SPATIOTEMPORAL FORECASTING OF SOLAR POWER GENERATION WITH DEEP LEARNING

by

İsmail Toyhan Yumru

B.S., Industrial Engineering, Boğaziçi University, 2016

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Industrial Engineering

Boğaziçi University

2020

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my advisor Asst. Prof. Mustafa Gökçe Baydoğan for his patience and the continuous support of my study and research. His immense knowledge and guidance helped me throughout my studies and the course of this thesis. If it were not for his generosity and understanding, this path I took would be harder to complete.

My gratitude also extends to my colleagues Burak Tabak, Dilara Aykanat and Mert Yüksekgönül for their candid friendship, mentorship, motivation and generous help.

I would like to thank my family, especially my dear brother Zaferhan Yumru and my precious sister Nazlı Yumru for their endless love, support and understanding. Very few people are as good listeners as they are. Special thanks to Oğulcan Ekiz, for his constant support and guidance, on saturday skype sessions. And to Luci, thanks for being a little ray of sunshine everyday. It helped me a lot through the most stressful days.

Last but not least, I would like to thank my partner İlayda Zafer for being there for me through thick and thin.

I acknowledge the support of TUBITAK 2210 National Graduate Scholarship Programme during my masters study.

ABSTRACT

SPATIOTEMPORAL FORECASTING OF SOLAR POWER GENERATION WITH DEEP LEARNING

Solar power is one of the most rapidly growing carbon-free power generation solutions. It is considered as a key element in the fight against global climate crisis; however, rapid expansion in the distributed PV power, i.e. plants with less than 1 MW capacity, brings about some problems to the electricity markets. Spatially dispersed positioning of hundreds of plants cause significant variations in the power supply where trading operations depend on accurate forecasts of the future production. In this study, several deep learning techniques are implemented for the day-ahead solar power forecasting problem to predict the aggregated output of over a thousand solar stations distributed over a large area in the Central Anatolian Region. Four different architectures in the literature are adapted to the spatiotemporal numerical weather prediction (NWP) data, along with the proposed parallel locally-connected long short-term memory (PLC-LSTM) architecture. All models are put through a distributed heuristic hyperparameter tuning process using multiple graphical processing units (GPUs). Best-performing trials of each model are selected according to their validation results and compared with each other, together with persistence and an individual plant naive model as benchmarks. The results show that deep learning models work considerably well in spatiotemporal PV forecasting problem, compared to benchmarks. Also, it is seen that even simple architectures can perform close to models with a higher degree of complexity, when a good combination of parameters is obtained with a thorough search procedure. Although there is not a single dominating architecture prevailing in all kinds of performance metrics, PLC-LSTM shows promising results by finding a sweet spot of complexity between the shared-weight and fully-connected architectures, considering the bias and variance of the corresponding models.

ÖZET

UZAYZAMANSAL DERİN ÖĞRENME İLE GÜNEŞ ENERJİSİ ÜRETİMİ TAHMİNLEMESİ

Güneş enerjisi, en hızlı gelişen karbonsuz elektrik üretim yöntemlerindendir. Küresel iklim krizi ile mücadeledeki önemli rolüne karşın, dağıtık (1 MW kapasitenin altındaki) güneş enerjisi santrallerinin sayısındaki hızlı büyüme, elektrik piyasalarında bazı problemleri beraberinde getirmektedir. Coğrafi olarak geniş bir alanda konumlanmış yüzlerce santral, saatlik elektrik arzında önemli sapmalara yol açabilmektedir. Dolayısıyla, elektrik piyasası işlemlerinde ileri tarihli üretim tahminlemelerinin en yüksek isabetle gerçekleşmesi önem arz etmektedir. Bu çalışmada, Orta Anadolu bölgesinde geniş bir alana dağılmış binden fazla güneş enerjisi santralinin saatlik toplam üretim değerlerinin gün öncesinde tahminlenmesi için çözüm üreten çeşitli derin öğrenme teknikleri uygulanmıştır. Halihazırda literatürde bulunan dört farklı derin öğrenme mimarisi, bu çalışmada sunulan paralel yerel bağlantılı uzun-kısa süreli bellek ağı ile birlikte sayısal hava durumu tahminlerinin uzay-zamansal yapısına uyarlanmıştır. Bütün modeller, birden fazla grafik işlemci biriminin kullanıldığı dağıtık yapıdaki bir sezgisel hiperparametre eniyileme işleminden geçirilmiştir. Her model için en iyi çalışan deneme, doğrulama veri kümesi kullanılarak seçildikten sonra modeller kendi aralarında ve referans modellerle karşılaştırmalı olarak raporlanmıştır. Sonuçlar referanslarla kıyaslandığında derin öğrenme modellerinin uzay-zamansal güneş enerjisi tahminleme probleminde iyi çalıştığını göstermiştir.Ayrıca, iyi bir parametre seçilimi sağlandığında düşük karmaşıklıktaki modellerin daha karmaşık modellere yakın performans verebileceğini görülmüştür. Her hata ölçevi bakımından galip gelen tek bir model olmasa da, önerilen modelin yanlılık ve varyans ikilemi göz önünde bulundurulduğunda paylaşımlı parametreli ağ ve tam bağlantılı ağ arasında konumlanması gelecek vaat edicidir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$DSWRF_t$	Value of downward shortwave radiation flux at a grid point at time t
$DSWRF_{t,i,h}$	Average $DSWRF_t$ values at the grid points encircling the actual location of plant i
h	Hour
i	Plant index
L	Loss function
n	Time interval between the observation and the prediction
t	Observation index
$x_{i,h}$	Past production values of plant i at hour h, which belong to the same quantile as $DSWRF_{t,i,h}$
y_t	Actual aggregated production at time t
$\hat{y}_t$	Predicted aggregated production at time t
$\hat{y}_{t,i,h}$	Predicted production at time t and hour h of plant i
ϵ	Noise

LIST OF ACRONYMS/ABBREVIATIONS

2D	Two Dimensional
3D	Three Dimensional
AC	Alternative Current
ADAM	Adaptive Moment Estimation
ANN	Artificial Neural Networks
CNN	Convolutional Neural Network
DB	Database
DC	Direct Current
DL	Deep Learning
DPBias	Daily Percent Bias
DSWRF	Downward Shortwave Radiation Flux
DWMAPE	Daily Weighted Mean Absolute Percentage Error
FC-ANN	Fully-Connected Artificial Neural Network
FC-LSTM	Fully-Connected Long Short-Term Memory
GHI	Global Horizontal Irradiance
HPO	Hyperparameter Optimization
IEA	International Energy Agency
INM	Individual Naive Model
LASSO	Least Absolute Shrinkage and Selection Operator
LSTM	Long Short-Term Memory
MP	Market Participants
MW	Megawatt
PBias	Percent Bias
PLC-LSTM	Parallel Locally-Connected Long Short-Term Memory
PV	Photovoltaic
SGD	Stochastic Gradient Descent
SVR	Support Vector Regression
WMAPE	Weighted Mean Absolute Percentage Error

1. INTRODUCTION

Photovoltaics (PV) is one of the cleanest and cheapest ways of generating electricity. Increased cell efficiencies and reduced costs cause an ever-increasing supply in the number of installations globally. Government incentives make it feasible to invest in low capacity plants for commercial or industrial purposes. Householders are now able to generate their own electricity on their rooftops. In general, these kind of installations under 1 megawatt (MW) capacity are regarded as distributed applications, whereas larger plants are considered as utility-scale. According to International Energy Agency (IEA) renewables report [4], in 2018, distributed PV additions accounted for 40% of total PV growth worldwide and exceeded the net capacity additions of coal and nuclear combined [4]. In the main case forecast, global renewable electrical capacity increases 50%, i.e 1220 gigawatt (GW), by 2024, from 2502 GW in 2018. Solar PV, including utility-scale and distributed applications, accounts for almost 60% of all renewable capacity expansion over the forecast period, followed by wind, hydropower and bioenergy [4].

Rapid expansion in the distributed PV power brings about some problems to the electricity markets. In general, market participants (MP) provide a viable production plan to the market operator before the actual generation occurs. The imbalance between the planned, i.e. traded, and actual generation is then penalized in order to ensure the zero-balance property of the market and also to encourage MP to have a more accurate production plan. However, growing capacity of distributed solar power plants pose a risk of ever-increasing stochasticity in the electricity supply during daytime. Market participants make use of data-driven tools and processes in order to reduce the impact of this uncertainty during their daily trading operations. In general, since the primary sources of variability are mainly weather conditions, various modeling techniques are used to provide decision makers with accurate forecasts based on historical and anticipated weather data.

In its broadest sense, the research question is to predict the amount of solar power produced at a particular site or region, within a certain interval of time in the future. There are many kinds of approaches proposed in this domain that attempt to provide a robust solution to this problem. These approaches vary remarkably with respect to the specific needs of the problem holder, according to two main properties of their production setup: spatial horizon and resolution, and temporal horizon and resolution.

Spatial horizon stands for the total area spanned by the power stations whereas spatial resolution stands for the number of solar installations within that area. The time horizon, i.e. forecast horizon, stands for the time difference between the latest observation and the forecasted time interval, whereas time resolution stands for the duration of each observation within that time interval. These properties shape the design decisions during the modeling phase and in some cases bring out the necessity of some exogenous variables. For instance, for a single-site problem, it is more straightforward to come up with a forecast compared to a multi-site problem in which the stations are located at variable distances to each other. In the latter, stations can be forecasted individually and the aggregated forecast can be obtained in a bottom-up fashion, or they can be clustered into several groups in order to avoid the computational burden of individual forecasts. Similarly, a very short forecast horizon, i.e. 5 min, can be handled with an autoregressive approach whereas further horizons, i.e. 6 hours, 24 hours etc., require exogenous weather forecasts to be included in the model, in order to be able to have reliable predictions.

Existing studies in the literature attack the problem of spatiotemporal PV foresting using different techniques, such as autoregression, penalized regression, kriging and artificial neural networks (ANN), as discussed in Chapter 2. ANN-based, i.e. deep learning (DL), approaches provide high flexibility in terms of designing a problem-specific modeling layout and convenience by implicit feature engineering. Mathe et al. [5] and Chai et al. [6] exhibit the most recent deep learning work in the spatiotemporal PV forecasting literature. In the former, convolutional neural network (CNN) and long short-term memory (LSTM) are connected in a serial manner while the lat-

ter use the convolutional LSTM (ConvLSTM) model, which is initially proposed for precipitation nowcasting [3]. ConvLSTM embeds convolutions within the LSTM architecture, instead of serially connecting CNN and LSTM. This way, it offers an alternative to fully-connected LSTM (FC-LSTM) where spatial information is learned through a complex dense weight layout.

In this study, alternative machine learning algorithms are implemented to predict the aggregated output of over a thousand solar power stations with approximately 1 GW collective capacity, distributed within 70000 km^2 area. The forecasts are made in the day-ahead setup, i.e. 24-hours ahead, with each observation corresponding to an hour of total production. Weather forecasts provided by a numerical weather prediction (NWP), which is able to span the entire area, are used as exogenous variables and autoregressive effects are not taken into account due to the day-ahead setup. Models with varying levels of complexity, namely fully-connected artificial neural network (FC-ANN), convolutional neural network (CNN), fully-connected long short-term memory (LSTM) network and convolutional LSTM (ConvLSTM) network are deployed. Additionally, an architecture composed of parallel locally-connected LSTMs (PLC-LSTM) is proposed, which is specifically designed for large-area spatiotemporal learning. The proposed method splits the total area into subregional kernels, which connect the 2D spatial plane to distinct local LSTM modules. Outputs of LSTM modules are then fed into dense layers before the final output. This architecture offers an alternative to the shared-weight and fully-connected structures of ConvLSTM and FC-LSTM respectively.

This thesis is organized as follows: Chapter 2 provides a brief review of the current literature. A background information about simple benchmarks and other machine learning models studied in this thesis are provided in Chapter 3, together with brief explanations about the core concepts of artificial neural networks. Data schema, proposed model and parametrization approaches are described in Chapter 4. Design of experimentation, hyperparameter optimization methodology and search space are depicted in Chapter 5. Afterwards, results for each model are reported and

compared in Chapter 6. Finally, conclusions and potential future work are shared in the last chapter.

2. LITERATURE REVIEW

There is a vast number of studies conducted in the domain of solar power forecasting. Naturally, the starting point of the most of them is the solar irradiance, which is the electromagnetic radiation power per unit area. After all, if the effective solar irradiance received by the plane of array is known together with the cell temperature, PV power output roughly becomes a deterministic function mainly shaped by cell specifications provided by the manufacturer. Earlier studies such as [7–11] mainly concentrate on the estimation of clear sky solar irradiance, which is total irradiance at a location given that no clouds are present, using atmospheric radiative transfer models. Some of these models make use of atmospheric measurements such as water vapor, ozone, aerosol optical depth and Linke turbidity coefficient, whereas some simpler ones only use solar position and geographical location [12]. These clear sky models play a crucial role by providing an upper bound for the amount of radiation that can reach a particular location. However, sole use of clear sky models is not sufficient for most of the time and in line with the advance of the photovoltaic (PV) power application, the use of machine learning methods is getting increasingly popular. It is possible to analyze the literature using several different categorizations with respect to the target variable in question, time horizon and resolution, spatial horizon and resolution, forecasting hierarchy, type of exogenous variables used, and forecasting methodology.

The *target variable* of the solar power forecasting studies is generally either the estimation of solar irradiance [13–17] or the amount of PV power output [18–24]. The data is obtained from global horizontal irradiance (GHI) measurements of weather stations or PV power plants respectively. Literature reviews conducted in this domain take studies of both kinds into account, since the two are almost interchangeable, i.e., GHI is the main ingredient of PV power output. In addition to GHI, there are other sources of variability such as shading, soiling and dust, overheating, panel degradation and DC/AC conversion loss that can affect the PV power output. However, weather conditions are considered to be the main source of variability in the power output,

especially when the forecasted output is an aggregate of multiple power plants.

Time horizon and resolution are other aspects that differentiate the studies in the area. In many countries, electricity is traded by utility companies and power generation plants before the actual transmission occurs. Thus, it is very common for MP to perform operations for accurate day-ahead and intra-day forecasts. Therefore, selection of the time horizon of a study generally depend on the problem holder's operational needs. Similarly, time resolution of the forecasting studies are also heavily influenced by the electricity markets' structures. In Turkey, for instance, settlement of electricity trading operations are handled at a 1-hour resolution. In some studies, time resolution increases to as frequent as 15-minutes [16, 18, 19, 25, 26] or even 1-second [13–15].

Spatial horizon and resolution of the studies differ according to the number of the GHI or PV output measurement locations and the area spanned by these locations. Inman et al. [1] provides a comprehensive summary of datasets used by various solar power forecasts. Some of these studies work only on a single location whereas others involve forecasting of multiple locations either individually or at an aggregate level. This difference in the spatial horizon and resolution generally stems from the research objectives, as it is in the case of time horizon, especially from the data available at hand. However, comparison of different solar power forecasting methodologies gets harder due to this diversity in the spatial structure of the data.

Forecasting hierarchy comes into play when there are more than one location in the spatial horizon. In this case, PV power output series coming from different can be modeled individually, aggregated into several clusters, or even into a single series. Each level in the hierarchy comes with its own advantages and difficulties. For instance, working on each series individually can be an expensive method in terms of computational time whereas working on a single aggregate level may cause a loss of information which is embedded in each individual series.

Exogeneous variables are another key aspect. As mentioned earlier, weather conditions are the main source of variability in solar power forecasting. There are several types of sensory information such as satellite imaging, sky imagers, and weather stations. The first two of them are typically used for estimation of GHI at a given location, using the cloud cover at certain altitudes or positions of the clouds in the sky images, whereas in the third, devices such as pyranometers are used for the measurement of solar irradiance. Sensory data is widely used in many studies for both solar irradiance and PV power output forecasting. However, sensory data is generally not available for a multi-location problem containing hundreds of PV production plants. Another option for obtaining exogenous weather data is using a NWP model. NWP's are advantageous to sensory data in terms of accessibility, spatial coverage and availability of future forecasts. For instance, Global Forecast System (GFS) provides openly accessible numerical forecasts with a 0.25° resolution. It is a great option for the forecasting of sites where no sensory data is available, which is generally the case for developing and under-developed countries. There are many approaches that combine NWP model forecasts with a machine learning approach in order to model PV power output.

Regarding the *forecasting methodology*, there are several approaches both in statistical learning and machine learning domains that attack the problem of solar power forecasting. These approaches are listed thoroughly in a number of extensive literature surveys, including a text-mining approach which analyzes a thousand papers in the solar power forecasting domain [27]. Although most of the studies in the solar forecasting domain concentrate on single site (or single plant) forecasting, extracting the spatiotemporal relationships of multiple locations in a particular region is often essential for a network of solar power stations. In a recent work, Silva and Brito [28] lists a number of spatiotemporal solar forecasting studies that uses a variety of modeling techniques such as autoregression [25, 29], least absolute shrinkage and selection operator (LASSO) [13, 16], kriging [13, 14, 30], artificial neural networks (ANN) [25, 31, 32] and support vector regression (SVR) [29, 33]. Most of the studies conducted in the spatiotemporal domain deal with a very short term forecasting horizon, which is functional only for intra-day market operations. In spatiotemporal studies, the most common dif-

ficulty is handling the large dimensionality due to increased number of locations to be considered as inputs to the models. In addition, neighboring locations tend to have highly correlated features, hence introducing even more difficulty in parametric models. Therefore, dimensionality reduction via parameter regularization is considered in a number of approaches, i.e LASSO. In parametric models, apart from the variable selection, feature engineering is another process that affects the performance of the resulting model to a great extent. It becomes important to embed nonlinear relationships of multiple variables, possibly between the multi-step lagged features of various locations. This introduces a large number of feature engineering possibilities which could be reduced with a domain knowledge in atmospheric sciences, i.e. meteorology.

On the other hand, deep learning techniques implicitly engineer the necessary features which eventually allow working with raw data. Mathe et al. [5] propose a spatiotemporal deep learning achitecture, PVNet, which combines CNN and LSTM in a serial manner, in order to forecast aggregated PV output of Germany within the day-ahead forecast horizon using the NWP forecasts as input. In a very recent work, Chai et al. [6] implement ConvLSTM architecture, which is originally proposed for precipitation nowcasting [3], to the synthetic 5-min solar PV power measurement offered by NREL, within the short term forecast horizon, i.e 15 minutes to 1 hour. Their implementation does not utilize NWP as input. Instead, the synthetic power measurements of 56 PV power plants are spatially gridded into an 8x8 2D frame and empty grids are interpolated using the neighboring grids, in order to be able work with CNN. Convolutional Neural Networks are well known for their success in image recognition applications. The shared-weight structure of the CNN networks are convenient for reducing the number of weights and hence reducing the likelihood of overfitting. Although this property makes a perfect match for the needs of the vast majority of image recognition problems, it may have some disadvantages in spatiotemporal PV forecasting, especially those with a large area.

Firstly, empty locations where no solar generation occur are also included in the model during convolution of the filters. This is a major problem when the area of

study covers large distances containing over a thousand PV stations, as in the case of our study. In this case, the majority of the area is empty, or sparsely populated, and inclusion of these empty subregions are redundant for the forecasting of the aggregated series. Secondly, CNN is not able to differentiate the local weather characteristics of different subregions belonging in the same area in question. High variations between distant subregions might occur due to different geographical formations, micro-climate effects or simply large difference of latitude. For instance, variation in the prevailing wind direction among subregions might pose a problem for a model that takes wind direction as an input, since it produces a spatial non-linearity which is not subject to translational invariance.

Deep learning methods provide a great flexibility in designing a problem-specific solution. A number of properties such as depth, width and connectivity of layers, weight regularization, type and specifications of the optimizer, learning rate and many more can be controlled parametrically to find the model that fits best to the problem at hand. However, this flexibility brings about a high sensitivity to parameter combinations. Hyperparameter tuning becomes an important part of any deep learning study and is computationally expensive, especially when the model itself is a complex one. This is one of the major drawbacks of deep learning applications in general.

In this study, alternative deep learning architectures are implemented to be able to compare the forecasting power of fully connected, convolutional and locally ensembled architectures on the spatiotemporal solar power forecasting problem. First, fully connected and CNN architectures are implemented separately, without taking time-series property into account. After that, in order to include the time series relationships, a fully-connected LSTM and a convolutional LSTM are implemented. Both of the existing DL studies incorporate convolutional approaches, either used in a serial manner [5] or embedded in the LSTM [6]. Therefore, a parallel ensemble of multiple LSTMs is proposed to check the hypothesis if the locally connected weights can substitute the conventional methods, since there is no study considering a locally-connected ANN architecture for spatiotemporal PV power forecasting. All architectures are put through

a hyperparameter optimization process and best-performing parameter combinations of each one are compared.

3. BACKGROUND

3.1. Benchmark Models

Following methods are used as benchmarks in order to see if the proposed methods have additional predictive power compared to a simple approach.

3.1.1. Persistence Model

This is a basic forecasting approach which assumes that the hourly production values will be the same as the most recent observations,

$$\hat{y}_t = y_{t-n}$$

where y_{t-n} is the most recently observed value, $\hat{y}_t$ is the prediction and n is the time interval between the observation and the prediction.

Most of the studies in the literature use this approach as a benchmark. However, data availability is often not taken into account. In a day-ahead forecasting setup, for instance, predictions are produced the day before the actual generation occurs. Therefore the most recently available data is generally two days before the predicted day. In this study, both 24-hour and 48-hour persistence will be reported, named as PM_{24} and PM_{48} respectively, as a benchmark. However, only the latter can be considered as an actual benchmark since the former is practically not available in the day-ahead setup.

3.1.2. Individual Naive Model

Individual Naive Model (INM) approach uses weather predictions as an external predictor. Each solar power plant is predicted individually using the corresponding

median production values of the past downward shortwave radiation flux (DSWRF) quantiles.

$$\hat{y}_{t,i,h} = median\left(x_{i,h}\right) = \begin{cases} {x_{i,h}}^{(\frac{m+1}{2})} & , m \text{ odd} \\ \frac{1}{2}\left({x_{i,h}}^{(\frac{m}{2})} + {x_{i,h}}^{(\frac{m}{2}+1)}\right) & , m \text{ even} \end{cases}$$

where $\hat{y}_{t,i,h}$ is the predicted production of plant i for the h^{th} hour of the day, and $x_{i,h}$ are the past production values of plant i at hour h, which belong to the same quantile as $DSWRF_{t,i,h}$. The value $DSWRF_{t,i,h}$ is obtained for each solar plant by averaging the $DSWRF_t$ values at the grid points encircling the actual location of plant i.

3.1.3. Seasonal Autoregressive Integrated Moving Average Exogenous Model (SARIMAX)

Seasonal Autoregressive Integrated Moving Average Exogenous Model (SARIMAX) is included in order to provide a time series-based alternative to models proposed in this study. SARIMAX makes use of average DSWRF values of the whole region as the exogenous regressor. Other parameters related to auto regression (AR), differencing, moving average (MA) and seasonality are selected using the Hyndman-Khandakar algorithm [34], which iterates to find the best fit, until the Akaike Information Criterion (AIC) stops improving. SARIMAX model is constructed abiding the day-ahead setup, where the latest observed production is 48 hours ahead of the forecasted production.

3.2. Artificial Neural Networks

Heuristics try to solve mathematical optimization problems with an algorithmic structure in which the solution is iteratively enhanced in a similar way with some phenomena that is observed in the nature. There are many heuristic methods in the literature such as genetic/evolutionary algorithms, particle swarm optimization and ant colony optimization, to name three. These algorithms benefit from biomimicry for the purpose of optimizing an objective. Although they mostly not yield to an exact

optimal solution to the problem, they are commonly used in some cases where a good-enough solution is more practical rather than waiting for an exact solution to complete its long runtime, especially in problems with a high complexity.

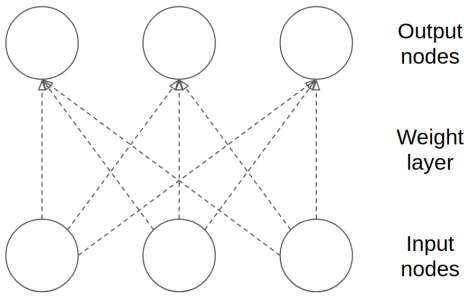

Figure 3.1. Simple Perceptron

Among other heuristics, artificial neural networks are perhaps the most popular and the ones that made the biggest leap, especially in the past decade. These models mimic the way the brain functions by adopting the concept of neurons -building blocks of the brain. As in the brain, each neuron, or perceptron as illustrated in Figure 3.1, has its own inputs and outputs, with a threshold above which the signal is fired forward, called the activation. In time, many different ANN architectures have emerged from this simple idea, each being built specifically for the type of the problem aimed to be solved. However, regardless of their design, all ANN architectures with a continuous response variable try to estimate the following function f with a given set of input(s) X and output(s) y, in line with all other regression models. The objective is

$$min\ L(y, f(X)) \tag{3.1}$$

where L is the loss function, i.e. sum of square errors,

$$y = E[y|X] + \varepsilon \tag{3.2}$$

$$= \hat{y} + \varepsilon \tag{3.3}$$

$$= f(X) + \varepsilon \tag{3.4}$$

and ε is noise with zero mean.

ANN model initializes with random weights. At each iteration, model is fed with a single sample (online) or a batch of samples, i.e input tensor. The input tensor is consecutively multiplied by weight tensors, represented by each hidden layer in the graph model. The result of this tensor multiplication becomes the output of this single iteration, after which the error is measured using the true value of the corresponding output. Next, the weights are updated through the direction which minimizes the error. Directions are obtained analytically by computing the gradient of the loss function with respect to each weight. Starting from the final layer, gradients of each weight associated with the corresponding hidden nodes are calculated. Calculations start from the final layer in order to avoid redundant computation, since each gradient also includes the gradients of its successor nodes, by the definition of chain rule. This process of calculating the weights according to the error of each iteration is called backpropagation.

After all of the gradients are calculated, a gradient descent heuristic such as SGD [35] or ADAM [36] is used in order to minimize the loss function. Basic logic behind these heuristics is that they iteratively move towards the negative of the gradient, i.e. towards a local minimum. The decision of how far to go along these directions, so called the learning rate, is a parameter to be tuned.

Backpropagation is one of the main considerations in the design of an artificial neural network. For instance, as the depth of the network increase, the number of terms added to the chain rule increase and the magnitude of the gradient updates for each weight gets closer to zero (or infinity). This phenomenon is called vanishing (or exploding) gradients and causes serious problems, especially in recurrent neural networks. Another design aspect of an ANN is the model complexity, since a too complex model pose a risk of overfitting. In general, complexity increases as the number of weights increase in a model. In order to avoid this, several techniques are used such as dropout, L1 and L2 regularization.

L1 and L2 regularization, also called lasso and ridge respectively, are penalization techniques that help to reduce the number of non-zero weights, hence decreasing the overall model complexity. The idea is simply to add a penalization term to the objective function mentioned above.

$$min\ L(y, f(X)) + \lambda * Regularization(w) \tag{3.5}$$

where λ is the penalization coefficient and w are the weights of the model. The $Regularization$ function is $\sum_{k=1}^{k=N} {w_k}^2$ for the ridge and $\sum_{k=1}^{k=N} |w_k|$ for the lasso, where N is the number of weights subject to penalization.

Dropout [37] is another method for regularization. At each iteration, randomly selected subset of hidden nodes are ignored, as illustrated in Figure 3.2. In the forward pass, the output of a selected hidden node is set to zero, and in the backward pass, gradients of the weights associated with the same node are also set to zero. This way, the model trains only a randomly selected part of the network at each iteration. This simple but powerful technique helps to increase ANN's generalization ability, by decreasing complexity and avoiding potential overfitting.

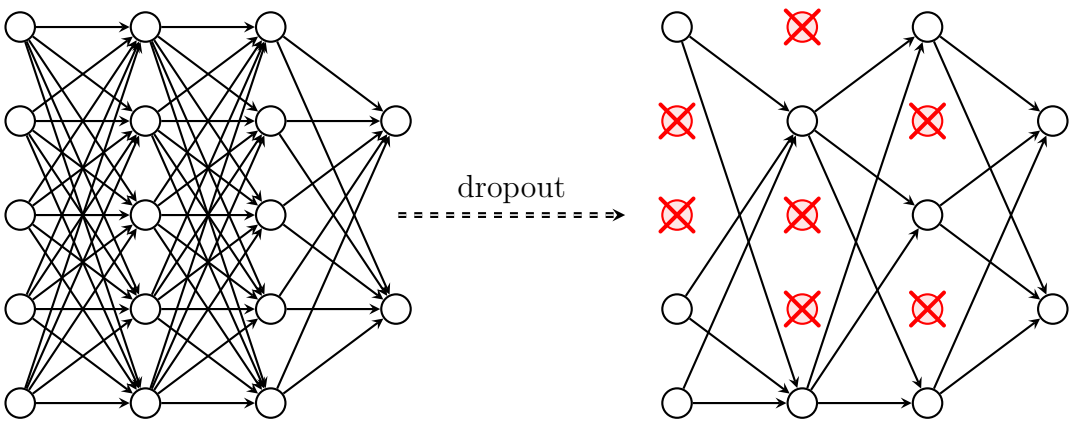

Figure 3.2. Illustration of dropout mechanism. [1]

Artificial neural networks are universal approximators [38]. They can approximate any function, given a certain depth, width and activation function setup. Since they are built with directed graphs, they are very flexible in terms of the modeling layout. An advantage of graph-based design of ANN is that it is quite convenient for object oriented programming. Libraries such as Tensorflow [39] and Pytorch [40] provide great opportunities not only for implementing state-of-the-art algorithms in the literature, but also for building complex architectures from scratch, without having to struggle with gradient formulation and coding burden of long derivations for the backpropagation.

This property makes almost every model proposed in the literature unique. In general, it might be misleading to assume that a specific model would outperform others on a particular problem. Therefore, a proper comparison among several competing models can only be done by applying them on the same dataset, i.e training and testing on exactly the same samples.

Although there are plenty of different aspects that differentiate various ANN architectures, the main distinction is the graph layout built between the input and output layers. Following sections cover the architectures employed in this thesis.

3.2.1. Fully-Connected ANN (FC-ANN)

A fully connected graph layout is obtained when each node in the network is connected to all nodes that are in the preceding and succeeding layers, as illustrated on the right in Figure 3.3 . The number of hidden layers (depth), or the number of hidden nodes in a single layer (width) can be altered according to the size of the input and output vectors and the complexity, i.e. nonlinearity, of their underlying relationship. At each hidden node, output vector of the preceding hidden layer is multiplied with the corresponding weight vector. After that, a bias term is (w_0) is added before the activation function is applied, as illustrated on the left in Figure 3.3. In this study, the input data is a 2D matrix consisting of NWP grids, where each element in the matrix

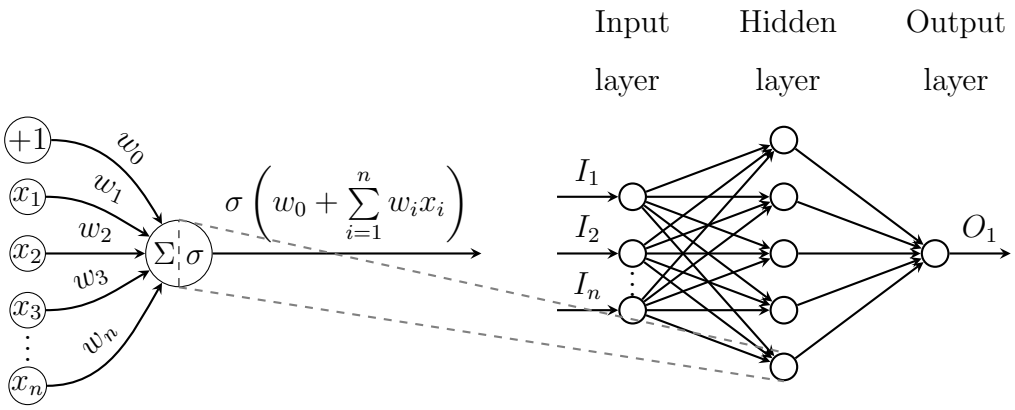

Figure 3.3. The graph layout of a fully-connected multi layer perceptron on the right and input/output structure of a single hidden node on the left. [1]

represent a single latitude/longitude combination. In order to feed this data into the FC-ANN layout, the matrix is flattened and a single vector is obtained. The output layer consists of a single node, for the observed production of each hour.

3.2.2. Convolutional Neural Network (CNN)

A fully connected multi-layer perceptron is a highly robust design that can learn almost any function. However, as the number of inputs increase, the model complexity increases dramatically. In the case of images, the number of inputs is the number of pixels times the channels. For instance, an extreme example would be an RBG image with 1080p resolution which provides over 6.2 million inputs to the fully connected network. Therefore, a fully connected approach is computationally impractical for computer vision problems which aim to sense digital images and videos. Convolutional networks decreases the number of weights to be trained by taking another approach. Instead of flattening the 2D image or matrix into a 1D vector, a kernel matrix, i.e. filter, is applied spatially. This filter slides along both dimensions of the image, each time outputting a single value. This single value results from the element-wise product of the input and kernel matrices, followed by summation of all the values, as illustrated in Figure 3.4. This process is called convolution, hence the name of the architecture.

A convolutional filter carries the same weights throughout the processing of the image. For that reason, convolution process does not train the weights on a particular region of the input. Instead, it teaches the weights in a way that the filter gets activated when fed with a particular visual feature such as a certain shape, edge, color combination and so on. Each convolutional layer can contain an arbitrary number of filters, which is subject to parameter tuning. Each filter is expected to grab a certain visual feature related to the output. Therefore, it is a general practice to increase the number of filters as complexity in the relationship between the input and output increases.

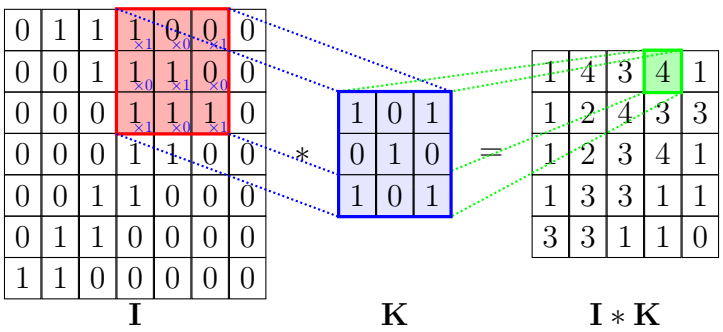

Figure 3.4. Demonstration of a convolutional kernel matrix. Kernel matrix K slides through the input matrix I, producing a single output at each pass. [1]

After applying a convolutional layer, the spatial dimension of the output is less than the input's. Stacking multiple convolutional layers serially, the input image can be downscaled to a lower resolution image with arbitrary number of latent channels. If the downscaling affect is not desired, the input matrix can be padded so that the dimensionality can be kept the same, as illustrated in Figure 3.5.

Another dimensionality reduction technique used in this context is pooling. Pooling is a simple arithmetic process which works similar to convolution. Here, instead of weighted averaging of the pixels within a kernel, arithmetic operations such as min, max or simple averaging is used, as illustrated in Figure 3.6. In this way, no additional weights are added to the chain rule while the image is still downsampled.

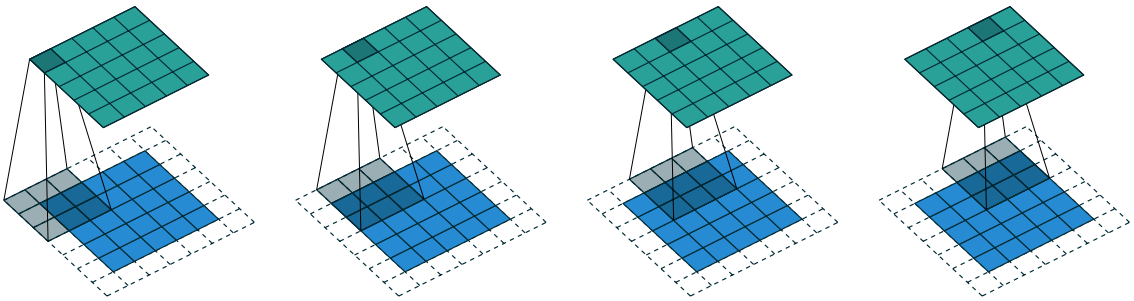

Figure 3.5. Demonstration of a 3x3 convolution with a stride of 1 and padding. [2]

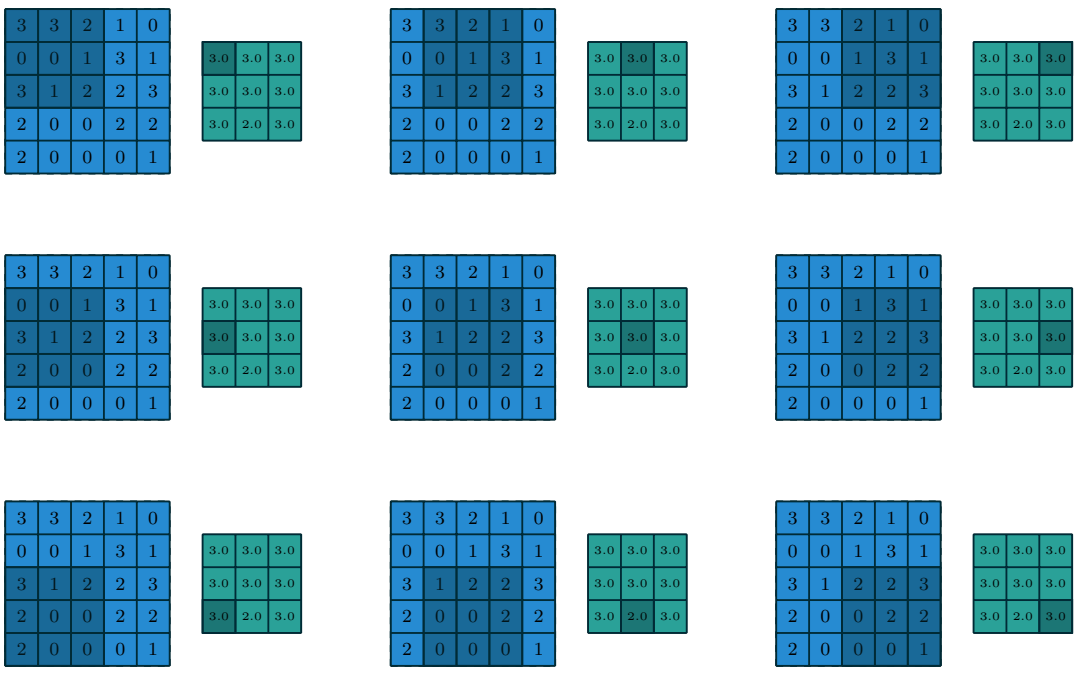

Figure 3.6. Demonstration of max pooling. [2]

In the case of a low resolution input, it might be beneficial to increase the number of data points, in order to obtain a higher resolution. Gaps between consecutive points can be filled with linear interpolation or even with its nearest neighbor. Transpose convolutional layers provide a great opportunity for upsampling a 2D input image, instead of using a deterministic approach such as nearest neighbor, linear, bilinear, bicubic and trilinear interpolation techniques. However, it adds more complexity to the model while upsampling because of the introduction of the additional weights to be learned.

3.2.3. Long Short Term Memory (LSTM)

The architectures mentioned in previous sections does not take temporality into account. Modeling the temporal relationships of sequential input is made possible by the introduction of recurrent neural networks [41]. In simple words, the main difference between RNN and MLP is that in RNN, each hidden node in the network is also fed with its previous state, i.e. previous output of the same hidden node in the sequence. The main problem with this design is that gradients vanish very quickly, especially when the sequence length is increased. Long short term memory networks are a special type of RNN which is proposed in order to solve this major problem. In LSTMs, the recurrent feedback mechanism is controlled with gates. The hidden units in the RNN are replaced with LSTM cells that are composed of input, output and forget gates, as illustrated in Figure 3.7. Gated structure of each cell yields to additive terms in the gradients of cell state, which helps preventing the gradients from vanishing. In bidirectional LSTMs, recurrent information is fed not only from the previous states, but also from the future states of the sequence. This is achieved by simply duplicating the cell and providing the input in reversed sequence, as illustrated in Figure 3.8. Bidirectional LSTMs are particularly useful when all time steps of the input sequence are available for the predicted time step.

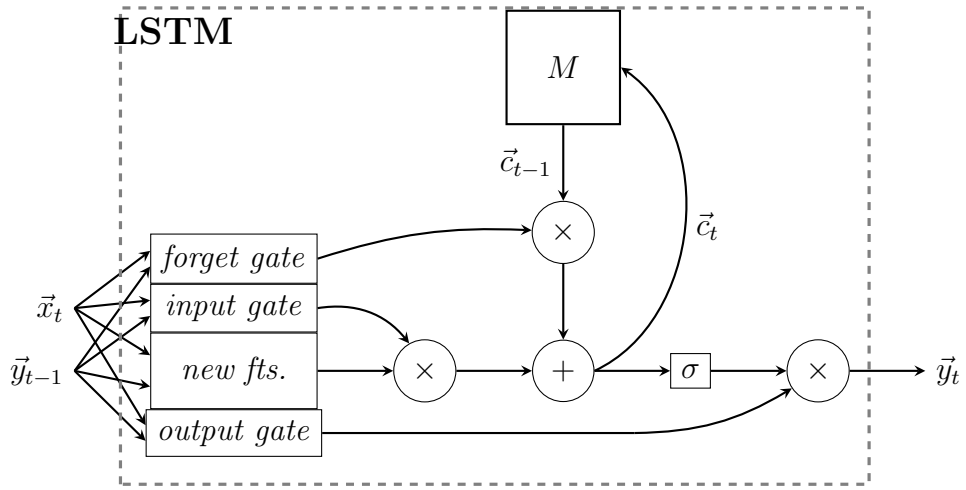

Figure 3.7. Structure of an LSTM cell. [1]

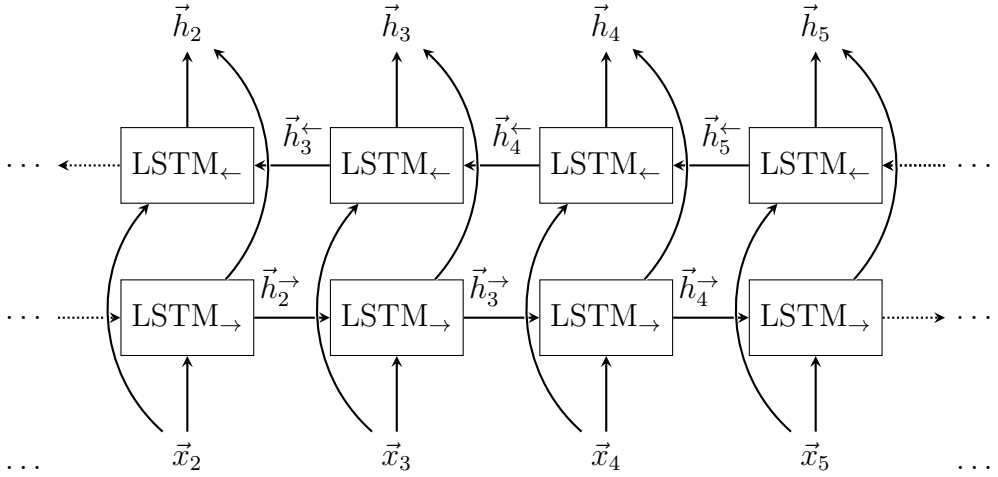

Figure 3.8. Structure of a bidirectional LSTM network. [1]

3.2.4. Convolutional LSTM (ConvLSTM)

The major drawback of FC-LSTM in handling spatiotemporal data is its usage of full connections in input-to-state and state-to-state transitions in which no spatial information is encoded. [3] The ConvLSTM is fundamentally a variant of LSTM, where the distinguishing feature is that all the inputs, cell outputs, hidden states and gates are 3D tensors. In this way, the spatial data properties can be well included and preserved during training and predicting processes by exchanging the internal matrix products via convolutional operations [6].

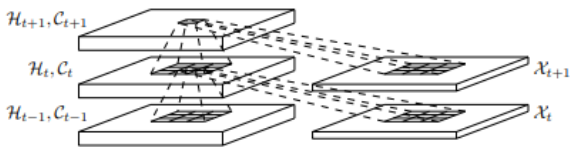

Figure 3.9. Convolutional LSTM [3]

Equations of a single convolutional LSTM cell are shown below, where $*$ is the convolution operator and $\circ$ is element-wise multiplication [3].

$$i_t = \sigma(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \circ C_{t-1} + b_i) \tag{3.6}$$

$$f_t = \sigma(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \circ C_{t-1} + b_f) \tag{3.7}$$

$$C_t = f_t \circ C_{t-1} + i_t \circ tanh(W_{xc} * X_t + W_{hc} * H_{t-1} + b_c) \tag{3.8}$$

$$o_t = \sigma(W_{xo} * X_t + W_{ho} * H_{t-1} + W_{co} \circ C_t + b_o) \tag{3.9}$$

$$H_t = o_t * tanh(C_t) \tag{3.10}$$

3.3. Performance Metrics

In this section, performance metrics that are used to measure the forecasting power of the models implemented in the study.

3.3.1. Percent Bias

Percent bias (PBias) is the total prediction error divided by the total actual production.

$$PBias = \frac{\sum_{t=1}^{N}(y_t - \hat{y}_t)}{\sum_{t=1}^{N} y_t}$$

3.3.2. Daily Percent Bias

Daily percent bias (DPBias) is the daily total prediction error divided by the daily total actual production.

$$DPBias = \frac{\sum_{h=1}^{24}(y_h - \hat{y}_h)}{\sum_{h=1}^{24} y_h}$$

Median and standard deviation values are reported using $\text{DPBias}_{\text{med}}$ and $\text{DPBias}_{\text{sd}}$ respectively.

3.3.3. Weighted Mean Absolute Percentage Error

This metric is used instead of mean absolute percentage error in order to put emphasis on the amount of total absolute error within the forecasted period. This is important due to the production characteristics of the solar power, where the most part of the production occur during summer.

$$WMAPE = \frac{\sum_{t=1}^{N} |y_t - \hat{y}_t|}{\sum_{t=1}^{N} y_t}$$

3.3.4. Daily Weighted Mean Absolute Percentage Error

Daily WMAPE is similar to WMAPE except that it is calculated for each day separately.

$$DWMAPE = \frac{\sum_{h=1}^{24} |y_h - \hat{y}_h|}{\sum_{h=1}^{N} y_h}$$

Median and standard deviation values are reported using $\text{DWMAPE}_{\text{med}}$ and $\text{DWMAPE}_{\text{sd}}$ respectively.

4. METHODOLOGY

4.1. Data

In this section, spatiotemporal datasets with varying temporal and spatial resolutions are explained. These datasets can be grouped into two main categories: PV power output and exogenous data that provide external information such as weather conditions, clear sky irradiance and sunlight position.

4.1.1. Photovoltaic Power Production

Main source of data is hourly electricity generation collected from over 1100 PV power plants with nearly 1 GWe installed capacity, located in various sites of central Anatolian region in Turkey. Compared to other work in the literature, this data has some unique properties in various aspects. Firstly, it is one of the largest in terms of the number of PV stations and the total installed capacity. Secondly, the capacity of each plant is under 1 MW. Therefore, the installed capacity of the overall system is distributed evenly throughout the region. Nevertheless, the distribution is not exactly uniform since an individual plant is generally found within a cluster of neighboring plants. Lastly, there is a distance of nearly 300 kilometers between the plants farthest from each other. This property is crucial since weather conditions vary remarkably as the distances increase, hence the need for multi-location numerical weather predictions.

4.1.2. Exogenous Data

Another source of data is Global Forecast System (GFS) [42] by the United States National Centers for Environmental Prediction (NCEP). GFS is a NWP model that provides historical occurrences as well as future expectations of a number of atmospheric conditions such as temperature, wind speed and direction, humidity and precipitation. These variables are commonly used as predictors to machine learning

models built for renewable power generation prediction. In this study, downward shortwave radiation flux (DSWRF) is the variable that is commonly used in almost all the models described below, as it is a reliable estimate of solar irradiance. It covers the entire world with a spatial resolution of 0.25° (roughly 28 kilometers) and a temporal resolution of 3 hours. A bounding box consisting of 84 grid points is used to cover the region spanned by solar power plants.

Additionally, hourly clear sky global horizontal irradiance (GHI) values [43] and sunlight position [44], i.e. altitude and azimuth, are obtained for each hour of the corresponding coordinates of the NWP grid. Although these are calculated deterministically, they offer some additional information regarding the spatiotemporal behavior of PV output. For instance, clear sky GHI provide an upper bound for the irradiance at a certain location at a particular time. Lastly, daily earth-sun distances [45] are also introduced in some trials to see if it provides any additional information regarding the seasonality of earth's movement around the sun. In addition to earth-sun distance, one-hot encoded binary hour columns are added to this auxiliary data. Unlike above mentioned data, auxiliary data is not spatiotemporal, i.e. does not contain a spatial dimension.

Spatiotemporal exogenous data (NWP, clear sky and sunlight positons) gathered from several sources are concatenated into a multidimensional array, i.e. tensor. This way, they can be used by DL models for different spatial and temporal transformations. Four of the features are selected for a single hour in the data in order to illustrate this process in Figure 4.1. These features, i.e. channels, are merged into a 3D tensor as seen in Figure 4.2, which constitutes the single observation that is fed into non-sequential models such as FC-ANN and CNN. Recurrent models such as ConvLSTM, FC-LSTM and the proposed architecture use a 4D tensor as a single observation, consisting of a sequence of 3D tensors, illustrated in Figure 4.3.

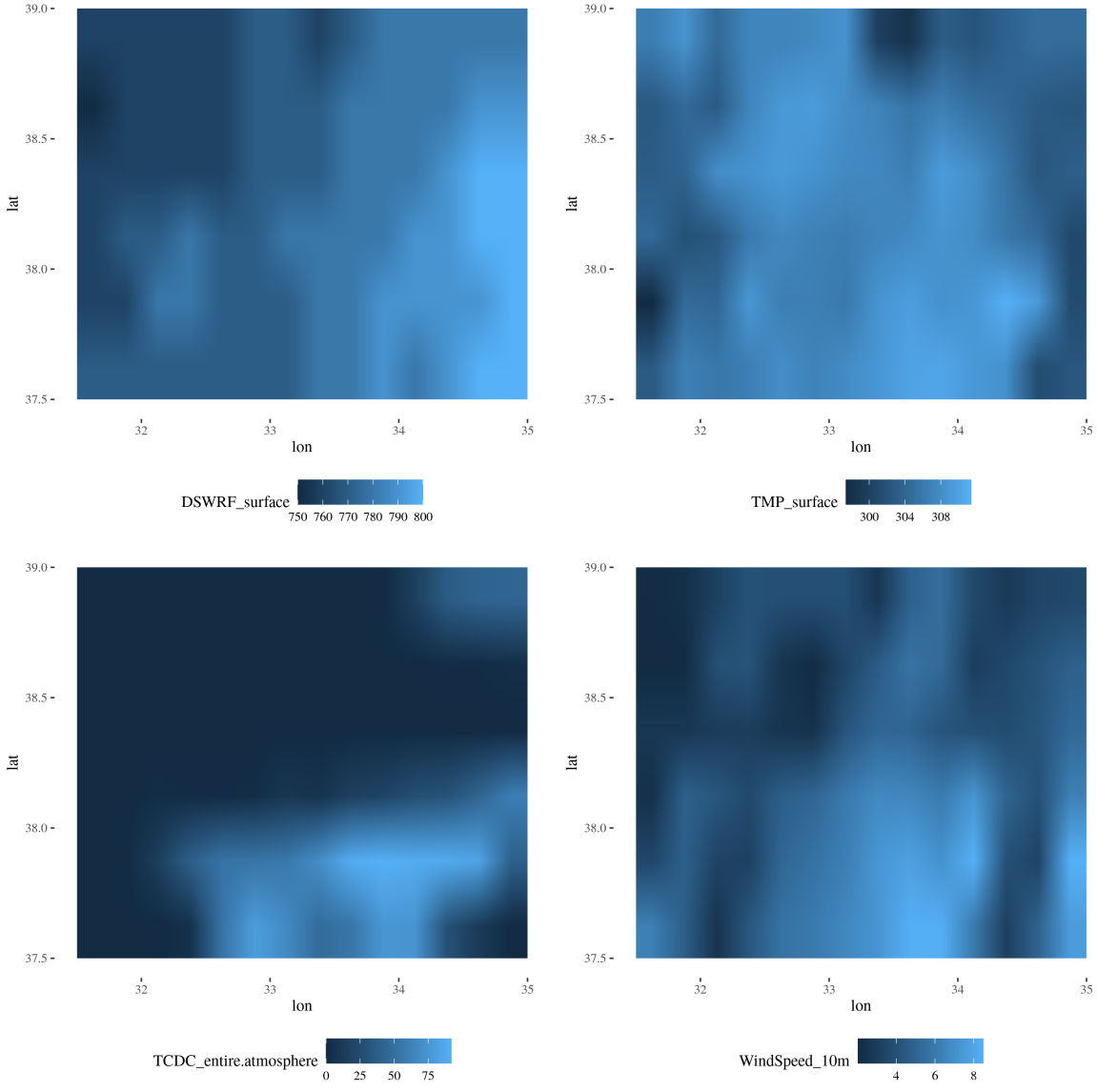

Figure 4.1. 2D spatial tensors of four features representing a single hour in the data.

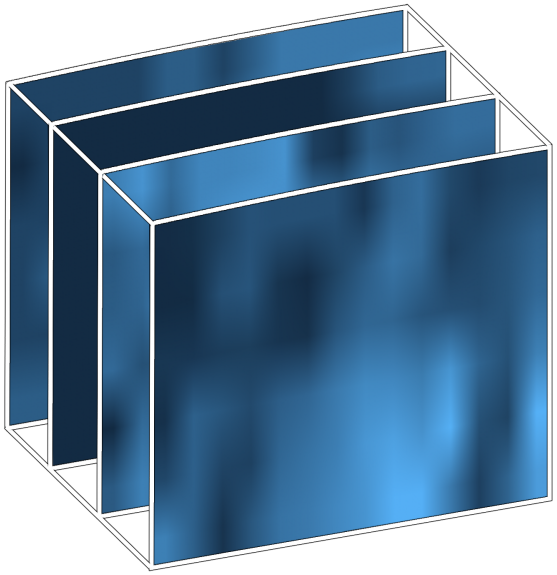

Figure 4.2. A 3D tensor containing values of four channels of a single hour.

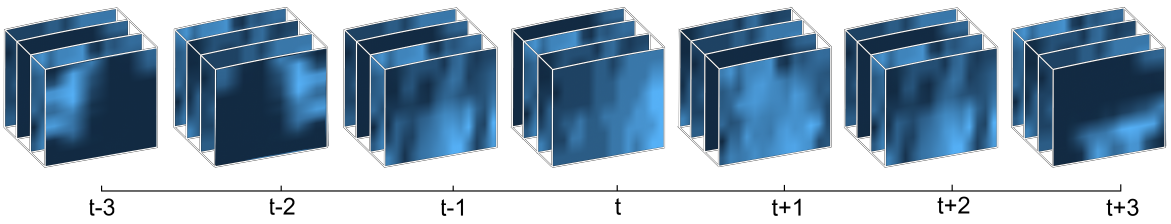

Figure 4.3. A 4D tensor representing a 7-hour sequence.

4.2. Proposed Architecture

In this study, PLC-LSTM architecture is proposed to be an alternative to the fully-connected and shared-weight structures of the layouts explained in background chapter. The 2D spatial frame of the input tensor is divided into subregional kernels. Similar to convolutions, these kernels are controlled parametrically with a tunable kernel size and stride. Then, instead of convoluting with a shared-weight filter, an FC-LSTM is connected to each kernel individually, constituting a parallel ensemble of multiple LSTM modules. Similar to CNN, kernel size controls the height and width of the spatial region spanned by each locally-connected LSTM module. Stride controls the amount of shift while sliding the kernel horizontally and vertically throughout the 2D spatial plane. Together, kernel size and stride control the total number of FC-LSTM modules. Subregions might be disjoint, as illustrated in Figure 4.4, or contain intersecting areas, depending on the selected size and stride parameters. These local learners are then followed by dense fully-connected layers before the final output.

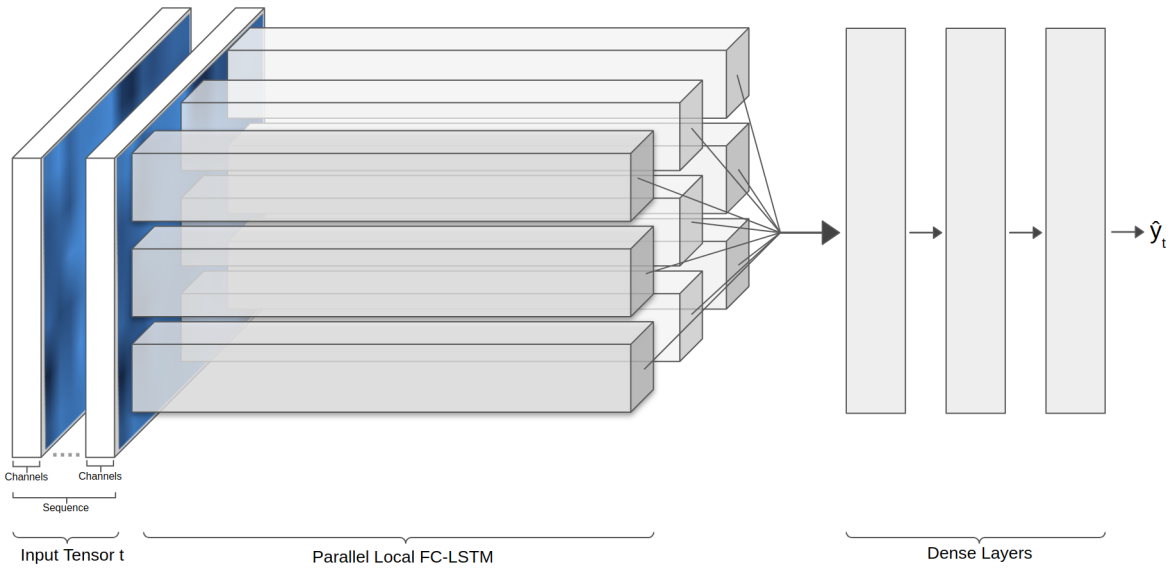

Figure 4.4. Layout of the proposed PLC-LSTM model

The idea behind PLC-LSTM is to reduce the complexity of the fully-connected architecture while avoiding the shared-weight structure of CNN. In a fully-connected layout, all locations are fed into the model as input and construct a number of linear

combinations, i.e. as many combinations as the number of nodes in the next layer. This layout introduce a redundant complexity as the area of the problem increases, since a linear combination of two distant locations is not necessarily meaningful in terms of the PV power output. PLC-LSTM avoids this by grouping nearby locations within the kernel and isolating the input and weights associated to each kernel until the end of LSTM output. Afterwards, outputs of each subregion are fed into fully-connected dense network. Here, the model implicitly decides which regions are associated the most with the aggregated output. The advantage of not using a convolutional approach here is that each kernel has its own locally learned weights, instead of having the same weight with all other kernels. This is beneficial for two reasons. Firstly, as opposed to many image recognition problems, PV power forecasting problem does not require translational invariance property since the locations of the power plants does not change. A locally observed meteorological phenomenon that causes a significant shift in the power production does not necessarily result in a similar effect when seen at another location. Secondly, a locally-separated structure allows us to disregard a subregion if the installed capacity within the kernel is below a certain threshold. As the size of the studied area of aggregated power increases, these become more of an advantage with the increased sparsity of PV power installations and also the increased chance of observing different kinds of weather events occuring in different subregions simultaneously.

4.3. Parametrization

Both the models mentioned in the background chapter and the proposed model require a large number of design-related decisions. In order to provide a flexible experimentation setup, most of these decision variables in the modeling structure are parameterized and hard-coded properties are avoided as much as possible. These parameters are grouped into 4 categories, namely *training*, *optimizer*, *dataset* and *model*, in order to have an easier control during the tuning process. In the implementation, each group is only passed to the corresponding part of the modeling framework. This modular structure in parametrization is especially practical for experimenting with new

ideas and allows major changes in the architectural design of deep learning models. The parameters commonly used by all models are described in Tables 4.1 and 4.2.

Table 4.1. Description of common training and optimizer parameters used by all model architectures.

Parameter Group	**Parameter Name**	**Parameter Description**
Training Config	max_epochs	Maximum number of epochs to train the model
	batch_size	Number of the observations sampled at each batch
	eval_per_epoch	Sets the period for evaluation of error metrics for train and validation sets
	loss	Loss function used as the objective of gradient descent
	lr_exp_decay_rate	Rate of the exponential decay applied to the learning rate after each epoch
Optimizer Config	optimizer	Specifies the gradient descent algorithm used for weight optimization
	lr	Learning rate
	weight_decay	Penalization parameter for weight regularization

Apart from the common ones, each model has its own set of unique parameters, stemming from the implementation details or the design of the model itself. For instance, bilinear upsampling is used for enlarging the 2D spatial input for fully-connected and CNN models using the *scale_factor* parameter described in Tables 4.3 and 4.4. In addition, CNN requires some convolution-specific parameters such as *kernel_size* and *groups*.

Table 4.2. Description of common dataset and model parameters used by all model architectures.

Parameter Group	**Parameter Name**	**Parameter Description**
Dataset Config	train_start	Start date of training set
	valid_start	Start date of validation set
	valid_end	End date of validation set
	include_hours	Hours of the day to be included in the model
	add_dswrf	Adds DSWRF feature as an input
	add_ghi	Adds GHI feature as an input
	add_altitude	Adds altitude feature as an input
	add_azimuth	Adds azimuth feature as an input
	add_temp	Adds temperature feature as an input
	add_ws10	Adds wind speed at 10 meters as an input
	add_tcdc_entire	Adds total cloud density cover of entire atmosphere feature as an input
	add_aux	Adds auxiliary features as an input
	scale_type	Controls the type of the scaling, i.e. standardization or normalization
Model Config	spatial_size	Height and width of the input
	input_channels	Number of channels in the input
	bias	Logical parameter controlling the addition of bias terms for all layers
	dropout	Dropout rate
	sigmoid	Logical parameter for final sigmoid activation

Table 4.3. Description of model parameters used by fully-connected model.

Parameter Group	**Parameter Name**	**Parameter Description**
Model Config	hidden	Array of number of hidden units for each layer
	scale_factor	Factor of bilinear upsampling of the input

Table 4.4. Description of model parameters used by CNN model.

Parameter Group	**Parameter Name**	**Parameter Description**
Model Config	hidden	Array of number of filters for each convolutional layer
	scale_factor	Factor of bilinear upsampling of the input
	groups	Controls the connection between input and output channel in the first convolutional layer
	kernel_size	Size of the convolution kernel

Recurrent models, i.e. models which include time-series dimension, require two additional parameters, namely *time_window* and *alignment*, together controlling the length of the sequence and position of the predicted observation within the sequence respectively. For instance, when the *time_window* is set to an integer k and *alignment* is *right*, the predicted observation is positioned $(k+1)^{th}$ at the sequence. When *alignment* is *center*, predicted observation is again aligned at the $(k+1)^{th}$ position, however, this time the total length of the sequence becomes $2k+1$. Lastly, when the *alignment* is *custom*, the model expects the *time_window* parameter to be an array of two, adding the provided number of observations before and after the predicted observation. This way, it allows even more control over the fine tuning of temporal relationships using custom numbers for lag and lead inputs.

Set of parameters that customize ConvLSTM model are described in Table 4.5. There has been made a slight modification to the original model [3], by introducing *conv_per_cell* parameter which controls the number of serially connected 2D convolution layers of in each cell.

Parameters of FC-LSTM and PLC-LSTM are described in Tables 4.6 and 4.7 respectively. Both of them commonly include *num_layers*, *hidden* and *bidirectional* parameters since PLC-LSTM contains multiple FC-LSTM modules in parallel. In addition to these parameters, PLC-LSTM requires *unfold_size* and *unfold_step* that control how multiple subregional modules span the overall area. For instance, increasing the *unfold_size* decreases the number of subregions since now each locally-connected module trains over a larger area. Specifying a *unfold_step* lower than the *unfold_size* makes subregions intersect with each other whereas specifying a higher *unfold_step* makes the overall area spanned more sparsely.

Table 4.5. Description of model parameters used by ConvLSTM model.

Parameter Group	**Parameter Name**	**Parameter Description**
Model Config	num_layers	Number of serial ConvLSTM layers
	conv_per_cell	Number of 2D convolutional layers in each ConvLSTM cell
	fc_hidden_nodes	Controls the number of hidden nodes in the final fully connected layers
	hidden	Array of number of filters for each convolutional layer
	kernel_size	Size of the convolution kernel
Dataset Config	time_window	Controls the number of time steps in a single sequence
	alignment	Controls the alignment of the predicted time step within the time window

Table 4.6. Description of model parameters used by FC-LSTM model.

Parameter Group	**Parameter Name**	**Parameter Description**
Model Config	num_layers	Number of serial LSTM layers
	hidden	The number of features in the hidden state in each cell
	bidirectional	Logical parameter that changes LSTM into a bidirectional-LSTM
Dataset Config	time_window	Controls the number of time steps in a single sequence
	alignment	Controls the alignment of the predicted time step within the time window

Table 4.7. Description of model parameters used by PLC-LSTM model.

Parameter Group	**Parameter Name**	**Parameter Description**
Model Config	unfold_size	Size of each locally-connected subregion
	unfold_step	Amount of shift, i.e. stride, between consecutive subregions
	num_layers	Number of serial layers in each locally-connected LSTM
	hidden	The number of features in the hidden state in each cell
	bidirectional	Logical parameter that changes each locally-connected LSTM into a bidirectional-LSTM
Dataset Config	time_window	Controls the number of time steps in a single sequence
	alignment	Controls the alignment of the predicted time step within the time window

5. EXPERIMENTS

5.1. Modeling Pipeline

In this section, machine learning pipeline that is built for a single experiment, i.e. trial, is explained. An experiment stands for training, validation and testing of a certain model architecture with a group of predefined parameter set. The pipeline is designed to be independent of the parameter tuning setup so as to allow any optimization technique to be implemented during the hyperparameter search step. At the beginning of each experiment, parameters are set either manually or by a tuning algorithm, and passed to *run_model* function within a json file containing configurations of that particular experiment. This function is the wrapper for the pipeline, containing calls to other functions for loading and processing data, model initialization, training, evaluation and reporting. After the *run_model* is initiated, first, parameters related to data processing and model-specific modifications are passed to *load_data* function and following key tasks are handled.

- Datasets mentioned in section 4.1 are loaded into the environment.
- Solar power production data is converted into hourly utilization values: $\frac{production_h}{capacity_h}$
- Variables in NWP weather data are filtered according to parameter set defined in dataset configuration.
- Spatiotemporal weather data is converted into a multidimensional array and standardized.
- All datasets are filtered by hour according to *include_hours* parameter. This feature is added to be able to remove the night-time observations when the production is consistently zero.
- If the model handle time-series, a sequence is generated for each timestamp, controlled by *alignment* and *time_window* parameters in dataset configuration.
  (i) *alignment* controls how the target observation y_h is positioned along the sequence. Possible values are *center*, *right* and *custom*.

(ii) *time_window* controls the length of the sequence.

- All datasets are split into train,validate and test sets according to the specified date values in dataset configuration.

After the data is loaded and processed, *get_model* function is called in order to create the Pytorch model. At this step, layout and weights in the model definition is initialized with the parameters specified in the configuration file and the model is moved to GPU memory. Then, optimizer is initialized similarly and the model weights are introduced to the optimizer.

The preprocessed data and the initialized model are then passed to *training_loop* function in which the weights are iteratively updated with respect to observations in the train set. At each iteration, first, a batch of size *batch_size* is sampled with replacement, i.e. bootstrapped, from the train set. Then, the batch is passed to the feed-forward network and a predicted value $\hat{y}$ obtained for each observation in the batch. Batch loss $L(y, \hat{y})$ is computed and gradients with respect to each weight are obtained using Pytorch autograd function [40]. Lastly, the optimizer is called and the network weights are updated using the *learning_rate*. Adam [36] is used as the optimizer method and the *learning_rate* is decreased at each epoch using exponential decay method. Each epoch contains $\frac{N}{batch\_size}$ iterations where N is the number of observations in the train set. In order to be able to monitor how the model performs over the course of the training procedure, both of the train and validation sets are fed to the model separately every *eval_per_epoch* epochs without backpropagation. Although it might seem that feeding the train set for a second time is redundant since we already have fitted values from the feed-forward step, those results are coming from the bootstrapped subsamples and not exactly comparable to the validation set. Therefore, the parameter *eval_per_epoch* is introduced to have more control on the frequency of the evaluation steps, since evaluating both the train and validation sets after each epoch leads to a considerable increase in the computation time. This evaluation step also keeps track of the best performing epoch of the training procedure and also the model state at best epoch. The performance comparison is made using WMAPE metric explained in section 3.3.

Finally, when *max_epochs* is reached, resulting model on the last epoch is evaluated on the test set. Then, the fitted values for train, validation and test sets are computed and saved using the model states of both the latest epoch and the best performing epoch. In addition to fitted values of each hour, performance of train and validation sets at each epoch are saved, in order to be able to visualize the training procedure later on. Overall performance of the test set of an experiment is not reported, in order to avoid cherry-picking during the selection of the best performing model. It is important to note that the test results that are reported in this study are selected considering validation results only. The state of the trained model, i.e. network weights, at the best performing epoch and at the last epoch are also saved, in order to allow a future work to be done on a previously fitted model, such as retraining the model with new data or a transfer learning application.

5.2. Hyperparameter Tuning

Unlike some other machine learning algorithms, i.e. random forests, deep learning models are highly sensitive to initial configuration of parameter settings and weight initialization. Therefore, parameter tuning is an essential part of deep learning model development process. Since the running time of these models are not relatively fast, parameter search space should be kept as small as possible. In this study, a hyperparameter optimization framework, Optuna [46] is utilized. Optuna provides a define-by-run API that allows users to construct the parameter search space dynamically, an efficient implementation of both searching and pruning strategies, and built-in functionality for parallel distributed computing [46].

Following steps are taken for hyperparameter tuning. First, a subset of parameters is selected to be tuned. At each trial, each parameter is sampled from a given range of continuous or discrete distribution, depending on the parameter. The sampling is made by Tree-structured Parzen Estimator (TPE), which is a hyperparameter optimization algorithm [47]. TPE samples parameters based on the previous outcomes of the tuning. At each iteration, for each hyperparameter, TPE fits one gaussian mixture

model (GMM) $l(x)$ to the set of hyperparameter values associated with the smallest (best) loss function values, and another GMM $g(x)$ to the remaining hyperparameter values. It chooses the hyperparameter value x that maximizes the ratio $l(x)/g(x)$. [48] After each parameter is sampled by TPE, the model is run with training and validation sets. During the training, validation error metric is periodically feed back to a pruning algorithm as an intermediate result, in order to stop unpromising trials. For this purpose, a slight modification has to be made in the *training_loop* function, in order to make the training algorithm communicate with the HPO algorithm. In this study, percentile stopping rule is used for pruning, which stops the trial as described in Figure 5.1.

```
Initialize a new study object or load from database if already initialized
while time < timeout do
  Create a new trial_i within the study
  Sample parameters using TPE sampler
  Start model training
  for s = 1 to n_evaluation_steps do
    Report error metrics for training and validation sets
    if trial_number > n_startup_trials then
      if validation_error_s > p^th percentile of all validation_error_s then
        Stop trial_i
      end if
    end if
  end for
end while
```

Figure 5.1. Psuedo code for hyperparameter optimization.

5.3. Hyperparameter Search Space

In this section, tuning values of each parameter are specified. In order to keep the resulting hyperparameter search space at a reasonable size, some of them are tuned while some are fixed at a constant value. Values of the common parameters used by all

modes are shown in Table 5.1 and values of parameters that are specific to each model are presented in Appendix A.

Table 5.1. Common parameters used by all model architectures.

Parameter Group	**Parameter Name**	**Parameter Type**	**Sampling Type**	**Value(s)**
Training Config	max_epochs	constant		100
	batch_size	constant		128
	eval_per_epoch	constant		4
	loss	hyperparameter	categorical	[”MAE”, ”WMAPE”]
	lr_exp_decay_rate	constant		0.999
Optimizer Config	optimizer	constant		”adam”
	lr	hyperparameter	loguniform	
	weight_decay	hyperparameter	loguniform	
Dataset Config	train_start	constant		”2017-06-01”
	valid_start	constant		”2019-01-01”
	valid_end	constant		”2020-01-01”
	include_hours	constant		[5,22]
	add_dswrf	constant		True
	add_ghi	hyperparameter	categorical	[True, False]
	add_altitude	hyperparameter	categorical	[True, False]
	add_azimuth	hyperparameter	categorical	[True, False]
	add_temp	hyperparameter	categorical	[True, False]
	add_ws10	hyperparameter	categorical	[True, False]
	add_TCDC_entire	hyperparameter	categorical	[True, False]
	add_aux	hyperparameter	categorical	[True, False]
	scale_type	constant		”standardization”

6. RESULTS

In this chapter, performance results of all models are summarized. First, each architecture is analyzed by summarizing all trials, in order to have a general perspective of how the models behave under different parameter combinations. Afterwards, best-performing trials are selected among all trials and the results are reported for each model in a comparative manner.

6.1. Fully-Connected ANN

During the training step, the whole validation set is periodically fed to the model without updating the weights. Each time, the validation error is logged using several metrics such as mean square error and weighted mean absolute percentage error. Best-performing epoch and the weight state, i.e. model object of that epoch are kept for each trial for a further use in testing. Other than that, keeping the information of the best epoch is important to able to see when the model stops improving. In Figure 6.1, distribution of best-performing epochs for all trials of hyperparameter tuning is illustrated for fully-connected ANN. Here, we see that most trials kept improving until the maximum number of epochs reached. This is a signal that the model would keep improving with a higher number of maximum epochs. However, as the number of epochs increases, learning rate decreases due to exponential decay and marginal improvement gained by each iteration also decreases, since the weight updates gets smaller and weights converge to a local solution. Through the course of parameter tuning, HPO algorithm considers several different input feature combinations, listed in Table B.2. Alterations in the input feature combinations are illustrated in Figure 6.2.

The distributions of MSE and WMAPE metrics at each evaluation step, i.e. every four epochs, are illustrated in Figures 6.3 and 6.4. It can be seen that fully-connected trials keep improving until they reach the maximum number of trials. Although it may seem straightforward that increasing the maximum number of epochs is necessary,

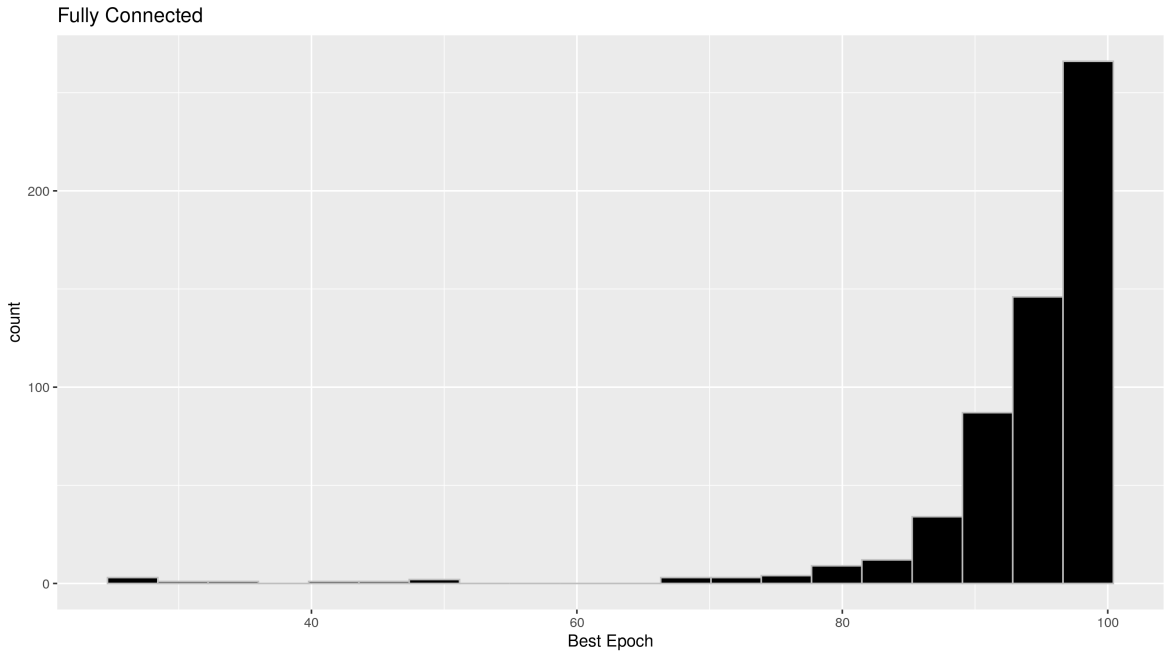

Figure 6.1. Distribution of the best epochs of the completed trials of fully-connected ANN.

doing so would not necessarily yield to a significantly better result due to exponential decay and decreased marginal gain, as mentioned earlier.

6.2. Convolutional Neural Network

CNN approach is similar to the fully-connected layout, in terms of spatial-only embedding of the weather information and not taking sequential time-series dimension into account. For this reason, results of the CNN architecture is comparable to the fully-connected layout, since other approaches also deal with the temporality and hence are more complex.

Similar to the fully-connected model, most of the trials reach their best values at the latest epochs and keep improving with very small margins, as seen in Figure 6.5. Unlike the fully-connected model, CNN performs variably with different input feature combinations, as seen in Figure 6.6. Although the performance gradually becomes

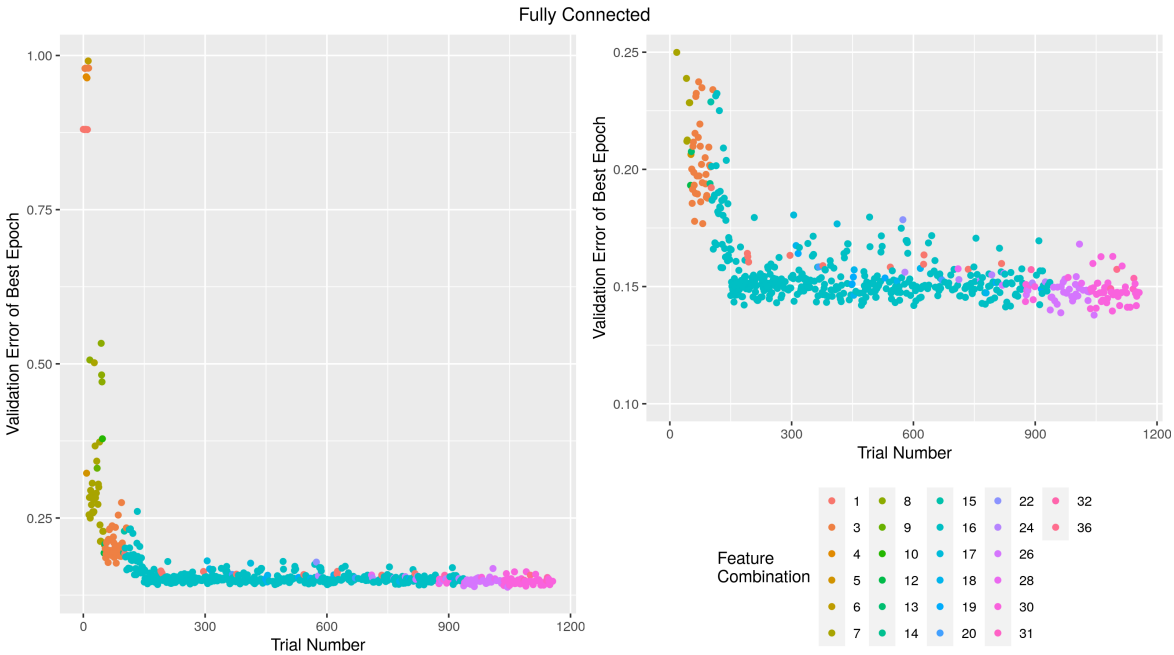

Figure 6.2. The outline of searched feature combinations through the course of trials for the fully-connected ANN. On the right, initial trials with outlier values are omitted for a better view.

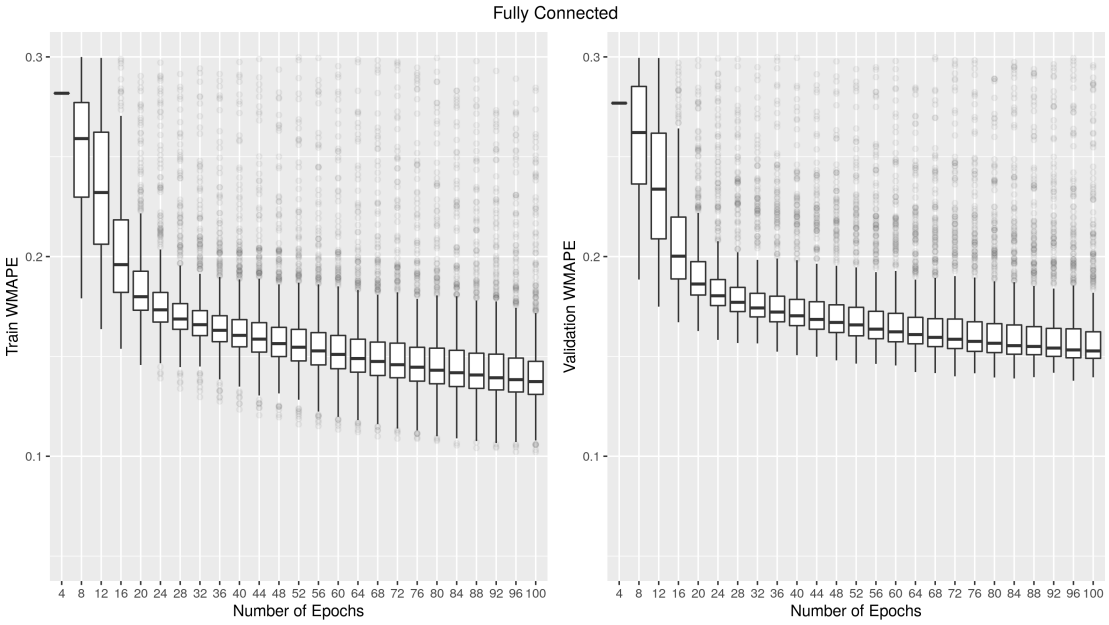

Figure 6.3. Distribution of train and validation WMAPE at each epoch of fully-connected trials.

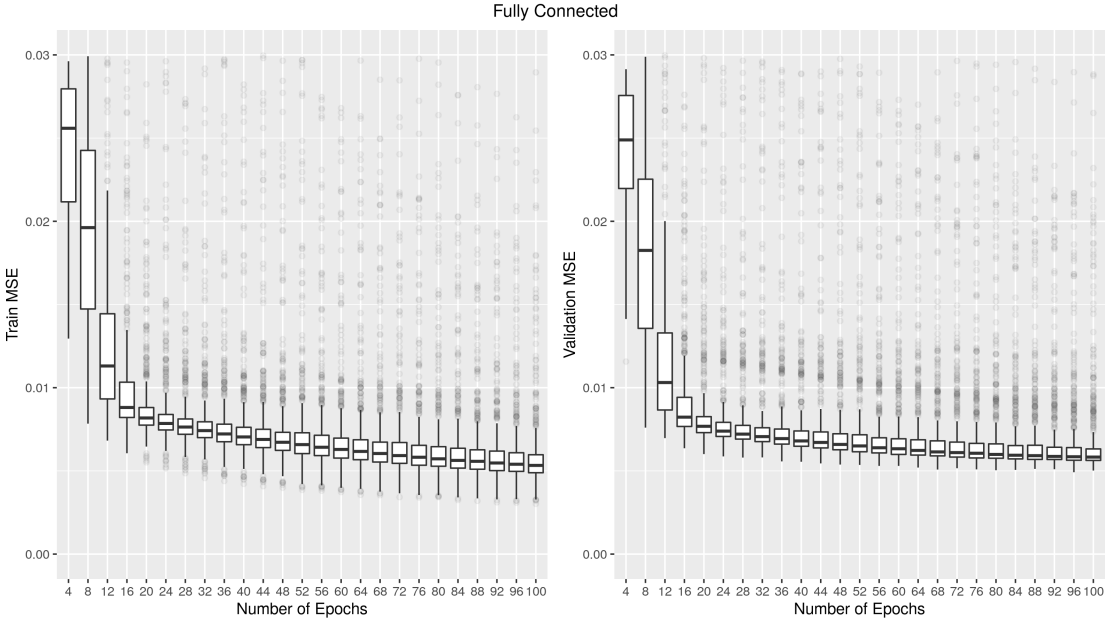

Figure 6.4. Distribution of train and validation MSE at each epoch of fully-connected trials.

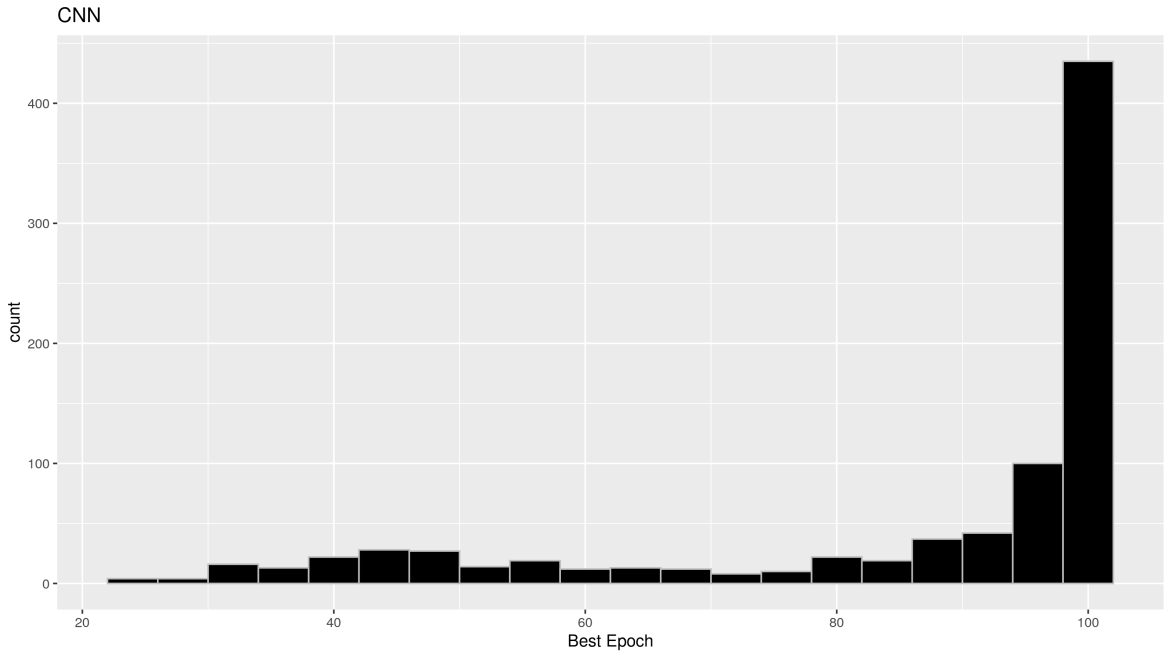

Figure 6.5. Distribution of the best epochs of the completed trials of CNN.

more stable as the HPO proceeds, the tuning algorithm seems to converge easier in the fully-connected case. Also, the behavior of WMAPE and MSE seem to differ from each other, illustrated in Figures 6.7 and 6.8 respectively. Since the target variable is continuous between the interval [0,1], squared error does not penalize the outlier values as it normally would outside of the [0,1] interval. However, WMAPE increases in line with the proportion of the absolute error to the total production. In solar power production, an important part of the total energy is produced during the peak hours and WMAPE metric works better for monitoring this behavior. Therefore, this difference might be signaling the lack of predictive power for the peak hours of the daily production.

6.3. Convolutional LSTM

Unlike FC-ANN and CNN models, ConvLSTM takes sequential time-series dimension into account. In this regard, it is a natural rival to FC-LSTM and PLC-LSTM architectures. Instead of piling-up close to *max_epochs*, best performing epochs

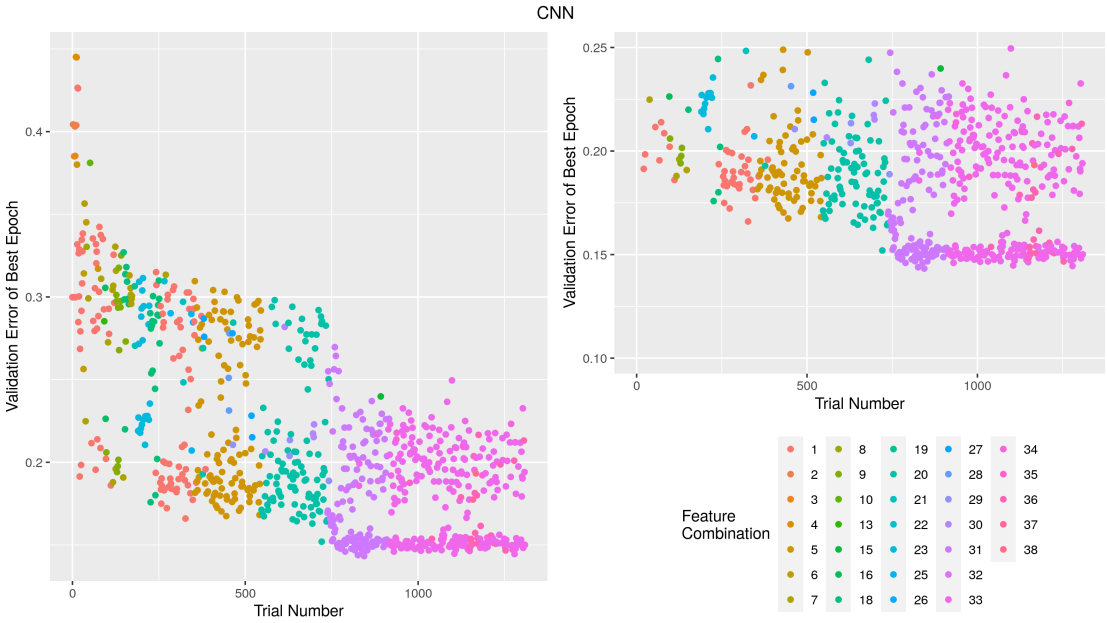

Figure 6.6. The outline of searched feature combinations through the course of trials for the CNN. On the right, initial trials with outlier values are omitted for a better view.

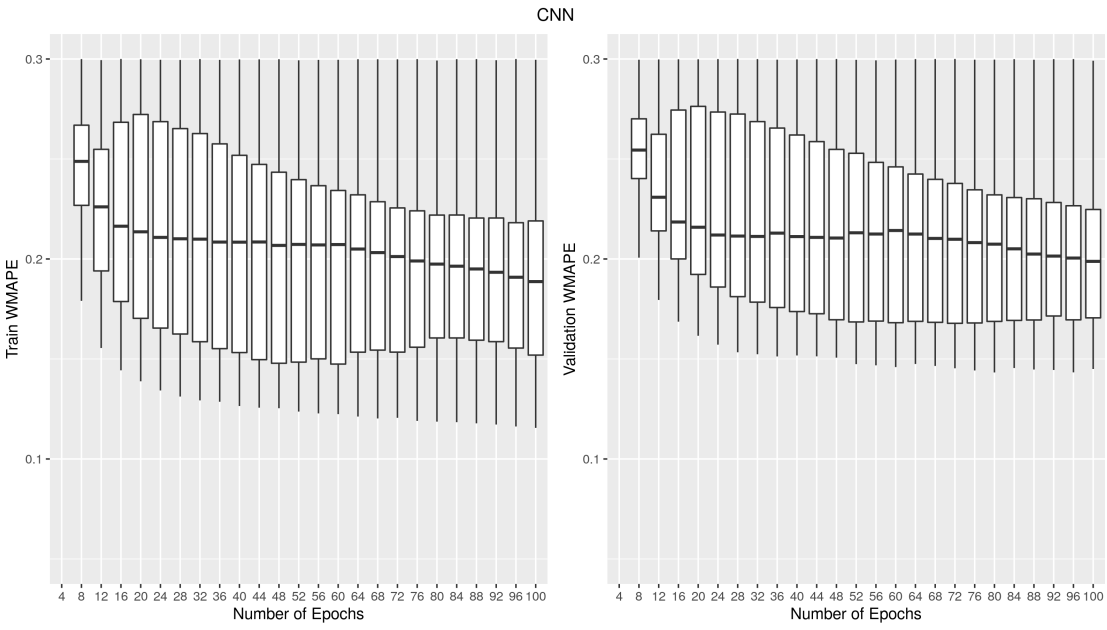

Figure 6.7. Distribution of train and validation WMAPE at each epoch of CNN trials.

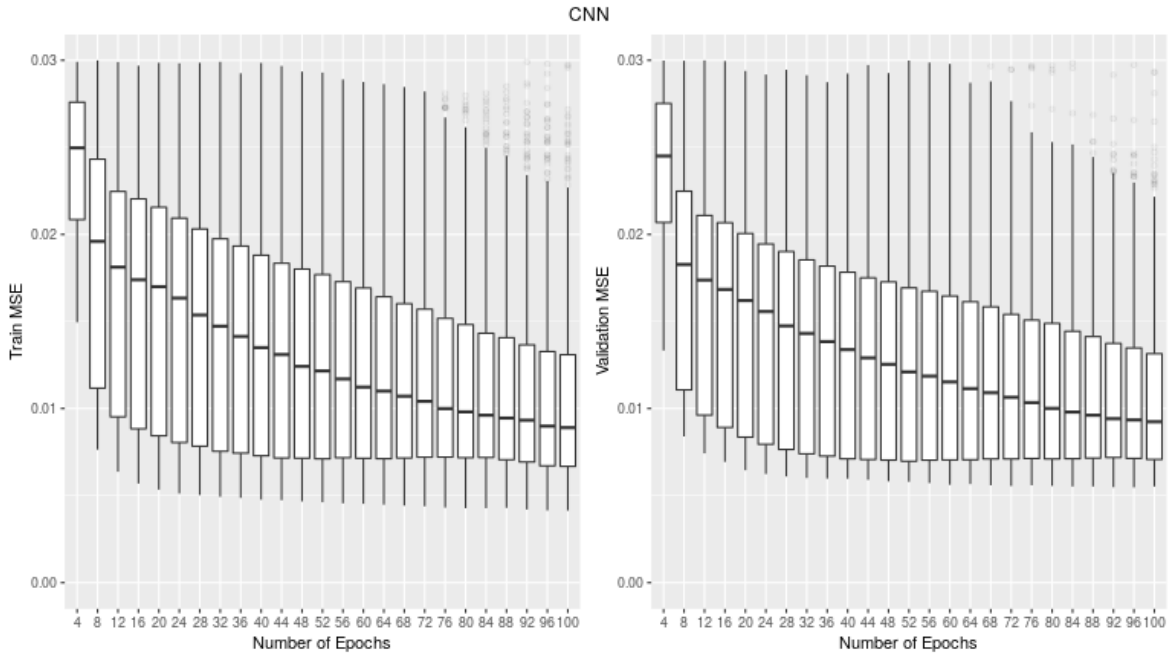

Figure 6.8. Distribution of train and validation MSE at each epoch of CNN trials.

are distributed more evenly compared to spatial-only architectures. This is an expected behavior since ConvLSTM is a more complex architecture and is able fit the train data faster when the HPO tries a relatively high learning rate within the search space.

6.4. Fully-Connected LSTM

Fully-connected LSTM trials produce their best validation errors mostly during the first half of the training, as seen in Figure 6.13. This validates the aforementioned complexity increase with the inclusion of the time-series dimension in recurrent networks, compared to fully-connected and CNN architectures. In Figure 6.14, feature combinations throughout the trials are depicted. Unlike other models, FC-LSTM trials seem to be dominated by a single feature combination.

Performance metrics are illustrated in Figures 6.15 and 6.16. Here, it can again be seen that in general, validation errors hit the lowest sooner than the other models' trials. Although these figures do not represent a single model training procedure but all

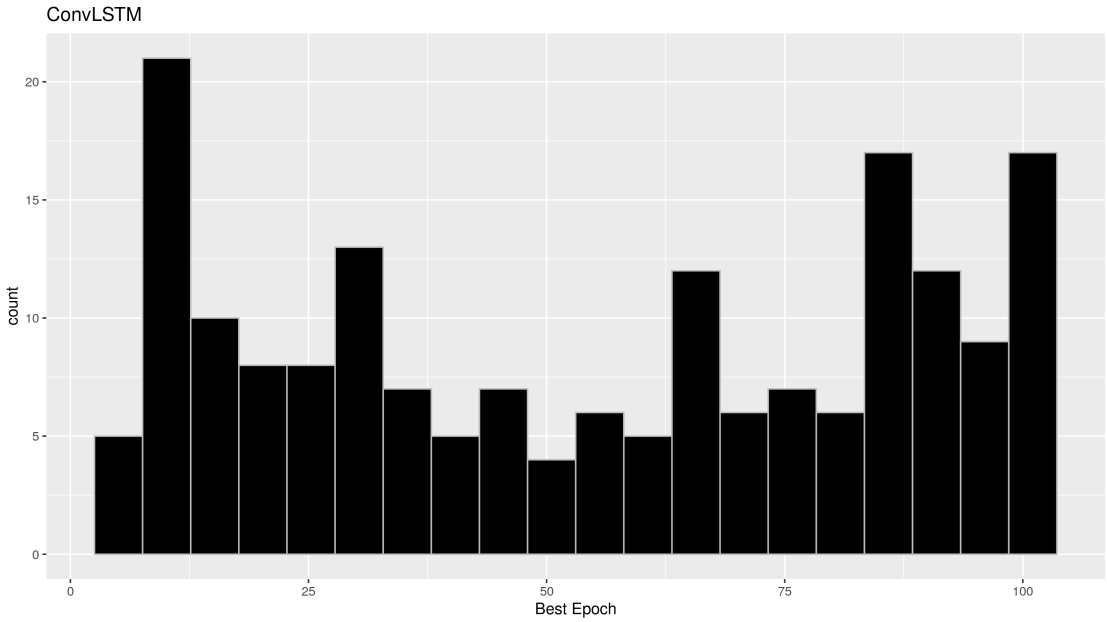

Figure 6.9. Distribution of the best epochs of the completed trials of ConvLSTM.

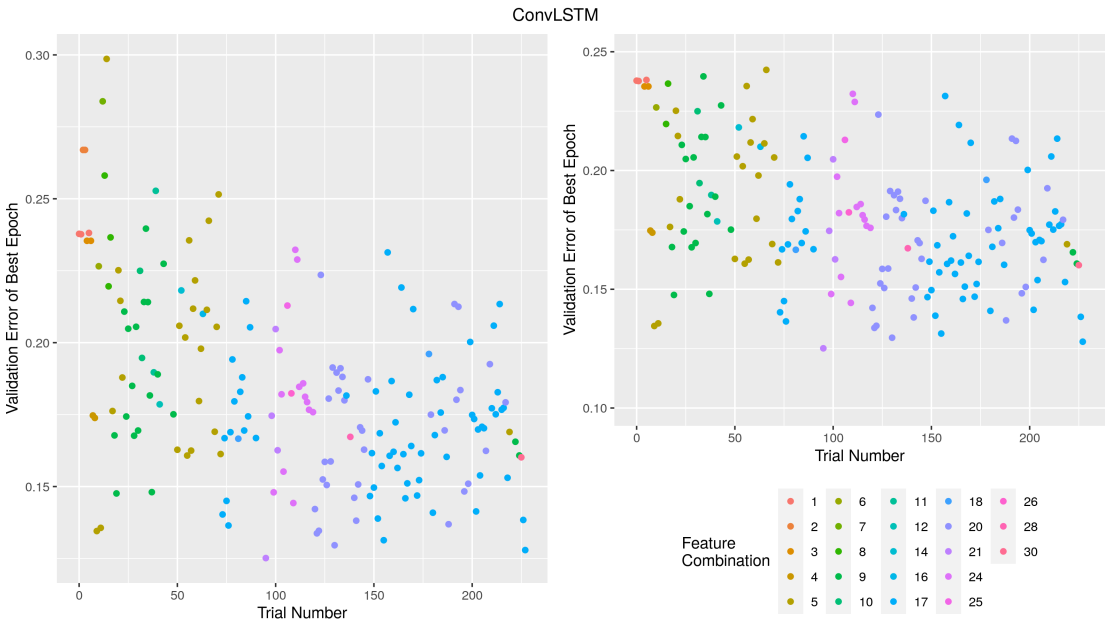

Figure 6.10. The outline of searched feature combinations through the course of trials for the ConvLSTM. On the right, initial trials with outlier values are omitted for a better view.

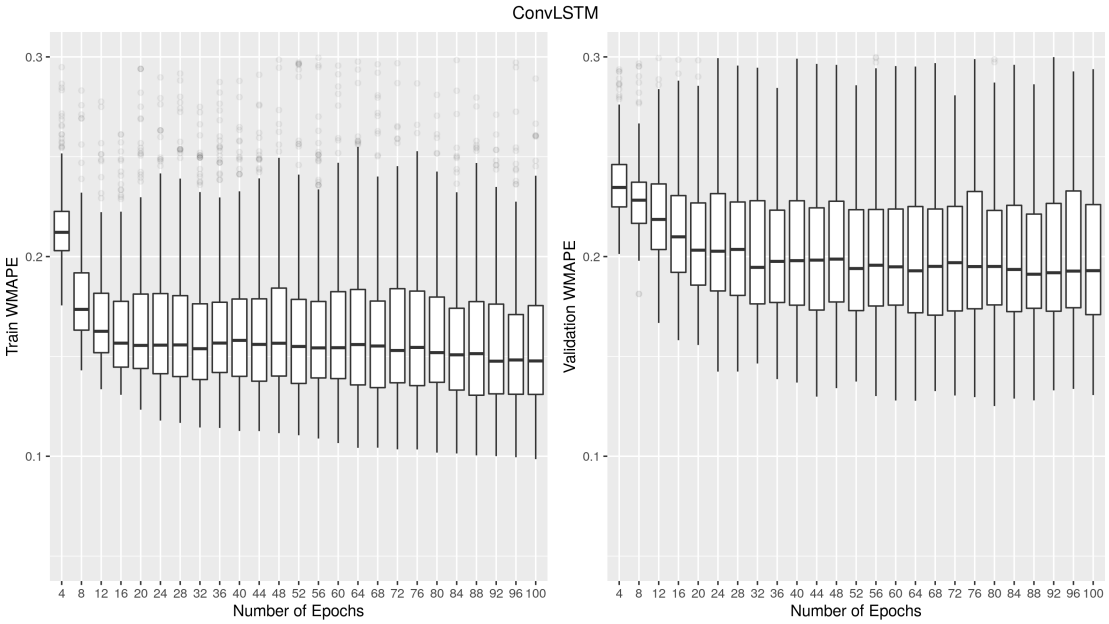

Figure 6.11. Distribution of train and validation WMAPE at each epoch of ConvLSTM trials.

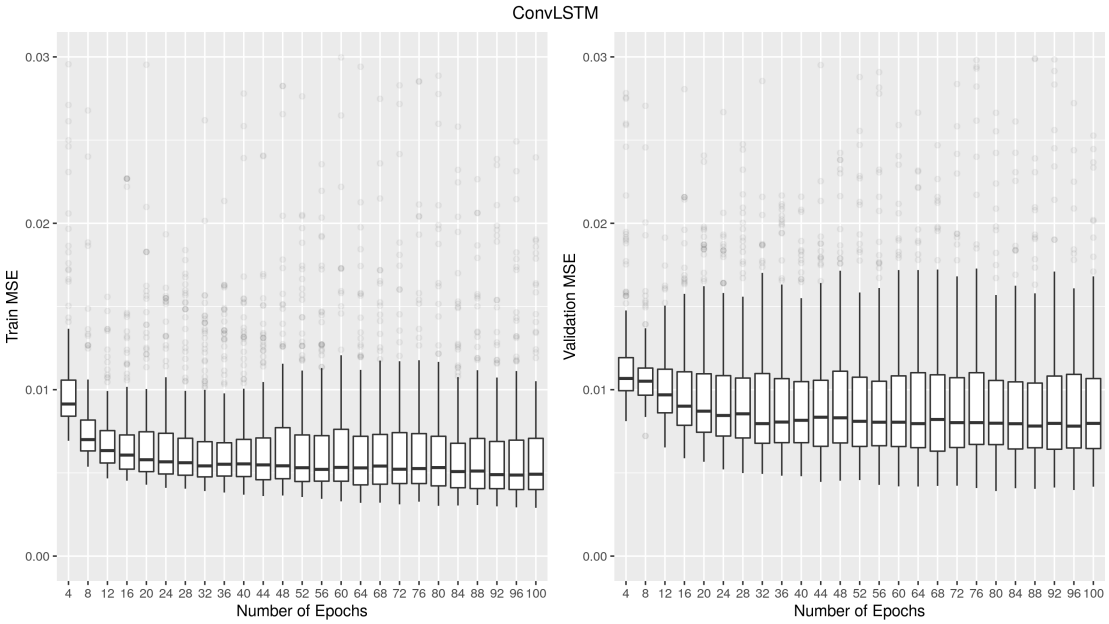

Figure 6.12. Distribution of train and validation MSE at each epoch of ConvLSTM trials.

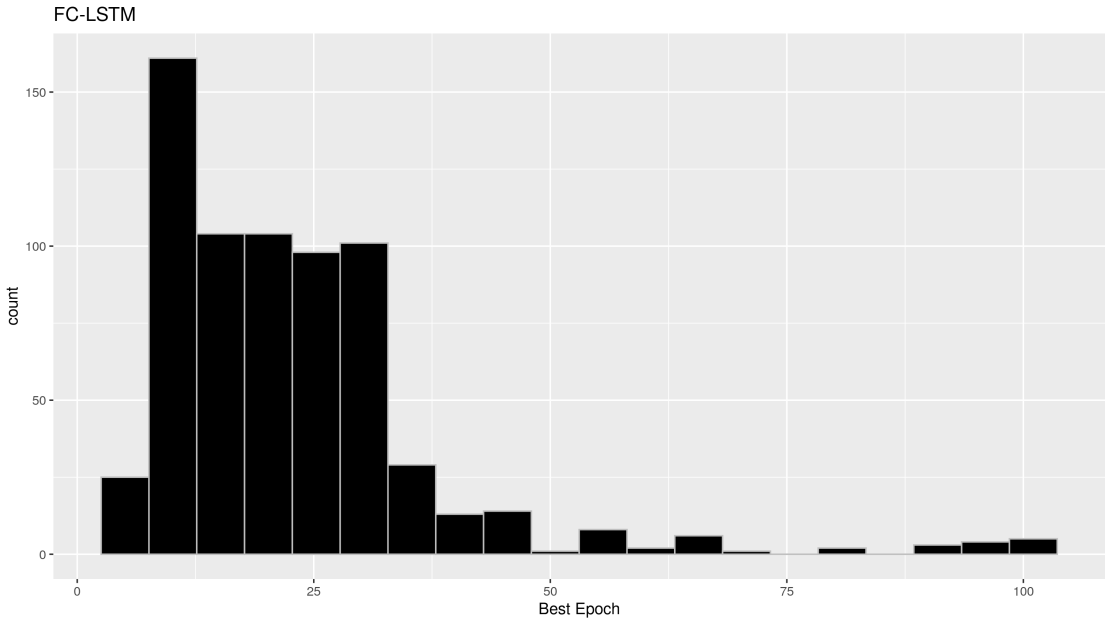

Figure 6.13. Distribution of the best epochs of the completed trials of FC-LSTM.

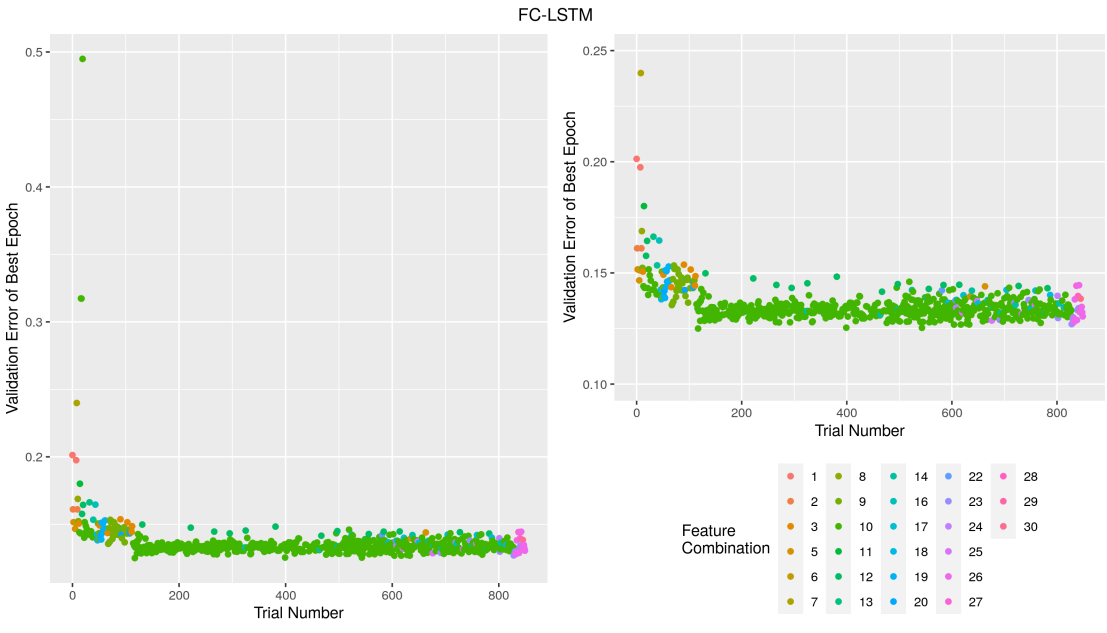

Figure 6.14. The outline of searched feature combinations through the course of trials for the FC-LSTM. On the right, initial trials with outlier values are omitted for a better view.

of them, it can be concluded that the FC-LSTM trials tend to suffer from overfitting in general, since the median train error keeps decreasing for some epochs after the lowest point of the median validation error.

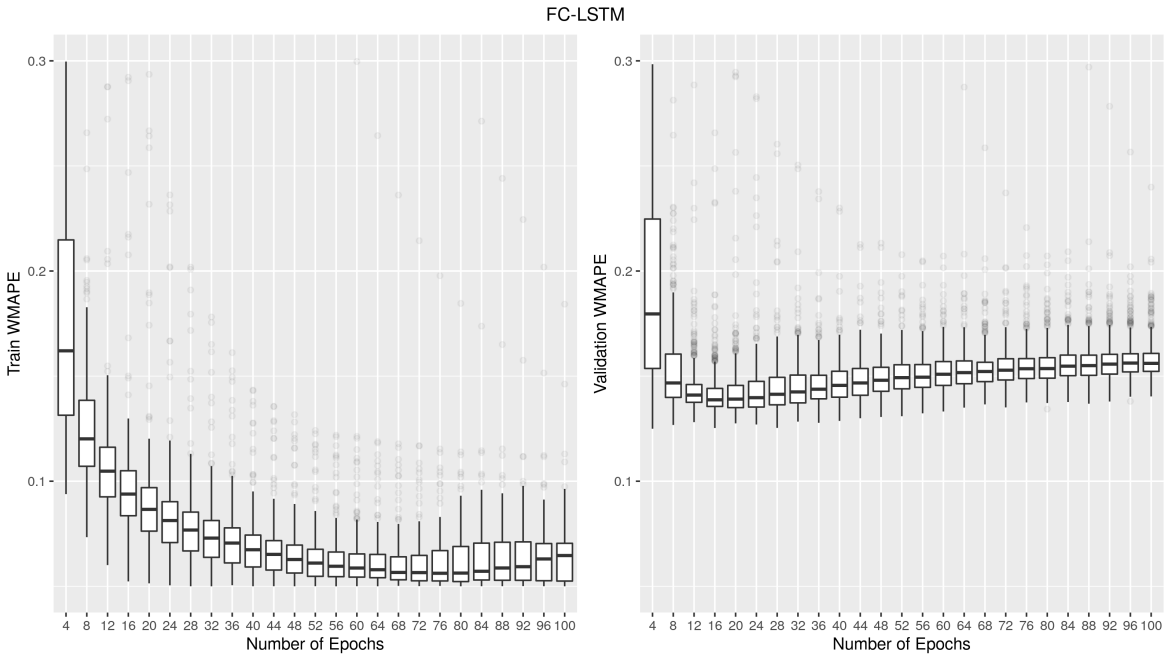

Figure 6.15. Distribution of train and validation WMAPE at each epoch of FC-LSTM trials.

6.5. Parallel Locally-Connected LSTM

Results of the PLC-LSTM trials slightly differ from the FC-LSTM. The distribution of the best epoch is more heavy-tailed compared to the FC-LSTM as seen in Figure 6.17, which means that larger number of trials kept improving until the *max_epochs* is reached. Trials seem to converge around two feature combinations and the latter seems to perform slightly better as the trial numbers increase in Figure 6.18.

Train and validation errors show that PLC-LSTM trials also fit to the data on early epochs, similar to FC-LSTM trials. However, train error decreases in a smoother fashion, hence the area of the gap between median train and validation curves is less than the gap of FC-LSTM. This is a promising indicator that PLC-LSTM is more

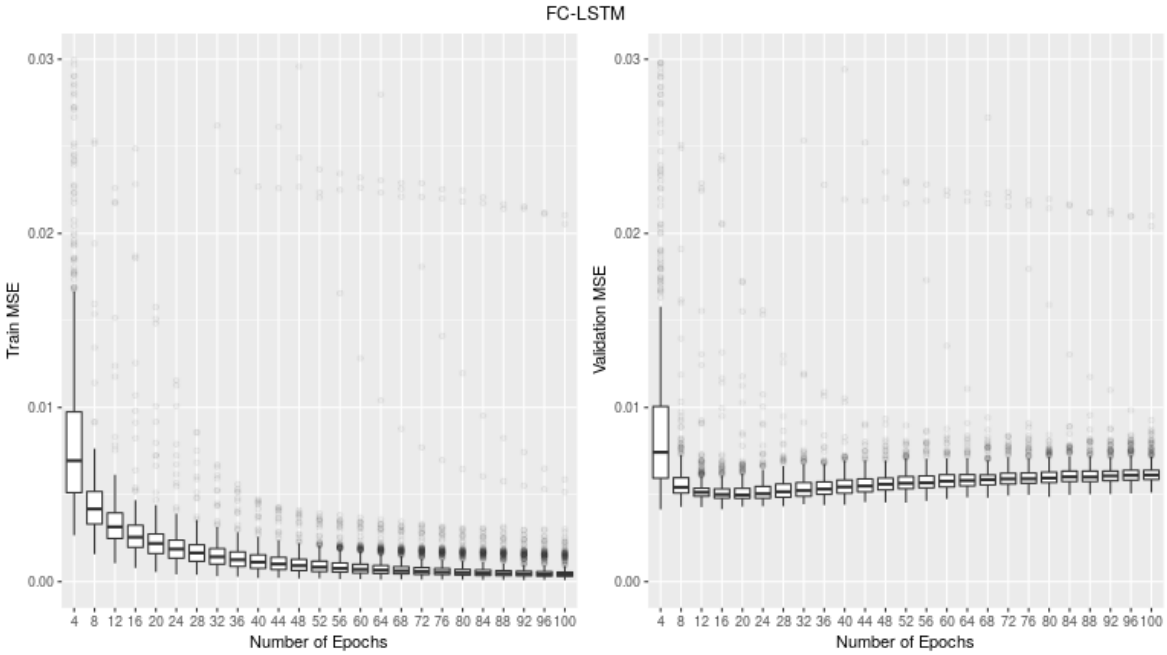

Figure 6.16. Distribution of train and validation MSE at each epoch of FC-LSTM trials.

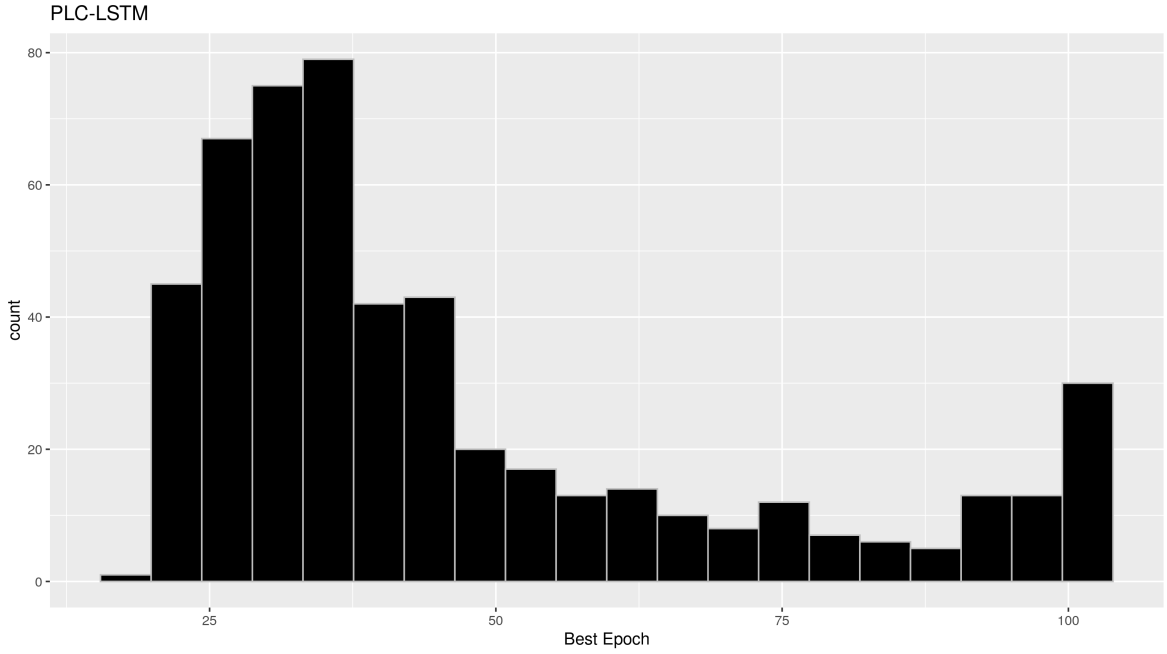

Figure 6.17. Distribution of the best epochs of the completed trials of PLC-LSTM.

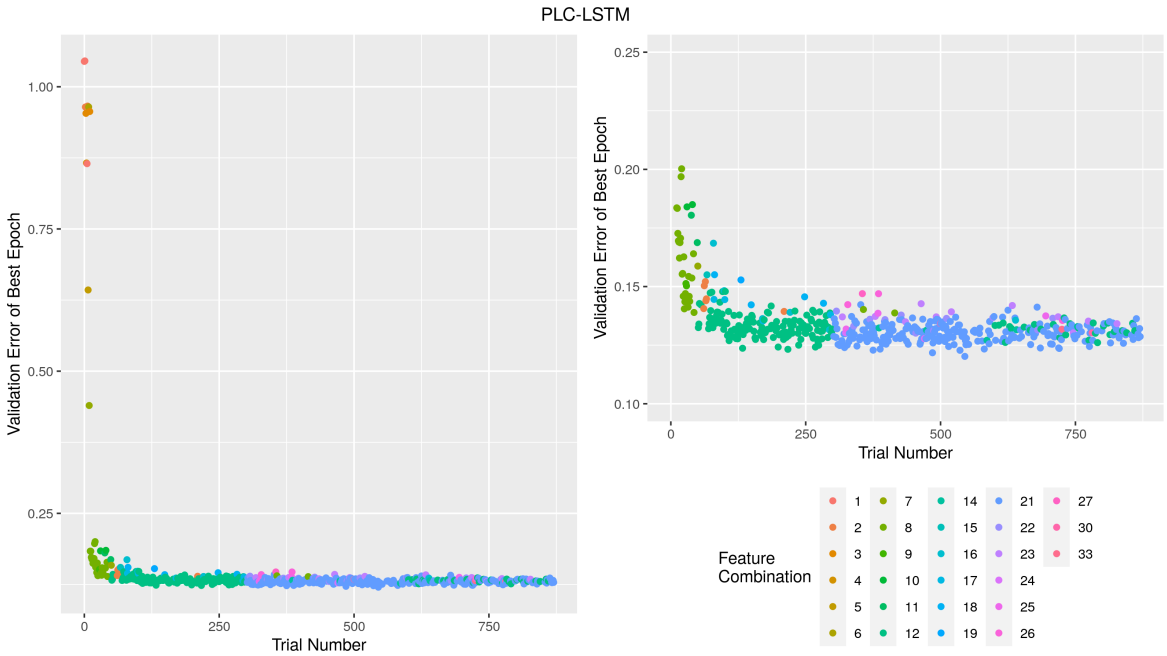

Figure 6.18. The outline of searched feature combinations through the course of trials for the PLC-LSTM. On the right, initial trials with outlier values are omitted for a better view.

robust against changing parameters, in terms of overfitting.

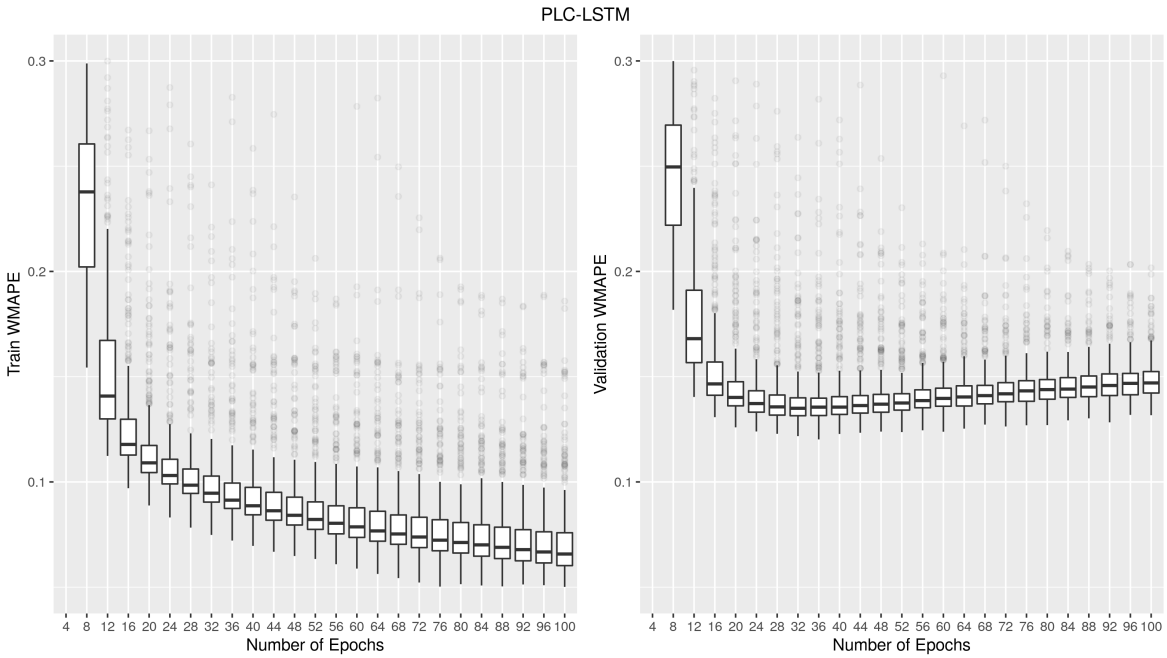

Figure 6.19. Distribution of train and validation WMAPE at each epoch of PLC-LSTM trials.

6.6. Best-Performing Trials

In this section, best-performing trials of each model are reported. After analyzing each model's behavior over all trials, model selection criteria are determined with respect to two key aspects that represent each trial. First criterion considers training and validation consistency, which is obtained by summing up the differences between the validation and training errors at each epoch. A *consistency_rank* is assigned to each trial in increasing order, i.e. lowest value taking the lowest rank. Second criterion accounts for trial performance in terms of validation error. The minimum validation WMAPE error obtained until the *max_epochs* is considered as the performance of that trial. A *performance_rank* is assigned to each trial in the order of increasing WMAPE, similar to the *consistency_rank*. Finally, winners are selected by summing up both ranks and selecting the lowest sum for each model. This approach is adopted in order to prevent an overfitted trial to be selected in a case when only its validation error is

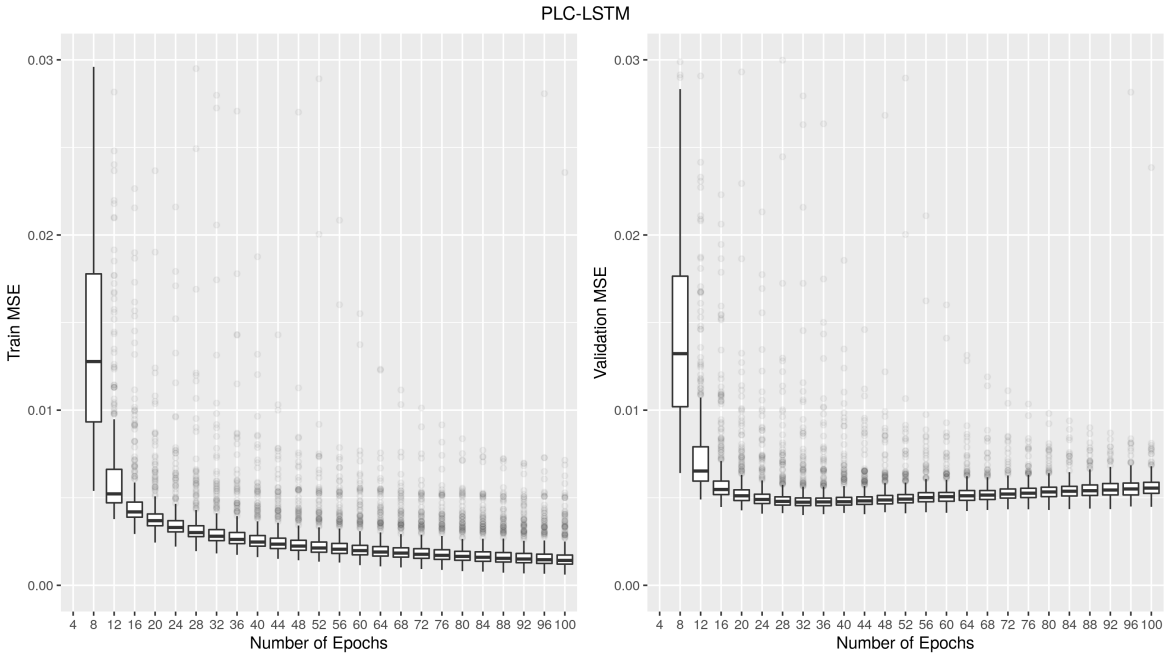

Figure 6.20. Distribution of train and validation MSE at each epoch of PLC-LSTM trials.

considered. Similarly, it is more likely to end up with a high-error trial, if training and validation consistency was considered as the only selection criterion. This way, it is aimed find a sweet spot between underfitting and overfitting among all trials.

Selected trials for the fully-connected and CNN layouts are illustrated in Figures 6.21 and 6.22 respectively. Train and validation errors at each epoch reveal that both trials provide a highly consistent and smooth learning curves, especially CNN. The gap between the training and validation errors of the FC-ANN model tend to increase more than the CNN model. However, when the hourly distribution of error is examined, it is seen that FC-ANN architecture provides more unbiased predictions and in the median, CNN underestimates the production during peak hours, both for training and validation sets. This is an undesired result since the mid-day hours are the most crucial times for the solar power forecasting problem.

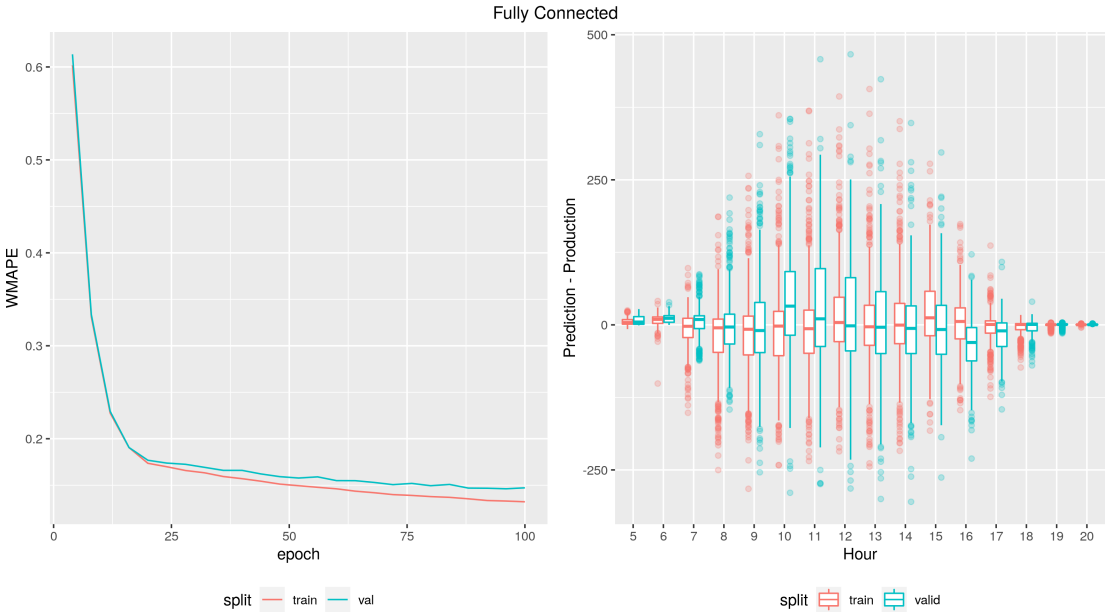

Figure 6.21. Train and validation error at each epoch (left) and distribution of error at each hour (right) for the fully-connected model

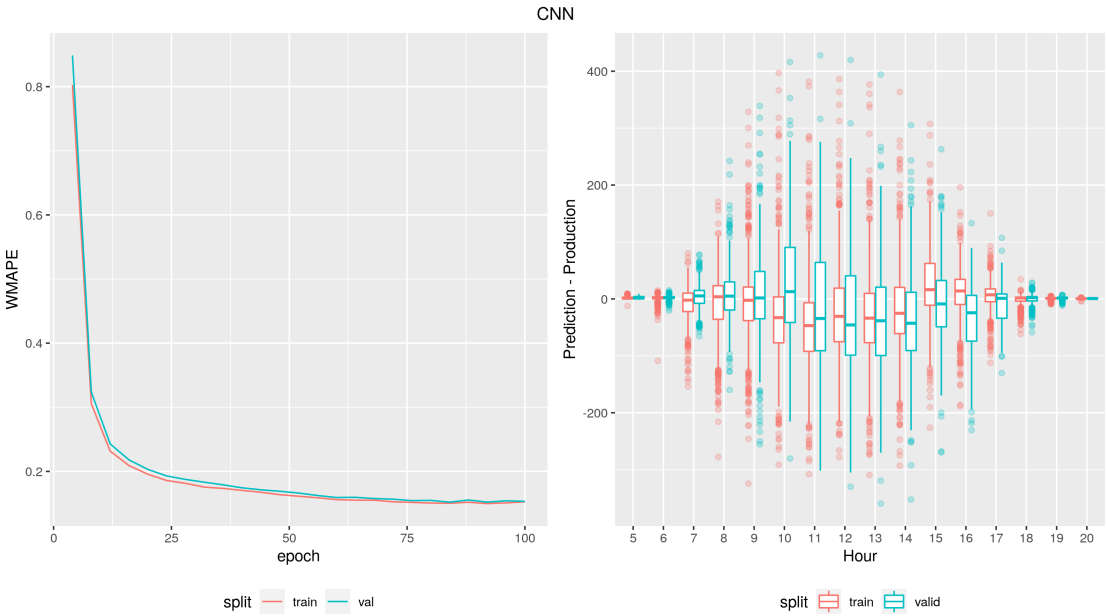

Figure 6.22. Train and validation error at each epoch (left) and distribution of error at each hour (right) for the CNN model

Convolutional LSTM model, illustrated in Figure 6.23, does not yield to a learning curve that is as smooth as the previous ones. Hourly distribution of error also points out that there are inconsistencies between the behavior of train and validation sets. The model underestimates the morning hours of the training set whereas it overestimates the same hours of the validation set. Although the peak hours are more consistent in terms of training and validation errors, production is underestimated in the median, similar to the CNN architecture.

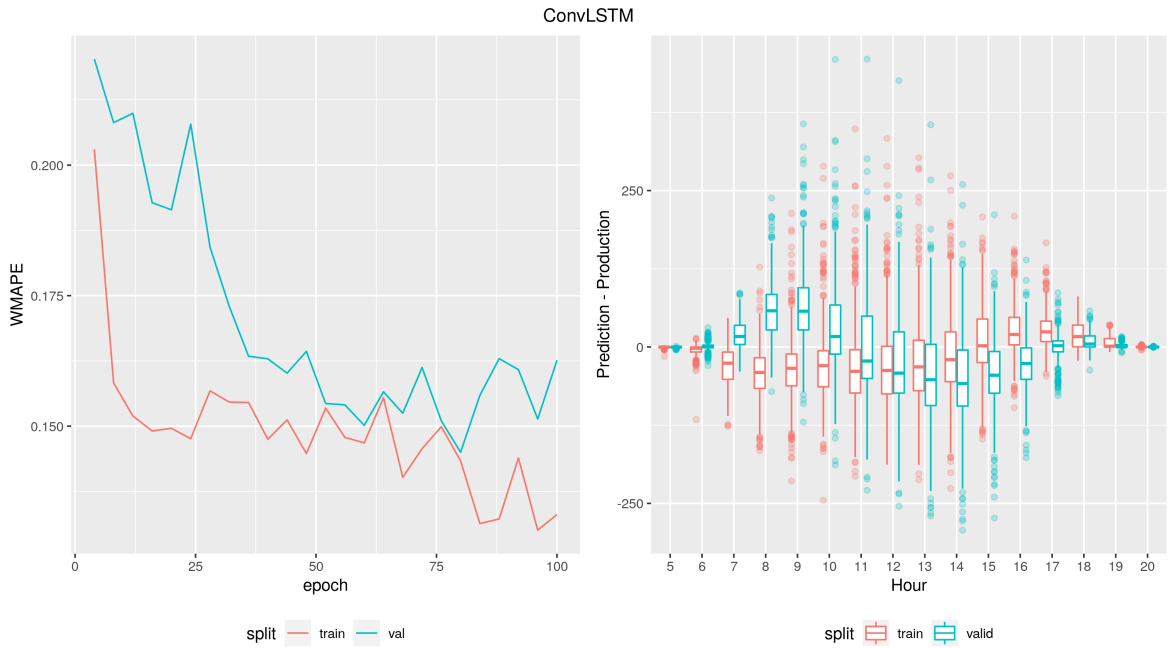

Figure 6.23. Train and validation error at each epoch (left) and distribution of error at each hour (right) for the ConvLSTM model

Selected trials for the FC-LSTM and PLC-LSTM layouts are illustrated in Figures 6.24 and 6.25 respectively. At first glance, both learning curves seem to have smoother shapes than the ConvLSTM model. Also, compared to less complex models like FC-ANN and CNN, learning curves validates the anticipated overfitting introduced by the increased complexity. The gap between the training and validation errors of the FC-LSTM model tend to increase more than the PLC-LSTM model. When the drastic decrease in the train error is taken into account, it can be said that the FC-LSTM model is able to memorize the training observations. This causes the model to lose its

ability of generalization and the validation error starts to increase towards the end of the course of training. This effect does not take place in the selected PLC-LSTM trial. Even though the PLC-LSTM has a higher gap between training and validation curves compared to less complex models, the learning curve converges at a lower validation error. This is a promising sign for the desired complexity of the proposed model, as described in Section 4.2. When the hourly distribution of error is examined, it is seen that both FC-LSTM and PLC-LSTM architectures provide more unbiased predictions during peak hours relative to the previous models.

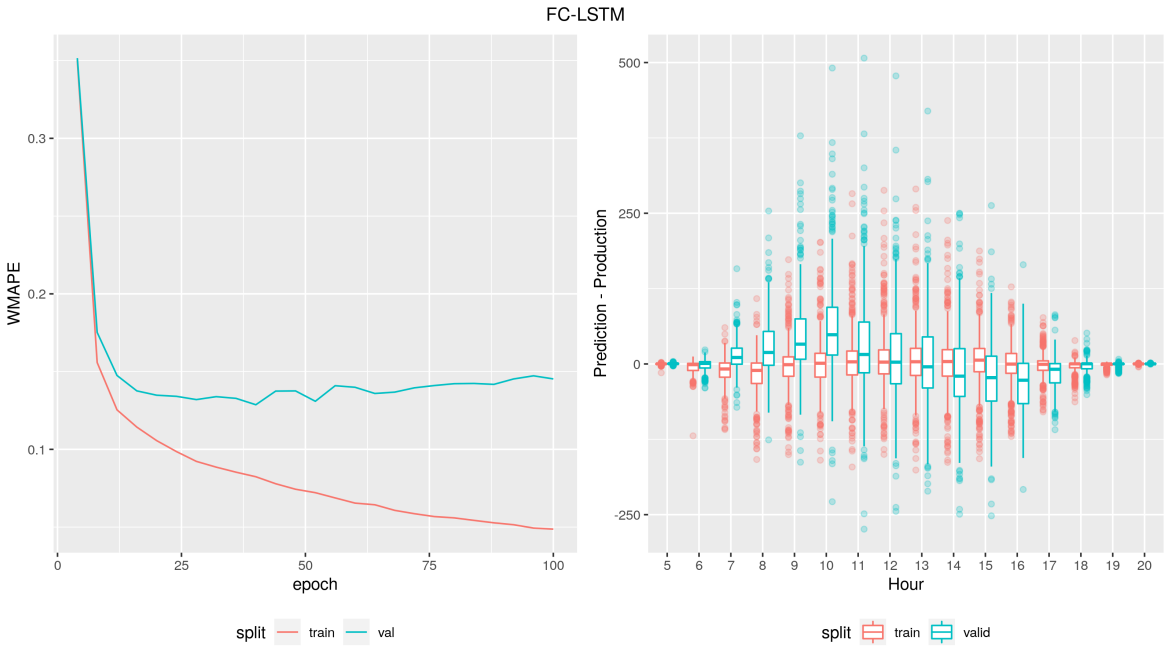

Figure 6.24. Train and validation error at each epoch (left) and distribution of error at each hour (right) for the FC-LSTM model

After selecting the best-performing trial for each model, observations in the test set are predicted using the model weights saved after the best epoch. The results of each model are reported in the Tables 6.1 and C.1, together with the benchmarks mentioned in section 3.1. All models perform better than benchmarks regarding the overall WMAPE and $\text{DWMAPE}_{\text{med}}$. Hyndman-Khandakar algorithm ends up with $\text{SARIMAX}(3,0,0)(2,1,0)_{24}$ model, which performs better than PM_{48}, yet over-performed by PM_{24}. As expected, Individual Naive Model (INM) performs better than

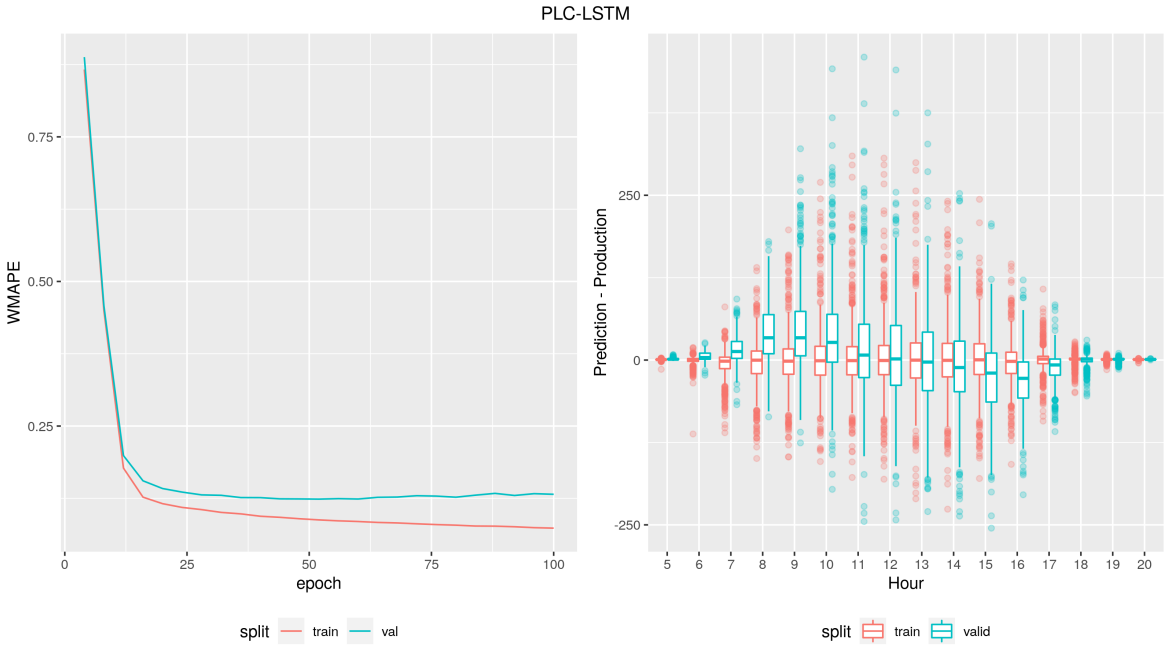

Figure 6.25. Train and validation error at each epoch (left) and distribution of error at each hour (right) for the PLC-LSTM model

persistence models, i.e PM_{24} and PM_{48}, since it makes use of the weather information for predicting the production of each plant individually. It is important to note that deep learning models does not integrate individual production series of each plant into their architectures. Instead, they use the aggregated production series as the target variable and INM makes a tough benchmark in that sense. It enables us to see how good a naive approach is able to get when provided with a higher resolution production data. Also, it is informative to see the amount of the predictive power provided by the DL modeling approaches, even though they are trained with the aggregated output.

According to test WMAPE results, CNN and FC-ANN perform slightly worse than the higher-complexity models. In line with above training and validation results, CNN results in a higher bias in the test set. Between the two, fully-connected model is more comparable to the recurrent models, in terms of daily median WMAPE ($DWMAPE_{med}$) and daily percentage bias ($DPBias_{med}$) metrics. However, standard deviation of these daily metrics tend to be higher than the recurrent models, meaning

that FC-ANN has a higher spread in terms of error. Nevertheless, FC-ANN architecture yields promising test results despite its model simplicity. Among the recurrent models, ConvLSTM has the highest overall PBias, WMAPE, daily median WMAPE ($\text{DWMAPE}_{\text{med}}$) in the test set. However, standard deviation of the daily WMAPE ($\text{DWMAPE}_{\text{sd}}$) and daily percentage bias ($\text{DPBias}_{\text{sd}}$) are the lowest among other models. This mean that although it has a higher error in overall and daily median metrics, ConvLSTM is more robust to changing weather conditions since the spreads of the daily error metrics are narrower compared to other two recurrent architectures. In a similar vein, PLC-LSTM has a narrower spread compared to FC-LSTM, although daily median WMAPE ($\text{DWMAPE}_{\text{med}}$) and daily percentage bias ($\text{DPBias}_{\text{med}}$) values are very close to each other. Here, test results confirm that the intended complexity level is achieved with the proposed PLC-LSTM model, as mentioned in Section 4.2.

Table 6.1. Test results of all models, together with the benchmarks.

Model	**WMAPE**	**PBias**	**$\text{DWMAPE}_{\text{med}}$**	**$\text{DWMAPE}_{\text{sd}}$**	**$\text{DPBias}_{\text{med}}$**	**$\text{DPBias}_{\text{sd}}$**
INM	0.2538	0.0896	0.2432	0.3002	0.1167	0.3857
P24	0.3552	-0.0058	0.3250	0.6534	0.0024	0.7957
P48	0.4661	-0.0181	0.4504	0.7906	-0.0736	0.9926
$\text{SARIMAX}(3,0,0)(2,1,0)_{24}$	0.4153	-0.0206	0.3934	0.6803	-0.0015	0.8521
ConvLSTM	0.2108	-0.0350	0.2036	0.1277	-0.0333	0.2228
CNN	0.2215	-0.0506	0.2102	0.2161	-0.0561	0.3219
Fully Connected	0.2244	-0.0003	0.1985	0.2743	0.0194	0.3644
FC-LSTM	0.1959	0.0170	0.1947	0.2273	-0.0025	0.3052
PLC-LSTM	0.2021	0.0129	0.1978	0.1945	0.0097	0.2727

Monthly test results shown in Table C.1 demonstrate that there is no single best model that outperforms the others at all times. Interestingly, non-sequential models, i.e. CNN and FC-ANN, performs slightly better than the sequential ones in the first and second months respectively. On the contrary, sequential models perform significantly better in the third and forth months, compared to non-sequential models. this results may indicate that the sequential models gave more emphasis on the observations with a higher level of production. Also, the gap between the INM and the best-performing DL model is narrower during the winter months. This signals a potential improvement that could be made using the individual series in DL models, especially for the winter.

7. CONCLUSION

Renewable energy is one of the most important solutions that provide us an opportunity for reducing carbon emissions, in the fight against global climate crisis. Since the amount of renewable energy produced is fully dependent on the weather conditions, these solutions lead to an increase in the stochasticity of the overall energy supply at a particular grid. In order to cope with this uncertainty, decision makers make use of statistical modeling approaches. However, conventional methods are not sufficient since predicting the electricity production of hundreds of distinct power plants scattered across a large area require spatiotemporal forecasting techniques.

In this study, alternative deep learning architectures are implemented to be able to compare the changes in forecasting power with the changing model complexity. Each architecture has its own unique place within the scale of the bias-variance trade-off The proposed parallel locally-connected LSTM (PLC-LSTM) architecture offers a sweet spot between the bias of ConvLSTM and the variance of FC-LSTM architectures, and the results are promising in terms of overall generalization ability. Another outcome of this study is to validate the importance of a heuristic-based hyperparameter optimization process. It can be seen that even a simplistic approach such as the fully-connected ANN architecture can perform quite close to models with a higher degree of complexity, when a good combination of parameters is obtained. Although it is a computationally expensive approach, it is very hard to come up with a similar result using a brute-force grid search.

There are a number possible future work that can be done to extend the scope of this study. Firstly, a sliding-window cross validation setup would increase the likelihood of selecting a better-fit model. However, retraining the model from the scratch for each sliding validation fold takes a considerable amount of time for a deep learning model. An approach similar to transfer learning can be adopted which uses the weights of the model trained in the previous fold to be fed only with the newly added

observations belonging to the next fold. Another good extension would be fine tuning of other essential parameters of PLC-LSTM, such as *unfold_size* and *unfold_step*, which were kept constant to be able to have a smaller hyperparameter search space. In addition, providing a larger spatial area and a higher-resolution NWP grid would highlight the ability of PLC-LSTM to scale up spatially without increasing the complexity of each locally-connected module. Lastly and most importantly, the architecture of PLC-LSTM is quite convenient for learning with higher resolution target data, instead of the aggregated output. Production of each plant can be learned individually or could be easily aggregated within local kernels before reaching a region-wise aggregated output. As the individual naive model (INM) signals the potential improvement of using individual production series, an extension to PLC-LSTM can be made in this manner within the scope of a future work.

Spatiotemporal solar power forecasting is a developing field of study, especially within the deep learning domain. Although existing conventional or cross-domain architectures work well to some extent, models that are built specifically for the spatiotemporal PV forecasting might be the key to unlock the predictive potential of deep learning in this area and the proposed PLC-LSTM is one of the first architectures that adapt to the specific needs of this problem.

REFERENCES

1. Veličković, P., "TikZ", `https://github.com/PetarV-/TikZ`, 2018, last accessed on April 23, 2020.

2. Dumoulin, V. and F. Visin, "A guide to convolution arithmetic for deep learning", *arXiv:1603.07285 [cs, stat]*, 2018, `http://arxiv.org/abs/1603.07285`, arXiv: 1603.07285.

3. Shi, X., Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong and W.-c. Woo, "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting", *arXiv:1506.04214 [cs]*, 2015, `http://arxiv.org/abs/1506.04214`, arXiv: 1506.04214.

4. Agency, I. E., *Renewables 2019: Analysis and forecasts to 2024*, Market Report Series: Renewables, OECD, 2019.

5. Mathe, J., N. Miolane, N. Sebastien and J. Lequeux, "PVNet: A LRCN Architecture for Spatio-Temporal Photovoltaic PowerForecasting from Numerical Weather Prediction", *arXiv:1902.01453 [cs]*, 2020, `http://arxiv.org/abs/1902.01453`, arXiv: 1902.01453.

6. Chai, S., Z. Xu, Y. Jia and W. K. Wong, "A Robust Spatiotemporal Forecasting Framework for Photovoltaic Generation", *IEEE Transactions on Smart Grid*, p. 1–1, 2020.

7. Rigollier, C., O. Bauer and L. Wald, "On the clear sky model of the ESRA — European Solar Radiation Atlas — with respect to the heliosat method", *Solar Energy*, Vol. 68, No. 1, p. 33–48, 2000.

8. Mueller, R., "Rethinking satellite-based solar irradiance modellingThe SOLIS clear-sky module", *Remote Sensing of Environment*, Vol. 91, No. 2, p. 160–174,

2004.

9. Molineaux, B., P. Ineichen and N. O'Neill, "Equivalence of pyrheliometric and monochromatic aerosol optical depths at a single key wavelength", *Applied Optics*, Vol. 37, No. 30, p. 7008, 1998.

10. Ineichen, P. and R. Perez, "A new airmass independent formulation for the Linke turbidity coefficient", *Solar Energy*, Vol. 73, No. 3, p. 151–157, 2002.

11. Gueymard, C. A., "REST2: High-performance solar radiation model for cloudless-sky irradiance, illuminance, and photosynthetically active radiation – Validation with a benchmark dataset", *Solar Energy*, Vol. 82, No. 3, p. 272–285, 2008.

12. Inman, R. H., H. T. Pedro and C. F. Coimbra, "Solar forecasting methods for renewable energy integration", *Progress in Energy and Combustion Science*, Vol. 39, No. 6, pp. 535–576, 2013, `https://doi.org/10.1016/j.pecs.2013.06.002`.

13. Yang, D., Z. Ye, L. H. I. Lim and Z. Dong, "Very short term irradiance forecasting using the lasso", *Solar Energy*, Vol. 114, p. 314–326, 2015.

14. Aryaputera, A. W., D. Yang, L. Zhao and W. M. Walsh, "Very short-term irradiance forecasting at unobserved locations using spatio-temporal kriging", *Solar Energy*, Vol. 122, p. 1266–1278, 2015.

15. Amaro e Silva, R., S. E. Haupt and M. C. Brito, "A regime-based approach for integrating wind information in spatio-temporal solar forecasting models", *Journal of Renewable and Sustainable Energy*, Vol. 11, No. 5, p. 056102, 2019.

16. Yang, D., Z. Dong, T. Reindl, P. Jirutitijaroen and W. M. Walsh, "Solar irradiance forecasting using spatio-temporal empirical kriging and vector autoregressive models with parameter shrinkage", *Solar Energy*, Vol. 103, p. 550–562, 2014.

17. Lan, H., C. Zhang, Y.-Y. Hong, Y. He and S. Wen, "Day-ahead spatiotempo-

ral solar irradiation forecasting using frequency-based hybrid principal component analysis and neural network", *Applied Energy*, Vol. 247, p. 389–402, 2019.

18. Lonij, V. P., A. E. Brooks, A. D. Cronin, M. Leuthold and K. Koch, "Intra-hour forecasts of solar power production using measurements from a network of irradiance sensors", *Solar Energy*, Vol. 97, p. 58–66, 2013.

19. Tascikaraoglu, A., B. M. Sanandaji, G. Chicco, V. Cocina, F. Spertino, O. Erdinc, N. G. Paterakis and J. P. Catalão, "Compressive Spatio-Temporal Forecasting of Meteorological Quantities and Photovoltaic Power", *IEEE Transactions on Sustainable Energy*, Vol. 7, No. 3, p. 1295–1305, 2016.

20. Bessa, R. J., A. Trindade and V. Miranda, "Spatial-Temporal Solar Power Forecasting for Smart Grids", *IEEE Transactions on Industrial Informatics*, Vol. 11, No. 1, p. 232–241, 2015.

21. Bessa, R., A. Trindade, C. S. Silva and V. Miranda, "Probabilistic solar power forecasting in smart grids using distributed information", *International Journal of Electrical Power and Energy Systems*, Vol. 72, p. 16–23, 2015.

22. Zhang, B., P. Dehghanian and M. Kezunovic, "Spatial-temporal solar power forecast through use of Gaussian Conditional Random Fields", *2016 IEEE Power and Energy Society General Meeting (PESGM)*, p. 1–5, IEEE, 2016, `http://ieeexplore.ieee.org/document/7741503/`.

23. Agoua, X. G., R. Girard and G. Kariniotakis, "Probabilistic Models for Spatio-Temporal Photovoltaic Power Forecasting", *IEEE Transactions on Sustainable Energy*, Vol. 10, No. 2, p. 780–789, 2019.

24. Nam, S. and J. Hur, "A hybrid spatio-temporal forecasting of solar generating resources for grid integration", *Energy*, Vol. 177, p. 503–510, 2019.

25. Yang, C., A. A. Thatte and L. Xie, "Multitime-Scale Data-Driven Spatio-Temporal

Forecast of Photovoltaic Generation", *IEEE Transactions on Sustainable Energy*, Vol. 6, No. 1, p. 104–112, 2015.

26. Dambreville, R., P. Blanc, J. Chanussot and D. Boldo, "Very short term forecasting of the Global Horizontal Irradiance using a spatio-temporal autoregressive model", *Renewable Energy*, Vol. 72, p. 291–300, 2014.

27. Yang, D., J. Kleissl, C. A. Gueymard, H. T. Pedro and C. F. Coimbra, "History and trends in solar irradiance and PV power forecasting: A preliminary assessment and review using text mining", *Solar Energy*, Vol. 168, pp. 60–101, 2018, https://doi.org/10.1016/j.solener.2017.11.023.

28. e Silva, R. A. and M. C. Brito, "Impact of network layout and time resolution on spatio-temporal solar forecasting", *Solar Energy*, Vol. 163, pp. 329–337, 2018, https://doi.org/10.1016/j.solener.2018.01.095.

29. Zagouras, A., H. T. Pedro and C. F. Coimbra, "On the role of lagged exogenous variables and spatio–temporal correlations in improving the accuracy of solar forecasting methods", *Renewable Energy*, Vol. 78, p. 203–218, 2015.

30. Yang, D., C. Gu, Z. Dong, P. Jirutitijaroen, N. Chen and W. M. Walsh, "Solar irradiance forecasting using spatial-temporal covariance structures and time-forward kriging", *Renewable Energy*, Vol. 60, p. 235–245, 2013.

31. Vaz, A., B. Elsinga, W. van Sark and M. Brito, "An artificial neural network to assess the impact of neighbouring photovoltaic systems in power forecasting in Utrecht, the Netherlands", *Renewable Energy*, Vol. 85, p. 631–641, 2016.

32. Gutierrez-Corea, F.-V., M.-A. Manso-Callejo, M.-P. Moreno-Regidor and M.-T. Manrique-Sancho, "Forecasting short-term solar irradiance based on artificial neural networks and data from neighboring meteorological stations", *Solar Energy*, Vol. 134, p. 119–131, 2016.

33. Lazzaroni, M., S. Ferrari, V. Piuri, A. Salman, L. Cristaldi and M. Faifer, "Models for solar radiation prediction based on different measurement sites", *Measurement*, Vol. 63, p. 346–363, 2015.

34. Hyndman, R. J. and Y. Khandakar, "Automatic Time Series Forecasting: The forecast Package for R", *Journal of Statistical Software*, Vol. 27, No. 3, 2008.

35. Kiefer, J. and J. Wolfowitz, "Stochastic Estimation of the Maximum of a Regression Function", *The Annals of Mathematical Statistics*, Vol. 23, No. 3, pp. 462–466, 1952.

36. Kingma, D. P. and J. Ba, "Adam: A Method for Stochastic Optimization", *arXiv:1412.6980 [cs]*, 2017, `http://arxiv.org/abs/1412.6980`, arXiv: 1412.6980.

37. Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, Vol. 15, No. 56, pp. 1929–1958, 2014.

38. Leshno, M., V. Y. Lin, A. Pinkus and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function", *Neural Networks*, Vol. 6, No. 6, pp. 861–867, 1993.

39. Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu and X. Zheng, "TensorFlow: A System for Large-Scale Machine Learning", *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, USENIX Association, Savannah, GA, 2016.

40. Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito,

M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library", *Advances in Neural Information Processing Systems 32*, pp. 8026–8037, Curran Associates, Inc., 2019.

41. Rumelhart, D. E., G. E. Hinton and R. J. Williams, "Learning representations by back-propagating errors", *Nature*, Vol. 323, No. 6088, p. 533–536, 1986.

42. National Centers For Environmental Prediction, *NCEP GFS 0.25 Degree Global Forecast Grids Historical Archive*, 2015, `https://rda.ucar.edu/datasets/ds084.1/, last accessed on May 15, 2020`.

43. Holmgren, W. F., C. W. Hansen and M. A. Mikofski, "pvlib python: a python package for modeling solar energy systems", *Journal of Open Source Software*, Vol. 3, No. 29, p. 884, 2018.

44. Thieurmel, B. and A. Elmarhraoui, *suncalc: Compute Sun Position, Sunlight Phases, Moon Position and Lunar Phase*, 2019.

45. Nauss, T., H. Meyer, F. Detsch and T. Appelhans, *satellite: Handling and Manipulating Remote Sensing Data*, 2019.

46. Akiba, T., S. Sano, T. Yanase, T. Ohta and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework", *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 2623–2631, ACM, 2019.

47. Bergstra, J., R. Bardenet, Y. Bengio and B. Kégl, "Algorithms for Hyper-Parameter Optimization", *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, p. 2546–2554, Curran Associates Inc., Red Hook, NY, USA, 2011.

48. Bergstra, J., D. Yamins and D. Cox, "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures", *International Conference on Machine Learning*, p. 115–123, PMLR, 2013.

APPENDIX A: MODEL PARAMETERS

Table A.1. Parameters used by FC-ANN.

Parameter Group	**Parameter Name**	**Parameter Type**	**Sampling Type**	**Value(s)**
Model Config	hidden	hyperparameter	discrete uniform	[64, 32, 16, 4], [[128, 64, 32, 16], [256, 128, 64, 32]]
	scale_factor	constant		3

Table A.2. Parameters used by CNN.

Parameter Group	**Parameter Name**	**Parameter Type**	**Sampling Type**	**Value(s)**
Model Config	hidden	hyperparameter	discrete uniform	[[64, 32], [128, 64]]
	scale_factor	constant		3
	groups	constant		1
	kernel_size	hyperparameter	discrete uniform	[2,3]

Table A.3. Parameters used by ConvLSTM.

Parameter Group	**Parameter Name**	**Parameter Type**	**Sampling Type**	**Value(s)**
Model Config	num_layers	hyperparameter	discrete uniform	[2, 3]
	conv_per_cell	hyperparameter	discrete uniform	[1, 2]
	fc_hidden_nodes	hyperparameter	discrete uniform	[8, 16, 32]
	hidden	hyperparameter	discrete uniform	[[64, 32], [64, 32, 16], [128, 64], [128, 64, 32]]
	kernel_size	hyperparameter	discrete uniform	[2, 3]
Dataset Config	time_window	constant		6
	alignment	constant		"center"

Table A.4. Parameters used by FC-LSTM.

Parameter Group	**Parameter Name**	**Parameter Type**	**Sampling Type**	**Value(s)**
Model Config	num_layers	hyperparameter	discrete uniform	[1, 2]
	hidden	hyperparameter	discrete uniform	[64, 128, 256]
	bidirectional	constant		True
Dataset Config	time_window	constant		6
	alignment	constant		"center"

Table A.5. Parameters used by PLC-LSTM.

Parameter Group	**Parameter Name**	**Parameter Type**	**Sampling Type**	**Value(s)**
Model Config	unfold_size	constant		[3,2]
	unfold_step	constant		[2,1]
	num_layers	hyperparameter		[1, 2]
	hidden	hyperparameter		[32, 16]
	bidirectional	constant		True
Dataset Config	time_window	constant		6
	alignment	constant		”center”

APPENDIX B: FEATURE COMBINATIONS

Table B.1. Feature combinations used in FC-LSTM trials.

Feature Combination	add_TCDC_entire	add_altitude	add_aux	add_azimuth	add_ghi	add_temp	add_ws10
1	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE
2	FALSE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE
3	FALSE	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE
5	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE
6	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE
7	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE
8	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
9	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE
10	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE	TRUE
11	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE	TRUE
12	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
13	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE
14	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
15	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
16	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE	TRUE
17	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE
18	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE
19	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
20	FALSE	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE
21	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE
22	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
23	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE
24	FALSE	FALSE	TRUE	TRUE	TRUE	FALSE	TRUE
25	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE	FALSE
26	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
27	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE
28	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
29	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE
30	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE

Table B.2. Feature combinations used in FC-ANN trials.

Feature Combination	**add_TCDC_entire**	**add_altitude**	**add_aux**	**add_azimuth**	**add_ghi**	**add_temp**	**add_ws10**
1	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
3	FALSE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE
4	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE
5	FALSE	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE
6	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE
7	FALSE	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE
8	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
9	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
10	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
11	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE
12	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
13	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
14	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	FALSE
15	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE
16	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE
17	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE
18	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE
19	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
20	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE
21	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE
22	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE
23	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE
24	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE
25	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
26	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
27	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE
28	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
29	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
30	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE
31	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
32	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
33	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE
34	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
35	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
36	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE
37	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE

Table B.3. Feature combinations used in CNN trials.

Feature Combination	add_TCDC_entire	add_altitude	add_aux	add_azimuth	add_ghi	add_temp	add_ws10
1	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
2	FALSE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE
3	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
4	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE
5	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE	FALSE
6	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
7	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
8	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE	TRUE
9	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE
10	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE
11	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
12	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
13	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
14	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
15	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE
16	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
17	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE
18	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	FALSE
19	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
20	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
21	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE
22	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE	TRUE
23	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE
24	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
25	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
26	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
27	FALSE	FALSE	TRUE	TRUE	TRUE	FALSE	FALSE
28	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
29	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
30	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE
31	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE
32	FALSE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE
33	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	TRUE
34	FALSE	TRUE	TRUE	TRUE	FALSE	FALSE	TRUE
35	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE
36	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE
37	FALSE	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE
38	TRUE	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE

Table B.4. Feature combinations used in ConvLSTM trials.

Feature Combination	**add_TCDC_entire**	**add_altitude**	**add_aux**	**add_azimuth**	**add_ghi**	**add_temp**	**add_ws10**
1	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
2	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	FALSE
3	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
4	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	FALSE
5	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE
6	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE
7	FALSE	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE
8	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE
9	TRUE	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE
10	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE
11	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE
12	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE
13	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
14	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE
15	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE
16	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE
17	TRUE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE
18	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
20	TRUE	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE
21	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE
22	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE
23	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE
24	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE
25	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
26	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
27	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE
28	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE
29	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
30	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE

Table B.5. Feature combinations used in PLC-LSTM trials.

Feature Combination	**add_TCDC_entire**	**add_altitude**	**add_aux**	**add_azimuth**	**add_ghi**	**add_temp**	**add_ws10**
1	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE
2	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE
3	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE
4	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
5	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	TRUE
6	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE
7	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE
8	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
9	FALSE	TRUE	TRUE	TRUE	FALSE	FALSE	TRUE
10	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
11	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE
12	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE	TRUE
13	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE
14	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
15	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	TRUE
16	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
17	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
18	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE	TRUE
19	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE
20	FALSE	FALSE	TRUE	TRUE	TRUE	FALSE	FALSE
21	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE
22	TRUE	TRUE	TRUE	TRUE	FALSE	FALSE	TRUE
23	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
24	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
25	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
26	FALSE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE
27	TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE
28	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE
29	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
30	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE
31	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
32	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE
33	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE	FALSE
34	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE

APPENDIX C: MONTHLY TEST RESULTS

Table C.1. Monthly test results of all models, together with the benchmarks.

Month	Model	WMAPE	PBias	$\text{DWMAPE}_{\text{med}}$	$\text{DWMAPE}_{\text{sd}}$	$\text{DPBias}_{\text{med}}$	$\text{DPBias}_{\text{sd}}$
1	INM	0.2659	-0.0918	0.2253	0.1870	0.0754	0.3360
	P24	0.5020	-0.0008	0.3463	0.7912	0.0016	1.0019
	P48	0.5662	-0.0129	0.4808	0.9374	-0.1910	1.1838
	$\text{SARIMAX}(3,0,0)(2,1,0)_{24}$	0.4587	-0.0317	0.3893	0.8806	-0.0612	1.0600
	ConvLSTM	0.2775	-0.0943	0.2733	0.1419	-0.0126	0.3041
	CNN	0.2710	-0.0006	0.2622	0.2664	0.0333	0.4000
	Fully Connected	0.2815	0.0142	0.2874	0.3245	0.0289	0.4489
	FC-LSTM	0.2787	-0.0449	0.2846	0.1817	-0.0213	0.3394
	PLC-LSTM	0.2704	-0.0126	0.2650	0.1550	0.0456	0.2856
2	INM	0.2777	0.0469	0.3100	0.2272	0.0748	0.3499
	P24	0.4081	-0.0071	0.3661	0.5996	-0.0013	0.7735
	P48	0.6011	-0.0347	0.5797	0.7560	-0.1490	1.0428
	$\text{SARIMAX}(3,0,0)(2,1,0)_{24}$	0.5717	-0.0254	0.5477	0.7329	-0.0015	0.9885
	ConvLSTM	0.2503	-0.0415	0.2463	0.1183	-0.0549	0.2281
	CNN	0.2602	-0.0415	0.2768	0.1519	0.0044	0.2761
	Fully Connected	0.2486	-0.0105	0.2229	0.1751	0.0045	0.2899
	FC-LSTM	0.2414	0.0539	0.2558	0.2410	0.0610	0.3030
	PLC-LSTM	0.2734	0.0713	0.3307	0.1933	0.1129	0.2862
3	INM	0.2927	0.1948	0.2512	0.4609	0.2090	0.5085
	P24	0.3465	-0.0103	0.3453	0.7730	0.0261	0.8954
	P48	0.4872	-0.0026	0.4504	0.9185	-0.0736	1.1083
	$\text{SARIMAX}(3,0,0)(2,1,0)_{24}$	0.3950	0.0011	0.3434	0.6214	0.0756	0.7777
	ConvLSTM	0.2050	0.0056	0.2118	0.1305	-0.0058	0.1941
	CNN	0.2272	-0.0175	0.1901	0.2603	-0.0561	0.3663
	Fully Connected	0.2416	0.0402	0.1888	0.3619	0.0500	0.4520
	FC-LSTM	0.2064	0.0769	0.1583	0.2989	0.0570	0.3586
	PLC-LSTM	0.2007	0.0661	0.1775	0.2517	0.0417	0.3168
4	INM	0.2026	0.1072	0.1629	0.2222	0.1081	0.2590
	P24	0.2634	-0.0036	0.2648	0.2853	0.0450	0.3843
	P48	0.3228	-0.0235	0.2542	0.3728	-0.0038	0.5202
	$\text{SARIMAX}(3,0,0)(2,1,0)_{24}$	0.3189	-0.0304	0.2587	0.3360	-0.0576	0.4868
	ConvLSTM	0.1613	-0.0369	0.1565	0.0737	-0.0399	0.1284
	CNN	0.1712	-0.1061	0.1479	0.1168	-0.1020	0.1194
	Fully Connected	0.1697	-0.0342	0.1621	0.1405	-0.0471	0.1553
	FC-LSTM	0.1221	-0.0253	0.1184	0.0953	-0.0442	0.1301
	PLC-LSTM	0.1294	-0.0534	0.1267	0.0909	-0.0669	0.1278