

Master's Thesis : Comparison of probabilistic forecasting deep learning models in the context of renewable energy production

Auteur : Stassen, Théo

Promoteur(s) : Ernst, Damien

Faculté : Faculté des Sciences appliquées

Diplôme : Master en ingénieur civil en informatique, à finalité spécialisée en "intelligent systems"

Année académique : 2019-2020

URI/URL : <http://hdl.handle.net/2268.2/10466>

Avertissement à l'attention des usagers :

Tous les documents placés en accès ouvert sur le site le site MatheO sont protégés par le droit d'auteur. Conformément aux principes énoncés par la "Budapest Open Access Initiative"(BOAI, 2002), l'utilisateur du site peut lire, télécharger, copier, transmettre, imprimer, chercher ou faire un lien vers le texte intégral de ces documents, les disséquer pour les indexer, s'en servir de données pour un logiciel, ou s'en servir à toute autre fin légale (ou prévue par la réglementation relative au droit d'auteur). Toute utilisation du document à des fins commerciales est strictement interdite.

Par ailleurs, l'utilisateur s'engage à respecter les droits moraux de l'auteur, principalement le droit à l'intégrité de l'oeuvre et le droit de paternité et ce dans toute utilisation que l'utilisateur entreprend. Ainsi, à titre d'exemple, lorsqu'il reproduira un document par extrait ou dans son intégralité, l'utilisateur citera de manière complète les sources telles que mentionnées ci-dessus. Toute utilisation non explicitement autorisée ci-avant (telle que par exemple, la modification du document ou son résumé) nécessite l'autorisation préalable et expresse des auteurs ou de leurs ayants droit.

MASTER THESIS

Graduation Studies conducted for obtaining the Master's degree in
Computer Engineering

Comparison of probabilistic forecasting deep learning models in the context of renewable energy production

Stassen Théo (s150804)

Promoter : D. Ernst

2019-2020

Chapter 1

Introduction and problem statement

1.1 Introduction

Blacklight Analytics is a enterprise that develops solutions regarding the management of local electrical distribution networks. In the recent years, distribution systems need to integrate more and more renewable generation in their networks. Since networks cannot be quickly upgraded and at low cost, new generators are connected to the network under non-firm access contract. The assets in the networks have been designed to transmit power in one direction, from the global network, on which the local network is connected, to the local network. This configuration implies situations where the temporary high production in the local network creates congestion problems in the assets, when energy generated must be injected into the global network. There is a need for a practical method able to limits dynamically the generators production such that the system can be considered safe, i.e has a very low probability that the electrical power flowing through one asset is higher than the maximum tolerated power is this asset.

Blacklight Analytics develops a solution to this problem. The method casts the problem into a stochastic decision process. This process is divided in three phases : (i) generation of a network model, (ii) forecasting of the power produced or consumed, (iii) computation of the generator limits. The phase (ii) , for each network generator, takes as input generator production historical and output future production probability distribution . This output will be used to define generator limits. If the forecasted probability distribution indicates that there is a non-negligible risk that the production of the generator outnumbers its maximum acceptable value, above which it encounters risks, it means that the generator capacity must be voluntarily restrained immediately. In the current implementation, the forecasting method uses Gaussian process regression to forecast individually each source and combine information using covariance estimation. This master thesis subject addresses the question of what is the best forecasting method to implement in this described context, and on the basis of which elements of comparison. The growing

scientific field of Deep Learning has a great potential to be exploited to achieve this goal.

Therefore, the concrete goal of this Master Thesis is the comparison of different probabilistic forecasting models, mostly from the deep learning scientific field, in the context of the prediction of renewable energy production. The implementation code can be accessed at <https://github.com/Naaapp/MasterThesis>.

1.2 Document Organisation

We introduce the main goal that we want to achieve in this master thesis, i.e the comparison of different probabilistic forecasting models in the context of the prediction of renewable energy production, presents the related works and state mathematically the problem (1). After defining the problem, we describe the toolkit that will be used to tackle it, *GluonTS*, with its historical, functioning, different main components, as well as the different forecasting models implemented within (2). Then, we perform a comparison, using a classical metric as element of comparison, of all implemented models, after discussing the values of their hyperparameters (3). After this comparison, we discuss the relevance of using a classical metric to evaluate the quality of the forecast, knowing our previously defined goal, and from this discussion we introduce a new metric and the corresponding loss function (4). Come next the comparison of all implemented models using this newly defined metric (5) which allow us to conclude about the best forecasting model in the defined context.

1.3 Related Works

Time series forecasting is a well-defined and extensively explored scientific field. It is considered as part of the probability scientific field for many years, from very simple or naive forecasting models (*Naive*, *Seasonal Naive*, *Moving Average*, etc) [1] to more complex models as the *Autoregressive integrated moving average (ARIMA)* [2] or *Exponential smoothing*, introduced by statistician Robert Goodell Brown in 1956 [3]. *Exponential smoothing* and *ARIMA* models are the two most widely used approaches to time series forecasting, and provide complementary approaches to the problem. While *exponential smoothing* models are based on a description of the trend and seasonality in the data, *ARIMA* models aim to describe the autocorrelations in the data.

Probabilistic time series forecasting stems from single-valued time series forecasting. For instance, *ETS* forecasting is an *exponential smoothing* method that can generate single values but also prediction intervals [4].

In recent years, advances in deep learning has led to interesting results in probabilistic forecasting field. Recent publications ([5], [6]) and forecasting competitions results ([7], [8]) have shown the relevance of using deep learning based models to obtain better results than, for example, ETS or ARIMA methods.

Regarding the implementation of these techniques, deep learning frameworks, such as *Tensorflow* [9], *MXNet* [10] and *Pytorch* [11] are popular solutions. Considering the time series modeling, a certain number of commercial and open-source solutions exist but does not provides toolkits focused on modern deep learning. For example the *R-forecast* package [12] provide a plethora of models and tools for classical forecasting methods and contains neural forecasting models, however these pre-date modern deep learning methods and the toolkit only contain stand-alone implementations of simple local models. There is a lack of state-of-the-art architectures.

The lack of specific probabilistic time-series forecasting toolkit has been recently filled with *GluonTS* (<https://gluon-ts.mxnet.io>) [13], a deep learning library that bundles components, models and tools for time series applications such as single-valued and probabilistic forecasting or anomaly detection. It relies on the deep-learning framework *MXNet* [10], developed by *Amazon Web Services*, and is implemented in programming language *Python*. As it provides all the services needed to produce experiments and comparisons of deep learning probabilistic forecasting models, it is the main implementation tool of this master thesis. All models that will be compared in this work are *GluonTS* pre-implemented models.

Deep Learning probabilistic forecasting of renewable energy production is a subject that has already been covered by various scientific teams, with different contexts, goals and techniques. Various results has been compiled and analysed in different scientific review papers. Probabilistic forecasting in the context of wind turbines generation has been reviewed in [14]. Deep learning techniques for renewable energy forecasting has been reviewed in [15]. These reviews exposes notably a great number of different models and techniques.

Review [15] is more a presentation of state-of-the art techniques than a comparison, as it doesn't presents quantitative results. Review [14] presents results extracted from the scientific papers presenting the different models. These results are obtains with completely different datasets and other elements of the context are unknown.

As it is performed using the same toolkit, with the same dataset and context, this works presents a more reliable comparison and will explore the possibility of using other metrics than the ones used in these referred reviews, knowing the goal that has been fixed, which will possibly differs from the goal pursued in these reviews. As this work is focused on the models that are pre-implemented in *GluonTS*, some state-of-the-art models that are compared in this reviews are not compared in this work. These models are generally not open-source or accessible anyway.

1.4 Problem Statement

Before entering into solution description, we need to define mathematically the problem of time series deep-learning forecasting, whereabouts *GluonTS* has been conceived.

As different types of models has fundamental differences in the way they express

the problem, the mathematical formulation differs. Two mathematical formulation are presented, corresponding to two types of models found among the implemented models. The first mathematical formulation corresponds to the feed-forward models. The second mathematical formulation corresponds to the RNN models.

Models dataset is composed of certain number of time series, in training dataset and in testing dataset. In general the training and testing dataset is composed of the same time series (this affirmation is discussed in section 2.3). This situation is the one considered in this section.

Let the following variables :

- I the number of time series considered
- T the number of time steps in time series
- $x_{i,t}$ a scalar variable representing the value of the time series i at time step t .
- a, b, c, d, e specific values of t , with $e = T$
- $t = 0 < t = a < t = b < t = c < t = d < t = e$
- $predict_length = b - a$, fixed hyper parameter of the problem. Indicates the number of time steps about which the models must make a prediction simultaneously.
- $context_length = c - b$, fixed hyper parameter of the problem. Indicates the number of time steps that the model consider as input simultaneously.

We also define $L : \mathbb{R} \times (\mathbb{R} \rightarrow \mathbb{R}) \rightarrow \mathbb{R}$ the loss function taking a value and a probability distribution as input and giving a loss value as output. As example of function L , we can mention the negative log-likelihood :

$$L(y, \phi(x)) = -\ln(\phi(x = y)) \quad (1.1)$$

The details of loss implementation are discussed in section 2.4.

Each time series can be conceptually decomposed as two pieces. The first piece, $[x_{i,0}, \dots, x_{i,d-1}]$ correspond to the part of the time series whereupon we want to train. The second piece $[x_{i,d}, \dots, x_{i,e-1}]$ correspond to the part of the time series whereupon we want to test. The training set is composed only of the first pieces of each time series when the training set is composed of the two pieces (i.e. the totality of the time series).

At the end of the training, the model neural network has been trained on the whole training data, and can be tested on the testing data.

1.4.1 First mathematical formulation : Feed Forward models

This formulation corresponds to the way the models that can be described as Feed Forward (see *SimpleFeedForward* model in section 2.7.1) compute a solution of the problem.

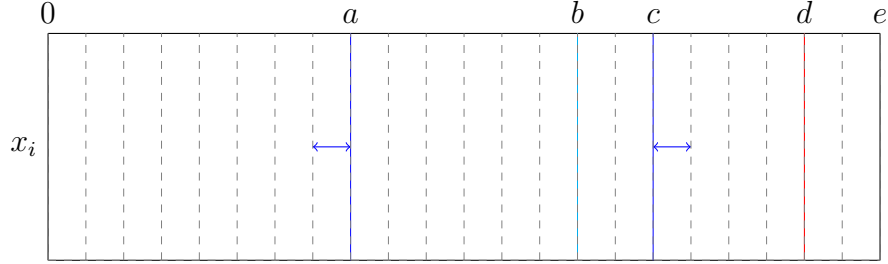


Figure 1.1: Feed forward models forecast : General case illustration

Considering an interval $[x_{i,a}, \dots, x_{i,c-1}]$ in the time series i , our goal is to estimate the distribution of future window $[x_{i,b}, \dots, x_{i,c-1}]$ given its history $[x_{i,a}, \dots, x_{i,b-1}]$. The estimated distribution is defined as :

$$p_{\theta}(x_{i,b}, \dots, x_{i,c-1} | x_{i,a}, \dots, x_{i,b-1}) \quad (1.2)$$

Where θ denotes the models parameters. This distribution can be factorised as :

$$p_{\theta}(x_{i,b}, \dots, x_{i,c-1} | x_{i,a}, \dots, x_{i,b-1}) = \prod_{t=b}^{c-1} p_{\theta}(x_{i,t} | x_{i,a}, \dots, x_{i,b-1}) \quad (1.3)$$

We define NN_{θ} , a neural network parametrized by θ . Its hypothesis space is defined as $\{h : \mathbb{R}^{b-a} \rightarrow \mathbb{R}^{k(c-b)}, h \in H\}$, with k the number of parameters needed to define the output distribution. We consider the case of a gaussian distribution, with $k = 2$. The evaluation of NN_{θ} can be described as :

$$[\mu_{\theta i,b}, \dots, \mu_{\theta i,c-1}, \sigma_{\theta i,b}, \dots, \sigma_{\theta i,c-1}] = NN_{\theta}(x_{i,a}, \dots, x_{i,b-1}) \quad (1.4)$$

The estimated distribution of a variable $x_{i,t}$ of the future window ($t \in [b, c-1]$) is expressed as, with $\phi_{\mu, \sigma} : \mathbb{R} \rightarrow \mathbb{R}$ a gaussian function parametrized by parameters μ and σ :

$$p_{\theta}(x_{i,t} | x_{i,a}, \dots, x_{i,b-1}) = \phi_{\mu_{\theta i,t}, \sigma_{\theta i,t}}(x_{i,t}) \quad (1.5)$$

The estimated distribution of the future window as defined in equation 1.2 is expressed as :

$$p_{\theta}(x_{i,b}, \dots, x_{i,c-1} | x_{i,a}, \dots, x_{i,b-1}) = \prod_{t=b}^{c-1} \phi_{\mu_{\theta_i,t}, \sigma_{\theta_i,t}}(x_{i,t}) \quad (1.6)$$

This estimation is performed for different intervals $[x_{i,a}, \dots, x_{i,c-1}]$. We consider disjoint adjacent future windows, from the first possible future window ($b = \text{context_length}$), see figure 1.2, to the last possible future window ($b = d - \text{predict_length}$), see figure 1.4, and for all time series $i \in [0, I]$.

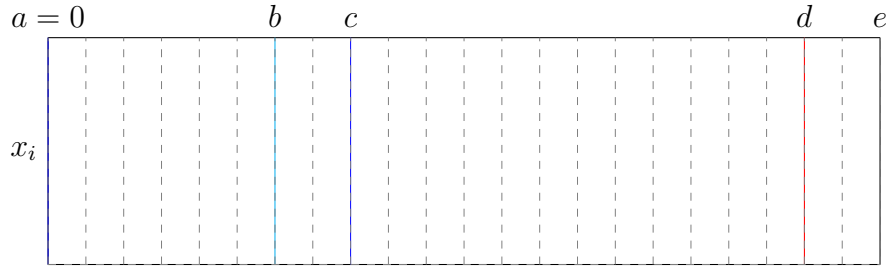


Figure 1.2: Feed forward models forecast : First possible interval illustration

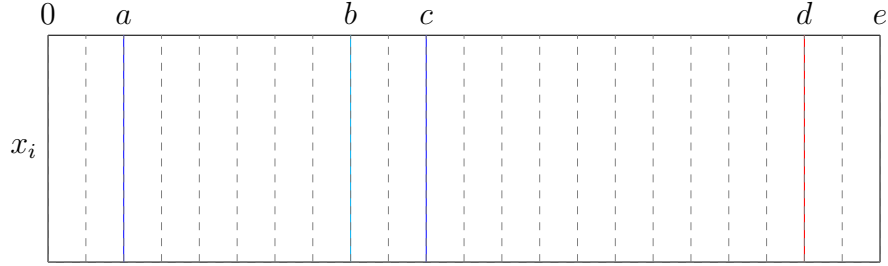


Figure 1.3: Feed forward models forecast : Second possible interval illustration

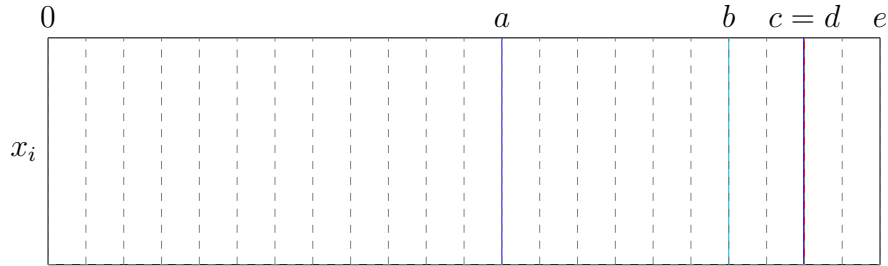


Figure 1.4: Feed forward models forecast : Last possible interval illustration

Finally, the global loss of the estimation is defined as the sum, for each time series and each time step, of the loss function L with as argument the real value and the estimated distribution :

$$loss(\theta) = \sum_{i=1}^I \sum_{t=b-a}^{d-1} L(x_{i,t}, \phi_{\mu_{\theta i,t}, \sigma_{\theta i,t}}(x_{i,t})) \quad (1.7)$$

1.4.2 Second mathematical formulation : RNN models

This formulation corresponds to the way the models that can be described as RNN (see *CanonicalRNN* model in section 2.7.2) express the problem.

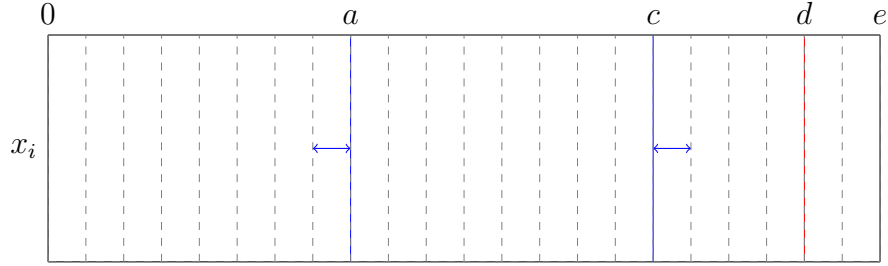


Figure 1.5: RNN models forecast: General case illustration

Considering an interval $[x_{i,a}, \dots, x_{i,c-1}]$ in the time series i , our goal is to estimate, $\forall t \in [a+1, c-1]$, the distribution of $x_{i,t}$ given $x_{i,t-1}$ and its hidden states $h_{i,t}$. The estimated distribution is defined as :

$$p_{\theta}(x_{i,t} | x_{i,t-1}, h_{i,t}) \quad (1.8)$$

Where θ denotes the models parameters.

We define f_{θ} , a function parametrized by θ . Its hypothesis space is defined as $\{h_f : \mathbb{R} \rightarrow \mathbb{R}, h_f \in H_f\}$. Its evaluation can be described as, $\forall t \in [a+1, c-1]$:

$$h_{i,t} = f_{\theta}(x_{i,t-1}, h_{i,t-1}) \quad (1.9)$$

We define g_{θ} , a function parametrized by θ . Its hypothesis space is defined as $\{h_g : \mathbb{R} \rightarrow \mathbb{R}^k, h_g \in H_g\}$. With k the number of parameters needed to defines the output distribution. We consider the case of a gaussian distribution, with $k = 2$. Its evaluation can be described as, $\forall t \in [a+1, c-1]$::

$$[\mu_{\theta i,t}, \sigma_{\theta i,t}] = g_{\theta}(h_{i,t}) \quad (1.10)$$

The estimated distribution as defined in equation 1.8 is expressed as, $\forall t \in [a + 1, c - 1]$, with $\phi_{\mu, \sigma} : \mathbb{R} \rightarrow \mathbb{R}$ a gaussian function parametrized by parameters μ and σ :

$$p_{\theta}(x_{i,t}|x_{i,t-1}, h_{i,t}) = \phi_{\mu_{\theta_{i,t}}, \sigma_{\theta_{i,t}}}(x_{i,t}) \quad (1.11)$$

This estimation is performed for different intervals $[x_{i,a}, \dots, x_{i,c-1}]$. We consider disjoint adjacent interval, from the first possible future interval ($c = context_length + predict_length$), to last possible future window ($c = d$), and for all time series $i \in [0, I]$.

Finally, the global loss of the estimation is defined as the sum, for each time series and each time step considered, of the loss function L with as argument the real value and the estimated distribution :

$$loss(\theta) = \sum_{i=1}^I \sum_{t=b-a}^{d-1} L(x_{i,t}, \phi_{\mu_{\theta_{i,t}}, \sigma_{\theta_{i,t}}}(x_{i,t})) \quad (1.12)$$

Chapter 2

GluonTS

2.1 *GluonTS*: from origins to nowadays

To tackle the problem that has been formalised mathematically in the last chapter, the main implementation tool that have been chosen is the toolkit *GluonTS*. To begin the presentation of this toolkit, a summary of its history might be useful.

GluonTS has been developed by a *Amazon Web Service* team to fill the gap of time series modeling toolkit [13]. From its origins, its goal is to provide a deep learning library that bundles components, models and tools for time series applications such as forecasting or anomaly detection. It simplifies all aspects of scientific experiments with time series models, which justify its use in this master thesis.

At the time of writing this master thesis, *GluonTS* is a very recent toolkit, which is coherent with the fact that the deep learning forecasting field is recent itself. Its first release version (*v0.1.0*) is dated from March 3rd 2019. The implementation of the code has been performed on the version *v0.4.2* released on November 26th 2019 and has been updated for version *v0.5.0*, released on May 12nd 2020. Models *Wavenet* (see 2.7.10) and *NBEATS* (see 2.7.11) have notably been added in this update. These versions are still considered as beta versions. This implied several bugs and implementation issues. In particular some models (for example an implementation of the *DeepState* model described in paper [16]) failed to run correctly, and the run of “custom” models (implemented using the provided model template functionality) is incompatible, due to a running error, with the “hybridization” functionality, an CPU-GPU optimization method which provides important training time reducing. In addition, some functionalities that would be useful are not currently implemented, as alternative loss functions for pre-implemented models (see 4.3 section).

2.2 *GluonTS* Functioning

The *GluonTS* functioning could be summarised as the interaction between different components that are cited here and presented in details in following sections.

The *GluonTS* toolkit defines a dataset structure, containing in particular the time series information (see section 2.3). This dataset is given as input to a *GluonTS* model. This model is pre-implemented or implemented using the provided “model template” functionality, which allows to create any kind of models following a defined structure. The selected model uses the dataset information for its training and testing. When the model is evaluated, it gives in all cases as output a distribution for each of the testing time steps, for each time series (see 1.4). The type of output distribution is an hyperparameter of the problem and is discussed in section 2.6. Concretely, the output information is not the distribution parameters themselves but a set of randomly drawn sample of the predicted distribution. The model, if belongs to deep learning models category, must be trained before evaluation, using the loss function (see section 2.4). The resulting output distributions could be evaluated using the appropriate metrics (see section 2.5).

2.3 Datasets

GluonTS interface imposes a certain structure of datasets, objects containing data used for training and testing models. This section describes what is this structure, what are the original data information at our disposition and how these data are traduced into datasets.

Datasets are composed of some mandatory and some optional components. ‘*dataset.train*’ is an iterable collection of data entries used for training. Each entry corresponds to one time series. ‘*dataset.test*’ is an iterable collection of data entries used for inference. ‘*dataset.metadata*’ contains metadata of the dataset such as the frequency of the time series, the context length and the prediction length (see below).

GluonTS is optimized to handle multiple time series in parallel. The training time will be significantly increased if the dataset is composed of only one time series of very long length compared to a dataset composed of multiples time series of smaller length.

As mentioned in section 1.4, in general the training and testing sets are composed of the same time series. The definition of the task whereabouts this master thesis is dedicated, the forecasting of wind turbines production, admits situations not compatible with this general affirmation. In fact, the goal is to forecast the production of an energy generator using a model which possibly could not be trained on the current generator production history, but on others generator production history. The comparison must be performed on different configurations of training/testing data, including where the training data and testing data comes from the same generator(s) history and where it is not the case. This allow us to test how the forecast quality is influenced by the variation of configuration of input data.

Original data provided by “Blacklight Analytics” is composed of different time series :

- A continuous 6 months history of a wind turbine production , in kWh
(*6months-minutes.csv*)
- An history of a wind turbine production, in negative MWh
(*mesure_p_gestamp.csv*)
- A 2 days history of two wind turbines production, in negative MWh
(*2eol_measurements.csv*)

The constant time between two time steps in these time series is 1 minute. Using *GluonTS* terminology, the time series are at a “frequency” of 1 minute. All the time series are converted to positive MWh. Nowadays the different studied configuration, because of the *GluonTS* behaviour optimized to handle multiple time series and behaviour, these original times series are manually splitted before putting them in dataset structure. Each original time series are manually splitted in certain number of time series of same size. This size is considered as a hyper parameter of the problem.

The configurations considered are the following :

- All the disposable times series are used as training and testing data. Referred as “Config A” in the following sections.
- *2eol_measurements.csv* time series used as testing data and the other time series as training data. Referred as “Config B” in the following sections.

We will see in chapter 3 that the configuration A/B does not influences significantly the choice of the optimal parameters, as the experimentation that have been performs tends to show. Knowing that testing different configurations systematically is in the majority of the time redundant and the training time needed for the experimentation is multiplied, the different configurations forecasting results will not be systematically shown.

GluonTS allow the use of features, i.e. information used as input to the models additionally to time series information. Between the tested models, only *DeepAR* (2.7.3) is compatible with this functionality. In our case, it could be interesting to differentiate the different wind turbines information. Time series sources can be described using static categorical features. For example, in the configuration A, splitted time series coming from time series *6months-minutes.csv* will have a categorical value of 0, time series from *mesure_p_gestamp.csv* the value of 1, times series from *2eol_measurements.csv* values of 2 and 3. This functionality could be useful whether the different sources have fundamental differences in behaviour. Nevertheless, the concrete use of this functionality in the code fails for an unknown reason. As it is an optional functionality that could be use on only one model, we decide to give up this functionality.

As defined in section 1.4, *predict_length* and *context_length* are hyperparameters of the problem. *BlackLight Analytics* is interested in a prediction interval of 10 minutes. The hyperparameter is set definitely to 10, as the frequency is 1 minute. *context_length*, indicating the number of time steps considered as input for the model network, is considered as an hyperparameter to tune (3.3.1 and 5.2.1).

2.4 Loss

Loss functions are a key component of the deep learning forecasting process, as in any deep learning applications, as they evaluate the quality of the prediction during the training process, in order to update the model network in consequence. We have seen how the loss is expressed in the problem statement in section 1.4. In this section we discuss about the loss function L , that appears in this loss expression, that *GluonTS* will use.

GluonTS implements for its predefined models only one type of loss function. This loss function is, for the majority of the models, the inverse of the log-density of the output distribution, as defined in the equation 2.1. In some models, the fundamental difference in terms of architecture justify a different loss. For example the *MQRNN* and *MQCNN* models use a combination of different quantile losses as loss function (see 2.7.7).

If $\phi(x)$ is a probability distribution and y an observed value, the loss function L is :

$$L_{\text{default}}(y, \phi(x)) = -\ln(\phi(x = y)) \quad (2.1)$$

The use of this loss expresses efficiently that the predicted distribution must corresponds to the real distribution (distribution that is unknown, but from which observed values can be considered as samples). In the chapter 3 of this master thesis, the model comparison will be done using this default loss, considering that the objective pursued is the minimization of the difference between predicted distribution and the real distribution.

The chapter 4 of this master thesis will discuss the relevance of this affirmation and the use of this default loss.

2.5 Metrics

Metric functions are a key component of the deep learning forecasting process as much as loss functions. When the loss evaluates the quality of the forecast during training, the metric evaluate the forecast during testing. In this section, we discuss about the optimal choice of metrics that we will use, taking account the choice of loss.

Concerning the evaluation of the results, *GluonTS* provides a module using the trained model and testing data to provide quantitative results, as metrics (“Aggregate”, for all time series, or “Item”, for each series separately). Metrics proposed by *GluonTS* includes *MSE*, *MSIS*, *RMSE*, *MASE*, and other classical measures.

All metric function takes as argument N randomly drawn sample values of the predicted probability distribution $([x_1, ..., x_N])$ and one observed value y . For exam-

ple the *MSE* metric function is defined as :

$$MSE(y, [x_1, \dots, x_N]) = \frac{1}{N} \sum_{i=0}^N (y - x_i)^2 \quad (2.2)$$

“Item” metric value is the mean of metric function value for each time step of the testing interval. In fact, for models that falls under the feed-forward category, the testing consists in evaluating the model with, as prediction interval, the testing interval (see 1.4) and compute the metric of each time step of this interval. For RNN models category, the testing consists in evaluating the first time step of the testing interval, obtains a distribution as output, sample from it a value and use this value as argument for the following time step evaluation, and continue until the end of the testing interval. “Aggregate” metric value is the mean of “Item” metrics values for each time series.

In chapter 3, as we use the default loss, metric function must correspond to this loss, as it has been defined in section 2.4.

The problem is that the metric function cannot be equivalent to loss function, as these are functions with different inputs, the loss function taking as input the predicted distribution instead of randomly drawn point sample. Nowadays, in fine, the use of metrics like *MSE*, *MSIS*, *RMSE*, *MASE* to evaluate the quality of the model express the same goal as the use of the defined loss function, i.e. the minimization of the difference between predicted and real distribution. A high value of one of these metrics in mean is proportional to a high value in mean of the loss in mean. The metric chosen for the first model comparison in chapter 3 is *MASE*, the “Mean Squared Averaged Error”, that is available for all model (in contrast to *MSE*, not available for *MQCNN* and *MQRNN* models). It is defined as the *MSE* divided by the absolute error of a naive one step model prediction.

2.6 Distribution

In previous section, in particular in the problem statement (1.4), we formulate the problem as probabilistic, which means that the prediction output is not a single value but a distribution. We still have to define what type of distribution is given at output.

GluonTS proposes different types of output distribution. In most of the models the output of the neural network is technically a vector of values which is transformed to a vector of distribution parameters of the chosen distribution. Finally is given as output a defined number of sampled point of the distribution that has been parametrized using the parameters.

In current implementation, three different output distribution could be used, *Gaussian*, *Laplace* and *PiecewiseLinear*, for the majority of models (some models, *MQCNN*, *MQRNN*, *NBEATS* and *Wavenet* does not outputs samples of a distribution function, but quantiles of the distribution itself, see 2.7).

Student and *Beta* has been rejected because the quantiles of the distribution are not obtainable, which is not acceptable as we are interested in the quantiles values in the chapter 4. *Uniform* has been rejected because of loss concerns in execution. Other options are available, as multiple kernel gaussian. The study [14] assert that the two most common choices of output distribution in the context of the wind turbines probabilistic forecasting are *Gaussian* and *Beta* distribution. This affirmation will be verified (concerning *Gaussian*) in sections 3.3.5 and 5.2.5.

The *GluonTS* models are mandatory to output an object containing samples of the distribution, not the distribution itself (parameters values). Parameters values could be saved in the case of custom models in order to obtain precises information about the distribution that has been predicted.

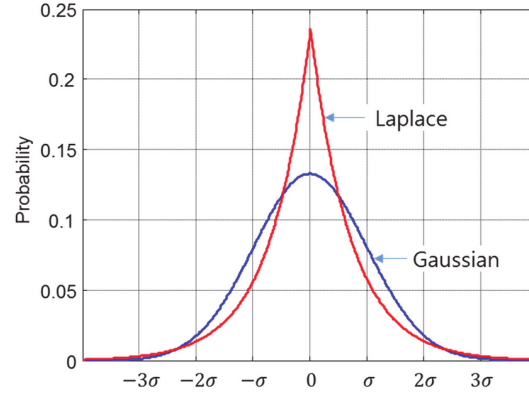


Figure 2.1: Comparison between *Gaussian* distribution and *Laplace* distribution (https://www.researchgate.net/figure/Gaussian-distribution-and-Laplace-distribution_fig7_321825093)

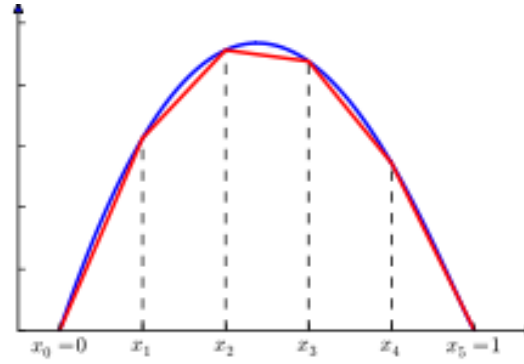


Figure 2.2: Comparison between *Gaussian* distribution and *PiecewiseLinear* distribution (https://en.wikipedia.org/wiki/Piecewise_linear_function)

2.7 Models presentation

GluonTS, as a toolkit focused on providing tools to implement, evaluate and compare time series forecasting models, provides a wide range of different forecasting models.

Before evaluate it, the models must be theoretically introduced.

All the pre-implemented deep learning models that have been tested are presented here. The models own hyperparameters are mentioned in corresponding section. Own hyperparameter values comparison is performed in section 3.4.

2.7.1 FeedForward

The first mathematical formulation of the problem (in section 1.4.1) corresponds to the way models that could be described as “Feed Forward models” expresses the problem. This implemented model, named *FeedForward*, neural network consists in an MLP (Multi-layer Perceptron).

Its number of hidden layers and number of hidden nodes per layer are considered as tunable hyper parameters whose values will be discussed in corresponding comparison sections (3.4.1 and 5.3.1).

2.7.2 Recurrent Neural Network

The second mathematical formulation of the problem (in section 1.4.2) corresponds to the way models that could be described as “Recurrent Neural Network models” expresses the problem. This implemented model, named *CanonicalRNN*, neural network consists in an RNN (Recurrent Neural Network). The denomination “Canonical” is employed to avoid confusion with other RNN-based models, as *DeepAr* and *MQRNN*.

Its number of hidden layers and number of hidden nodes per layer is considered as tunable hyperparameters whose the values will be discussed in corresponding comparison sections (3.4.2 and 5.3.2).

2.7.3 DeepAR

The model *DeepAr* is the implementation of a “DeepAR” model close the one described in paper “*DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks*” [17]. This model is described as “*A forecasting method based on autoregressive recurrent network (..) that tailors a similar LSTM-based recurrent neural network architecture to the probabilistic forecasting problem*”.

As the model neural network consists in a autoregressive neural network similar to RNN, tunable hyperparameters are the same than for *CanonicalRNN model*, i.e the number of hidden layers and the number of hidden nodes per layer. Their values are discussed in corresponding comparisons sections (3.4.3 and 5.3.3). This model can use static features information.

2.7.4 Deep Factors

The model *DeepFactor* is the implementation of the model described in paper “*Deep Factors for Forecasting*” [18]. This model is composed of a “global” and a “local” model. It uses is global (RNN) model to learn patterns across multiple related time series and an arbitrary local model to model the time series on a per time series basis. The referred paper evokes in particular : “*We assume that each time series is governed by the following two components: fixed and random. Fixed effects are common patterns that are given by linear combinations of K latent global deep factors. These deep factors can be thought of as dynamic principal components or eigen time series that drive the underlying dynamics of all the time series*” In the current implementation proposed by *GluonTS*, the local model is a RNN.

The tunable hyperparameters of the model are : the number of hidden layers and the number of hidden nodes per layer of the global model network and the number of factors . Their values are discussed in corresponding comparisons sections (3.4.4 and 5.3.4).

2.7.5 Gaussian Process

The model *GaussianProcess* is an implementation of a model using concept of gaussian processes , introduced in [19]. This model does not includes tunable hyperparameters.

2.7.6 NPTS

The model *NPTS* is the implementation of the “Non-Parametric Time Series Forecaster” model, described in paper [20]. It falls into the class of simple forecasters that use one of the past observed targets as the forecast for the current time step. It randomly samples a past time index as the prediction for the current time step. This model does not includes tunable hyperparameters, and doesn’t belongs to the deep learning model category, but is considered as one by *GluonTS*.

2.7.7 MQCNN

The model *MQCNN* is the implementation of one variant of the model described in paper “*A Multi-Horizon Quantile Recurrent Forecaster*” [21].

It belongs to the category of “Sequence-to-Sequence” models, for which *GluonTS* provides a template allowing to create a various models of this category. *MQCNN* is a pre-implemented model constructed on this template. “Sequence-to-sequence” models are composed of two main components. The encoder network, that encodes information about context interval in a latent state. And the decoder network, which generates the forecast of the prediction interval by combining the latent information with the features from the prediction range. In *MQCNN*, the encoder is a Convolutional Neural Network and the decoder an Multi-layer Perceptron. Unlike the

majority of the probabilistic forecasting models, the output is not a distribution of probability but the quantiles of this distribution, which are obtained individually by single-valued forecasting. This fundamental difference make impossible to modify the loss (see discussion in section 4.3).

The dimensions of the MLP (the number of nodes in the final layer, the number of hidden layers and the number of hidden nodes per layer) are tunable parameters. Their values are discussed in corresponding comparisons sections (3.4.5 and 5.3.5).

2.7.8 MQRNN

The model *MQRNN* is the implementation of one variant of the model described in paper "*A Multi-Horizon Quantile Recurrent Forecaster*" [21].

The model is the same than *MQCNN* excepting that the encoder is a Recurrent Neural Network. The tunable hyper parameters are the dimensions of this RNN (the number of nodes in the final layer, the number of hidden layers and the number of hidden nodes in each layers of the neural network). Their values are discussed in corresponding comparisons sections (3.4.6 and 5.3.6).

2.7.9 Transformer

The model *Transformer* is the implementation of "Transformer" model architecture, as it was defined in paper [22]. It is described in this paper as "*The first sequence transduction model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention*".

The tunable hyperparameters of the model are : the dimensions of the transformer network and the number of heads in the multi-head attention mechanism . Their values are discussed in corresponding comparisons sections (3.4.7 and 5.3.7).

2.7.10 Wavenet

The model *Wavenet* is the implementation of "Wavenet" model architecture, as it was defined in paper [23], with a quantized target. This model network is composed of dilated causal convolutional layers. Both residual and parameterised skip connections are used throughout the network, to speed up convergence and enable training of much deeper models.

The tunable hyperparameters are the number of residual channels in the architecture and the number of skip channels in the architecture. Their values are discussed in corresponding comparisons sections (3.4.8 and 5.3.8).

2.7.11 NBEATS

The model *NBEATS* is the implementation of a model based on a "NBEATS" Network, as described in the paper [24]. This model is defined as a deep neural ar-

chitecture based on backward and forward residual links and a very deep stack of fully-connected layers. *GluonTS* documentation precises that there are noteworthy differences in this implementation compared to the paper version. For example the parameter L_H defined in the paper is not implemented.

The tunable hyperparameters are the number of stacks the NBEATS network should contains and and the number of blocks per stack. Their values are discussed in corresponding comparisons sections (3.4.9 and 5.3.9).

Chapter 3

Model comparison using default loss and metrics

3.1 Technical Aspects

Before entering into sections implying quantitative experimental results, we need to define the hardware configuration. The code has been runned locally on a single-GPU computer with the configuration resumed in the following figure :

Operating System	Linux Ubuntu 20.04
CPU	Intel(R) Core(TM) i5-6500 CPU @ 3.20GHZ
GPU	Nvidia(R) GEFORCE GTX 1060 6 GB
RAM	8.00 Go
Main Programs	MXNet v1.6 (CUDA 9.6 compatible version), GluonTS v0.5.0

Table 3.1: Hardware Configuration

3.2 Testing protocol definition

Now that the different GluonTS components has been described and that the key components loss functions and metrics functions has been discussed, the different models that have been presented could be compared (after discussing their hyperparameter values). Each comparison using as element of comparison the metric *MASE*, as defined in section 2.5. The models uses their default loss as defined in section 2.4.

Before the comparison between different models, we need to compare the impact of the values of the different hyperparameters of the problem. The hyperparameters of the problem are :

- The context length, defined in section 1.4.

- The size of time series
- The learning rate of the model training
- The number of epochs of the training
- The output distribution, defined in section 2.6
- The own hyperparameters of the different models

The default input data configuration is A (see 2.3). Results for different input data configuration are not systematically showed because experimentation that has been performed (but are not all presented here) show that the configuration A/B does not significantly influences the choice of hyperparameters. Some results are presented : 3.3.5 and 3.3.3 and shows similar results for the two configurations. and the presentation of results for the two configurations would be redundant and time-consuming.

Concretely, the following subsections of the section 3.3 are organised as follows :

- We select one hyperparameter, commons to different models, to tune.
- The model is run for different values of this hyperparameter, with other hyperparameters fixed to default *GluonTS* values if the hyperparameter as not been already tuned or fixed to values that has been observed as optimal during previous hyperparameter tuning. All these hyperparameters values are described as “default” in the following subsections.
- Results are presented as histogram, with on the abscissa the hyperparameter value and on the ordinate the *MASE* metric value obtains by the trained model.
- We deduce the optimal value of the parameter from these results, knowing that the goal is the minimization of the *MASE* value

Concretely, the following subsections of the section 3.4 are organised as follows :

- We select one model between the implemented models presented in section 2.7.
- For each of it tunable hyperparameter, the model is run for a range of this hyperparameter values, with other hyperparameters fixed to default *GluonTS* values if the hyperparameter as not been already tuned or fixed to values that has been observed as optimal during previous hyperparameter tuning. All these hyperparameters values are described as “default” in the following subsections. Some models does not have tunable hyperparameter.
- Results are presented as histogram, with on the abscissa the model hyperparameter value and on the ordinate the *MASE* metric value obtains by the trained model.
- Using these results, optimal value of the parameter is obtains, knowing that the goal is the minimization of the *MASE* value

- We present the plot of the forecasting result, with all hyperparameters of the model fixed to optimal values as it has been discussed in this section. The predicted distribution is visually represented by its quantiles. The plot results are all presented with the same values of quantiles, i.e. the quantile *quantile*(0.99) and the respective *quantile*(0.01) but also the *quantile*(0.9), *quantile*(0.1) and the median.

Finally we had the section presenting the comparison of all implemented models, with all their hyperparameters tuned. Results are presented as an histogram with on the abscissa the different models tested and on the ordinate the *MASE* metric value obtained by the trained model.

In the following sections, the usage of the parameter value notation [a,b,..], for example [10,20], corresponds to a vector parameter, describing in general a neural network architecture, where a,b,.. are the number of cells in each layer.

3.3 Hyperparameters commons to different models comparison

3.3.1 Context length

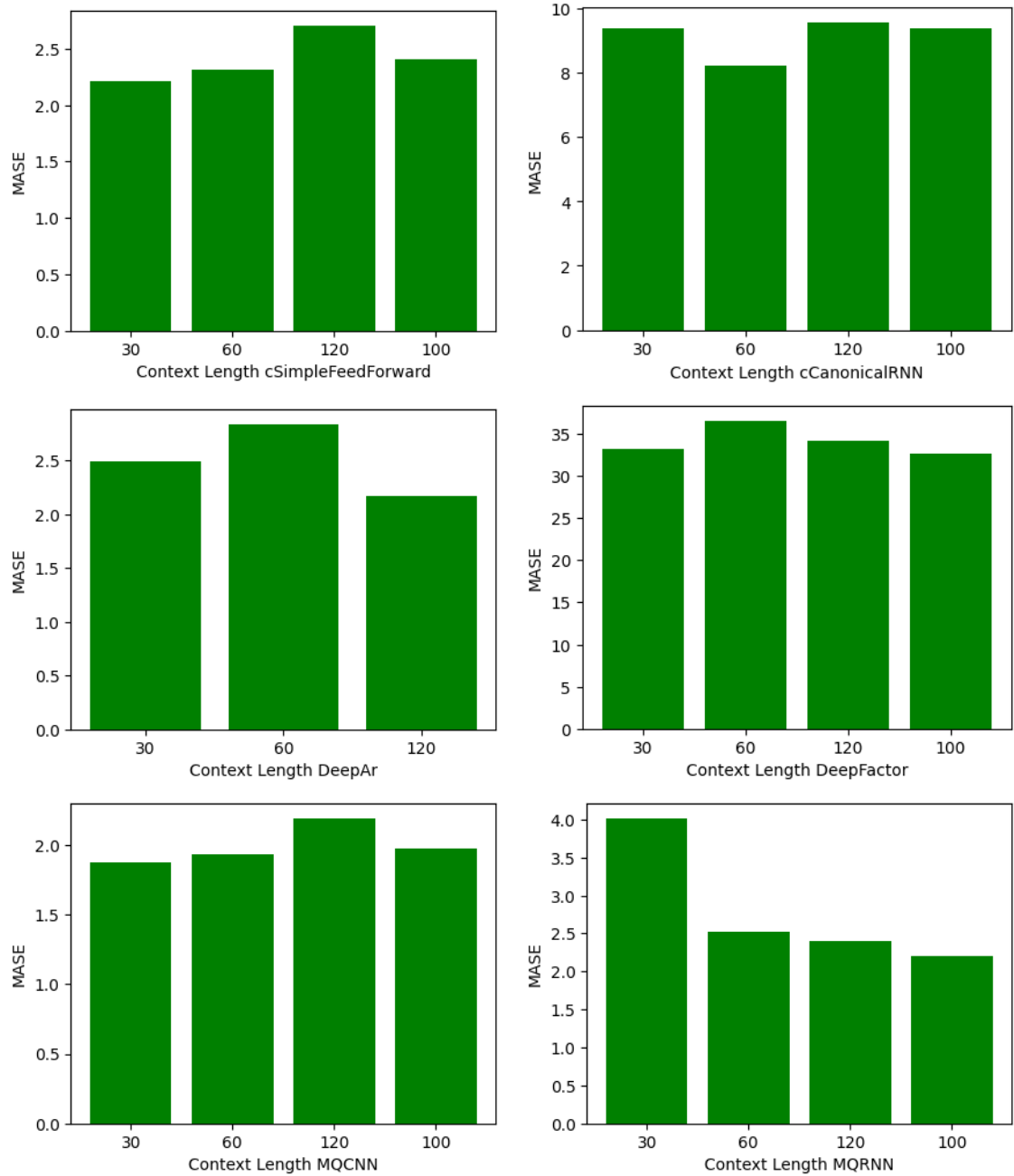


Figure 3.1: Comparison of different context lengths for different models (Models: *SimpleFeedForward*, *CanonicalRNN*, *DeepAr*, *DeepFactor*, *MQCNN*, *MQRNN*, *Wavenet* (Default hyperparameter values), Config A)

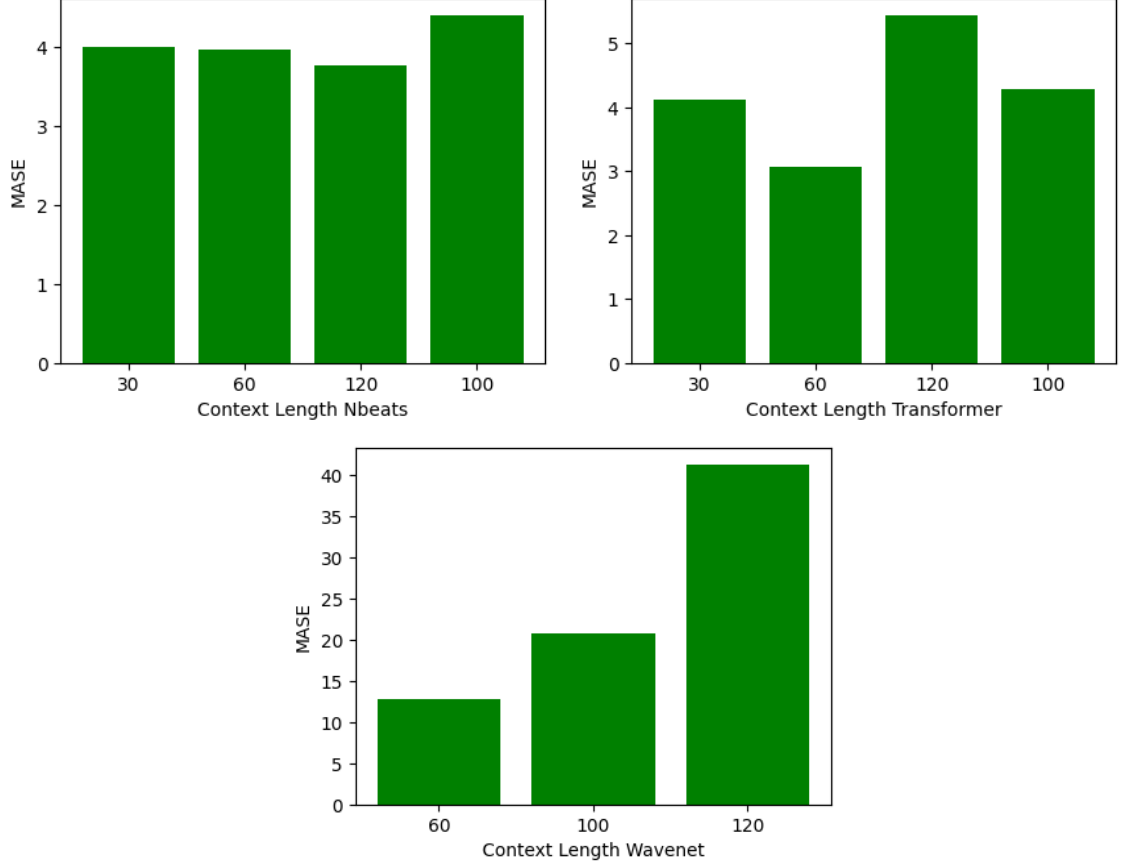


Figure 3.2: Comparison of different context length for different models (Models: *NBEATS*, *Transformer*, *Wavenet* (Default hyperparameter values), Config A)

The context length, as it was defined in section 1.4, is a crucial hyperparameter of the defined problem, as it define the input size of the model neural network, which might have impact on the prediction result.

Context lenght is implied as a hyperparameter in 9 tested models. As results varies in function of the models, results are presented for each of them, in figures 3.1 and 3.2.

We observe that this hyperparameter influences widely the results for a certain number of models. For example there is a factor 1.25 between the better and worst context lenght value for *SimpleFeedForward* or *DeepAr* model and a factor of 2 for *MQRNN*. These results allows us to deduce the optimal value of context length for each models. Models *SimpleFeedForward* and *MQCNN* optimal value is 30, *CanonicalRNN*, *Wavenet* and *Transformer* is 60, *MQRNN* and *DeepFactor* is 100, *DeepAr* and *NBEATS* is 120. These values are considered as default values in the following comparison sections.

3.3.2 Time series size

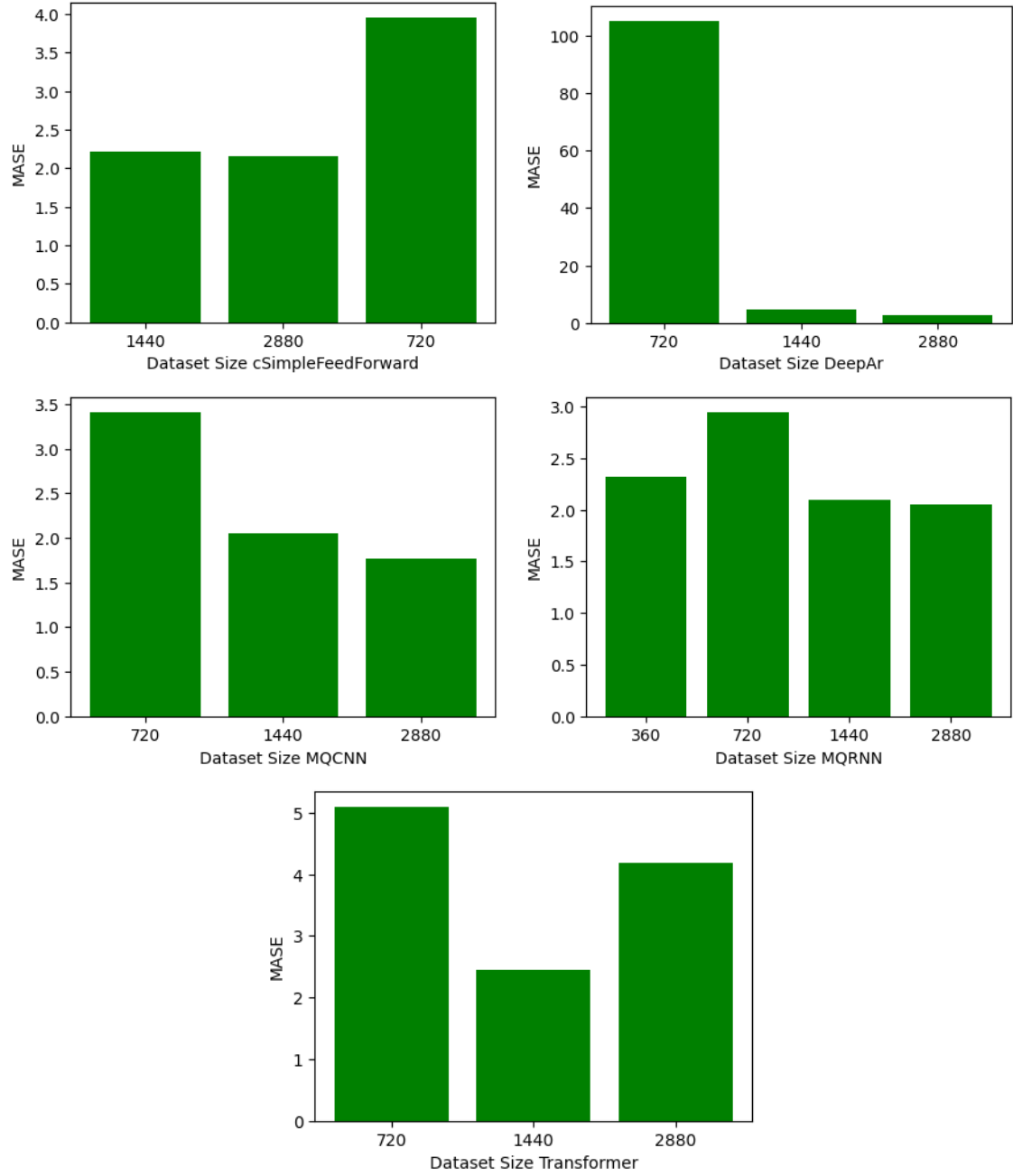


Figure 3.3: Comparison of different time series sizes (Models: *SimpleFeedForward*, *DeepAr*, *MQCNN*, *MQRNN*, *Transformer* (Default parameters values), Config A)

The size of time series that are given to dataset object might influence the prediction quality. It can be seen as how much the initial time series are splitted manually before putting them in dataset. Splitting the initial time series implies losing information (all the training intervals straddling two splitted time series).

Different values tested corresponds to time series of 2-day length (2880 time steps),

1-day length (1440 time steps), 1/2-day length (720 time steps) and 1/4-day length (360 time steps), bigger values than 2800 are less convenient as one of the original time series at our disposition is 2 days long.

Results, presented in figure 3.3.2, allows us to deduce the optimal value of time series size for each models. Model *Transformer* optimal value is 1440. All other models optimal values are 2880 (Results are only showed for SimpleFeedForward, DeepAr, MQCNN and MQRNN). These values are considered as default values in the following comparison sections.

3.3.3 Learning rate

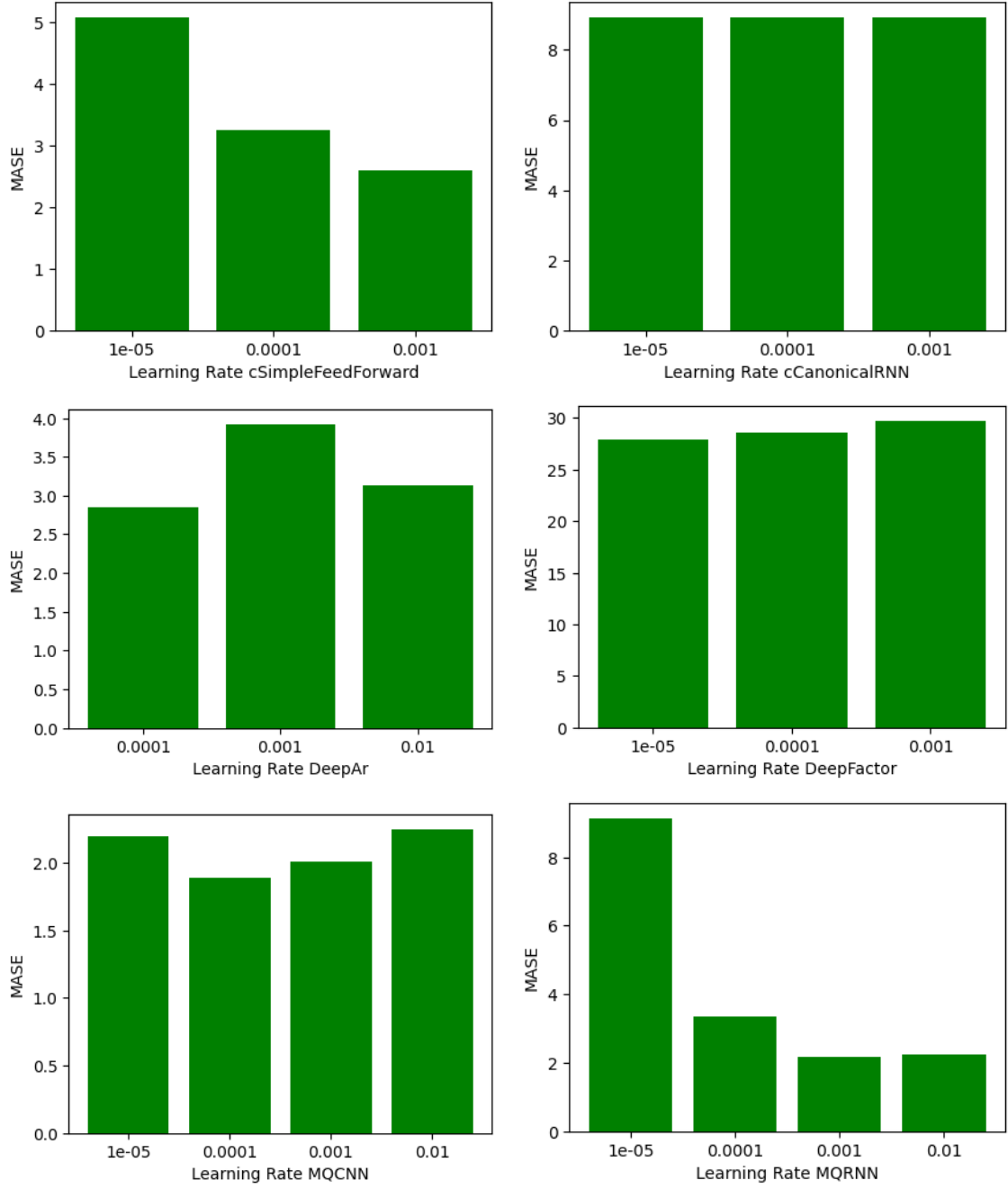


Figure 3.4: Comparison of different learning rates for different models (Models: *SimpleFeedForward*, *CanonicalRNN*, *DeepAr*, *DeepFactor*, *MQCNN*, *MQRNN* (Default hyperparameter values), Config A)

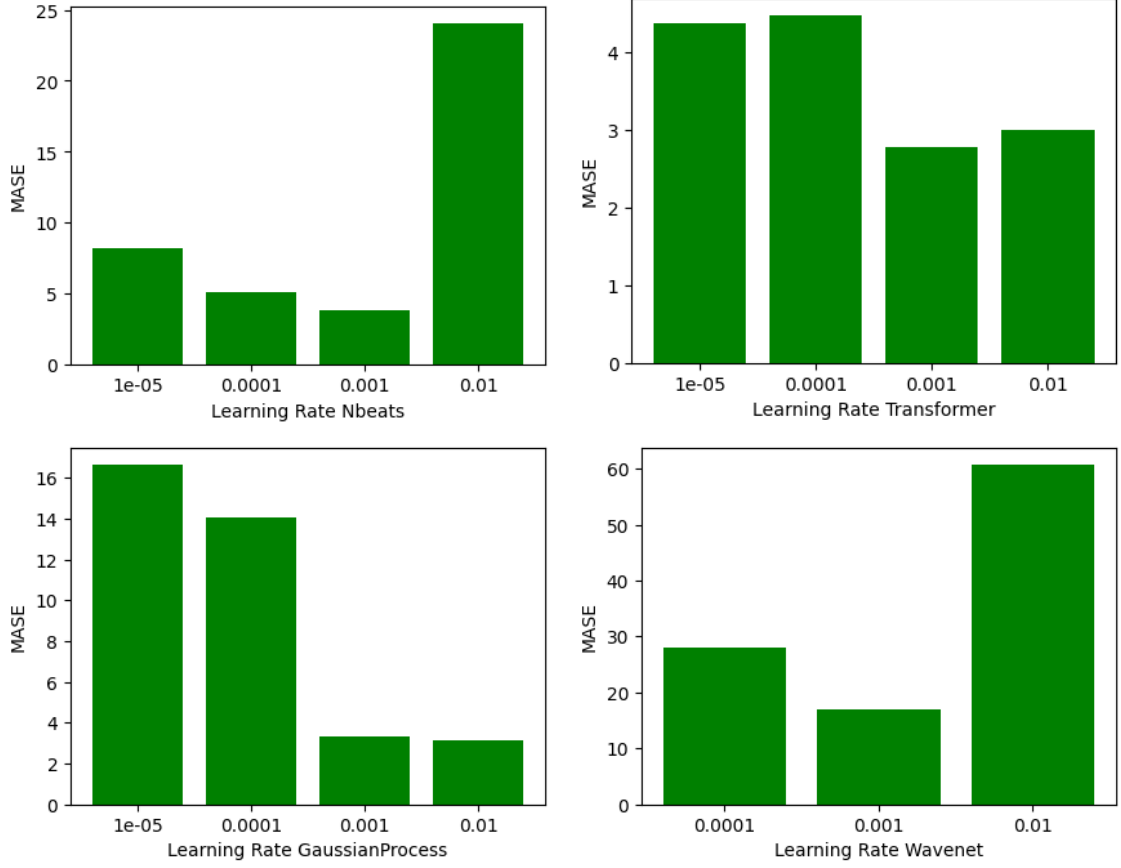


Figure 3.5: Comparison of different learning rates for different models (Models: *NBEATS*, *Transformer*, *GaussianProcess*, *Wavenet* (Default hyperparameter values), Config A)

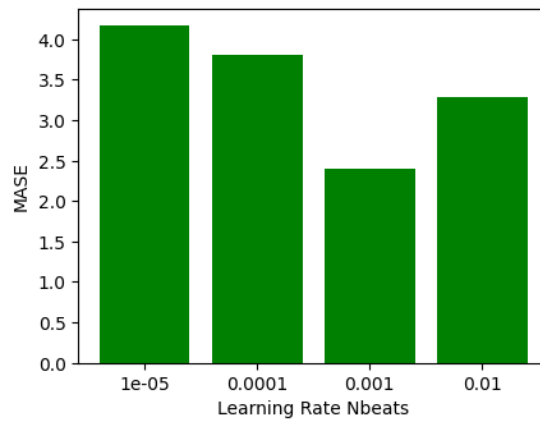


Figure 3.6: Comparison of different learning rates (Models: *NBEATS* (Default hyperparameter values), Config B)

Learning rate is a key element of deep learning training, as it drives the weight updates of the models neural networks. The default learning rate in *GluonTS* for all deep learning models is 10^{-3} .

Learning rate is implied as a hyperparameter in 10 tested models. As results varies in function of the models, results are presented for each of them, in figures 3.4 and 3.5.

We observe that this hyperparameter influences widely the results for all models except *CanonicalRNN* and *DeepFactor*. These results allows us to deduce the optimal value of learning rate for each models. Models *SimpleFeedForward*, *CanonicalRNN*, *MQRNN*, *Nbeats*, *Wavenet* and *Transformer* optimal value is 10^{-3} , *GaussianProcess* is 10^{-2} , *DeepAr* and *MQCNN* is 10^{-4} and *DeepFactor* is 10^{-5} . These values are considered as default values in the following comparison sections.

3.3.4 Epochs

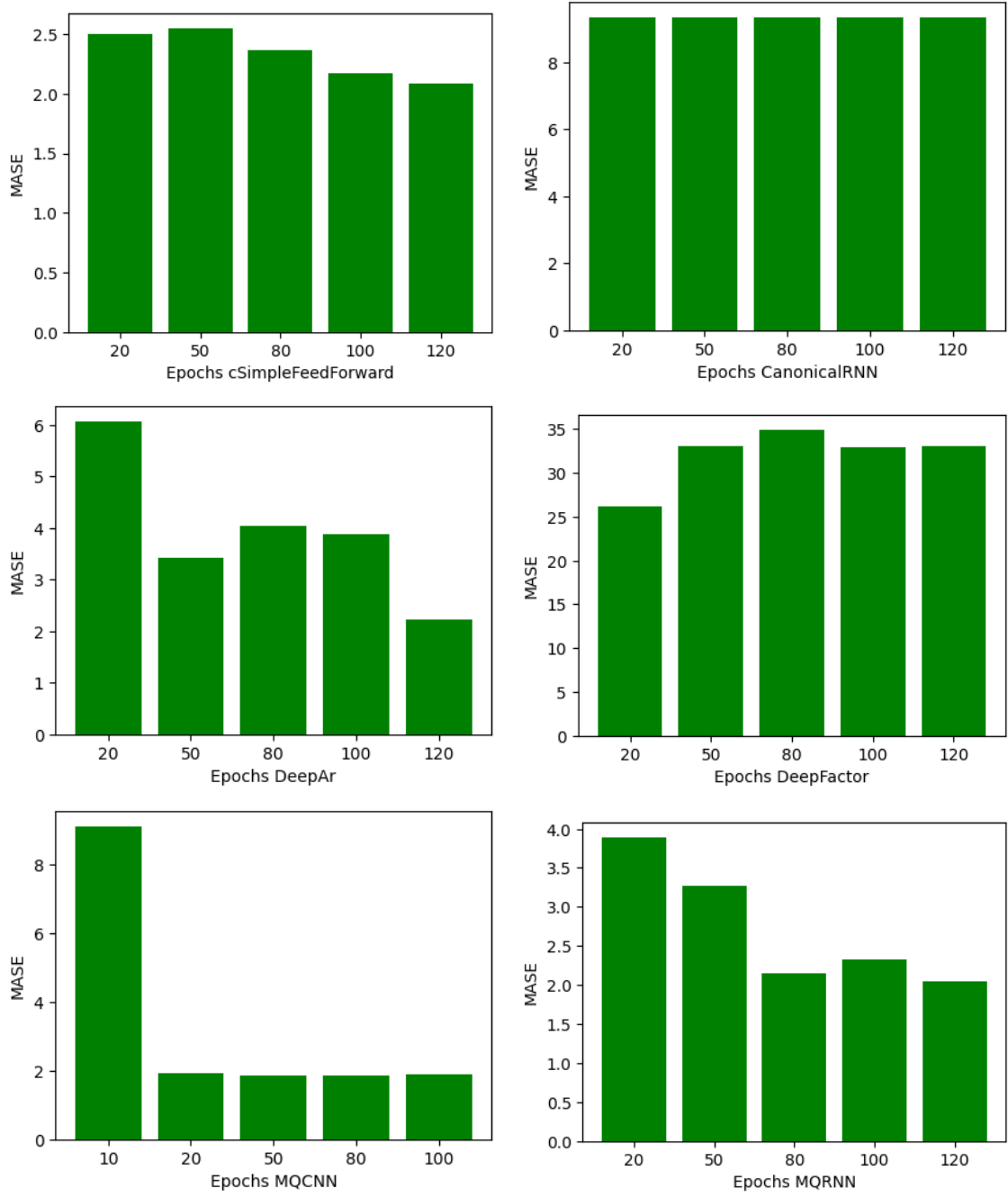


Figure 3.7: Comparison of different epochs values for different models (Models: *SimpleFeedForward*, *CanonicalRNN*, *DeepAr*, *DeepFactor*, *MQCNN*, *MQRNN* (Default hyperparameter values), Config A)

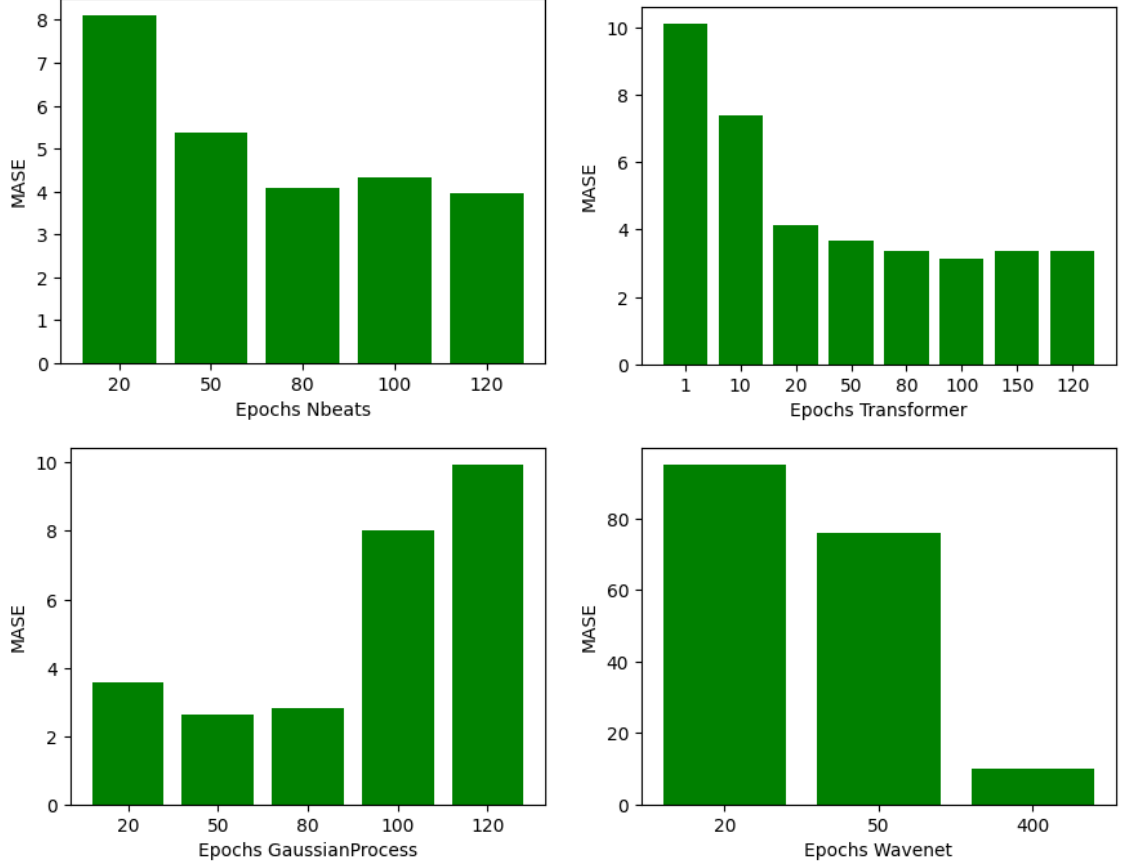


Figure 3.8: Comparison of different epochs values for different models (Models: *NBEATS*, *Transformer*, *GaussianProcess*, *Wavenet* (Default hyperparameter values), Config A)

Models training implies of a certain number of epochs. This number influences the results, as the model must be trained aptly to avoid underfitting or overfitting. The default number of epochs in *GluonTS* is 100 epochs for all models except *Wavenet* (200 epochs).

Epochs is implied as a hyperparameter in 10 tested models. As results varies in function of the models, results are presented for each of them, in figures 3.7 and 3.8.

We observe that this hyperparameter has a great influence in results. These results allows us to deduce the optimal value of epochs for each models. Models *Deep Factor* and *CanonicalRNN* optimal value is 20, *GaussianProcess* and *MQCNN* is 50, *MQRNN* is 80, *Transformer* is 100, *NBEATS*, *DeepAr* and *SimpleFeedForward* is 120, *Wavenet* is 400. These values are considered as default values in the following comparison sections.

3.3.5 Output Distribution

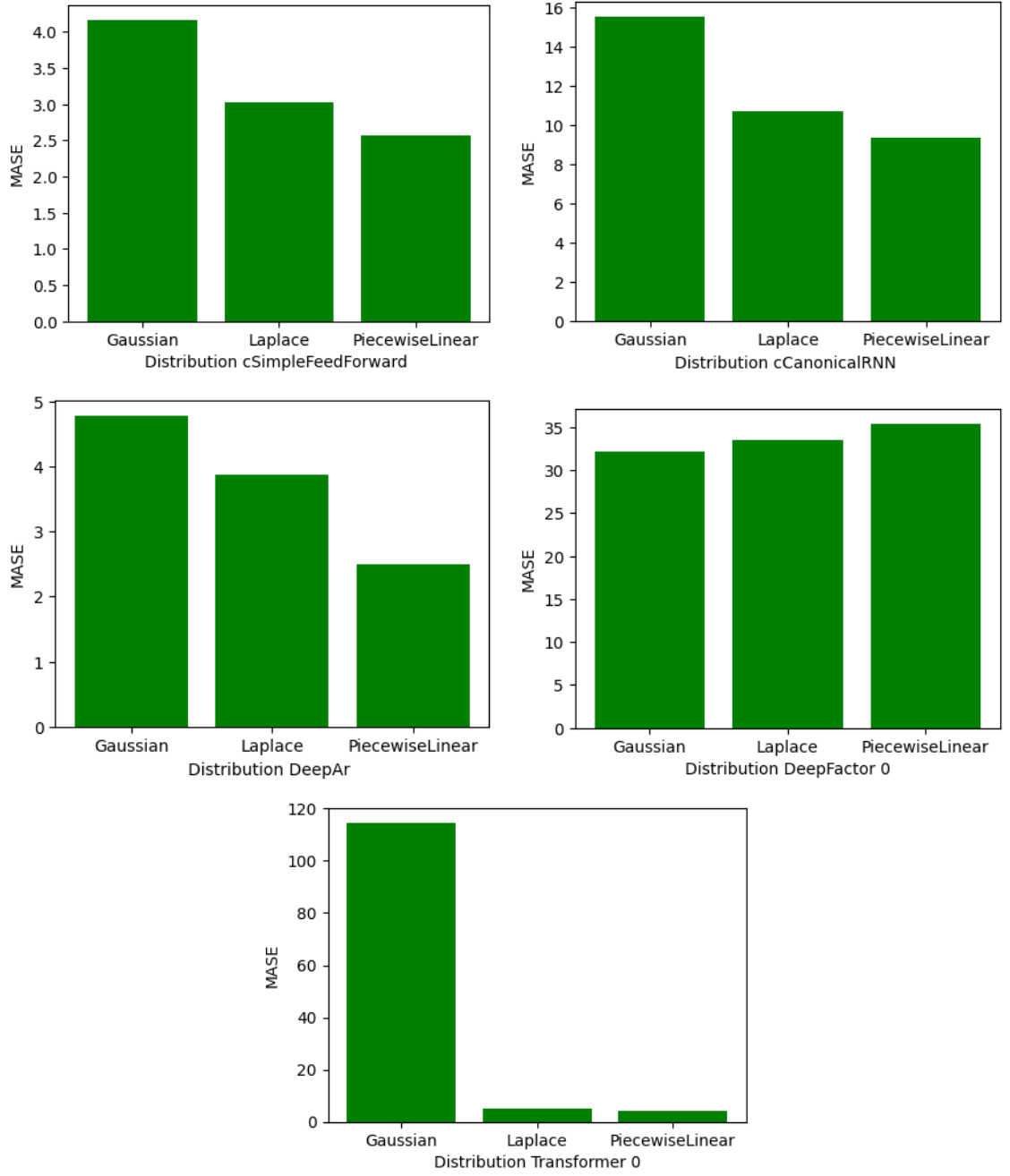


Figure 3.9: Comparison of different output distributions for different models (Models: *SimpleFeedForward*, *CanonicalRNN*, *DeepAr*, *DeepFactor*, *Transformer* (Default hyperparameter values), Config A)

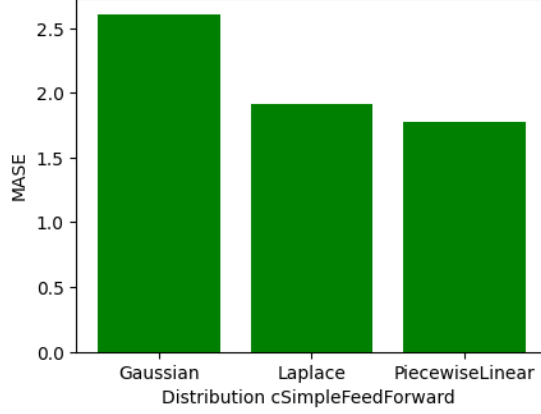


Figure 3.10: Comparison of different output distributions (Model: *SimpleFeedForward* (Default hyperparameter values), Config B)

In the case of the models that outputs a probability distribution via distribution parameters (as it is the case in problem statement section 1.4), the type of distribution must be specified. The models *MQRNN*, *MQCNN*, *GaussianProcess*, *Nbeats*, *Wavenet* are excluded from this subsection, as they does not used distribution parameters. The set of possible output distribution has been defined in section 2.6.

Results shows that, for all models with exception of the *DeepFactor* model case, *Gaussian* distribution (and *Laplace*) is outperformed by *PiecewiseLinear* distribution, which gives optimal results among the three output distribution considered. This optimal distribution are considered as default in following comparison sections. These result contradict the affirmation that the better output distributions in the context of wind turbines production forecasting are *Gaussian* and *Beta*, as stated in [14], and shows impressive improvements using the *PiecewiseLinear* distribution, especially in the case of *Transformer* model.

3.3.6 Number of Pieces of PiecewiseLinear distribution

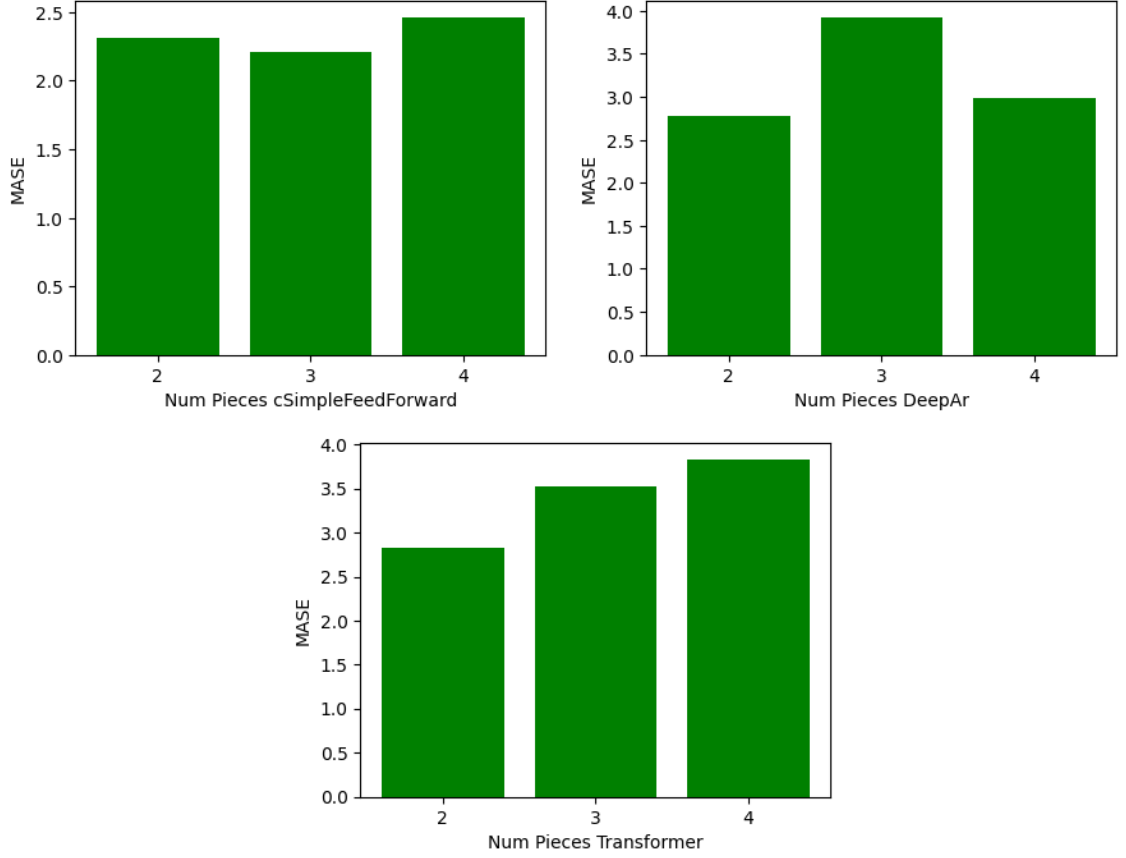


Figure 3.11: Comparison of different number of pieces for *PiecewiseLinear* distribution for different model (Models: *SimpleFeedForward*, *DeepAr*, *Transformer* (Default parameter values), Config : A)

The piecewise linear output distribution, which belongs to the set of disposable output distribution, is composed of a certain number of pieces, number that we must define. As it influences the shape of the distribution, we might study the impact of this parameter.

Starting from a 2 pieces output distribution, we observe in the results that adding pieces does not improve significantly the prediction where increasing the size of the neural network, as it increases the number of output distribution parameters . We consider thence a number of 2 pieces for models using *PiecewiseLinear* distribution as default in the following comparison sections.

3.4 Different models hyperparameters comparison and plot results

3.4.1 Feed Forward

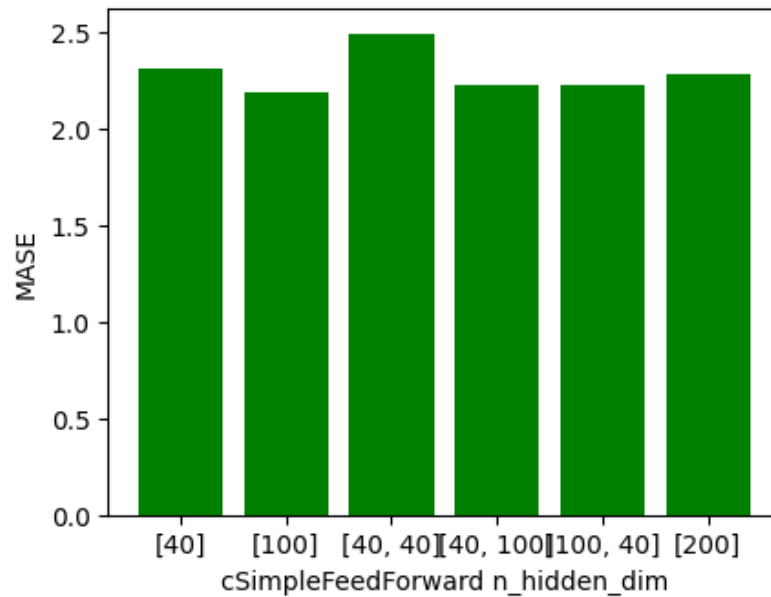


Figure 3.12: Comparison of different n_hidden_dim values for *SimpleFeedForward* model (Default hyperparameter values), Config A)

FeedForward model is an implemented model described in section 2.7.1. Its own tunable hyperparameter is the dimensions of the neural network hidden layers, named n_hidden_dim . The default value in GluonTS is [40]. Results shows that the optimal value is [100]. Plots results, illustrating the model predictions, are presented in figure 3.13.

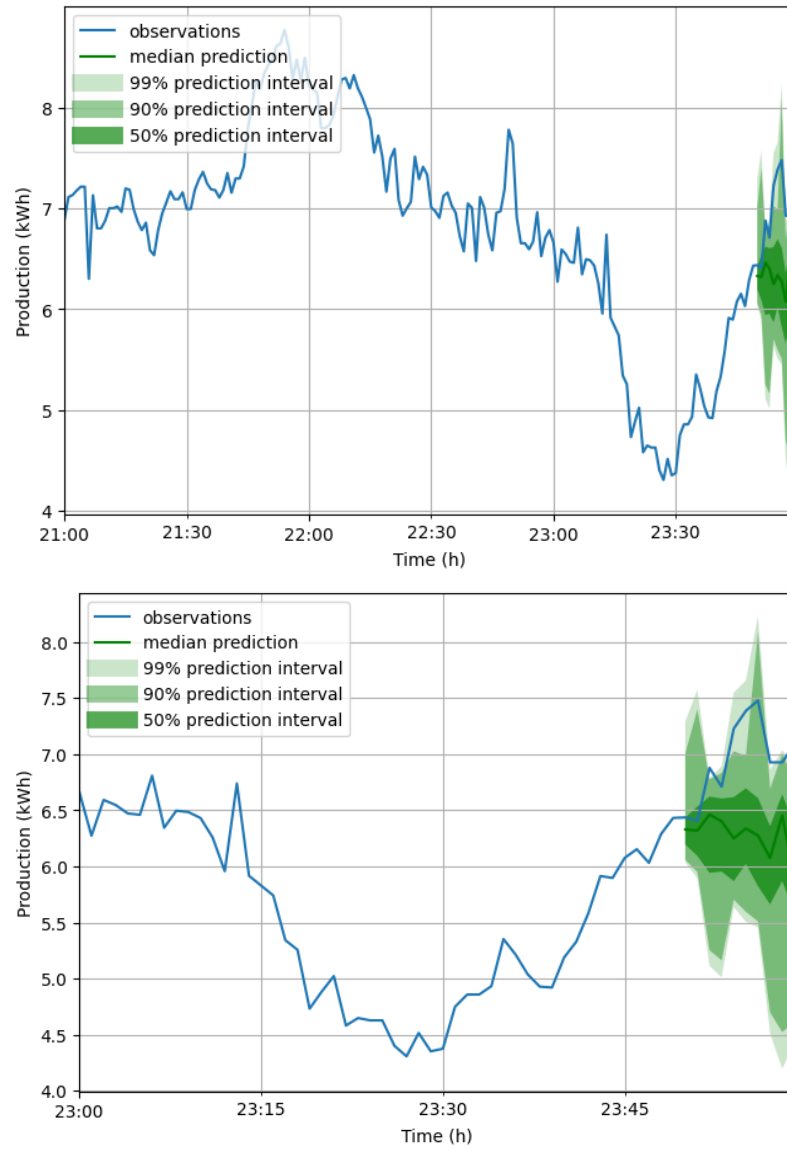


Figure 3.13: Forecast result of *SimpleFeedForward* model at 3 hours and 1 hour scale ($n_hidden_dim = [100]$, Default hyperparameter values), Config A)

3.4.2 Canonical RNN

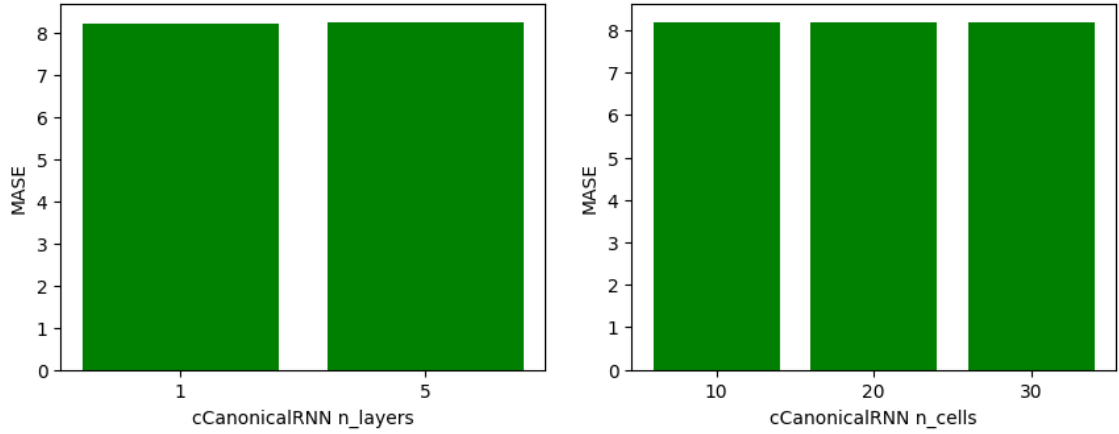


Figure 3.14: Comparison of different n_layers and n_cells values for Canonical RNN model (Default hyperparameter values, Config A)

CanonicalRNN model is an implemented model described in section 2.7.2. Its own tunable hyperparameters are the dimensions of the neural network hidden layers, represented by n_layers and n_cells . The default values in *GluonTS* are 1 for n_layers and 20 for n_cells . Results shows that different values of these parameters make almost no difference in terms of results. Plots results, illustrating the model predictions, are presented in figure 3.15.

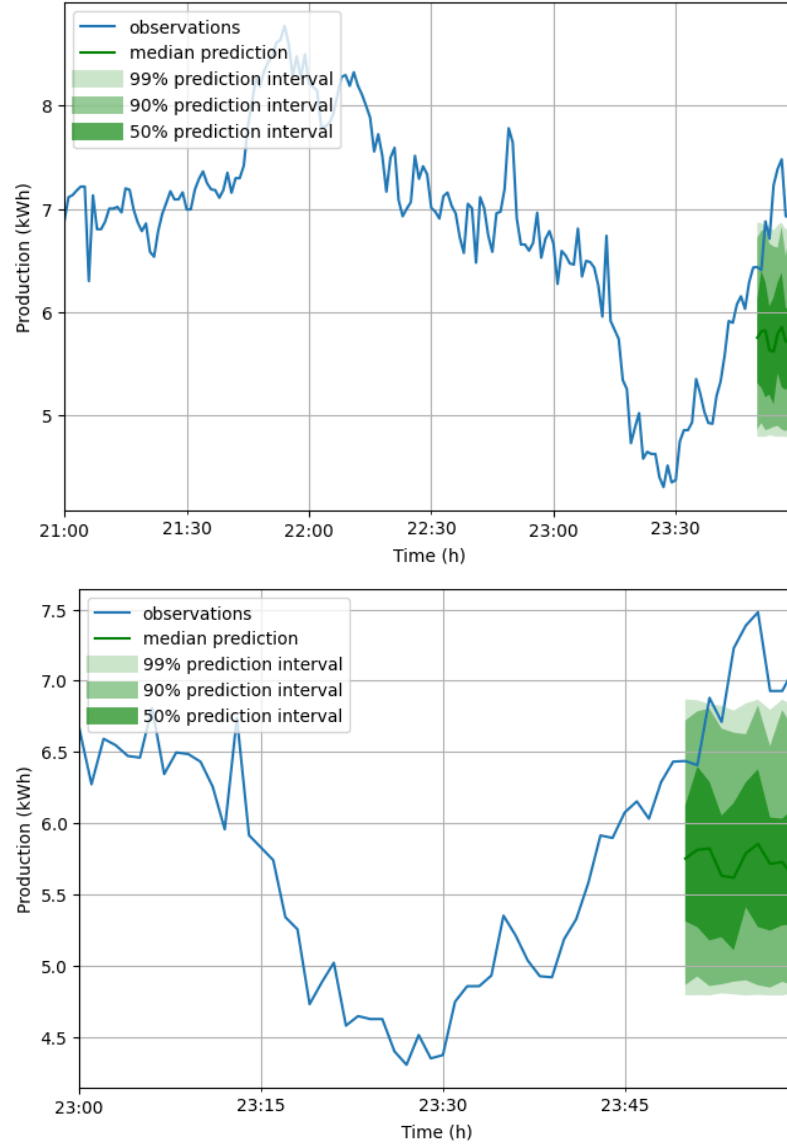


Figure 3.15: Forecast result of *CanonicalRNN* model at 3 hours and 1 hour scale ($n_{layers} = 4$, $n_{cells} = 20$ Default hyperparameter values, Config A)

3.4.3 Deep AR

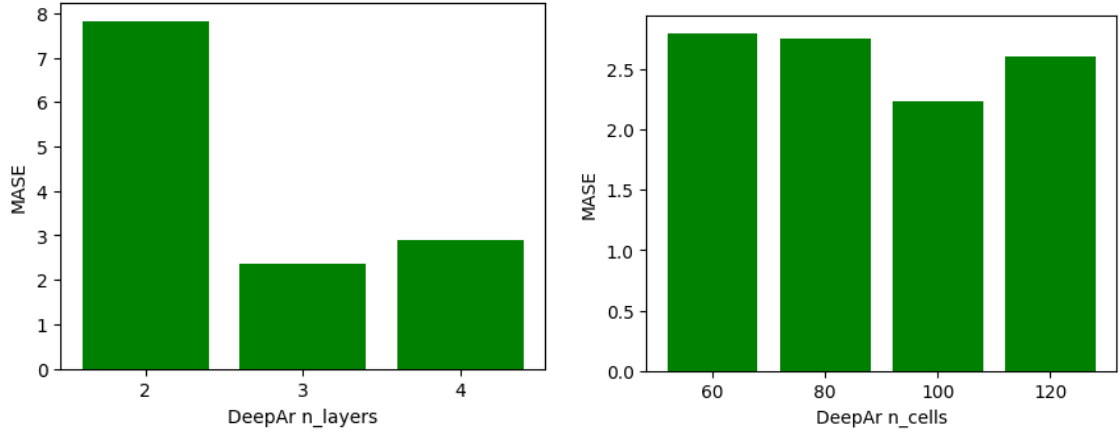


Figure 3.16: Comparison of different n_{layers} and n_{cells} values for *DeepAR* model (Default hyperparameter values, Config A)

DeepAr model is an implemented model described in section 2.7.3. Its own tunable hyperparameters are the dimension of the neural network hidden layers, represented by n_{layers} and n_{cells} . The default values in *GluonTS* are 2 for n_{layers} and 40 for n_{cells} . Results shows that the optimal values are 3 for n_{layers} and 100 for n_{cells} . Plots results, illustrating the model predictions, are presented in figure 3.17.

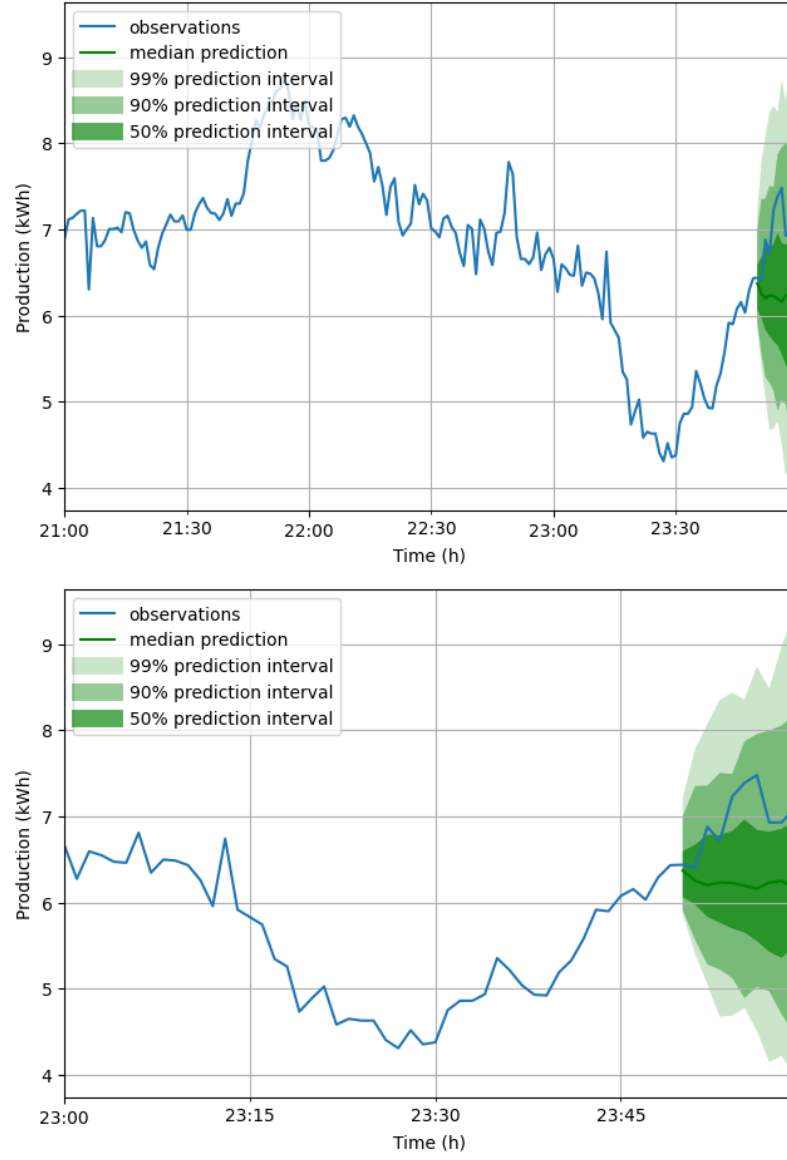


Figure 3.17: Forecast result of *DeepAr* model at 3 hours and 1 hour scale ($n_{layers} = 3$, $n_{cells} = 100$, Default hyperparameter values, Config A)

3.4.4 Deep Factor

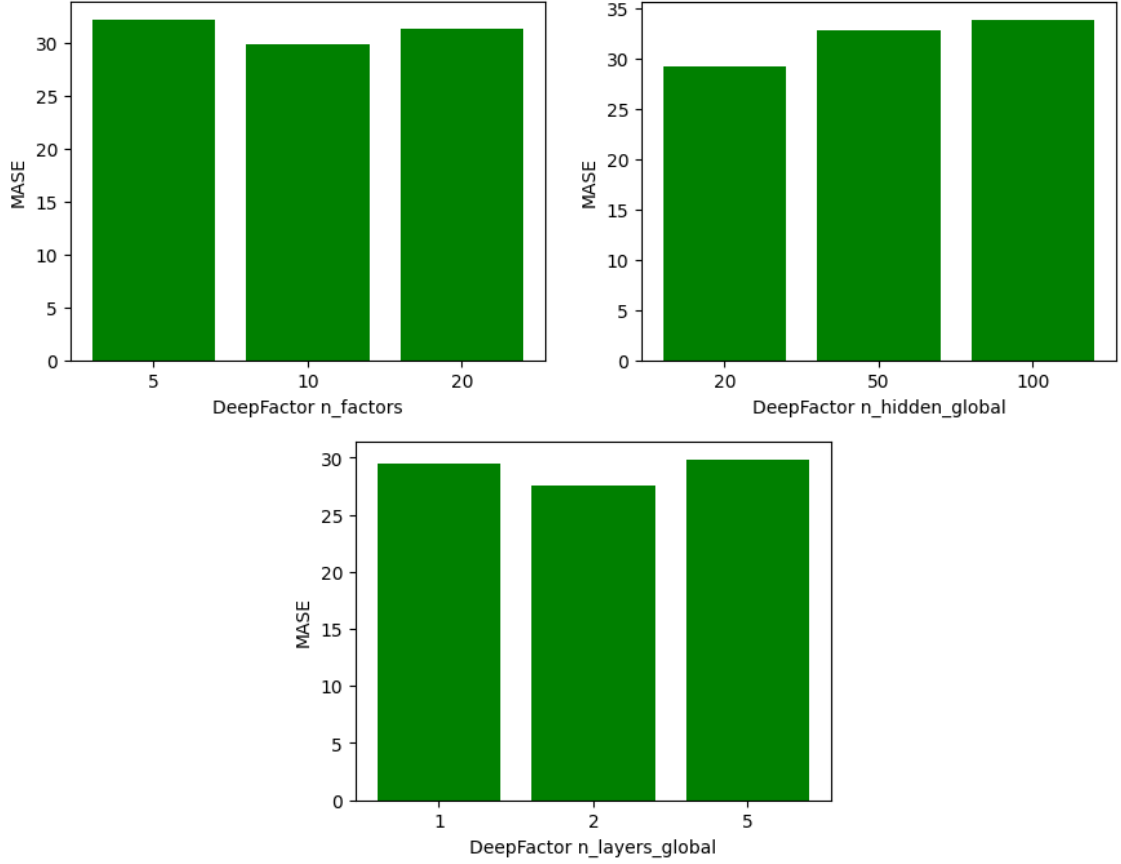


Figure 3.18: Comparison of different $n_factors$ values, n_hidden_global values and n_layers_global values for *DeepFactor* model (Default hyperparameter values, Config A)

DeepFactor model is an implemented model described in section 2.7.4. Its own tunable hyperparameters are the number of factors ($n_factors$) and the dimensions of the global neural network hidden layers, represented by n_layers_global for the number of layers and n_hidden_global for the number of cells in each layers. The default values in *GluonTS* are 10 for $n_factors$, 50 for n_hidden_global and 1 for n_layers_global . Results shows that the optimal values are 10 for $n_factors$, 20 for n_hidden_global and 2 for n_layers_global . We can nevertheless point that results are in any cases very high in *MASE* compared to the other models that have already been presented. Plots results, illustrating the model predictions, are presented in figure 3.19.

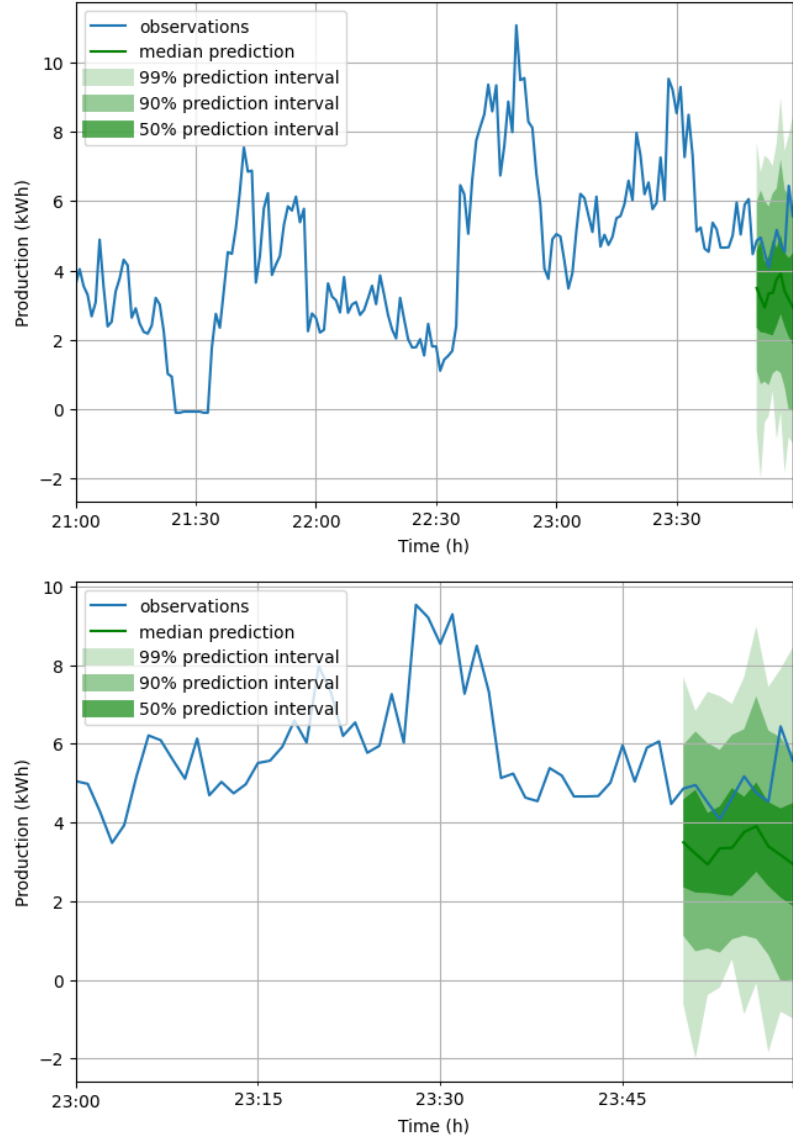


Figure 3.19: Forecast result of *DeepFactor* model at 3 hours and 1 hour scale ($n_factors = 10$, $n_hidden_global = 20$, $n_layers_global = 2$, Default hyperparameter values, Config A)

3.4.5 MQCNN

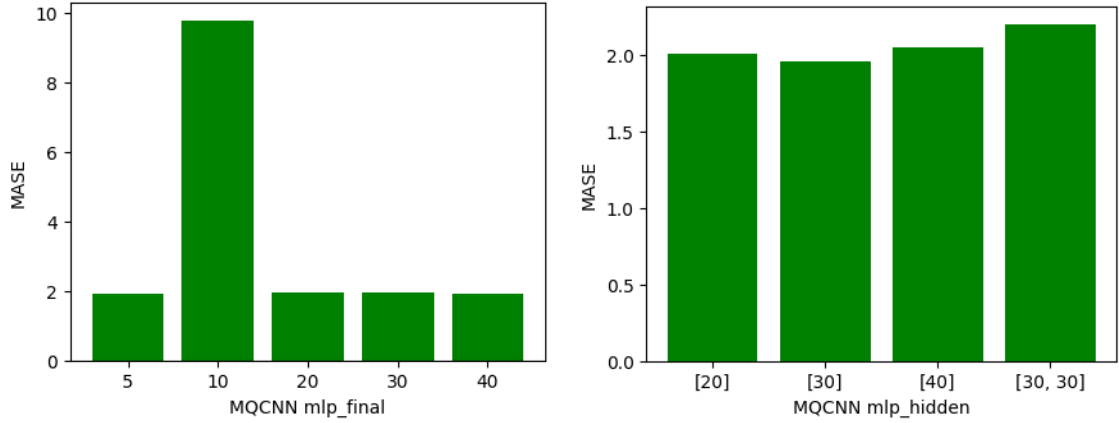


Figure 3.20: Comparison of different *mlp_final_dim* and *mlp_hidden* values for *MQCNN* model (Default hyperparameter values, Config A)

MQCNN model is an implemented model described in section 2.7.7. Its own tunable hyperparameters are the dimensions of the neural network layers, represented by *mlp_final* for the dimension of the final layer and *mlp_hidden* for the dimension of hidden layers. The default values in GluonTS are 20 for *mlp_final* and [30] for *mlp_hidden*. Results shows that the optimal values are 40 for *mlp_final* and [30] for *mlp_hidden*. Plots results, illustrating the model predictions, are presented in figure 3.21.

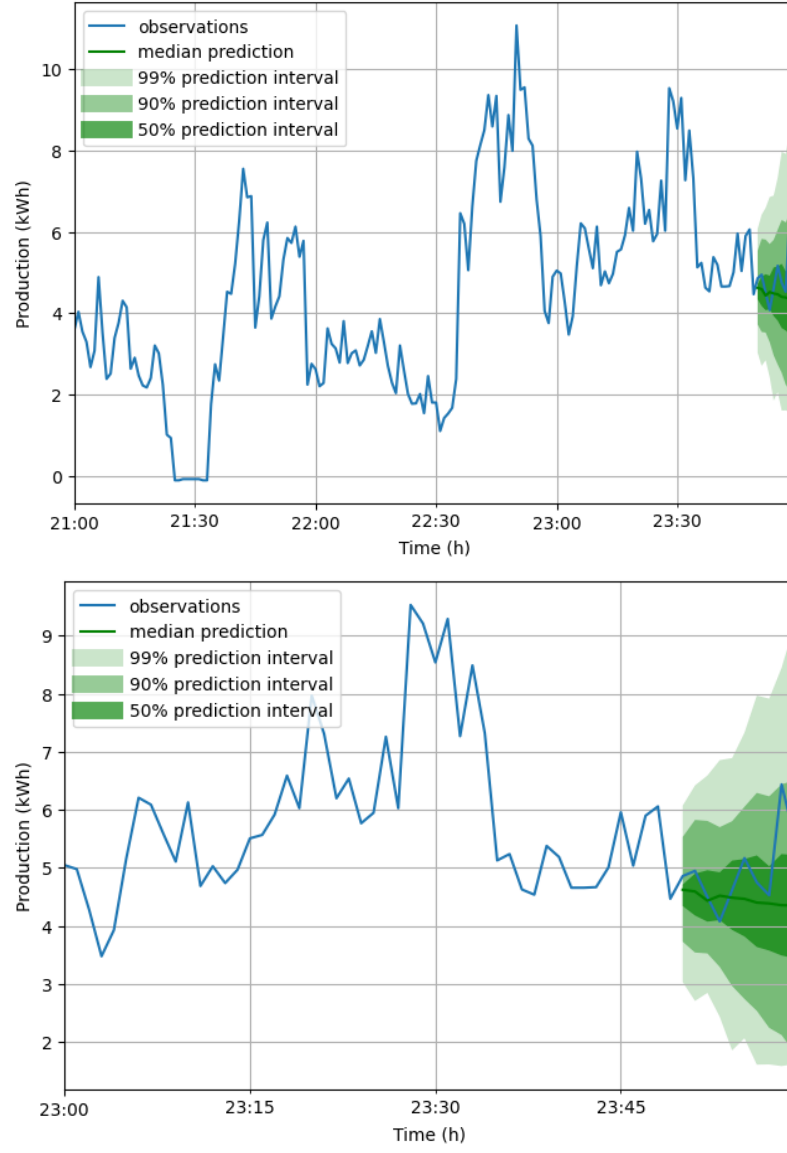


Figure 3.21: Forecast result of *MQCNN* model at 3 hours and 1 hour scale ($mlp_final_dim = 40$ and $mlp_hidden = [30]$, Default hyperparameter values, Config A)

3.4.6 MQRNN

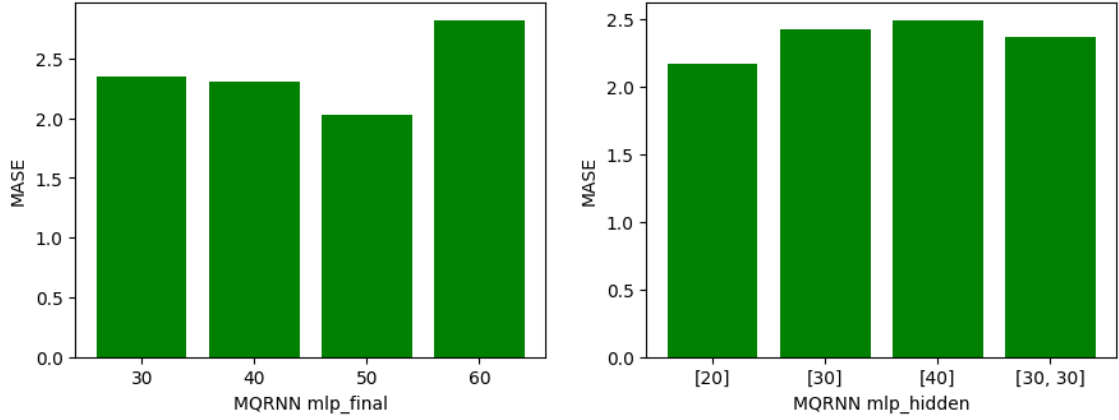


Figure 3.22: Comparison of different *mlp_final_dim* and *mlp_hidden* values for MQRNN model (Default hyperparameter values, Config A)

MQRNN model is an implemented model described in section 2.7.8. Its own tunable hyperparameters are the dimension of the neural network layers, represented by *mlp_final* for the dimension of the final layer and *mlp_hidden* for the dimension of hidden layers. The default values in GluonTS are 20 for *mlp_final* and [30] for *mlp_hidden*. Results shows that the optimal values are 50 for *mlp_final* and [20] for *mlp_hidden*. Plots results, illustrating the model predictions, are presented in figure 3.23.

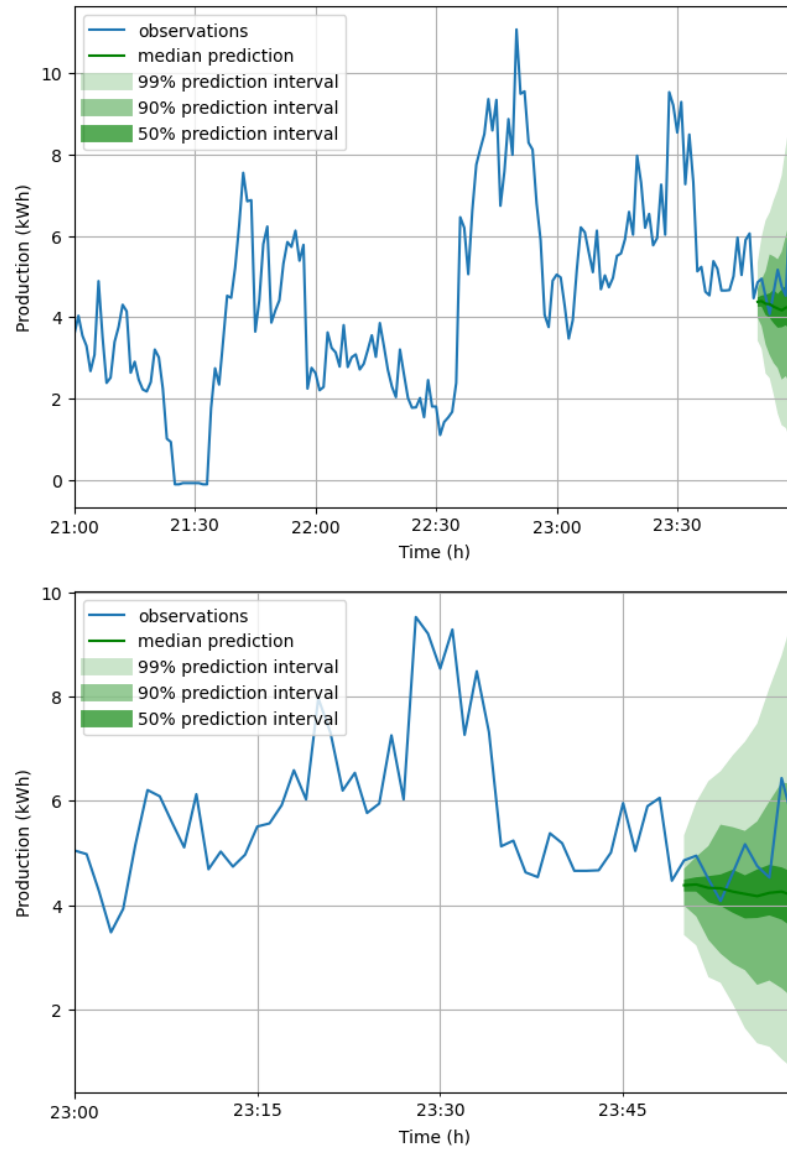


Figure 3.23: Forecast result of MQRNN model at 3 hours and 1 hour scale ($mlp_final_dim = 50$ and $mlp_hidden = [30]$, Default hyperparameter values, Config A)

3.4.7 Transformer

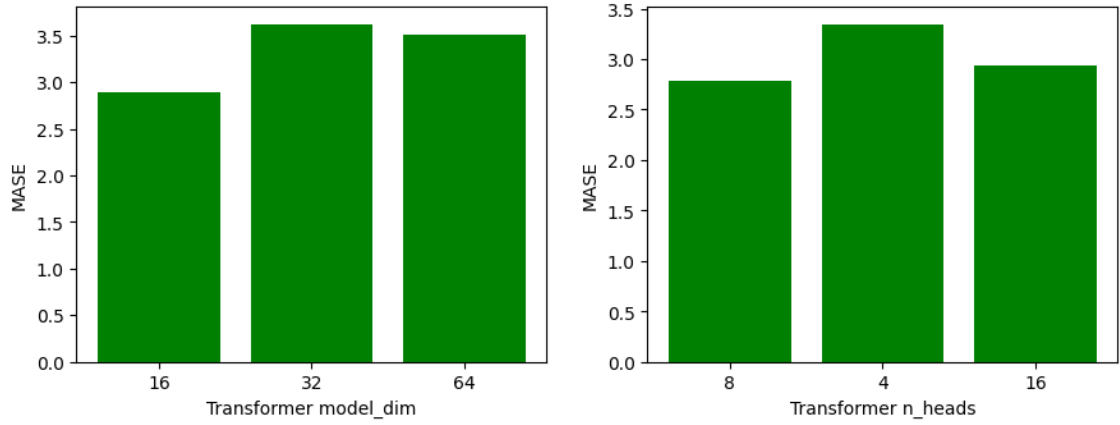


Figure 3.24: Comparison of different *model_dim* and *n_heads* values for *Transformer* model (Default hyperparameter values, Config A)

Transformer model is an implemented model described in section 2.7.9. Its own tunable hyperparameters are the dimension of the transformer network (*model_dim*) and the number of heads in the multi-head attention mechanism (*n_heads*). The default values in GluonTS are 16 for *model_dim* and 8 for *n_heads*. Results shows that the optimal values are the same that the default ones. Plots results, illustrating the model predictions, are presented in figure 3.25.

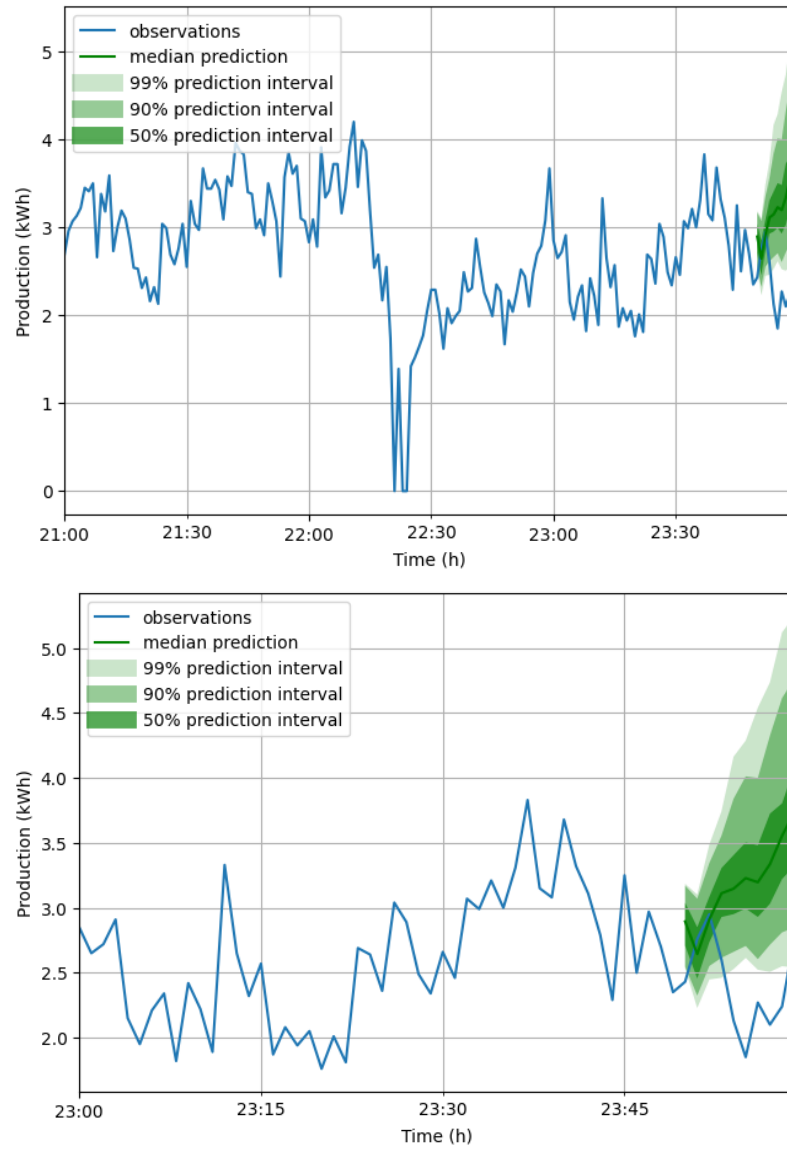


Figure 3.25: Forecast result of *Transformer* model at 3 hours and 1 hour scale ($model_dim = 16$ and $n_heads = 8$, Default hyperparameter values, Config A)

3.4.8 Wavenet

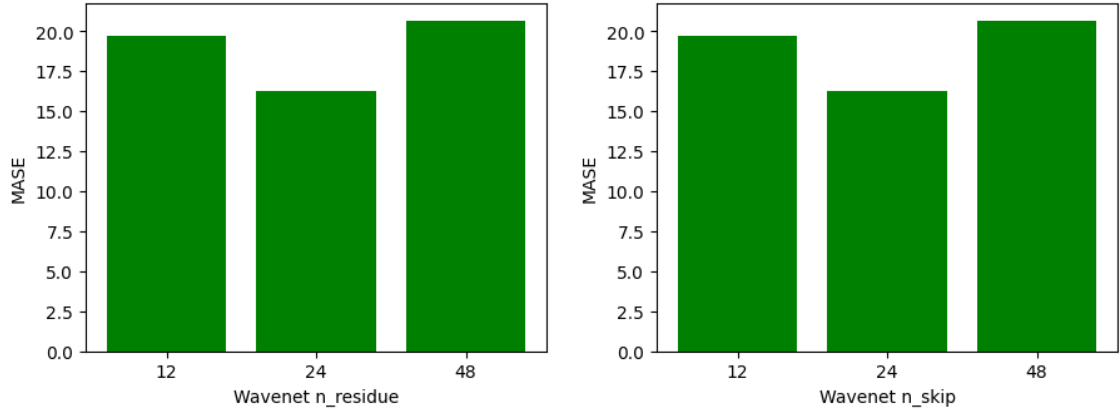


Figure 3.26: Comparison of different $num_residue$ and n_skip values for *Wavenet* model (Default hyperparameter values, Config A)

Wavenet model is an implemented model described in section 2.7.10. Its own tunable hyperparameters are the number of residual channels in wavenet architecture (n_res) and the number of skip channels in wavenet architecture (n_skip). The default values in GluonTS are 24 for n_res and 8 for n_skip . Results shows that the optimal values are the same that the default ones.

Plots results, illustrating the model predictions, are presented in figure 3.27.

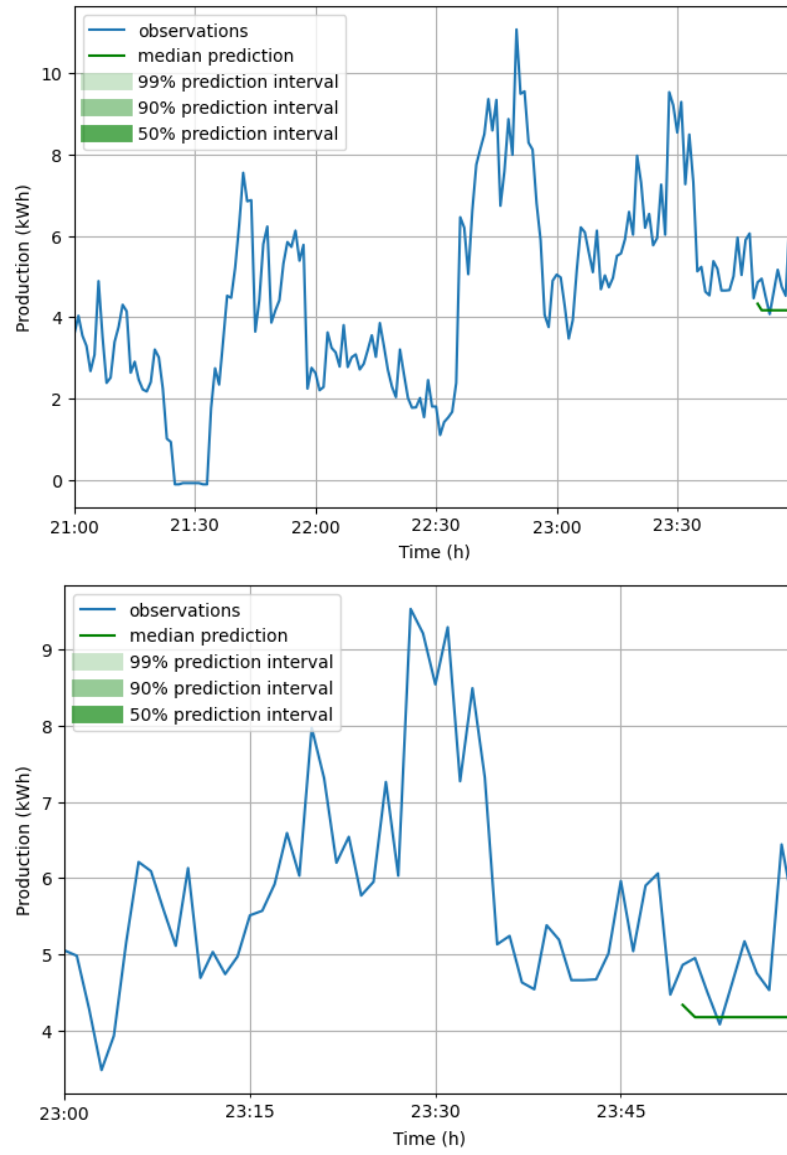


Figure 3.27: Forecast result of Wavenet model at 3 hours and 1 hour scale ($num_residue = 24$, $n_skip = 32$, Default hyperparameter values, Config A)

3.4.9 NBEATS

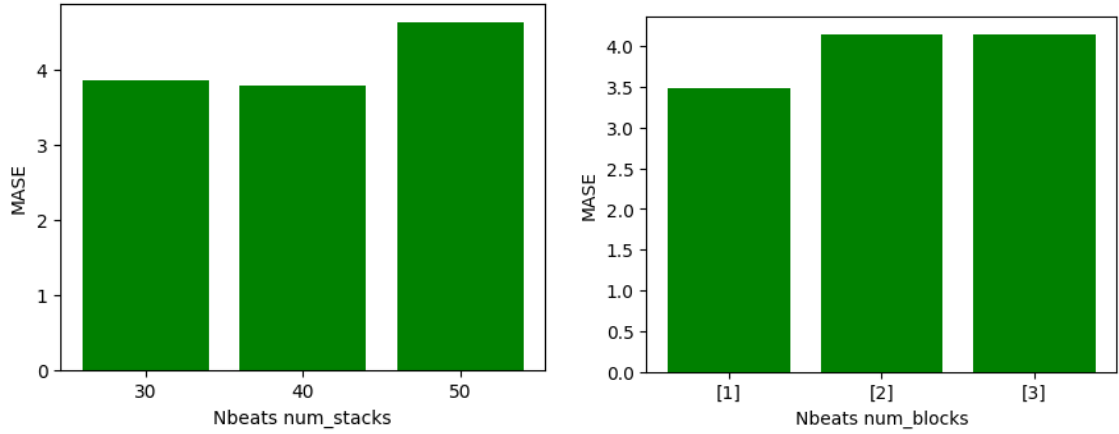


Figure 3.28: Comparison of different *num_stacks* and *num_blocks* values for *NBEATS* model (Default hyperparameter values, Config A)

Nbeats model is an implemented model described in section 2.7.11. Its own tunable hyperparameters are the number of stacks the NBEATS network should contains (*num_stacks*) and the number of blocks per stack (*num_blocks*). The default values in GluonTS are 30 for *num_stacks* and [1] for *num_blocks*. Results shows that the optimal values are 40 for *num_stacks* and [1] for *num_blocks*. Plots results, illustrating the model predictions, are presented in figure 3.29.

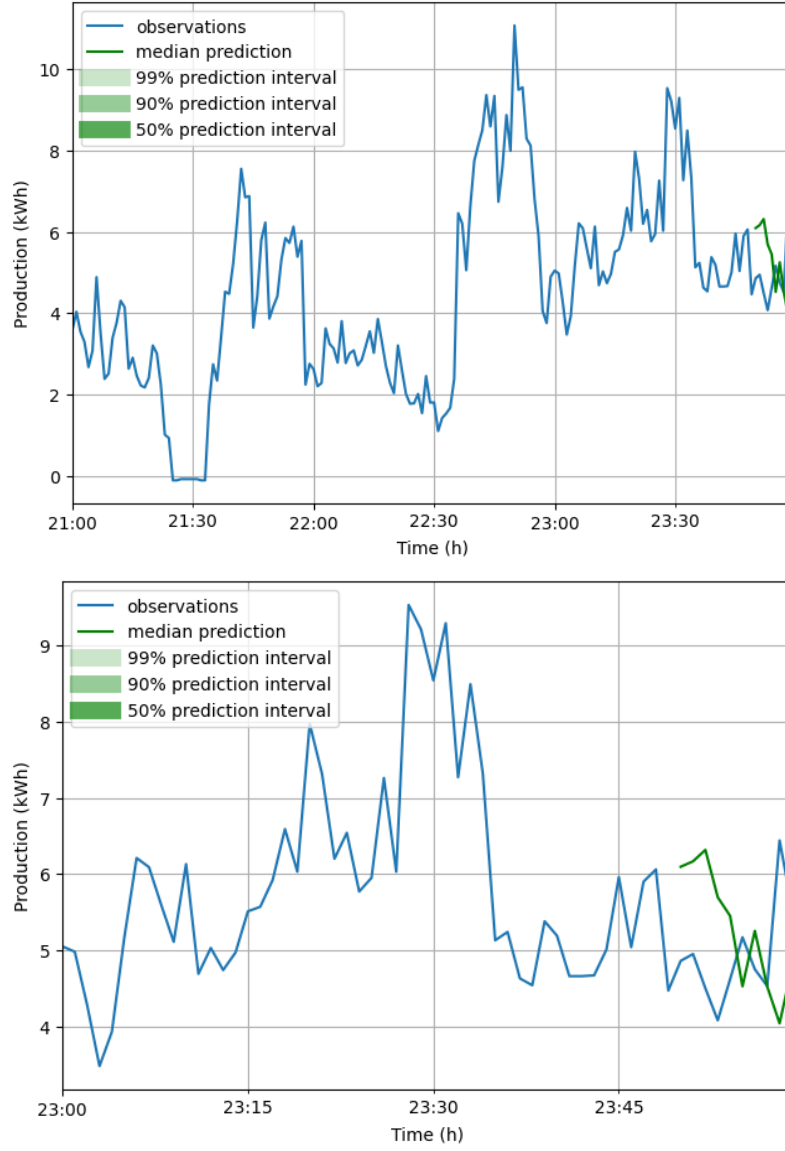


Figure 3.29: Forecast result of NBEATS model at 3 hours and 1 hour scale ($num_stacks = 40$ and $num_blocks = [1]$, Default hyperparameter values, Config A)

3.4.10 Gaussian Process

GaussianProcess model is an implemented model described in section 2.7.5. It has no own tunable parameters. Plots results, illustrating the model predictions, are presented in figure 3.30.

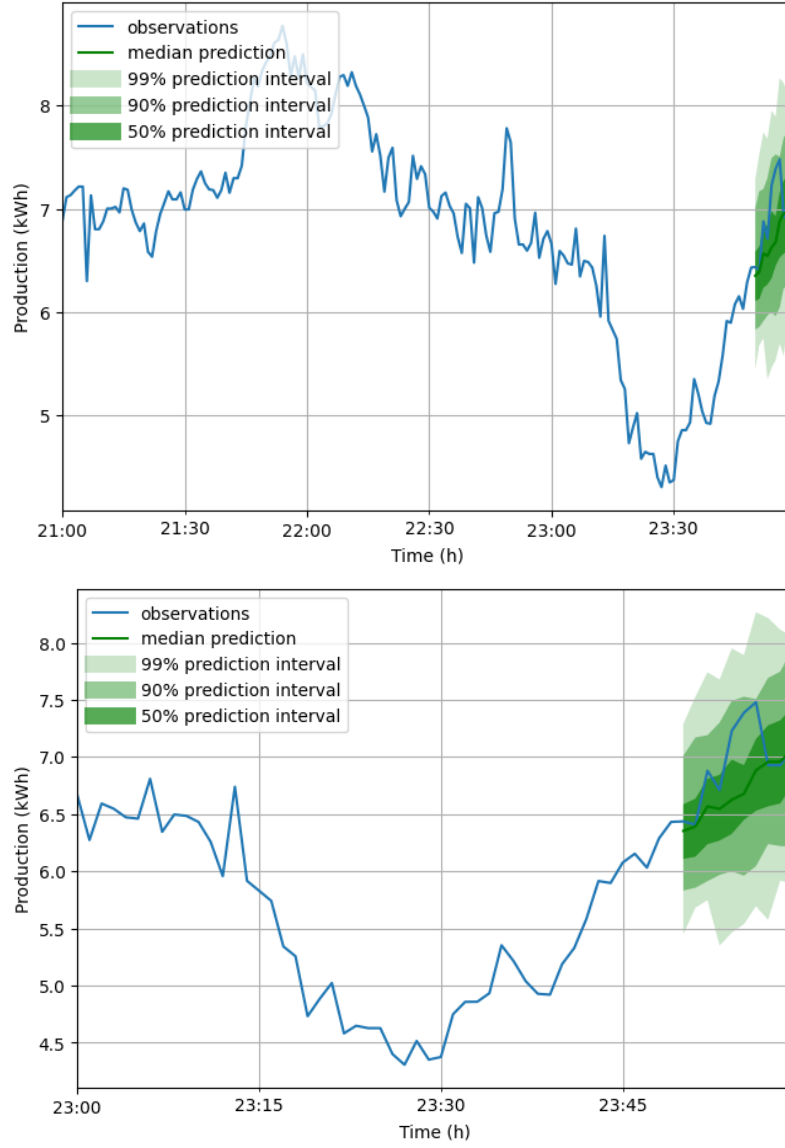


Figure 3.30: Forecast result of *GaussianProcess* model at 3 hours and 1 hour scale (Default hyperparameter values, Config A)

3.4.11 NPTS

NPTS model is an implemented model described in section 2.7.6. It doesn't have any own tunable parameters.

Plots results, illustrating the model predictions, are presented in figure 3.31.

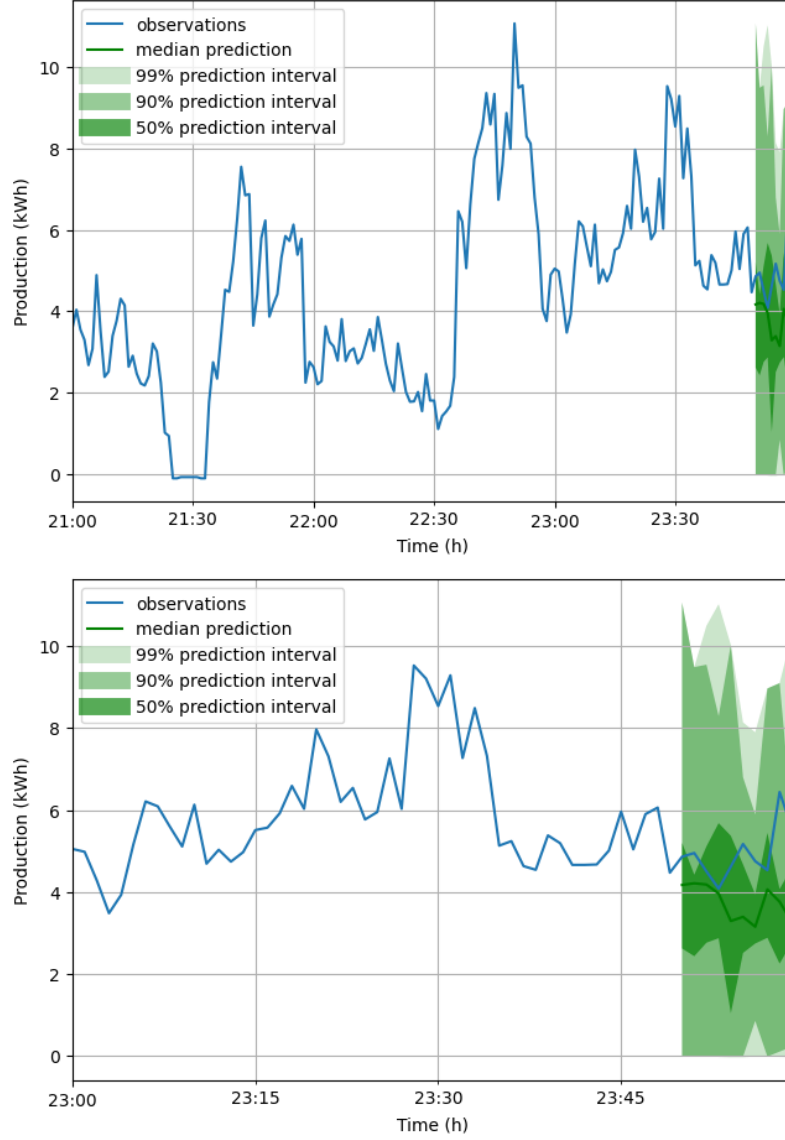


Figure 3.31: Forecast result of NPTS model at 3 hours and 1 hour scale (Config A)

3.4.12 ETS

ETS model is an model that does not belongs to the deep learning domain that has been mentionned in section 1.1. It will be used in the comparison of all models to compare all the deep learning solutions to a “classical” forecasting solution.

Plots results, illustrating the model predictions, are presented in figure 3.31.

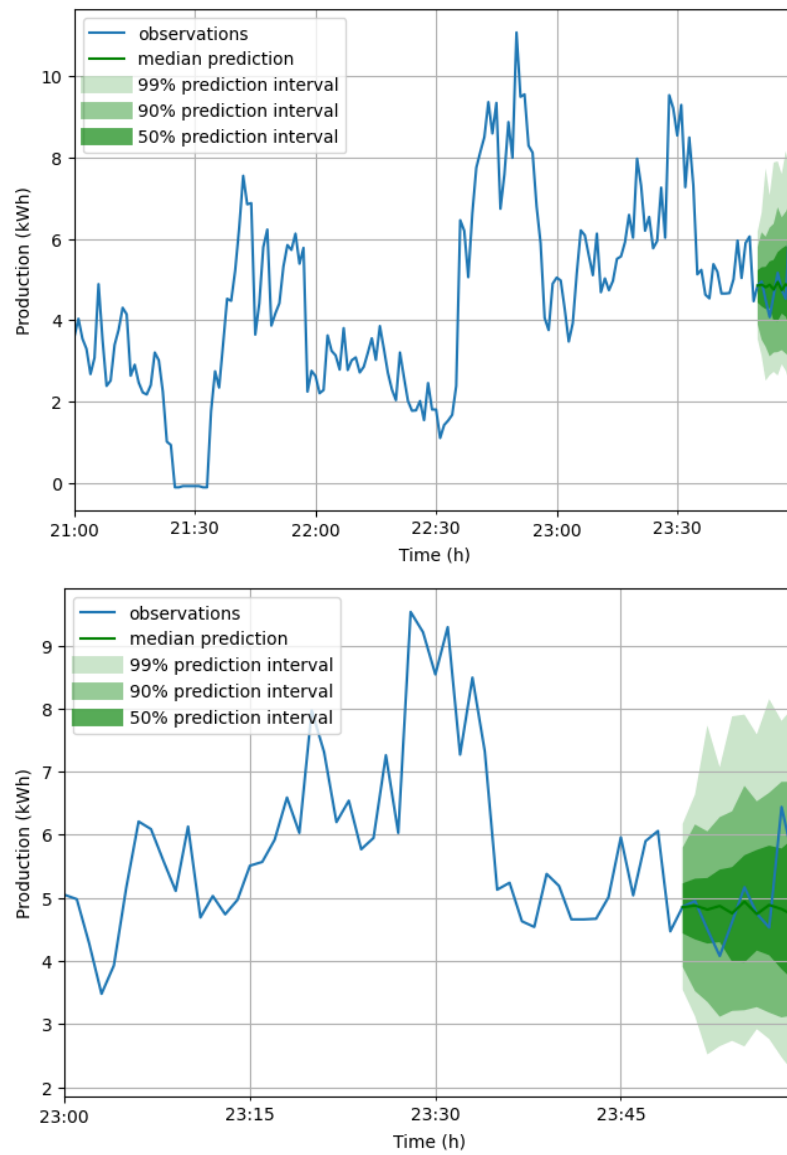


Figure 3.32: Forecast result of ETS model at 3 hours and 1 hour scale (Config A)

3.5 Comparison of all tuned models

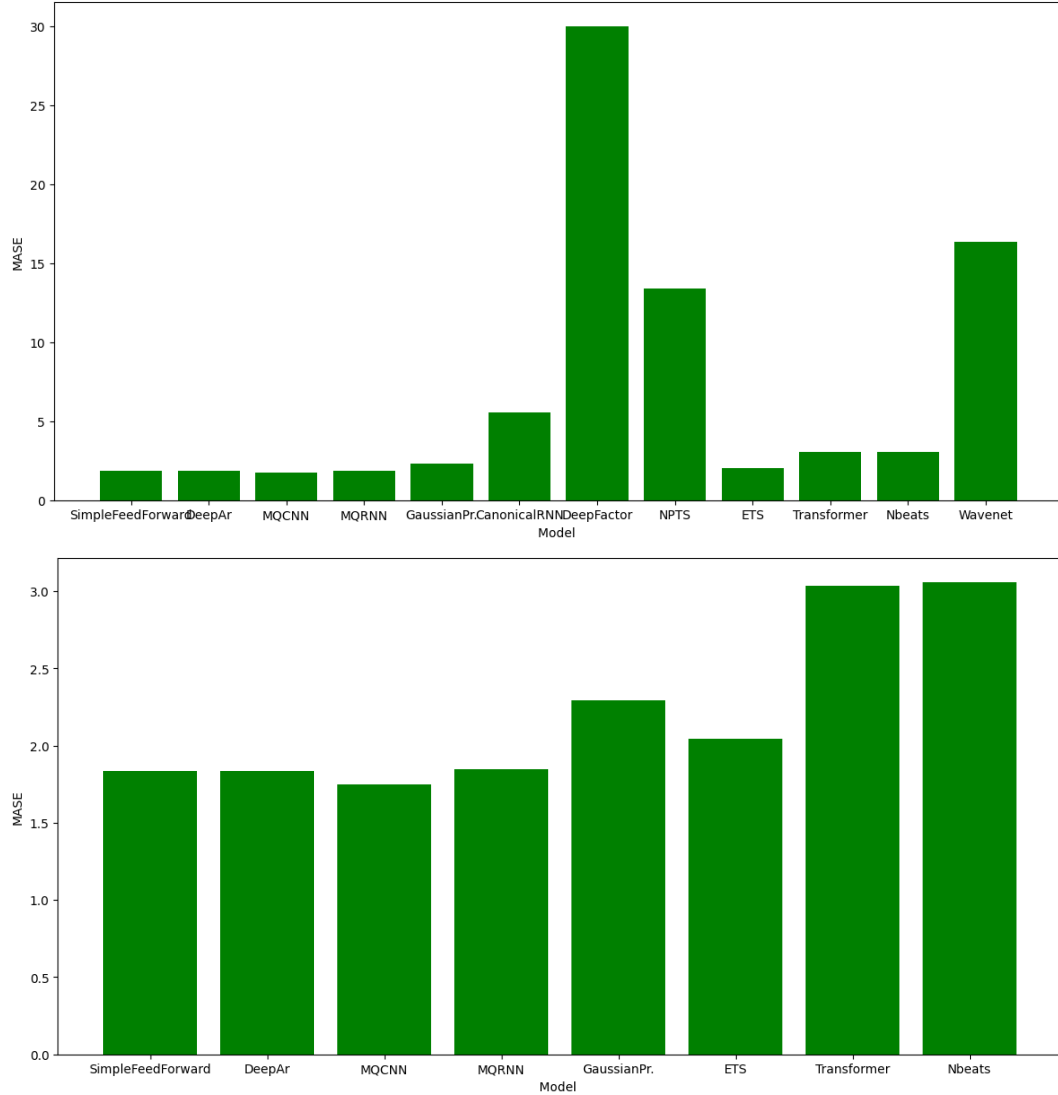


Figure 3.33: Comparison of different models (Default hyperparameter values, Config A). First figure represents all the models, the second represents only the 8 best models in terms of *MASE*

A first observation is that there are very large differences in terms of *MASE* metric value the different between models. We can divide the 12 presented models between four groups.

DeepFactor and *NPTS* provides very bad predictions, with *MASE* superior to 10. There are clearly solutions to avoid in this context. *Wavenet* and *CanonicalRNN*, if they performed better than the previous ones, are also considered as unsatisfactory, with *MASE* between 5 and 10. *Transformer* and *Nbeats* provides interesting results, but are clearly perfectible, with *MASE* between 2.5 and 5. The last group is composed of six models, *SimpleFeedForward*, *MQCNN*, *MQRNN*, *DeepAR*,

GaussianProcess and *ETS*. They results in MASE, between 1.7 and 2.5, are tight. Nowadays, we can classify them, and obtain what can be considered as the better model between the 11 models compared, with as metric *MASE* : *MQCNN*, followed by *MQRNN*, *DeepAr*, *SimpleFeedForward* and *GaussianProcess*.

3.6 Time global comparison

	Models			
	SimpleFeedForward	CanonicalRNN	DeepAr	DeepFactor
Epoch time (s)	1.487	9.785	2.253	1.764
Number of epochs	120	20	120	20
Total time (s)	178.44	195.70	270.36	35.28

Table 3.2: Comparison of different models training times (Default hyperparamter values, Config A)

	Models				
	MQCNN	MQRNN	Transformer	NBEATS	Wavenet
Epoch time (s)	2.059	3.016	2.398	6.007	14.544
Number of epochs	50	80	100	120	120
Total time (s)	102.95	241.28	239.8	720.84	1745.28

Table 3.3: Comparison of different models training times (Default hyperparamter values, Config A)

Another element of comparison is the training time of the different models. If the majority of the models have acceptable total training time (between 30 seconds and 4 minutes), the training time of NBEATS (12 minutes) and Wavenet (29 minutes) is not negligible.

Chapter 4

Using another metrics to compare forecasting models

4.1 Discussion about the purpose of the default metrics and loss considering the goal

Now that the comparison of the models as been done with as goal the minimization of the difference between predicted distribution and real distribution, as defined in sections concerning the loss (2.4) and the metrics (2.5), we can interrogate the pertinence of this "per default" goal in the particular context where the probabilistic forecasting models must be used.

As it as been introduced in section 1.1, the forecasting models usage we are interested in is the inclusion in a stochastic decision process, where we need to forecast the future generator production to anticipate risk of congestion. The stochastic decision process, is, more than everything, interested in a forecasting process that gives a precise estimation of what we could call the "security" quantile value, for each time step of the prediction window, i.e of the value for which there is a "very low" probability that the production at this time step outnumber this value (in this context we consider a probability of 1 %). That value will be used to determine the generator production capacity. For example if the security quantile value forecasted is superior to the maximum acceptable value, above which the generator encounter risks, it means that the generator capacity must be voluntarily restrained immediately.

In other terms, considering that the model output is a probability distribution, the goal fixed is not to minimize the difference (in any ways to define it) between the predicted and observed time series but to minimize the difference between the predicted "security quantile", i.e the $quantile(1 - e)$ of the predicted distribution, knowing that e is the probability of error considered as acceptable (1% in this context), and the real security quantile (the quantile of the real probability distribution of the source). A too low predicted security quantile signify that the risk of production oversupply is underestimated, leading to situations where the system will estimate that the network is safe wrongly. Inversely, a too high predicted security

quantile signify that the risk is overestimated, leading to situations where the production capacity will be restrained unnecessarily. This situation is nevertheless less problematic than the first one (as it will causes lack of optimization but no risk of generator damage). The minimization must be for all time steps of the prediction range.

This objective is not disconnected from the classical objective consisting in minimizing the difference between the predicted and observed probability distribution. If the predicted distribution is close to the observed distribution, the predicted security quantile tends to be close to the observed security quantile. Nowadays, we highlight the need to use another metric to focus the evaluation of the prediction on the security quantile specifically.

4.2 Introducing a new metric

A new metric function is essential to evaluate correctly the models, knowing the goal we want them to pursue (4.1). We propose here a metric that fulfills this need.

The metric role must be to indicates if the predicted quantile security value is over/under-estimated, for each time steps of the prediction interval and for each time series. The natural way of proceeding would be to compare the values of the predicted security quantile with the values of the observed security quantile, but this technique is not applicable considering that the observed security value is unknown. The only observed information at our disposition is the observed (point) value. The metric that we define must use only this information.

The proposed metric, *Coverage* uses an implemented metric, *coverage_q*. This metric is defined as, taking as argument N randomly drawn sample values of the predicted probability distribution $([x_1, \dots, x_N])$, one observed value y , q the quantile considered and *quantile_q* $([x_1, \dots, x_N])$ a function giving the quantile q value for the approximated distribution constructed on the samples $[x_1, \dots, x_N]$:

$$coverage_q(y, [x_1, \dots, x_N]) = \mathbb{I}(y < quantile_q([x_1, \dots, x_N])) \quad (4.1)$$

This metric indicated if the observed value is below the predicted distribution quantile q . The *Coverage* metric is defined as, for e is the probability of error considered as acceptable :

$$Coverage(y, [x_1, \dots, x_N], e) = coverage_{1-e}(y, [x_1, \dots, x_N]) - (1 - e) \quad (4.2)$$

The *Coverage* “Item” metric value corresponds to the average of *Coverage* for all time steps of the prediction interval for time series i . The *Coverage* “Aggregate” Metric value corresponds to the average of *Coverage* Item metrics for all time series i . The Aggregate Metric Coverage value must tends to 0 to achieve the goal. If $Coverage(x) > x$, the quantile value is too high (the window is too large) and if $Coverage(x) < x$ the quantile value is too tight.

This metric is not sufficient to evaluate the quality of a model prediction. The model could have a good *Coverage* value because the predicted distribution is very spread out, ensuring a positive value of *Coverage*, or because the predicted distribution is too tight but shifted up compare to the real distribution.

Classical metric as *MASE* could be used as second metric to indicates, in addition to the security quantile estimation quality, the whole distribution prediction accuracy.

4.3 Custom Loss

As a new metric function has been introduced to fulfill the needs implied by the goal being pursued (4.1), the loss function must be discussed as well. The loss function must take account if the predicted security quantile is over/under-estimated.

We could express a loss strictly equivalent to the metric Coverage described in section 4.2. It would be defined as the metric Coverage but with the probability of distribution itself and not the samples of this probability.

Problem is that the defined metric make sense when averaged to form “Item” and “Aggregates” metrics. The value of one Coverage metric evaluation is binary. We could also use a different, non-binary loss that penalize values bigger than $quantile(1 - e)$ This proposed loss is the following, for $\phi(x)$ is the output distribution and y the observed value :

$$L_{alt}(y, \phi(x)) = e^{y - quantile_{1-e}(\phi(x))} \quad (4.3)$$

The loss increases exponentially if y superior to the security quantile, and is remains low if not.

The only way to use other loss than the default loss for the pre-implemented models of GluonTS is to define custom models copying the originals at the exception of intern modifications to change loss function.

A major implementation problem issue arises. Between the different pre-implemented models, some can be copied and used as custom models with different loss function. It is the case of “*Simple*”, “*SimpleFeedForward*”, “*CanonicalRNN*”, *DeepAr* and *DeepFactor*. Some models that can be implemented in this way faces a problem. In the current version of GluonTS the use of a custom model implies that the hybridisation, important part of the implementation optimization, cannot be used, because of a running erro, resulting in very important training time. In particular *DeepAr* and *DeepFactor* model are hardly affected by the absence of hybridisation. The training time without hybridisation (presented in 3.6) is multiplied by a factor 15. Loss function will not be modified for these models, despite the fact that it is technically possible. Other models cannot change they loss because of the way they are implemented, because of how they evaluates the loss, or because they use another loss minimization organisation (*MQCNN* and *MQRNN*).

Cannot be able to modify the loss does not means that the models cannot be used to achieve the goal, as the base loss is connected to the new goal anyway, as explains in section 4.1.

In the models where the loss can be changed, to express the fact that we want to optimize the *Coverage* and the *MASE*, the implemented custom loss is a combination of the loss functions L_{default} and L_{alt}

$$\text{custom_loss}(x, \phi) = L_{\text{default}}(x, \phi) + \alpha * L_{\text{alt}}(x, \phi) \quad (4.4)$$

With α an hyper parameter. Its value is a subject of a study in section 5.2.7. We will see the results for different weighting between the different losses.

Chapter 5

Model comparison using quantile-based metrics

5.1 Testing protocol definition

Now that the metric functions and loss functions have been discussed considering the goal that we pursued, the different models that have been presented could be compared (after discuss their hyper-parameter values), using as principal element of comparison the introduced metric *Coverage*, as defined in section 4.2, and with *MASE* metric as second element of comparison. As it was discussed in section 4.3, the models that could compute and use the alt loss use the custom loss, a weighted sum of the default and alt loss. The others use their default loss.

Before the comparison between different models, we need to compare the impact of the values of the different hyperparameters of the problem. The hyperparameters of the problem are :

- The context length, defined in section 1.4.
- The size of time series
- The learning rate of the model training
- The number of epochs of the training
- The output distribution, defined in section 2.6
- The individual hyperparameters of the different models
- For the models using custom loss, the value of α , defined in section 4.3

The default input data configuration is A (see 2.3). Results for different input data configuration are not systematically showed because experimentation that has been performed (but are not all presented here) show that the configuration A/B does not significantly influences the choice of hyperparameters. Some results are presented : 3.3.5 and 3.3.3 and shows similar results for the two configurations.

Exhaustive presentation of results for the two configurations would be redundant and time-consuming.

Concretely, the following subsections of the section 3.3 are organised as follows :

- We select one hyperparameter, commons to different models, to tune.
- The model is run for different values of this hyperparameter, with other hyperparameters fixed to default *GluonTS* values if the hyperparameter as not been already tuned or fixed to values that has been observed as optimal during previous hyperparameter tuning. All these hyperparameters values are described as “default” in the following subsections.
- Results are firstly presented as histogram, with on the abscissa the hyperparameter value and on the ordinate the *Coverage* metric value obtains by the trained model.
- Results are secondly presented as a scatter diagram, with on the abscissa the *MASE* metric obtains by the trained model and on the ordinate the *Coverage* metric value obtains by the trained model.
- We deduce the optimal value of the parameter from these results, knowing that the goal is the minimization of the difference between 0 and *Coverage* value, and the minimization of the *MASE* value. Coverage optimization is prioritized, but if between two possible values of parameters the *Coverage* value is similar, the value of *MASE* will be considered. We estimate that two values of *Coverage* as similar if there difference is at or below the scale of 0.001.

Concretely, the following subsections of the section 3.4 are organised as follows :

- We select one model between the implemented models presented in section 2.7.
- For each of it tunable hyperparameter, the model is run for a range of this hyperparameter values, with other hyperparameters fixed to default *GluonTS* values if the hyperparameter as not been already tuned or fixed to values that has been observed as optimal during previous hyperparameter tuning. All these hyperparameters values are described as “default” in the following subsections. Some models does not have tunable hyperparameter.
- Results are firstly presented as histogram, with on the abscissa the hyperparameter value and on the ordinate the *Coverage* metric value obtains by the trained model.
- Results are secondly presented as a scatter diagram, with on the abscissa the *MASE* metric obtains by the trained model and on the ordinate the *Coverage* metric value obtains by the trained model.
- We deduce the optimal value of the parameter from these results using the same method as in section 3.3.

- Plot results are presented for some models where, as some default hyperparameters values differs from default values in the previous comparison in chapter 3 , results are significantly different from the previously presented plots.

Finally we had the section presenting the comparison of all implemented models, with all their hyperparameter tuned. Results are firstly presented as an histogram with on the abscissa the different models tested and on the ordinate the *Coverage* metric value obtains by the trained model. Results are secondly presented as a scatter diagram, with on the abscissa the *MASE* metric obtains by the trained model and on the ordinate the *Coverage* metric value obtains by the trained model.

5.2 Global Hyperparameters comparison

5.2.1 Context length

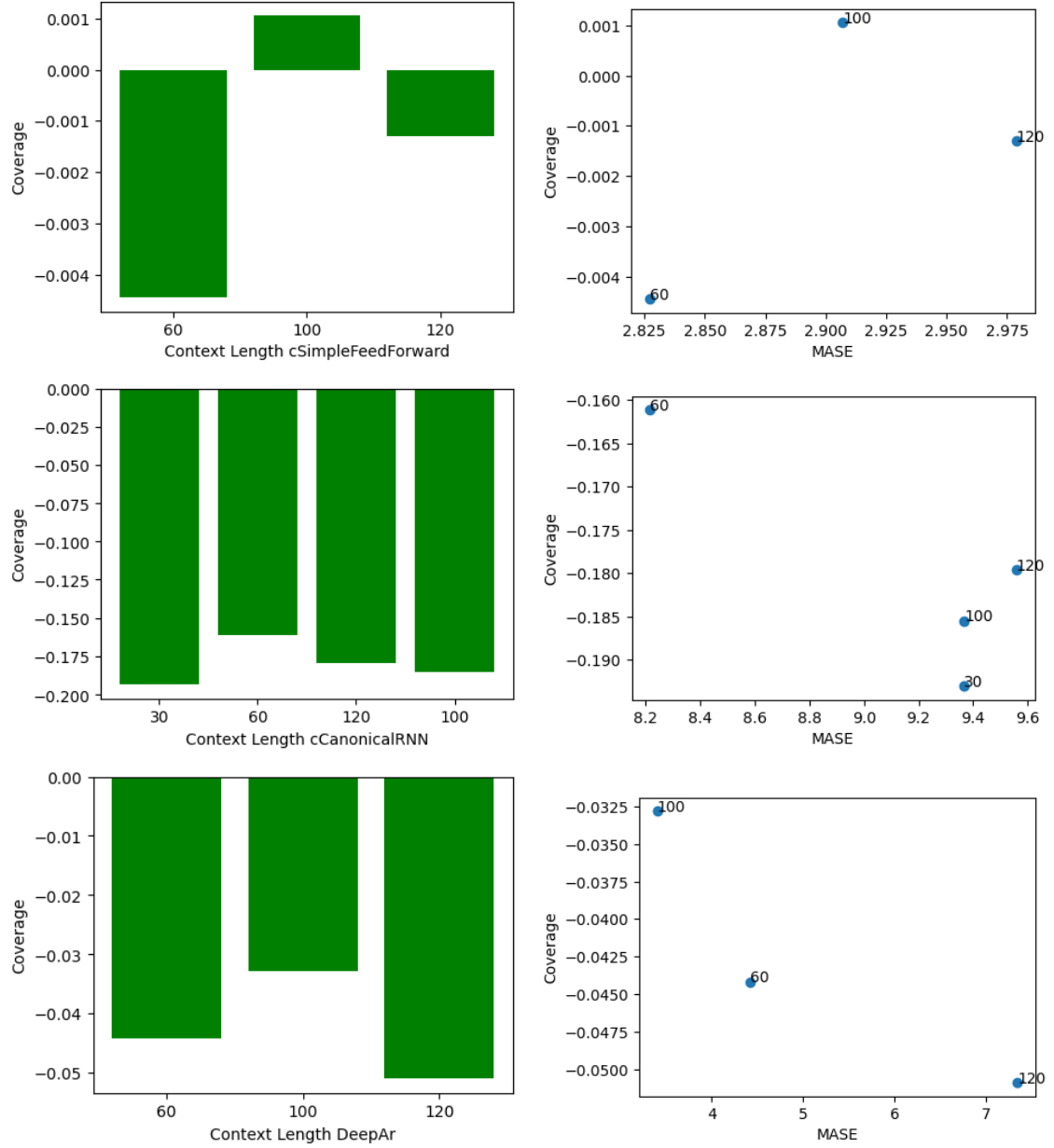


Figure 5.1: Comparison between different context length for different models (Models: *SimpleFeedForward*, *CanonicalRNN*, *DeepAr* (Default hyperparameter values), Config A)

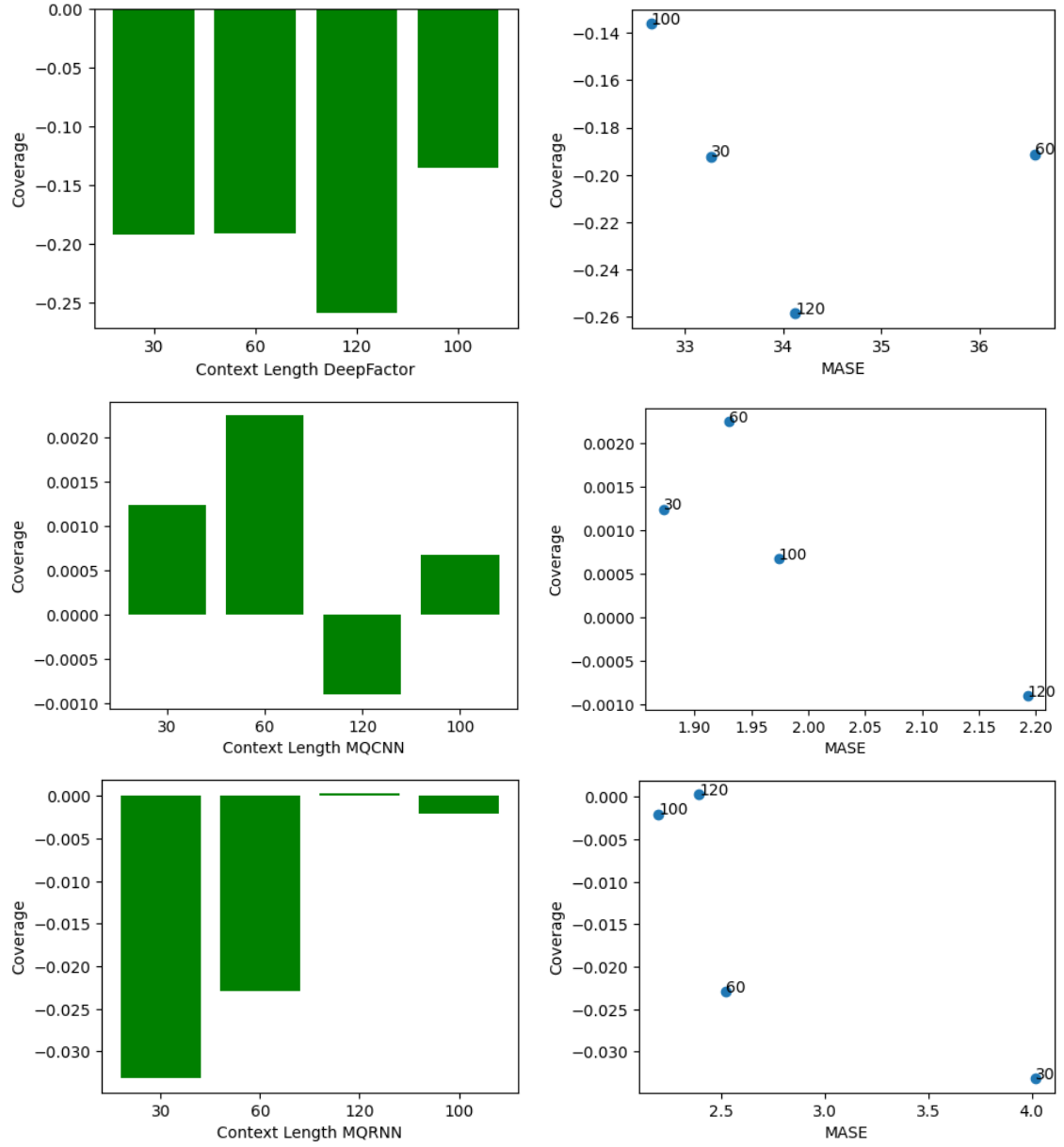


Figure 5.2: Comparison between different context length for different models (Models: *DeepFactor*, *MQCNN*, *MQRNN* (Default hyperparameter values), Config A)

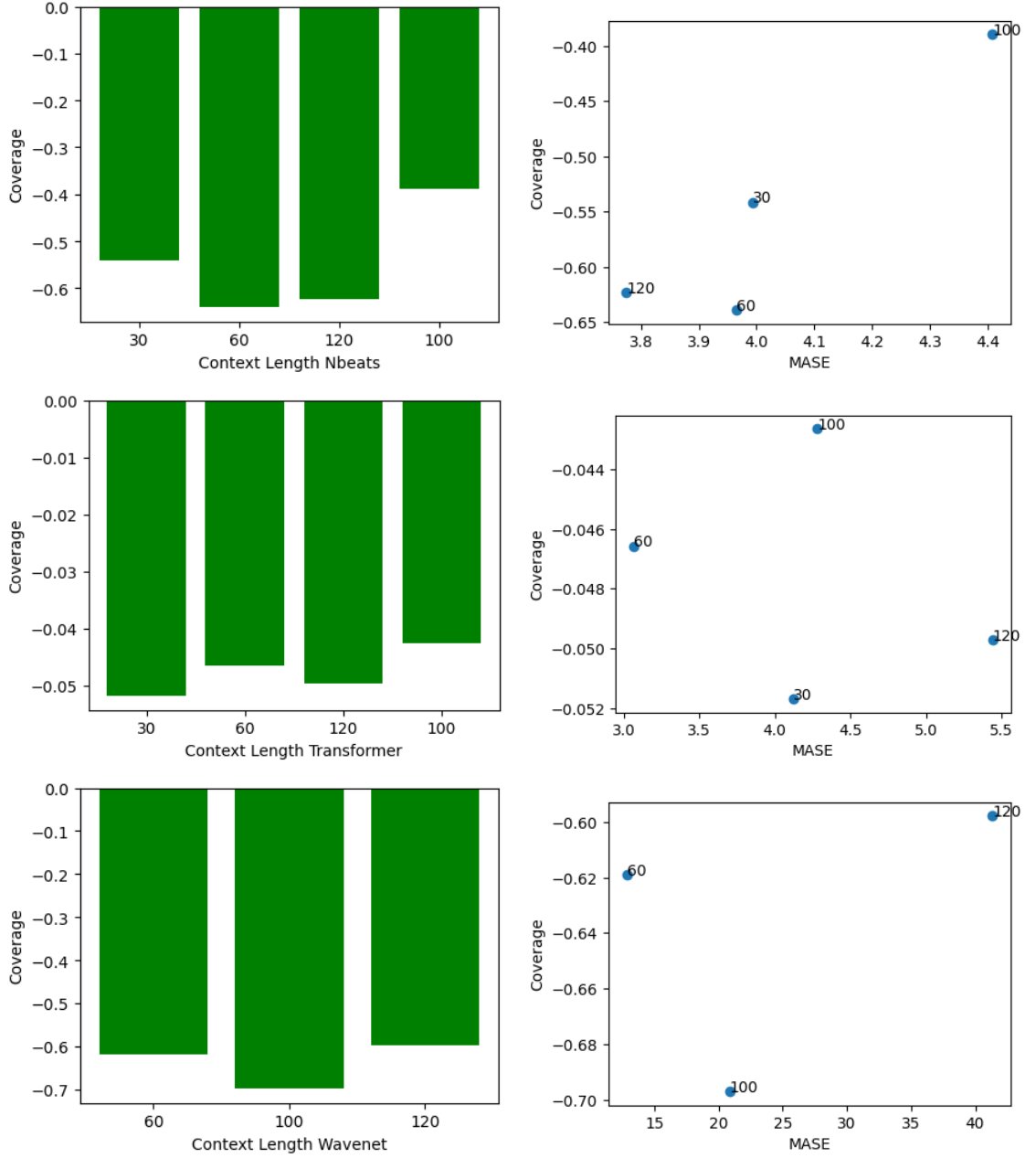


Figure 5.3: Comparison between different context lengths for different models (Models: *NBEATS*, *Transformer*, *Wavenet* (Default hyperparameter values), Config A)

We compare, as in section 3.3.1, different values of context length hyperparameter for different models.

Optimal values are the same as in the referred section, as the best value of Coverage (or a value very close to the best) is obtained for the value of the parameter that is also best for *MASE*, for all models excepts *NBEATS* where the optimal value considering *Coverage* is 100. These values are considered as default values in the following comparison sections.

5.2.2 Time series size

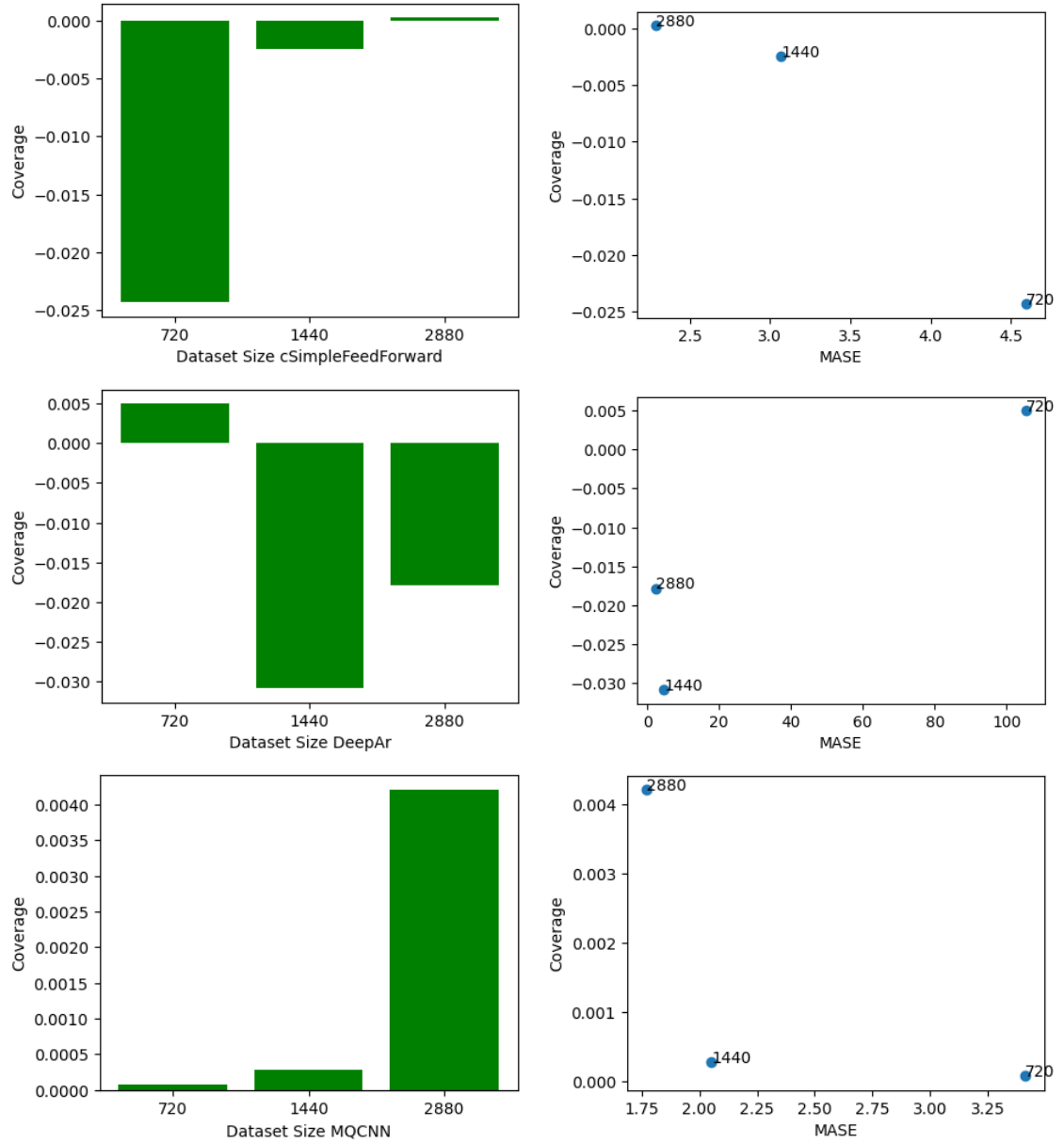


Figure 5.4: Comparison between different time series size for different models (Models: *SimpleFeedForward*, *DeepAr*, *MQCNN* (Default parameters values), Config A)

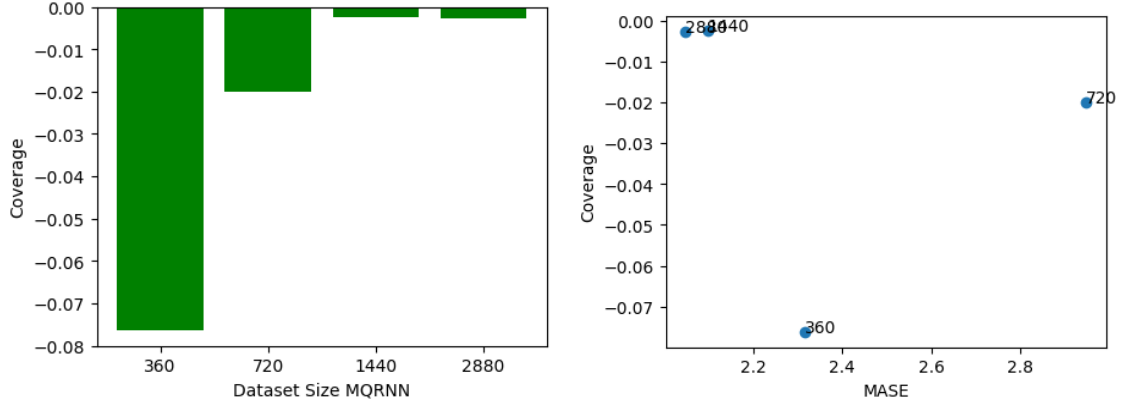


Figure 5.5: Comparison between different time series size (Model: *MQRNN* (Default parameters values), Config A)

We compare, as in section 3.3.2, different values of time series size hyperparameter for different models.

Optimal values are the same as in the referred section, as the best value of Coverage (or a value very close to the best) is obtained for the value of the parameter that is also best for *MASE*, for all models. These values are considered as default values in the following comparison sections.

5.2.3 Learning rate

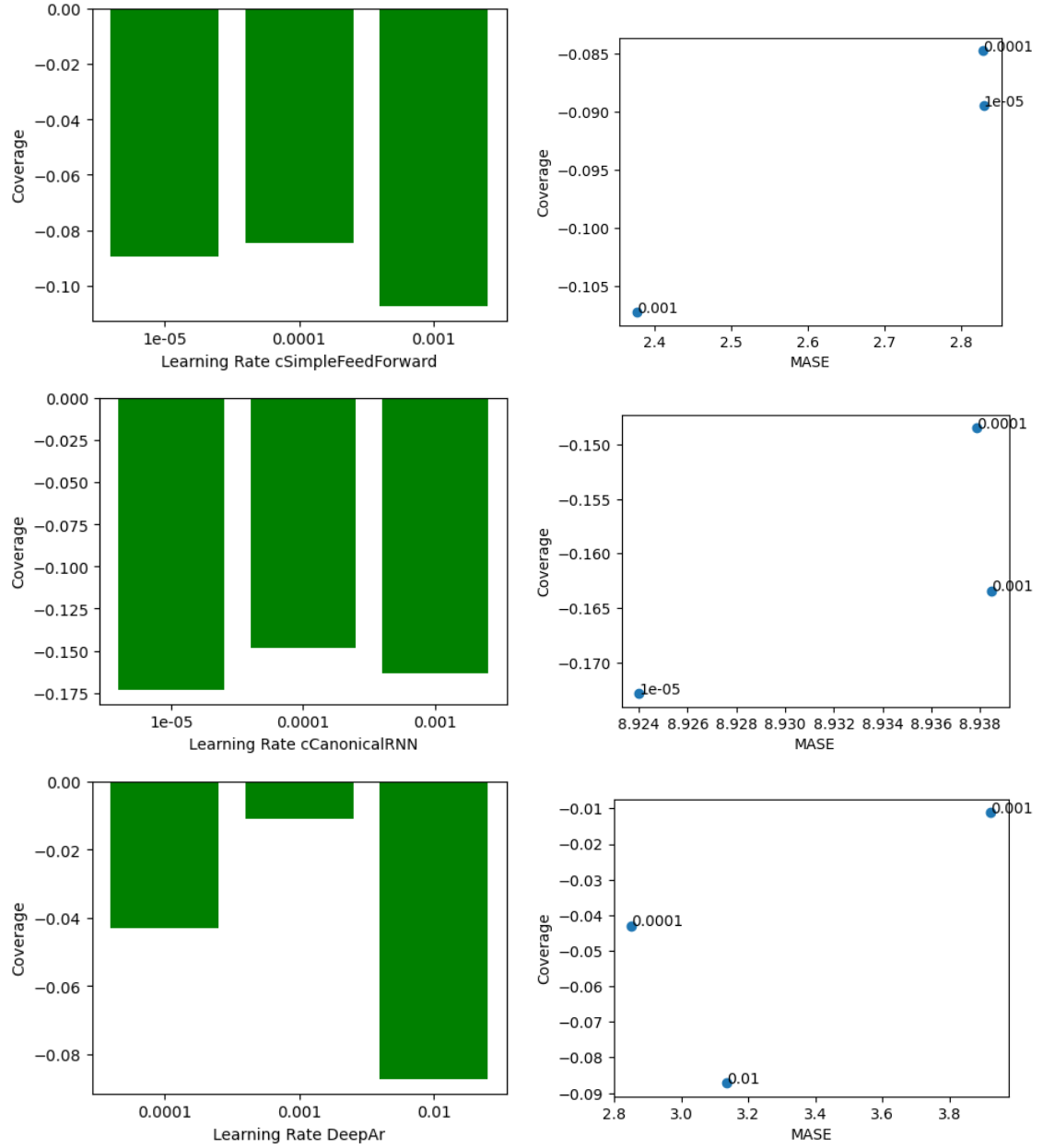


Figure 5.6: Comparison between different learning rates for different models (Models: *SimpleFeedForward*, *CanonicalRNN*, *DeepAr* (Default hyperparameter values), Config A)

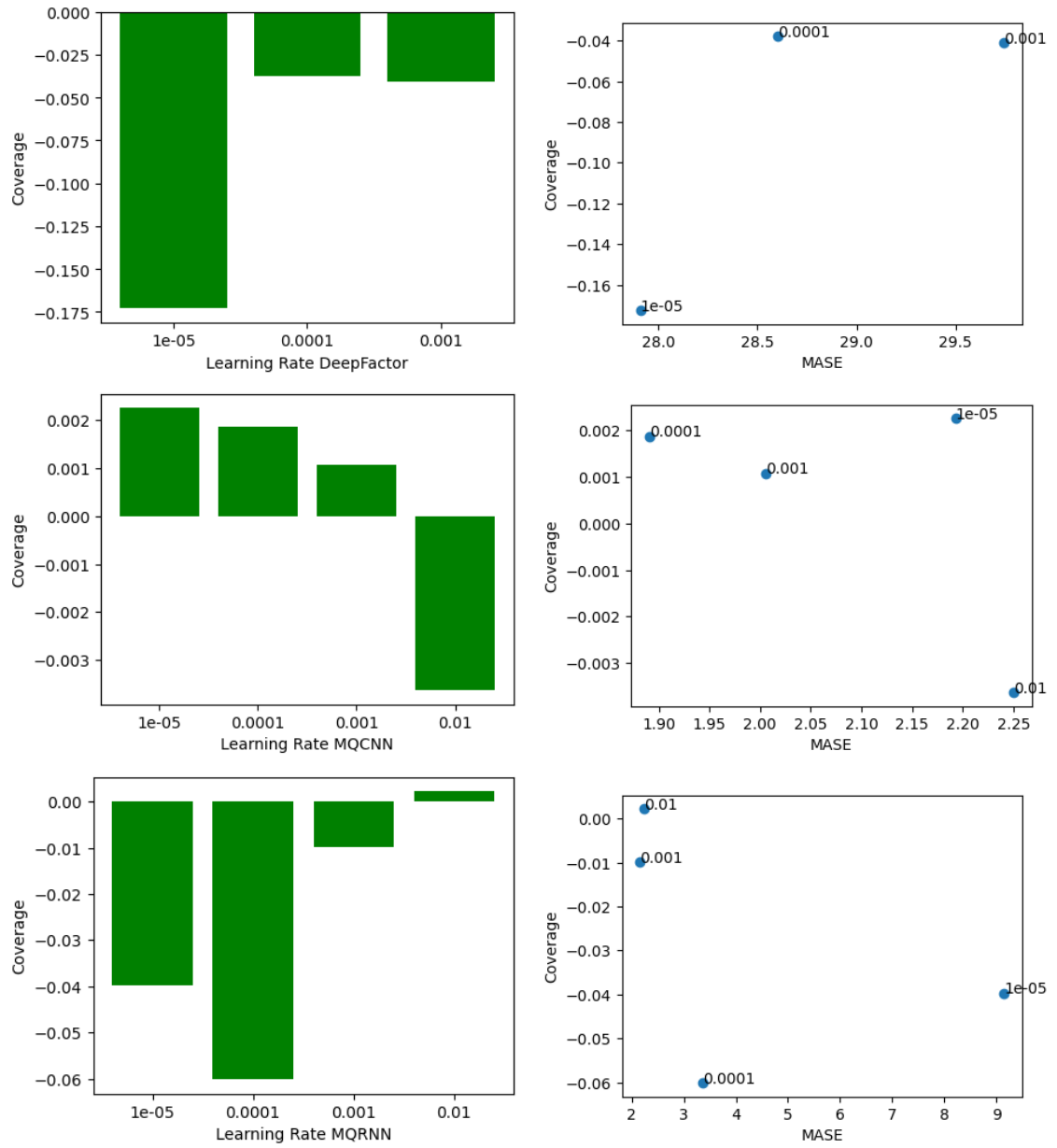


Figure 5.7: Comparison between different learning rates for different models (Models: *DeepFactor*, *MQCNN*, *MQRNN* (Default hyperparameter values), Config A)

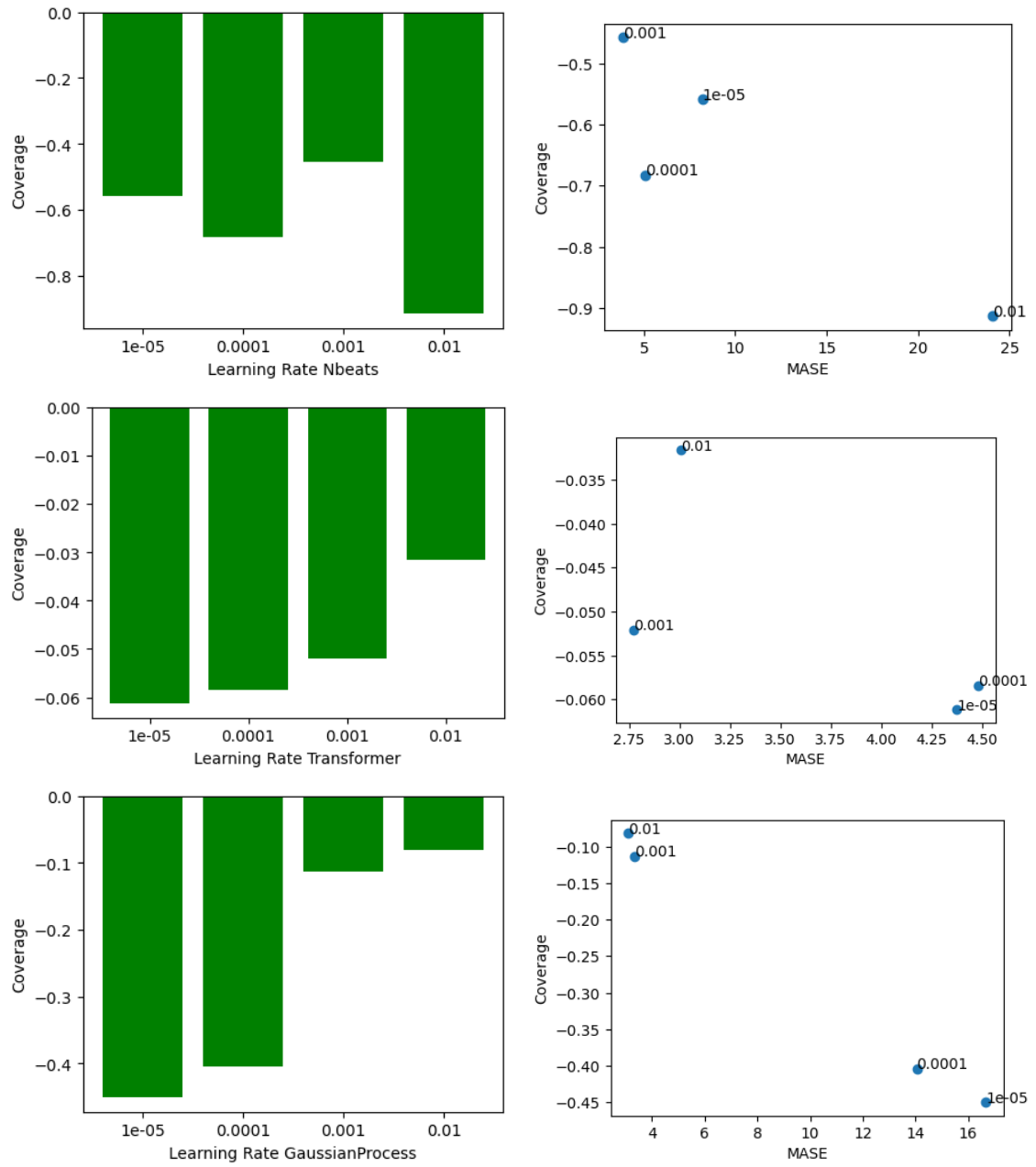


Figure 5.8: Comparison between different learning rates for different models (Models: *NBEATS*, *Transformer*, *GaussianProcess* (Default hyperparameter values), Config A)

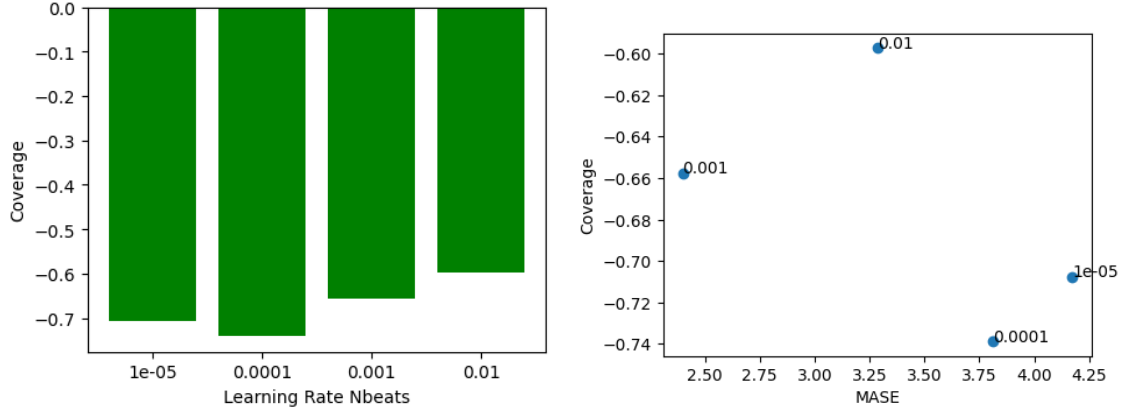


Figure 5.9: Comparison between different learning rates (Model: *NBEATS* (Default hyperparameter values), Config B)

We compare, as in section 3.3.3, different values of learning rate hyperparameter for different models.

These results allows us to deduce the optimal value of learning rate for each models. Models *DeepAr* , *Nbeats*, *Wavenet* optimal value is 10^{-3} , *MQRNN*, *GaussianProcess*, *Transformer* is 10^{-2} , *SimpleFeedForward*, *CanonicalRNN*, *MQCNN*, *DeepFactor* is 10^{-4} . These values are considered as default values in the following comparison sections.

5.2.4 Epochs

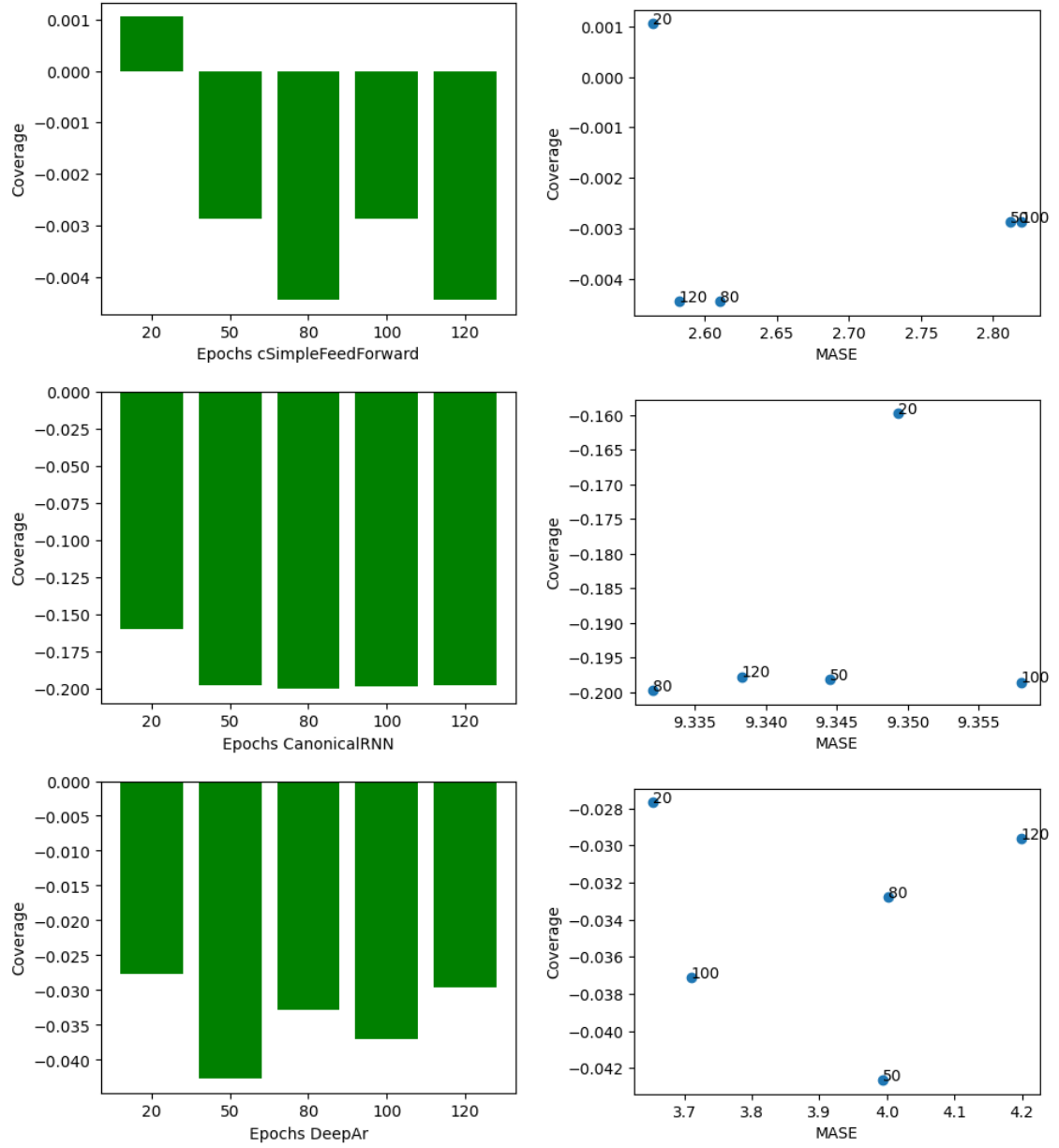


Figure 5.10: Comparison between different epochs values for different models (Models: *SimpleFeedForward*, *CanonicalRNN*, *DeepAr* (Default hyperparameter values), Config A)

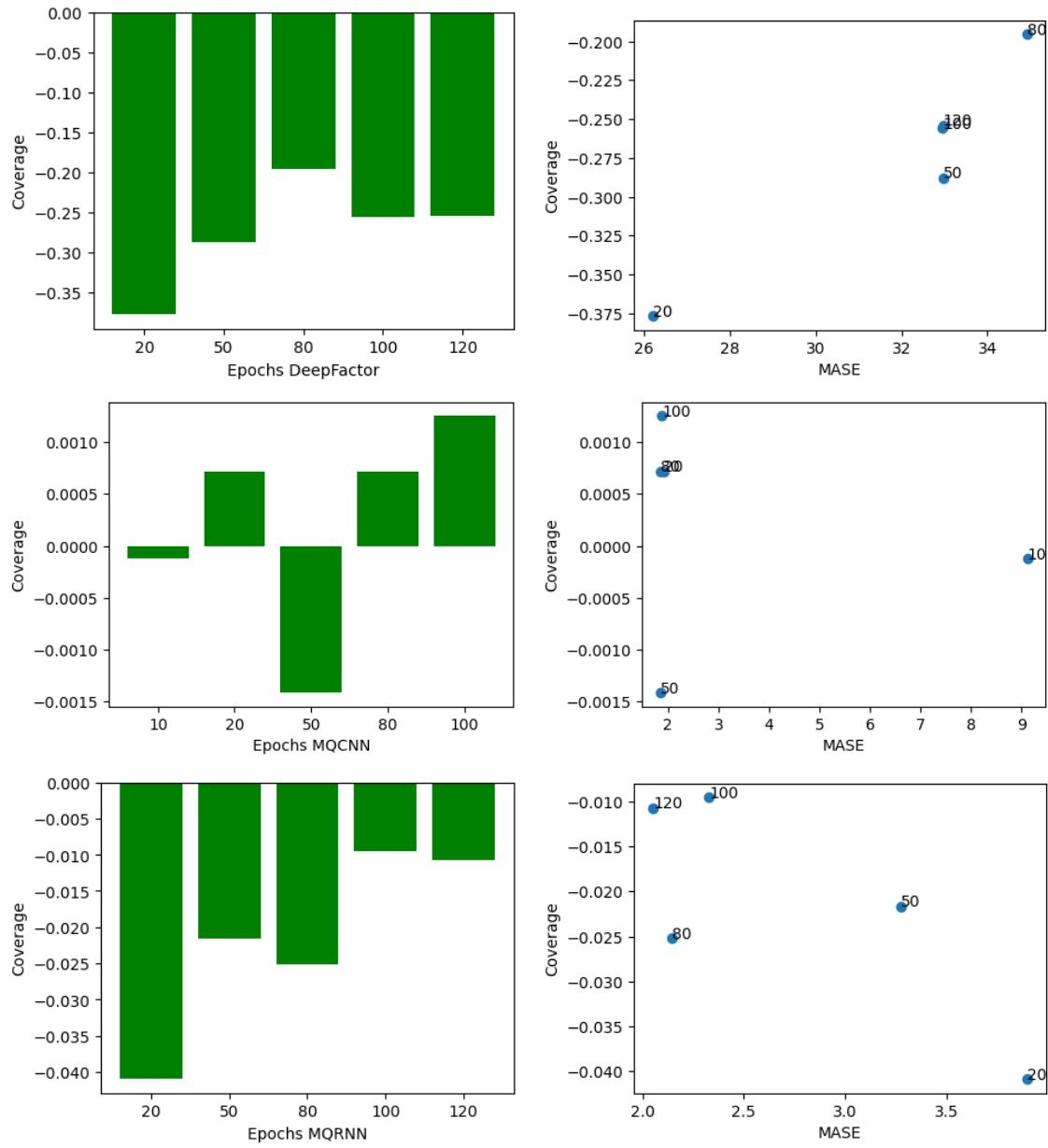


Figure 5.11: Comparison between different epochs values for different models (Models: *DeepFactor*, *MQCNN*, *MQRNN* (Default hyperparameter values), Config A)

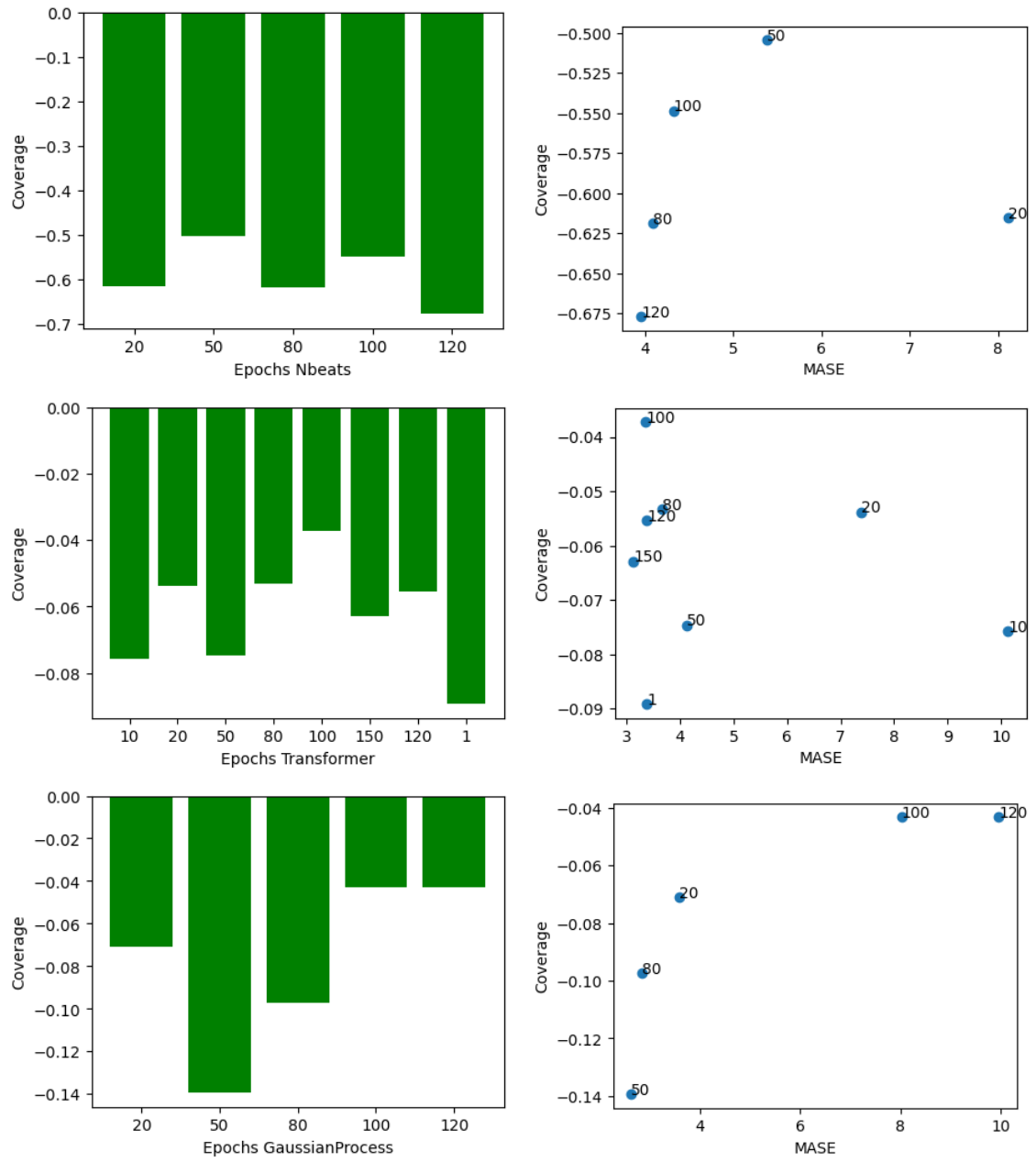


Figure 5.12: Comparison between different epochs values for different models (Models: *NBEATS*, *Transformer*, *GaussianProcess* (Default hyperparameter values), Config A)

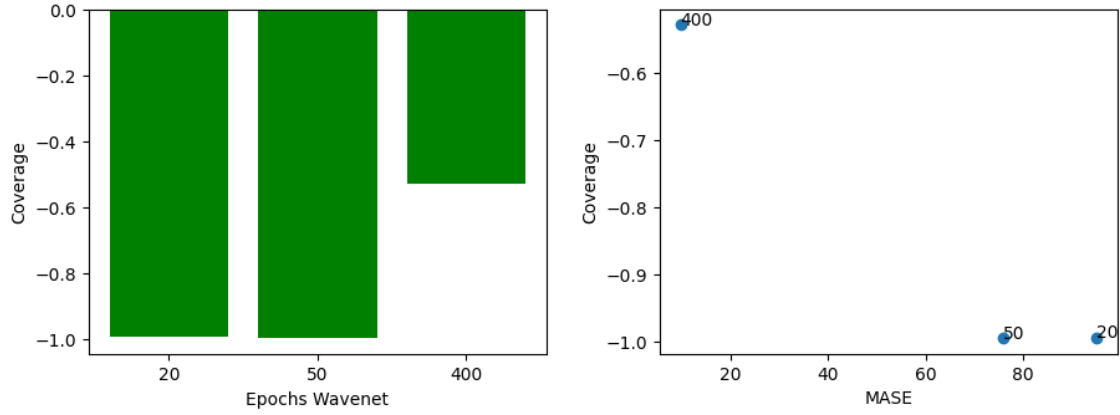


Figure 5.13: Comparison between different epochs values for different models (Models: *Wavenet* (Default hyperparameter values), Config A)

We compare, in continuity with 3.3.4, different values of epochs hyperparameter.

These results allows us to deduce the optimal value of epochs for each models. Models *CanonicalRNN*, *SimpleFeedForward*, *DeepAr* optimal value is 20, *NBEATS* is 50, *Deep Factor* is 80, *Transformer*, *MQCNN*, *GaussianProcess* is 100, *Wavenet* and *MQRNN* is 120. *Wavenet* is 400. These values are considered as default values in the following comparison sections.

5.2.5 Output Distribution

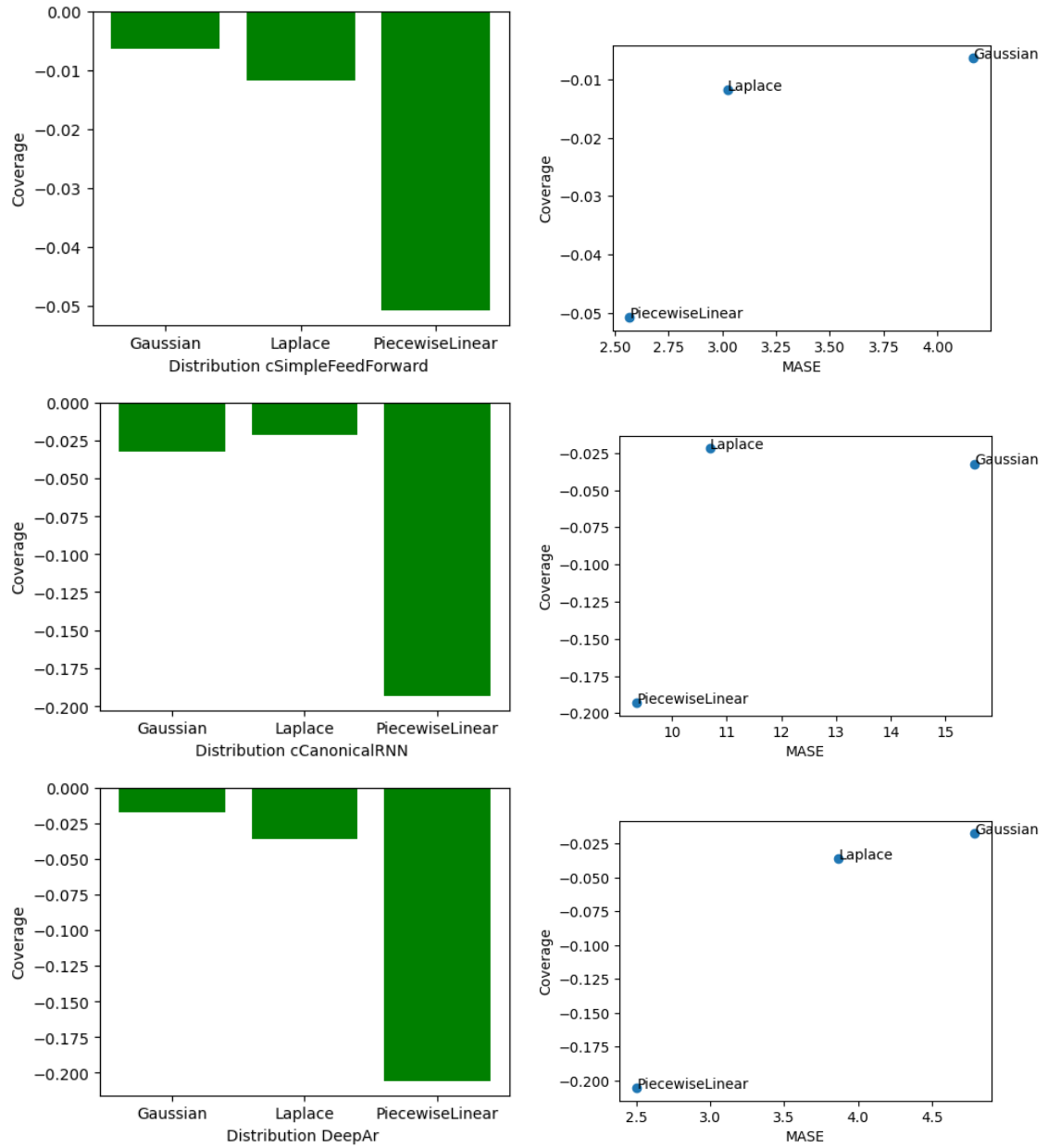


Figure 5.14: Comparison between different output distributions for different models (Models: *SimpleFeedForward*, *CanonicalRNN*, *DeepAr* (Default hyperparameter values), Config A)

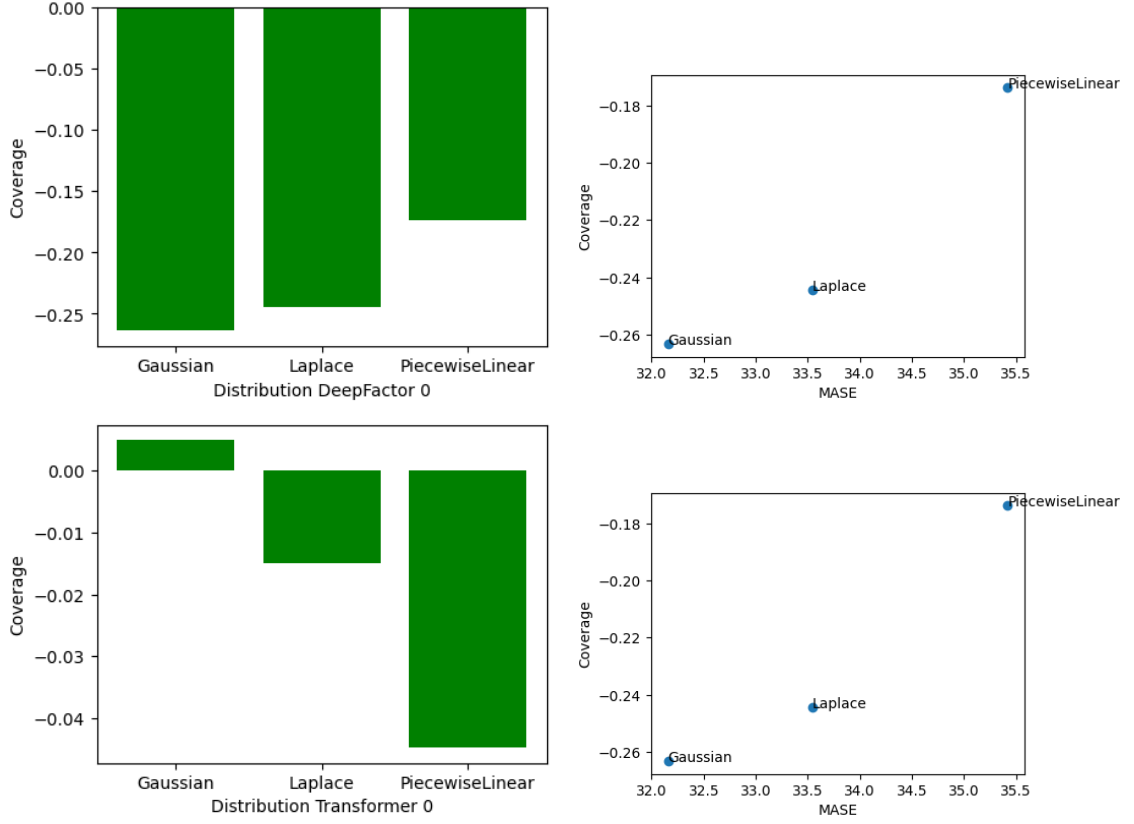


Figure 5.15: Comparison between different output distributions for different models (Models: *DeepFactor*, *Transformer* (Default hyperparameter values), Config A)

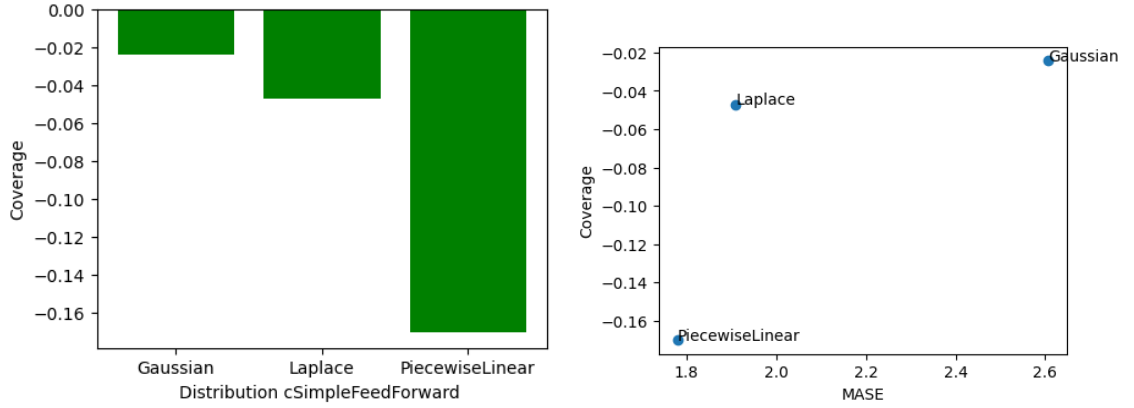


Figure 5.16: Comparison between different output distributions (Model: *SimpleFeedForward* (Default hyperparameter values), Config B)

We compare, in continuity with 3.3.5, different values of output distribution type hyperparameter.

These results allows us to deduce the optimal output distribution for each models. Models *SimpleFeedForward*, *CanonicalRNN*, *DeepAr* optimal choice is *Laplace*,

Transformer and *DeepFactor* is *PiecewiseLinear*. These values are considered as default values in the following comparison sections.

5.2.6 Number of Pieces of PiecewiseLinear distribution

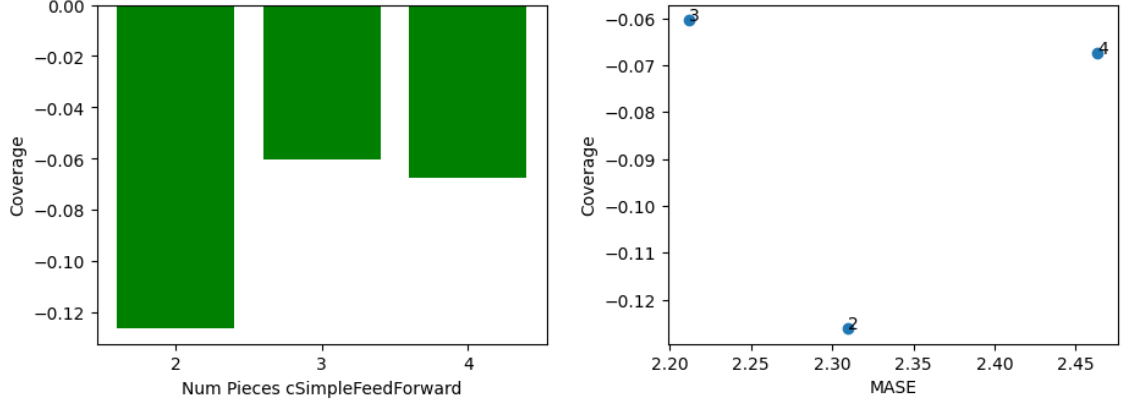


Figure 5.17: Comparison between different number of pieces for PiecewiseLinear distribution (Model: *SimpleFeedForward* (Default parameter values), Config : A)

We compare, in continuity with 3.3.6, different values of number of *PiecewiseLinear* output distribution pieces hyperparameter.

The optimal number of pieces is 3 for all models (only *SimpleFeedForward* comparison is presented).

5.2.7 Alpha

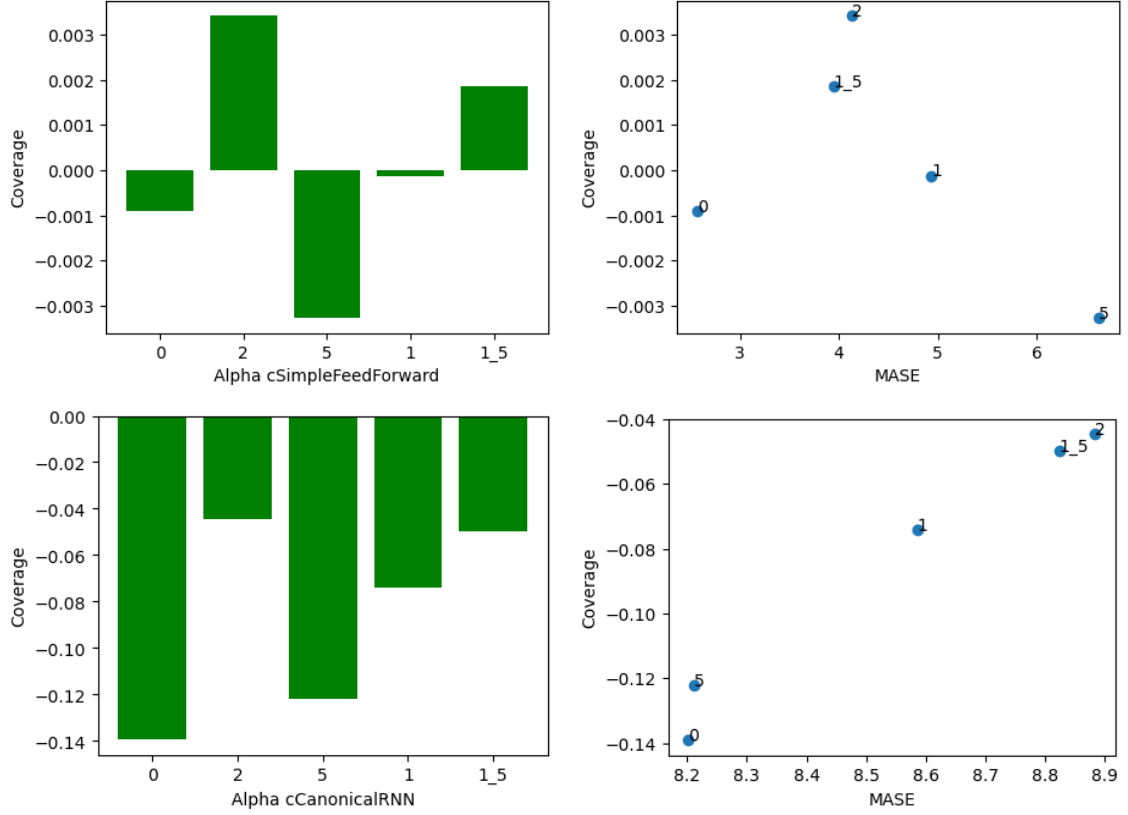


Figure 5.18: Comparison between different α values for different models (Models: *SimpleFeedForward*, *CanonicalRNN* (Default hyperparameter values), Config A)

Hyperparameter α has been introduced in section 4.3, as a weight between default loss and alternative loss, components of the custom loss. More α value increases, more the alternative loss impacts the custom loss. In this section we interrogate the impact of the α on results.

α is implied as a hyperparameter in all models where the custom loss as been implemented. As results varies in function of the models, results are presented for each of them, in figures 5.18.

These results allows us to deduce the optimal value of α for each models. Model SimpleFeedForward does not improve in terms of *Coverage* when α increases, as its *Coverage* value is already near the optimum for $\alpha = 0$, but we observe a loss in terms of *MASE*. The value 0 is considered as optimal. CanonncalRNN value is 2, as it improves significantly the *Coverage* without a significative increases of *MASE*. These values are considered as default values in the following comparison sections. We can globally observe that *Coverage* increased proportionally to α value, which is coherent with the goal of the alt loss, i.e penalising predictions where the security quantile value is underestimated.

5.3 Different models parameters comparison and results

The different models are described in the 2.7 section

5.3.1 Simple Feed Forward

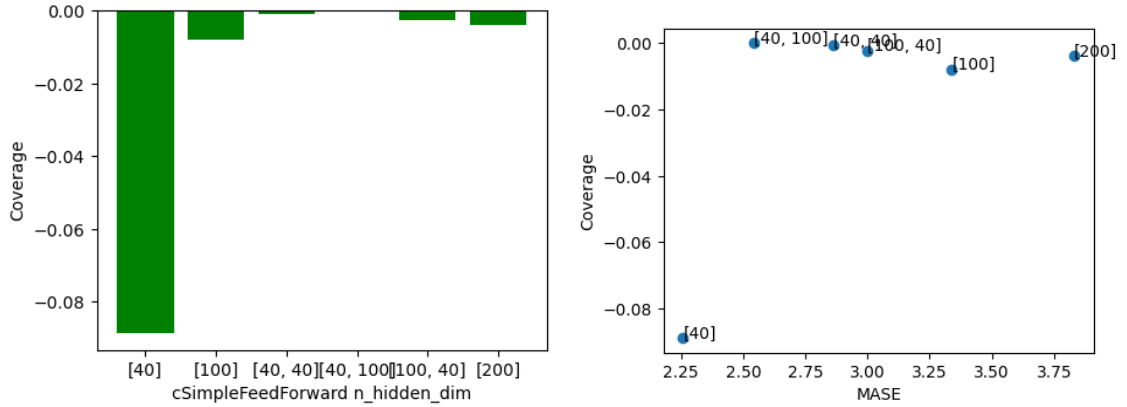


Figure 5.19: Comparison between different n_hidden_dim values for *SimpleFeedForward* model (Default hyperparameter values), Config A)

We compare, in continuity with 3.4.1, different values of own *SimpleFeedForward* hyperparameter.

Results shows that the optimal value of n_hidden_dim is [40,100]. Plots results, illustrating the model predictions, are presented in figure 5.20 because, as some default hyperparameters values differs from default values in 3.4.1, results are significantly different from the previously presented plots. We can observe that the model prediction interval is larger than in the previous plots.

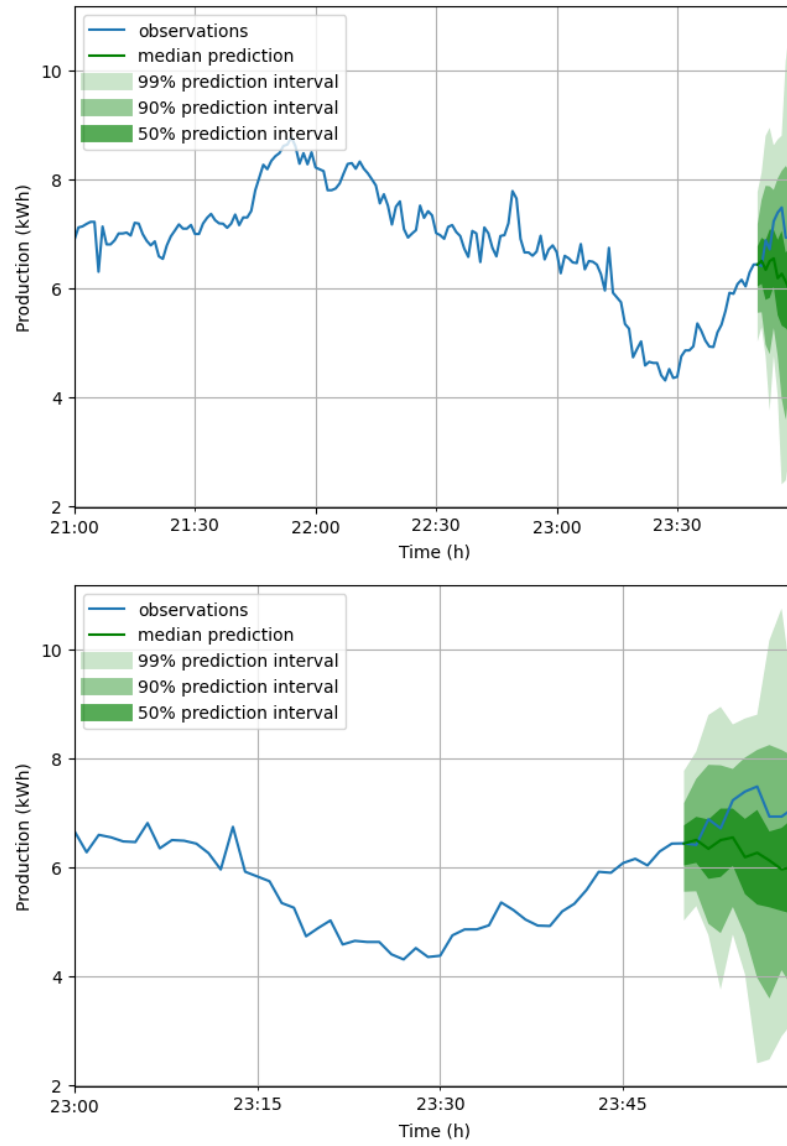


Figure 5.20: Forecast result of *SimpleFeedForward* model at 3 hours and 1 hour scale ($n_hidden_dim = [100]$, Default hyperparameter values), Config A)

5.3.2 Canonical RNN

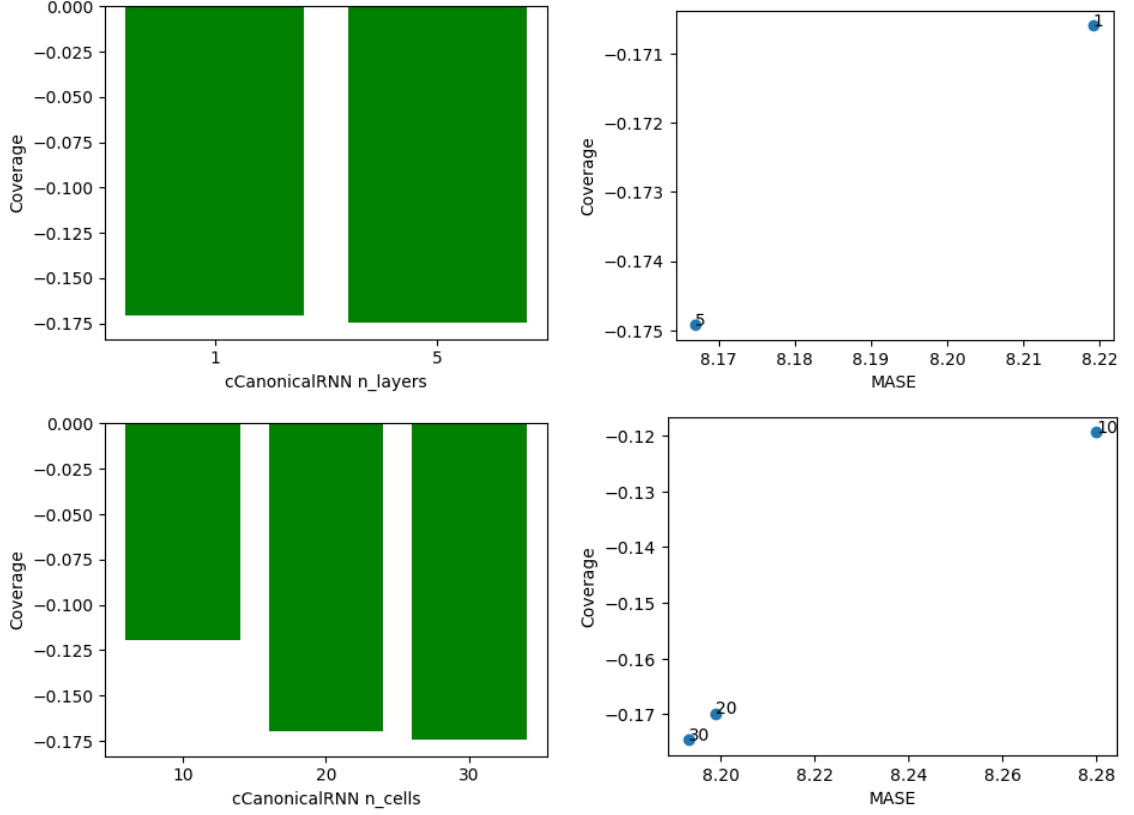


Figure 5.21: Comparison between different n_{layers} and n_{cells} values for *CanonicalRNN* model (Default hyperparameter values, Config A)

We compare, in continuity with 3.4.2, different values of own *CanonicalRNN* hyperparameters.

Results shows that the optimal value of n_{layers} is 1 and the optimal value of n_{cells} is 10 . Plots results, illustrating the model predictions, are presented in figure 5.20 because, as some default hyperparameters values differs from default values in 3.4.2 , results are significantly different from the previously presented plots. We can observe that the model prediction interval is larger than in the previous plots.

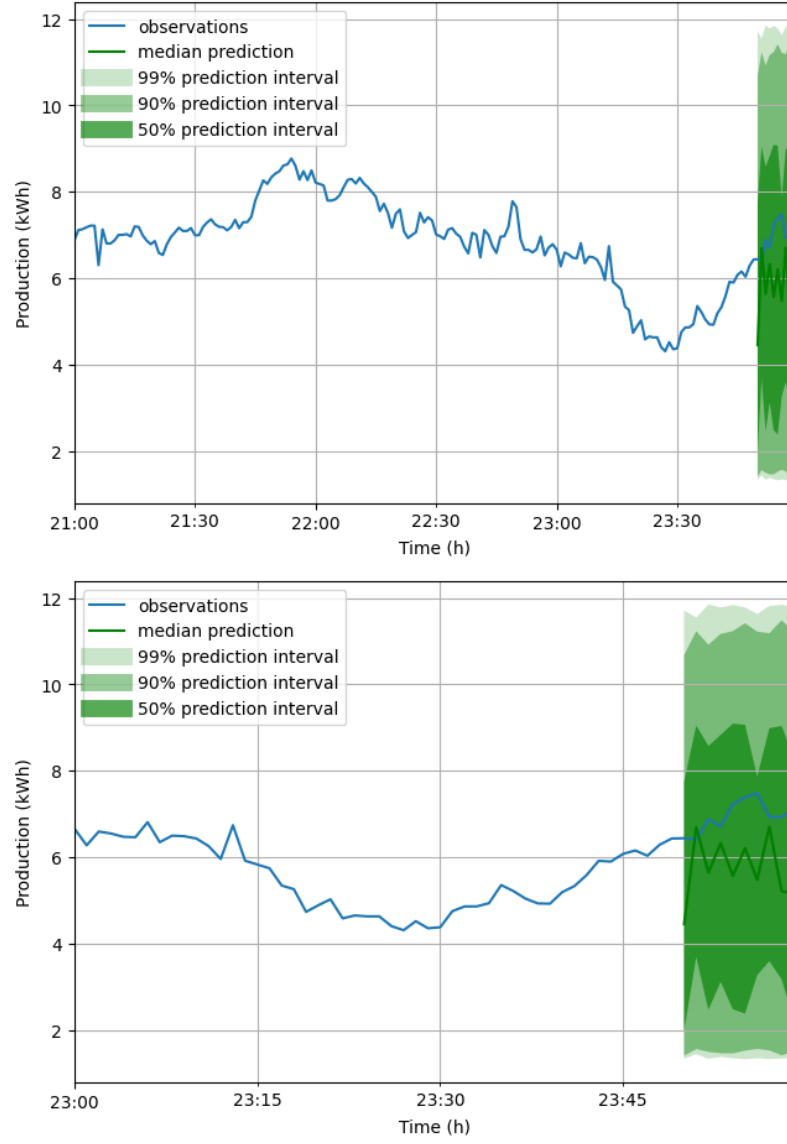


Figure 5.22: Forecast result of *CanonicalRNN* model at 3 hours and 1 hour scale ($n_layers = 4$, $n_cells = 20$ Default hyperparameter values, Config A)

5.3.3 Deep AR

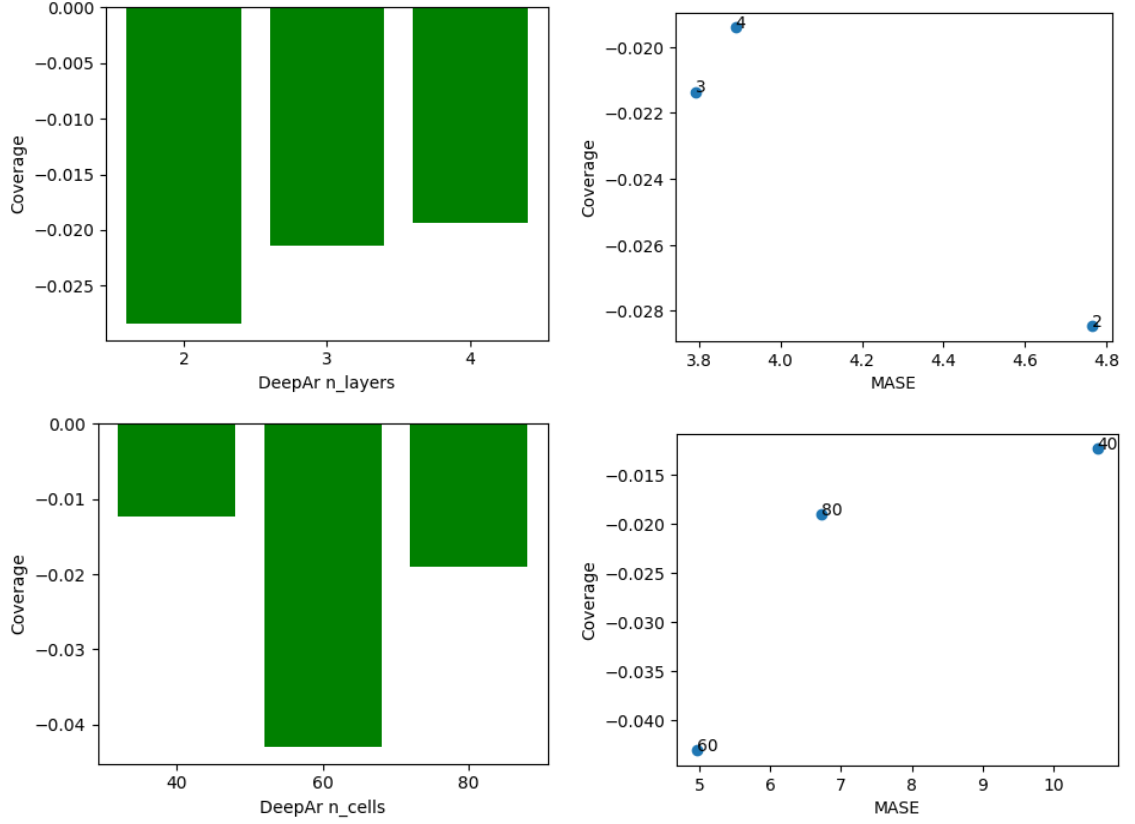


Figure 5.23: Comparison between different n_{layers} and n_{cells} values for *DeepAR* model (Default hyperparameter values, Config A)

We compare, in continuity with 3.4.3, different values of own *DeepAR* hyperparameters.

Results shows that the optimal value of n_{layers} is 4 and the optimal value of n_{cells} is 40, but the loss in MASE (multiplied by 2) is not valuable, we keep a value of 60. Plots results, illustrating the model predictions, are presented in figure 5.20 because, as some default hyperparameters values differs from default values in 3.4.3, results are significantly different from the previously presented plots. We can observe that the model prediction interval is larger than in the previous plots.

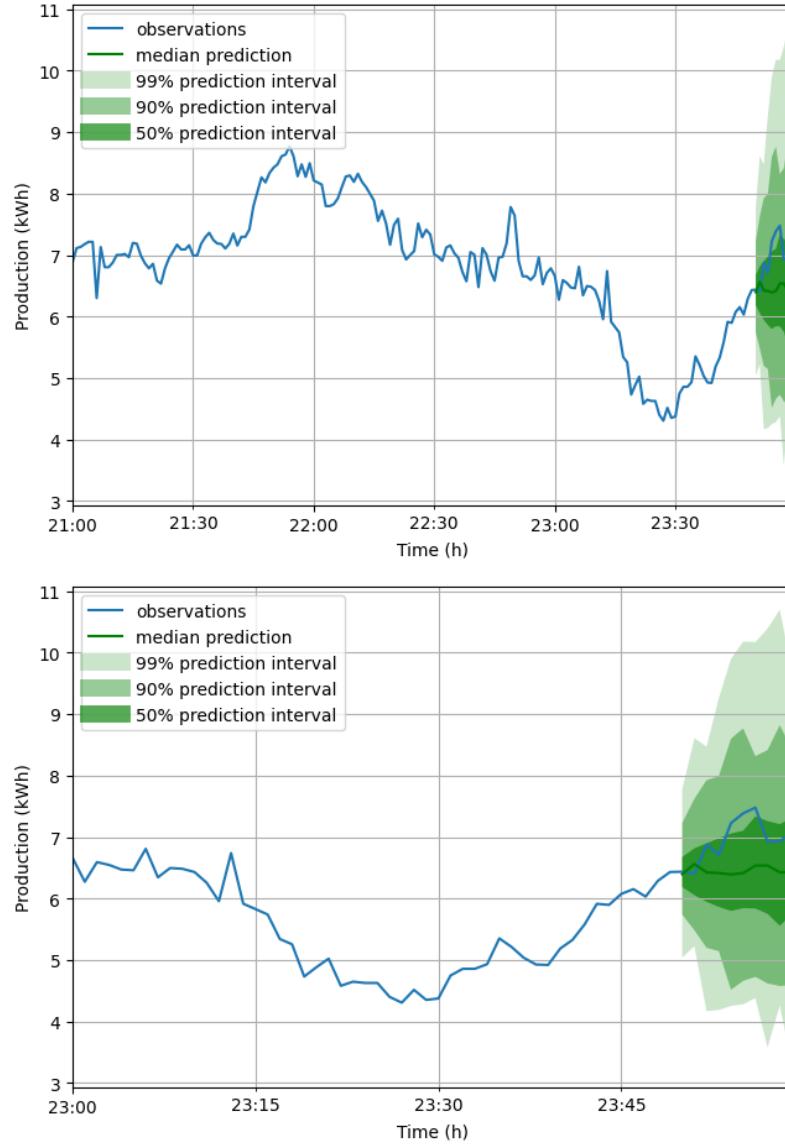


Figure 5.24: Forecast result of *DeepAr* model at 3 hours and 1 hour scale ($n_{layers} = 3$, $n_{cells} = 80$, Default hyperparameter values, Config A)

5.3.4 Deep Factor

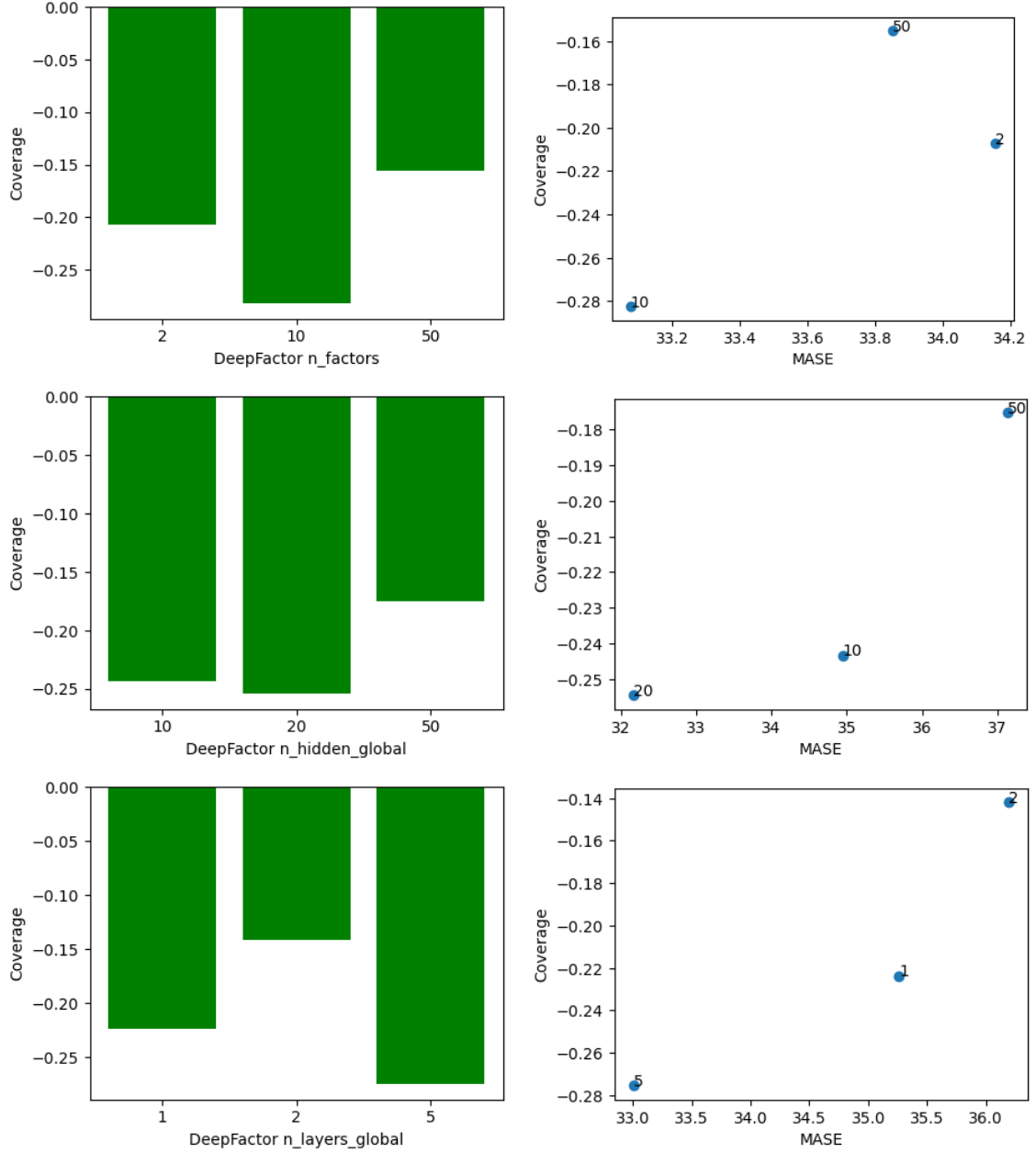


Figure 5.25: Comparison between different $n_factors$ values, n_hidden_global values and n_layers_global values for *DeepFactor* model (Default hyperparameter values, Config A)

We compare, in continuity with 3.4.4, different values of own *DeepFactor* hyperparameters.

Results shows that the optimal value of $n_factors$ is 60, n_hidden_global is 50 and the optimal value of n_layers_global is 2. If some default hyperparameters values differs from default values in 3.4.4, plot results are very similar.

5.3.5 MQCNN

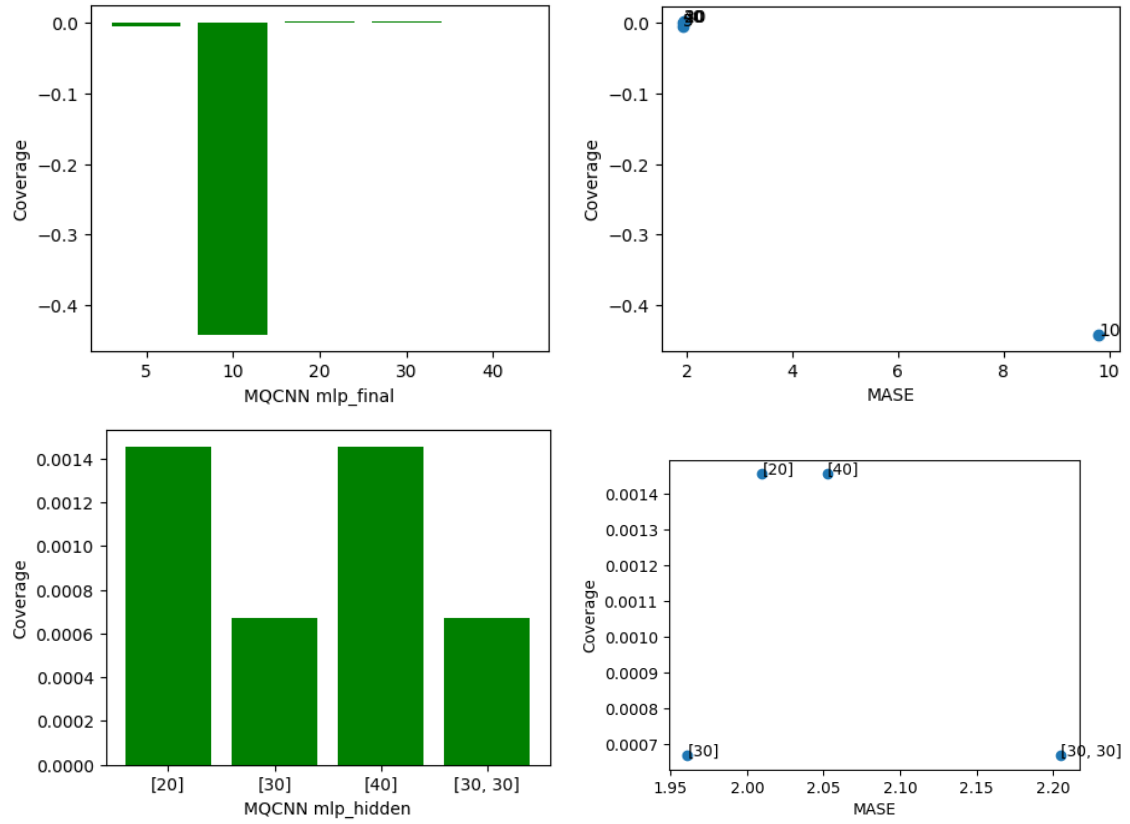


Figure 5.26: Comparison between different mlp_final_dim and mlp_hidden values for *MQCNN* model (Default hyperparameter values, Config A)

We compare, in continuity with 3.4.5, different values of own *MQCNN* hyperparameters.

Results shows that the optimal value are the same than the previous ones. If some default hyperparameters values differs from default values in 3.4.5, plot results are very similar.

5.3.6 MQRNN

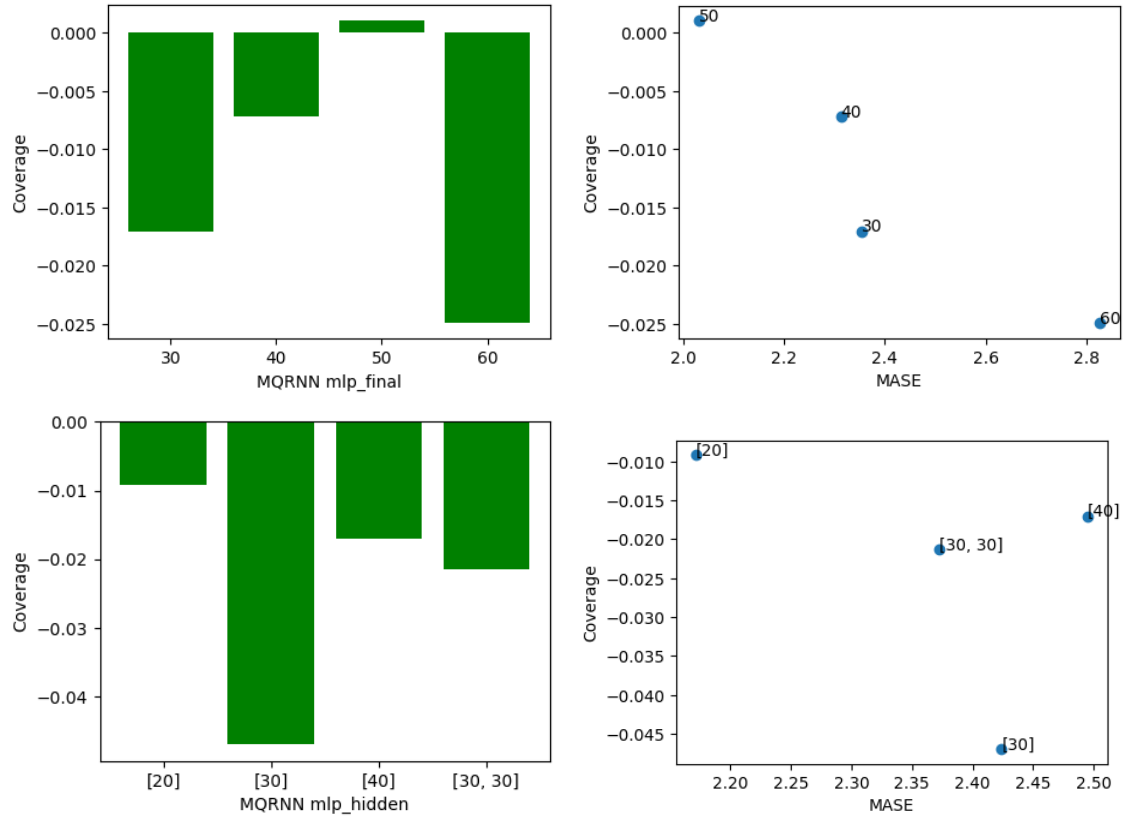


Figure 5.27: Comparison between different `mlp_final_dim` and `mlp_hidden` values for *MQRNN* model (Default hyperparameter values, Config A)

We compare, in continuity with 3.4.6, different values of own *MQRNN* hyperparameters.

Results shows that the optimal value are the same than the previous ones. If some default hyperparameters values differs from default values in 3.4.6, plot results are very similar.

5.3.7 Transformer

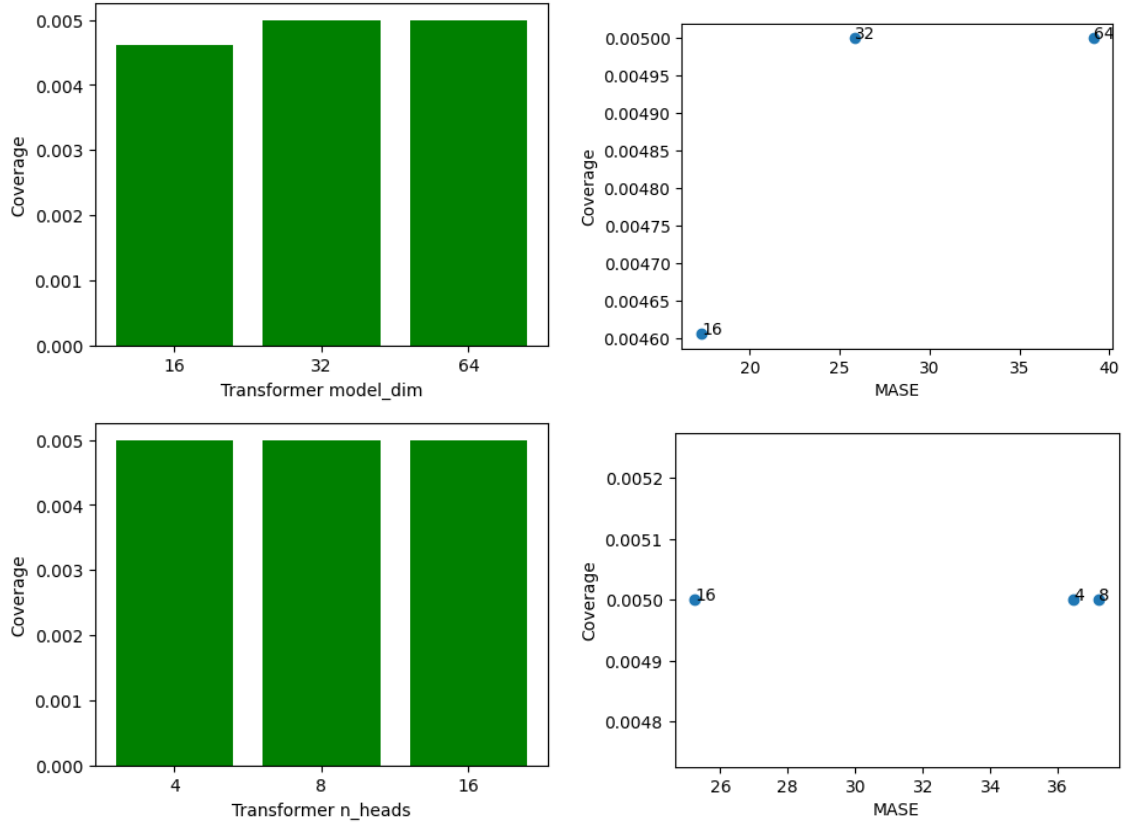


Figure 5.28: Comparison between different *model_dim* and *n_heads* values for *Transformer* model (Default hyperparameter values, Config A)

We compare, in continuity with 3.4.7, different values of own *Transformer* hyperparameters.

Results shows that the optimal value are the same than the previous ones. If some default hyperparameters values differs from default values in 3.4.7, plot results are very similar.

5.3.8 Wavenet

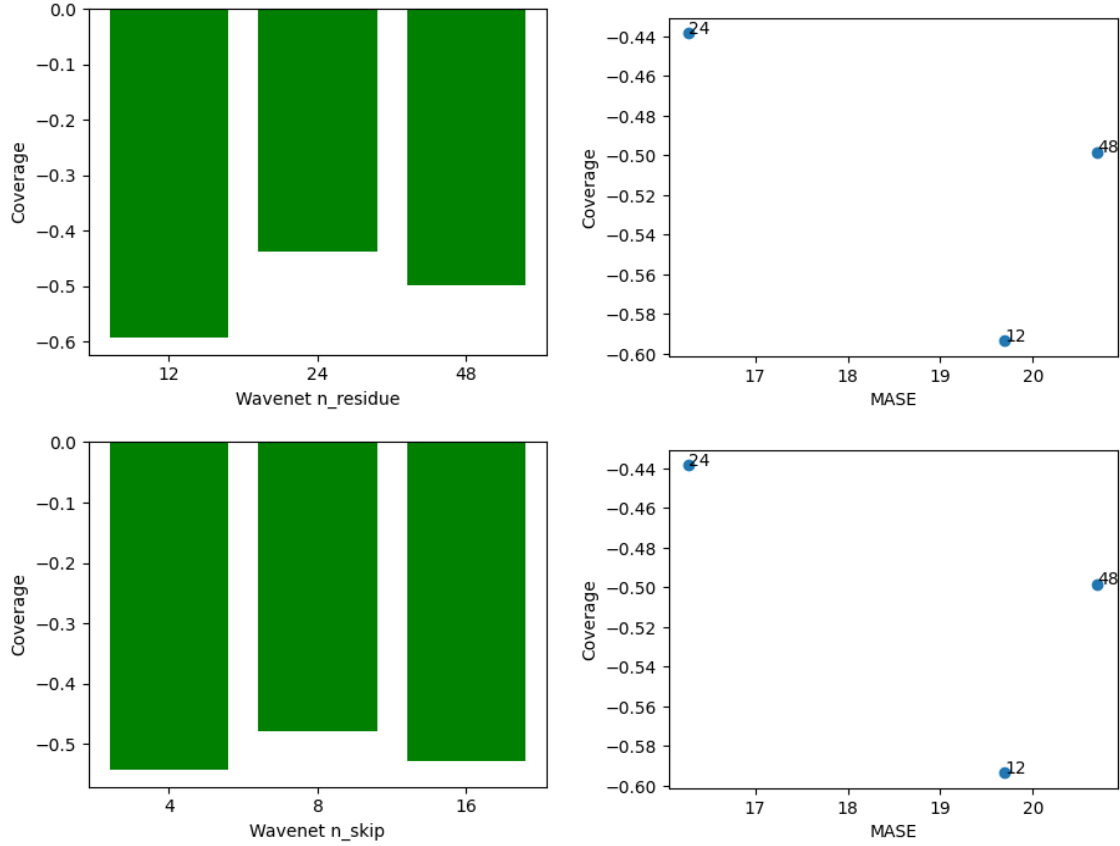


Figure 5.29: Comparison between different *num_residue* and *n_skip* values for *Wavenet* model (Default hyperparameter values, Config A)

We compare, in continuity with 3.4.8, different values of own *Wavenet* hyperparameters.

Results shows that the optimal value are the same than the previous ones. If some default hyperparameters values differs from default values in the referred section, plot results are very similar.

5.3.9 NBEATS

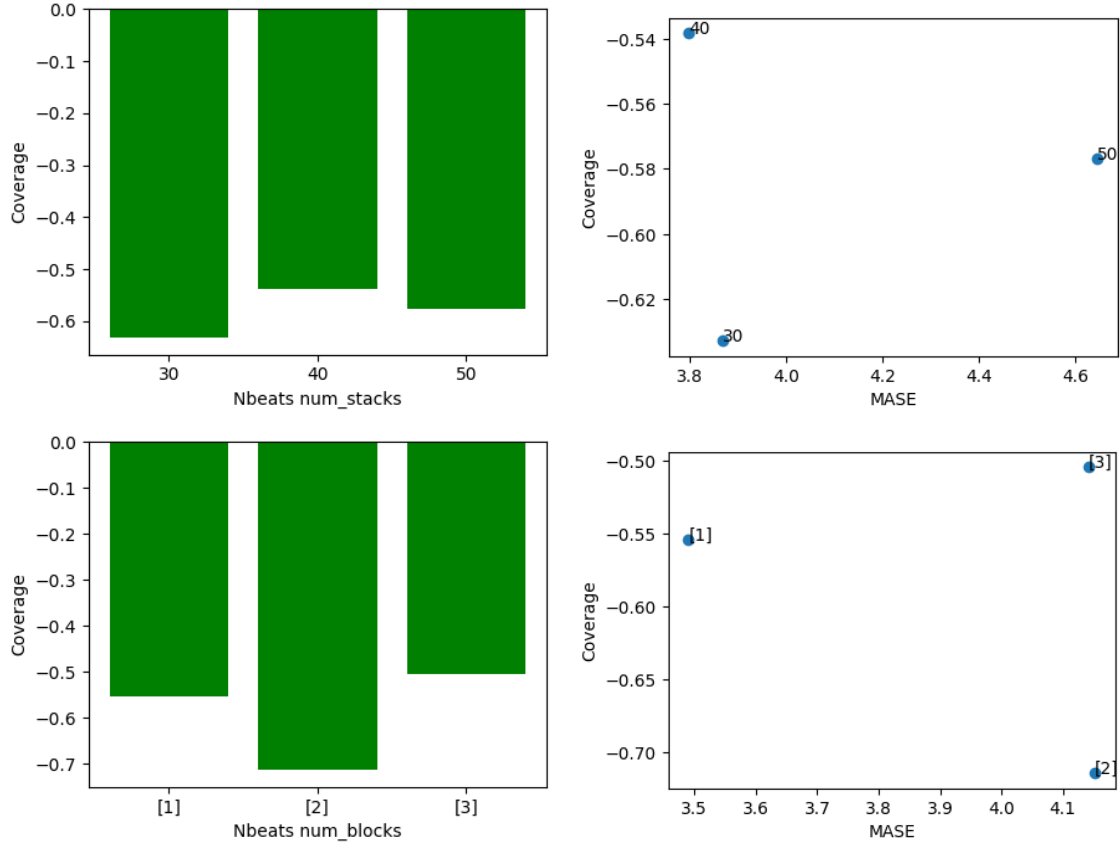


Figure 5.30: Comparison between different *num_stacks* and *num_blocks* values for *NBEATS* model (Default hyperparameter values, Config A)

We compare, in continuity with 3.4.9, different values of own *Wavenet* hyperparameters.

Results shows that the optimal value of *num_stacks* is 40 and *num_blocks* is [3]. If some default hyperparameters values differs from default values in 3.4.9, plot results are very similar.

5.4 Comparison of all tuned models

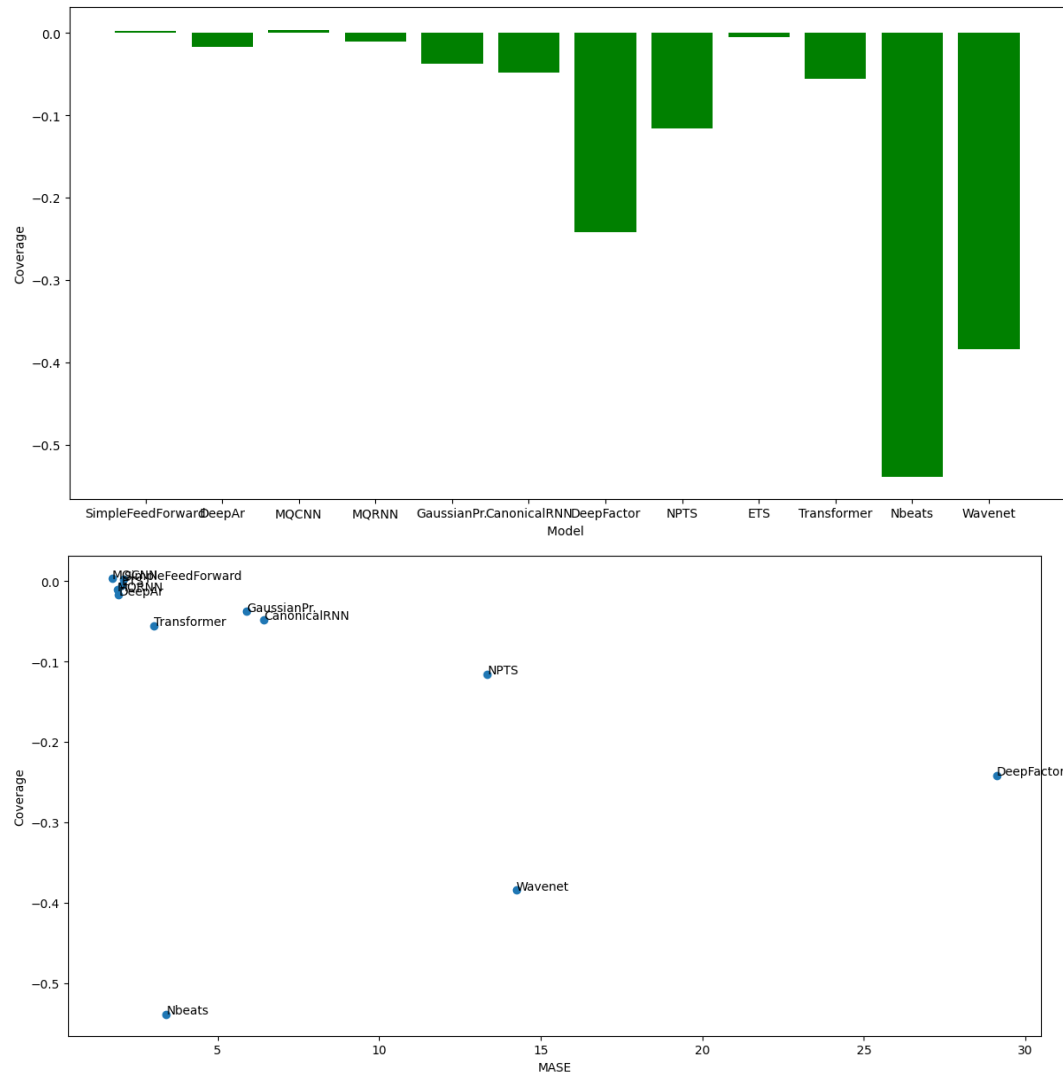


Figure 5.31: Comparison between all different models (Default hyperparameter values, Config A)

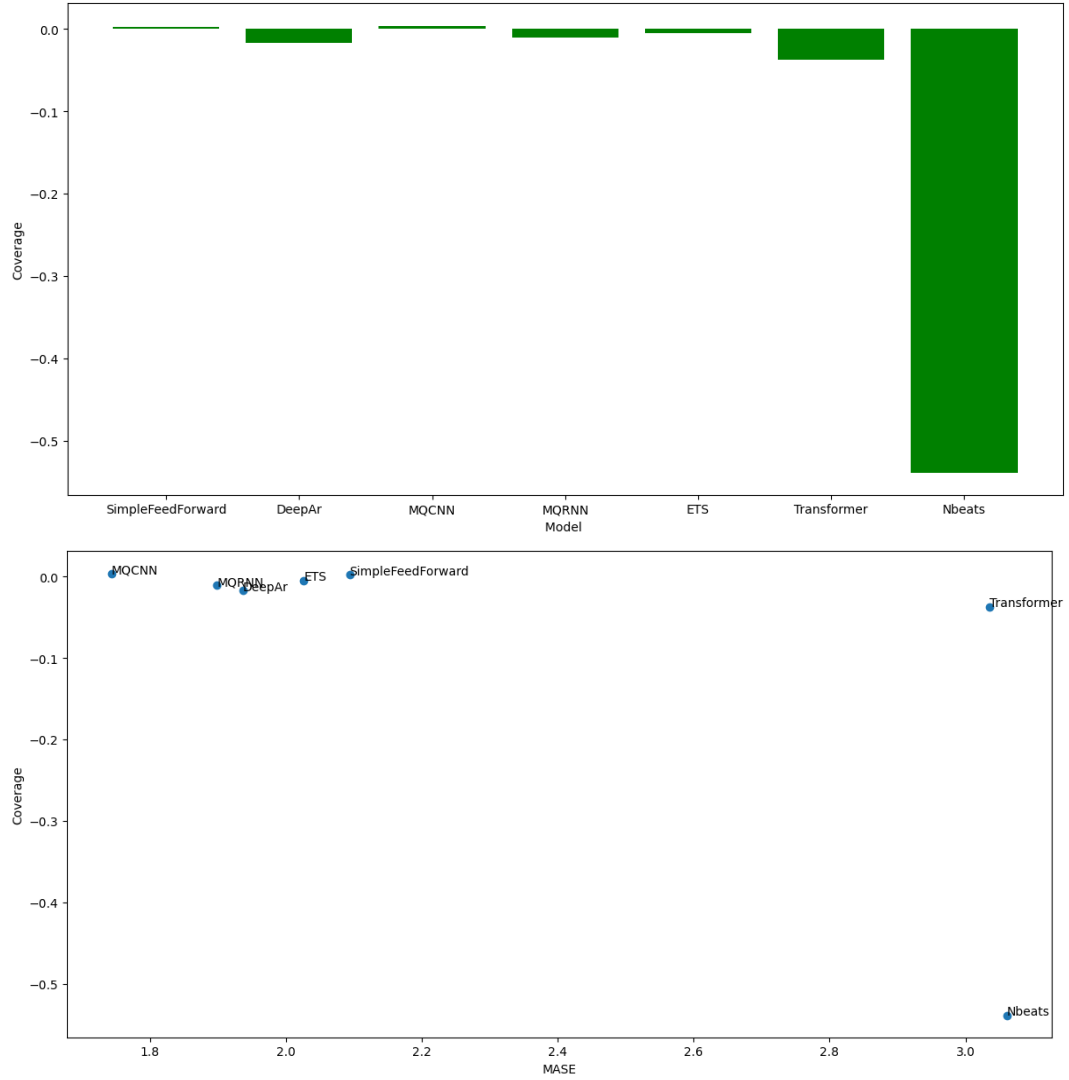


Figure 5.32: Comparison between the 7 best models in terms of *MASE* (Default hyperparameter values, Config A)

A first observation is that there are very large differences in terms of *Coverage* metric value between the different models.

In section 3.5, we have classified the models into four groups in function of their *MASE*. Models of the group 1, *DeepFactor* and *NPTS*, the worst in terms of *MASE*, performed also bad considering the *Coverage*. *Wavenet*, of the group 2 *Nbeats* of the group 3, provides really poor performances in terms of *Coverage* and are not acceptable as forecasting solution either. All these models has a *Coverage* value inferior to 0.1, which is considered as really bad, as it means that the models indicates that 99% of the observed points will be below the security quantile value that it provides where in reality there are less than 89% between this value, in mean.

Transformer, *GaussianProcess* and *CanonicalRNN* provides interesting results in terms of *Coverage* but their results in terms of *MASE* are not satisfying. We can conclude that in this context *GaussianProcess* and *Transformer* are not acceptable

solutions, in contrast with the chapter 3 results. The differences in default hyperparameter values, to optimize *Coverage* metric values results, impact the *MASE* metric values results.

It leaves us four models, *SimpleFeedForward*, *DeepAr*, *MQCNN* and *MQRNN* that rivalize with the *ETS* model. The differences in default hyperparameter values, to optimize *Coverage* results, impact the *MASE* results significantly for *DeepAr* and *SimpleFeedForward*. The four models provides *Coverage* values similar (*MQRNN* and *DeepAr*) or better (*MQCNN* and *SimpleFeedForward*) than *ETS*. Three models, *MQCNN*, *MQRNN* and *DeepAr*, outperforms *ETS* in terms of *MASE*.

From these results, we can easily concludes that, in the defined testing context, model *MQCNN* outperforms in all domains a classical technique like *ETS*, where *MQRNN*, *SimpleFeedForward* and *DeepAr* provides results equivalent to *ETS* (some of them with a better *Coverage* and worse *MASE*, or vice-versa). The other models provides less brilliant results, from correct to really bad.

Chapter 6

Conclusion

To summarize the different steps that has been accomplished, we have introduce the main goal that we want to achieve, the comparison of different probabilistic forecasting models in the context of the prediction of renewable energy production, to protect assets from oversupply (1). Then, we have presented what are the tools needed to accomplish the goal and define the context (datasets, *predict_length*, etc) on which the comparison will be performed (2). We perform a comparison considering a default metric to evaluate models performances (3). Furthermore, we discuss what might be the most relevant metric considering the main goal that we want to achieve (4) and we finally perform a comparison using metrics and loss that have been introduced (5).

The answer to the question of what is the better forecasting model in the defined context between all the tested models, the model that provides the better results, in terms of *Coverage* and *MASE*, is definitely *MQCNN*, which outperforms for the two metrics considered all the other presented models. It is followed by three models, *MQRNN*, *DeepAr* and *SimpleFeedForward*. All the other models presented (*CanonicalRNN*, *Transformer*, *NBEATS*, *DeepFactor*, *Wavenet*, *NPTS*, *GaussianProcess*) are outperformed by these four models and the model *ETS*. We must insist that these results are obtains in a defined context : for the datasets that has been presented, for a *predict_lenght* of 10 and with as goal the minimization of the difference between the real and predicted security quantiles and secondly the minimization of the difference between real and predicted distributions.

An important note is that the model *ETS*, a “classical” forecasting model which does not belongs to the deep learning field, is overpassed only by *MQCNN* and *SingleFeedForward* in terms of *Coverage* and by *MQCNN*, *MQRNN* and *DeepAr* in terms of *MASE* (in the context where the main metric to optimize is *Coverage*). From these results, we can conclude that the great majority of the deep learning techniques that have been tested does not outperforms easily and very significantly classical techniques like *ETS*. Nevertheless, *MQCNN* (*Coverage* $\simeq 0$, *MASE* = 1.7), outperforms significantly *ETS* (*Coverage* = -0.005, *MASE* = 2.1). One more time, these results are presented for the defined context, and conclusion could not be drawn for complete different contexts.

These conclusions drives us to a reflection about what other context would be interesting to explore. Firstly, as mentioned in 1.3, the comparison performed in this master thesis is about the models that are implemented in *GluonTS*. Some state-of-the-art models that are compared in reviews [25] and [15] are not compared here because of this choice. In addition of that, some implemented models of *GluonTS* has not been usable, because of running errors that will be certainly corrected in future versions. The comparison of these other models would be obviously interesting, as they use different techniques to tackle the problem of probabilistic forecasting, with some of them possibly very compatible / better in the context of wind turbines production probabilistic forecasting. But, if this comparison would be interesting, it is not easy to performs it, as some models implementations are complex and not accessible. Another element of context is the input data. The comparison is performed for a fixed amount of input information. Other comparison, with possible more data, or data containing more varied types of wind turbines, would possible provides different results. The size of the prediction interval would also possibly greatly influences the results but increases it will take us away of the defined goal. The use of other distribution output (*Beta* or *Student* for example) could be interesting as well.

Bibliography

- [1] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, Section 3.1.
- [2] ———, *Forecasting: Principles and Practice*, Section 8.
- [3] R. G. Brown, *Smoothing Forecasting and Prediction of Discrete Time Series*. 1963.
- [4] E. S. Gardner, *Exponential smoothing: The state of the art. Journal of Forecasting*. 1985.
- [5] D. C. Maddix, Y. Wang, and A. Smola, “Deep factors with gaussian processes for forecasting”, 2018.
- [6] R. Wen, K. Torkkola, B. Narayanaswamy, and D. Madeka, “A multi-horizon quantile recurrent forecaster”, 2017.
- [7] N. Laptev, J. Yosinski, L. E. Li, and S. Smyl, “Time-series extreme event forecasting with neural networks at uber”, 2017.
- [8] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, “The m4 competition: Results, findings, conclusion and way forward”, 2017.
- [9] M. Abadi, 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”, 2016.
- [10] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, “Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems”, 2015.
- [11] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performancedeep learning library”, 2019.
- [12] R. J. Hyndman and Y. Khandakar., “Automatic time series forecasting: The forecastpackage for r”, *Journal of Statistical Software*, 2008.

- [13] A. Alexandrov, K. Benidis, M. B.-S. and Valentin Flunkert, J. Gasthaus, T. Januschowski, D. C. Maddix, S. Rangapuram, D. Salinas, J. Schulz, L. Stella, A. C. Türkmen, and Y. Wang, “Gluonts: Probabilistic time series models in python”, 2019.
- [14] Y. Zhang, J. Wang, and X. Wang, “Review on probabilistic forecasting of wind power generation”, 2014.
- [15] H. Wanga, X. Z. Zhenxing Leia, B. Zhouc, and J. Peng, “A review of deep learning for renewable energy forecasting”, 2019.
- [16] S. S. Rangapuram, M. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski, “Deep state space models for time series forecasting”, 2018.
- [17] D. Salinas, V. Flunkert, and J. Gasthaus, “Deepar: Probabilistic forecasting with autoregressive recurrent networks”, 2017.
- [18] Y. Wang, A. Smola, D. C. Maddix, J. Gasthaus, D. Foster, and T. Januschowski, “Deep factors for forecasting”, 2019.
- [19] D. J. C. MacKay, “A practical bayesian framework for backpropagation networks”, 1992.
- [20] H. L. Shang and R. J. Hyndman, “A practical bayesian framework for back-propagation networks”, 2011.
- [21] R. Wen, K. Torkkola, B. Narayanaswamy, and D. Madeka, “A multi-horizon quantile recurrent forecaster”, 2018.
- [22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need”, 2017.
- [23] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio”, 2016.
- [24] B. N. Oreshkin, D. Carпов, N. Chapados, and Y. Bengio, “N-BEATS: neural basis expansion analysis for interpretable time series forecasting”, 2019.
- [25] H.-z. Wanga, G.-q. Lia, G.-b. Wang, J.-c. Peng, H. Jiang, and Y.-t. Liu, “A review of deep learning for renewable energy forecasting”, 2016.
- [26] R. J. Williams, G. E. Hinton, and D. E. Rumelhart, “Learning representations by back-propagating errors”, *Nature*, 1986.