Germany as visible in fig. 4.1. The area represents 25% of the installed solar PV capacity with 14.2 GW in December 2020 and has an areal footprint of roughly one-third of the size of Germany. The datasets for this region are extracted and pre-processed in a similar manner as described in section 4.1. The models are trained on a dataset ranging from 01-01-2015 00:00:00 to 31-12-2019 23:45:00 and are not further tuned for the specific task. The performance of the proposed model 1 and model 2 are again evaluated against an unseen dataset, ranging from 01-01-2020 00:00:00 to 31-12-2020 23:45:00. The overall performance results are presented in table 5.5. Similar to the forecast for Germany as a whole, the performance of model 1 is superior to all other models, although less significantly. On the contrary, the forecast accuracy is lower for model 2 compared to the Entsoe intra-day forecast for 50Hertz. Moreover, comparing the results to table 5.1, it is apparent that the sMAPE and tMAPE of all forecast models are higher for the 50Hertz regional forecast. This is in line with the results from literature on solar PV power forecasting, as forecast for larger regions are more reliable due to the averaging effect of local inaccuracies.

Table 5.5: Comparison of the forecast accuracy of the benchmark and proposed models for the 50Hertz grid region using the performance metrics MAE, RMSE, sMAPE and tMAPE for the year 2020. The best performance is marked with bold font.

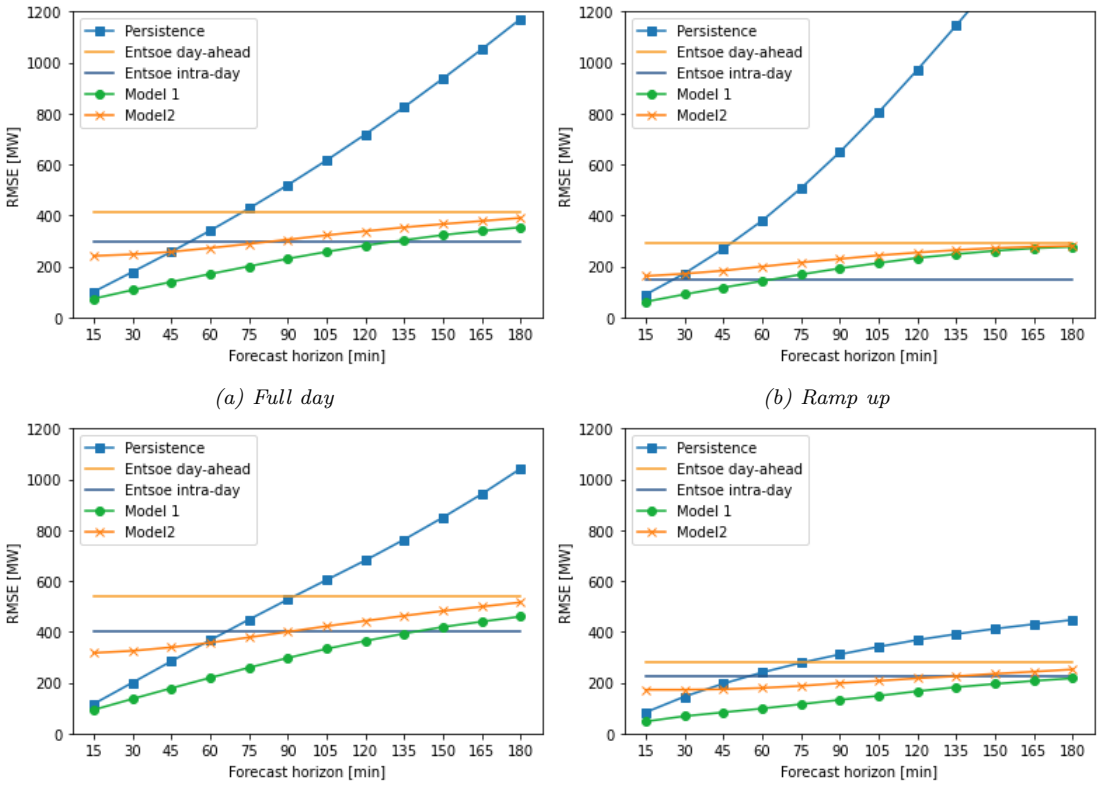
Model	**RMSE (MW)**	**MAE (MW)**	**sMAPE (%)**	**tMAPE (%)**
Persistence	684	433	42.8	2.9
Entsoe day-ahead	412	282	20.6	1.8
Entsoe intra-day	297	195	14.7	1.3
Model 1	**248**	**154**	**9.9**	**1.0**
Model 2	317	213	13.1	1.4

Next, the regional forecast performance is evaluated while considering the period of the forecast horizon relative to the characteristic daily solar ramp. In table 5.2 the solar PV power prediction errors are grouped in the ramp up, peak, ramp down events. Contrary to the countrywide forecast, it is observed that the accuracy of model 1 is lower for the ramp up in terms of squared and absolute error compared to the Entsoe intra-day forecast. Although, the lower accuracy occurs with higher actual solar PV feed-in power, demonstrated by the lower sMAPE score. This indicates that the majority of the error impact occurs close in time to the solar peak rather than close to sunrise. Moreover, comparing the results to table 5.2, it is striking that the sMAPE metric for models 1 and 2 are mainly higher in the peak of the solar ramp, and less so in the ramp up and ramp down. This indicates that the higher inaccuracy of regional forecast compared to the countrywide forecast occurs in the peak of the solar ramp. In local case study 1, presented later in this section, a day is presented where the accuracy of both models 1 and 2 are lower mainly in the solar peak.

Table 5.6: Comparison of the proposed models and benchmark forecast accuracy for the 50Hertz grid region and split in the three characteristic solar ramp events. The best performance is marked with bold font and the RMESE, MAE and sMAPE in (MW), (MW) and (%), respectively.

	Ramp up			Peak			Ramp down		
Model	**RMSE**	**MAE**	**sMAPE**	**RMSE**	**MAE**	**sMAPE**	**RMSE**	**MAE**	**sMAPE**
Persistence	945	612	58.7	636	430	13.0	323	248	69.5
Entsoe day-ahead	292	186	20.7	542	417	13.7	281	188	30.6
Entsoe intra-day	**147**	**91**	13.0	404	304	9.9	224	147	23.5
Model 1	202	124	**12.6**	**322**	**221**	**7.1**	**148**	**90**	**11.2**
Model 2	233	147	13.8	417	313	10.3	207	137	16.4

It is again interesting to analyse the performance of the proposed models over the forecast horizon. fig. 5.7 shows the prediction accuracy of the models for each time step across the 3 hour forecast horizon. When comparing these results to fig. 5.1, it is interesting to see that model 2 has a stronger dependency on forecast horizon, with a higher relative accuracy at shorter forecast horizons and a stronger accuracy degradation. Similarly, it is observed that the accuracy of model 1 degrades faster over the forecast horizon for all ramp events. Now, on average, only outperforming the Entsoe intra-day forecast for a 60-minute forecast in the ramp up and a 135-minute forecast in the peak. The development of two individual solar PV forecasting models in this research project was motivated by the live availability of their respective data input streams. The actual solar PV feed-in power input feature of model 1 for 50Hertz is available with a 15-30 minute delay, less than the aggregated publication delay over Germany. Looking at the forecast of model 1 for 50Hertz, the first relevant forecast point is roughly 30-minutes after forecast initiation, compared to 5-minutes for model 2. Even if the prediction accuracy across the forecast horizon of model 1 in fig. 5.7 is shifted backward by 30-minutes, the error metrics indicate a higher accuracy compared to model 2.



(a) Full day *(b) Ramp up*

(c) Peak *(d) Ramp down*

Figure 5.7: Comparison of the prediction accuracy of the models for the 50Hertz grid region across the 3-hour forecast horizon, with a split for the full day, ramp up, peak and ramp down solar ramp events.

To further analyze difference in the performance results between the regional forecast and the forecast for Germany two case studies are presented. The aim of these case studies is to highlight several characteristic differences in the data input, model behavior and forecast characteristics. In the first case study a winter day broken cloud cover is presented. Next, in the second study a spring day with overcast conditions is analyzed.

Regional case study 1: Winter day with broken cloud cover
As stated at the beginning of this section, a forecast over a large number of individual solar PV power plants spread over a wide area reduces the variability and fluctuation of the total solar PV power output. This results in a more reliable forecast for aggregated regions compared to individual sites or small regions. This effect is even stronger for weather conditions that feature high spatial and temporal variation in the cloud cover. To illustrate this effect, this case study presents a winter day with rapidly changing broken cloud cover on 10 February 2020. A forecast time series is presented in section 5.3, starting from 08:00 to 12:30 with a display resolution of 30-minutes.

In the first forecast run visible in section 5.3 at 08:00 both models 1 and 2 point to a peak solar PV power output of about 2 GW, in line with the day-ahead forecast and higher than the intra-day forecast. The corresponding satellite image at 08:00 shows a variable cloud cover with both patches of thick cloud cover and cloud breakup. In the satellite images from 08:00 to 09:30 it can be observed that the cloud cover has a high spatial and temporal variability. In the satellite images from 9:30 to 10:30 the cloud cover seems to get more concentrated on the central region of 50Hertz, resulting in a lower solar PV power output. However, it takes until the forecast run of 10:30, for both models 1 and 2, to fully incorporated these conditions in the forecasts for the next 3-hour. At that point, the cloud cover changes again with clearing conditions in the south-western region of 50Hertz. This results in the actual solar PV power output to move back towards the day-ahead forecast. It is observed in the images of 11:00 that both models pick up the increasing power output, but are not able to accurately forecast the magnitude of the change. In the remaining forecast runs the power output continues to change rapidly within the 3-hour forecast period, where both models show their inability to predict these rapid changes. The resulting impact of highly variable cloud cover on the forecast models in this case study is 2-fold. The models are not able to predict rapid changes in the cloud cover and produce accurate highly flexible 3-hour horizon forecast. Additionally, as the effects of changing cloud cover have a larger impact on the solar PV power generation around the peak of the solar ramp, the model accuracy is most influenced around this time.

The results of four forecast runs on the same day for the complete German net region are presented in fig. 5.9. Specifically, the forecast runs of 09:00, 10:00, 10:30 and 11:30 on 10 February 2020 are shown. Here it is observed that the variable cloud cover and rapidly changing conditions are present everywhere over Germany. This results in a variable actual solar PV power output, as observed in fig. 5.9, but less significant and volatile than observed for the 50Hertz control region. The effects of the strong temporal and spatial cloud cover variability is smoothed out over the larger area, resulting in less fluctuating input features and a more accurate 3-hour forecast. Yet, it is still observed that under these conditions the models, especially model 2, have a lower prediction accuracy. This observation is in accordance with the results shown in fig. 5.3, where the relation between model performance and cloud conditions is investigated and showed the most unfavorable performance under broken cloud cover conditions.

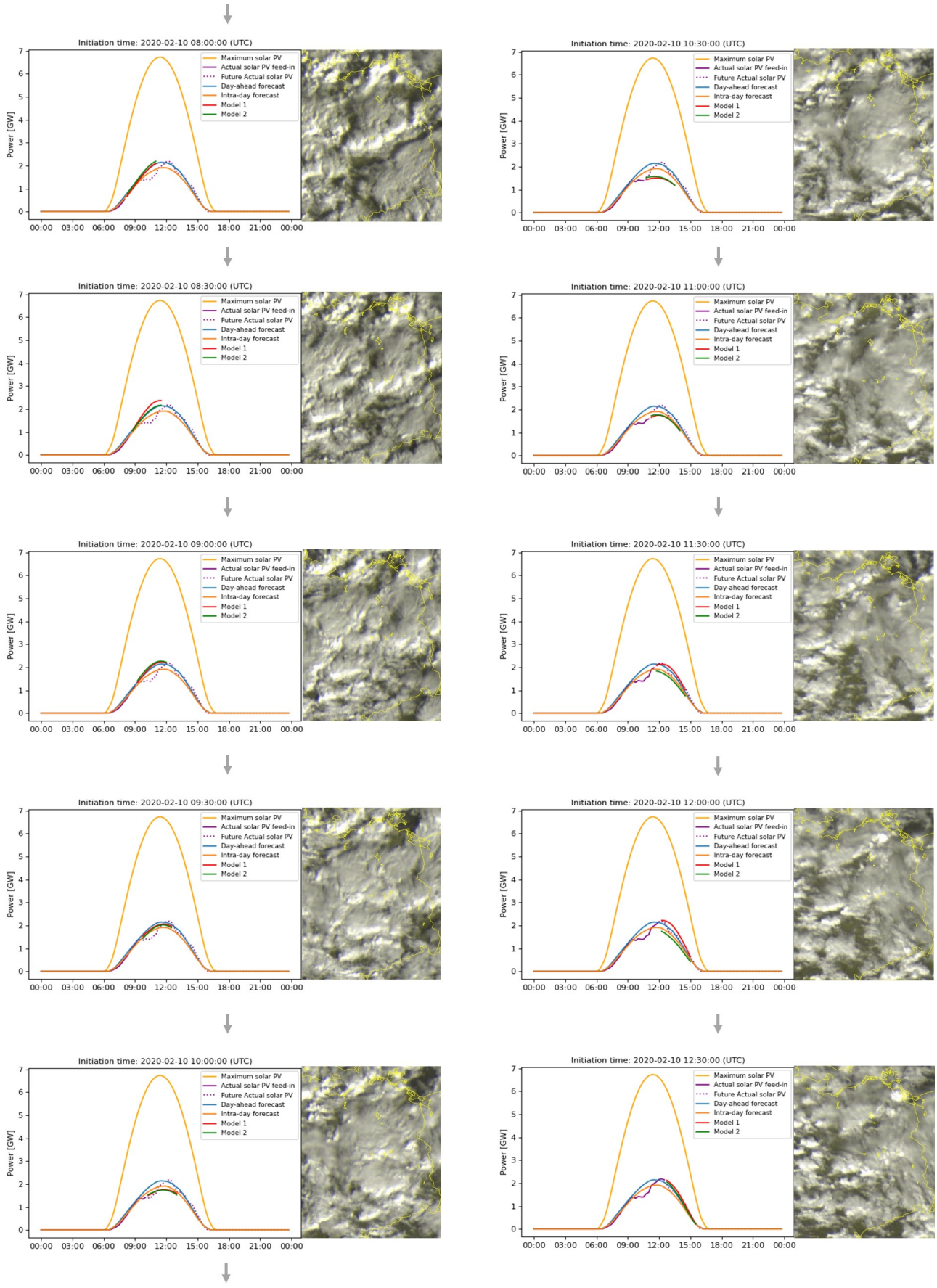
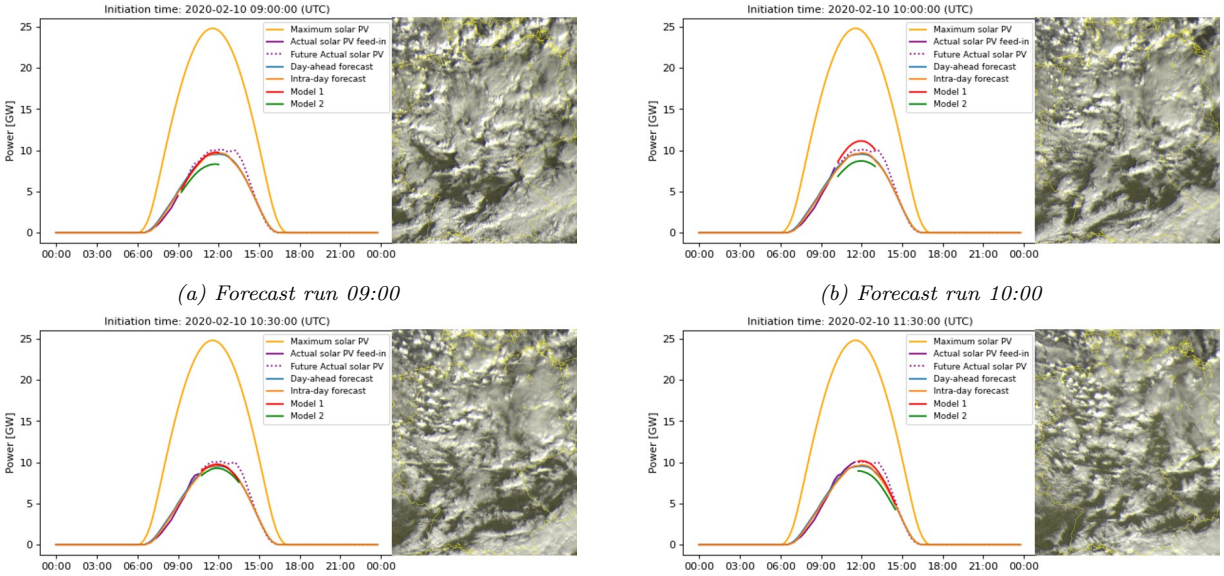


Figure 5.8: Time series of the forecasts made for regional 50Hertz case study 1, with accompanying satellite images over the grid control region. The first forecast subfigure is presented top left with the arrows indicating the chronological order.



(a) Forecast run 09:00

(b) Forecast run 10:00

(c) Forecast run 10:30

(d) Forecast run 11:30

Figure 5.9: Four forecast runs for Germany on the same day as the regional case study 1, 10 February 2020.

Regional case study 2: Spring day with overcast conditions

In this case study a day is presented with overcast cloud conditions in spring, with uniform cloud clearing. A forecast time series is presented in section 5.3, starting from 07:00 to 11:30 with a display resolution of 30-minutes. This case presents a day for which the spatial and temporal variation of the cloud cover is significantly less and also more uniform compared to regional case study 1. In the satellite image accompanying the 07:00 forecast run it can be observed that there is cloud cover over the entire 50Hertz grid control region and the forecast of models 1 and 2 initially follow the intra-day forecast. In the following satellite images until the forecast at 10:00 it can be observed that there is a clearing of cloud conditions in the Southern/central part of the 50Hertz grid control region, resulting in a higher solar PV power forecast for both models 1 and 2. Next the cloud cover seems to increase again after the 10:00 forecast run and the models forecasts are adjusted lower. It is observed that both models 1 and 2 are able to more accurately predict the solar PV power output compared to case study 1, as the changes in cloud cover are more uniform with less spatial and temporal variability. Moreover, this case show that if conditions are changing, although more uniform, both models 1 and 2 are able to pick up changes in conditions relative to the intra-day and day-ahead forecast, and subsequently provide and accurate solar PV power forecast.

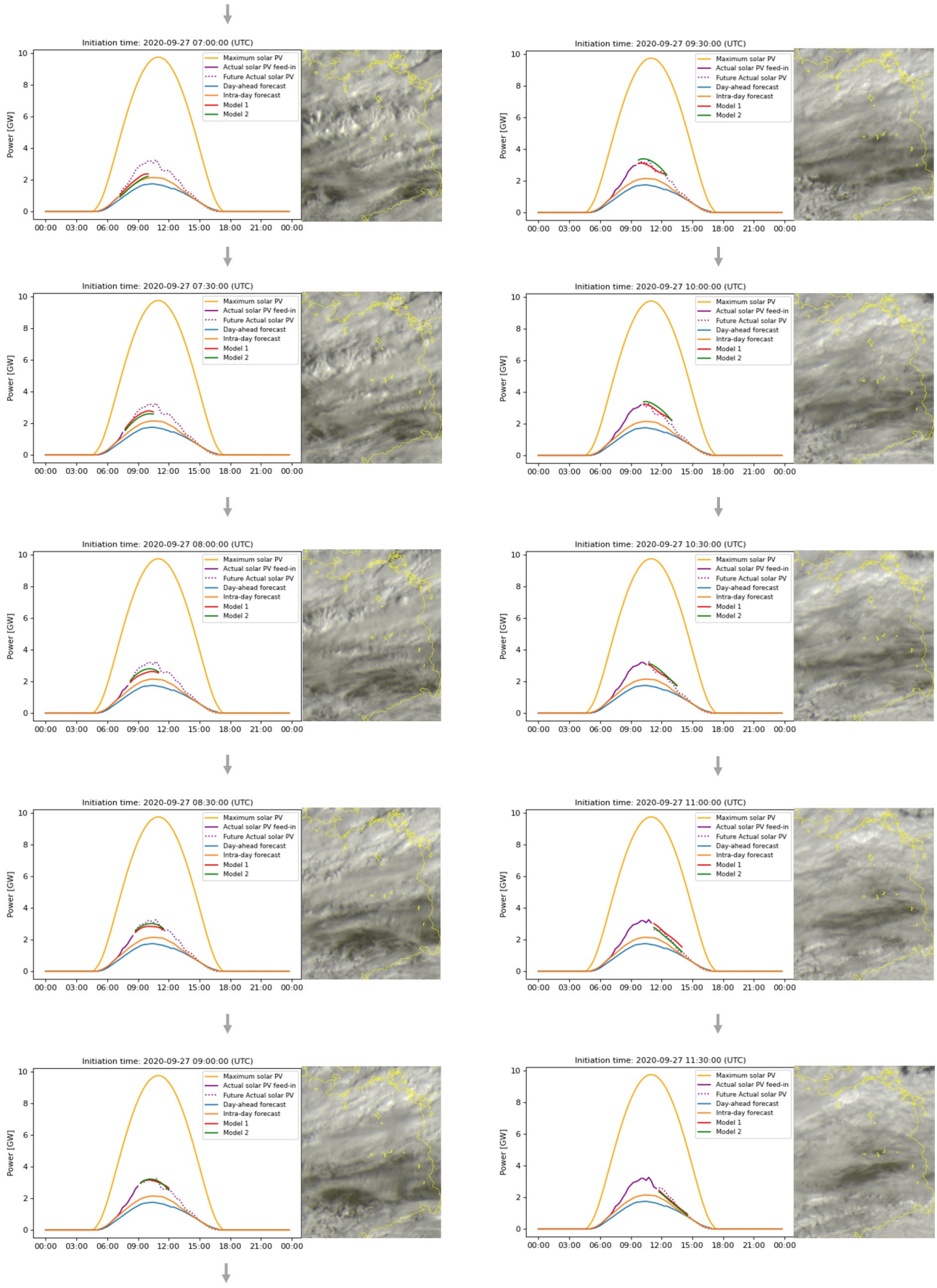


Figure 5.10: Time series of the forecasts made for regional 50Hertz case study 2, with accompanying satellite images over the grid control region. The first forecast subfigure is presented top left with the arrows indicating the chronological order.

Chapter 6: Conclusion & recommendations

The problem statement at the start of this report stresses the need for accurate solar PV power forecasts, given the continuous massive adoption of solar PV in the grid. The increasing power generation from solar PV poses many environmental and economic benefits, however the variability of its power output may also threaten the stability of our electricity supply. More accurate solar PV forecasts on various timescales have been identified as the most economical and efficient solutions to support the continuous integration of solar PV. Moreover, a knowledge gab has been identified with respect to the application of satellite images for solar PV power forecast in a straightforward, yet accurate way. To this end, the research objective of this project was to investigate and leverage on the recent developments in the field of deep learning, with respect to convolutional neural networks (CNN) and Long Short Term Memory (LSTM), to process real-time satellite images and solar PV power generation data in a deterministic intra-day solar PV power forecast.

This study concentrated on the proposal of two deep learning frameworks for an aggregated solar PV power forecast over Germany with a forecast horizon of 3 hours and interval of 15-minutes. Moreover, the forecast needed to be applicable in a real-time operation setting with a short forecast lag. This requires a forecast model which has a forecast frequency of 15-minutes and utilizes real-time data. For these settings two main data streams are available: the actual solar PV feed-in power in Germany, and high resolution visible satellite images. The proposed methodology comprises of the development of two dedicated deep learning models to process these data streams. This approach includes numerous data pre-processing steps, extensive experimentation with deep learning architectures and model hyperparameter tuning. The outcome of this approach were two forecast models, called model 1 and model 2.

In model 1, an encoder-decoder LSTM network is shown to be an effective architecture to process multivariate one-dimensional time series input data into a multi-step solar PV power forecast. The input features of the model consists of: actual solar PV power generation data, a NWP model based day-ahead solar PV power forecast, and the date-time parameters quarter-of-day and day-of-year. In model 2, a CNN-LSTM network is shown to be an adequate architecture to process multivariate and three-dimensional time series input data, including satellite images, into a multi-step solar PV power forecast. This model processes the same input features as model 1, except that the actual solar PV power generation data is replaced by satellite image derived solar PV generation projections. The difference in input feature selection results in a forecast lag of maximum 60-minutes for model 1 and only 5-minutes for model 2 in real-time operations.

The overall forecast accuracy of the two proposed models for the year 2020 are compared to a persistence model and a NWP model based day-ahead and intraday forecast provided by the German TSOs. The proposed models were shown to be equal or better than the benchmark models, and in turn, to be an accurate tool for continuously updating short-term solar PV forecast. In particular, the best prediction accuracy is obtained using the latest actual solar PV generation data and alternatively using satellite images. The further contributions of this research project are five-fold.

First, the satellite images are shown to be a good proxy for the latest actual solar PV power generation and subsequent power predictions. The CNN-LSTM architecture has shown the ability to derive spatial and temporal relations of clouds in satellite images and process these to a projection of their corresponding solar PV generation. Given the dependency on real-time data for short-term solar PV power forecast, this method proves to be an attractive alternative data source if actual solar PV generation data is not available in real-time. However, due to the highly dynamic nature of clouds and weather condition, a large number of cloud cover settings can occur in satellite images. As a results, it is possible that several characteristic cloud cover settings are unrepresented in the model training dataset, leading to forecast inaccuracies.

Next, the superior performance of models 1 and 2 in the period of peak solar PV generation compared to the solar ramp up period, stresses the underlying weakness of statistical forecast methods. The reliance on past input data, that represent the future weather conditions, make these models less fitting for predictions in the morning solar ramp, as data is not at all or insufficiently available.

The third contribution presents itself in the analysis of the reducing prediction accuracy of both statistical models 1 and 2 over longer forecast horizons. This characteristic illustrates the complex non-linear relations between input data and a multi-step target sequence. In comparison to a trivial persistence model, the highly

non-linear encoder-decoder LSTM network demonstrates its robustness in coping with longer multi-step forecast horizons. However, depending on weather conditions and forecast initiation time, the statistical deep learning models show inferior prediction accuracy for forecast horizons longer than 2-3 hours in comparison to the NWP based intra-day and day-ahead forecasts.

Next, it is found that both physics based benchmark models and the two proposed statistics based models show a dependency in prediction accuracy on weather conditions. The proposed forecast models 1 and 2 show the lowest accuracy under conditions featuring strong spatial and temporal variation in the cloud cover, such as with broken cloud cover or under convective cloud formation. Moreover, it is found that both models have the ability to identify events with convective cloud formation and translate these conditions in their forecasts. However, it also shown that both models do not have the ability to forecast these events before they onset.

The final contribution arises from the comparison study of the two proposed models employed for a country-aggregated solar PV forecast compared to a regional forecast. It is found that forecasts for larger regions are more reliable, which is likely due to the averaging effect of local inaccuracies. This is especially the case for weather conditions with high spatio-temporal variability in cloud cover.

In light of the encouraging results observed in this research project, one might question whether data-driven deep machine learning methods can replace more complex satellite image based cloud motion vector techniques or NWP based forecast models. The machine learning techniques used in model 1 and model 2 are shown to be powerful short-term forecasting methods and should definitely have a place in a comprehensive solar PV power forecasting toolbox. However, they also suffer from limitations highlighted in the contributions of this research project. Moreover, the black-box characteristic of deep learning models limits the understanding of the intricate relations between model input features and the output target. This results in a time consuming trail-and-error based approach to improve a models prediction accuracy, without fundamentally understanding the impact of the changes made. Additionally, short-term forecasts employed in a real-time operational setting need to be available with a short delay to real-time. The developed models make use of actual solar PV generation data and satellite images that result in a forecast lag of 60-minutes and 5-minutes, respectively. Given the short forecast horizon of 3-hours, the delay in data availability still significantly influences the relevance and functionality of the forecasts, especially for model 1. As the forecast delay is of significant importance to real-time power trading, the model performance cannot exclusively be judged based on the forecast accuracy expressed by error metrics, but should also be viewed in light of the forecast delay. Finally, the proposed statistical models are evaluated against physics based intra-day and day-ahead forecast models that are only generated once per day. Although these benchmark models are off significant importance to the energy market, both their fundamental working principles as their application is different from the two proposed models; thus making the forecast accuracies harder to compare. Taking all this into consideration, it seems apparent that deep learning based solar PV power forecasting models are rather an addition to a comprehensive solar PV power forecast toolbox than a replacement.

In reflection on the development of the deep learning solar PV power forecasting models in this research project, recommendation for future research are summarized as follows:

1. The CNN-LSTM architecture of model 2 is designed to process the spatio-temporal information in a sequence of satellite images to solar PV generation data. However, in the proposed architecture the extraction of spatio-temporal relations is decoupled; by first using a CNN to process the spatial features in satellite images to a representation vector, and subsequently extract temporal features from the matrix of representation vectors with a LSTM. In order to couple the extraction of spatial and temporal features, the novel ConvLSTM should be further investigated. Although in the course of this research project such a model has been developed, the training of the ConvLSTM architecture surpassed the limits of the available computational resources. This limiting factor should be considered and dealt with in future research.
2. The CNN-LSTM architecture designed for model 2 aims to extract the spatio-temporal information in a sequence of satellite images. An alternative approach is suggested in which a CNN is developed to process individual satellite images to their corresponding solar PV power output. This approach removes the training error dependency on the LSTM model connected to the CNN, however it would also remove the temporal dependency between consecutive image. Investigating this approach can lead to new insights regarding which model elements and input-output relations are of importance to further improve solar PV power forecast accuracy.

3. In this research project, satellite images in RGB format from Weerslag where used to derive the solar PV generation. The use of these images was motivated by the availability of a dataset and a direct available method to obtain a real-time data stream of new images. However, the implementation of more advanced data products or individual channels of the SEVIRI imager operated by EUMETSAT can provide additional context on the weather and cloud conditions. For example, the Snow RGB product of EUMETSAT allows for the separate identification of fog, low clouds and snow during day time. Although datasets are available for a number of these products, a straightforward access method to real-time data has not been found in the course of this project.
4. In order to improve the forecast capabilities of satellite images based models (such as model 2) around sunrise, the additional use of infrared satellite images as input feature could be a promising approach. Infrared satellite images provide information on cloud cover and fog even when the sun is below elevations for which the visible channel satellite images do not provide information.
5. In the analysis of the forecast results the stationarity of the input data was discussed, with a focus of the normalization of the actual solar PV feed-in power by means of a clear sky model. In the early stages of model development, it was found that partially stationary solar PV feed-in data improved model performance. More specifically, the data was normalized with a bias towards lower normalized solar PV power near sunrise and sunset. In future research, considering the addition of day-ahead solar PV forecast data to improve the performance in the morning solar ramp, it should be investigated if this decision is still valid.
6. Based on a literature study of stacked LSTM networks and model experiments it was found that the addition of up to 3 LSTM layers improved the forecast accuracy of model 1. However, after the addition of the Entsoe day-ahead forecast as input feature to the model, it was later found that the inclusion of stacked LSTM layers did neither improve nor reduce the model training and validation error. Although the understanding of the intricate relations between model design, input feature and the output targets is extremely difficult, it should be further investigated if stacked LSTM layers are desirable or of no additional value to this forecast problem.
7. In order to add context to the model on seasonal and day-to-day patterns, two input features were introduced: the day-of-year (DOY) and the quarter-of-day (QOD). As a pre-processing step the DOY and QOD are normalized using a cosine function. However, with just applying the cosine transformation there is symmetry across the period of the cyclical features. In future work it is advised to use a two dimensions for the cyclical feature normalization, with both a cosine and sine function.
8. The encoder-decoder LSTM network was selected based on its characteristics found in literature. Although it has demonstrated its robustness in coping with multi-step forecasts, it is suggested that in future research its performance is evaluated against other neural network architectures, such as the Gated Recurrent Unit (GRU) or a deep MLP.
9. The selection process of model architecture and hyperparameter settings in this research project was guided by a combination of theory from literature and numerous trail-and-error experiments. After demonstrating in this project that the two proposed forecast approaches are a valuable tool for short-term solar PV predictions, it should be investigated if a systematic grid search of several hyperparameters, requiring more time and computational resources, can improve model results.
10. In view of the conclusion that the deep learning models 1 and 2 are an addition rather than a replacement in the solar PV power forecasting toolbox, an ensemble model of physical and statistical models is a promising avenue for future research. A suggestion of such a hybrid model is the addition of NWP model derived irradiance, temperature or cloud cover maps to the CNN-LSTM architecture of model 2. This method allows for the extraction of spatial and temporal features in the NWP model data, without manually engineering physical equations and providing information on PV systems characteristics and locations.

Bibliography

[1] International Energy Agency, *Renewables 2020 Analysis and forecast to 2025*, 2020. [Online]. Available: https://webstore.iea.org/download/direct/4234?fileName=Renewables_2020-PDF.pdf

[2] M. Schmela, A. Beauvais, N. Chevillard, M. Heisz, M. Herrero Cangas, M. Labordena, T. Lewis, and R. Rossi, *EU Market Outlook For Solar Power*, Dec 2020. [Online]. Available: https://www.solarpowereurope.org/wp-content/uploads/2020/12/3520-SPE-EMO-2020-report-11-mr.pdf?cf_id=26111

[3] S. Shivashankar, S. Mekhilef, H. Mokhlis, and M. Karimi, "Mitigating methods of power fluctuation of photovoltaic (pv) sources – a review," *Renewable and Sustainable Energy Reviews*, vol. 59, p. 1170–1184, Jun 2016.

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

[5] M. Q. Raza, M. Nadarajah, and C. Ekanayake, "On recent advances in pv output power forecast," *Solar Energy*, vol. 136, p. 125–144, Oct 2016.

[6] A. Beauvais and M. Schmela, *Grid Intelligent Solar Unleashing the Full Potential of Utility-Scale Solar Generation in Europe*, Dec 2018. [Online]. Available: https://www.solarpowereurope.org/wp-content/uploads/2019/05/1819-SPE-Grid-Intelligent-Solar-report.pdf?cf_id=23321

[7] H. Wirth, *Recent Facts about Photovoltaics in Germany*, Dec 2020. [Online]. Available: https://www.ise.fraunhofer.de/content/dam/ise/en/documents/publications/studies/recent-facts-about-photovoltaics-in-germany.pdf

[8] 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. 109792, Oct 2020.

[9] F. Antonanzas-Torres, R. Urraca, J. Polo, O. Perpiñán-Lamigueiro, and R. Escobar, "Clear sky solar irradiance models: A review of seventy models," *Renewable and Sustainable Energy Reviews*, vol. 107, p. 374–387, Jun 2019.

[10] M. Diagne, M. David, P. Lauret, J. Boland, and N. Schmutz, "Review of solar irradiance forecasting methods and a proposition for small-scale insular grids," *Renewable and Sustainable Energy Reviews*, vol. 27, 2013.

[11] L. Liu, Y. Zhao, D. Chang, J. Xie, Z. Ma, Q. Sun, H. Yin, and R. Wennersten, "Prediction of short-term pv power output and uncertainty analysis," *Applied Energy*, vol. 228, p. 700–711, Oct 2018.

[12] E. Lorenz and D. Heinemann, "Prediction of solar irradiance and photovoltaic power," *Comprehensive Renewable Energy*, vol. 1, p. 239–292, 2012.

[13] European Centre for Medium-Range Weather Forecasts, "Documentation and support." [Online]. Available: https://www.ecmwf.int/en/forecasts/documentation-and-support

[14] A. Dolara, S. Leva, and G. Manzolini, "Comparison of different physical models for pv power output prediction," *Solar Energy*, vol. 119, p. 83–99, Sep 2015.

[15] Y. Li, Y. Su, and L. Shu, "An armax model for forecasting the power output of a grid connected photovoltaic system," *Renewable Energy*, vol. 66, p. 78–89, Jun 2014.

[16] P. Bacher, H. Madsen, and H. A. Nielsen, "Online short-term solar power forecasting," *Solar Energy*, vol. 83, no. 10, p. 1772–1783, Oct 2009.

[17] H. T. Pedro and C. F. Coimbra, "Assessment of forecasting techniques for solar power production with no exogenous inputs," *Solar Energy*, vol. 86, no. 7, p. 2017–2028, Jul 2012.

[18] G. Reikard, "Predicting solar radiation at high resolutions: A comparison of time series forecasts," *Solar Energy*, vol. 83, no. 3, p. 342–349, Mar 2009.

[19] M. Bouzerdoum, A. Mellit, and A. Massi Pavan, "A hybrid model (sarima–svm) for short-term power forecasting of a small-scale grid-connected photovoltaic plant," *Solar Energy*, vol. 98, p. 226–235, Dec 2013.

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016, http://www.deeplearningbook.org.

[21] N. K. Chauhan and K. Singh, "A review on conventional machine learning vs deep learning," *2018 International Conference on Computing, Power and Communication Technologies (GUCON)*, Sep 2018.

[22] S. Al-Dahidi, O. Ayadi, M. Alrbai, and J. Adeeb, "Ensemble approach of optimized artificial neural networks for solar photovoltaic power prediction," *IEEE Access*, vol. 7, p. 81741–81758, 2019.

[23] W. Lee, K. Kim, J. Park, J. Kim, and Y. Kim, "Forecasting solar power using long-short term memory and convolutional neural networks," *IEEE Access*, vol. 6, p. 73068–73080, 2018.

[24] M.-T. Do, T. Soubdhan, and B. Robyns, "A study on the minimum duration of training data to provide a high accuracy forecast for pv generation between two different climatic zones," *Renewable Energy*, vol. 85, p. 959–964, Jan 2016.

[25] G. Li, H. Wang, S. Zhang, J. Xin, and H. Liu, "Recurrent neural networks based photovoltaic power forecasting approach," *Energies*, vol. 12, no. 13, p. 2538, Jul 2019.

[26] J. Lago, K. De Brabandere, F. De Ridder, and B. De Schutter, "Short-term forecasting of solar irradiance without local telemetry: A generalized model using satellite data," *Solar Energy*, vol. 173, p. 566–577, Oct 2018.

[27] M. Abdel-Nasser and K. Mahmoud, "Accurate photovoltaic power forecasting models using deep lstm-rnn," *Neural Computing and Applications*, vol. 31, no. 7, p. 2727–2740, Oct 2017.

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

[29] J. Kuhnert, E. Lorenz, and D. Heinemann, "Satellite-based irradiance and power forecasting for the german energy market," *Solar Energy Forecasting and Resource Assessment*, p. 267–297, 2013.

[30] W. Greuell, J. F. Meirink, and P. Wang, "Retrieval and validation of global, direct, and diffuse irradiance derived from seviri satellite observations," *Journal of Geophysical Research: Atmospheres*, vol. 118, no. 5, p. 2340–2361, Mar 2013.

[31] E. Lorenz, J. Hurka, G. Karampela, D. Heinemann, H. G. Beyer, and M. Schneider°, "Qualified forecast of ensemble power production by spatially dispersed grid-connected pv systems," *Proceedings of the 23rd European photovoltaic solar energy conference and exhibition, Spain*, 01 2007.

[32] F. Klumpp, "Potential for large scale energy storage technologies – comparison and ranking including an outlook to 2030," *Energy Procedia*, vol. 73, p. 124–135, 2015.

[33] EPEX Spot, "Trading brochure epex spot 2019-2020," 2019. [Online]. Available: https://www.epexspot.com/sites/default/files/2019-02/2019-01-17_TradingBrochure_V2.pdf

[34] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436–444, May 2015.

[35] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, p. e00938, Nov 2018.

[36] S. Sharma and A. Athaiya, "Activation functions in neural networks," *International Journal of Engineering Applied Sciences and Technology*, vol. 4, no. 12, p. 310–316, 2020.

[37] G. R. Yang and X.-J. Wang, "Artificial neural networks for neuroscientists: A primer," *Neuron*, vol. 107, no. 6, p. 1048–1070, Sep 2020.

[38] S. Ruder, "An overview of gradient descent optimization algorithms," *ArXiv*, vol. abs/1609.04747, 2016.

[39] G. Montavon, O. B, and K.-R. Muller, *Neural networks : tricks of the trade.* Springer, 2012.

[40] D. Soydaner, "A comparison of optimization algorithms for deep learning," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 34, no. 13, p. 2052013, Apr 2020.

[41] C. A. R. de Sousa, "An overview on weight initialization methods for feedforward neural networks," *2016 International Joint Conference on Neural Networks (IJCNN)*, p. 52–59, Jul 2016.

[42] L. Datta, "A Survey on Activation Functions and their relation with Xavier and He Normal Initialization," *arXiv e-prints*, p. arXiv:2004.06632, Mar. 2020.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *arXiv preprint arXiv:1502.01852*, 2015.

[44] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," 2018.

[45] C. Garbin, X. Zhu, and O. Marques, "Dropout vs. batch normalization: an empirical study of their impact to deep learning," *Multimedia Tools and Applications*, vol. 79, no. 19-20, p. 12777–12815, Jan 2020.

[46] D. Kontogiannis, D. Bargiotas, and A. Daskalopulu, "Minutely active power forecasting models using neural networks," *Sustainability*, vol. 12, no. 8, p. 3177, Apr 2020.

[47] W. Wang and Y. Yang, "Development of convolutional neural network and its application in image classification: a survey," *Optical Engineering*, vol. 58, no. 04, p. 1, Apr 2019.

[48] R. DiPietro and G. D. Hager, "Chapter 21 - deep learning: Rnns and lstm," in *Handbook of Medical Image Computing and Computer Assisted Intervention*, ser. The Elsevier and MICCAI Society Book Series, S. K. Zhou, D. Rueckert, and G. Fichtinger, Eds. Academic Press, 2020, pp. 503–519. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128161760000260

[49] F. Xiao, "Time series forecasting with stacked long short-term memory networks," *arXiv*, 2020.

[50] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, p. 1735–1780, Nov 1997.

[51] L. Chen and M. Xu, "Piecewise time series prediction based on stacked long short-term memory and genetic algorithm," *2020 Chinese Automation Congress (CAC)*, Nov 2020.

[52] S. Srivastava and S. Lessmann, "A comparative study of lstm neural networks in forecasting day-ahead global horizontal irradiance with satellite data," *Solar Energy*, vol. 162, p. 232–247, Mar 2018.

[53] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013.

[54] S. Du, T. Li, and S.-J. Horng, "Time series forecasting using sequence-to-sequence deep learning framework," *2018 9th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)*, Dec 2018.

[55] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[56] J. Brownlee, *Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery, 2018. [Online]. Available: https://books.google.nl/books?id=o5qnDwAAQBAJ

[57] Transnet BW, *Renewable Energies: Feeding in Photovoltaics*, 2016. [Online]. Available: https://www.transnetbw.com/files/pdf/transparenz/Detailinformationen_Einspeisung-Fotovoltaik.pdf

[58] Entsoe, "Entso-e transparency platform," 2021. [Online]. Available: https://transparency.entsoe.eu/dashboard/show

[59] Clean Energy Wire, "Map of transmission grid operators in germany," Sep 2014. [Online]. Available: https://www.cleanenergywire.org/experts/transnetbw

[60] Bundesnetzagentur, "Veröffentlichung von eeg-registerdaten." [Online]. Available: https://www.bundesnetzagentur.de/DE/Sachgebiete/ElektrizitaetundGas/Unternehmen_Institutionen/ErneuerbareEnergien/ZahlenDatenInformationen/EEG_Registerdaten/EEG_Registerdaten_node.html

[61] ERGEG, *Report on Transparency*, Sep 2007. [Online]. Available: https://www.bundesnetzagentur.de/SharedDocs/Downloads/EN/BNetzA/PressSection/PressReleases/2007/070913ReportElectrctyRegionalInitveId11480pdf.pdf;jsessionid=5C7D8E19251CEFD9FFF953EA1C84BF15?__blob=publicationFile&v=3

[62] 50Hertz, "Photovoltaics - current forecast," Mar 2021. [Online]. Available: https://www.50hertz.com/en/Transparency/GridData/Production/Photovoltaics

[63] J. Schmetz, P. Pili, S. Tjemkes, D. Just, J. Kerkmann, S. Rota, and A. Ratier, "An introduction to meteosat second generation (msg)," *Bulletin of the American Meteorological Society*, vol. 83, no. 7, p. 977–992, Jul 2002.

[64] Weerslag, "Satelliet bewolkingdagzonderfilter." [Online]. Available: https://weerslag.nl/BewolkingDagZonderFilter#

[65] Sat24, "Europees satellietbeeld." [Online]. Available: http://www2.sat24.com/history.aspx?culture=nl

[66] M. Reno, C. Hansen, and J. Stein, *Global Horizontal Irradiance Clear Sky Models: Implementation and Analysis*, Mar 2012. [Online]. Available: https://energy.sandia.gov/wp-content/gallery/uploads/SAND2012-2389_ClearSky_final.pdf

[67] P. Ineichen, "Comparison of eight clear sky broadband models against 16 independent data banks," *Solar Energy*, vol. 80, no. 4, p. 468–478, Apr 2006.

[68] A. B. Meinel and M. P. Meinel, "Applied solar energy. an introduction," 1 1976.

[69] F. Kasten and A. T. Young, "Revised optical air mass tables and approximation formula," *Applied Optics*, vol. 28, no. 22, p. 4735, Nov 1989.

[70] A. Smets, K. Jäger, O. Isabella, R. van Swaaij, and M. Zeman, *Solar Energy: The physics and engineering of photovoltaic conversion, technologies and systems.* UIT Cambridge Limited, 2016.

[71] G. V. Rozenberg, *Twilight.* Springer US, 1966.

[72] Stanford Vision Lab, "Imagenet," 2017. [Online]. Available: http://www.image-net.org/

[73] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, p. 3320–3328.

Appendices

Appendix A: Transfer learning

CNNs have been extensively used in various computer visual recognition task and have shown to outperform more traditional ANN in this field [47]. One of the most studied and basic tasks of visual recognition is image classification. Hence, CNN architectures applied to different visual recognition tasks are often derived from models that have shown good results in image classification [47]. Transfer learning is a method where an earlier developed and trained model is reused as a starting point for a new model aiming to tackle a related task [20]. This approach is often applied as the development and training of new models requires vast amount of computational and time resources, plus an extensive training dataset.

A common transfer learning approach is to select a base network, which is pre-trained for a challenging and broad image classification task such as the 1000-class ImageNet [72] database [73]. The first N layers of the base network are copied to the first N layers of the target model. The remaining layers of the target network are then designed, initialized and trained towards the specific task of the model. Some famous deep CNN architectures that have achieved good results in image classification tasks and have been pre-trained on the ImageNet dataset include: VGG16/19, ResNet, DenseNet, GoogleNet [47].

In the training process of the target model one can decide to back-propagate the errors obtained from the loss function of the new task through the copied layers in order to fine-tune them for the desired task, or leave the transferred layers frozen to training. Deep CNN networks exhibit an interesting characteristic when a model is trained on images: here generally the first-layer filters represent either color blobs or Gabor filters [73]. This phenomena occurs for completely different datasets and training task, hence the first-layer filters of a CNN are called 'general filters'. Contrary, the last-layer filters are known to be highly dependent on the dataset and the model task, and are thus called 'specific filters'. In the transfer learning technique it is the aim to transfer the general filters from the base network and possibly fine-tune the last-layer specific filters. Of course, if the first-layers are general and the last-layers are specific, there must be some transition zone governing which layers to transfer and which layers to fine-tune. However, this transition zone is model and task dependent and very hard to determine. Hence, often the quantity and quality of the target dataset, together with the number of parameters in the first N layers of the copied base model, influence the decision whether or not to fine-tune the parameters of the base model [73]. In case the number of parameters is large and the target dataset is small, the decision to fine-tune can result in overfitting and is best avoided [73]. Contrary, if the number of parameters is small and target dataset is large, overfitting is less of an issue and fine-tuning could improve model performance [73]. However, if the target dataset is large enough, there would not necessary be the need for transfer learning, as filters can be learned from scratch.

The transfer learning method is investigated during the development of the satellite image based CNN-LSTM solar PV power forecasting model. Numerous experiments are performed with the architecture of model 2A; here the LSTM encoder-decoder part of the network is maintained the same, while the CNN part is altered. An important difference with the final model 2A presented in this report is that at the time of the experiments the model was designed to forecast the future solar PV power output directly. So the target variable in the to be discussed experiments consists of the 12 time-step sequence $(P_{actual,t+1}, \ldots, P_{actual,t+12})$, compared to $(P_{actual,t-11}, \ldots, P_{actual,t})$ for the final model 2A presented in section 4.3.2.

In fig. A.1 the architecture of three experimental CNN architectures are presented, called model A, B and C. In model A the VGG16 network is used as the base network for the CNN, where both the architecture and the model weights trained on the ImageNet database are transferred. Additionally, all the CNN-layers are set to non-trainable. The top of the VGG16 network is removed and the last pooling layer is connected to a flatten layer and subsequently the encoder-decoder LSTM network. VGG16 is a CNN consisting of 13 convolutional layers and 5 max pooling layers that are structured in repeating blocks, as visualized in fig. A.1. Model B is exactly the same as model A, except that the last convolutional layer is set to trainable. Model C has exactly the same architecture as model 2A presented in section 4.3.2; here the CNN consists of a custom designed 8-layer structure (4 convolutional and 4 max-pooling layers) and all layers are trainable. The hyperparameter setting of the experiments are equal to those of model 2A as described in section 4.3.2. Similarly, the years 2015-2018 of the dataset are used for training and the complete year 2019 is used for validation. Similar experiments have been performed with other base networks, such as VGG19, ResNet and DenseNet, however VGG16 showed the best overall training performance and is therefore discussed here.

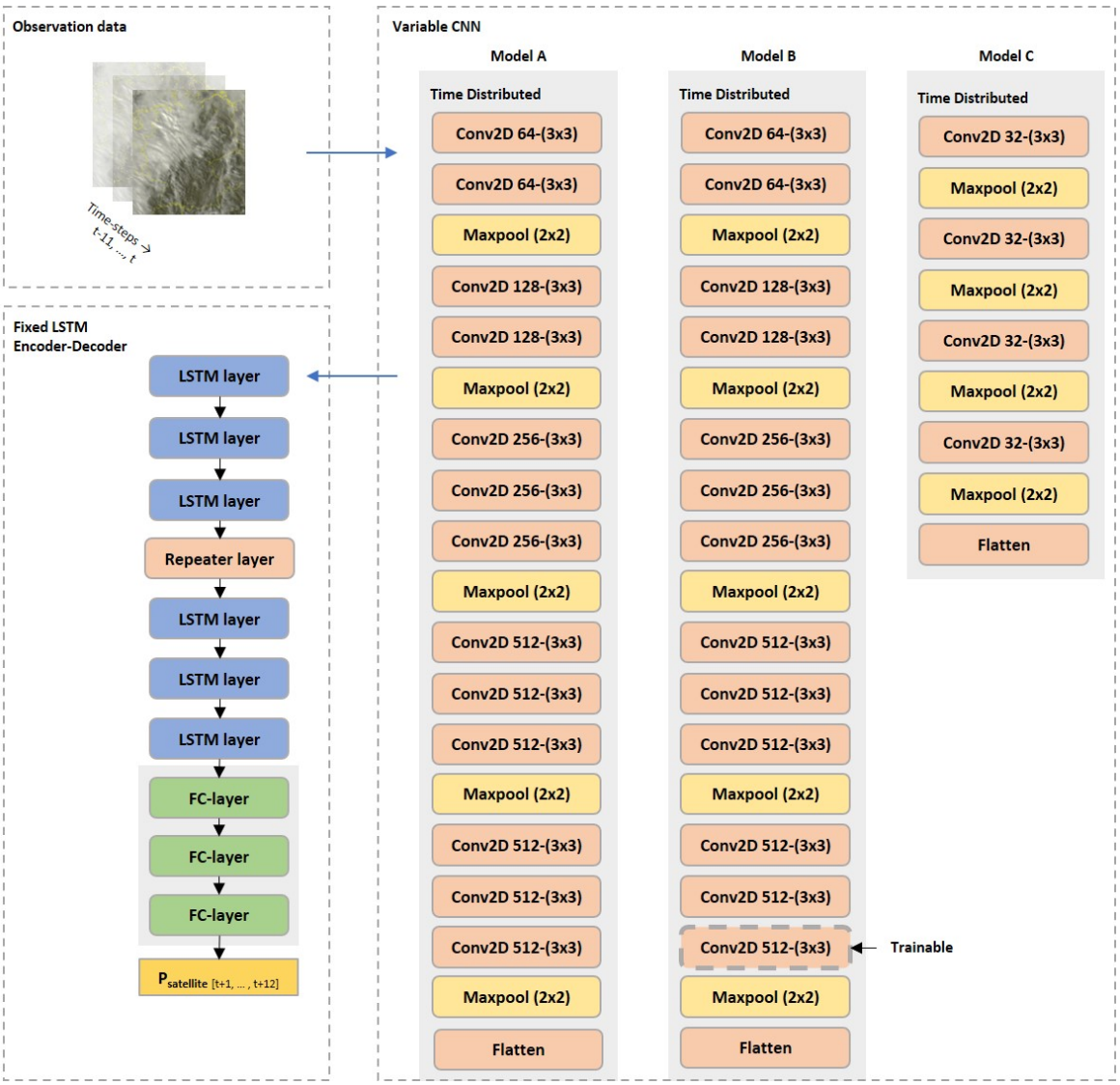


Figure A.1: Overview of three experimental CNN-LSTM architectures, with different CNN configurations, called model A, B and C. The value in angle brackets represents the kernel size of the filter, with the value in front the number of filters.

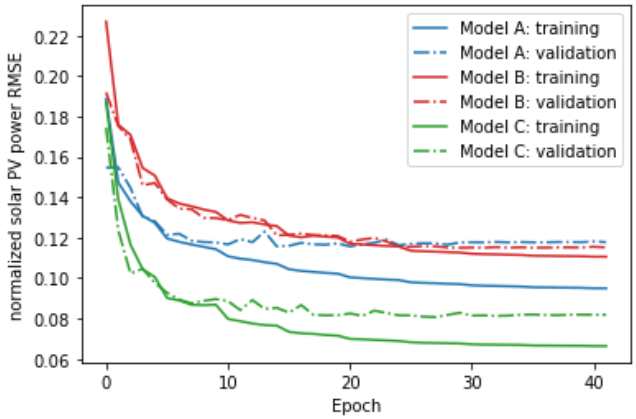


Figure A.2: Training and validation RMSE loss of experimental models A, B and C.

The training and validation RMSE loss of the three experimental models are presented in fig. A.2. Based on these training and validation results the following observations are made and commentary is provided for future research in the development of satellite image and CNN based forecast models:

- The RMSE accuracy results show that the custom designed shallow CNN of model C is superior to models A and B with the transferred VGG16 CNN. The transfer of the shallow layer general filters and deep layer specific filters of VGG16 does not aid the forecast accuracy for this specific task or provide a good starting point. Possibly the model task, taking into consideration the connection to the LSTM, is too different from the task of the pre-trained VGG16 CNN. Additionally, the superior performance of the custom designed model C indicates that the target dataset is possibly large enough to train the filters from scratch and transfer learning is not necessarily needed.
- Transfer learning is often applied when the development and training of new models requires vast amount of computational and time resources. The training time of one epoch on the available virtual-machine configuration is about 50, 90 and 25 minutes for the models A, B and C, respectively. Even though the transferred convolutional layers of model A are non-trainable, the time to perform 1 epoch is double the time it takes to train 1 epoch of model C. It is common to transfer a base model which is pre-trained for a challenging and broad image classification task, hence it is often a complex and deep network. Due to this characteristic, even when the fine-tuning of layers is avoided, there is still the need for significant computational resources. In case fine-tuning of deeper layers is performed, as for model B, the training time increases significantly. Training only the last convolutional layer in model B already adds 40 minutes to the training time and training the last three convolutional layers would add 7 hours to the training time on the available configuration. Taking into consideration the training time and required computational resources, the custom design of a shallow CNN seems to be the better approach in this case.
- It is observed that fine-tuning the last convolutional layer in model B results in a slower training process compared to model A and at the end of the training process the same validation RMSE. Additionally, as the number of parameters in the last convolutional layer of model B is relatively small compared to the dataset, overfitting due to fine-tuning does not seem to be a problem in this case. The complex connection of the CNN and LSTM network makes it difficult to interpret the training and validation loss of the models. Hence it is advised to validate the results and get a better understanding of the fine-tuning of layers using a simpler regression model with several dense layers instead of the encoder-decoder LSTM network.

Appendix B: Model 1 script

```
# -*- coding: utf-8 -*- #
"""
Created on Thu Okt 26 08:43:15 2020

@author: Gijs van Ouwerkerk
"""
# General #
import pandas as pd
import numpy as np
from numpy import array
import datetime
import pathlib
import math
# Created modules #
from load_data import load_data
# Deep learning #
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import RepeatVector
from keras.layers import TimeDistributed
from keras.callbacks import History
from keras.callbacks import LearningRateScheduler
from keras import metrics
from keras.callbacks import ModelCheckpoint
# Neptune #
import neptune
from neptunecontrib.monitoring.keras import NeptuneMonitor

#%% Define model functions #


def split_dataset(df, N_enddays_in_test, n_out):
    """
    Parameters
    ----------
    df : input dataframe
    N_enddays_in_test : days in validation dataset
    n_out : forecast output length (12 steps = 3 hours)

    Returns
    -------
    train : model training dataset
    test : model validation dataset
    test_start : start date of validation set
    test_end : end data of validation set
    """
    # Add normalized day-ahead forecast #
    df['norm_dayahead'] = (df['dayahead']/df['maxsolar']).fillna(0).replace(
        [np.inf, -np.inf], 0).clip(0,1)
    # Translate day-ahead forecast to input feature #
    df['sh_norm_dayahead'] = df['norm_dayahead'].shift(-n_out,axis=0).fillna(0)
    # Normalize QOD #
    df['quarter'] = abs(((np.cos(2 * math.pi * df['quarter'] / df['quarter'].max()
                                 ) + 1)/2)-1)
    # Normalize DOY #
    df['day'] = (np.cos(2 * math.pi * df['day'] / df['day'].max()) + 1)/2
    # Return start and end date of validation dataset #
    test_start = df.index.astype(str)[-N_enddays_in_test*96]
```

```
    test_end = df.index.astype(str)[-1]
    # Data to array of objects #
    data = df.values
    # Split data into train and validation dataset #
    train, test = data[0:-N_enddays_in_test*96], data[-N_enddays_in_test*96:]
    return train, test, test_start, test_end

def create_training_set(train, n_input, n_out):
    """
    Parameters
    ----------
    train : training dataset
    n_input : number of input timesteps
    n_out : number of output timesetps

    Returns
    -------
    LSTM model training input and output sets
    """
    # Select input features #
    input_cols = [0, 1, 9, 11]
    # Select target feature #
    predictor_cols = 9
    # Create lists to fill with X and y train data #
    X_train_steps, y_train_steps = list(), list()
    start_in = 0
    # Loop over history to create dataset [samples, timesteps, features] #
    for _ in range(len(train)):
        end_in = start_in + n_input
        end_out = end_in + n_out
        if end_out <= len(train):
            X_train_steps.append(train[start_in:end_in, input_cols])
            y_train_steps.append(train[end_in:end_out, predictor_cols])
        start_in += 1
    return array(X_train_steps), array(y_train_steps)

def create_validation_set(test, n_input, n_out):
    """
    Parameters
    ----------
    test : validation dataset
    n_input : number of input timesteps
    n_out : number of output timesetps

    Returns
    -------
    LSTM model validation input and output sets
    """
    # Select input features #
    input_cols = [0, 1, 9, 11]
    # Select target feature #
    predictor_cols = 9
    # Create lists to fill with X and y validation data #
    X_val_steps, y_val_steps = list(), list()
    start_in = 0
    # Loop over history to create dataset [samples, timesteps, features] #
    for _ in range(len(test)):
        end_in = start_in + n_input
        end_out = end_in + n_out
        if end_out <= len(test):
            X_val_steps.append(test[start_in:end_in, input_cols])
            y_val_steps.append(test[end_in:end_out, predictor_cols])
        start_in += 1
```

```
    return array(X_val_steps), array(y_val_steps)

def build_model(train, test, n_input, n_out, model_save_name, para_dict):
    """
    Parameters
    ----------
    train : Training dataset
    test : Validation dataset
    n_input : Number of input timesteps
    n_out : Number of output timesteps
    model_save_name : Model name
    para_dict : Information for datalogging

    Returns
    -------
    model: LSTM model
    """
    # Create object for model callbacks #
    history_model = History()
    # Prepare training and validation datasets #
    X_train_steps, y_train_steps = create_training_set(train, n_input, n_out)
    X_val_steps, y_val_steps = create_validation_set(test, n_input, n_out)
    # Reshape dataset output into [samples, timesteps, features] #
    y_train_steps = y_train_steps.reshape((y_train_steps.shape[0],
                                           y_train_steps.shape[1], 1))
    y_val_steps = y_val_steps.reshape((y_val_steps.shape[0],
                                       y_val_steps.shape[1], 1))
    # Define training hyperparameters #
    verbose, epochs, batch_size = 1, 50, 96
    n_timesteps = X_train_steps.shape[1]
    n_features = X_train_steps.shape[2]
    n_outputs = y_train_steps.shape[1]
    # Learn rate schedule #
    initial_learning_rate = 0.002
    drop_rate_Nep = 0.5
    epochs_drop_Nep = 5
    def lr_step_decay(epoch, lr):
        drop_rate = drop_rate_Nep
        epochs_drop = epochs_drop_Nep
        return initial_learning_rate * math.pow(drop_rate,
                                                math.floor(epoch/epochs_drop))
    # Neptune log: Add extra parameters for neptune reference #
    para_extra = {'batch_size': batch_size,
                  'epochs': epochs,
                  'optimizer': 'Adam',
                  'initial_LR': initial_learning_rate,
                  'drop_rate': drop_rate_Nep,
                  'epochs_drop': epochs_drop_Nep}
    para_dict.update(para_extra)
    # Neptune log: Create experiment #
    neptune.create_experiment(name=model_save_name, params=para_dict,
                              upload_source_files=('Model_1.py'),
                              tags=['model_1'])
    # Define deep learning model #
    model = Sequential()
    # Encoder #
    model.add(LSTM(200, return_sequences=True,
                   input_shape=(n_timesteps, n_features)))
    model.add(LSTM(200, return_sequences=True))
    model.add(LSTM(200, return_sequences=False))
    model.add(RepeatVector(n_outputs))
    # Decoder #
    model.add(LSTM(200, return_sequences=True))
```

```
    model.add(LSTM(200, return_sequences=True))
    model.add(LSTM(200, return_sequences=True))
    model.add(TimeDistributed(Dense(200, activation='relu',
                                    kernel_initializer='he_normal')))
    model.add(TimeDistributed(Dense(100, activation='relu',
                                    kernel_initializer='he_normal')))
    model.add(TimeDistributed(Dense(1)))
    # Compile model with additional hyperparameter selection #
    model.compile(loss='mse', optimizer='adam',
                  metrics=['mse','mae', metrics.RootMeanSquaredError()])
    # Model checkpoint save #
    path = pathlib.Path.cwd().joinpath('Saved_models', 'Model_1', model_save_name)
    path.mkdir(parents=True, exist_ok=True)
    path = path.joinpath('best_model.h5')
    checkpoint = ModelCheckpoint(path, monitor='val_root_mean_squared_error',
                                 verbose=1, save_best_only=True, mode='min')
    # Neptune log: save model summary #
    model.summary(print_fn=lambda x: neptune.log_text('model_summary', x))
    # Fit model #
    model.fit(X_train_steps, y_train_steps,
              validation_data=(X_val_steps, y_val_steps),
              epochs=epochs, batch_size=batch_size, verbose=verbose,
              callbacks=[history_model, NeptuneMonitor(),
                         LearningRateScheduler(lr_step_decay, verbose=1),
                         checkpoint])

    # Save model to local folder #
    path = pathlib.Path.cwd().joinpath('Saved_models', 'Model_1', model_save_name)
    path = path.joinpath('final_model.h5')
    model.save(path)
    # Neptune log: save model to neptune #
    neptune.log_artifact(str(path))
    return model

def forecast(model, history, n_input):
    """
    Parameters
    ----------
    model : Trained model
    history : Set containing historic input feature data
    n_input : Number of input timesteps

    Returns
    -------
    y_hat : predictions for 1 forecast run
    """
    # Flatten data #
    data = array(history)
    # Select input features #
    input_cols = [0, 1, 9, 11]
    # Retrieve last input feature observations #
    input_x = data[-n_input:, input_cols]
    input_x = input_x.reshape((1, input_x.shape[0], input_x.shape[1]))
    # Generate forecast #
    y_hat = model.predict(input_x, verbose=0)
    y_hat = y_hat[0]
    return y_hat


def make_forecast(model, train, test, n_input):
    """
    Parameters
    ----------
```

```
    model : Trained model
    train : Training set for history
    test : Validation set for predictions
    n_input : Number of input timesteps

    Returns
    -------
    predictions : Complete forecast for validation set
    """
    # History list of quarterly data #
    history = [x for x in train]
    # Walk-forward loop over validation dataset #
    predictions = list()
    for i in range(len(test)):
        # Prediction for 1 forecast #
        y_hat_sequence = forecast(model, history, n_input)
        # Store the prediction #
        predictions.append(y_hat_sequence)
        # Get real observation and add to history for predicting next quarter #
        history.append(test[i, :])
    # Compile predictions #
    predictions = array(predictions)
    return predictions


def predictions_to_datetime(test_start, test_end, predictions, n_out):
    """
    Parameters
    ----------
    test_start : Validation dataset start time
    test_end : Validation dataset end time
    predictions : Generated predictions for validation dataset
    n_out : Forecast output timesteps

    Returns
    -------
    df_predictions : Datetime based dataframe with prediction data
    """
    # Validation dataset with datetime index #
    test_index = pd.date_range(test_start,test_end,freq='15T')
    # Create empty dataframe to fill #
    df_predictions = pd.DataFrame({})
    # Loop over prediction instances and n_output steps #
    for i in range(0,len(predictions)):
        t_step = range(1,n_out+1)
        index_step = pd.date_range(test_index[i], periods=n_out, freq='15T')
        fill = pd.DataFrame(t_step,index_step)
        # add column with moment of initiation 't=0' #
        fill['inititation_LSTM'] = test_index[i] - datetime.timedelta(minutes=15)
        df_predictions = df_predictions.append(fill)
    # Reshape predictions #
    df_predictions['predictions'] = predictions.reshape(
        predictions.shape[0]*predictions.shape[1],1)
    return df_predictions


#%% Run model  #

#================================== Variables ==================================#
# Load dataset with range: #
start = '2015-01-01 00:00:00'
end = '2020-12-31 23:45:00'
# model variables #
```

```
n_input = 12
n_out = 12
N_enddays_in_test = 366
# model save details #
model_save_name = 'test'
model_N = '2.58'
# Max solar #
maxsolarmodel = 'maxsolar23_2015_2020'
#================================== Neptune Data log ============================#
para_dict = {'Data_start': start,
          'Data_end': end,
          'Days_val':N_enddays_in_test,
          'n_input': n_input,
          'n_output': n_out,
          'maxsolar': maxsolarmodel,
          'Save_name': model_save_name,
          'CNN': 'none',
          'model_N': model_N}
#================================== RUN ========================================#
# Load data, make model, evaluate #
df = load_data(start,end, maxsolarmodel)
train, test, test_start, test_end = split_dataset(df, N_enddays_in_test, n_out)
model = build_model(train, test, n_input, n_out, model_save_name, para_dict)
predictions = make_forecast(model, train, test, n_input)
df_predictions = predictions_to_datetime(test_start, test_end, predictions, n_out)
```

Appendix C: Model 2 script

```
# -*- coding: utf-8 -*- #
"""
Created on Thu Nov 26 14:43:15 2020

@author: Gijs van Ouwerkerk
"""
# General #
import pandas as pd
import numpy as np
import pathlib
import datetime
from numpy import load
from numpy import array
import math
# Created modules #
from load_data import load_data
from time_generator import TimeseriesGenerator
# Deep learning #
from keras.models import load_model
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import RepeatVector
from keras.layers import TimeDistributed
from keras.callbacks import History
from keras.callbacks import LearningRateScheduler
from keras import metrics
from keras.callbacks import ModelCheckpoint
from keras.models import Model
from keras.layers import Input
from keras.layers.merge import concatenate
# Neptune #
import neptune
from neptunecontrib.monitoring.keras import NeptuneMonitor


#%% Define model functions #


def load_img_dataset(dataset_name):
    """
    Parameters
    ----------
    dataset_name : External satellite image dataset
    Returns
    -------
    X_data : Return satellite dataset as array of images
    """
    # Open path of stored data set #
    load_path = pathlib.Path(r"C:").resolve().joinpath(
        'Users','gijs','Intraday_solar_model', 'Data', 'Satellite_arrays',
        dataset_name)
    # Load image data NpzFile #
    data = load(load_path)
    # Extract array from Npz type file #
    X_data = data['arr_0']
    return X_data

def split_dataset(df, X_data, N_enddays_in_test):
```

```
    """
    Parameters
    ----------
    df : input dataframe
    N_enddays_in_test : days in validation dataset
    n_out : forecast output length (12 steps = 3 hours)

    Returns
    -------
    train : model training dataset
    test : model validation dataset
    test_start : start date of validation set
    test_end : end data of validation set
    """
    # Add normalized day-ahead forecast #
    df['norm_dayahead'] = (df['dayahead']/df['maxsolar']).fillna(0).replace(
        [np.inf, -np.inf], 0).clip(0,1)
    # Translate day-ahead forecast to input feature #
    df['sh_norm_dayahead'] = df['norm_dayahead'].shift(-n_out,axis=0).fillna(0)
    # Normalize QOD #
    df['quarter'] = abs(((np.cos(2 * math.pi * df['quarter'] / df['quarter'].max()
                                 ) + 1)/2)-1)
    # Normalize DOY #
    df['day'] = (np.cos(2 * math.pi * df['day'] / df['day'].max()) + 1)/2
    # Normalize RBG channels in satellite image dataset #
    X_data = (X_data/255).astype('float32')
    # Return start and end date of validation dataset #
    test_start = df.index.astype(str)[-N_enddays_in_test*96]
    test_end = df.index.astype(str)[-1]
    # Return targed data as array #
    y_data = df.values.astype('float32')
    # Split y data into test and train #
    y_train, y_test = y_data[0:-N_enddays_in_test*96],
    y_data[-N_enddays_in_test*96:]
    # Split X data into test and train (images) #
    X_train, X_test = X_data[0:-N_enddays_in_test*96],
    X_data[-N_enddays_in_test*96:]
    return y_train, y_test, X_train, X_test, test_start, test_end

def create_training_set(y_train, n_input, n_out):
    """
    Parameters
    ----------
    y_train : training dataset
    n_input : number of input timesteps
    n_out : number of output timesetps

    Returns
    -------
    Array of target variable in sequence training set
    """
    # Select target feature normalized power #
    predictor_cols = 9
    # Create list to fill with y train data #
    y_train_steps = list()
    start_in = 0
    # Loop over history to create dataset [samples, timesteps, target] #
    for _ in range(len(y_train)):
        end_in = start_in + n_input
        end_out = end_in + n_out
        if end_out <= len(y_train):
            y_train_steps.append(y_train[end_in:end_out, predictor_cols])
        start_in += 1
```

```
    return array(y_train_steps)

def create_validation_set(y_test, n_input, n_out):
    """
    Parameters
    ----------
    y_test : validation dataset
    n_input : number of input timesteps
    n_out : number of output timesetps

    Returns
    -------
    Array of target variable in sequence validation set
    """
    # Select target feature normalized power #
    predictor_cols = 9
    # Create list to fill with y validation data #
    y_val_steps = list()
    start_in = 0
    # Loop over history to create dataset [samples, timesteps, target] #
    for _ in range(len(y_test)):
        end_in = start_in + n_input
        end_out = end_in + n_out
        if end_out <= len(y_test):
            y_val_steps.append(y_test[end_in:end_out, predictor_cols])
        start_in += 1
    return array(y_val_steps)

def build_model(y_train, y_test, X_train, X_test, y_train_steps, y_val_steps,
                n_input, n_out, pixels_h, pixels_b, channels, model_save_name,
                para_dict):
    '''
    Parameters
    ----------
    y_train: non-sequalized data as predictor training set
    y_test: non-sequalized data as predictor test set
    X_train: Satellite image training dataset
    X_test: Satellite image test dataset
    y_train_steps : Sequence of training target data
    y_val_steps : Sequence of validation target data
    n_input : Number of input timesteps
    n_out : Number of output timesteps
    pixels_h : Vertical pixels satellite images
    pixels_b : Horizontal pixels satellite images
    channels : Number of channels in satellite images
    model_save_name : Model name
    para_dict : Information for datalogging

    Returns
    -------
    model: CNN-LSTM model
    '''
    # Create object for model callbacks #
    history_model = History()
    # Define training hyperparameters #
    verbose, epochs, batch_size = 1, 50, 96
    # Prepare training and validation datasets via custom generator #
    # Create training set #
    y_train_steps_long = np.concatenate((np.zeros(shape=(
        n_input,n_out)), y_train_steps, np.zeros(shape=(n_out-1,n_out))), axis=0)
    y_train_steps_long = y_train_steps_long.reshape((
        y_train_steps_long.shape[0], y_train_steps_long.shape[1], 1))
    train_generator = TimeseriesGenerator(
```

```
        X_train, y_train[:,[0, 1, 11]], y_train_steps_long, length=n_input,
        batch_size=batch_size)
    # Create validation set generator #
    y_val_steps_long = np.concatenate((np.zeros(shape=(
        n_input,n_out)), y_val_steps, np.zeros(shape=(n_out-1,n_out))), axis=0)
    y_val_steps_long = y_val_steps_long.reshape((
        y_val_steps_long.shape[0], y_val_steps_long.shape[1], 1))
    val_generator = TimeseriesGenerator(
        X_test, y_test[:,[0, 1, 11]], y_val_steps_long, length=n_input,
        batch_size=batch_size)
    # Learn rate schedule generator #
    initial_learning_rate = 0.002
    drop_rate_Nep = 0.5
    epochs_drop_Nep = 5
    def lr_step_decay(epoch, lr):
        drop_rate = drop_rate_Nep
        epochs_drop = epochs_drop_Nep
        return initial_learning_rate * math.pow(drop_rate,
                                                math.floor(epoch/epochs_drop))
    # Neptune log: Add extra parameters for neptune reference #
    para_extra = {'batch_size': batch_size,
                  'epochs': epochs,
                  'optimizer': 'Adam',
                  'initial_LR': initial_learning_rate,
                  'drop_rate': drop_rate_Nep,
                  'epochs_drop': epochs_drop_Nep}
    para_dict.update(para_extra)
    # Neptune log: Create experiment #
    neptune.create_experiment(name=model_save_name, params=para_dict,
                              upload_source_files=('Model_2.py'),
                              tags=['Model_2'])

    # Import pre-trained model 2A #
    path = pathlib.Path.cwd().joinpath('Saved_models', 'Model_2A.h5')
    own_cnn = load_model(path)
    # Set layers to non-trainable #
    for layer in own_cnn.layers:
        layer.trainable = False
    # Define deep learning model #
    # Satellite image datastream #
    visible1 = Input(shape=(n_input, pixels_h, pixels_b, channels))
    flat1 = own_cnn(visible1)
    # Generation datastream #
    visible2 = Input(shape=(n_input,3))
    # Merge two models #
    merge = concatenate([flat1, visible2])
    # Encoder - LSTM #
    LSTM1 = LSTM(200, activation='relu', return_sequences=True)(merge)
    LSTM2 = LSTM(200, activation='relu', return_sequences=True)(LSTM1)
    LSTM3 = LSTM(200, activation='relu', return_sequences=False)(LSTM2)
    # Decoder - LSTM #
    Repeater = RepeatVector(n_out)(LSTM3)
    LSTM4 = LSTM(200, activation='relu', return_sequences=True)(Repeater)
    LSTM5 = LSTM(200, activation='relu', return_sequences=True)(LSTM4)
    LSTM6 = LSTM(200, activation='relu', return_sequences=True)(LSTM5)
    hidden1 = TimeDistributed(Dense(200, activation='relu',
                                    kernel_initializer='he_normal'))(LSTM6)
    hidden2 = TimeDistributed(Dense(100, activation='relu',
                                    kernel_initializer='he_normal'))(hidden1)
    output = TimeDistributed(Dense(1))(hidden2)
    # Define inputs and outputs #
    model = Model(inputs=[visible1, visible2], outputs=output)
    # Compile model with additional hyperparameter selection #
```

```
    model.compile(loss='mse', optimizer='adam',
                  metrics=['mse','mae', metrics.RootMeanSquaredError()])
    # Model checkpoint save #
    path = pathlib.Path.cwd().joinpath('Saved_models', 'Model_2', model_save_name)
    path.mkdir(parents=True, exist_ok=True)
    path = path.joinpath('best_model.h5')
    checkpoint = ModelCheckpoint(path, monitor='val_root_mean_squared_error',
                                 verbose=1, save_best_only=True, mode='min')
    # Neptune log: save model summary #
    model.summary(print_fn=lambda x: neptune.log_text('model_summary', x))
    # Fit model #
    model.fit(train_generator, steps_per_epoch=len(train_generator),
              validation_data=val_generator, validation_steps=len(val_generator),
              epochs=epochs, verbose=verbose,
              callbacks=[history_model, NeptuneMonitor(),
                         LearningRateScheduler(lr_step_decay, verbose=1),
                         checkpoint])
    # Save model to local folder #
    path = pathlib.Path.cwd().joinpath('Saved_models', 'Model_2', model_save_name)
    path = path.joinpath('final_model.h5')
    model.save(path)
    print("Saved model to disk")
    # Neptune log: save model to neptune #
    neptune.log_artifact(str(path))
    return model

def forecast(model, X_history, X_history_extra, n_input):
    """
    Parameters
    ----------
    model : Trained model
    X_history : Dataset of passed satellite images
    X_history_extra : Dataset of other passed features
    n_input : Number of output timesteps

    Returns
    -------
    y_hat : predictions for 1 forecast run

    """
    # Flatten data stream1 #
    data = array(X_history)
    # Retrieve last satellite image observation as sequence #
    input_x = data[-n_input:]
    input_x = input_x.reshape((1, input_x.shape[0], input_x.shape[1],
                               input_x.shape[2], input_x.shape[3]))
    # flatten data stream 2 #
    data2 = array(X_history_extra)
    # Retrieve last solar PV power feed-in data as sequence #
    input_x_extra = data2[-n_input:]
    input_x_extra = input_x_extra.reshape(1, input_x_extra.shape[0],
                                          input_x_extra.shape[1])
    # Generate forecast  #
    y_hat = model.predict([input_x, input_x_extra], verbose=0)
    y_hat = y_hat[0]
    return y_hat

def make_forecast(model, X_train, X_test, y_test, n_input):
    """
    Parameters
    ----------
    model : Trained model
    X_train : Image training set for history
```

```
    X_test : Generation dataset for history
    y_test : Real observations for addition to history
    n_input : Number of input timesteps

    Returns
    -------
    predictions : Complete forecast for validation set
    """
    # X_history is a list of quarterly images #
    X_history = [x for x in X_train[-n_input:]]
    # X_history is a list of quarterly other imput features #
    input_cols = [0, 1, 11]
    X_history_extra = [x for x in y_train[-n_input:, input_cols]]
    # Walk-forward loop over validation dataset #
    predictions = list()
    for i in range(len(X_test)):
        # Prediction for 1 forecast #
        y_hat_sequence = forecast(model, X_history, X_history_extra, n_input)
        # Store the predictions #
        predictions.append(y_hat_sequence)
        # Get real observation and add to X_history for predicting next quarter #
        X_history.append(X_test[i,:,:,:])
        # Get real observation of norm power and add to X_history_extra #
        X_history_extra.append(y_test[i, input_cols])
        # Remove quarter data not used anymore in forecast for RAM saving #
        X_history.pop(0)
    # compile predictions #
    predictions = array(predictions)
    return predictions

def predictions_to_datetime(test_start, test_end, predictions, n_out):
    """
    Parameters
    ----------
    test_start : Validation dataset start time
    test_end : Validation dataset end time
    predictions : Generated predictions for validation dataset
    n_out : Forecast output timesteps

    Returns
    -------
    df_predictions : Datetime based dataframe with prediction data
    """
    # Validation dataset with datetime index #
    test_index = pd.date_range(test_start,test_end,freq='15T')
    # Create empty dataframe to fill #
    df_predictions = pd.DataFrame({})
    # Loop over prediction instances and n_output steps #
    for i in range(0,len(predictions)):
        t_step = range(1,n_out+1)
        index_step = pd.date_range(test_index[i], periods=n_out, freq='15T')
        fill = pd.DataFrame(t_step,index_step)
        # add column with moment of initiation 't=0' #
        fill['inititation_CNNLSTM'] = test_index[i]-datetime.timedelta(minutes=15)
        df_predictions = df_predictions.append(fill)
    # Reshape predictions #
    df_predictions['predictions'] = predictions.reshape(
        predictions.shape[0]*predictions.shape[1],1)
    return df_predictions


#%% Run model  #

```

```
#================================== Variables ==================================#
# Load dataset with range: #
start = '2015-01-01 00:00:00'
end = '2020-12-31 23:45:00'
# model variables #
n_input = 12
n_out = 12
dataset_name = '2015-01-01_tm_2020-12-31_40_32.npz'
pixels_h = 40
pixels_b = 32
channels = 3
N_enddays_in_test = 366
# model save details #
model_save_name = 'test'
model_N = '7.32'
# Max solar #
maxsolarmodel = 'maxsolar23_2015_2020'
#================================== Neptune Data log ============================#
para_dict = {'Data_start': start,
          'Data_end': end,
          'Days_val':N_enddays_in_test,
          'n_input': n_input,
          'n_output': n_out,
          'maxsolar': maxsolarmodel,
          'Save_name': model_save_name,
          'CNN': 'Own',
          'model_N': model_N}
#================================== RUN ========================================#
# Load data, make model, evaluate #
X_data = load_img_dataset(dataset_name)
df = load_data(start,end, maxsolarmodel)
y_train, y_test, X_train, X_test, test_start, test_end = split_dataset(
    df, X_data, N_enddays_in_test)
y_train_steps = create_training_set(y_train, n_input, n_out)
y_val_steps = create_validation_set(y_test, n_input, n_out)
model = build_model(y_train, y_test, X_train, X_test, y_train_steps,
                    y_val_steps, n_input, n_out, pixels_h, pixels_b, channels,
                    model_save_name, para_dict)
predictions = make_forecast(model, X_train, X_test, y_test, n_input)
df_predictions = predictions_to_datetime(test_start, test_end, predictions, n_out)
```

Appendix D: Model 2A script

```
# -*- coding: utf-8 -*- #
"""
Created on Thu April 10 14:01:15 2021

@author: gijs
"""
# General #
import pandas as pd
import numpy as np
import pathlib
from numpy import load
from numpy import array
import math
# Created modules #
from load_data import load_data
# Deep learning #
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import RepeatVector
from keras.layers import TimeDistributed
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.models import Model
from keras.layers import Input
from keras.preprocessing.sequence import TimeseriesGenerator
from keras.callbacks import History
from keras.callbacks import LearningRateScheduler
from keras import metrics
from keras.callbacks import ModelCheckpoint
# Neptune #
import neptune
from neptunecontrib.monitoring.keras import NeptuneMonitor


#%% Define model functions #

def load_img_dataset(dataset_name):
    """
    Parameters
    ----------
    dataset_name : External satellite image dataset
    Returns
    -------
    X_data : Return satellite dataset as array of images
    """
    # Open path of stored data set #
    load_path = pathlib.Path(r"C:").resolve().joinpath(
        'Users','gijs','Intraday_solar_model', 'Data', 'Satellite_arrays',
        dataset_name)
    # Load image data NpzFile #
    data = load(load_path)
    # Extract array from Npz type file #
    X_data = data['arr_0']
    return X_data

def split_dataset(df, X_data, N_enddays_in_test):
```

```
    """
    Parameters
    ----------
    df : input dataframe
    N_enddays_in_test : days in validation dataset
    n_out : forecast output length (12 steps = 3 hours)

    Returns
    -------
    train : model training dataset
    test : model validation dataset
    test_start : start date of validation set
    test_end : end data of validation set
    """
    # Normalize RBG channels in satellite image dataset #
    X_data = (X_data/255).astype('float32')
    # Return start and end date of validation dataset #
    test_start = df.index.astype(str)[-N_enddays_in_test*96]
    test_end = df.index.astype(str)[-1]
    # Return targed data as array #
    y_data = df.values.astype('float32')
    # Split y data into test and train #
    y_train, y_test = y_data[0:-N_enddays_in_test*96], y_data[-N_enddays_in_test*96:]
    # Split X data into test and train (images) #
    X_train, X_test = X_data[0:-N_enddays_in_test*96], X_data[-N_enddays_in_test*96:]
    return y_train, y_test, X_train, X_test, test_start, test_end

def create_training_target(y_train, n_input, n_out):
    """
    Parameters
    ----------
    y_train : training dataset
    n_input : number of input timesteps
    n_out : number of output timesetps

    Returns
    -------
    Array of target variable in sequence training set
    """
    # Select target feature normalized power #
    predictor_cols = 9
    # Create list to fill with y train data #
    y_train_steps = list()
    start_in = 0
    # Loop over history to create dataset [samples, timesteps, target] #
    for _ in range(len(y_train)):
        end_in = start_in + n_input
        end_out = end_in + n_out
        if end_out <= len(y_train):
            y_train_steps.append(y_train[end_in:end_out, predictor_cols])
        start_in += 1
    return array(y_train_steps)

def create_validation_target(y_test, n_input, n_out):
    """
    Parameters
    ----------
    y_test : validation dataset
    n_input : number of input timesteps
    n_out : number of output timesetps

    Returns
    -------
```

```
    Array of target variable in sequence validation set
    """
    # Select target feature normalized power #
    predictor_cols = 9
    # Create list to fill with y validation data #
    y_val_steps = list()
    start_in = 0
    # Loop over history to create dataset [samples, timesteps, target] #
    for _ in range(len(y_test)):
        end_in = start_in + n_input
        end_out = end_in + n_out
        if end_out <= len(y_test):
            y_val_steps.append(y_test[end_in:end_out, predictor_cols])
        start_in += 1
    return array(y_val_steps)

def build_model(X_train, X_test, y_train_steps, y_val_steps, n_input, n_out,
                pixels_h, pixels_b, channels, model_save_name, para_dict):
    '''
    Parameters
    ----------
    X_train: Satellite image training dataset
    X_test: Satellite image test dataset
    y_train_steps : Sequence of training target data
    y_val_steps : Sequence of validation target data
    n_input : Number of input timesteps
    n_out : Number of output timesteps
    pixels_h : Vertical pixels satellite images
    pixels_b : Horizontal pixels satellite images
    channels : Number of channels in satellite images
    model_save_name : Model name
    para_dict : Information for datalogging

    Returns
    -------
    model: CNN-LSTM regression model
    '''
    # Create object for model callbacks #
    history_model = History()
    # Define training hyperparameters #
    verbose, epochs, batch_size = 1, 50, 96
    # Prepare training and validation datasets via keras utils generator #
    # Create training set generator #
    y_train_steps_long = np.concatenate((np.zeros(shape=(
        n_input+n_out,n_out)), y_train_steps[:-1,:]), axis=0)
    y_train_steps_long = y_train_steps_long.reshape((
        y_train_steps_long.shape[0], y_train_steps_long.shape[1], 1))
    generator = TimeseriesGenerator(
        X_train, y_train_steps_long, length=n_input, batch_size=batch_size)
    # Create validation set  generator #
    y_val_steps_long = np.concatenate((np.zeros(shape=(
        n_input+n_out,n_out)), y_val_steps[:-1,:]), axis=0)
    y_val_steps_long = y_val_steps_long.reshape((
        y_val_steps_long.shape[0], y_val_steps_long.shape[1], 1))
    val_generator = TimeseriesGenerator(
        X_test, y_val_steps_long, length=n_input, batch_size=batch_size)
    # Learn rate schedule generator #
    initial_learning_rate = 0.001
    drop_rate_Nep = 0.5
    epochs_drop_Nep = 15
    def lr_step_decay(epoch, lr):
        drop_rate = drop_rate_Nep
        epochs_drop = epochs_drop_Nep
```

```
        return initial_learning_rate * math.pow(drop_rate,
                                                math.floor(epoch/epochs_drop))
    # Neptune log: Add extra parameters for neptune reference #
    para_extra = {'batch_size': batch_size,
              'epochs': epochs,
              'optimizer': 'Adam',
              'initial_LR': initial_learning_rate,
              'drop_rate': drop_rate_Nep,
              'epochs_drop': epochs_drop_Nep
              }
    para_dict.update(para_extra)
    # Neptune log: Create experiment #
    neptune.create_experiment(name=model_save_name, params=para_dict,
                              upload_source_files=('Model_2A.py'),
                              tags=['Model_2A'])
    # Built CNN model #
    own_cnn = Sequential()
    own_cnn.add(Conv2D(32, (3, 3), activation='relu',
                       kernel_initializer='he_uniform', padding='same',
                       input_shape=(pixels_h, pixels_b,channels)))
    own_cnn.add(MaxPooling2D((2, 2)))
    own_cnn.add(Conv2D(32, (3, 3), activation='relu',
                       kernel_initializer='he_uniform', padding='same'))
    own_cnn.add(MaxPooling2D((2, 2)))
    own_cnn.add(Conv2D(32, (3, 3), activation='relu',
                       kernel_initializer='he_uniform', padding='same'))
    own_cnn.add(MaxPooling2D((2, 2)))
    own_cnn.add(Conv2D(32, (3, 3), activation='relu',
                       kernel_initializer='he_uniform', padding='same'))
    own_cnn.add(MaxPooling2D((2, 2)))
    # Define CNN model in complete architecture #
    visible1 = Input(shape=(n_input, pixels_h, pixels_b, channels))
    cnn = TimeDistributed(own_cnn)(visible1)
    flat1 = TimeDistributed(Flatten())(cnn)
    # Encoder - LSTM #
    LSTM1 = LSTM(200, activation='relu', return_sequences=True)(flat1)
    LSTM2 = LSTM(200, activation='relu', return_sequences=True)(LSTM1)
    LSTM3 = LSTM(200, activation='relu', return_sequences=False)(LSTM2)
    # Decoder - LSTM #
    Repeater = RepeatVector(n_out)(LSTM3)
    LSTM4 = LSTM(200, activation='relu', return_sequences=True)(Repeater)
    LSTM5 = LSTM(200, activation='relu', return_sequences=True)(LSTM4)
    LSTM6 = LSTM(200, activation='relu', return_sequences=True)(LSTM5)
    hidden1 = TimeDistributed(Dense(200, activation='relu',
                                    kernel_initializer='he_normal'))(LSTM6)
    hidden2 = TimeDistributed(Dense(100, activation='relu',
                                    kernel_initializer='he_normal'))(hidden1)
    output = TimeDistributed(Dense(1))(hidden2)
    # Define inputs and outputs #
    model = Model(inputs=visible1, outputs=output)
    # Compile model with additional hyperparameter selection #
    model.compile(loss='mse', optimizer='adam',
                  metrics=['mse','mae', metrics.RootMeanSquaredError()])
    # Model checkpoint save #
    path = pathlib.Path.cwd().joinpath('Saved_models', 'Model_2A', model_save_name)
    path.mkdir(parents=True, exist_ok=True)
    path = path.joinpath('best_model.h5')
    checkpoint = ModelCheckpoint(path, monitor='val_root_mean_squared_error',
                                 verbose=1, save_best_only=True, mode='min')
    # Neptune log: save model summary #
    model.summary(print_fn=lambda x: neptune.log_text('model_summary', x))
    # Fit model #
    model.fit(generator, steps_per_epoch=len(generator),
```

```
248                 validation_data=val_generator, validation_steps=len(val_generator),
249                 epochs=epochs, batch_size=batch_size, verbose=verbose,
250                 callbacks=[history_model, NeptuneMonitor(),
251                            LearningRateScheduler(lr_step_decay, verbose=1),
252                            checkpoint])
253     # Save model to local folder #
254     path = pathlib.Path.cwd().joinpath('Saved_models', 'Model_2A', model_save_name)
255     path = path.joinpath('final_model.h5')
256     model.save(path)
257     print("Saved model to disk")
258     # Neptune log: save model to neptune #
259     neptune.log_artifact(str(path))
260     return model
261 
262 #%% Run model  #
263 
264 #================================== Variables ==================================#
265 # Load dataset with range: #
266 start = '2015-01-01 00:00:00'
267 end = '2020-12-31 23:45:00'
268 # model variables #
269 n_input = 12
270 n_out = 12
271 dataset_name = '2015-01-01_tm_2020-12-31_40_32.npz'
272 
273 pixels_h = 40
274 pixels_b = 32
275 channels = 3
276 N_enddays_in_test = 366
277 # model save details #
278 model_save_name = 'Sat_test'
279 model_N = '6.37'
280 # Max solar #
281 maxsolarmodel = 'maxsolar23_2015_2020'
282 #================================== Neptune Data log ============================#
283 para_dict = {'Data_start': start,
284           'Data_end': end,
285           'Days_val':N_enddays_in_test,
286           'n_input': n_input,
287           'n_output': n_out,
288           'maxsolar': maxsolarmodel,
289           'Save_name': model_save_name,
290           'CNN': 'VGG16',
291           'model_N': model_N}
292 #================================== RUN ========================================#
293 # Load data, make model, evaluate #
294 X_data = load_img_dataset(dataset_name)
295 df = load_data(start,end, maxsolarmodel)
296 y_train, y_test, X_train, X_test, test_start, test_end = split_dataset(df, X_data, N_enddays_i
297 y_train_steps = create_training_target(y_train, n_input, n_out)
298 y_val_steps = create_training_target_validate(y_test, n_input, n_out)
299 model = build_model(X_train, X_test, y_train_steps, y_val_steps, n_input, n_out, pixels_h, pix
```

Appendix E: Data generator script

```
"""
Created on Fri Mar  5 09:37:30 2021

@author: Gijs van Ouwerkerk
"""
import numpy as np
import json
import keras

class TimeseriesGenerator(keras.utils.Sequence):
    def __init__(self, img, data, targets, length,
                 sampling_rate=1,
                 stride=1,
                 start_index=0,
                 end_index=None,
                 shuffle=False,
                 reverse=False,
                 batch_size=128):

        if len(data) != len(targets):
            raise ValueError('Image and sequence data must'+
                             'have same length as target data')

        self.img = img
        self.data = data
        self.targets = targets
        self.length = length
        self.sampling_rate = sampling_rate
        self.stride = stride
        self.start_index = start_index + length
        if end_index is None:
            end_index = len(data) - 1
        self.end_index = end_index
        self.shuffle = shuffle
        self.reverse = reverse
        self.batch_size = batch_size

        if self.start_index > self.end_index:
            raise ValueError('`start_index+length=%i > end_index=%i')

    def __len__(self):
        return (self.end_index - self.start_index +
                self.batch_size * self.stride) // (self.batch_size * self.stride)

    def __getitem__(self, index):
        if self.shuffle:
            rows = np.random.randint(
                self.start_index, self.end_index + 1, size=self.batch_size)
        else:
            i = self.start_index + self.batch_size * self.stride * index
            rows = np.arange(i, min(i + self.batch_size *
                                    self.stride, self.end_index + 1), self.stride)

        samples2 = np.array([self.img[row - self.length:row:self.sampling_rate]
                             for row in rows])
        samples = np.array([self.data[row - self.length:row:self.sampling_rate]
                             for row in rows])
        targets = np.array([self.targets[row] for row in rows])
```

```
59
60          if self.reverse:
61              return samples[:, ::-1, ...], targets
62          return [samples2, samples], targets
63
64      def get_config(self):
65          img = self.img
66          if type(self.img).__module__ == np.__name__:
67              img = self.img.tolist()
68          try:
69              json_img = json.dumps(img)
70          except TypeError:
71              raise TypeError('Satellite image data not JSON Serializable:', img)
72
73          data = self.data
74          if type(self.data).__module__ == np.__name__:
75              data = self.data.tolist()
76          try:
77              json_data = json.dumps(data)
78          except TypeError:
79              raise TypeError('Sequence data not JSON Serializable:', data)
80
81          targets = self.targets
82          if type(self.targets).__module__ == np.__name__:
83              targets = self.targets.tolist()
84          try:
85              json_targets = json.dumps(targets)
86          except TypeError:
87              raise TypeError('Targets not JSON Serializable:', targets)
88
89          return {
90              'img': json_img,
91              'data': json_data,
92              'targets': json_targets,
93              'length': self.length,
94              'sampling_rate': self.sampling_rate,
95              'stride': self.stride,
96              'start_index': self.start_index,
97              'end_index': self.end_index,
98              'shuffle': self.shuffle,
99              'reverse': self.reverse,
100             'batch_size': self.batch_size
101         }
102     def to_json(self, **kwargs):
103         config = self.get_config()
104         timeseries_generator_config = {
105             'class_name': self.__class__.__name__,
106             'config': config
107         }
108         return json.dumps(timeseries_generator_config, **kwargs)
109
110 def timeseries_generator_from_json(json_string):
111     full_config = json.loads(json_string)
112     config = full_config.get('config')
113
114     img = json.loads(config.pop('img'))
115     config['img'] = img
116     data = json.loads(config.pop('data'))
117     config['data'] = data
118     targets = json.loads(config.pop('targets'))
119     config['targets'] = targets
120     return TimeseriesGenerator(**config)
```