Deltares

CIP: Confidence intervals for SWAN wave forecasts

surrogate North Sea wave model to apply wind ensembles

CIP: Confidence intervals for SWAN wave forecasts
surrogate North Sea wave model to apply wind ensembles

Author(s)
Caroline Gautier
Elias de Korte
Guus van Hemert
Joana van Nieuwkoop

CIP: Confidence intervals for SWAN wave forecasts
surrogate North Sea wave model to apply wind ensembles

Client	Rijkswaterstaat Water, Verkeer en Leefomgeving
Contact	Joost Driebergen
Reference	-
Keywords	Surrogate model, Machine Learning, SWAN, waves, wind ensembles, uncertainty

Document control	
Version	2.0
Date	20-12-2024
Project number	11210320-018
Document ID	11210320-018-BGS-0001
Pages	40
Classification	
Status	final

Author(s)

	Caroline Gautier Elias de Korte Guus van Hemert Joana van Nieuwkoop	

Summary

Generally, the performance of the SWAN wave models used at Rijkswaterstaat Operational Systems (RWsOS) is well known and model bias is within a few percent. In contrast the spread can be large in certain cases, in particular for low frequency waves He10. Therefore, we aim to model uncertainty to anticipate deviations in the wave forecast from actual wave conditions. An important constraint for RWsOS is that the future model is computationally feasible. The approach for this report is twofold. The uncertainty was first modelled and compared with observations to verify the uncertainty model forecast. Secondly, a fast data-driven surrogate model TurboSWAN was trained on the ensemble wave forecast input and output and compared to the physical wave model SWAN results.

The uncertainty in the wind is modelled, using the ECWMF TIGGE wind ensemble to force the wave model for a year of three-day forecasts (2022) and several additional non-sequential months including storms (in 2013 and 2017-2022). We assumed errors in the wind fields dominated. This was in part true. For longer lead times (after 2 days), uncertainty was well captured. However, at shorter lead times uncertainty is often underpredicted and likely comes from other uncertainty sources (e.g. boundary conditions, (missing) physics). Another limitation is the relatively coarse resolution of ECMWF TIGGE wind fields compared to the HARMONIE wind fields.

To create a faster model, a surrogate model TurboSWAN was trained on the ensemble wave forecast input and output. A convolutional neural network with a U-net based architecture was used to train TurboSWAN, mainly because these types of networks excel in spatial pattern recognition. TurboSWAN is very fast and already able to model the dominant dynamics. However, the evaluation shows that during storms errors at various measurement stations are still too large for an application. Over the entire grid, the mean error is approximately 0.5 m in most parts for significant wave height, with larger errors near the coast.

The prototype of TurboSWAN is considered a good starting point for further development. There are several ways in which we can improve both the uncertainty representation and the AI model TurboSWAN. We recommend to further study the underrepresentation of the ensemble spread in the ensemble model. The AI model TurboSWAN can be improved in several ways: testing different input combinations, tuning hyperparameters and testing variations of the net architecture.

Contents

1 Introduction

1.1 General

Within the Rijkswaterstaat Operational Systems (RWsOS), SWAN wave models are used to provide wave forecasts for both the Dutch coast and the larger lakes. These forecasts are essential for save navigation and safety during highwater conditions.

Based on several hindcast studies (Deltares, 2023a and Deltares, 2024a,b) the overall model performance of these wave forecasts is well known. The bias in significant wave height H_{m0} is in general good with just a few percent deviation. However, the scatter in wave height can be rather large, especially for the low frequency wave height H_{E10}. As the wave forecast can deviate significantly from the actual wave conditions, a measure of uncertainty of the wave forecast is requested by the wave forecast users, the Rijkswaterstaat Water Management Centre (WMCN) and the Hydro Meteo Centre (HMC). Confidence intervals of the wave forecast could indicate whether there is a large uncertainty in the forecast, for example due to uncertainty in the wind. The challenge here is to provide uncertainty of the wave forecast without adding a significant amount of computational time to the forecast suite.

This project - part of the Rijkswaterstaat Corporate Innovation Program (CIP) – is set up to find a suitable way to add an uncertainty range to wave forecasts. Note that the uncertainty of the forecast consists of different types of uncertainty, like model and input uncertainties (wave boundary conditions, wind, water levels etc.) and uncertainties for increasing lead times. The project focusses for now on the wind uncertainty by assessing ECMWF wind ensemble runs for the SWAN North Sea model. This choice is made, as the wind is a primary force for the waves in the North Sea and Lakes. A project follow-up is needed to bring the concept eventually to the end goal: confidence intervals are added to the operational wave forecasts of RWsOS.

1.2 Objective

The objective of this project is to develop a concept method to provide confidence intervals to operational wave forecasts, making use of ECMWF ensembles for wind input.

1.3 Set up of the report

The approach of this project is discussed in Chapter 2. Furthermore, in this chapter an overview is given of the data and computing power that has been used. Chapter 3 focuses on the SWAN wave computations, with a description of the model set up and the ensemble wave forecast runs. In addition, the ensemble wave forecast is analysed. Subsequently, Chapter 4 gives an overview of the TurboSWAN surrogate model set up and the results. In Section 4.5 the surrogate model is run with the wind ensembles from ECMWF for a short period. Finally, conclusions are drawn in Chapter 5 and an outlook for follow-up study is given.

2 Approach and data

2.1 Approach

Although several sources of uncertainty add up to the overall uncertainty of a forecast, the focus for this study is on uncertainties in the wind input as it is expected that the wind contributes largely to the wave forecast uncertainty. This means that uncertainties related to the wave model (physics, numerical schemes, convergence, resolution, schematisation) and other model input than wind (bathymetry, water levels, currents, boundary conditions) are not considered.

The European Centre for Medium range Weather Forecasts (ECMWF) delivers with their Ensemble Prediction System (EPS) one control weather forecast starting from the best guess initial conditions and fifty members starting from slightly perturbed initial conditions. This model will be used in this study as input for the SWAN North Sea forecast.

In an operational setting, it would be too time consuming to perform 51 SWAN forecast runs to assess the spread in wave parameters caused by the 51 ensemble members of wind. Therefore, a method is needed to be able to run all 51 ensemble members in an acceptable amount of time. Various methods were considered:

- **Increasing computing power** for example by using a supercomputer and running the ensembles simultaneous. This option may be interesting in the future. For the operational system it is not yet feasible, as the current set up does not allow for more models to be run.
- **Decreasing resolution of the current wave model** for example by using Multilevel Monte Carlo techniques. Van Ooijen (2023) studied Multilevel Monte Carlo methods to speed up the ensemble wave forecast. His conclusion was that ensemble forecasts can be sped up by a factor four to eight without becoming much more inaccurate. However, the ensemble seemed to underestimate the uncertainty systematically. Some recommendations were given to continue the research and improve the results. Although this might be a feasible method in the future, it was chosen not to focus on this method in the current study.
- **Use of an alternative fast wave model** for example a fast alternative wave model or machine learning model. Deltares (2023b) looked at fast wave models, as an alternative for SWAN, however either the speed up in the North Sea was marginal or the accuracy was not acceptable. Another option is to use a machine learning model that has been trained on wave data. As this option could produce a wave model that is considerably faster than SWAN (51 ensembles in just a minute) and could be acceptable in terms of accuracy, this option was chosen to explore further.

This means that in this study we aim to develop a machine learning model, a so-called surrogate model. This model is based - without any physical equations - on similar input that SWAN receives, like spatial fields of wind speed, wind direction, water level, current, wave boundaries and bathymetry. It will produce three wave parameters as output on the whole model domain. Hereto the model is trained with thousands of combinations of input and SWAN output, most of them made within this project on the Snellius supercomputer at Surf Sara. It must be realised that if the surrogate model performs perfectly, it will give identical results as SWAN would, but never better than that.

Ultimately, the wind ensembles can be used to force the fast wave model. The result, being the spread in wave parameters H_{m0}, $T_{m-1,0}$ and H_{E10}, will be applied on the outcome of the one

SWAN run which used the control wind field as input. In the present study the spread in wave direction is not included. In future we will discuss the needs with possible users.

2.2 Data

For this project we make use of the following data:

- The existing SWAN-North Sea model swan-noordzee-j22_6-v1a, consisting of a computational grid definition, bathymetry and model settings, see Factsheet SWAN-Noordzee.
- ECMWF wind ensemble members for the period 2007-2024 from the TIGGE project (“The International Grand Global Ensemble” TIGGE archive - TIGGE - ECMWF Confluence Wiki)
- Model input (water levels, currents, wave boundary conditions) for selected events as training data. Water levels and currents from DCSM WAQUA or FM and wave boundary conditions from ECMWF WAM.
- Observations of waves and wind to validate the wave model results, downloaded from matroos.deltares.nl.

2.3 Snellius

All computations have been done on Snellius: the National Supercomputer at SURF Sara. For this project we arranged the following computational capacity: 350.000 SBU on CPU thin nodes (78% used) and 300.000 SBU on GPU (87% used), as well as 7TB project space (100% used).

3 SWAN wave computations

3.1 Introduction wave computations

In this chapter the set up and analysis of SWAN ensemble computations has been described. The 50 ensemble members of ECMWF wind were used to force the SWAN-North Sea model. The SWAN model computations are used to train the surrogate wave model. In addition, it is checked to which extent wind and wave measurements fall within the confidence bounds of the ECMWF wind ensembles and the ensemble SWAN computations.

3.2 SWAN-North Sea

3.2.1 Period Selection

For the training of the TurboSWAN surrogate model (see Chapter 4), a realistic representation of wave conditions is selected to train the surrogate model, but within acceptable computational time. In this light one year (2022) of daily forecasts were computed, each with 72 hours forecast lead time. The year 2022 covered storms like Corrie, Malik and Eunice, but might not capture the full range of relevant system dynamics. To anticipate on this expectation, we modelled additional seven non-sequential months that captured storms in the years 2013 and 2017-2023. We expect this will diversify the training data of the dynamics that are most interested in predicting.

3.2.2 SWAN Version

The latest available Deltares SWAN version was used: swan_deltares41_45AB_2_omp.exe. This executable was compiled on Snellius.

3.2.3 Model input

Table 3.1 gives an overview of the input data that was used to force the SWAN-North Sea model.

The operational SWAN-North Sea model uses ECMWF WAM forecasts for its wave boundary conditions. These data have been available at Deltares since 2013 and can therefore be used for all periods. The data are available for every three hours and have been downloaded from Matroos.rws.nl. Note that these are not ensemble boundary conditions. ECMWF also provides a WAM wave ensemble that can be used to include uncertainty in the boundary conditions in the future.

For water levels and currents the operational SWAN-North Sea uses WAQUA DCSMv6 data produced by the RWsOS forecast. These data are available every hour and have been applied to SWAN with a 1-hour interval. Before 2021 the DCSMv6 model was forced with Hirlam wind fields, after 2021 with HARMONIE wind fields. The data have been downloaded from Matroos.rws.nl.

The operational SWAN-North Sea model uses HARMONIE (KNMI) fields as wind forcing, which comes on a high resolution 2.5 km x 2.5 km. However, at the moment HARMONIE does not offer wind ensembles and therefore the fifty wind ensemble members of ECMWF are used. A selection of the ECMWF global ensemble forecast data can be downloaded via the TIGGE dataset (ensemble forecast data from thirteen global weather prediction centers). ECMWF-TIGGE has a coarse resolution of 0.5º x 0.5º and does not have the wind stress included as a variable. Therefore, we directly used U_{10} wind fields from TIGGE without applying a conversion to a pseudo wind field, as commonly done with the operational wave models. The wind data is

available every six hours. In this study the same 6-hour interval has been applied up to a lead time of 72 hours. In TIGGE the forecasts were available twice daily (00:00h and 12:00h), however only the 00:00h forecast was used.

Table 3.1 Overview of model input data

Period	Boundary conditions	Boundary type	Water level & Flow	Wind forcing	Storms	Missing data
Year 2022	Knmi_ecmwf_waves	2D spectra	Dcsm_v6_harmonie	TIGGE	Corrie, Malik Eunice	
Dec 2023	Knmi_ecmwf_waves	2D spectra	Dcsm_v6_harmonie	TIGGE	Pia	
Feb 2023	Knmi_ecmwf_waves	2D spectra	Dcsm_v6_harmonie	TIGGE	Swell event	
Feb 2020	Knmi_ecmwf_waves	2D spectra	Dcsm_v6_hirlam	TIGGE	Ciara	
Jan 2019	Knmi_ecmwf_waves	2D spectra	Dscm_v6_hirlam	TIGGE		
Jan 2018	Knmi_ecmwf_waves	2D spectra	Dscm_v6_hirlam	TIGGE		
Oct 2017	Knmi_ecmwf_waves	2D spectra	Dscm_v6_hirlam	TIGGE		24 to 31 10
Dec 2013	Knmi_ecmwf_waves	2D spectra	Dscm_v6_hirlam	TIGGE		

3.2.4 Model numerics and physics

Table 3.2 gives an overview of the model numerics and physics.

Table 3.2 Overview of model numerics and physics

SWAN settings	SWAN-North Sea
Maximum wind drag	cdcap=0.00275 at 30 m/s
Directional resolution	45 bins * 8°
Frequency resolution	0.03 – 0.6 Hz.
Wind drag	Hwang (2011), default for ST6
Wave generation	ST6; GEN3 ST6 5.6E-6 17.5E-5 VECTAU U10P 31. AGROW
Whitecapping	ST6; SSWELL
Quadruplets	QUAD iquad=3
Triads	-
Bottom friction	JONSWAP; cfjon=0.038
Breaking	Battjes, Jansen (1978), alfa=1.0, gamma=0.73
Numerics	First order upwind scheme; BSBT
Convergence	NUM ACCUR 0.02 0.02 0.02 98 NONSTAT mxitns=20
Bathymetry	swan-ns-j22_6-v1a_adjust
Obstacles	-
Time step	1 hr

3.2.5 Computational procedure

SWAN was run sequentially through time with the 50 ensemble members in parallel, forced by the TIGGE winds. The forecast was done daily for analysis date T00:00:00 and forecasted 72

hours ahead, because the TIGGE wind ensemble include a 72 hour forecast[1]. After the first day of the forecast, a restart file was written that was used as restart for the next analysis. Note that the unformatted restart (binary format) was used to limit space uptake. For example, for the year 2022, for 365 analysis dates each ensemble member run with a 3-day forecast.

3.2.6 Model output

Hourly output was written at 20 output locations in 'tab' timeseries files, see Figure 3.1 for the output locations. Netcdf files with spatial fields were written every 6 hours to limited space uptake. Table 3.3 shows the variables that were written as output by SWAN.

Table 3.3 Output variables

Points every 1H	Map every 6H
TIME	
XP	XP
YP	YP
HSIG	HSIG
HSWELL	HSWELL
TMM10	TMM10
TPS	TPS
DIR	DIR
FDIR	FDIR
DSPR	DSPR
WATLEV	WATLEV
VEL	VEL
WIND	WIND

[1] A longer prediction horizon was not chosen, as the prediction horizon of the SWAN North Sea models is only 48 hours.

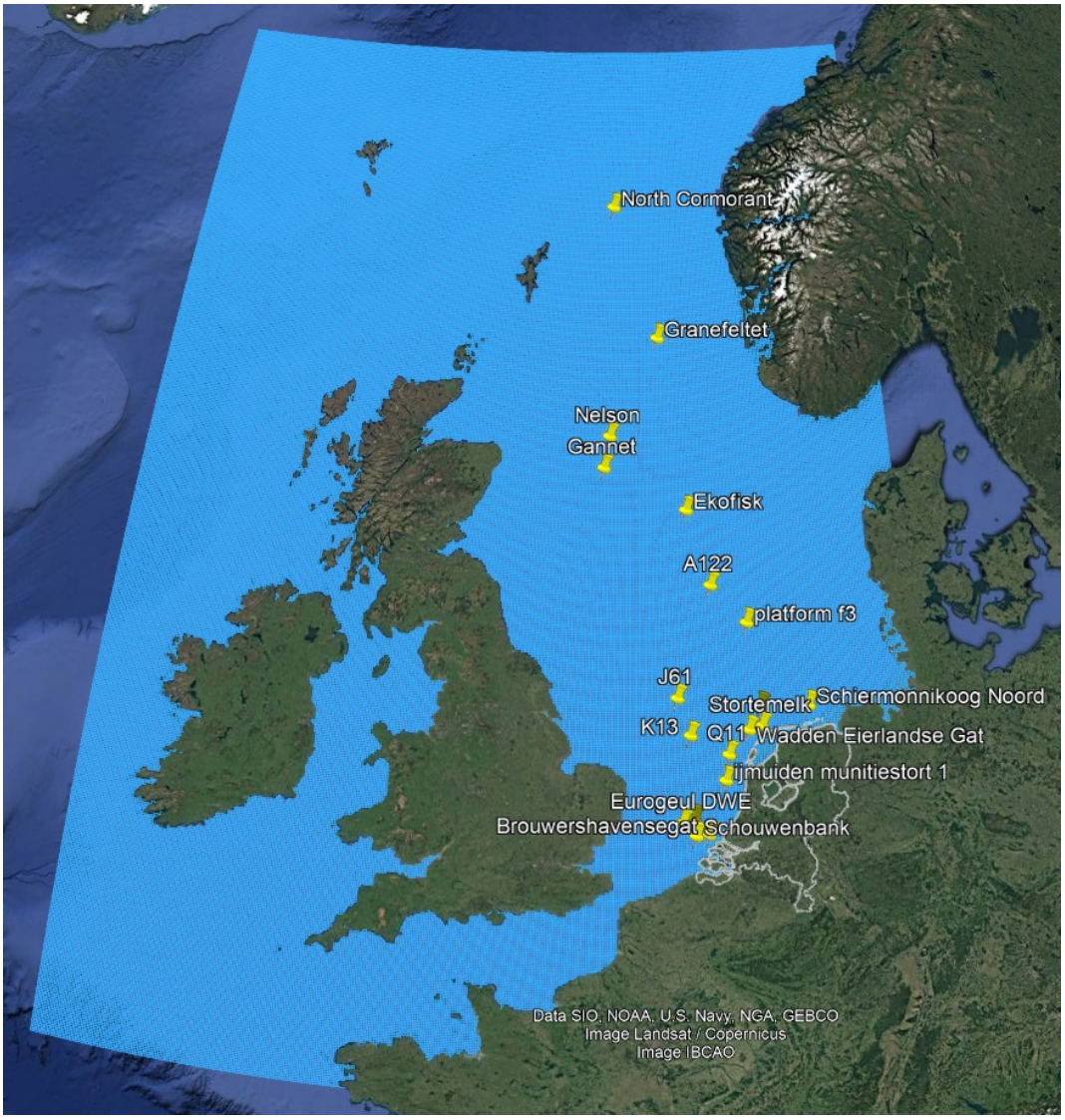


Figure 3.1 Output locations points

3.2.7 HPC (Snellius) workflow and tools / Code

The ensemble runs were executed on high-performance computer Snellius (SURF https://www.surf.nl/en/services/snellius-the-national-supercomputer). A toolbox (GitHub: swanprobpy) was developed in Python to download input data from MATROOS (SwanToolbox), generate all SWAN configuration files for all ensemble members and all analysis dates. A bash script (jobfile) submits the forecasts for all ensemble members in parallel and sequentially in time. The routine also makes sure that the next forecast uses the output of the previous forecast as a restart. After the restart files are used, the routine removes them to limit the storage. The workflow for setting-up the model configuration structure (and some processing) can be followed by using the notebooks, as available in GitHub. We used naming conventions for the directory structure for input and output that need to be followed, as explained in the notebooks.

In addition, postprocessing scripts were developed to generate timeseries and ensemble verification diagrams like Talagrand/rank histograms and exceedance histograms for the wind speed and wave variables. These are used to analyse how well the Ensemble Prediction System (EPS) grasps the uncertainty in the system.

3.2.8 Computational aspects on Snellius

Project space

The project storage space only for the SWAN model input and ensemble output for one full year was 1.3TB, given that the restart files are removed after they are used in the previous forecast and if they have the unformatted file extension. Output point files were written hourly, while map files were written every six hours. Note that this is excluding the storage needed for training the surrogate model, which approximately requires double the storage of SWAN data.

Type	Project Space
Full year	1.3 TB
Months of storms (7x)	1 TB
Tests (scaling)	0.25 TB
Total SWAN	~ 2.6 TB

Scaling tests

In order to take the most efficient approach in running the ensemble in parallel on Snellius, we conducted scaling tests for one month. The scaling is approximately linear at first, but efficiency decreases due to several reasons (e.g. overhead communication). The most efficient choice within the available SBUs (total cores x time), was using 10 nodes and 37 cores per task (ensemble member).

Setup	Job wall clock time	Selected
Jan2022_2nodes_7cores	01:26:05	
Jan2022_5nodes_19cores	00:58:31	
Jan2022_10nodes_37cores	00:29:05	x
Jan2022_12nodes_46cores	00:34:41	
Jan2022_17nodes_64cores	00:25:48	

Computation

We ran 50 ensemble members on 10 Genoa nodes, with 192 cores on each node. We assigned 37 cores per task (SWAN ensemble member). Job wall clock time for one full year was approximately 80 Hours (3.3 days). Because the sbatch routine loops through each ensemble member per forecast, we encountered job step limit issues. Therefore, each run had to be restarted after two months. The amount of SBU's required for the full year was approximately 160.000. To have some extra room for tests we requested 300.000 SBUs.

3.3 Ensemble Verification

We have a reasonable representation of uncertainty by using TIGGE wind dataset, but not optimal. Figure 3.2 shows an example of the ensembles (light blue), the ensemble mean (dark blue) and the measurements (black dots) for the wind and wave height during storm Corrie (Jan/Feb 2022). The measurements are not always within the ensemble range for both wind and waves. We observe that the ensemble spread grows with forecast time and that the observed values start to fall within the ensemble spread. Not primarily because the uncertainty in the wind is accurately modelled, but because the ensemble forecast gets more diffuse through time. Thus, it is likely that errors in the wind field dominate after a few days lead time, while other sources of uncertainty dominate the errors on short lead time. Additional comparisons can be found in Appendix A.

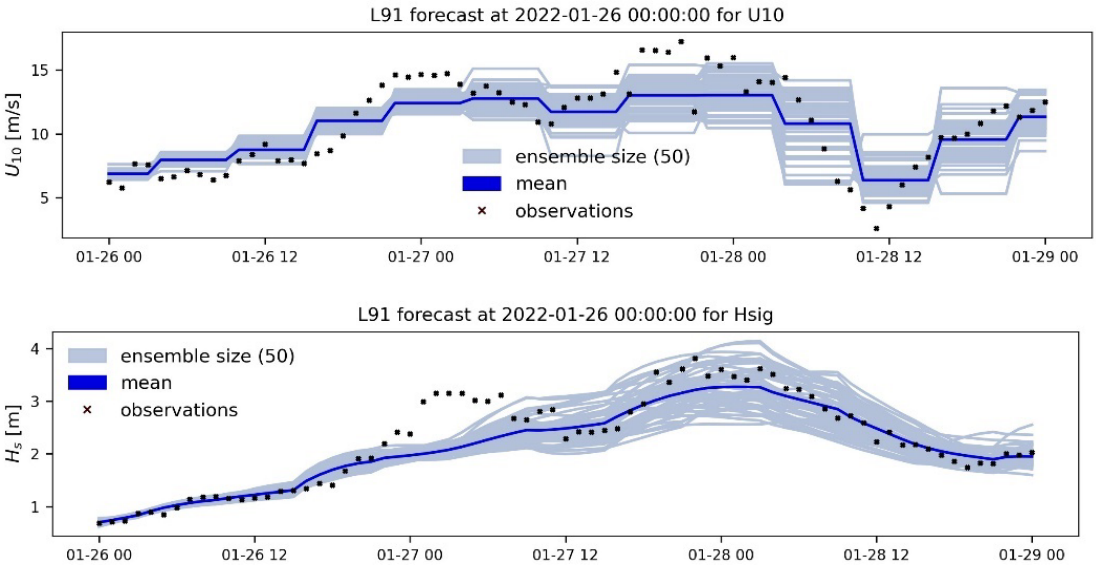


Figure 3.2 Timeseries at L91 for U_{10} and Hs. H_s has been computed with the SWAN model.

To give an indication of how reliable the ensemble forecast is compared to observed data, rank diagrams (also called Talagrand diagrams) have been made. A rank diagram is a type of histogram. However, the bins are based on a ranked list of forecast values. Subsequently, observations are placed in the appropriate bin. In the ideal situation, the rank diagram should have a uniform distribution as each bin represents equally likely scenarios.

Figure 3.3 shows the Talagrand and exceedance diagram for significant wave height for the station L91, for which time series are shown in Figure 3.2. Note that the ensemble verification is performed based on 72-hours lead time. The rank diagrams based on the ECMWF ensemble wind and wave data, see Figure 3.3 and 3.4, have a u-shape. The u-shape suggests that the ensemble spread is too small. At some stations the asymmetry points to a model bias. This is indicated by the asymmetry of rank 0 and rank 50, also visible in Figure 3.2. However, at the same time, the exceedance diagrams, see for example Figure 3.3, show that most of the observations are still close to the ensemble.

The mean Talagrand diagram over stations that have matched observations are shown in Figure 3.5 for significant wave height. The mean is taken over stations: North Cormorant, A122, Q11, L96, Ijmuiden munitiestort and Europlatform. It can be observed that most observations are larger than the largest ensemble member, indicated by bin of rank 50. SWAN tends to underestimate the significant wave height more than overestimate. A similar pattern can be observed for wind speed in Figure 3.6, although the bias is less strong, indicated by the more uniform u-shaped bins. For wind speed the mean over stations: L91, Q11, Platform k13, Platform F3 and North Cormorant were taken, based on observation availability.

Nonetheless, looking at the Talagrand diagrams for the wind we can conclude that our wave ensemble is not optimal at least partly because of the limitations of the TIGGE-ECMWF wind ensemble. For example, because of the coarse spatial and temporal resolution of the wind ensemble. Another important cause of the non-optimal ensemble is that we are only considering the wind forcing as a source of uncertainty, while in reality there are several more sources (e.g. model parameters, boundary conditions). Note that we evaluated the ensemble for 3-day lead time forecasts. Errors will become more dominant with increasing lead time. To get a more detailed overview, ensemble verification can be done for different lead times in the future (e.g. forecast for 1 day, 2 days and 3 days ahead).

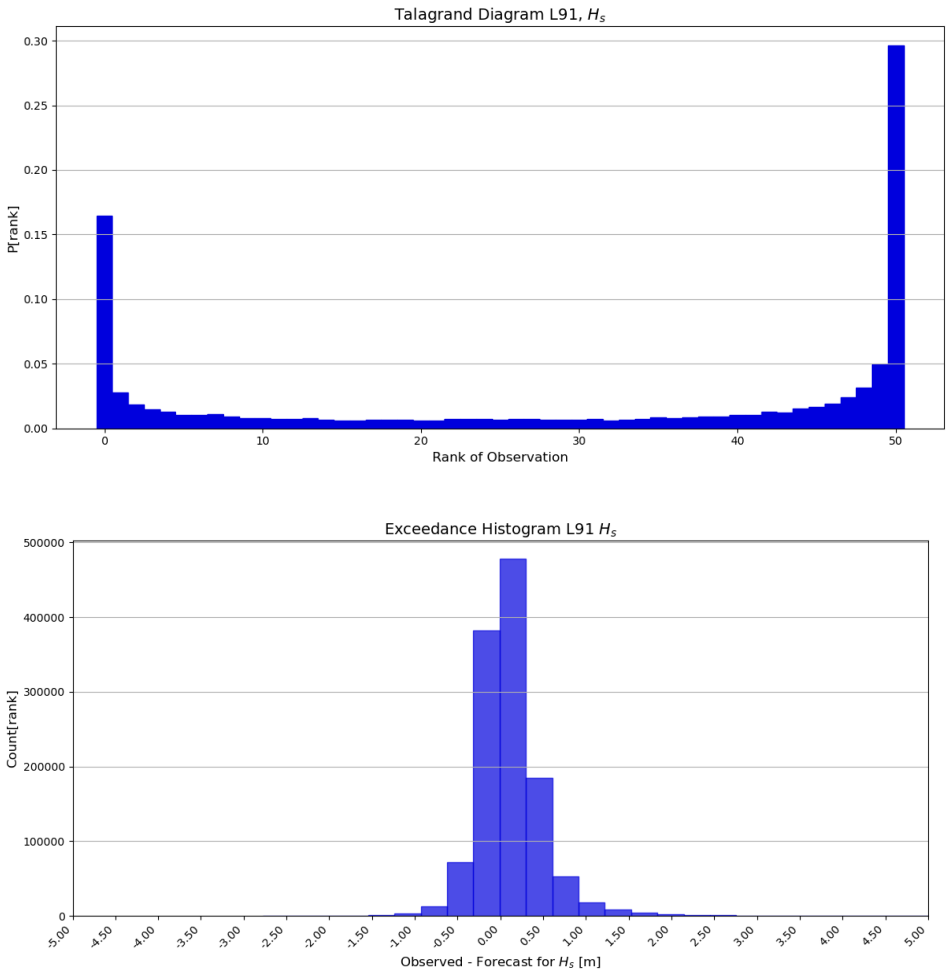


Figure 3.3 Talagrand and exceedance histogram for significant wave height at station L91 for all members. Exceedence in meter

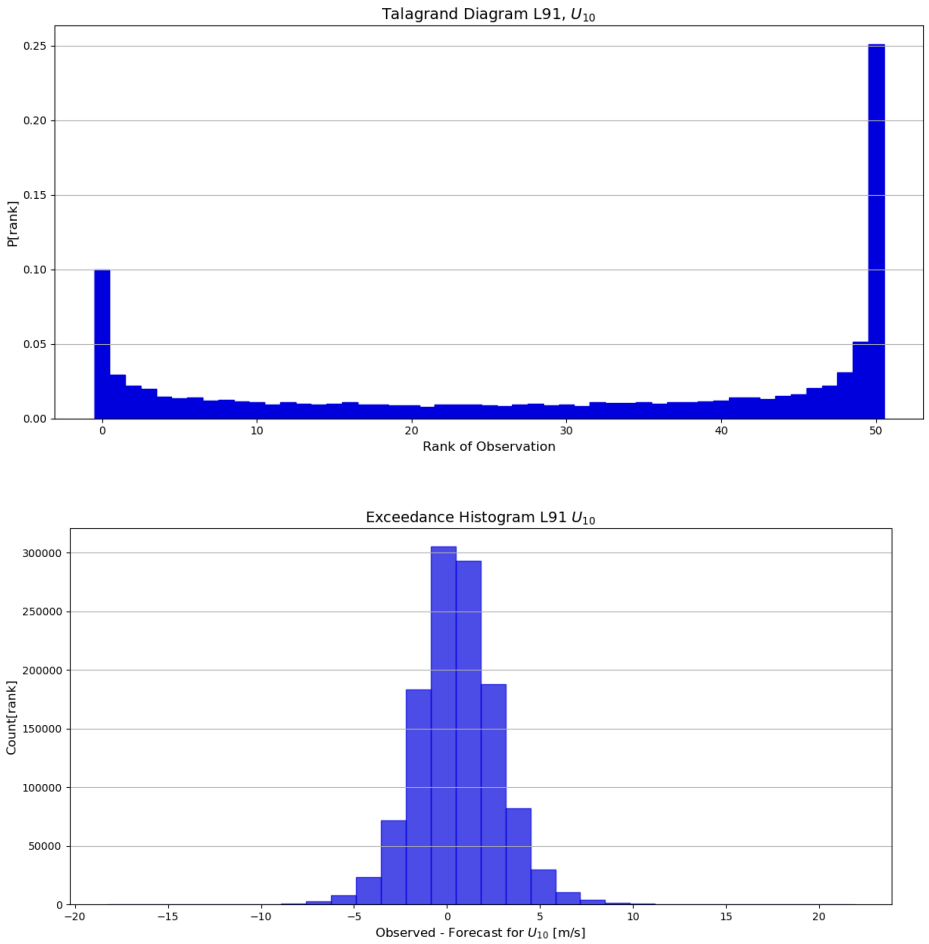


Figure 3.4 Talagrand and exceedance histogram for U_{10} wind speed at station L91 for all members.

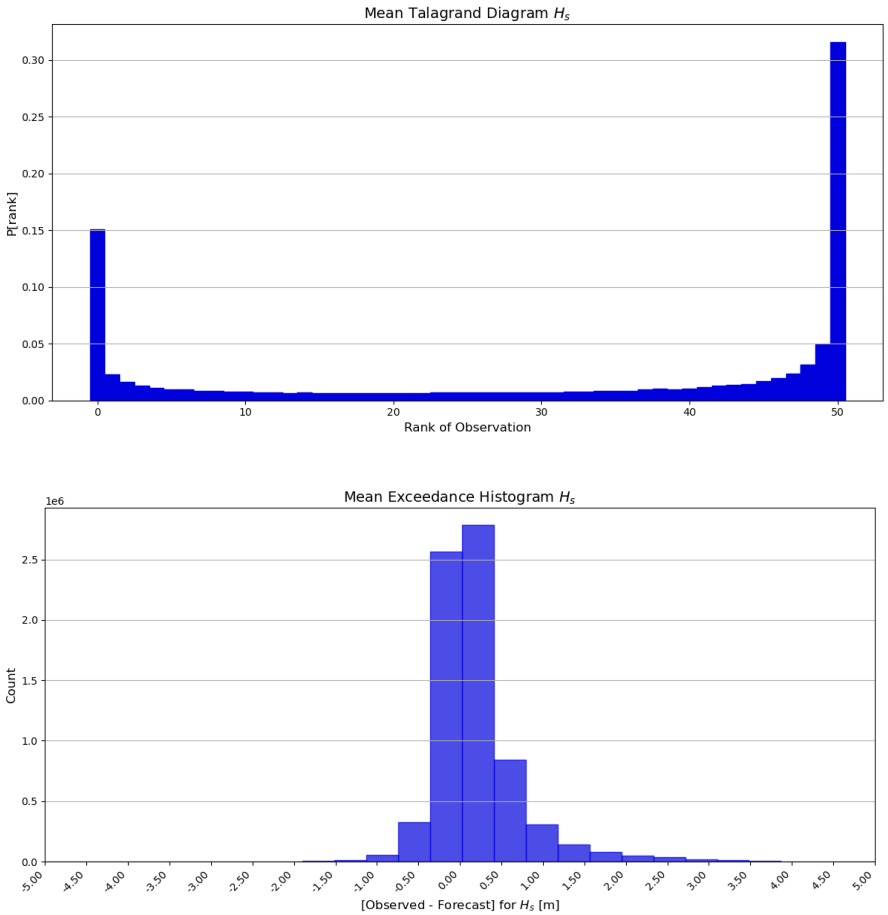


Figure 3.5 Mean Talagrand and exceedance histogram for significant wave height Hs over all stations.

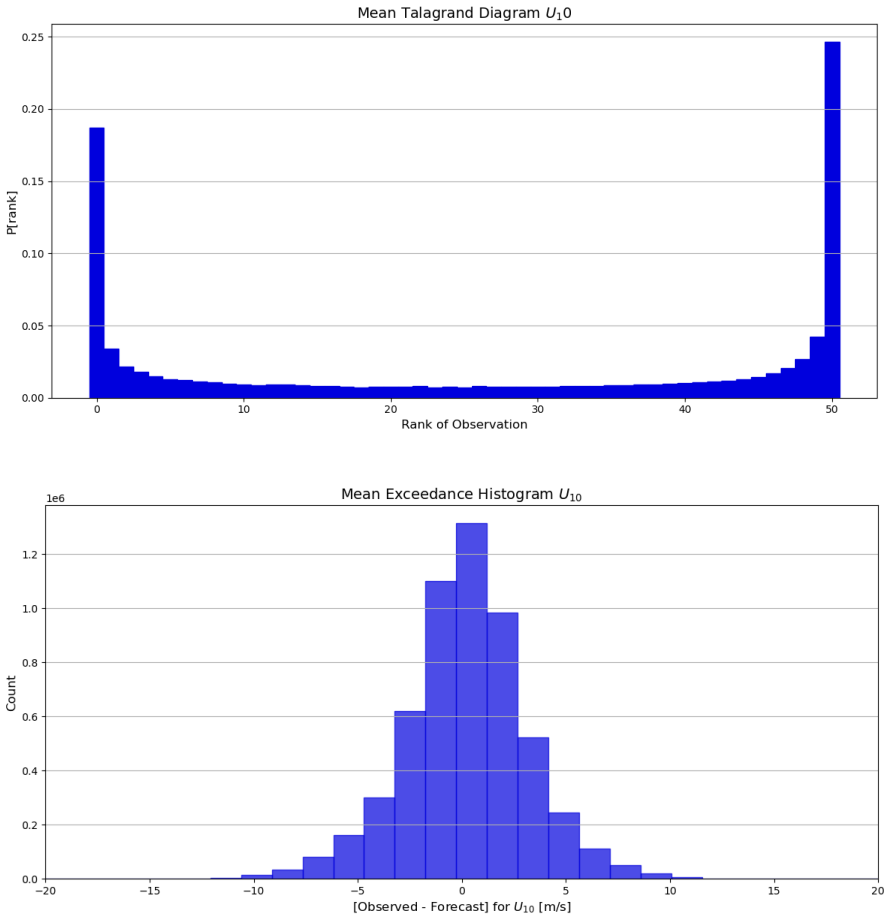


Figure 3.6 Mean Talagrand and exceedance histogram for U10 wind speed over all stations.

4 TurboSWAN surrogate model

4.1 Introduction

This chapter describes the TurboSWAN model, a fast neural network that is trained on a large set of SWAN runs. In future, the goal is to use this TurboSWAN model to operationally perform 50 ensemble runs with wind ensemble input and apply the spread in results as confidence interval on the one deterministic SWAN run.

4.2 Introduction machine learning model

4.2.1 Neural networks

Neural networks, or artificial neural networks, are inspired by the workings of biological neurons. Neural networks consist of an input layer, where it takes multiple inputs, one or more hidden layers and an output layer. Each layer is composed of nodes, or neurons, which are connected to the nodes of the subsequent layer. Each node has an assigned weight and threshold. The weight of a node determines the importance of that node in predicting the output, while the threshold manages the activation of the node. If the output of a node is above the threshold for that node, its output is sent to the next layer of the network.

Once an input layer is created, weights are assigned to each input in that layer. All inputs are multiplied by their respective weight and then summed. Next, the output is passed through an activation function, which gives the output of the layer. If this output exceeds the threshold, the output is passed to the subsequent layer. For a node with a threshold c receiving *n* inputs we have the following

$$f(x) = \begin{cases} \boldsymbol{z}, & \text{if } \boldsymbol{z} = \sum_{i=1}^{n} w_i x_i - c \geq 0, \\ 0, & \text{if } \boldsymbol{z} = \sum_{i=1}^{n} w_i x_i - c < 0, \end{cases}$$

where w_i is the weight of input x_i and $f(x)$ is the activation function of the node. This function is called the ReLU function and is commonly used in neural networks and can also be written in the simpler form $f(x) = \max(0,x)$. Two other common activation functions are the linear activation function $f(x) = x$, though this is usually only used in the final layer, and the sigmoid activation function $f(x) = \frac{1}{1+e^{-x}}$. The last layer in the network, the output layer, combines the outputs of the previous hidden layer, again using an activation function, to obtain a function which maps the input space into the output space. This is done by minimizing a cost or loss function, like for example the mean squared error. The more hidden layers the network has and the bigger the size of these layers, the better it is able to learn complex relations.

In general, there are two types of learning, namely supervised learning and unsupervised learning. If the cost function minimized during training is a function of the output of the network and the real output, it is called supervised learning. If the cost function is not dependent on the real output, it is called unsupervised learning.

The training process of a network, in this case supervised, works as follows. First, the weights are initialized. The weights are then slightly adjusted in each step to get closer to the desired output. For each input, the activations are propagated through the network, where the activation state is computed for each unit in the network, including the output. At the output layer, the output from the network is then compared with the desired output and the cost is computed, using the cost function. Then the partial derivative of the cost with respect to the activation is computed to check how much the cost can be reduced with a slight change in the activation. This partial derivative is then propagated back through the network to the activations

of the previous layer. As the activations are dependent on the weights, since f(x) = f(**wx**), the derivative of the cost with respect to the weights can be computed using the chain rule. With the help of this derivative, the weights can be adjusted in the direction that reduces the cost. This also shows why it is preferred for the activation function to have a non-zero derivative, as otherwise the derivative would vanish, causing troubles for the weight adjustments.

4.2.2 Convolutional Neural Networks

Convolutional Neural Networks, or CNN's in short, are a special type of artificial neural networks that excel in pattern recognition. Its main advantage over classic artificial neural networks is that it greatly reduces the number of parameters, allowing the model to solve larger problems. With this property, it is mainly used for image problems. The network typically takes an image with shape $H \times W \times D$ as input. This input image is then traversed by a filter, or kernel, with a shape of $f \times f \times D$, performing matrix multiplications with the part of the image the filter highlights at the time. The outcome of this multiplication is summed and then projected onto the feature map, which is passed to the next layer.

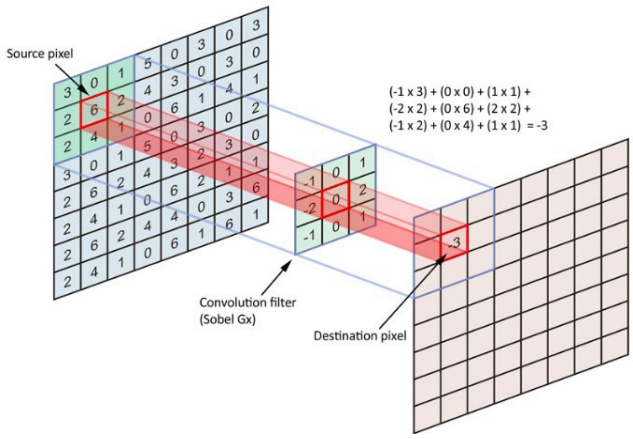


Figure 4.1: The operation of a 3×3×1 kernel on a 8×8×1 input image (Cornelisse n.d.).

This can be seen in Figure 4.1, where we have an image of shape 8×8×1 and a kernel of size 3×3×1. In this case, the kernel is at the top left where it performs a matrix multiplication with the highlighted part of the image. The outcome of this action, which is a vector, is then summed to get a scalar value, which is then projected onto the feature map. The filter starts at the top left corner of the image and moves over the image to the right with steps equal to the so-called stride value until it reaches the boundary of the image. After it reaches the boundary of the image it starts at the left of the image again and moves down with a stride value, which is typically the same stride value. It then moves to the right again and repeats this process until the entire image is traversed. After the last position of the filter, it will start at the left again, 1 below the original position (Albawi et al. 2017).

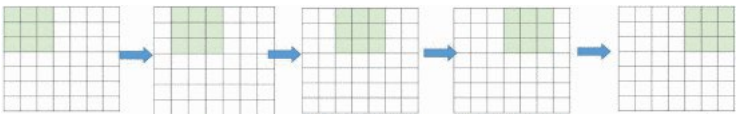


Figure 4.2: A kernel which traverses the image with a stride of 1.

Figure 4.2 shows how the filter traverses the image if it has a stride of 1. However, considering that we have a 7×7 image with a 3×3 filter and a stride of 1, the output image will have a reduced size of 5×5. This effect can also be seen in Figure 4.1, where the output of the first filter is projected on position (1,1) instead of position (0,0). Generally, if we have an image of

size $N \times N$ and a filter with size $F \times F$ with stride S, the output image will be of shape $O \times O$, where $O = 1 + \frac{N-F}{S}$.
Therefore, to allow the output image to retain its original shape, zero-padding is used on the input image. With zero-padding, the input of size $N \times N$ will be increased to $(N+2) \times (N+2)$ and the boundaries will be set to zero. This will not only make sure the images do not decrease in size, it also helps to prevent information loss at the boundaries.

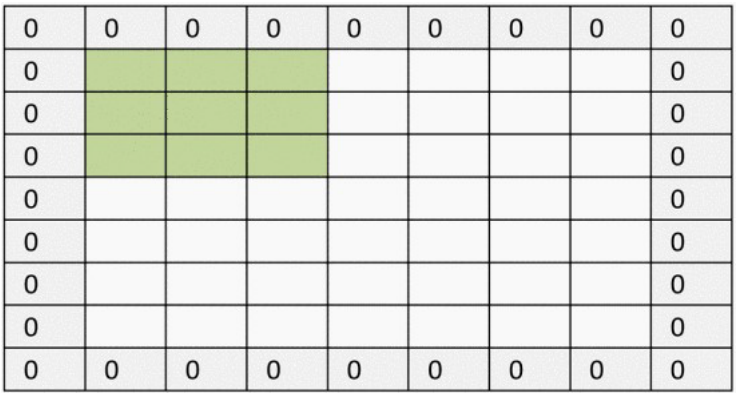


Figure 4.3: Zero-padding on an image of size 7×7 (Albawi et al. 2017).

Lastly, we would like to consider pooling layers. Pooling layers down-sample the input image, which reduces the complexity of the image for subsequent layers. The advantage of this is that the model will become easier to train as it reduces the number of parameters that need to be trained. There are two types of pooling, namely max-pooling, which is the most common, and average-pooling. In both pooling types, the input image is partitioned into smaller rectangles. If max-pooling is applied, the maximum of each sub-rectangle is returned, while for average-pooling the average is returned. The most common size for the sub-regions is 2×2 with a stride of 2. This means that both the width and the height of the image get halved in the pooling-layer, which can also be seen in Figure 4.4 where max-pooling is used.

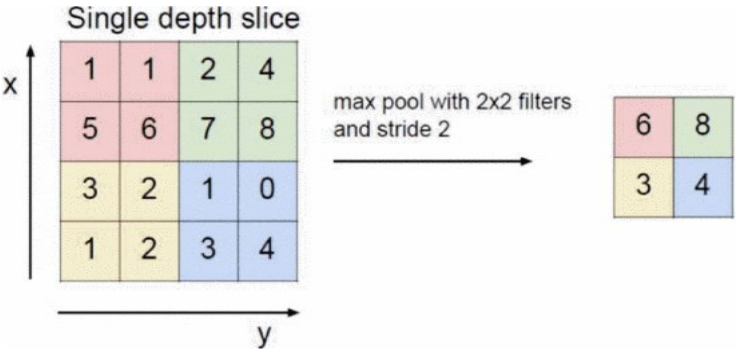


Figure 4.4: Max-pooling with a size of 2×2 and a stride of 2 (Albawi et al. 2017).

4.3 Training, validation and testing data

TurboSWAN was trained on the SWAN data with time steps of 6 hours. We used this time step of 6 hours to make sure the input fields are different enough to provide new information between the time steps, while still showing a temporal correlation. We chose a combination of a selection of input fields and boundary conditions for SWAN as input data for TurboSWAN. For the input fields, we used

- xwnd: x-component of wind velocity [m/s]
- ywnd: y-component of wind velocity [m/s]
- ssh: sea surface height [m+MSL]

- xcur: x-component of current velocity [m/s]
- ycur: y-component of current velocity [m/s]

These fields are of the 'current' time step, in other words, they refer to the same time step as the output fields we are interested in. Additionally, we included the fields of the time step before the current time step as well (6 hours earlier), to provide additional information to the network. The fields of the SWAN computations are of shape (481×421), which means that after concatenating both time steps, we feed the network input fields of shape (481×421×2).

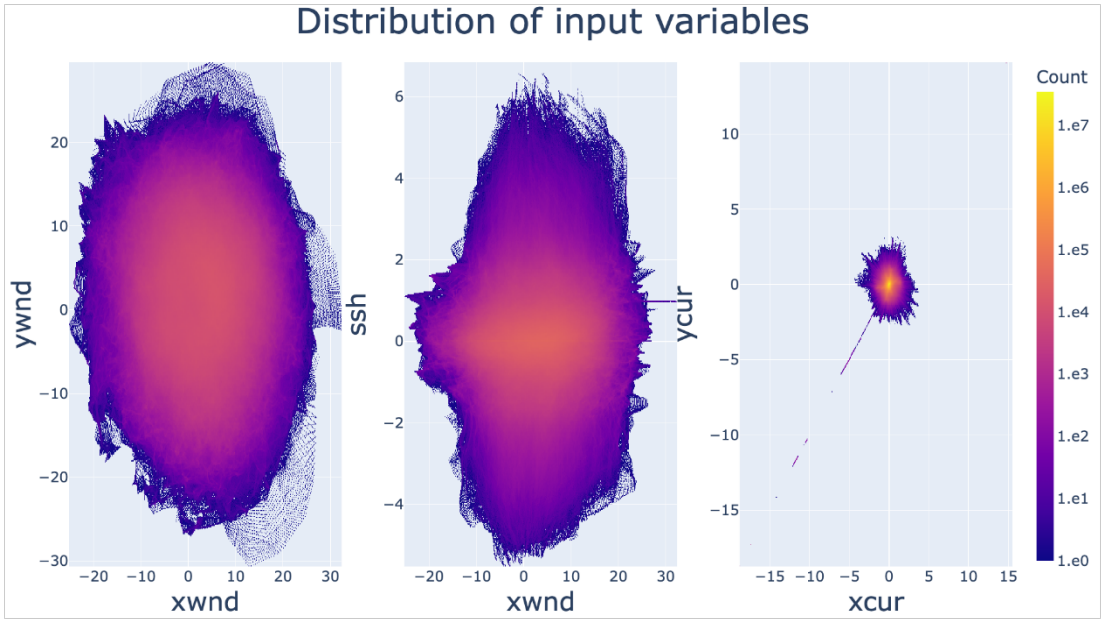
We combined all five variables into one input for the model, leading to a shape of (481×421×10) for the field input to TurboSWAN. For the boundary conditions, we included the boundary conditions of the current time step for the variables hs,hswe and tmm10. We had 22 boundary values for all three variables and concatenated them together, resulting in a shape of (22×3) for our second input.
For the output fields we chose:
- hs (significant wave height),
- theta0 (wave direction),
- tmm10 (spectral period Tm-1,0),
- tps (smoothed peak period),
- hswe (swell wave height or HE10).

Due to the periodic nature of theta0, we decomposed the field into its x and y component, using the following transformation:

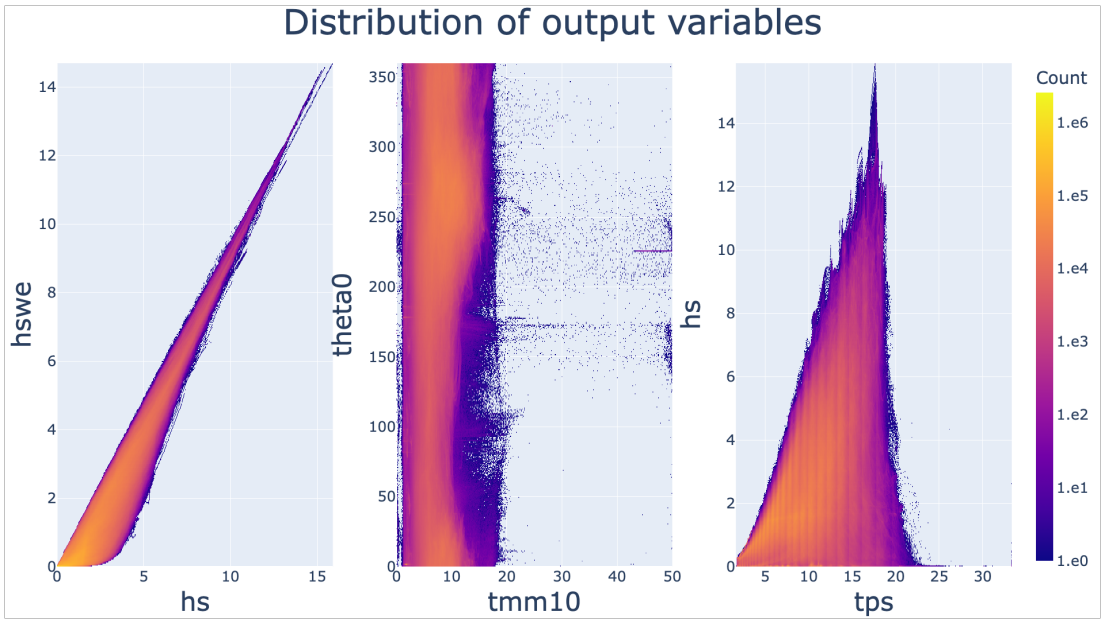
$$theta0_x = \cos(theta0),$$
$$theta0_y = \sin(theta0),$$

where theta0x and theta0y are the x- and y-components of theta0 respectively. We performed this transformations to deal with the limitations of neural networks in dealing with periodicity. Due to the periodicity, we have 0 ≡ 2π for theta0. Let us assume that theta0 = 0 at a point in the data and that the network overpredicts slightly. Then we want to have an equal loss for this over prediction at 0 as for an overprediction of the same amount at 2π. However, if we do not decompose the angle, then an overprediction at 2π would result in a higher loss, since it is being compared to 0. We also concatenated all output variables, leading to an output shape of (481×421×6).
The dataset consists of 25 ensembles of the year 2022 supplemented with 25 ensembles of the seven storms to add more extreme cases to the training data, both with a time step of 6 hours. Combined, this leads to a dataset of about 175,000 samples, of which about one third is storm data. We split this dataset randomly into training data and validation data, with 70% training data and 30% validation data. For testing, that is all plots and statistics produced to test the trained model, we used the full year 2022 and all storms combined of one of the ensembles not used during training and not present in the validation dataset.
In Figure 4.5 we show the distribution of the distribution of the input and output variables in the test dataset for 2022 and all storms combined. We plotted a scatter plot for pairs of input variables against each other in addition to pairs of output variables against each other. This is done for all time steps in 2022 and the storms and for all points in the fields. We can see a fairly uniform spread for the wind variables, where we do not introduce a bias in the network by having a skewed representation for xwnd and ywnd. Also for ssh, we do not have a big noticeable bias, with an even spread for both the positive values and the negative values. For the current we again see that the values are centered around 0. However, we do notice a linear line for both xcur and ycur negative, which indicates that there is an error in the input data. Though, the error is not too significant, since this is only a small percentage of the data. The

scatter plots for the output again show interesting insights. The wave height hs is shown to have a linear correlation with the swell hswe, which is to be expected. We have a fairly uniform spread that covers most of the wave directions. For tmm10, we have a few unrealistic outliers with values near 50. Lastly, the plot for the wave height against the period shows expected behaviour, where we do not have low periods for higher wave heights.



(a) Input



(b) Output

Figure 4.5 The distribution of input variables against each other and of output variables against each other in the test dataset for 2022 and all storms combined.

We normalized all input variables and output variables to the range of [0, 1] for the training of the network. Doing so makes sure that variables with high values, like theta0, do not have a higher impact on the loss than variables with comparatively lower values, like hs.

4.4 TurboSWAN

The neural network for the TurboSWAN model consists of an encoder part and a decoder part. Note that here the encoder and the decoder refer to the first half and second half of the network respectively. The encoder learns the patterns from the different input parameters and maps them to a latent space. The decoder will then learn to construct the output parameter maps from this latent space. In addition to the encoder-decoder structure, the network also has skip connections between the encoder and the decoder. When a neural network is very deep and has a lot of layers, as is the case with the network for TurboSWAN, information learnt in the earlier layers of the network can get lost due to vanishing gradients. To try to retain some of this information, we included these so-called skip connections between the different layers in the encoder and in the decoder. This type of network structure with skip connections is typically called a U-net, due to its shape, and has proven to be successful in retaining the information of low- and high-level features present in the data (Ronneberger et al. 2015).
To further prevent information loss due to the depth of the network, we also used residual blocks in the network. Residual blocks were introduced in the ResNet model to address the issue of vanishing gradients in deep neural networks (He et al. 2015). In Figure 4.6 we show the concept of residual blocks, as well as the structure of the residual blocks that we used for TurboSWAN. In our case, the blocks consist of two convolutional layers with w filters per layer. Formally, if we assume that we are interested in learning the function H(x), then the block will try to learn the mapping F(x) = H(x)−x. This means that we are now trying to learn F(x)+x instead.

The idea behind this is that, assuming that we have a shallower network with fewer layers, the deeper network would have to have multiple identity mappings to end up with the same result. To end up with the identity mapping from multiple layers turns out to be a difficult task. So instead, by formulating the problem like this, the block or part of the network needs to learn F(x) = 0 to end up with an identity mapping. This is a much easier task for the network, since it only needs to drive the weights to 0.

The network, which we present in Figure 4.7, consists of a combination of residual blocks, downsampling blocks, and upsampling blocks. In addition, we use multiple dense layers to combine the image input with the input from the boundary conditions. The downsampling blocks, shown in Figure 4.8, use three residual blocks followed by a single convolutional layer with a stride of 2. This convolutional layer serves the same purpose as the typical max-pooling layer, namely downsizing the image by reducing both dimensions by halve. However, where a max-pooling achieves this result by choosing the maximum number in small blocks of (2×2) cells, a convolutional layer with a stride of 2 does not necessarily use the maximum, but learns down-scaling instead. This approach can lead to better downsampling, since the network now tries to learn this process, though it will also come with a higher computational cost, since we introduce more learnable weights into the network. After each residual block, we have a skip connection to an upsampling block at the same level in the network (Figure 4.7). The upsampling block itself, which we show in Figure 4.9, has a structure similar to that of the downsampling blocks. It again uses three residual blocks, though these residual blocks are made of two deconvolutional layers instead of two normal convolutional layers. Likewise, the upsampling is done using a deconvolutional layer with a stride of 2 instead of a typical upsampling layer. The skip connections from the downsampling block are connected and concatenated to the block output and the layer output from the upsampling layer.
So far, we have been focusing on how we dealt with the image input variables and the image output variables. However, since the boundary conditions are not two-dimensional in shape, this approach cannot be used for them. Instead, we combine the boundary conditions for the three different variables into one long 1-dimensional array. Then, this array is passed through four dense layers. To attach it to the rest of the network, we first flatten the final layer of the encoder. The flattened layer is then concatenated to the output of the last dense layer for the

boundary conditions. This concatenated result will then pass through two more dense layers after which it is reshaped to a 2-dimensional image again, of the same shape as the image before flattening. After the reshaping, the output will go through the encoder. We show a schematic overview of the middle part in Figure 4.10.
Lastly, we mention the use of a zero padding layer and the use of a cropping layer at the start of the encoder and at the end of the decoder, respectively. Since the input and output for SWAN is of shape (481×421), it is not divisible by 2. This means that we are not able to directly halve the dimensions of the images for downsampling. To solve this issue, we used a zero padding layer, adding zeros to the boundaries, leading to a new shape of (512×512). We choose the specific value of 512 for the reason that it is a power of two, making subsequent downsampling operations possible as well. To get back to the original dimensions of the output, we crop the image in the cropping layer. We would like to note that we used batch normalization after every layer in the network, to help prevent the network from overfitting. The full network contains a total of more than 442 million trainable parameters.

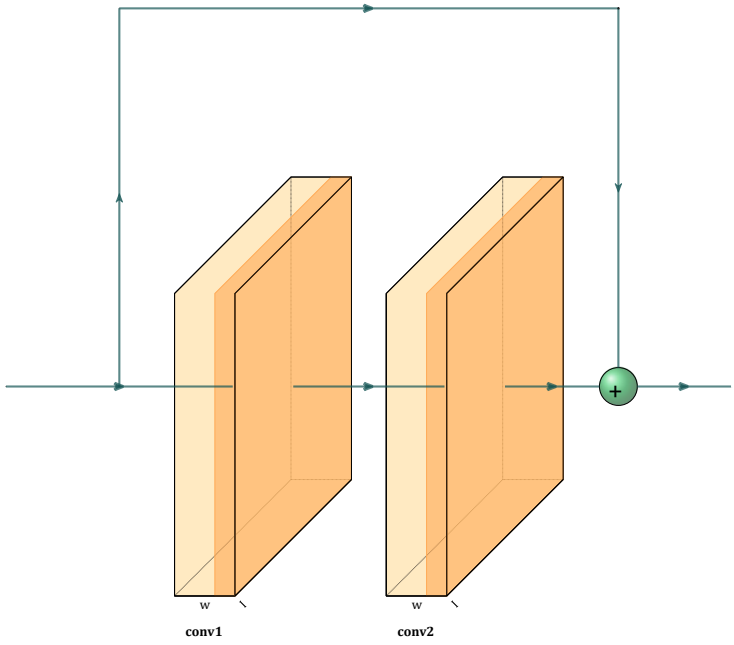


Figure 4.6: The structure of the residual blocks used in the TurboSWAN network. The block consists of two convolutional layers. The input to these layers is added to the output of these layers.

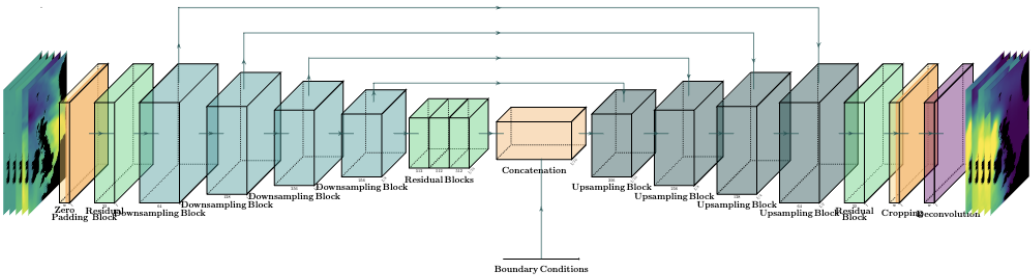


Figure 4.7: The architecture of the TurboSWAN network. The network uses multiple downsampling and upsampling blocks, combined with residual blocks. In the middle of the network the boundary conditions are added to the flattened image input.

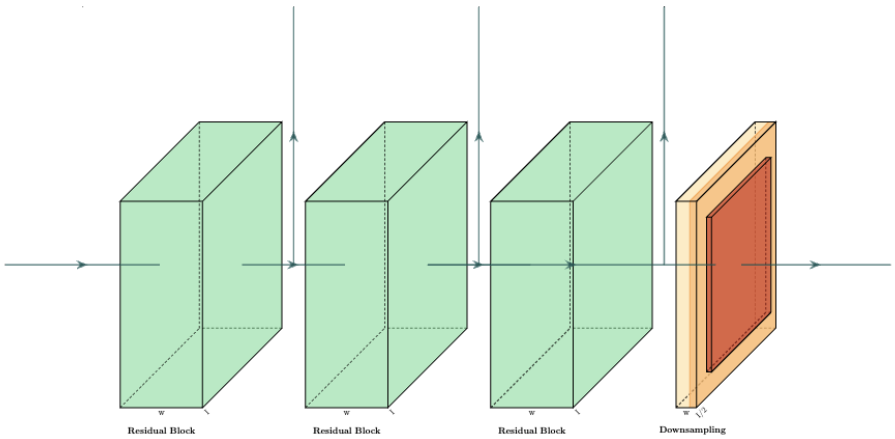


Figure 4.8: The structure of the downsampling blocks used in the TurboSWAN network. The block consists of three residual blocks and has a convolutional layer with stride 2 at the end. After each residual block, there is a skip connection to an upsampling block further down the network.

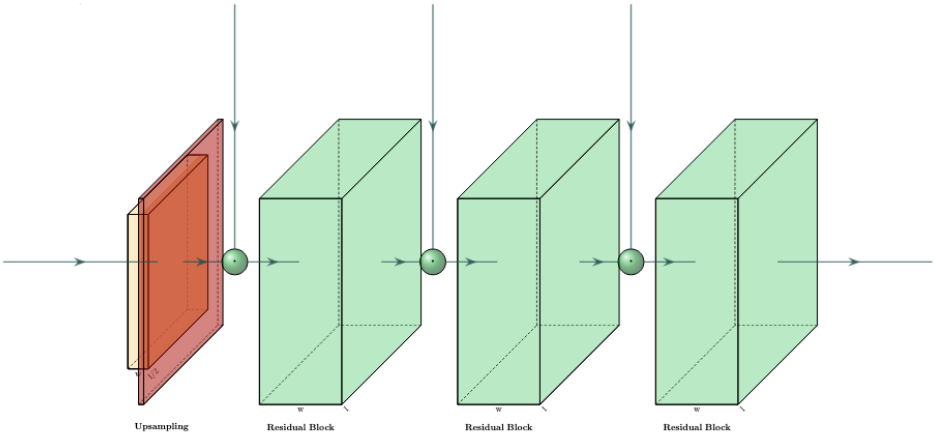


Figure 4.9: The structure of the upsampling blocks used in the TurboSWAN network. The block consists of three residual blocks and has a deconvolutional layer with stride 2 at the start. Before each residual block, the output of a residual block in a downsampling block earlier up the network.

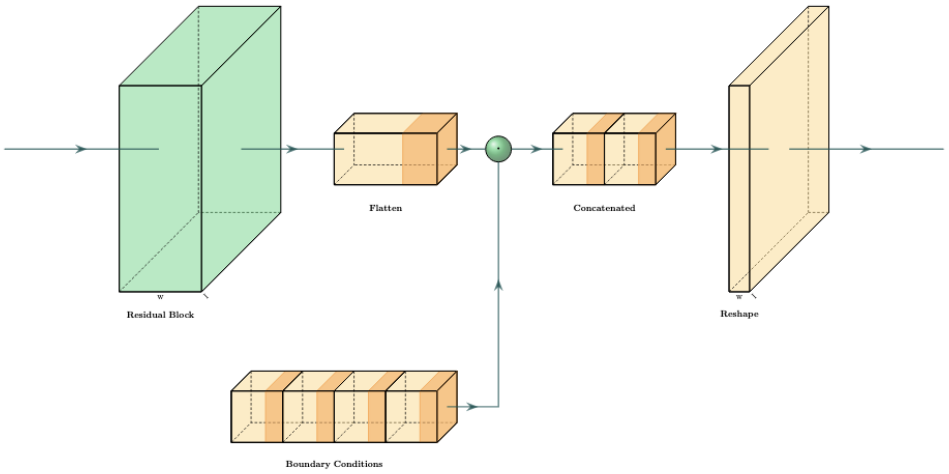


Figure 4.10: The structure of the concatenation of the boundary conditions to the image input at the middle of the TurboSWAN network. The boundary conditions pass through four dense layers before being concatenated to the flattened image data.

4.4.1 Network architecture choice

We chose a U-net based architecture with residual blocks for its proven capabilities in pattern recognition and reduction of information loss for deep networks. Another option for the architecture is to use a convolutional LSTM Shi et al. 2015.
Convolutional LSTMs are often used in problems with a temporal component in addition to a spatial component. Since we do have a temporal component in the input, it might be suitable for our problem. However, our problem does not necessarily predict the next time step. We go from a set of input fields, containing two timesteps, to a set of different output fields. So, we think it is more important for the network to properly learn the mapping from the input fields to the output fields, which is why we opted to not use a convolutional LSTM. We also considered to separate the different input fields as well as the output fields, instead of combining them all into one input and one output. Our choice for the latter was for the sake of keeping the model simpler and trying to reduce computational and memory requirements. Moreover, we did a short analysis on the number of layers and number of filters per layer, which lead to our current architecture, which performed best in the analysis. Additionally, this analysis led to a learning rate of 1e-4. For the loss function, we chose to use the mean squared error (MSE), which punished large deviations from the truth. We also tested the use of the root mean squared error, which has a lower punishment for large deviations, but this leads to similar results. Lastly, the network used the ReLU activation function for the hidden layers and a linear activation function for the output layer.

4.5 Comparison to SWAN results

After training the network for 47 epochs, which means the network went through the entire training and validation dataset 47 times, the network seems to have converged. Figure 4.11 shows the progress of the loss for both the training data and the validation data during training. In Figure 4.11, we can see that both losses already converged after 20 epochs, after which they barely decrease any further. In fact, the validation loss slightly increases again after epoch 30, indicating that the network is overfitting. The training loss reaches a final value of MSE ≈ 0.0063, whereas the validation loss has a value of MSE ≈ 0.024 after the final epoch. Note that since the output contains all output variables, the unit of the overall MSE is unit-less.

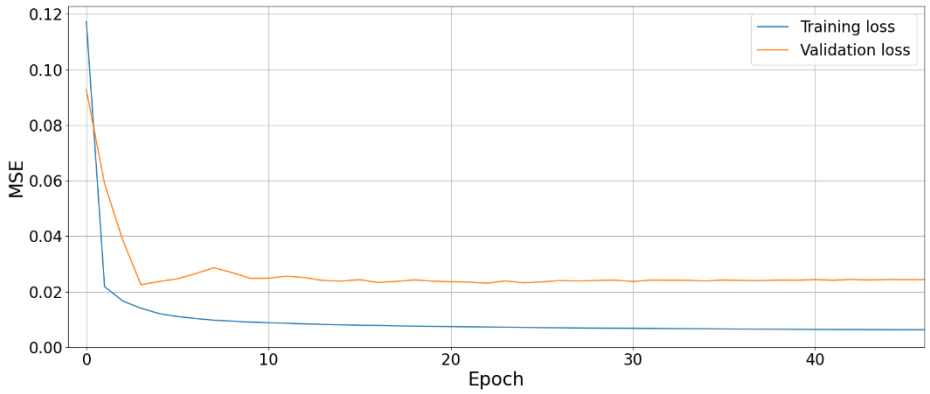


Figure 4.11: The training and validation loss of the network during training

To get a better grasp of the performance of TurboSWAN, we tested it on the test dataset. Figure 4.12 shows the RMSE taken over the whole year and the seven storms of the test dataset for the variable significant wave height (hs), swell wave height (hswe) and spectral wave period (tmm10). We also show a figure (Figure 4.14) of the mean values of these variables across the whole year, to get a better indication on how TurboSWAN performs. From Figure 4.12 we observe that for all three variables, TurboSWAN performs best in the southern North Sea region. Both hs and hswe have an error of about 0.5 m, though the error for hs is slightly lower. tmm10 has an error of approximately 1.5 s. All three variables also clearly have higher errors along the coastal areas as well as near the northern boundaries of the North Sea. This is especially apparent for tmm10. If we look at the mean errors in Figure 4.13, we see that TurboSWAN primarily under predicts for hs and hswe between Great-Britain and Norway. This is contrary to the south of Ireland, where we see on average a slight over prediction. For tmm10 we again have slight under prediction between Great-Britain and Norway, though not as large as for the other two variables. The mean errors also reveal that TurboSWAN has a substantial under prediction along most coastal areas with the exception of the area north of the Netherlands, where it heavily over predicts.

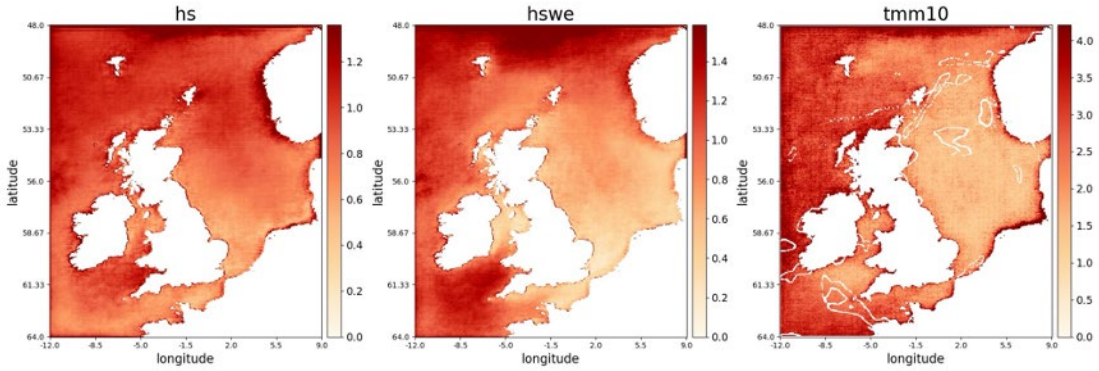


Figure 4.12: The RMSE taken over the whole test dataset (2022 and storms) for hs, hswe and tmm10.Wave heights in meters, wave period in seconds.

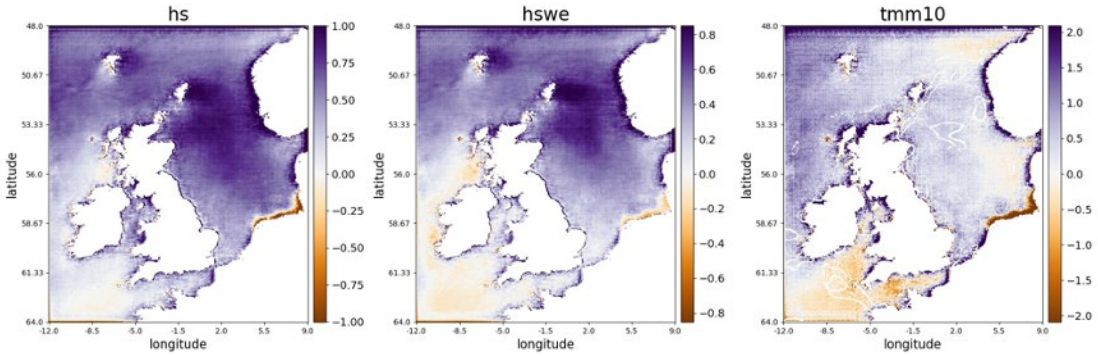


Figure 4.13: The mean error taken over the whole test dataset (2022 and storms) for hs, hswe and tmm10. Wave heights in meters, wave period in seconds.

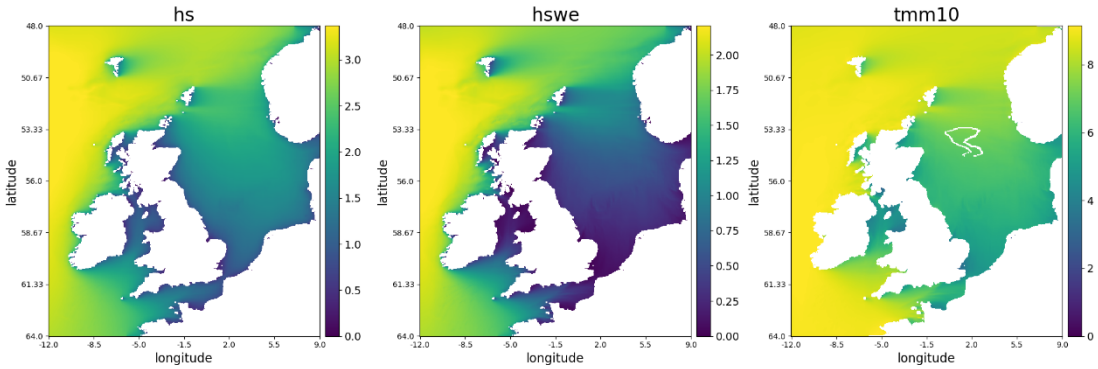


Figure 4.14: The mean value for the whole test dataset (2022 and storms) for hs, hswe and tmm10. Wave heights in meters, wave period in seconds.

Though these figures give a nice overview of the performance of TurboSWAN in the spatial domain, they fail to show its performance in the temporal domain. We assessed this performance by looking at the time series for the whole year at multiple locations across the North Sea. For this report, we chose to focus on the locations of Europlatform, K13 and North Comorant. If we look at the timeseries in figures Figure 4.15 - Figure 4.17, we see that at all three locations, TurboSWAN is able to trace SWAN quite well for both hs and tmm10. Also for hswe, TurboSWAN mimics SWAN well at K13. At North Cormorant, we have a clear overprediction during the spring and summer period, though we can also see that the pattern is still very similar. At Europlatform, we again have a clear over prediction, but here it's apparent throughout the year. Moreover, apart from some peaks, TurboSWAN has a different pattern.

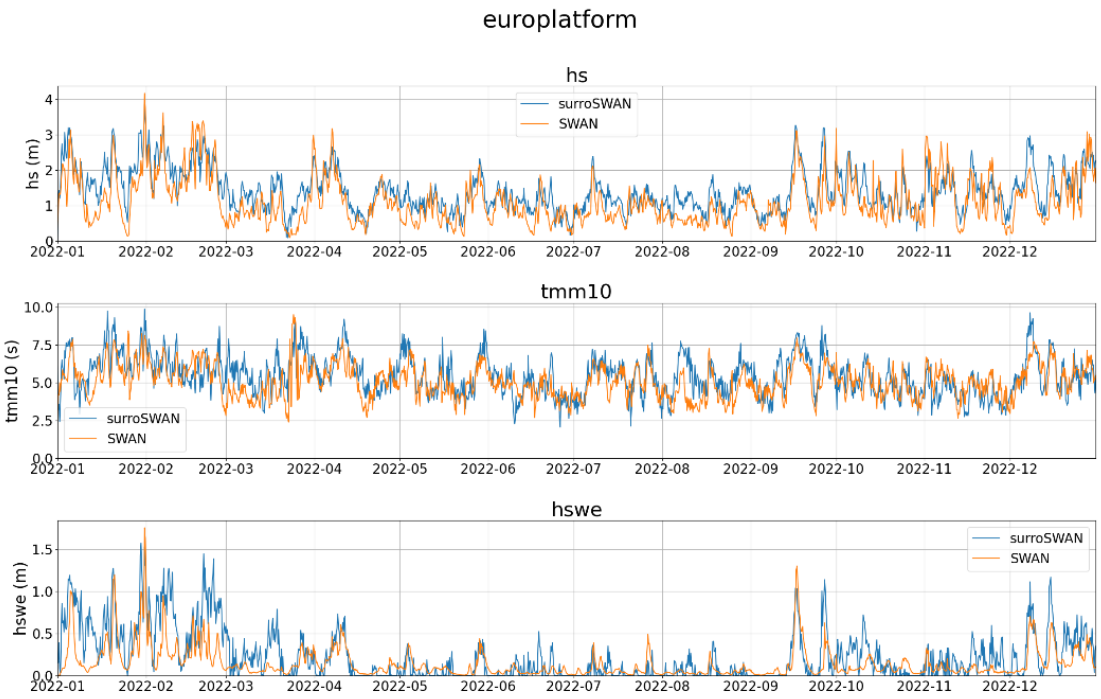


Figure 4.15: The timeseries [year-month] of hs, tmm10 and hswe at Europlatform for TurboSWAN (abusively indicated as surroSWAN) and SWAN.

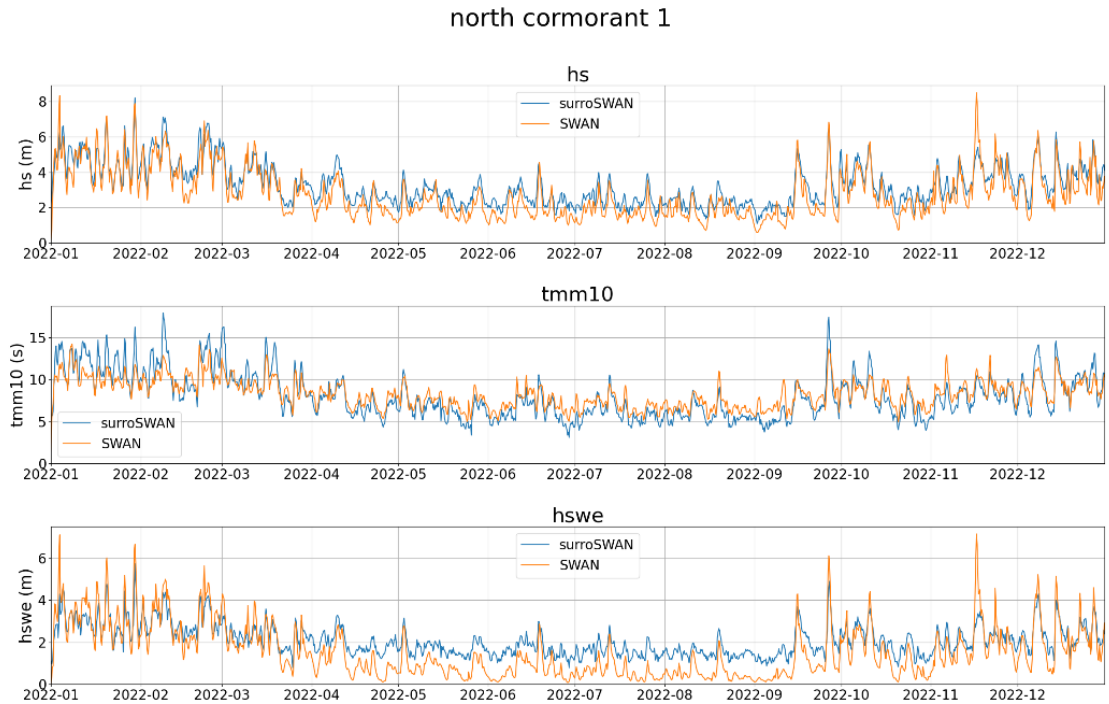


Figure 4.16: The timeseries [year-month] of hs, tmm10 and hswe at North Cormorant for TurboSWAN (abusively indicated as surroSWAN) and SWAN.

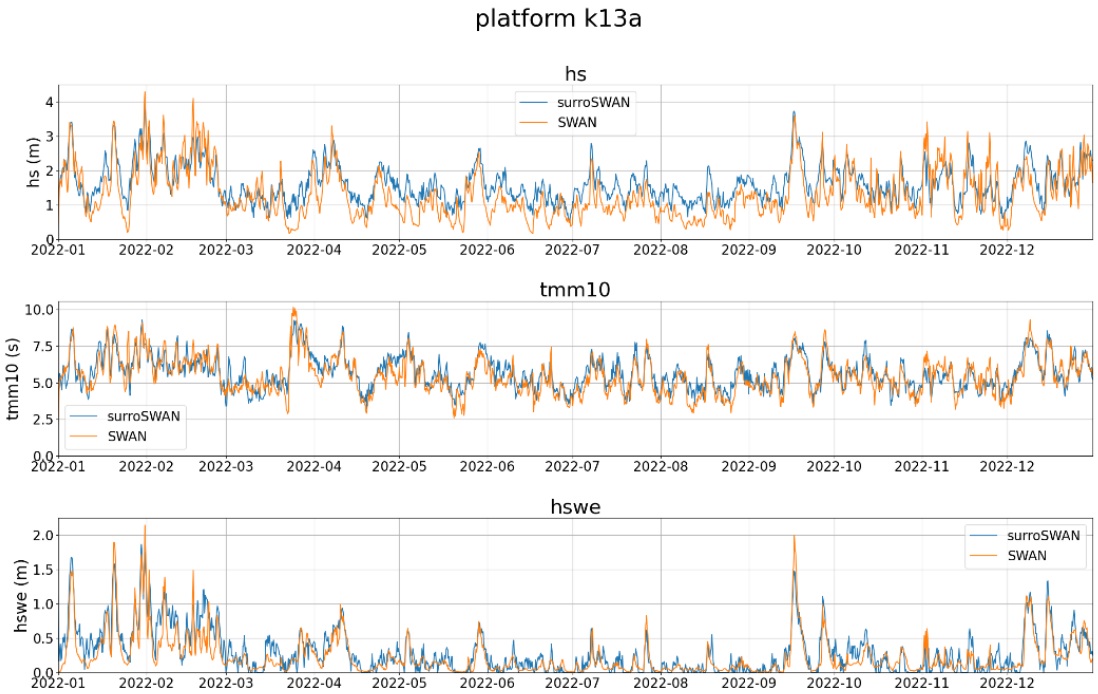


Figure 4.17: The timeseries [year-month] of hs, tmm10 and hswe at K13 for TurboSWAN (abusively indicated as surroSWAN) and SWAN.

To get more detail, we also considered the timeseries at the same locations for the brief period of the storm in January 2018 (timestep for both TurboSWAN and SWAN is 6 hours). We present these timeseries in figures Figure 4.18 - Figure 4.20. They show fairly good results at North Cormorant, though there are some clear under predictions and over predictions. At the other two locations, however, the results are less promising. While tmm10 still has similar values, hs and hswe have clear differences. Especially at Europlatform, we see a very big over prediction by TurboSWAN for the whole month in addition to a spiky pattern in TurboSWAN. Note that there is no link at all between the various parameters (hs, tmm10, hswell) in TurboSWAN whereas there is in SWAN and reality.

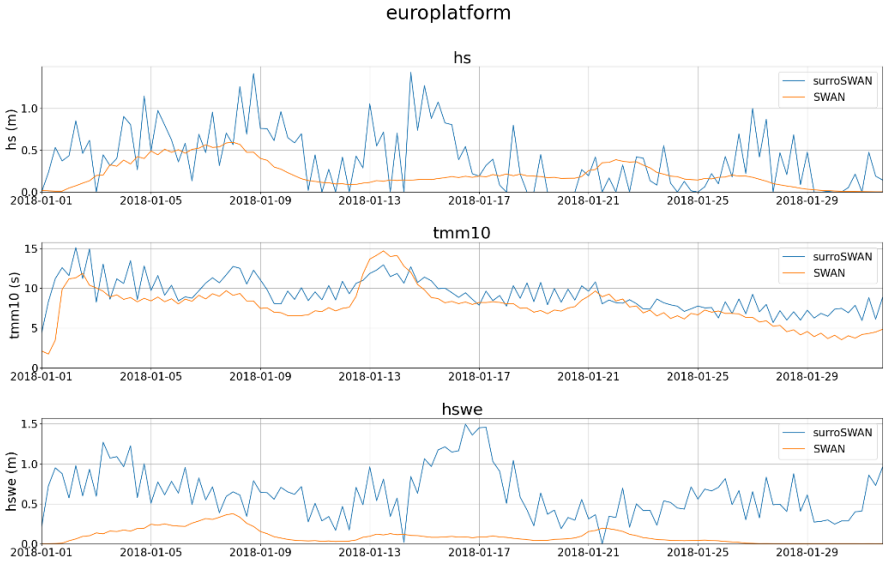


Figure 4.18: The timeseries of hs, tmm10 and hswe at the Europlatform for TurboSWAN and SWAN for the storm of January 2018.

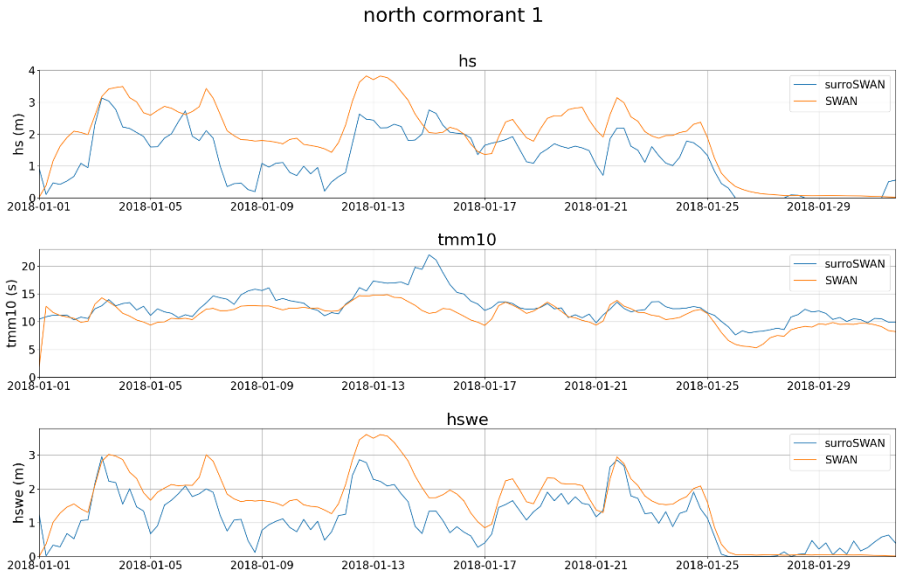


Figure 4.19: The timeseries of the hs, tmm10 and hswe at North Cormorant for TurboSWAN and SWAN for the storm of January 2018.

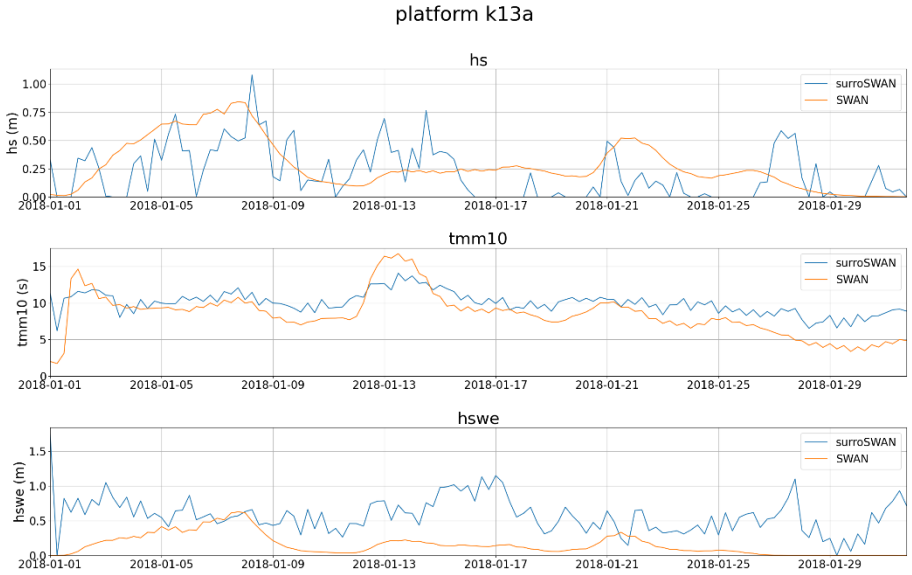


Figure 4.20: The timeseries of hs, tmm10 and hswe at K13 for TurboSWAN and SWAN for the storm of January 2018.

4.5.1 Computational time

As the main advantage of a neural network compared to SWAN is its faster computational time, we also had a look at how long it would take to compute the output fields 48 hours in advance for 50 ensembles. In TurboSWAN we have time steps of 6 hours, which means that we will have to compute 8 time steps for the results after 48 hours. Assuming the input is already given in advance, it takes approximately 25ms to compute one time step on an Apple M1 Max GPU. So, for the 8 time steps and 50 ensembles, it would take about 10 seconds on this GPU or a similar GPU, apart from IO. On the Apple M1 Max CPU, it takes approximately 1s to compute one time step, resulting in a total time of about 6 minutes and 40 seconds, again excluding IO.

4.6 Recommendations for further improvement

Though the current TurboSWAN model is already able to predict the SWAN values quite well, there are still various improvements that we want to explore to improve accuracy.

Firstly, we think that a different choice for the input can improve the results. Instead of combining all input variables into one, keeping them separated until deeper into the network can help the network learn the precise impact of each variable better. The same holds for the output variables.
To try to keep the memory and computational requirements during training lower, we also think it is a good idea to investigate whether the addition of the previous time step in the input leads to an actual increase in performance. In this first experiment it was included with the idea that it can give additional information to the network, to get better results. However, if this improvement is only minor, it might be better to not include them, such that the memory and computational requirements for the training can be distributed somewhere with a higher impact on the performance. Another way to reduce memory and computational requirements is to change the approach a bit. We could use a coarser grid for SWAN as input for the network, while still resulting in a fast method of obtaining ensemble forecasts. This might be an easier problem to learn, leading to more accurate results at the expense of a lower resolution.

Secondly, the number of layers in the network can be optimized as well. Although we did a small analysis on different hyper parameters for the network, we did this analysis on an earlier version of the dataset. In the future, we want to do another, more thorough, analysis on the new dataset to find a better choice of hyper parameters. Lastly, the current version of TurboSWAN used convolutional layers for down- and upsampling. As we stated earlier, this approach has the advantage of being able to learn the down- and upsampling process. However, this can also lead to less smooth results, which we saw in the predictions. Therefore, we would like to do a test where maxpooling is used for downsampling instead of a convolutional layer to test if this is indeed the case for our problem.

It is suggested to reconsider the way of splitting the dataset in training and validation since ensemble members of the same moments may be too much alike to have them in both sets.

5 Conclusions and outlook

5.1 Conclusions

In summary, we report on the progress of a method to provide uncertainty bands for the operational wave forecasts in the North Sea. The method has two components. The first is modelling the uncertainty through an ensemble of wind fields and understanding the validity of this. The second part is keeping computations manageable by training a faster approximate AI wave model.

Via the TIGGE project, ECMWF wind field ensemble (historic) forecasts are available which we used as input for circa 15 months of SWAN simulations with the SWAN-DCSM model. The SWAN input and map output offered a great training set for the TurboSWAN neural network.The uncertainty was assumed to be dominated by errors in the wind forcing. This was true in part. At longer lead times the spread captured most of the observations. However, at shorter lead times (less than 2 days) the errors are often underpredicted. This suggests other sources of uncertainty dominate at shorter lead times (e.g. wave boundary conditions, wave parameterization).

The AI approximate wave model TurboSWAN is very fast and captures the dominant dynamics. The TurboSWAN surrogate model is fast: 25ms to compute one time step. For the 8 time steps and 50 ensembles, it would take about 10 seconds on GPU (and approximately a factor 40 more on CPU). However, during storms and some other cases, errors at the validation stations are still too large for an application. Over the entire domain, the root-mean-square-error of TurboSWAN – compared to the SWAN results – is roughly 0.5 m for wave height and swell wave height. The wave period $T_{m-1,0}$ has an error of approximately 1.5 s. All three variables clearly have higher errors along the coastal areas as well as near the northern boundaries of the North Sea.

We consider both prototypes (modelling the uncertainty through an ensemble of wind fields and the use of a faster approximate AI wave model) a good starting point for further developments. There are many ways in which both components could be improved. For the modelling uncertainty in the first place, other sources of uncertainty could be considered that are dominant on shorter lead times. For the AI model TurboSWAN we could experiment with different inputs, tune hyperparameters and change the network architecture.

The methodology developed here is not limited to the North Sea and can in principle be retrained anywhere in the world, although other issues might arise with different system dynamics.

5.2 Outlook for 2025

The representation of uncertainty in the ensemble wave model can be improved in the future by understanding the underpredicted spread on shorter lead times. This could be done by evaluating for different lead times and including other sources of uncertainty (e.g. WAM wave ensemble at the boundary).

To improve AI wave model TurboSWAN, several directions are worthwhile considering. Testing variations of the input and output, tuning hyperparameters and testing variations of the net architecture are deemed most promising.

Once errors of TurboSWAN are within acceptable range for users (a few order smaller than ensemble spread), a full prototype can be developed and tested for operational use. The aim is to have the full prototype ready at the end of 2025, with more testing for operational use in 2026 if the prototype is deemed acceptable for its purpose. If all goes well, the method could be implemented for operational use in 2027.

Literature

Battjes, J.A. and J.P.F.M. Janssen (1978). Energy loss and set-up due to breaking of random waves, Proc. 16th Int. Conf. Coastal Engineering, ASCE, 569-587

Deltares (2023a). Actualization and validation of SWAN-North Sea and SWAN-Kuststrook models. Deltares report 11209278-005-ZKS-0005, final version, 18-08-2023.

Deltares (2023b). Spectral wave model comparison: A validation of Deltares spectral wave model. Deltares report 11209194-007-ZKS-0001, final version, 02-05-2023

Deltares (2024a). Hindcast SWAN-Markermeer. Ref 11210333-009-ZWS-0003 dd 4 Okt 2024.

Deltares (2024b). Hindcast SWAN-IJsselmeer inclusief IJssel-Vechtdelta. Ref 11210333-009-ZWS-0003 dd 21 Nov 2024.

Hwang, P.A., 2011: A Note on the Ocean Surface Roughness Spectrum. J. of Atmospheric and Oceanic Techn., Vol. 28, 436-443.

Van Ooijen (2023). Ensemble Forecasts for the SWAN model using Multilevel Monte Carlo techniques. TU Delft Master thesis, 14-11-2023.

Albawi, Saad, Tareq Abed Mohammed, and Saad Al-Zawi (2017). “Understanding of a convolutional neural network”. In: *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

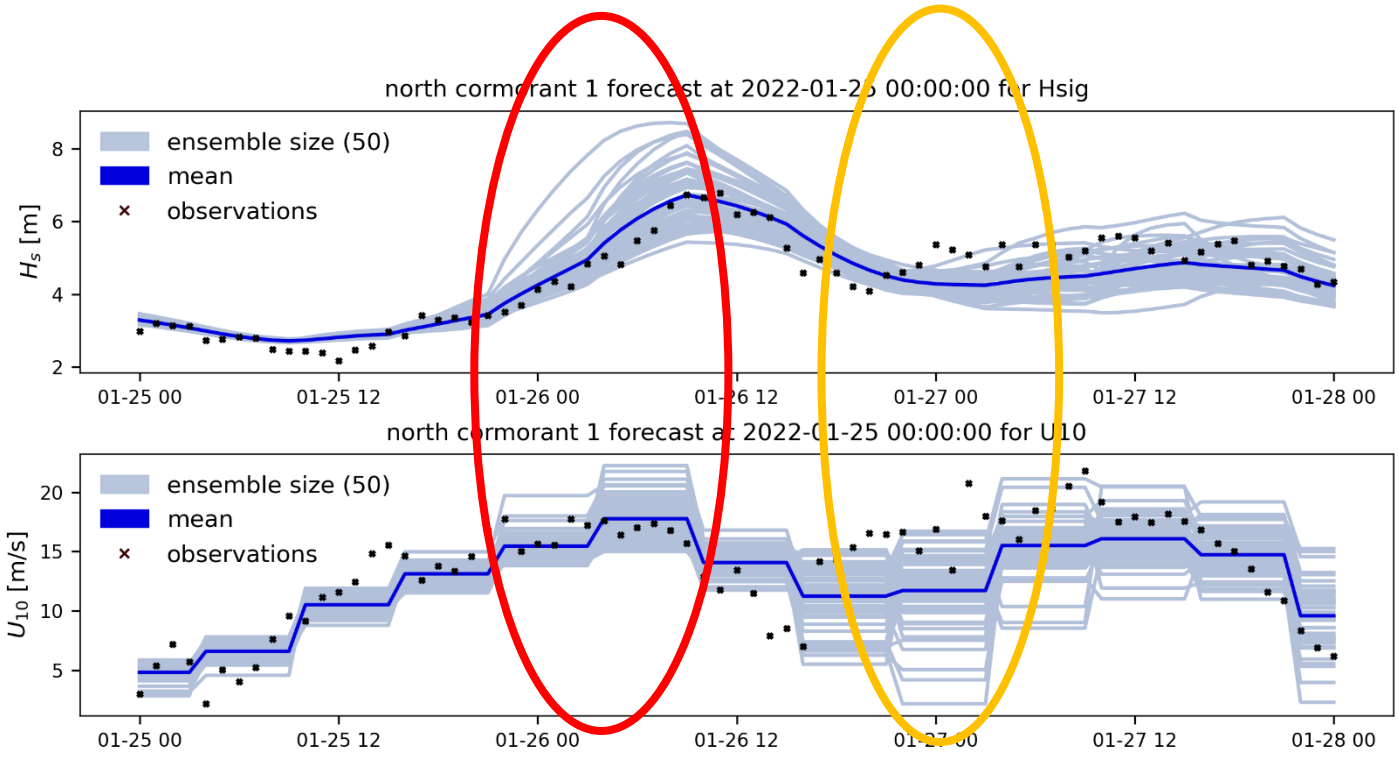
Cornelisse, Daphne (n.d.). *An intuitive guide to Convolutional Neural Networks*. https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neuralnetworks-260c2de0a050/.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). *Deep Residual Learning for Image Recognition*. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

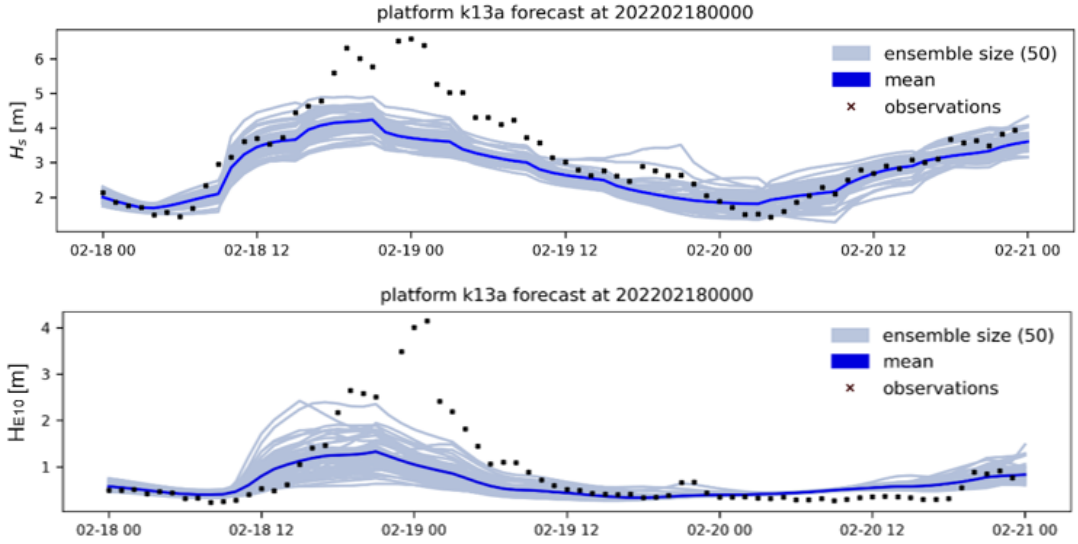
Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. arXiv: 1505.04597 [cs.CV]. URL: https://arxiv.org/abs/1505.04597.

Shi, Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo (2015). “Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting”. In: CoRR abs/1506.04214. arXiv: 1506.04214. URL: http://arxiv.org/abs/1506.04214.

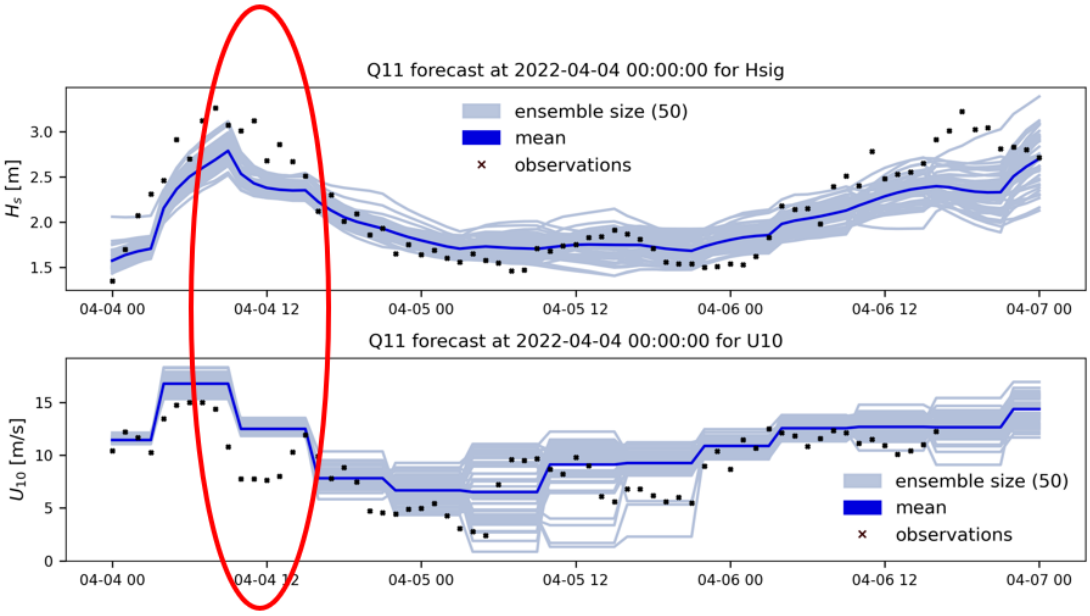
A Ensemble validation



North Cormorant wave height and wind speed ensemble members. Note that in the red shape the spread in wave height is larger than the spread in wind, whereas the yellow shape shows the opposite. The link between uncertainty in wind and wave is not always fixed.



Here, all ensembles drastically underestimate the observed wave height and swell wave height at February 19th, 2022.



This example shows an overestimation in wind speed and at the same time an underestimation in wave height at location Q11 at April 4th, 2022. The ensemble spread in the wind is not the same.

Deltares is an independent institute for applied research in the field of water and subsurface. Throughout the world, we work on smart solutions for people, environment and society.

Deltares

www.deltares.nl