Development of Neural Networks for Ship Speed Prediction

Christos Selimai

Diploma Thesis

School of Naval Architecture and Marine Engineering
National Technical University of Athens

Supervisor: Assistant Professor G. Papalambrou

Committee Members:

Professor G. Grigoropoulos
Associate Professor C. Papadopoulos

July 2021

Acknowledgements

This diploma thesis was carried out at the Laboratory of Marine Engineering of the School of Naval Architecture and Marine Engineering of the National Technical University of Athens, under the supervision of Assistant Professor George Papalambrou. I would, first of all, like to thank him for his constant support and feedback throughout this semester. His insight and knowledge into the subject matter, that steered me to this research in the first place, combined with his guidance and encouragement proved determining for the timely completion of this work. I would also like to thank committee members Professor Gregory Grigoropoulos and Associate Professor Christos Papadopoulos not only for evaluating my work, but for being two of the Professors who stood out and inspired me the past five years.

Next, I would like to thank PhD candidate Orfreas Bourchas for his assistance at every stage of this project and for his insightful suggestions. As this thesis concludes my academic studies, I would also like to extend my sincere gratitude to all the teaching staff of the School of Naval Architecture and Marine Engineering. Their immense knowledge and plentiful experience formed an academic environment of extraordinary quality which I tried my best to make the most of and for which I will forever be thankful. I leave this institution a better man and hopefully a good engineer, because of them.

Furthermore, I wish to acknowledge the John S. Latsis Public Benefit Foundation, the American Bureau of Shipping and the National Technical University of Athens for financially supporting my studies through scholarships and awards. Their support was immeasurable and it has truly been a great honor.

Last but not least, I feel sincerely grateful to my family, friends and colleagues for their unwavering love, support, encouragement and patience in the past five years. None of this would have been possible without them. Thank you.

Περίληψη

Τα τελευταία χρόνια ένα ολοένα και αυστηρότερο πλαίσιο διεθνών κανονισμών, με αφορμή την προσπάθεια μείωσης των εκπομπών αερίων του θερμοκηπίου, έχει ωθήσει τη ναυπηγική και ναυτιλιακή βιομηχανία στην αναζήτηση μεθόδων βελτιστοποίησης της απόδοσης των πλοίων. Αρωγό σε αυτό το έργο αποτελεί η ανάπτυξη της τεχνολογίας, αιχμή του δόρατος της οποίας είναι, σήμερα, οι αλγόριθμοι μηχανικής μάθησης. Αφορμή της παρούσας διπλωματικής εργασίας αποτελεί, λοιπόν, το νέο ερευνητικό πεδίο που δημιουργείται και αφορά στη χρήση των νευρωνικών δικτύων για την αποδοτικότερη επίλυση προβλημάτων της ναυτιλίας. Πιο συγκεκριμένα, μελετάται η δημιουργία ενός εργαλείου ακριβούς πρόβλεψης της ταχύτητας σε ανοιχτή θάλασσα μέσω της αξιοποίησης των χρονικών ιστοριών διαφόρων δεδομένων λειτουργίας που συλλέγονται εν πλω, με απώτερο σκοπό την πιθανή ένταξη του σε έναν πλήρη μηχανισμό βελτίωσης του ενεργειακού προφίλ.

Αρχικά, παρουσιάζεται συνοπτικά η επικρατούσα κατάσταση στο χώρο της ναυτιλίας, όπως διαμορφώνεται εξαιτίας της ανάγκης ελάττωσης του περιβαλλοντικού αποτυπώματος. Έχοντας υπόψη οτι η κάλυψη των απαιτήσεων δεν εναπόκειται, συνήθως, στην αποκλειστική εφαρμογή κάποιας καινοτόμου λύσης, αλλά στηρίζεται στο συνδυασμό τεχνικών και λειτουργικών μέτρων, σημειώνονται ορισμένες από τις πιο δημοφιλείς τακτικές που εφαρμόζονται, όπως τα συστήματα weather routing και βελτιστοποίησης ταξιδιών (voyage optimization), όπου και διαπιστώνεται η ύπαρξη σημαντικού περιθωρίου εξέλιξης λόγω της εξάρτησης τους από την εφαρμοζόμενη μέθοδο πρόβλεψης της ταχύτητας. Σε αυτή τη βάση, αναφέρονται οι διάφορες τεχνικές που έχουν επιστρατευτεί τα τελευταία χρόνια από τους ερευνητές για τη λύση του προβλήματος της πρόβλεψης της ταχύτητας, και της συμπεριφοράς γενικότερα, ενός πλοίου σε ανοιχτή θάλασσα. Η διαθέσιμη βιβλιογραφία περιλαμβάνει ένα ευρύ φάσμα προσεγγίσεων που εκτείνονται από θεωρητικά μοντέλα, τα οποία επικεντρώνονται στην πλήρη περιγραφή και επίλυση της εξίσωσης ισορροπίας, σε εφαρμογές στατιστικής ανάλυσης και, σε πρωτόλειο επίπεδο, μηχανική μάθηση.

Στη συνέχεια, γίνεται μια συνοπτική παρουσίαση του απαραίτητου θεωρητικού υποβάθρου για εξοικείωση με τον επιλεγμένο τρόπο αντιμετώπισης του προβλήματος στην παρούσα εργασία. Πιο συγκεκριμένα, μετά την περιγραφή της γενικής φιλοσοφίας και των βασικών εννοιών των νευρωνικών δικτύων, αναλύεται η συνήθης ροή εργασιών στα προβλήματα μηχανικής μάθησης, από την εύρεση των δεδομένων ως τη βελτιστοποίηση των ρυθμιστικών παραμέτρων των νευρωνικών δικτύων.

Το πρακτικό μέρος ξεκινά με τη διεξοδική ανάλυση της διαδικασίας προ-επεξεργασίας των καταγραφών από τον πραγματικό πλου ενός containership. Έπειτα από τις πρωταρχικές ενέργειες εξοικείωσης με τα χαρακτηριστικά των μετρήσεων και με οδηγό τα εργαλεία της στατιστικής ανάλυσης, σημειώνονται τα στάδια επεξεργασίας που χρησιμοποιήθηκαν για την εξασφάλιση ενός ποιοτικού συνόλου επιχειρησιακών δεδομένων, απαραίτητου για την επιτυχία της μεθόδου. Με τη βοήθεια του Keras API και του TensorFlow εξετάζονται, τότε, διάφορες κατηγορίες νευρωνικών δικτύων, και πιο συγκεκριμένα, το απλό γραμμικό μοντέλο, το Multi

Layer Perceptron, το Convolutional και τα Recurrent Neural Networks, με έμφαση στη Long – Short Term Memory αρχιτεκτονική. Μέσω των αποτελεσμάτων για διάφοροους ορίζοντες εισόδου και προβλέψεων, παρουσιάζεται, τότε, ικανοποιητική ακρίβεια στην πρόβλεψη της ταχύτητας, επιβεβαιώνοντας, επομένως, την ύπαρξη γονίμου εδάφους τόσο για εφαρμογή των εξεταζόμενων εργαλείων στη βιομηχανία, όσο και για περαιτέρω έρευνα.

Abstract

In recent years the ever tightening set of international rules regarding the reduction of greenhouse gases emissions has forced the shipbuilding and shipping industry to pursuit ways to optimize their ships' efficiency. A decisive contribution to this goal is made, nowadays, by certain new technologies available, the most innovative of which are machine learning algorithms. This new research field, created by the adoption of neural network models for dealing with maritime related problems, is the main motivation behind this thesis. More specifically, the possibility of establishing an accurate ship speed prediction tool that uses the time history of various features measured during a voyage and which could, in turn, be integrated in a complete optimization application, is examined.

At first, a short introduction is made on the current state of the shipping industry, in light of the need to reduce its carbon footprint. Keeping in mind that conforming with the current regulations is usually not a matter of a single breakthrough solution, but depends on the combination of various technical and operational measures, few of the most widespread complementary techniques, like weather routing and voyage optimization, are presented, while the room for significant improvements, due to their dependence on the accuracy of ship speed prediction, is also highlighted. Next, by referencing the available published work, a broad variety of different methods employed by researchers for forecasting ship speed in a seaway are described. Throughout the years, pure theoretical models, aiming to fully describe the physical mechanisms and provide a solution to the equilibrium equation, statistical methods and, as of lately, machine learning algorithms have been tested with variable, but notable nonetheless, levels of success.

Next, a brief demonstration of the theoretical background, required to familiarize the reader with the chosen way of dealing with the problem, is given. More specifically, once the main idea and basic concepts behind neural networks are described and the typical machine learning workflow, from data acquisition to the fine-tuning of the network's parameters, is presented.

The practical part begins with an extensive examination of the pre–processing procedures applied to the authentic set of measurements recorded during a containership's voyages. Following the primary exploratory data analysis, which is utilized in order to get a basic sense of the characteristics and issues of the dataset, the steps taken so as to ensure a high quality training set (extremely important in neural networks) are described in detail. With the help of Keras and Tensorflow, multiple architectures of neural networks are presented and compared. These include a simple linear model, a Multi Layer Perceptron, a Convolutional Neural Network and a Recurrent Neural Network called Long-Short Term Memory network, which were tested on a range of different input and prediction horizons. The results showed satisfying accuracy, thus confirming the existence of fertile ground for further research in the subject and the prospect of immediate implementation of this technology in the industry.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

From joining wooden planks to cross the Nile, to welding steel cruise ships that can accommodate more than 6500 people, the shipping industry has come a long way since its birth, several thousand years ago. Today, the international shipping industry is responsible for transporting approximately 90% of world trade, utilizing an estimate of 56000 cargo ships ranging from general cargo ships and bulk carriers to crude oil tankers, chemical tankers, LNG tankers and containerships. More than 1.5 million sailors, engineers and managers around the world make sure that cargo worth millions of dollars reaches its destination on time, safely and efficiently.

On this basis, it should come as no surprise, that moving the world fleet around the globe requires a great amount of power, which is predominately produced, for the time being, by burning fuel oil and consequently releases significant amounts of emissions, the most notable being CO2 and other Greenhouse Gases (GHG), SOx and NOx. Although in terms of CO2 emissions per tonne of cargo transported one mile, shipping is by far the most efficient form of commercial transport, maritime industry's total volume of atmospheric emissions corresponds to around 3% of global greenhouse gas emissions or 940 million tonnes of CO2 annually, based on a study by the International Maritime Organization (IMO) [8]. As a result of the increased awareness on environmental issues, a global approach to reduce shipping emissions led by the IMO has resulted to the adoption of a series of stringent measures, with the ambition of reducing CO2 emissions per transport work, as an average across international shipping, by at least 40% by 2030, pursuing efforts towards 70% by 2050, compared to 2008; and cutting the total annual GHG emissions from international shipping by at least 50% by 2050 compared to 2008, despite the expected rise in the global trading of goods.

The main strategy to achieve these goals was introduced with the inclusion of regulations on energy efficiency for ships in MARPOL Annex VI, where the Energy Efficiency Design Index (EEDI) for new ships and the Ship Energy Efficiency Management Plan (SEEMP) for all ships were made mandatory. The EEDI is a performance – based technical measure which aims at promoting optimum hull designs and more energy efficient (less polluting) equipment and engines. By requiring every new ship design to meet an incrementally tightening reference level of emitted grams of CO2 per transport work, based on its design parameters, continuous innovation and technical development of every component affecting fuel efficiency are encouraged.

Of course, judging from the ship owners' point of view, every measure taken should remain cost effective. As less evasive solutions, like fine-tuning the Main Engine's combustion parameters and retrofitted pollution control devices, reach their limits, the enhancement of the hull shape design and power plant arrangement optimization can be complemented with many technical and operational solutions, such as voyage planning and weather routing. These not only can assist in fulfilling the emission requirements but often deliver more fuel savings than the investment required, hence reducing significantly both the fuel oil cost (every ship's main daily expense) and GHG emissions.

Utilizing weather routing, the practice of selecting, based on the weather forecasting, the optimal route (as shown in Fig.1.1) in terms of fuel consumption and emissions, while achieving the required time of arrival, without exposing the crew, vessel and cargo to risk, has undoubtedly served its purpose in the past. However, as shipping companies attempt to minimize fuel consumption by slow steaming – the practice of operating cargo ships at significantly less than their maximum speed, usually employed in times of high fuel oil costs – and Virtual Arrival policy, the technique of reducing sailing speed in order to cut unproductive waiting time at the destination ports when there are known delays [9], the potential for improvement is significant. Existing ship performance estimation technologies rely heavily on experiments with model ships in water tanks or on physics model simulations, that do not take into account the complicated interactions of winds, waves, and sea currents, resulting in large margins of error.

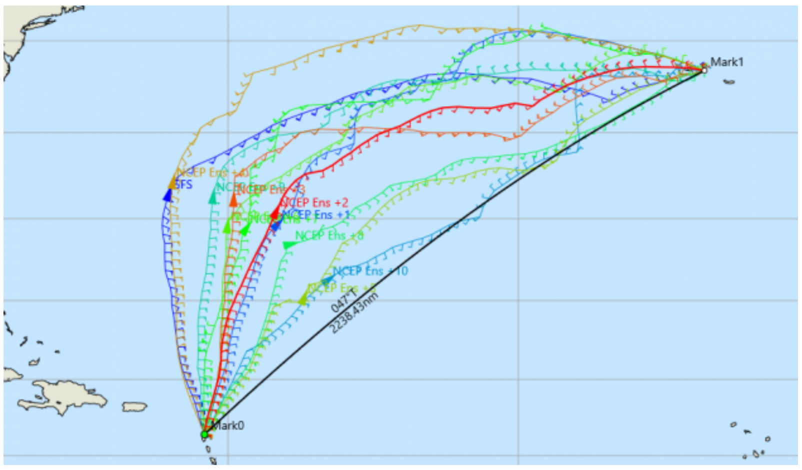


Figure 1.1: Alternative routes examined by a weather routing system

Since even the slightest improvement in propulsive performance can make a significant difference, the accurate monitoring of various equipment and procedures, both in a calm sea condition and in a seaway, becomes critical. Vessels around the world are gradually getting equipped with the latest technologies in sensors and data acquisition systems, aiming to collect and monitor high quality data related to the operational status of each subsystem and the ship as a whole. Instead of plainly visualizing the data to the monitors on board, a range of analyzing systems can be utilized to confirm that the design performance is achieved.

The resulting plethora of available data, gathered automatically sometimes in intervals as short as 1 minute, or less, for the full duration of a ship's voyage, can easily overwhelm human engineers and traditional data-processing application software and hinder their ability to take full advantage of the information collected. Nowadays, the field of analyzing and systematically extracting information from large and complex datasets like a vessel's operational data (qualified as the so-called Big Data) is dominated by machine learning algorithms, due to the extraordinary progress that deep learning has represented in terms of accuracy and performance. The way is thus paved for a systematic study of possible implementation of these systems in the maritime industry.

1.1 The problem, Motivation and Objectives

As more and more sensors are installed on board, the collection of large datasets makes it possible to establish a ship's propulsion performance model based on machine learning techniques. An accurate forecast of attainable ship speed is, then, a requirement for confirming that design performance is achieved not only in calm seas, but also in a seaway. Moreover, ship speed prediction is fundamental to successful cruise optimization and if, ideally, combined with accurate atmospheric and sea condition predictions and a thorough modelling of the ship's behavior in different environments, could result in a comprehensive and accurate weather routing system. All useful information could then be provided to the shipmaster, aiming to assist his decision-making process for selecting the optimal route variant [10].

A ship speed prediction system could also be integrated into a hull monitoring system. This way, the neural network can provide critical insight into the ship operation and should unexpectedly large deviations between the expected and measured speeds be recorded, indicate that action regarding its maintenance be taken in time.

While AI systems spread like wildfire in practically every industry, a more systematic approach to the process required to develop neural network for tackling a maritime related problem, like the prediction of a vessel's speed, is of significant interest. This is the main motivation for the current study.

In this thesis, a detailed examination of the process of developing several different kinds of artificial neural networks capable of predicting a range of future speed values is presented. From data preprocessing to model architecture selection and optimization, the detailed workflow is demonstrated. By the end of this study the more suitable architectures shall be determined by comparing each one's prediction accuracy and behaviour on the different combinations of input - expected output time steps.

1.2 Literature Review

The subject of predicting a vessel's speed is usually studied under the scope of evaluating ship performance as a whole. Although ship performance is a very diverse term, this generally means that treating ship speed comes packed with analyzing fuel oil consumption and engine power. Focusing on the prediction of ship speed, a variety of approaches have been taken; some address the physical mechanisms, others deal with statistically based algorithms and lately machine learning algorithms are making a dynamic appearance.

In [1] a thorough physical model based on a mature theoretical frame is employed as a means of predicting ship propulsive performance both in a calm sea condition and a seaway. More specifically, methods for practical prediction of steady forces in waves (added resistance in waves), wind forces (wind resistance), drift forces and steering forces are presented and utilized for the prediction of the engine's operating point. It is noted that the cases treated assume relatively mild weather conditions (Beaufort scale ≤ 7) so as to avoid implicating the cases of deliberate change in speed and heading for maintaining seakeeping performance in heavy weather conditions. Important weather factors taken into consideration include significant wave height H_S, wave period T, wind speed W and the angle it forms relative to the ship direction.

The flow diagram of the full performance prediction method used in the referenced study is shown in Fig.1.2.

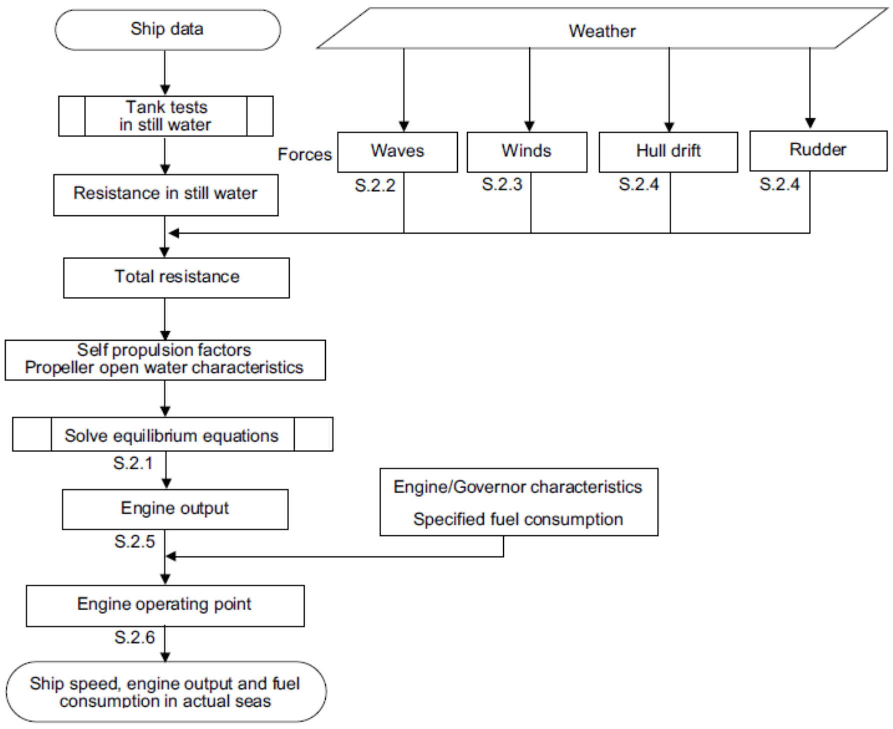

Ship data
Weather
Tank tests in still water
Forces
Waves
Winds
Hull drift
Rudder
S.2.2
S.2.3
S.2.4
S.2.4
Resistance in still water
Total resistance
Self propulsion factors
Propeller open water characteristics
Solve equilibrium equations
S.2.1
Engine output
Engine/Governor characteristics
Specified fuel consumption
S.2.5
Engine operating point
S.2.6
Ship speed, engine output and fuel consumption in actual seas

Figure 1.2: Flowchart for prediction of ship performance in actual seas [1].

Added resistance corresponds to the loss of energy related to the radiating waves, caused by the ship's motion, as well as the reflection and diffraction of incident waves, and can be considered the main reason behind a vessel's involuntary nominal velocity deviation from its service speed in ocean operation. On this basis, a variety of techniques with different levels of complexity, such as potential flow based methods, experimental methods and Computational Fluid Dynamics approaches have been utilized in order to forecast the added resistance, but with a resulting great dispersion of outcomes, based on the approach taken.

For the prediction of wind resistance, wind tunnel tests combined with methods based on the datasets of similar hulls are used, while the hull drifting and steering forces are taken into account through the use of regression or empirical equations. Once the evaluation of these external forces is complete, several tank test based techniques for the prediction of power are implemented and compared, before calculating the engine operating point by defining the mean effective pressure, overload protection and fuel index limits and simulating the control by constant frequency of engine revolution by the governor.

Without getting into much detail about the equations and methods used, it may have already become clear that using physical models is more fit for scientific research rather than getting practical results. Traditional methods focus on solving the equilibrium system between ship resistance and propeller thrust, hence require significant amount of time and computational resources, while lacking accuracy due to the generic nature of the theoretical calculations, the assumptions in modelling physical phenomena and the subsequent limitations in their scope of application. These techniques rely on experiments with model ships in water tanks, or on physics model simulations and do not necessarily meet all hull, weather and sea conditions. Even if the prediction error is calculated, there is no way of adapting the model without reverting back to scientific research.

In [2] two statistical procedures are investigated, aiming to establish a model for ship speed prediction, by utilizing one years' worth collection of measurement data from a 2800TEU containership. The paper investigates the use of statistical methods and advanced analysis to form a simple relation between speed V, measured propeller RPM and selected weather factors, assuming a constant hull fouling status.

The observations of ship positions and engine RPM under study were recorded every five minutes and the corresponding speed V was computed from the successive sailing positions, which means that it corresponds to speed over ground – not through water. The examined time series of RPM and V are shown in Fig.1.3. It is important to note that speed through water is usually used to evaluate ship performance, as ocean currents can cause changes in ship speed over ground. The same important environmental parameters, i.e. significant wave height H_s, mean wave period T and wind speed W projected with its angle relative to the vessel's direction, together with the wave velocity are extrapolated using data from the European Centre for Medium-Range Weather Forecasts servers. The routes selected to fit the models are equally spread over the year so that no seasonal bias is introduced.

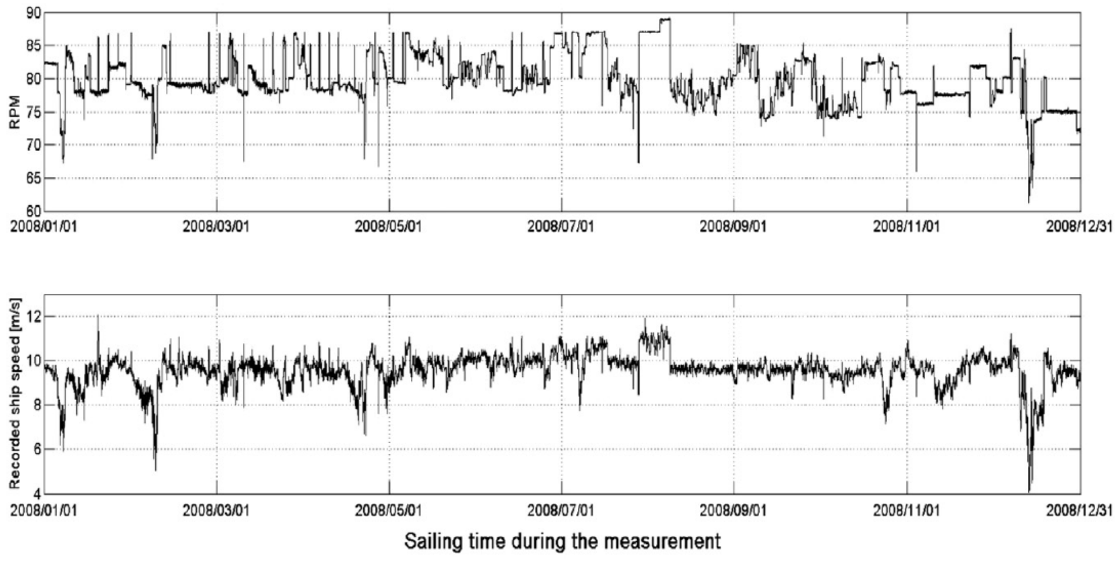


Figure 1.3: Time history of RPM and V under study [2].

First, a linear formulation for the ship speed prediction, based on research by MAN, is presented:

$$V^{pred} = \alpha + \beta(RPM - m) + c_1H_s + c_2T + c_3W + c_4C \tag{1.1}$$

The above parameters are the ones to be calculated using statistical modelling. It is noted that due to the different weather conditions ships meet between travelling eastbound and westbound through the Atlantic Ocean, two different instances of each model need to be fitted. The first model is established using simple linear regression analysis and by minimizing prediction errors, while in the second model the autocorrelation function of ship speed is taken into consideration in order to form a first order autoregressive model. In the third model, the parameters α,β are regarded as random (taking different values for each voyage) and estimated using a mix of the previous two models.

As expected, the prediction errors reduce significantly the more complex the applied technique gets, while the results based on the mixed effect model are as good as the ones extracted from the autoregressive model. The results are also judged based upon the accumulated error of sailing distance $\epsilon(T_s)$, as shown in Fig.1.4.

The fact that all tested methodologies used limited information of a small containership is highlighted, pointing towards the need of broader research to check the forecasting capability of these statistical techniques in different ships and environmental parameters. The need for eastbound-westbound separation should also be considered when assessing the practical application of the tested methods.

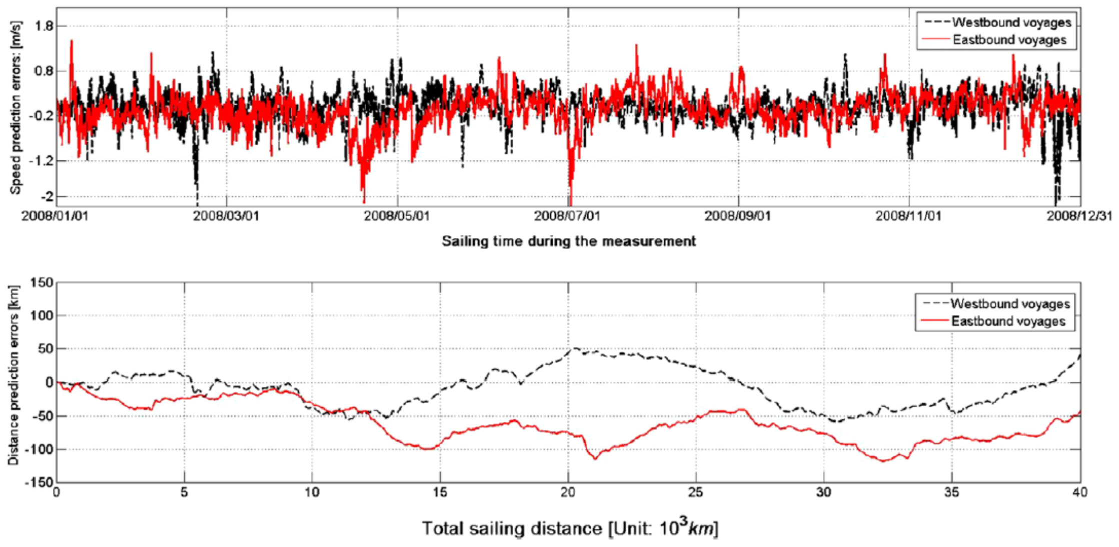


Figure 1.4: Errors of ship speed prediction $e(t)$ and errors of predicted sailing distance $\epsilon(T_s)$ [2]

Next, studies employing machine learning are examined. In general, the use of machine learning algorithms in maritime – related problems is not a new phenomenon. Classic machine learning techniques used include Support Vector Machines, a widely spread tool for classification and regression in which input data is mapped into high-dimensional space where it constructs an optimal separating hyperplane, and Fuzzy Logic regression models, which instead of specifying the relation between input and output with a single equation, uses local functions to globally approximate the nonlinear model. More specifically, SVM have been used in computer vision applications like ship detection and in ship motion prediction, while Fuzzy Logic has been used for maritime weather prediction [11].

Nowadays, Neural Networks have also made a dynamic entrance in the shipping and maritime field. Some examples include estimating ship propulsion efficiency, modelling, analyzing and predicting ship motions. The advantages are apparent; training a neural network is engineering oriented, does not require an extensive theoretical background, although the choice of input variables backed by theoretical and experimental data could prove helpful, and the procedure can be carried out quickly and efficiently, even on board. The complicated interactions of winds, waves and sea currents, that influence the state of ships at real sea waters, do not need to be explicitly modeled, but can be left for the neural network to discover through the training data.

In [11] artificial neural networks of various numbers of hidden layers and neurons and different types of activation functions are used to establish an accurate regression model for the fuel consumption of the main engine.

In [3] an AI powered high dimensional statistical analysis tool is developed in order to estimate ship performance in real sea waters. By using ship operation data, engine data logs, weather and sea sensing data and other ship related information, it is concluded that the most fuel-efficient route can be accurately determined, while the analysis could help create some guidelines for the design of fuel-efficient ships or assist the comparison of ship performance before and after ship maintenance.

More analytically, the researchers applied Fujitsu's own AI technology to big data analysis of ship operation and by breaking away from conventional physics models and allowing automatic grouping of the high-dimensional data by similarity, as shown in Fig.1.5, improved estimation accuracy to extremely high levels (a margin of error of 5% or less). The data were collected using a test ship and consisted of the following features, measured in one-second intervals:

- Bow Direction
- Speed Through Water (STW)
- True Wind Direction
- True Wind Speed
- Rudder Angle
- Controllable Pitch Propeller blade angle
- Shaft Revolutions
- Shaft Power
- Main Engine Revolutions
- Fuel Consumption

Data recorded at low speeds with STW of 5 knots or less were excluded from the training and evaluation processes.

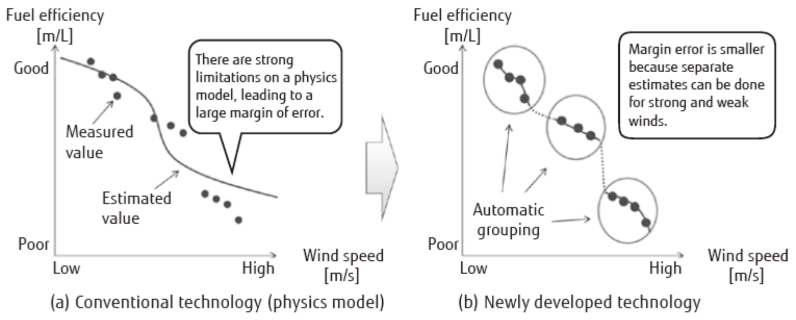

Fuel efficiency [m/L]
Good
Poor
Low
High
Wind speed [m/s]
There are strong limitations on a physics model, leading to a large margin of error.
Measured value
Estimated value
(a) Conventional technology (physics model)
Fuel efficiency [m/L]
Good
Poor
Low
High
Wind speed [m/s]
Margin error is smaller because separate estimates can be done for strong and weak winds.
Automatic grouping
(b) Newly developed technology

Figure 1.5: Differences between conventional and newly adapted technology [3].

The estimation error was evaluated using 10-fold cross validation, a validation method that repeats verification a total of ten times, by dividing the entire data into ten blocks, selecting and learning nine out of these ten blocks and using the remaining block for evaluating the estimation error, in order to evaluate estimation error in all blocks. As the error index, mean absolute percentage error (MAPE) obtained by the following question was used.

$$MAPE = \frac{1}{T}\sum_{t=1}^{T}\frac{|f(x_t) - y_t|}{|y_t|} \tag{1.2}$$

The mean absolute percentage error with respect to the estimation of the STW was 2.8%, and the mean absolute percentage error with respect to the estimation of the fuel consumption was 4.4%, showing that the proposed technology is capable of high accuracy estimation in both cases, with an error of 5% or less. The effectiveness of the proposed method is illustrated in Fig.1.6. Based on the above results, use of this technology allows prediction of ship speed and fuel consumption with high precision on routes to be traveled.

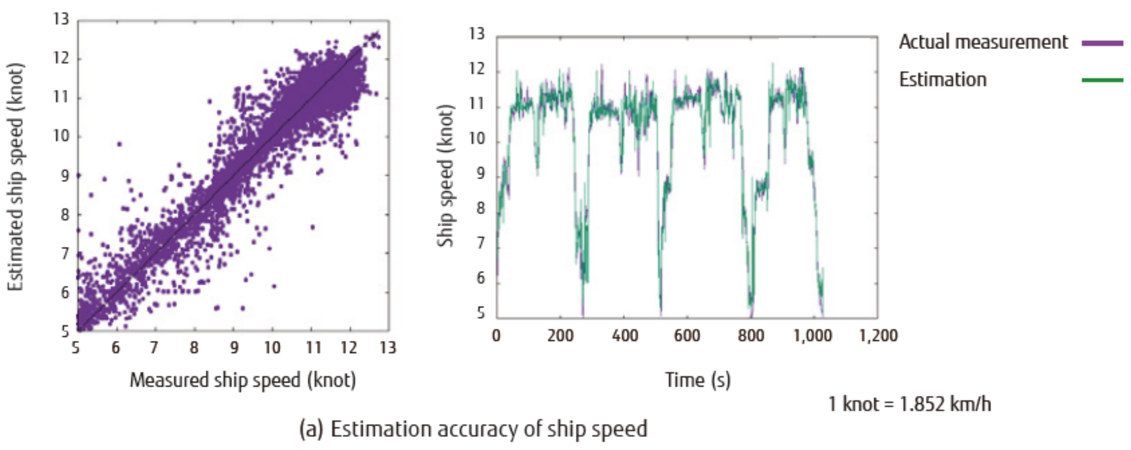
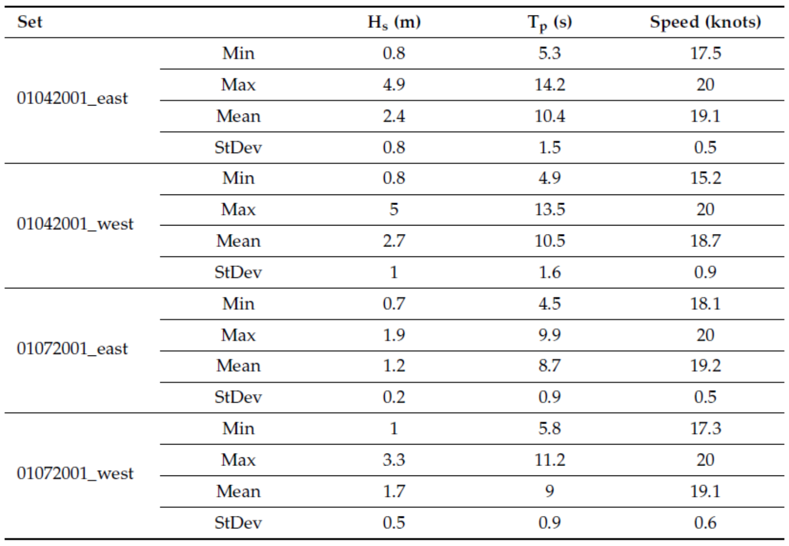


Figure 1.6: Verification of estimation accuracy of developed technology [3].

In [4] a neural network was also trained to predict ship speed and fuel oil consumption. The dataset used was the result of a number of simulations on a given oceanic route performed by a weather routing system, in order to realistically resemble the collection of data expected to be obtained on board a ship in operation. The model was trained using the output torque of the main engine Q, the revolutions per minute of the propulsion shaft RPM, the significant wave height H_s, the period of the waves T and the relative angle of wave encounter a as input features. A sensitivity analysis was also carried out to study the model's performance when inputs regarding the status of the engine were not used.

More specifically, the simulated dataset was first divided into four sets, corresponding to the assumed four voyages in the North Atlantic; westward and eastward on two occasions, two months apart. The sampling rate was not kept constant, as the consecutive observations were measured at control points set every 30 nautical miles, along the route under study. The information was, then, randomly split to the training, validation and testing datasets in fractions of 0.7, 0.15 and 0.15 respectively. The maximum, minimum and average values of H_s, T_p in each voyage are given in Fig.1.7.



Set		H_s (m)	T_p (s)	Speed (knots)
01042001_east	Min	0.8	5.3	17.5
	Max	4.9	14.2	20
	Mean	2.4	10.4	19.1
	StDev	0.8	1.5	0.5
01042001_west	Min	0.8	4.9	15.2
	Max	5	13.5	20
	Mean	2.7	10.5	18.7
	StDev	1	1.6	0.9
01072001_east	Min	0.7	4.5	18.1
	Max	1.9	9.9	20
	Mean	1.2	8.7	19.2
	StDev	0.2	0.9	0.5
01072001_west	Min	1	5.8	17.3
	Max	3.3	11.2	20
	Mean	1.7	9	19.1
	StDev	0.5	0.9	0.6

Figure 1.7: Maximum, minimum and average values of H_s, T_p in each voyage [4].

Two different Multi-Layer Perceptron (feedforward artificial neural networks) architectures, developed in MATLAB, were tested. Apart from the input and output layers, the first MLP contained a single hidden layer, while the second consisted of three hidden layers. In every node of the hidden layers the log sigmoid function was used as a non-linear activation function, due to its ability to produce smooth outputs, and the models were trained using the backpropagation method, focusing on MATLAB's default Levenberg–Marquardt optimization, which was then compared to the resilient backpropagation algorithm. The number of hidden neurons were selected after several testing runs were conducted on the basis of a sensitivity analysis of the results. As no specific information is given, it is assumed that the MLP model took in the input features one time step at a time and predicted the ship speed of the immediately following instance. Several other model characteristics, are not disclosed. Two examples of the output ship speed estimation are given in Fig.1.8.

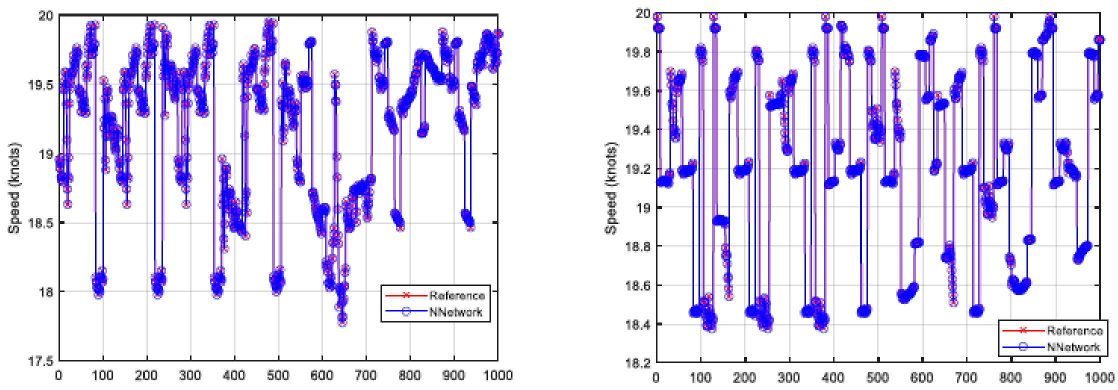
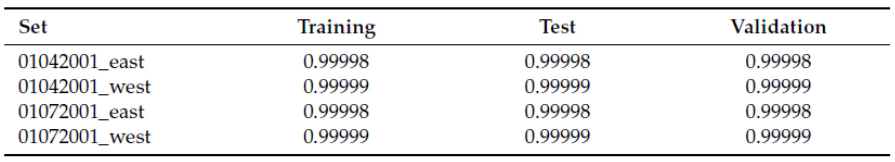


Figure 1.8: Ship speed estimation for 01042001east and 01072001east voyages [4].

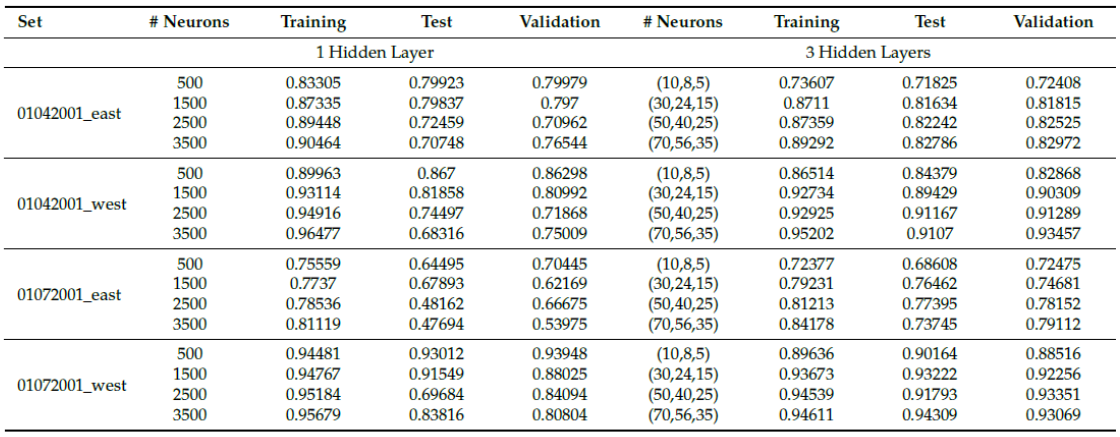
A correlation analysis is then carried out using the Pearson correlation coefficient r, which is utilized to measure the linear dependence between the model and the expected outputs. The r results presented in the study and given in Fig.1.9, along with the regression plots of each testing set, show good correspondence between the model predictions and the true results.



Set	Training	Test	Validation
01042001_east	0.99998	0.99998	0.99998
01042001_west	0.99999	0.99999	0.99999
01072001_east	0.99998	0.99998	0.99998
01072001_west	0.99999	0.99999	0.99999

Figure 1.9: r results of the neural network training for the speed estimation [4].

Furthermore, the model's ability to accurately predict the ship speed when only information on the sea conditions (H_s, T, a) are provided, was analyzed with the help of the sensitivity analysis. The configurations tested involved four different cases for the number of neurons; 500, 1500, 2500 and 3500 for the single hidden layer model and (10,8,5), (30,24,15), (50,40,25), (70,56,35) for the three hidden layers network. The loss of accuracy due to the lack of engine state data proved to be more obvious in the voyages where H_s and T were smaller, but the overall precision, as shown in Fig.1.10, was deemed satisfactory. Judging from the r results, the superior efficiency of the three hidden layers structure in the validation and testing datasets was also noted, compared to the inability of the single hidden layer model to maintain the accuracy achieved in the training dataset.



Set	# Neurons	Training	Test	Validation	# Neurons	Training	Test	Validation
	1 Hidden Layer				3 Hidden Layers			
01042001_east	500	0.83305	0.79923	0.79979	(10,8,5)	0.73607	0.71825	0.72408
	1500	0.87335	0.79837	0.797	(30,24,15)	0.8711	0.81634	0.81815
	2500	0.89448	0.72459	0.70962	(50,40,25)	0.87359	0.82242	0.82525
	3500	0.90464	0.70748	0.76544	(70,56,35)	0.89292	0.82786	0.82972
01042001_west	500	0.89963	0.867	0.86298	(10,8,5)	0.86514	0.84379	0.82868
	1500	0.93114	0.81858	0.80992	(30,24,15)	0.92734	0.89429	0.90309
	2500	0.94916	0.74497	0.71868	(50,40,25)	0.92925	0.91167	0.91289
	3500	0.96477	0.68316	0.75009	(70,56,35)	0.95202	0.9107	0.93457
01072001_east	500	0.75559	0.64495	0.70445	(10,8,5)	0.72377	0.68608	0.72475
	1500	0.7737	0.67893	0.62169	(30,24,15)	0.79231	0.76462	0.74681
	2500	0.78536	0.48162	0.66675	(50,40,25)	0.81213	0.77395	0.78152
	3500	0.81119	0.47694	0.53975	(70,56,35)	0.84178	0.73745	0.79112
01072001_west	500	0.94481	0.93012	0.93948	(10,8,5)	0.89636	0.90164	0.88516
	1500	0.94767	0.91549	0.88025	(30,24,15)	0.93673	0.93222	0.92256
	2500	0.95184	0.69684	0.84094	(50,40,25)	0.94539	0.91793	0.93351
	3500	0.95679	0.83816	0.80804	(70,56,35)	0.94611	0.94309	0.93069

Figure 1.10: Maximum, minimum and average values of H_s, T_p in each voyage [4].

A sensitivity analysis was also performed to compare the prediction capabilities of models optimized using the resilient backpropagation learning algorithm with the neural networks trained using the Lavenberg-Marguardt algorithm, where no noteworthy difference was observed.

Overall, these last two studies provide a solid basis for our research, despite discernible issues. First of all, as in many related papers, it is common practice to tackle the problem of speed or consumption prediction using model architectures that treat the dataset as a collection of independent input – output pairs, neglecting the explicit order dependence between consequent observations. However, as can be easily understood, the time dimension does play an important role given the nature of the problem. Weather and current phenomena and the progressive fouling of the hull are important time – related dependencies that if taken into account should provide significant advantages in accurately predicting ship speed. Therefore, it is important to investigate the use of neural networks that treat the input dataset as an ordered sequence and take into consideration multiple past time steps in order to forecast the future.

Additionally, the models developed predicted ship speed only a single time step forward. Short term predictions are useful, but in terms of providing information to a weather routing system, a longer prediction horizon should be investigated.

1.3 Thesis Outline

- **Chapter 2** introduces the essential context and theoretical background surrounding artificial neural networks.
- **Chapter 3** includes the analysis of the dataset under examination and a step by step explanation of the selected data pre-processing pipeline, the procedure of transforming raw data to meaningful inputs, using various tools.
- **Chapter 4** describes the main neural network design workflow, from feature engineering to architecture selection, and the various results with emphasis on the comparison between different designs.
- **Chapter 5** notes the conclusions reached in this thesis and highlights the basis of future work.

Chapter 2

Theoretical Background

2.1 Basics of Machine Learning

In recent years, artificial intelligence has become a field of intense research activity – and rightfully so. AI has been proven transformative in dealing with various problems, from medical diagnoses, self-driving cars and speech recognition, to product and movie recommendations on e-shops and streaming services. The term "AI" refers to "the effort to automate tasks normally performed by humans" and although its popularity has grown in the recent past, its fundamental concepts date back to 1950s. For a long period of time, AI techniques were outperformed by competing methods such as Support Vector Machines, mainly due to the lack of computational power. As the ability to train more complex machine learning algorithms grew thanks to the advances made in hardware (fast, massively paraller chips in GPUs, exponential progress in storage hardware), datasets and benchmarks (readily available large datasets) and algorithms (e.g. optimization schemes), different groups of researchers achieved significant milestones, by solving problems previously thought impossible to solve and drastically improving solution accuracy in others.

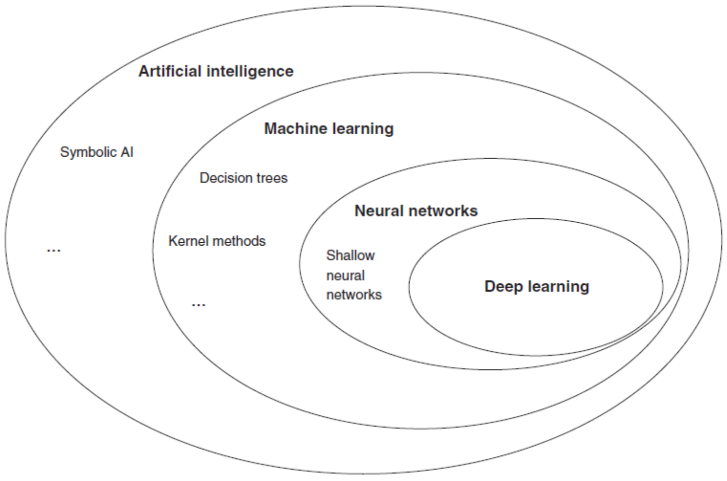


Figure 2.1: Artificial Intelligence, Machine Learning and Deep Learning [5].

While AI contains several approaches that fit the aforementioned definition, only a subset of them involves "machine learning", as shown in Fig.2.1. Machine learning is in its essence a different programming paradigm. As illustrated in Fig.2.2, instead of hard coding a complex set of rules for a specific task (e.g. pattern recognition), a machine learning system is fed various examples of input – expected output pairs and attempts to automatically discover the rules pertaining the dataset. The basis of this process lies in finding a meaningful transformation of the input dataset into a new one, suitable to solving the specific task. In machine learning there are two important phases: the training and the inference phase. In the training phase a pre-provided blueprint for the rules, encapsulated in a model, is tuned for a number of epochs in ways that result in an output progressively closer to the desired, by utilizing the distance between the algorithm's current and expected outputs as a feedback signal. Once the error has reduced adequately, the model is ready for the inference phase, where the learned rules are applied on new data – data that the training process has never seen – in order to produce original answers.

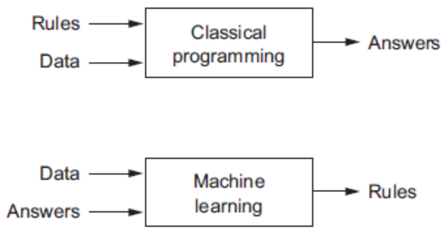


Figure 2.2: Machine Learning as a new programming paradigm [5].

Artificial neural networks are, in turn, a subfield of machine learning that focuses on finding successive layers of meaningful representations. Their name is derived from the loose resemblance to the way neurons in human brains are connected in a layered fashion. Here, the input data flows sequentially from one layer to the next, with each one of them distilling its information by applying a new transformation on the representation of the data. Each layer is characterized by a set of numerical values, called weights, that get altered systematically in a way that minimizes a certain loss function between the expected and current model output. The calculated loss score is then fed back to the model in order to adjust the weights in a direction that will lower the error, with the help of an optimizing function. If this process is repeated for an adequate number of iterations, the weights that minimize the loss score should be reached and the model will be trained. This typical neural network workflow is visualized in Fig.2.3.

A neural network containing a large number of successive layers is called deep, in contrast to neural networks containing one or two layers, which are called shallow. In deep learning all layers of representation are tuned at the same time, meaning that whenever the model adapts to a change, all other features that depend on it automatically adapt too. Stacking multiple layers allows increasingly complex, abstract representations to be learned by breaking them down into long series of intermediate steps - layer by layer.

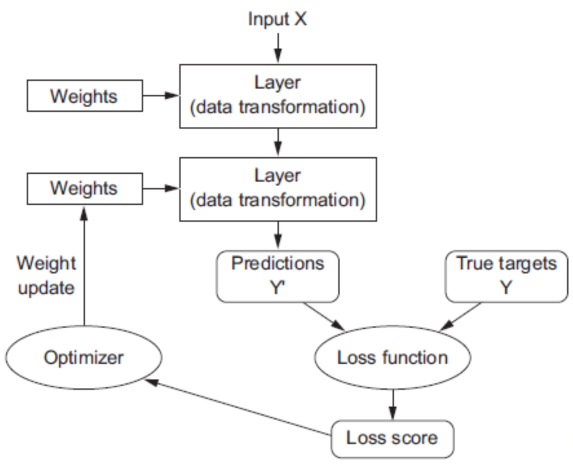


Figure 2.3: Deep Neural Network workflow [5].

The basic data structure in machine learning is multidimensional arrays called tensors. Ordered data like time series and sequences are stored in 3D tensors of shape:

$$(samples, timesteps, features)$$

Each sample contains a sequence of vectors (2D tensor) formed by the selected number of timesteps per sample and the corresponding features, as shown in Fig.2.4. However, deep learning models do not process the entire dataset at once; instead, the data, already sliced into samples, is grouped into small batches, called mini-batches.

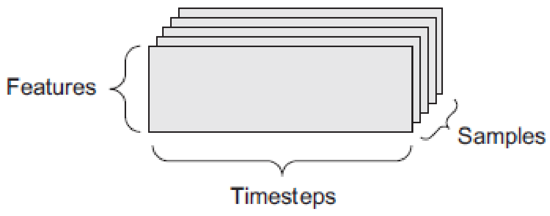


Figure 2.4: Time series data ordering [5].

As already mentioned, the building block of neural networks is the layer, a data-processing unit that applies transformations to the input data and extracts refined representations of it. The trainable parameters (weights) of the layer are initially randomly filled with small values and progressively tuned according the feedback signal provided by the chosen loss function, during the process of training. The gradual adjustment of the weights takes place after every *batch size* (usually between 8 and 128) number of mini-batches are processed and is based on the calculation of the gradient of the loss as a function of the network's coefficients, using an optimizing algorithm called Stochastic Gradient Descent (SGD). As shown in Fig.2.5, the direction opposite to the one indicated by the local gradient points towards the minima of the loss hyper-surface above the weight hyper-plane. There are multiple variants of SGD optimizer, such as RMSProp, developed to overcome certain issues of SGD, like getting stuck in local minima instead of approximating the global minimum.

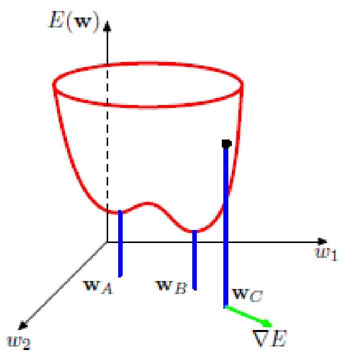


Figure 2.5: Geometrical view of the error function $E(w)$ as a surface sitting over the weight space. Point w_A is a local minimum and w_B is the global minimum. At any point w_C, the local gradient of the error surface is given by the vector ∇E [6].

In addition to the number of layers, the number of units per layer, the batch size, the loss function and the optimizing scheme, essential for defining the model architecture to be compiled are also:

- The metrics that will be used to monitor the progress during the training and testing phases,
- The number of epochs, which correspond to the iterations over the entire training dataset the training loops will go through.

2.2 Machine Lerning Workflow

At this point, it should be noted that designing and training a model is only a small part of the established approach to machine-learning problems, often referred to as the universal workflow of machine learning [5]. First of all, it is important to determine whether machine learning is, in fact, the right tool for the problem in hand. As machine learning and more specifically deep learning reach a level of public attention and industry investment never seen before, researchers often end up in a situation of having a deep-learning hammer, and every problem starting to look like a nail. In some instances, non-machine-learning techniques will work equally well or perhaps even better, at a significantly lower cost.

Giving a definite answer to the above question relies heavily on a detailed definition of the problem studied and the nature of the expected outcome. Moreover, it should be confirmed that the available dataset contains enough information in every input example to predict the output and, in total, sufficient instances for the model to learn the input-output relationship, if there is one, to begin with. Ship speed prediction using historical data measured on-board by a variety of sensors, falls under the category of multi-input time series forecasting, a common deep–learning regression problem. In addition, utilizing deep learning can prove beneficial in terms of accuracy, since a notable gap was observed when examining related work in Chapter 1.2., while the theoretical background of the problem, which has been studied for years by marine engineers, assures the existence of a relationship between the available features and the vessel's velocity.

Next, the evaluation process is prepared and the way of reliably measuring the success of the trained model is identified. Designing the validation process includes splitting the data into three homogeneous, nonoverlapping sets: a training set, a validation set and a test set. In the case of temporal prediction, the validation and test data should come from time intervals after the training data and the data preprocessing code must guard against data leakage between them, so that the model's predictive power is not overestimated. Defining the metric of success in a given problem indicates the suitable loss function. Two metrics commonly used to measure the accuracy of the network in regression problems are the Mean Squared Error and Mean Absolute Error:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2 \tag{2.1}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i) \tag{2.2}$$

where $\hat{Y}_i$ are the predicted values and Y_i the vector of observed values of the variable being predicted.

As mentioned in the previous chapter, the data ought to then be vectorized into tensors, the common data structure of machine learning. If different features of the data take values in different ranges, the data should be z-normalized to zero mean and unit standard deviation for each feature, to ensure good training performance, as values in vastly different scales result in weights of incomparable ranges that vary greatly from epoch to epoch, thus hindering the ability to locate the loss function minimum.

Once the tensors of input and target data are ready, the development of models can begin. First of all, it is important to establish a common sense baseline. In time series forecasting, "predicting" that the future value of a time series will be the same as the last input data point is a popular non-machine-learning standard that every trained model should beat, in order to demonstrate that machine learning can truly add value to the solution, as discussed earlier. Afterwards, the deep learning models can be built. Many times, picking the right network topology is more art than science. Some general guidelines, based on already tested applications, do exist, but specific problems may require further experimentation. Based on the nature of the time series problem under study, choosing some model parameters is more straightforward than others; in a prediction problem no last-layer activation is required, the Mean Squared Error is used as the loss function and the *RMSprop* with its default learning rate is used as the optimizer. The remaining parameters, like the type and number of layers, the number of units each layer contains, the batch size and the epochs for which each model was trained, are initially set to values fetched from available examples.

These are the parameters (often called hyperparameters so as to discern them from the weights of each layer) that are tuned to maximize the model's accuracy. The ideal model is one that stands right at the border between underfitting and overfitting, i.e. between optimization (best fit on the data seen during training) and generalization (being able to make accurate predictions for unseen data). Adding more layers, making each layer bigger in terms of units and training the model for more epochs are some of the ways the model can gain sufficient capacity to overfit the training data. By monitoring the training and validation losses through the chosen metrics, the point where the model's accuracy on the validation dataset begins to degrade signals the crossing to the overfitting region.

Hyperparameter tuning usually takes the most time, since the models are repeatedly modified, trained, evaluated on the validation dataset, modified again and so on. It should be highlighted that every time feedback from the validation process is used to tune the model, external information is leaked into the model. In case this is done systematically over many iterations, the model will eventually overfit to the validation process - even though it was never directly trained on any of the validation data - rather than being able to generalize well to fit other data. This is where the testing dataset proves its value as a set of never seen before instances. The purpose of the test set is to provide an unbiased estimate of the model's accuracy after hyperparameter tuning. A significantly worse performance on the test set in comparison with the performance measured on the validation data, is an indicator of either an unreliable validation procedure or overfitting to the validation data [7], [12], [13].

2.3 Linear Neural Network

The single hidden layer linear model is often referred to as a Perceptron. A Perceptron is composed of a single layer of Threshold Logic Units (TLUs). In this thesis, the tested Perceptron is also fully connected (or dense), meaning that all its neurons are connected to every neuron of the input layer. With each input connection associated with a weight, the TLU computes a weighted sum of its inputs [7]:

$$z = w_1 x_1 + w_2 x_2 + ... + w_n x_n = \boldsymbol{x^T w} \tag{2.3}$$

then applies a step function, the most common being the Heaviside step function, to that sum and outputs the result:

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = step(z) \tag{2.4}$$

A typical TLU is illustrated in Fig.2.6:

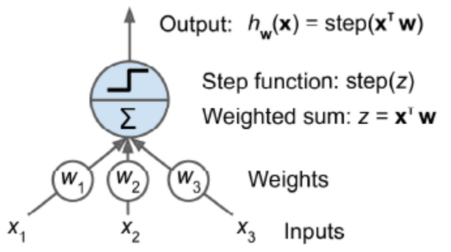

Output: $h_{\mathbf{w}}(\mathbf{x}) = \text{step}(\mathbf{x}^{\mathsf{T}}\,\mathbf{w})$
Step function: step(z)
Weighted sum: $z = \mathbf{x}^{\mathsf{T}}\,\mathbf{w}$
Σ
w_1
w_2
w_3
Weights
x_1
x_2
x_3
Inputs

Figure 2.6: Threshold logic unit [7].

At first, the inputs of a Perceptron are handled by the input layer; a combination of neurons that, in this case, output the last input of every feature. Then, the simple linear model computes a linear combination of the inputs and adds a bias.

$$h_{\boldsymbol{W},\boldsymbol{b}}(\boldsymbol{X}) = \phi(\boldsymbol{XW} + \boldsymbol{b}) \tag{2.5}$$

where:

- $\boldsymbol{X}$ represents the matrix of input features,

- weight matrix $\boldsymbol{W}$ contains all the connection weights except for the ones from the bias neuron (one row per input neuron and one column per artificial neuron),
- bias vector $\boldsymbol{b}$ contains all the connection weights between the bias neuron and the artificial neurons,
- ϕ represents the activation function

The training algorithm of a Perceptron was introduced by Roseblatt and its basis lies on the principle of reinforcing connections that help reduce the error. As the model makes its predictions for each batch of training instances, for every output neuron that failed to produce an accurate prediction, the connection weights from the inputs that would have produced a correct result are reinforced. The Perceptron weight update rule is given by:

$$w_{i,j}^{(nextstep)} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i \tag{2.6}$$

where:

- $w_{i,j}$ is the connection weight between the i^{th} input neuron and the j^{th} output neuron,
- x_i is the i^{th} input value of the current training instance,
- $\hat{y}_j$ is the output of the j^{th} output neuron for the current training instance,
- y_j is the target output of the j^{th} output neuron for the current training instance,
- η represents the learning rate

The model needs to predict multiple time steps, from a single input time step with a linear projection. A simple linear model based on the last input time step does better than the baseline, but it is clear that a Perceptron is often incapable of learning complex patterns, as its decision boundary is strictly linear.

2.4 Multi Layer Perceptron

Some of the limitations of the simple linear model can be eliminated by stacking multiple Perceptrons together, resulting in an architecture called Multilayer Perceptron (MLP). An MLP is composed of a simple stack of densely connected layers of TLUs, called hidden layers. Every layer except for the output layer includes a bias. A typical MLP configuration is shown in Fig. 2.7.

MLPs are trained with the help of the backpropagation algorithm, that was introduced by D. Rumelhart, G. Hinton and R. Williams. Although it remains a Gradient Descent technique (discussed in Ch. 2.2), an efficient scheme is employed to calculate the gradients. The algorithm utilizes two passes – a forward and a backward – to compute the gradient of the model's error, relative to every parameter. The forward pass is exactly like making predictions using the simple linear model, with the exception that all intermediate results are preserved. Once the model is done computing the network's output error using the selected loss function, the contribution of each output connection to the error is calculated using the chain rule. This process extends gradually backwards (reverse pass) until the input layer is reached, thus measuring the error gradient across all the connection weights.

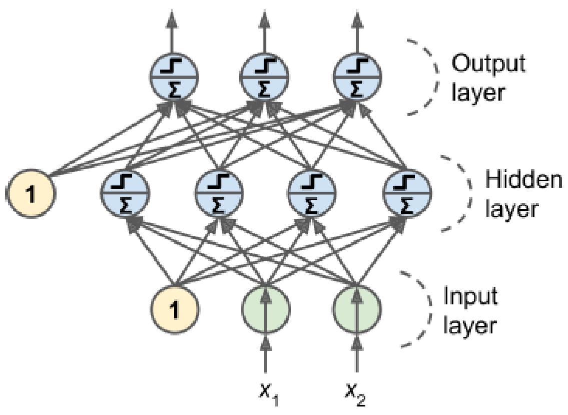


Figure 2.7: Example configuration of an MLP: Two inputs, one hidden layer of four units and three output neurons - the biases are usually implicit [7].

Utilizing these error gradients, a Gradient Descent step is taken in the direction of reducing the error.

In order for the backpropagation algorithm to work a well-defined non zero derivative must be available, so the step function of the TLUs is replaced by the logistic sigmoid function:

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{2.7}$$

Other popular activation functions include:

- the hyperbolic tangent function $tanh(z)$
- the Rectified Liner Unit $ReLU = max(0, z)$, which has become the default, thanks to its fast computation.

An activation function other than the step function is what differentiates a deep stack of layers from a single layer. If several linear transformations are simply chained together, the result will be another linear transformation. Activation functions provide the needed nonlinearity between the layers, thus allowing the model to solve more complex problems.

Focusing on regression MLPs, one output neuron is needed per output dimension. The argument being passed to each Dense layer is the number of hidden units of the layers, which translate to the number of dimensions in the representation space of the layer. More units allow more complex representations, but also greater computational cost and higher probability of overfitting (overoptimizing on the training dataset and failing to generalize to data outside of it). Thus, two key decisions regarding the architecture of a Dense model regard the number of layers to be used and the number of hidden units each layer will have. The loss function to be used during training is typically the Mean Squared Error, although the Mean Absolute Error is also monitored as a metric due to the high number of outliers in the training set.

2.5 Convolutional Neural Network

Convolutional Neural Networks (CNNs or convnets) originally emerged from the study of the brain's visual cortex, and they have been used in image recognition since the 1980s. In the last few years, thanks to the increase in computational power and the amount of available training data, CNNs have managed to achieve superhuman performance on some complex visual tasks [7]. The same properties that make convnets excel at computer vision, like their ability to operate convolutionally, extract features from local input patches, allowing for representation modularity and data efficiency, also make them highly relevant to sequence processing. Time can be treated as a spatial dimension, like the height or width of a 2D image. Such 1D convnets can be competitive with Recurrent Neural Networks on certain sequence-processing problems, usually at a considerably cheaper computational cost [5].

As shown in Fig.2.8, by extracting local 1D patches (subsequences) from sequences, 1D convolution layers gain the ability to detect very short sequential patterns. Because the same input transformation is performed on every patch, a pattern learned at a certain position in a sequence can later be recognized at a different position, making 1D convnets translation invariant (for temporal translations).

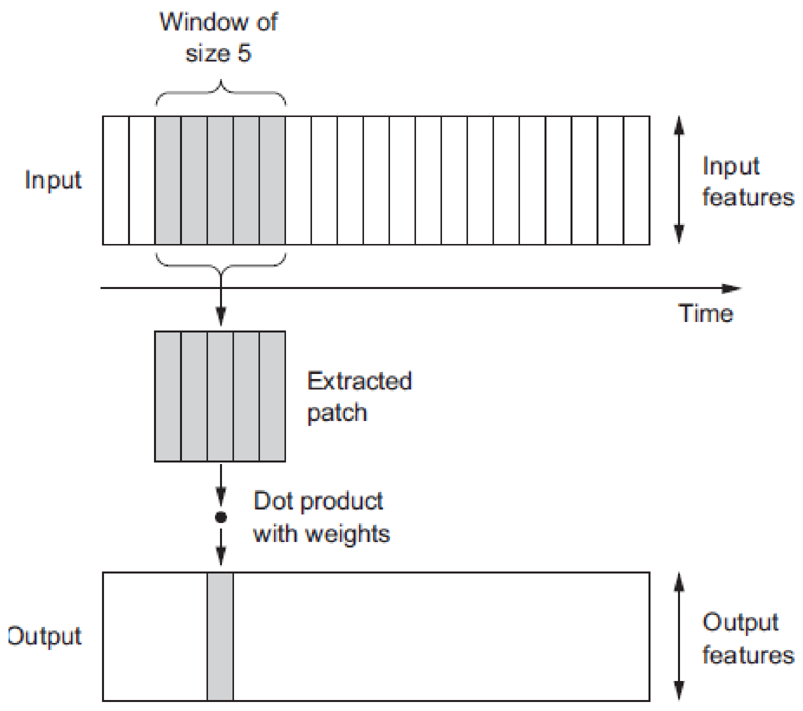


Figure 2.8: How 1D convolution works; each output timestep is obtained from a temporal patch in the input sequence [5].

Validation accuracy is expected to be somewhat less than that of the Long - Short Term Memory networks (discussed next), but runtime is faster on both CPU and GPU.

2.6 Recurrent / Long - Short Term Memory Neural Network

The fundamental deep – learning algorithms for sequence processing are Recurrent Neural Networks (RNNs). Feed-forward neural networks have one thing in common; no internal memory. Each input data point shown to them is processed independently, with no state maintained in between. However, it is obvious that the prediction of ship speed depends not only on the current ME *RPM* and Shaft Power but also on their history. A RNN iterates through the sequence elements and keeps a state regarding information to what is has seen so far, while processing sequences. This is achieved thanks to an internal loop, with the help of which each observation is no longer processed in a single step and immediately discarded.

A recurrent neural network differs from the aforementioned feedforward neural networks due to its connections pointing backward. At each time step, as illustrated in Fig.2.9, a recurrent neuron receives the inputs $x(t)$ as well as its own output from the previous time step, $y(t-1)$. Because of this, each recurrent neuron has two sets of weights w_x, w_y: one for the inputs $x(t)$ and another for the outputs of the previous time step $y(t-1)$. This means that the recurrent layer, as a whole, is assigned two weight matrices $\boldsymbol{W_x}$ and $\boldsymbol{W_y}$.

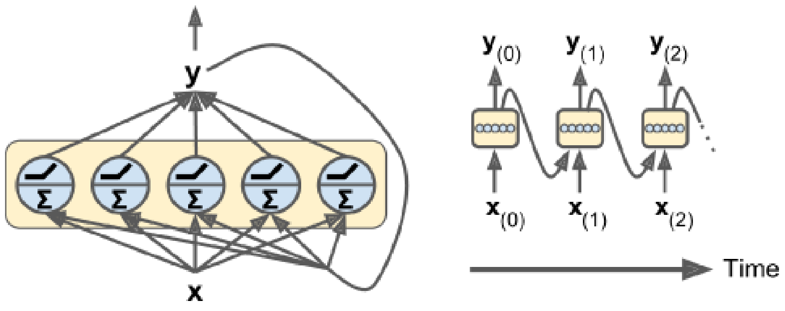


Figure 2.9: A layer of recurrent neurons (left) unrolled through time (right) [7].

Similarly to the feedforward networks, a recurrent layer's output can be computed in one shot for every input batch by gathering all the inputs at time step t in a single matrix $\boldsymbol{X}(t)$:

$$Y_{(t)} = \phi(\boldsymbol{X}_{(t)}\boldsymbol{W}_x + \boldsymbol{Y}_{(t-1)}\boldsymbol{W}_y) + b \tag{2.8}$$

where:

- $\boldsymbol{Y}_{(t)}$ is an $m * n_{neurons}$ matrix containing the layer's outputs at time step t for each instance in the batch (m is the number of instances in the batch),
- $\boldsymbol{X}_{(t)}$ is an $m * n_{inputs}$ matrix containing the inputs for all instances (n_{inputs} is the number of input features,
- $\boldsymbol{W}_x$ is an $n_{inputs} * n_{neurons}$ matrix containing all the connection weights for the inputs of the current time step,
- $\boldsymbol{W}_y$ is an $n_{neurons} * n_{neurons}$ matrix containing all the connection weights for the outputs of the previous time step,
- b is a vector size $n_{neurons}$ containing each neuron's bias term.

Since the output of a recurrent neuron at time step t is a function of all the inputs from previous time steps, one could say that it has a form of memory. A part of a neural network that preserves some state across time steps is called a memory cell (or simply a cell). A single recurrent neuron, or a layer of recurrent neurons, is a very basic cell, capable of learning only short patterns (typically about 10 steps long, but this varies depending on the task).

In general a cell's state at time step t, denoted $h(t)$ (the "h" stands for "hidden"), is a function of some inputs at that time step and its state at the previous time step: $\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$. Its output at time step t, denoted $y(t)$, is also a function of the previous state and the current inputs.

Simple RNN architectures do have a major issue. Long term dependencies, based on information seen many time steps before, are often impossible to learn due to the vanishing gradient problem. The Long Short-Term Memory (LSTM) algorithm was developed by Hochreiter and Schmidhuber as a solution to this problem. The essence of LSTM lies in saving information for later by adding a way to carry information across many timesteps.

The key idea is that the network can learn what to store in the long-term state, what to throw away, and what to read from it. The internal configuration of a LSTM cell is shown in Fig.2.10. As the long-term state $\boldsymbol{c}_{(t-1)}$ traverses the network from left to right, it first goes through a forget gate, dropping some memories, and then it adds some new memories via the addition operation (which adds the memories that were selected by an input gate). The result $\boldsymbol{c}_{(t)}$ is sent straight out, without any further transformation. So, at each time step, some memories are dropped and some memories are added. Moreover, after the addition operation, the long-term state is copied and passed through the tanh function, and then the result is altered by the output gate. This produces the short-term state $\boldsymbol{h}_{(t)}$ (which is equal to the cell's output for this time step,$\boldsymbol{y}_{(t)}$).

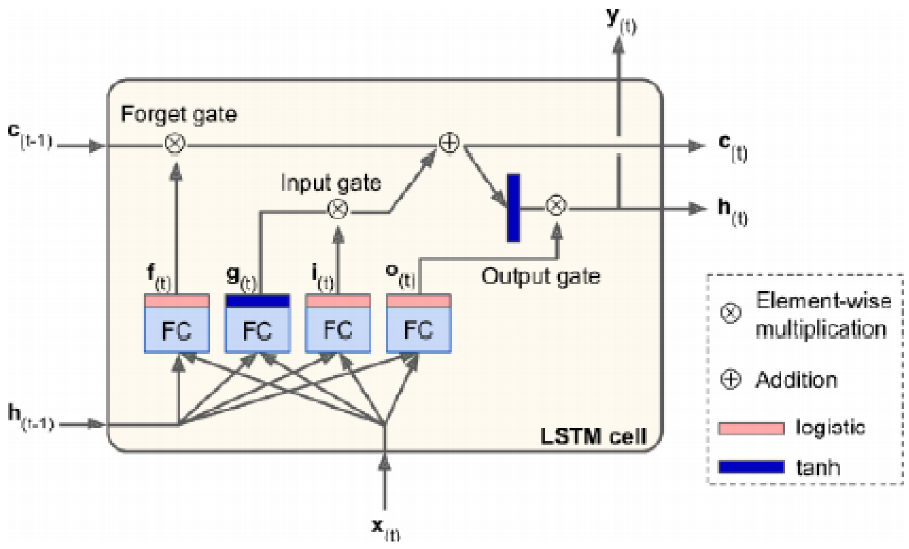


Figure 2.10: LSTM cell (FC: Fully Connected) [6].

First, the current input vector $\boldsymbol{x}_{(t)}$ and the previous short-term state $\boldsymbol{h}_{(t-1)}$ are fed to four different fully connected layers. They all serve a different purpose:

- The main layer is the one that outputs $g_{(t)}$. It has the usual role of analysing the current inputs $\boldsymbol{x}_{(t)}$ and the previous (short-term) state $\boldsymbol{h}_{(t-1)}$. In a basic cell, there is nothing other than this layer, and its output goes straight out to $\boldsymbol{y}_{(t)}$ and $\boldsymbol{h}_{(t-1)}$. In contrast, in an LSTM cell this layer's output does not go straight out, but instead its most important parts are stored in the long-term state (and the rest is dropped).

- The three other layers are gate controllers. Since they use the logistic activation function, their outputs range from 0 to 1. As you can see, their outputs are fed to element-wise multiplication operations, so if they output 0s they close the gate,and if they output 1s they open it. Specifically:
  - The forget gate (controlled by $f_{(t)}$) controls which parts of the long-term state should be erased.
  - The input gate (controlled by $i_{(t)}$) controls which parts of $g_{(t)}$ should be added to the long-term state.
  - The output gate (controlled by $o_{(t)}$) controls which parts of the long term state should be read and output at this time step, both to $\boldsymbol{h}_{(t)}$ and to $\boldsymbol{y}_{(t)}$.

In short, an LSTM cell can learn to recognize an important input (that's the role of the input gate), store it in the long-term state, preserve it for as long as it is needed (that's the role of the forget gate), and extract it whenever it is needed. This explains why these cells have been successful at capturing long-term patterns in time series, long texts, audio recordings, and more.

The following equations summarize how to compute the cell's long-term state, its short-term state, and its output at each time step for a single instance.

$$\boldsymbol{i}_{(t)} = \sigma(\boldsymbol{W}_{\boldsymbol{xi}}^{\boldsymbol{T}}\boldsymbol{x}_{(t)} + \boldsymbol{W}_{\boldsymbol{hi}}\boldsymbol{h}_{(t-1)} + \boldsymbol{b}_i) \tag{2.9}$$

$$\boldsymbol{f}_{(t)} = \sigma(\boldsymbol{W}_{\boldsymbol{xf}}^{\boldsymbol{T}}\boldsymbol{x}_{(t)} + \boldsymbol{W}_{\boldsymbol{hf}}\boldsymbol{h}_{(t-1)} + \boldsymbol{b}_f) \tag{2.10}$$

$$\boldsymbol{o}_{(t)} = \sigma(\boldsymbol{W}_{\boldsymbol{xo}}^{\boldsymbol{T}}\boldsymbol{x}_{(t)} + \boldsymbol{W}_{\boldsymbol{ho}}\boldsymbol{h}_{(t-1)} + \boldsymbol{b}_o) \tag{2.11}$$

$$\boldsymbol{g}_{(t)} = tanh(\boldsymbol{W}_{\boldsymbol{xg}}^{\boldsymbol{T}}\boldsymbol{x}_{(t)} + \boldsymbol{W}_{\boldsymbol{hg}}\boldsymbol{h}_{(t-1)} + \boldsymbol{b}_g) \tag{2.12}$$

$$\boldsymbol{c}_{(t)} = \boldsymbol{f}_{(t)} \times \boldsymbol{c}_{(t-1)} + \boldsymbol{i}_{(t)} \times \boldsymbol{g}_{(t)} \tag{2.13}$$

$$\boldsymbol{y}_{(t)} = \boldsymbol{h}_{(t)} = \boldsymbol{o}_{(t)} \times tanh(\boldsymbol{c}_{(t)}) \tag{2.14}$$

where:

- $\boldsymbol{W_{xi}}, \boldsymbol{W_{xf}}, \boldsymbol{W_{xo}}, \boldsymbol{W_{xg}}$ are the weight matrices of each of the four layers for their connection to the input vector $x_{(t)}$,
- $\boldsymbol{W_{hi}}, \boldsymbol{W_{hf}}, \boldsymbol{W_{ho}}, \boldsymbol{W_{hg}}$ are the weight matrices of each of the four layers for their connection to the previous short-term state $h_{(t-1)}$,
- $\boldsymbol{b_i}, \boldsymbol{b_f}, \boldsymbol{b_o}, \boldsymbol{b_g}$ are the bias terms for each of the four layers. Note that TensorFlow initializes bf to a vector full of 1s instead of 0s. This prevents forgetting everything at the beginning of training.

Besides all mentioned above, LSTM cells can be modified to run on the GPU instead of the CPU, making them even faster as they utilize the parallel processing capabilities of the GPU.

Chapter 3

Analysis of Ship Data

The constantly growing Python ecosystem has been for a long time the dominant platform for applied machine learning. Python is a popular general-purpose interpreted programming language supported by a great variety of libraries. More precisely, in this thesis the machine learning framework is provided by Keras. Keras is a high-level Deep Learning API, developed by F. Chollet, that relies on the TensorFlow backend for its operations. Via Tensorflow, Keras is able to run fast and efficiently on GPU utilizing the NVIDIA CUDA Deep Neural Network Library (cuDNN). Other libraries used include NumPy for providing array operations, Matplotlib for plotting data and Pandas for high – performance data handling featuring explicit tools for handling timeseries.

The code is run on Google Collab's notebooks, where high end cloud GPU support is offered free of charge for research purposes.

3.1 Data Acquisition

The dataset in hand was available at the Laboratory of Marine Engineering / National Technical University of Athens and consists of circa 350.000 measurements of different attributes collected from a containership during a period of 10 months. Several determinative characteristics of the containership and its voyages in question had not been disclosed, thus undermining the ability to physically interpret and validate the results. The knowledge of the ship's main dimensions, DWT, service speed, floating position (draft, angle of heel and trim), its engine's Mean Continuous Rating and the route it operated in, so as to gain access in weather and currents data, would have drastically increased the quality and quantity of the instances contained in the dataset and set a solid theoretical basis for our work, similar to the papers mentioned in Chapter 1.2.

The variables provided by the dataset were measured in 1-minute intervals and include the following:

- Entry Date and Time
- Average Main Engine Speed, in rpm
- Average Main Engine Shaft Power, in kW
- Average Speed Through Water, in kn
- Cumulative Main Engine Mass Fuel Oil Consumption, in kg

- Cumulative Main Engine Volume Fuel Oil Consumption, in m^3, and
- Average Main Engine Fuel Oil Temperature at inlet flowmeter, in $^\circ C$

The term “average” refers to the average value of the measured feature within each 1-minute interval, while the “cumulative” attributes represent the growing sum of successive additions measured during each interval.

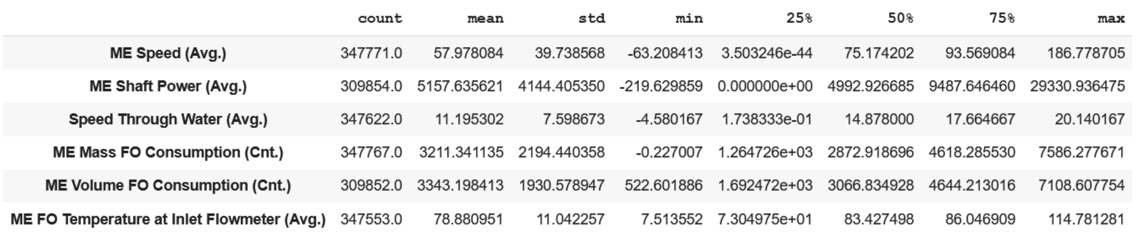
The given dataset contained several gaps (missing measurements) not only in bounded time periods, but also in random time stamps throughout the dataset, while false measurements and noise were also expected due to possible sensor malfunctions. Having a dataset “spattered” with various problematic observations is certainly in line with the average medium to low quality data acquisition systems in vessels today and can be considered as an opportunity to establish an in depth pre – processing method that, in the end, will extract a comprehensive ensemble, suitable for model training.

3.2 Data Preprocessing

3.2.1 Exploratory Data Analysis

In machine learning a model is only as good as the data it was trained on. Before any predictions are made, it is of great importance to statistically examine the dataset in detail and establish a suitable pipeline to manipulate its features in a way that will improve the network's success in the training phase.

Once the data is loaded in the correct format and chronologically sorted, it is important to quickly identify some of the most visible issues, in a process called exploratory data analysis. Calculating descriptive statistics, as shown in Fig.3.1, can help us get an idea of the distribution and statistical characteristics of each feature. Some of the immediately distinguishable problems include several missing values (around 1200 for every feature and a big gap for Shaft Power in the beginning) that need to be interpolated or discarded, outliers whether due to corrupt measurements or naturally occurring extremities which should be pinpointed and handled and negative values that sometimes do represent a natural phenomenon (negative speed describes astern propulsion) but in others have no physical meaning.



	count	mean	std	min	25%	50%	75%	max
ME Speed (Avg.)	347771.0	57.978084	39.738568	-63.208413	3.503246e-44	75.174202	93.569084	186.778705
ME Shaft Power (Avg.)	309854.0	5157.635621	4144.405350	-219.629859	0.000000e+00	4992.926685	9487.646460	29330.936475
Speed Through Water (Avg.)	347622.0	11.195302	7.598673	-4.580167	1.738333e-01	14.878000	17.664667	20.140167
ME Mass FO Consumption (Cnt.)	347767.0	3211.341135	2194.440358	-0.227007	1.264726e+03	2872.918696	4618.285530	7586.277671
ME Volume FO Consumption (Cnt.)	309852.0	3343.198413	1930.578947	522.601886	1.692472e+03	3066.834928	4644.213016	7108.607754
ME FO Temperature at Inlet Flowmeter (Avg.)	347553.0	78.880951	11.042257	7.513552	7.304975e+01	83.427498	86.046909	114.781281

Figure 3.1: Descriptive statics of the raw dataset.

The total number of NaN instances per feature are as follows:

```
Entry_Date                                    0
ME Speed (Avg.)                            1204
ME Shaft Power (Avg.)                     39121
Speed Through Water (Avg.)                 1353
ME Mass FO Consumption (Cnt.)              1208
ME Volume FO Consumption (Cnt.)           39123
ME FO Temperature at Inlet Flowmeter (Avg.)  1422
```

An important feature to investigate when preparing a dataset for model training is the correlation between the available features. In a correlation matrix, values close to 1 indicate strong correlation between the variables, thus underlining a probable redundancy, as the information contained in both features will be practically the same. As expected, ME Speed, ME Shaft Power and Speed Through Water are highly correlated, while the cumulative features are of no practical value in their current form (Fig.3.2).

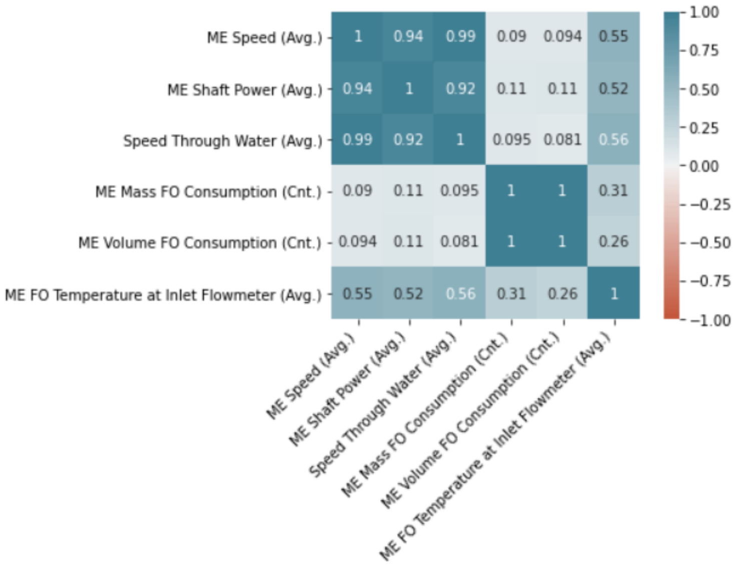


Figure 3.2: Raw data correlation matrix.

Beyond the correlation matrix, the relation between two features can be visualized via scatter plots. Special interest lies in the ME Speed vs. Speed through Water and ME Speed vs. ME Shaft Power plots. In the first one, Fig.3.3, a strong linear correlation is underlined, while the effect of adverse weather conditions is noted by some distinguishable vertical concentrations of measurements (same ME Speed but vastly different Speed through Water). The second plot, Fig.3.4, is relevant to the Propeller Law and once again the effect of different weather conditions manifests itself through the different C coefficients, in accordance to the theoretical relation.

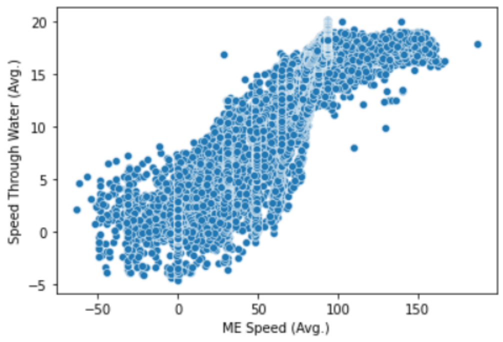


Figure 3.3: ME Speed vs. Speed through Water scatter plot.

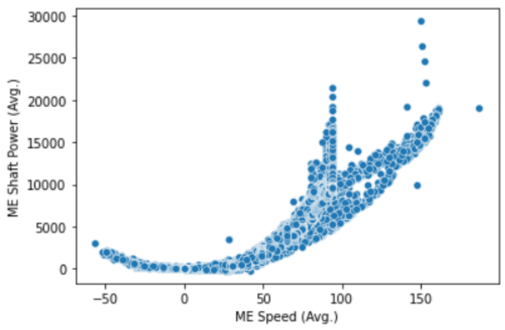


Figure 3.4: ME Speed vs. ME Shaft Power scatter plot.

Next, a suite of various plots can be utilized to gain an insight of the time series' temporal structure, each features' distribution of observations and other information. Simply visualizing every feature of the raw dataset in line plots provides some basic understanding of the distinct pattern of each feature, while it can also highlight possible trends, cycles and seasonality - which affect the selection of the proper model architecture - and the location of anomalies that will be addressed. In our case, every feature is plotted in Fig.3.5:

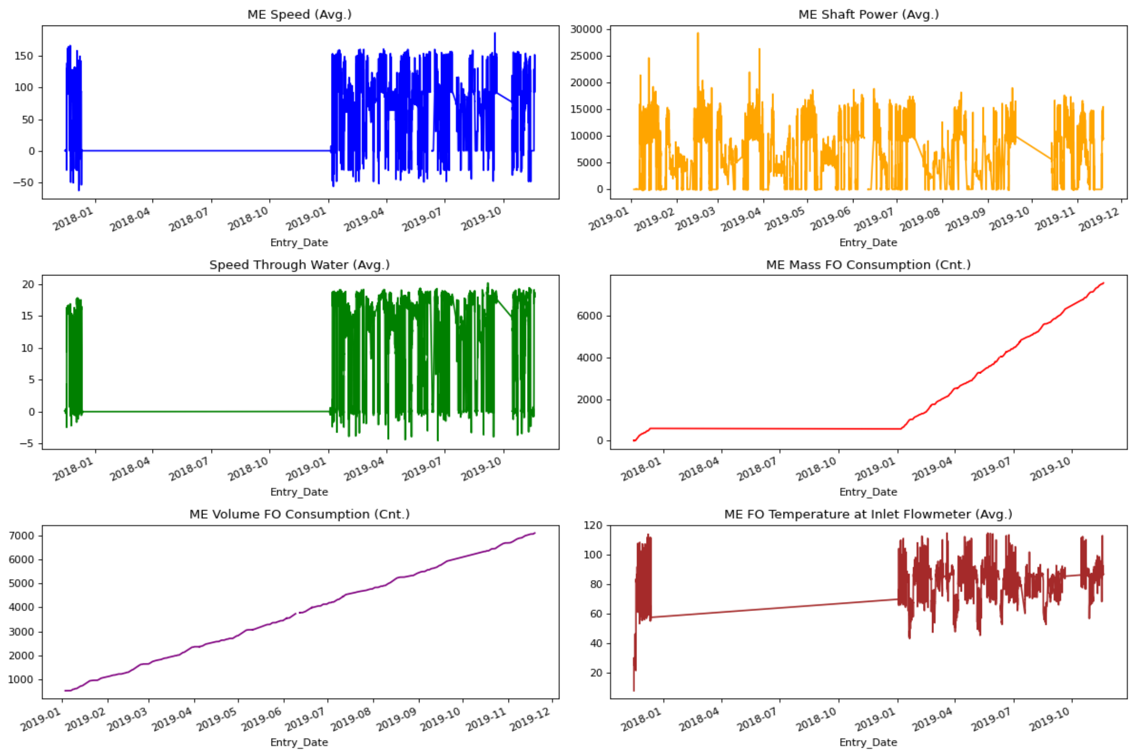


Figure 3.5: Raw data line plots

Another important visualization are histogram plots, where observations, stripped of their temporal order, are grouped into bins of height equal to the count of observations in each bin, thus highlighting the underlying distribution. Although a Gaussian distribution would indicate a well spread set of observations, the speed of a vessel is not expected to strongly resemble one. Even though the majority of the values should be clustered around its service speed, a heavy left tail around zero, due to the measurements taken in ports and harbors, and a few measurements above the service speed, are expected. Later data preparations techniques should deal with the anomalies present in Fig.3.6:

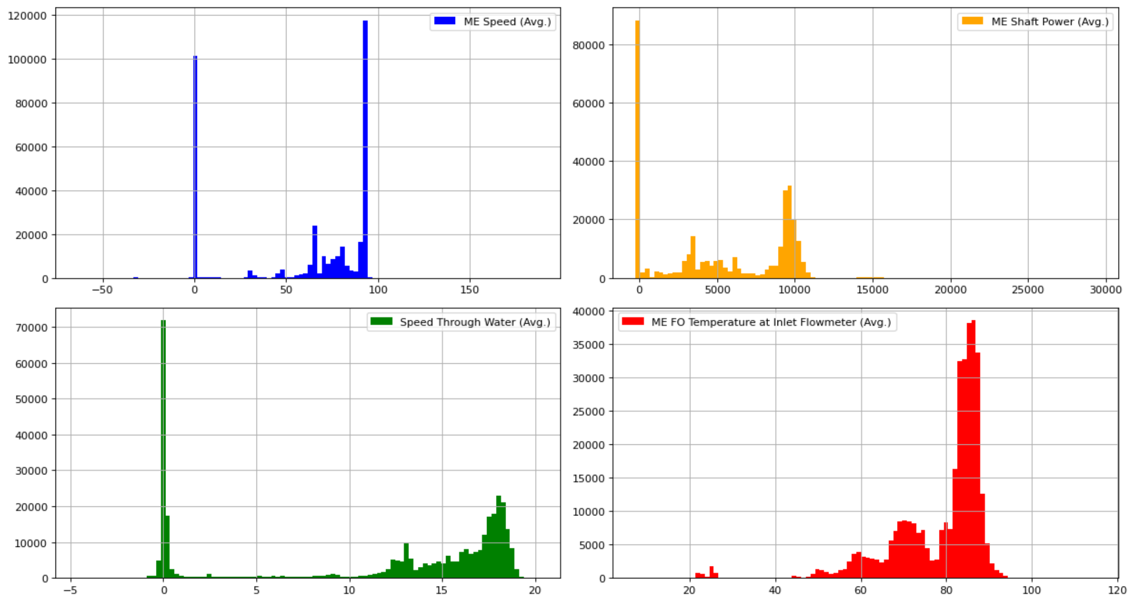


Figure 3.6: Raw data histograms

Another type of plot dealing with the dispersion of observations is the box and whisker plot. In this plot a box is drawn around the 25th (Q_1) and 75th (Q_3) percentiles of the data, a line is drawn at the median value and whiskers are drawn to summarize the general extents of observations. This can prove very helpful in spotting outliers, represented with dots above and below the whiskers, positioned at $Q_1 - 1.5IQR$ and $Q_3 + 1.5IQR$, where IQR stand for the interquartile range, from the 25th to the 75th percentile. The ME Shaft Power boxplot is illustrated in Fig.3.7. No significant outliers appear to be present at this point, but this will change after preprocessing is complete.

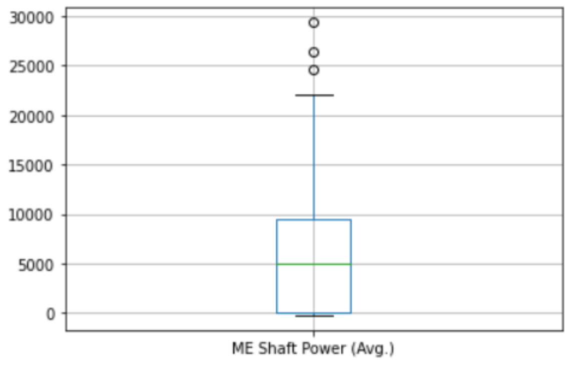


Figure 3.7: ME Shaft Power boxplot

A useful plot to explore when dealing with a time series, and especially when aiming to forecast a value based on past observations, is the lag plot. This scatter plot visualizes the relationship between each observation and a lag n of that observation (the observation at the $n-th$ previous time step) by displaying the observation at time t on the x-axis and the observation at time $t+n$ on the y-axis. Like in all scatter plots, if the points cluster along one of the two diagonals (bottom-left to top-right or bottom-right to top-left) a strong correlation is indicated, thus green-lighting its modeling. By repeating this process for various lag values, a general sense of the point where the correlation is weakening can be gained. For example, the lag plot of Speed through Water for $lag = 1$ and $lag = 5$, in Fig.3.8, Fig.3.9:

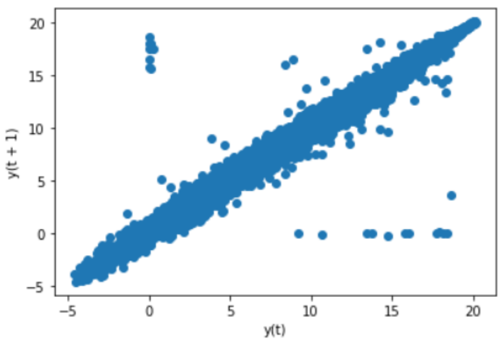


Figure 3.8: Speed through Water lag1 plot.

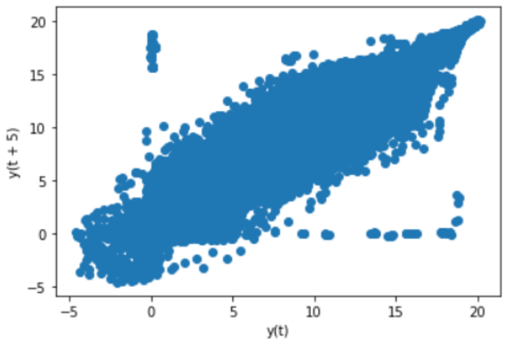


Figure 3.9: Speed through Water lag5 plot.

The strength and type of relationship between observations and their lags is also quantified using the metric called autocorrelation. In its essence, autocorrelation is defined as the correlation between a set of observation and their lag n values, resulting in a number between -1 and 1. A value close to the extremities suggests a strong correlation, while values in the region within the dotted horizontals are statistically insignificant. Both lag plots and autocorrelation plots provide useful information for the selection of an appropriate input time window width, a parameter which will be proven useful later. Considering that a vessel's speed cannot change very fast in practical operations, high autocorrelation is expected and confirmed by Fig.3.10.

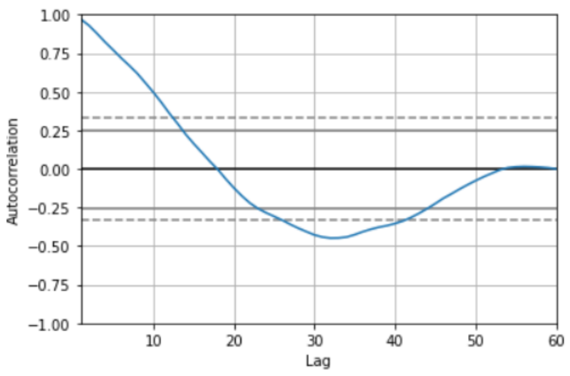


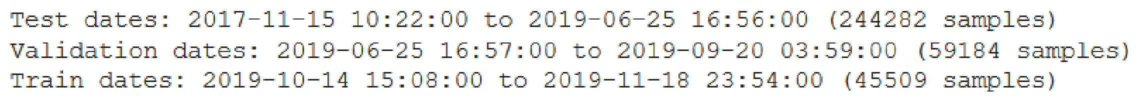
Figure 3.10: Speed through Water autocorrelation plot.

3.2.2 Pre-processing

Before any transformation is applied, the dataset must be split into the three distinct groups associated with the different phases of machine learning; a training, a validation and a testing dataset in a 70:15:15 ratio. Neglecting this step would lead to data leakage, which happens when data from outside the training dataset is used to create the model, hence allowing the model to learn something that would otherwise not know and could result in overly optimistic accuracies. It should also be noted that the data is not being randomly shuffled before splitting, as per common practice. This is for two reasons:

- It ensures that chopping the data into windows of consecutive samples is still possible.
- It ensures that the validation/test results are more realistic, being evaluated on data collected after the model was trained.

The split dataset consists of:



```
Test dates: 2017-11-15 10:22:00 to 2019-06-25 16:56:00 (244282 samples)
Validation dates: 2019-06-25 16:57:00 to 2019-09-20 03:59:00 (59184 samples)
Train dates: 2019-10-14 15:08:00 to 2019-11-18 23:54:00 (45509 samples)
```

Figure 3.11: Dataset after splitting.

First of all, empty instances (containing only the timestamp) are dropped from the datasets. The remaining NaN values are then linearly interpolated using the closest available preceding and following values. No estimation is possible on the time period of 2017 for which there are available ME Speed observations, but no Shaft Power measurements, thus the instances are dropped.

Next, a threshold is established in order to discard the observations made in harbour conditions. These observations do not contain any valuable information, are likely error – prone and because of their non-negligible count, would direct the model towards favoring results with close to zero values. Besides, forecasting the velocity during harbor maneuvering is trivial and of limited practical importance.

More specifically, the cumulative FO mass consumption is first used to build a new feature that corresponds to the rolling max of the column. Each measurement of the new feature is given by the maximum value inside a progressively expanding window between each instance's index position and the first element. A strictly increasing sequence of observations is thus created, aiming to avoid physically impossible drops in the cumulative fuel consumption. Lastly, the differences between each two consecutive values of the new feature are computed and recorded in order to indicate the positions where consumption is increased (indicative of movement). This is where the chosen threshold of 0.001 kg can be put into effect. The new feature is illustrated for the training (Fig.3.12), validation (Fig.3.13) and testing datasets (Fig.3.14).

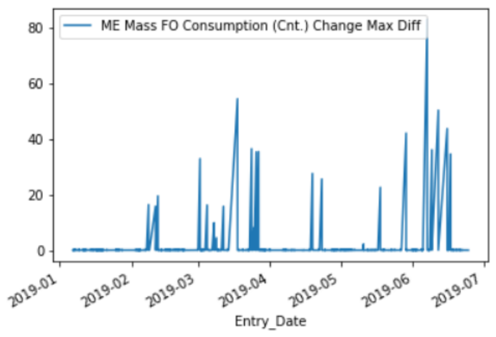


Figure 3.12: Training dataset differential change in fuel consumption.

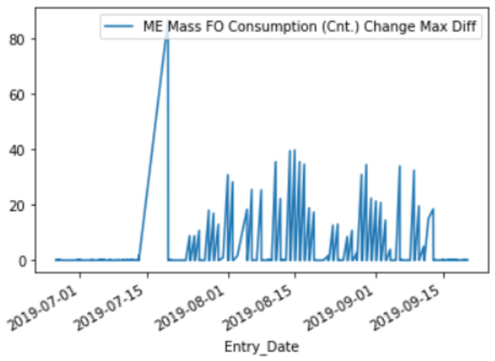


Figure 3.13: Validation dataset differential change in fuel consumption.

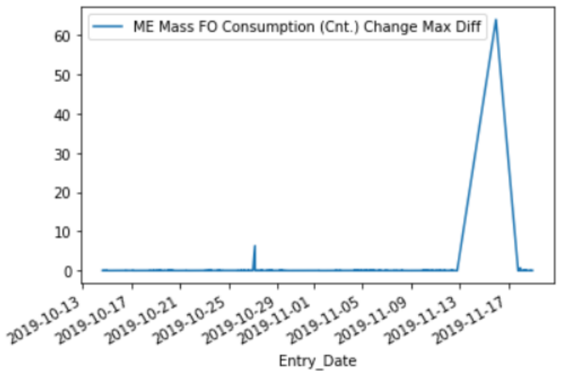


Figure 3.14: Testing dataset differential change in fuel consumption.

The next step in preprocessing deals with outliers. Based on the same principle boxplots use to detect outliers in Ch. 3.2.1, instances located outside the range defined by the minimum ($Q_1 - 1.5IQR$) and maximum ($Q_3 + 1.5IQR$) boundaries are discarded. It is important to notice that the outliers of Speed through Water, shown in Fig.3.15, are concentrated under the value of $7.5kn$ approximately, with corresponding ME Speed observations under $60rpm$, are in accordance to our effort to isolate measurements taken in ocean-going conditions.

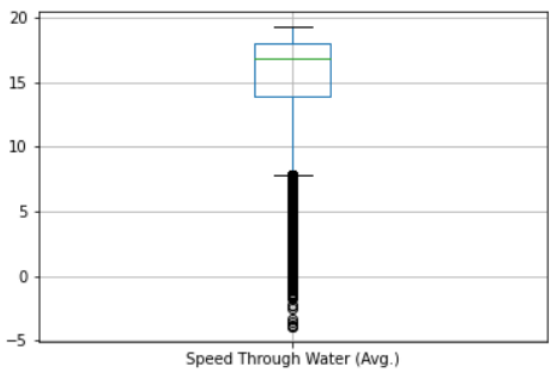


Figure 3.15: Processed Speed through Water boxplot.

Aiming to highlight the changes that took place thanks to preprocessing, the remaining training, validation and testing datasets are visualized using some characteristic plots in Fig.3.16 - Fig.3.21. The difference in histogram plots is an indicative sign of the successful handling.

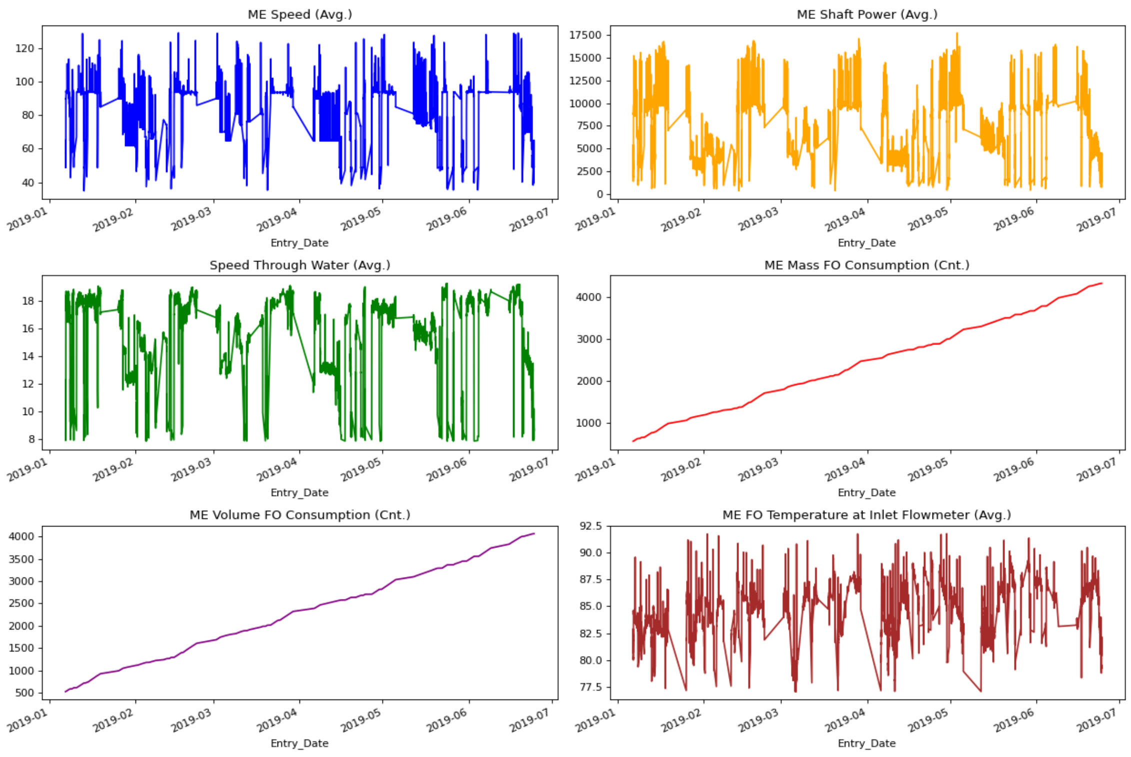


Figure 3.16: Processed training dataset line plots.

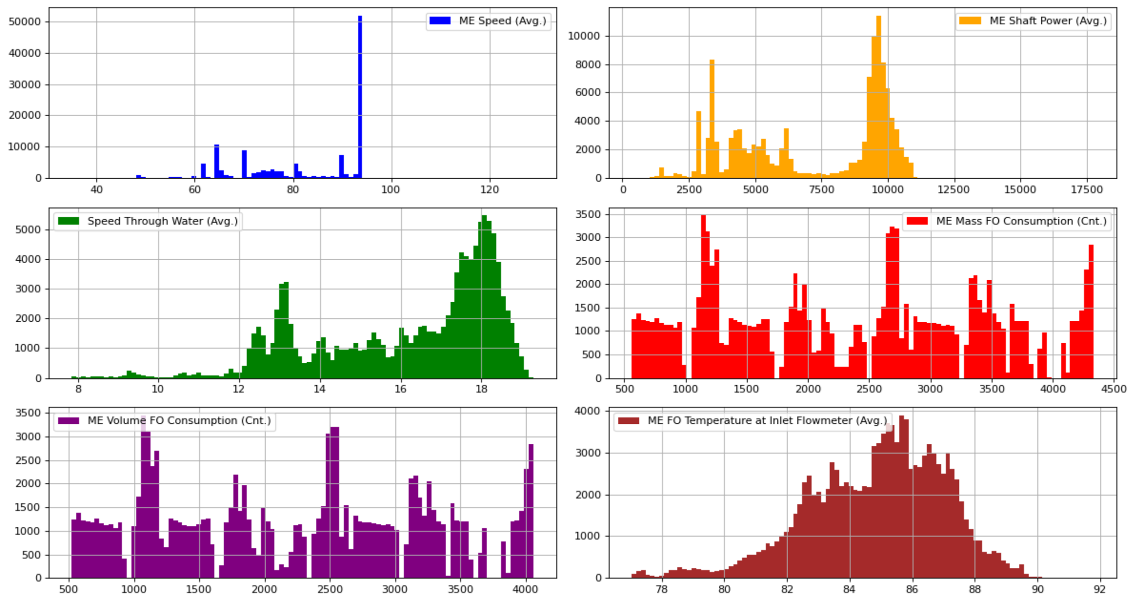


Figure 3.17: Processed training dataset histograms.

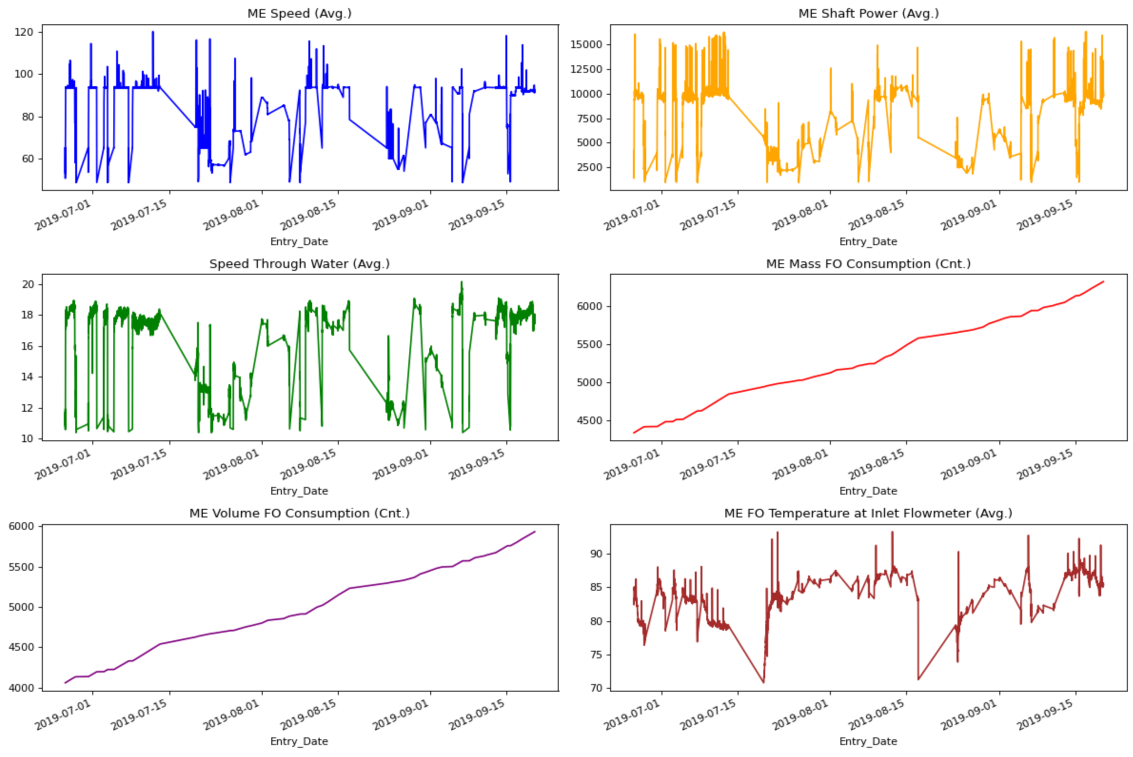


Figure 3.18: Processed validation dataset line plots.

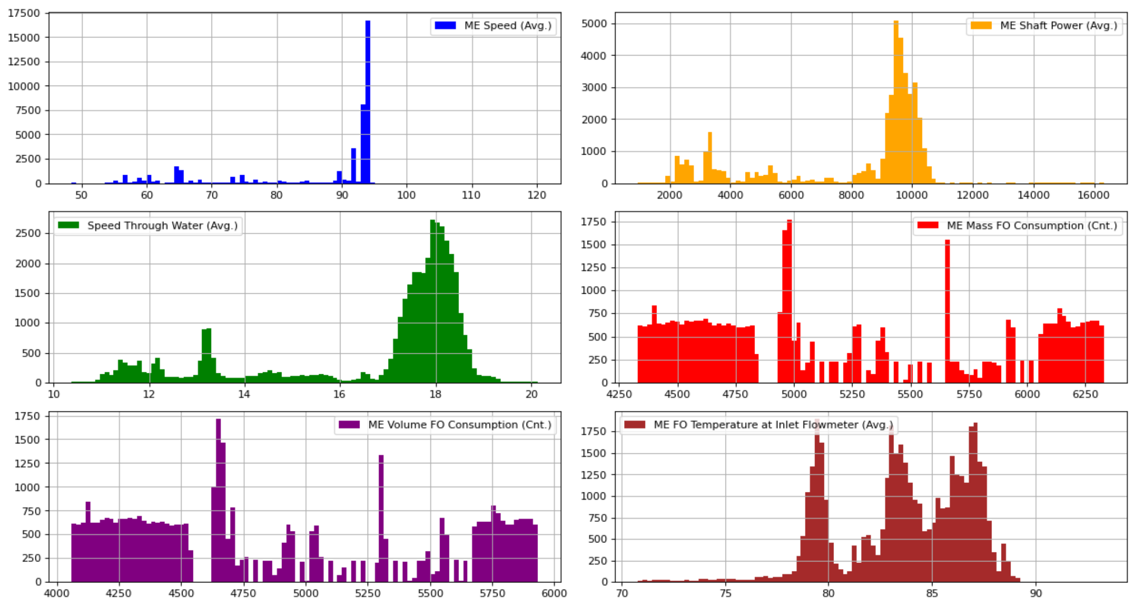


Figure 3.19: Processed validation dataset histograms.

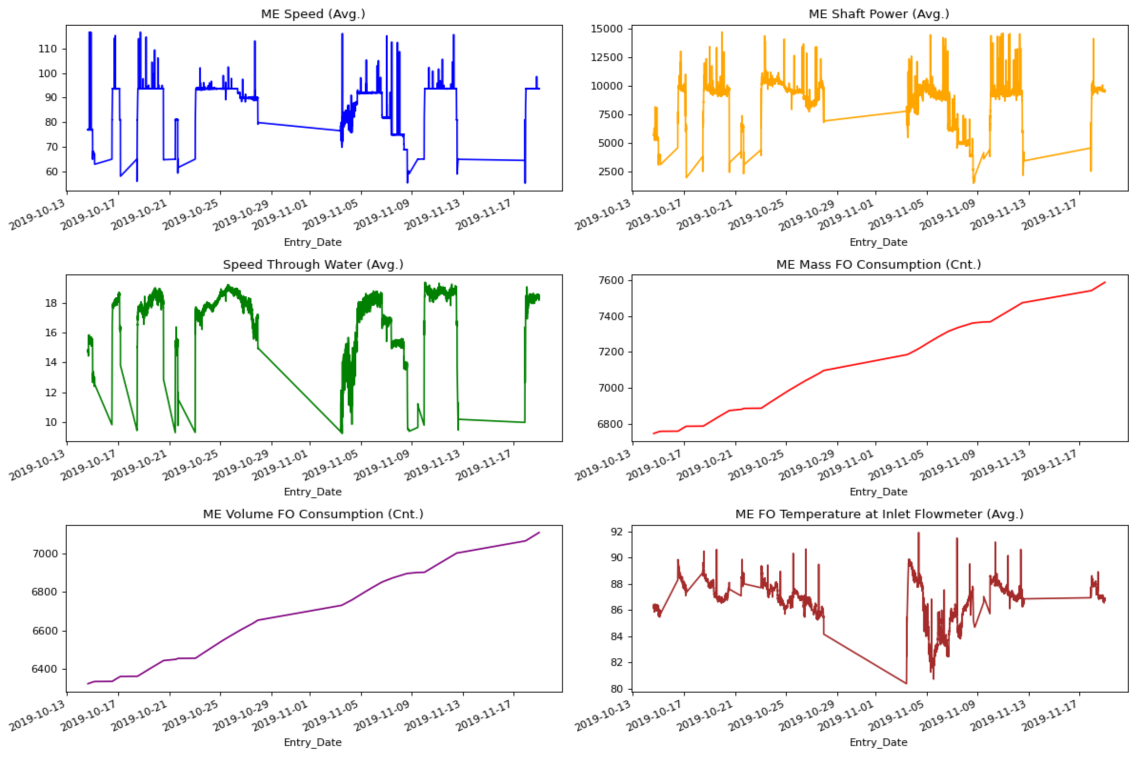


Figure 3.20: Processed testing dataset line plots.

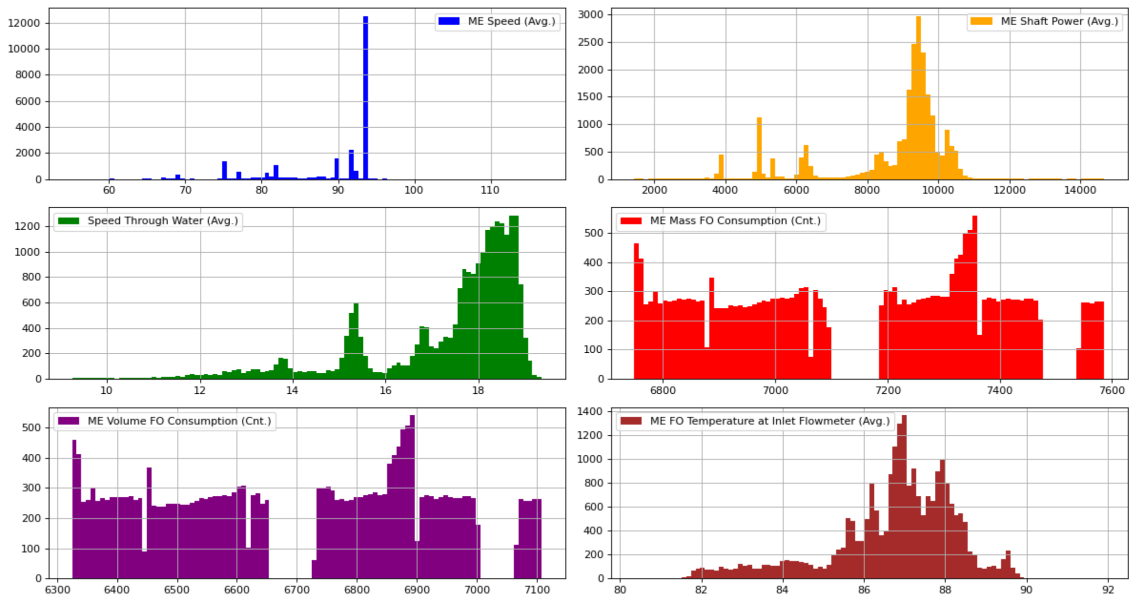


Figure 3.21: Processed testing dataset histograms.

The comparison scatter plots are also updated, as shown in Fig.**??**, Fig.**??**:

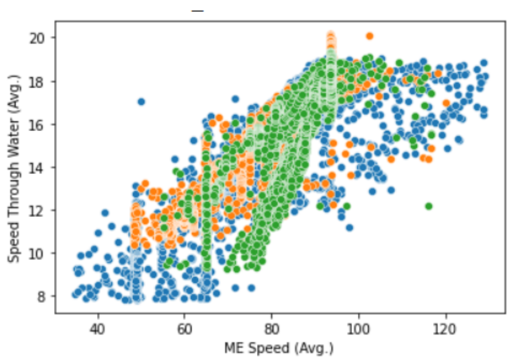


Figure 3.22: Processed ME Speed vs. Speed through Water scatter plot.

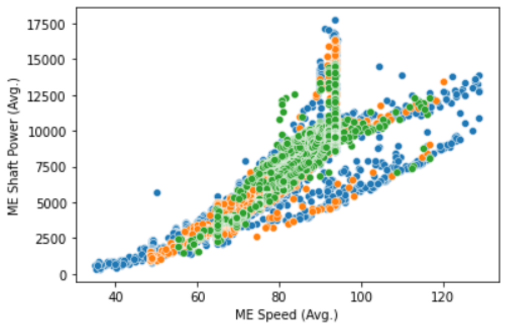


Figure 3.23: Processed ME Speed vs. ME Shaft Power scatter plot.

The last step of preprocessing focuses on managing the noise, i.e. the fine-grained variation between time steps. Moving average smoothing is a widespread type of smoothing used in time series forecasting and can significantly assist the training process by better exposing the signal of the underlying process. Calculating a moving average involves creating a new series where the values are determined using the average of raw observations in the original time series. The key decision when applying this type of smoothing is the selection of a suitable window width; the number of observations to be used when calculating the average value. More analytically, the window should be long enough to ensure the removal of noise, but short enough so that the main characteristics of the signal are not tampered. As for the moving part, it refers to the window sliding along the time series to calculate the consecutive local average values. Selecting a window size of 5 values (= 5 minutes), results to the series plotted in Fig.3.24.

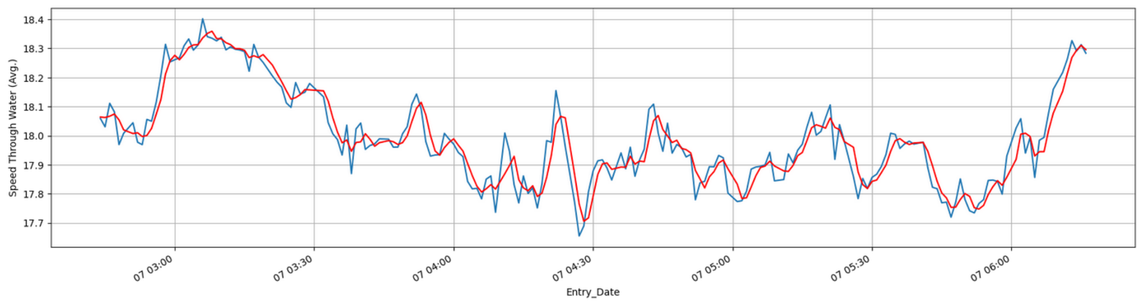


Figure 3.24: Smoothed training dataset subset.

The datasets are now ready to be normalized using the mean and standard deviation of the training dataset, so as to avoid data leakage, while the parameters ME Speed, ME Shaft Power, Speed Through Water, ME FO Temperature at Inlet Flowmeter and ME Mass FO Consumption differences are chosen as each networks' input features.

Chapter 4

Neural Network Designs

The problem of long term (more than one time steps into the future) ship speed prediction falls under the category of multi-input univariate regression time series forecasting. In its essence, this means forecasting a range of future values of a single feature, like ship speed, using multiple timely ordered historical data of various features, including the one being predicted.

As the process of tuning the hyperparameters does not provide meaningful information on the problem nor is of significant interest (one usually adds more layers and more units per layer till the accuracy starts stalling), it is not considered the main focus of this thesis. Nevertheless, the network topologies under study were selected after a methodical, although non-automated, model prediction accuracy optimization procedure. Each network architecture was tested on a range of input - output configurations; the prediction horizon varies between 10 - 20 - 30 timesteps, while the input window width is set to be 20 - 30 - 40 timesteps. For every configuration, 3 different snapshots of the training dataset are selected and plotted for visualization purpsoses, as shown in Fig.4.1.

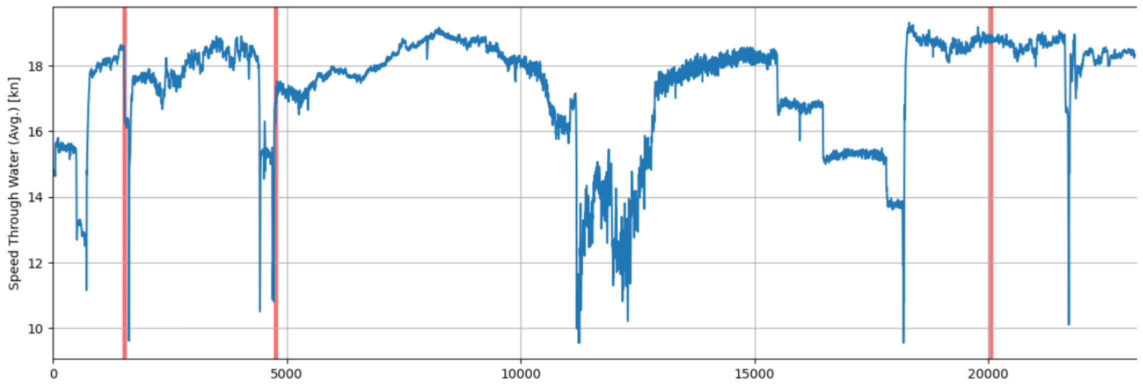


Figure 4.1: Selected snapshots' position in complete testing dataset.

Thus, the following results refer to *batch size* = 32, a number of epochs specified using early stopping callbacks, that interrupt training when no progress is measured on the validation set for 2 consecutive epochs, and:

- Linear: 1 hidden dense layer of *output steps* number of units
- MLP: 3 hidden dense layers of *output steps* number of units each
- Convolution: 1 hidden convolution layer of 256 units and 1 dense layer of *output steps* number of units

- Vanilla LSTM: 1 hidden LSTM layer of 10 units and 1 dense layer of *output steps* number of units
- Stacked LSTM: 2 hidden LSTM layer of 32 units each, 2 Dropout layers and 1 dense layer of *output steps* number of units

As already mentioned, when testing the predictive power of a neural network it is important to have a Common Sense Baseline, i.e. a point of comparison with the more complicated models, to make sure that the model does offer useful information. In this case, a reasonable baseline would be maintaining the last ship speed input and predicting no change. Truly, in ocean going conditions the vessel's speed is not expected to vary significantly, but it is obvious that this assumption will not work well the further in the future our predictions are. Please note that no model is involved in this process.

4.1 20 timesteps input

For the case of predicting 10 timesteps into the future, the input and label sets, together with the common sense baseline are shaped as follows:

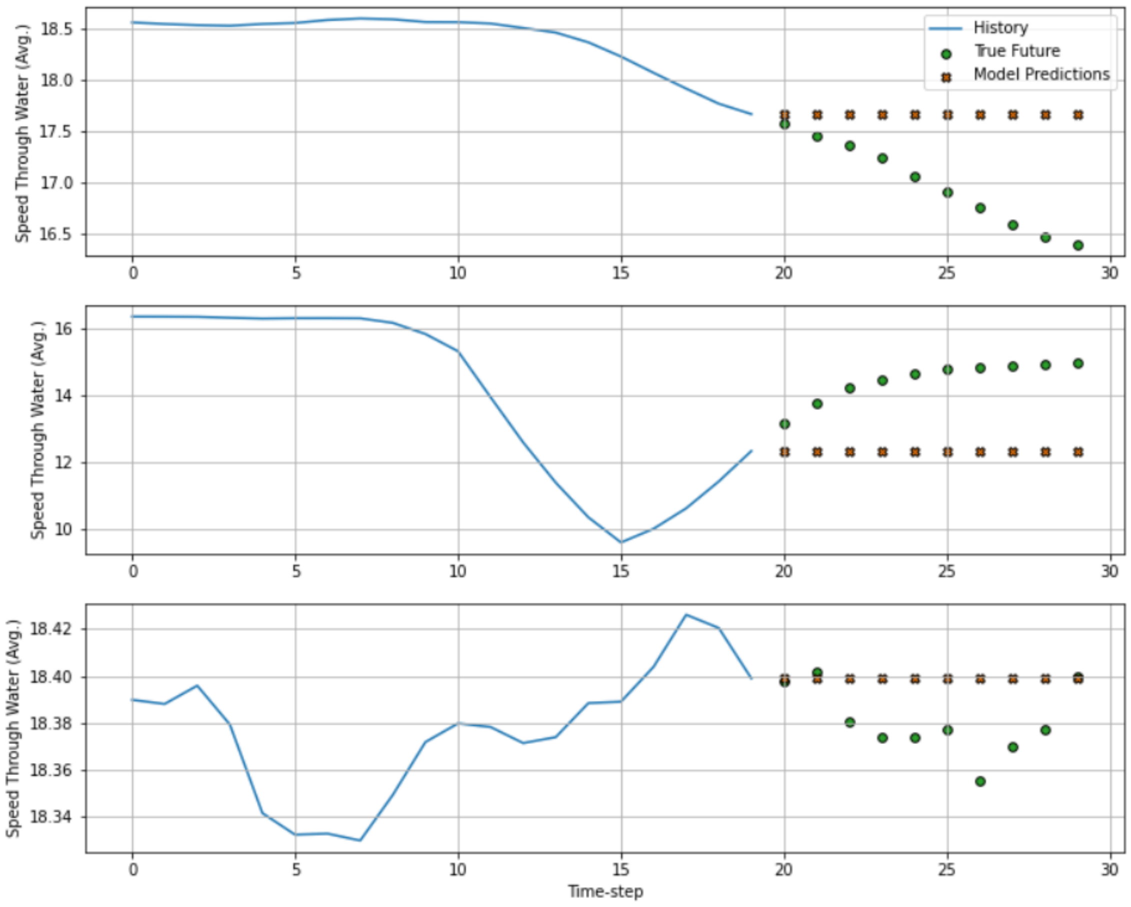


Figure 4.2: Indicative plots of the common sense baseline (20/10 configuration).

Using the simple linear system:

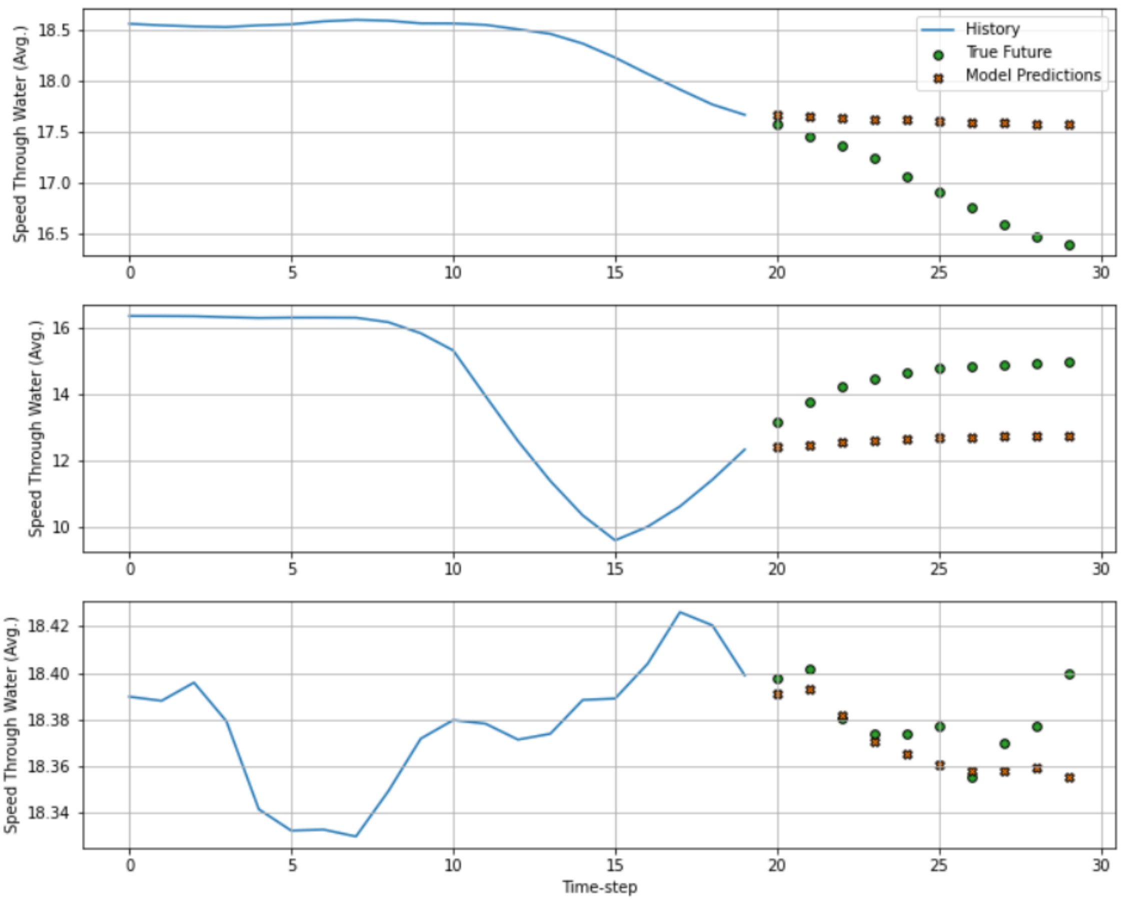


Figure 4.3: Indicative plots of the predictions using the simple linear network (20/10 configuration).

Using the MLP networks:

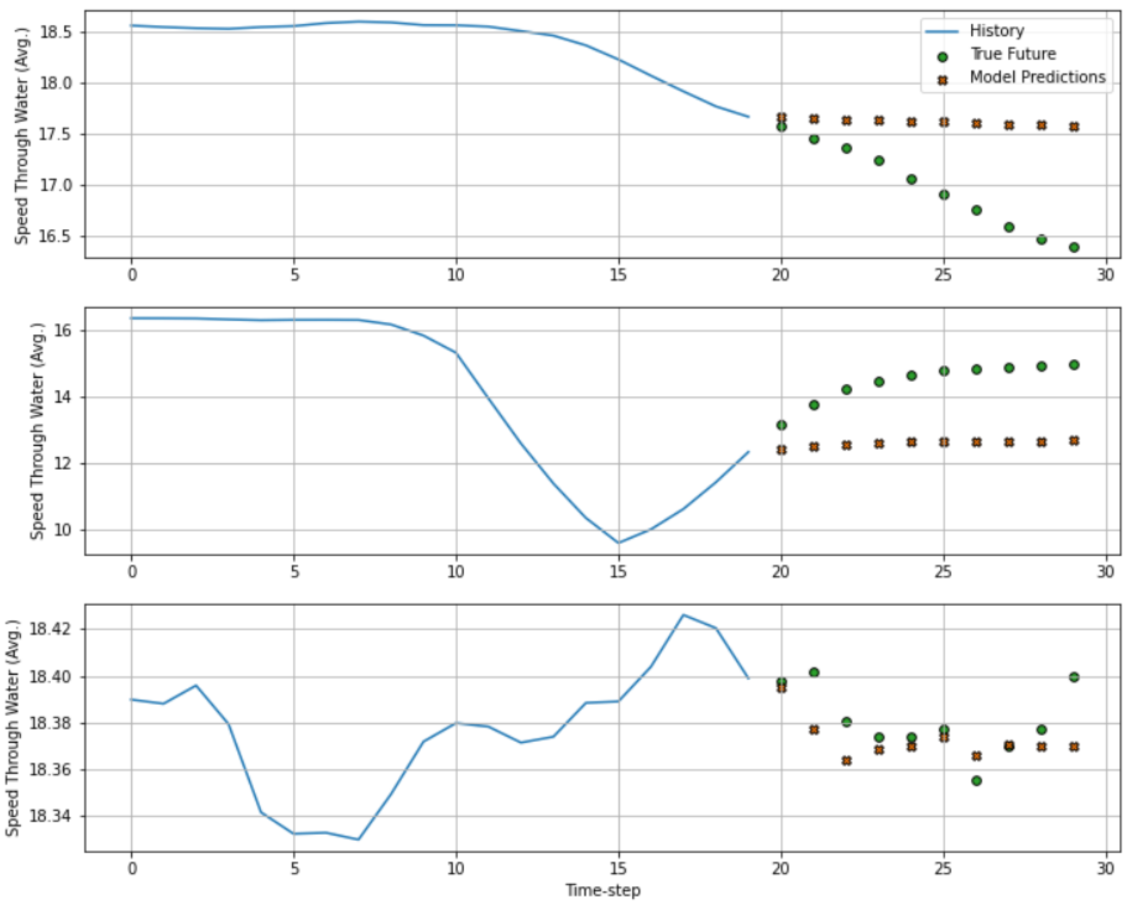


Figure 4.4: Indicative plots of the predictions using the MLP network (20/10 configuration).

Using the Convolution network:

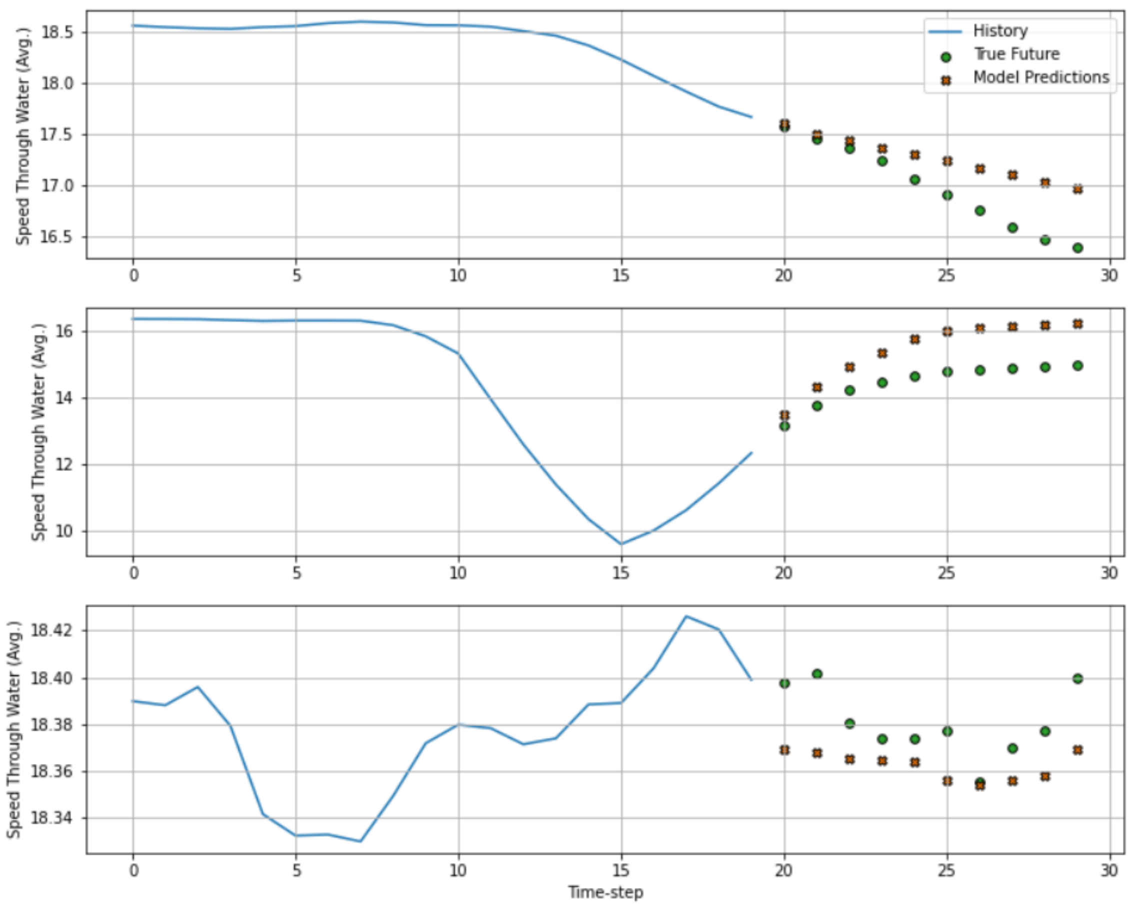


Figure 4.5: Indicative plots of the predictions using the Convolution network (20/10 configuration).

Using the Vanilla LSTM network:

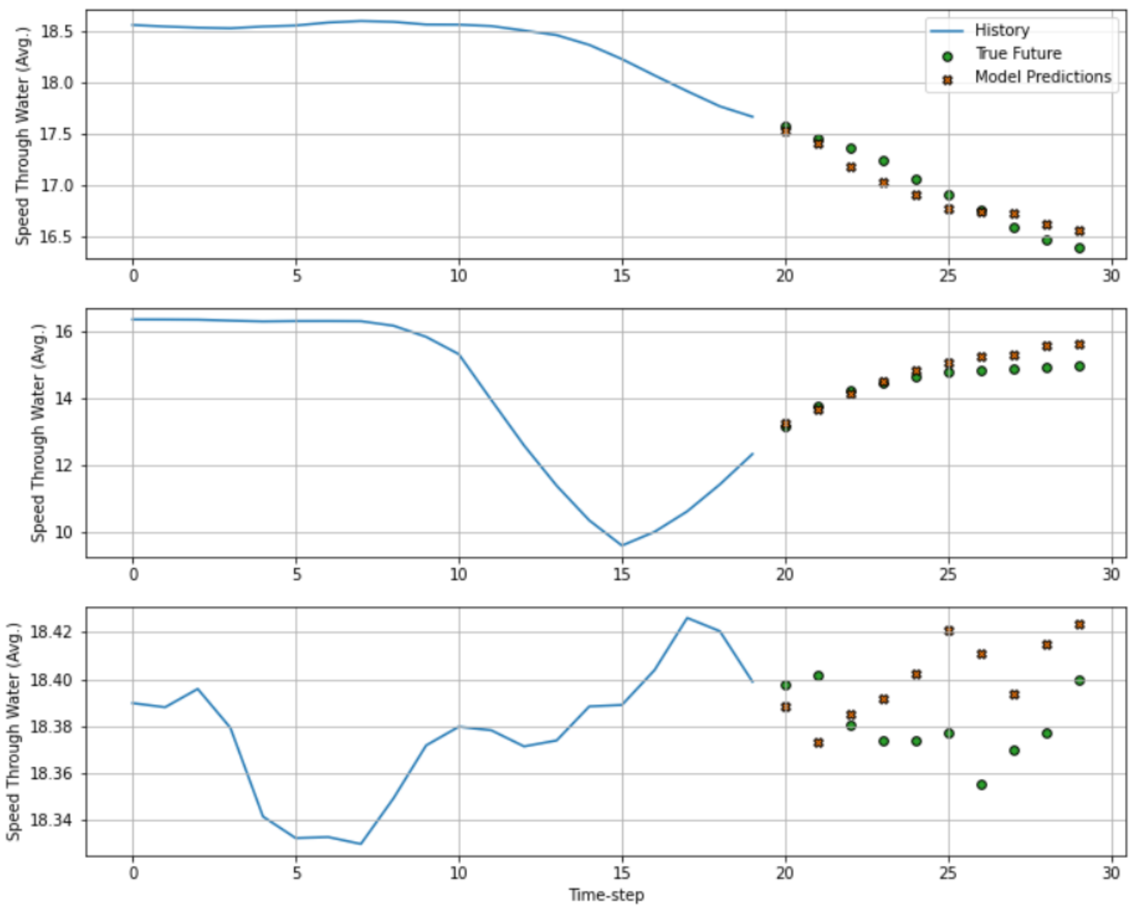


Figure 4.6: Indicative plots of the predictions using the Vanilla LSTM network (20/10 configuration).

Using the Stacked LSTM network:

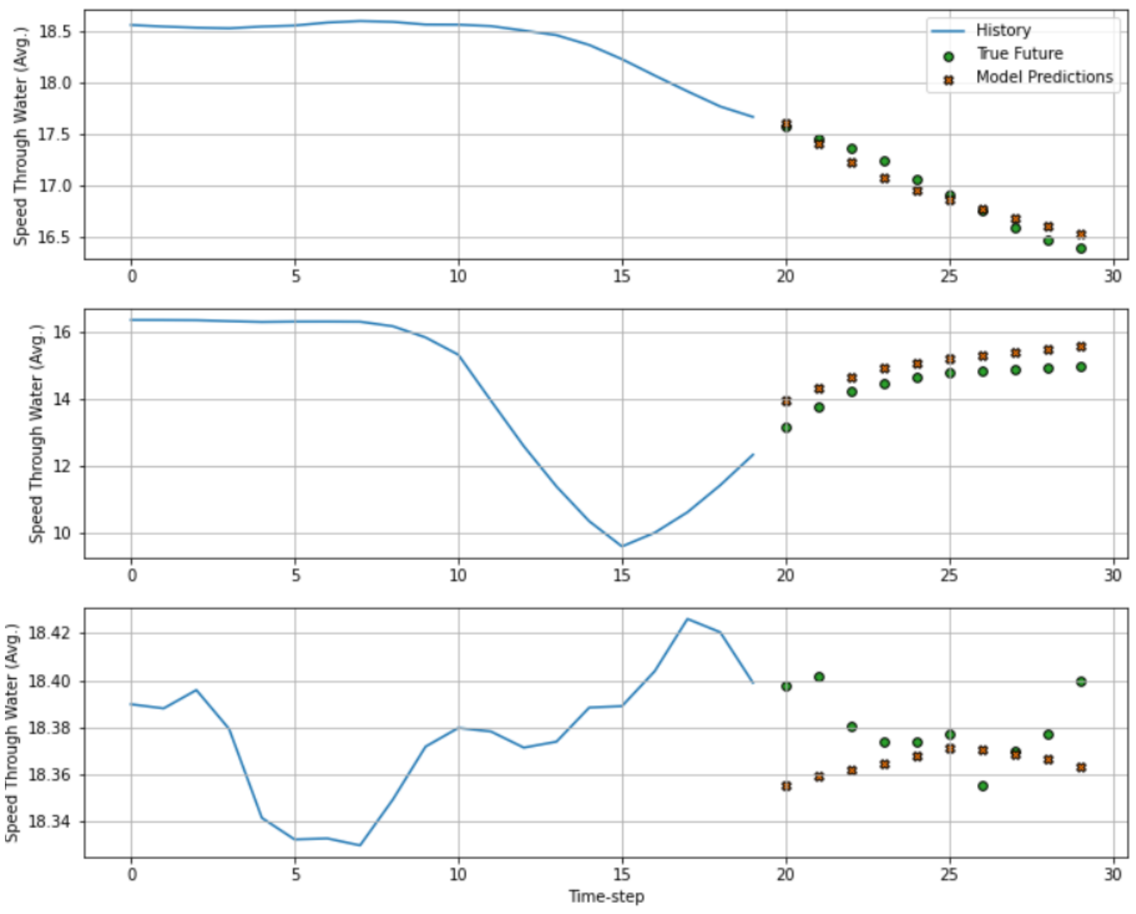


Figure 4.7: Indicative plots of the predictions using the Stacked LSTM network (20/10 configuration).

In comparison, the prediction mean absolute error is estimated as:

Table 4.1: Mean Absolute Percentage Error & Mean Absolute Error of 20/10 configuration

	MAPE %	MAE [kn]
Baseline	104.78	1.222
Linear	17.6	0.123
MLP	17.4	0.121
Convolution	17.6	0.158
LSTM	16.1	0.141
Stacked LSTM	15.9	0.124

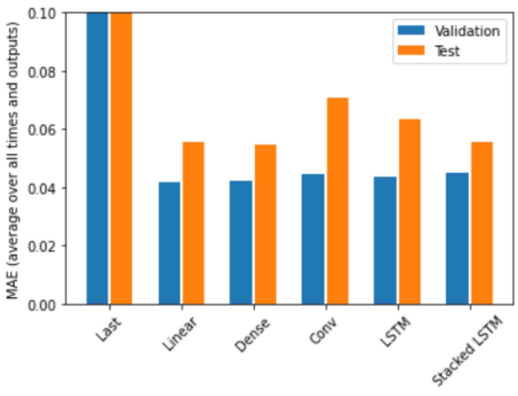


Figure 4.8: Performance comparison (20/10 configuration).

All networks display significant improvement of prediction accuracy, but the small diffrence between validation and test performance of Stacked LSTM indicates greater generalization power.

For the case of predicting 20 timesteps into the future, the input and label sets, together with the common sense baseline are shaped as follows:

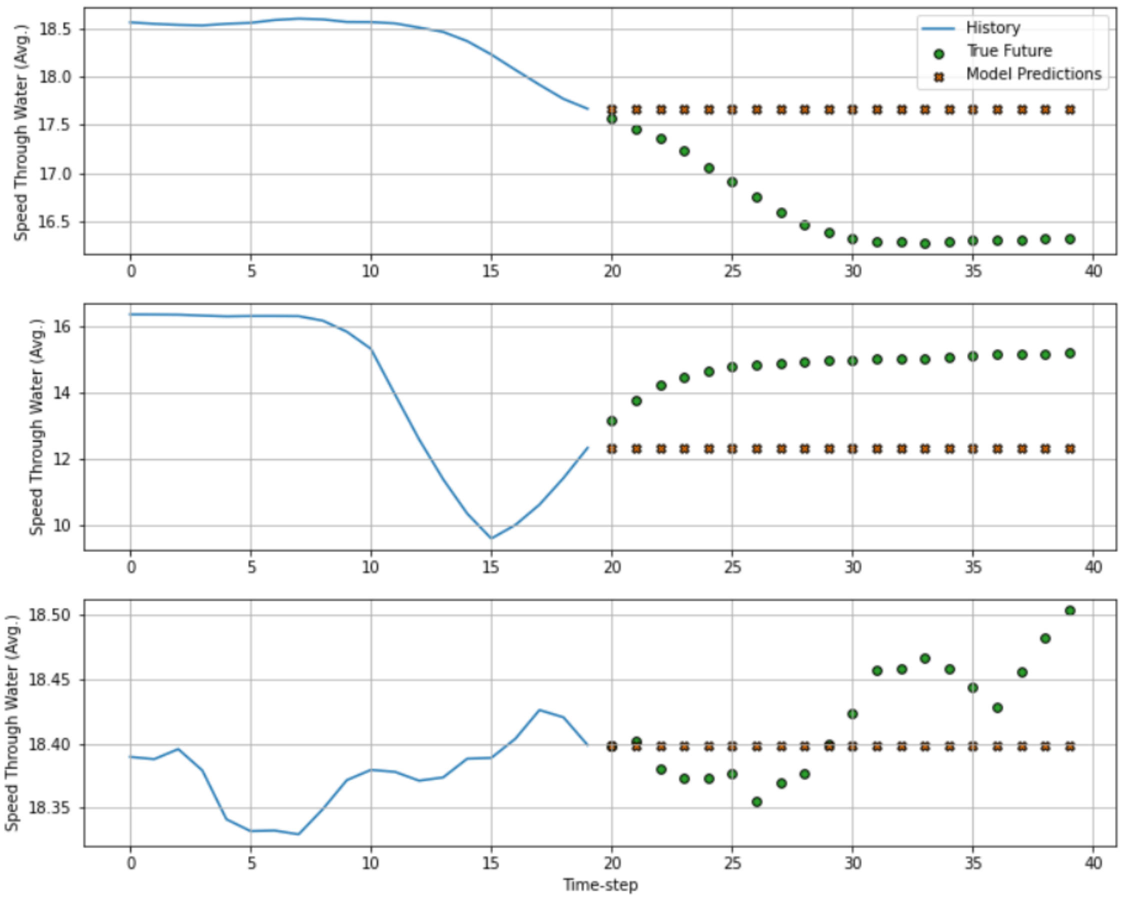


Figure 4.9: Indicative plots of the common sense baseline (20/20 configuration).

Using the simple linear system:

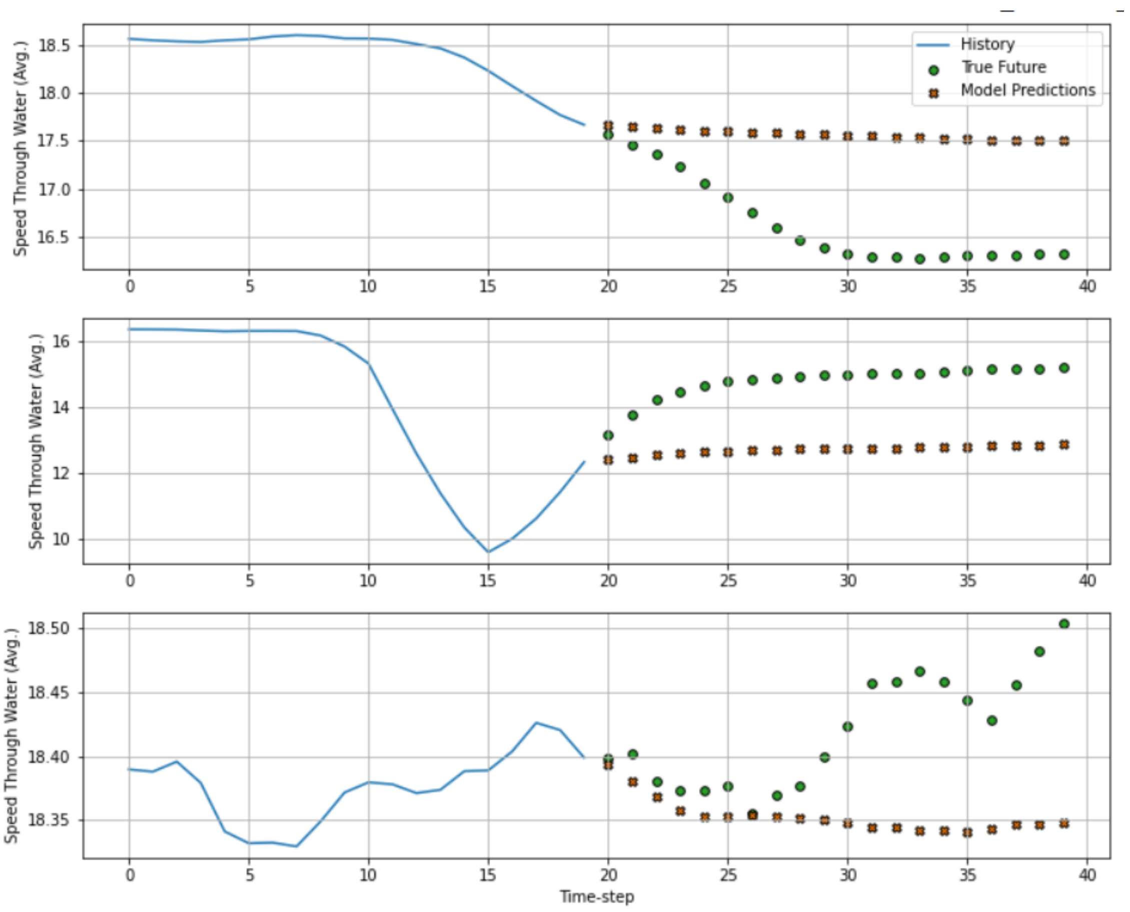


Figure 4.10: Indicative plots of the predictions using the simple linear network (20/20 configuration).

Using the MLP networks:

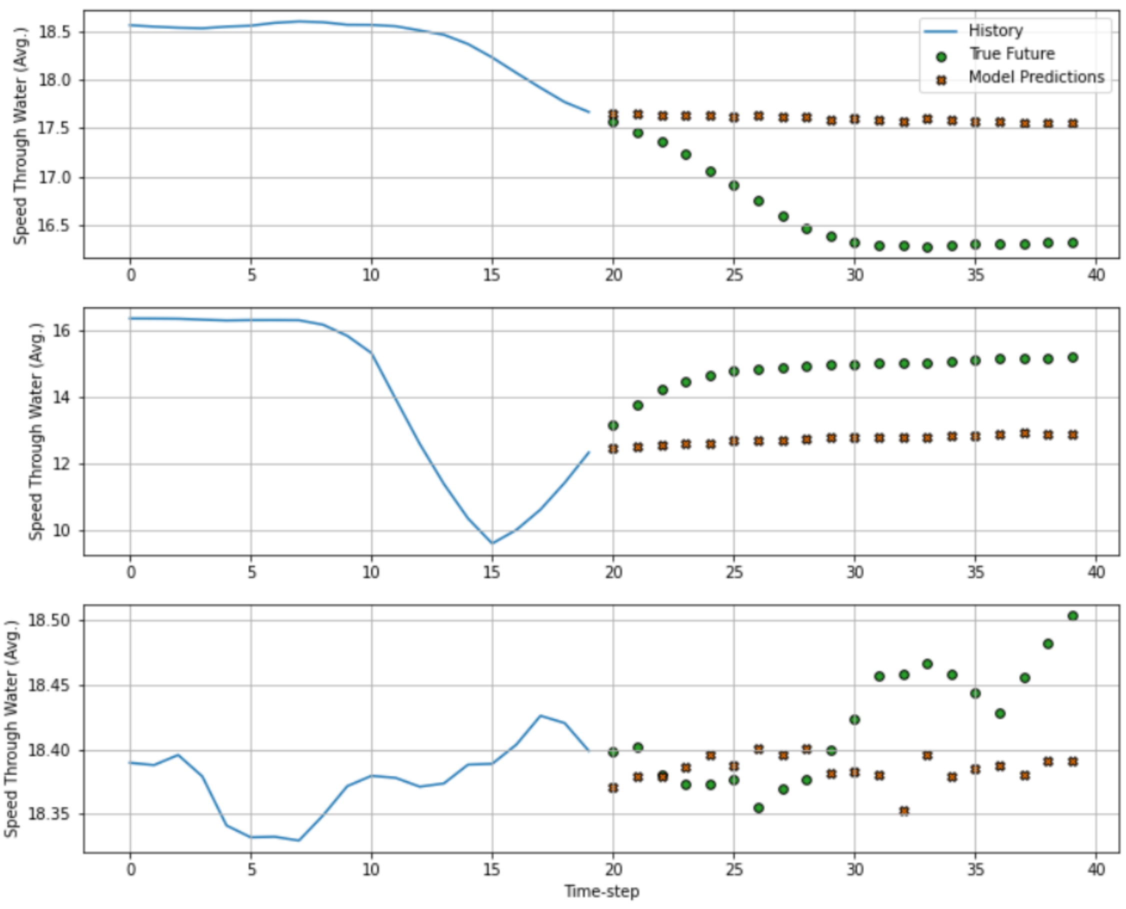


Figure 4.11: Indicative plots of the predictions using the MLP network (20/20 configuration).

Using the Convolution network:

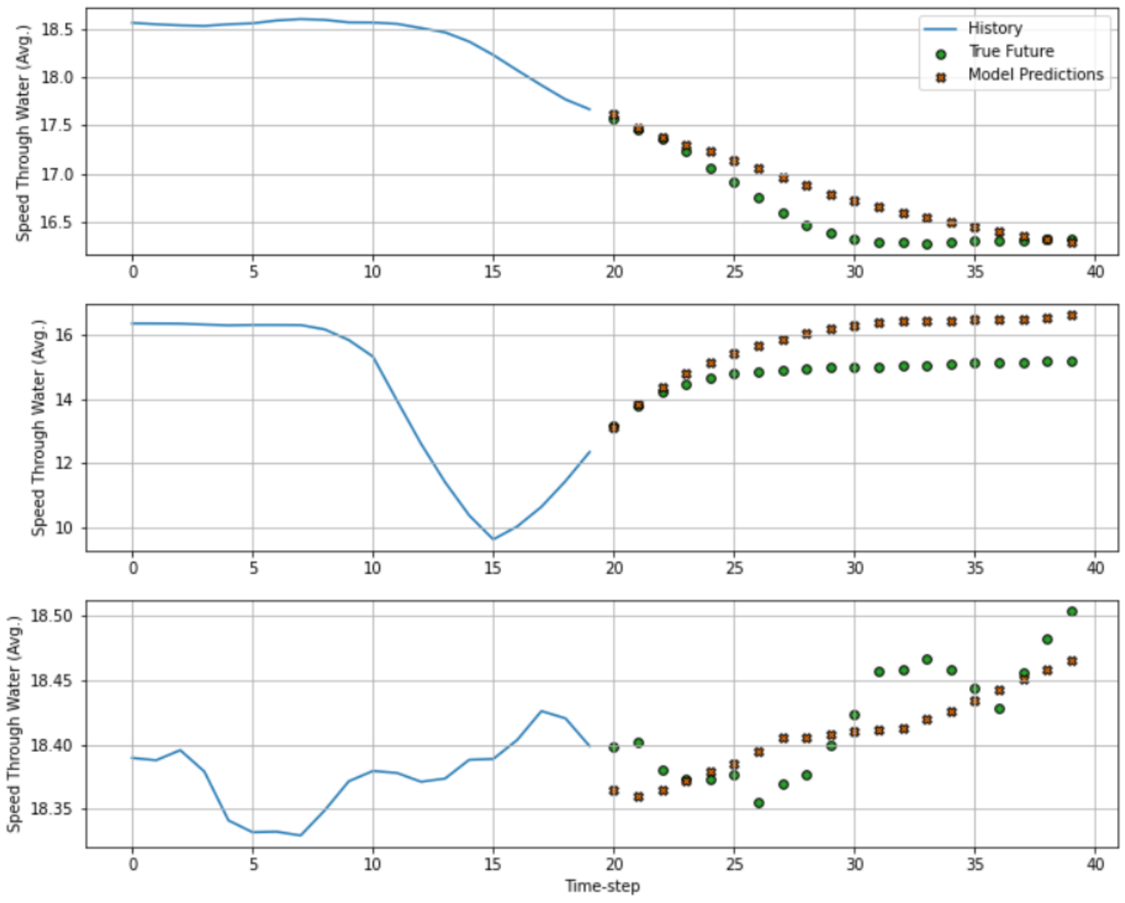


Figure 4.12: Indicative plots of the predictions using the Convolution network (20/20 configuration).

Using the Vanilla LSTM network:

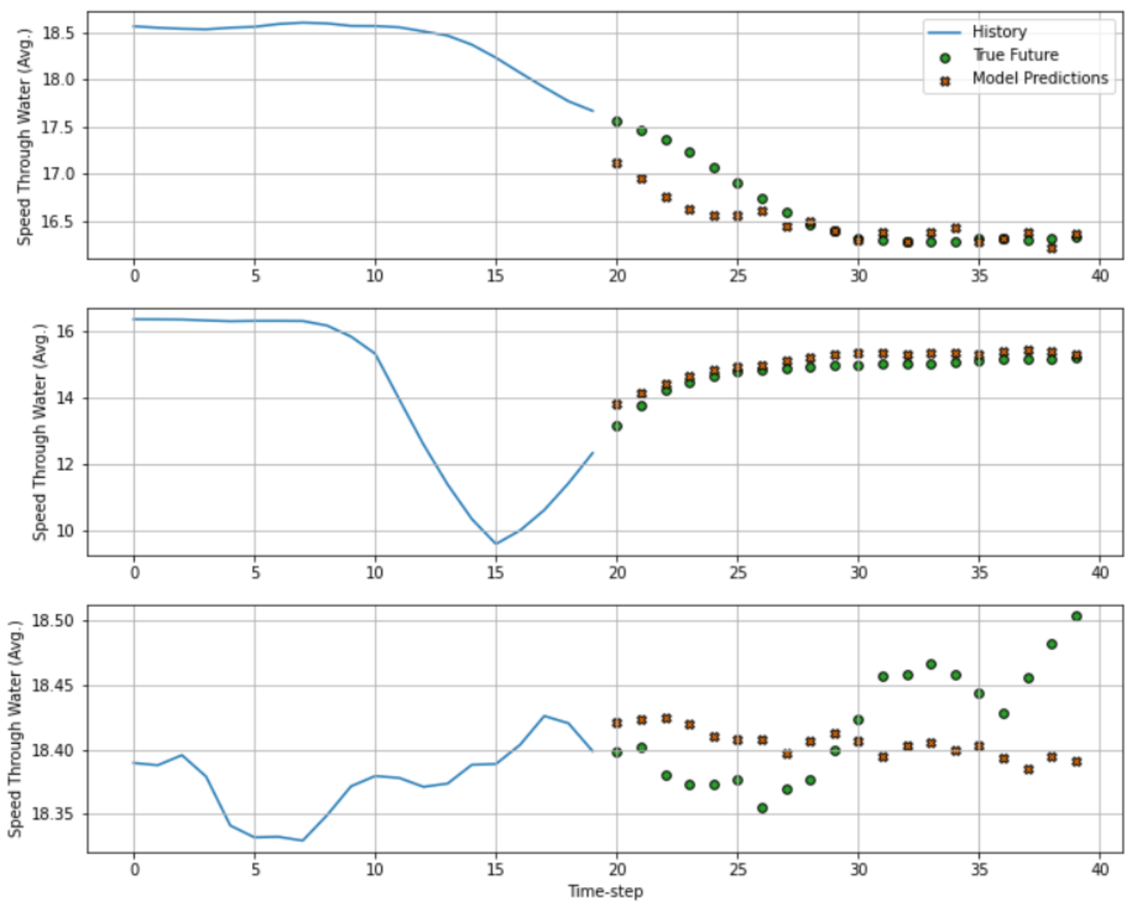


Figure 4.13: Indicative plots of the predictions using the Vanilla LSTM network (20/20 configuration).

Using the Stacked LSTM network:

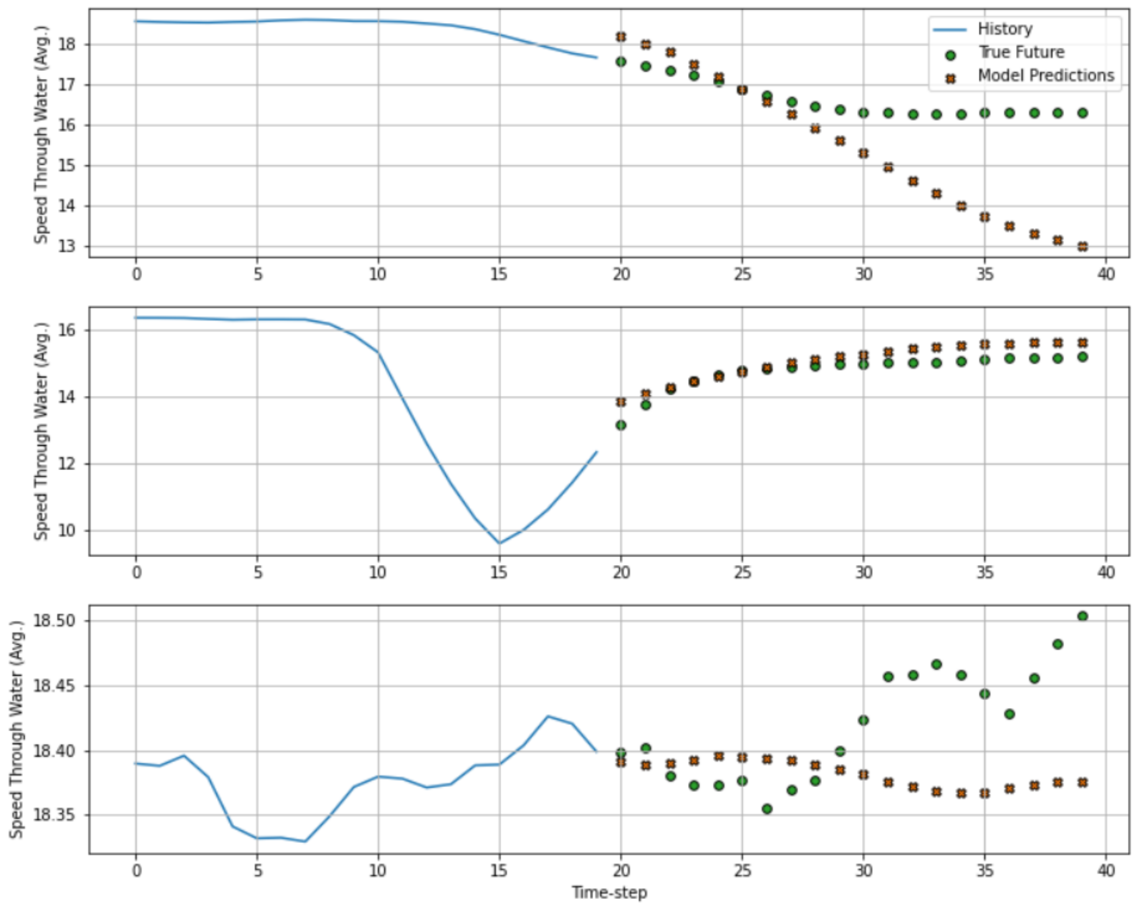


Figure 4.14: Indicative plots of the predictions using the Stacked LSTM network (20/20 configuration).

In comparison, the prediction mean absolute error is estimated as:

Table 4.2: Mean Absolute Percentage Error & Mean Absolute Error of 20/20 configuration

	MAPE %	MAE [kn]
Baseline	106.65	1.235
Linear	24.8	0.171
MLP	25.4	0.167
Convolution	22.7	0.229
LSTM	25.8	0.195
Stacked LSTM	25.1	0.182

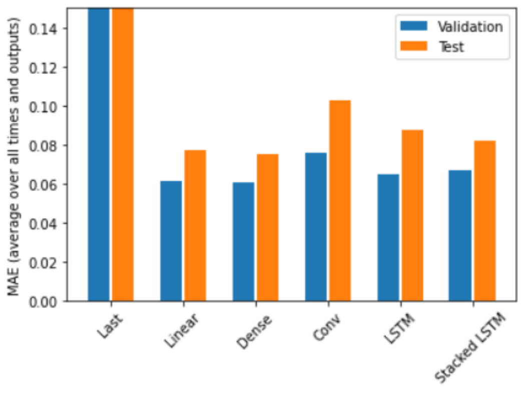


Figure 4.15: Performance comparison (20/20 configuration).

Lower accuracy in comparison with the 20/10 configuration is expected, as requesting more feature values means more error prone calculations. The MLP, Linear and Stacked topologies display similar behaviour.

For the case of predicting 30 timesteps into the future, the input and label sets, together with the common sense baseline are shaped as follows:

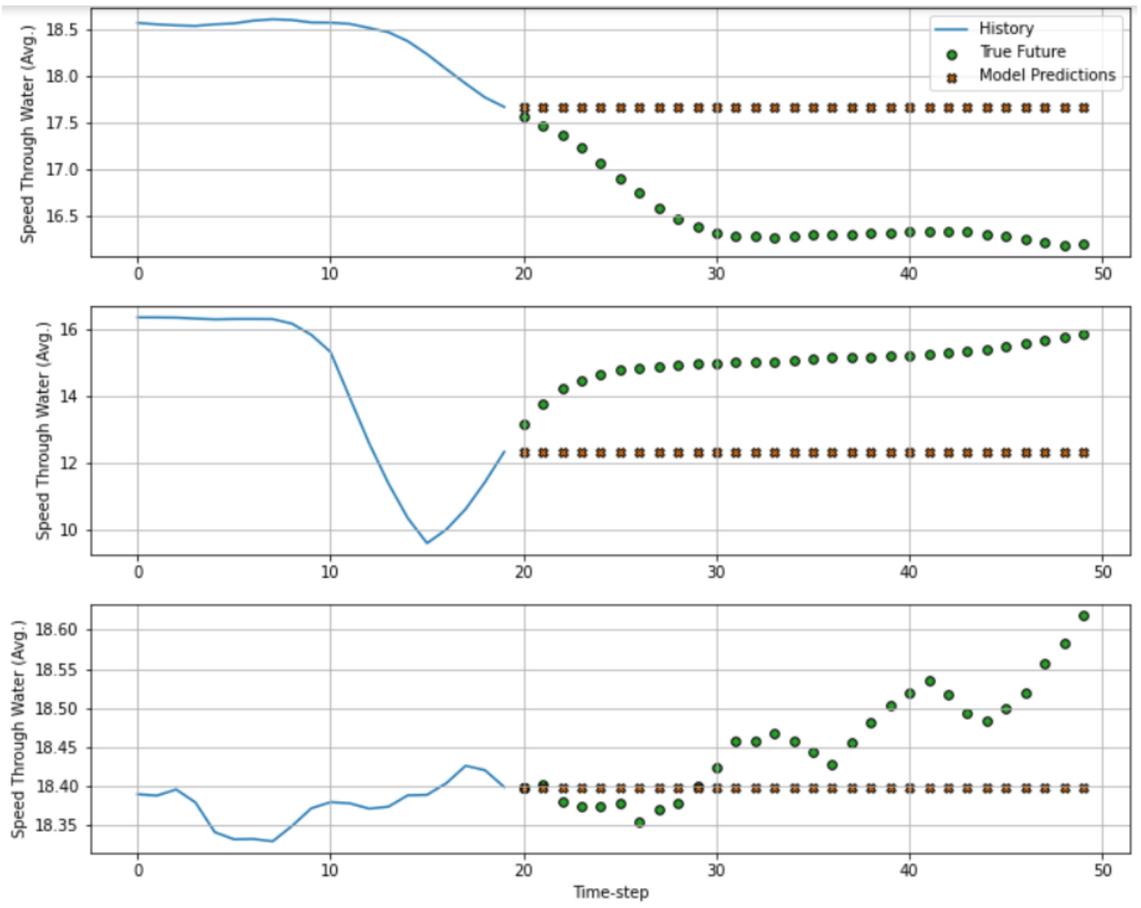


Figure 4.16: Indicative plots of the common sense baseline (20/30 configuration).

Using the simple linear system:

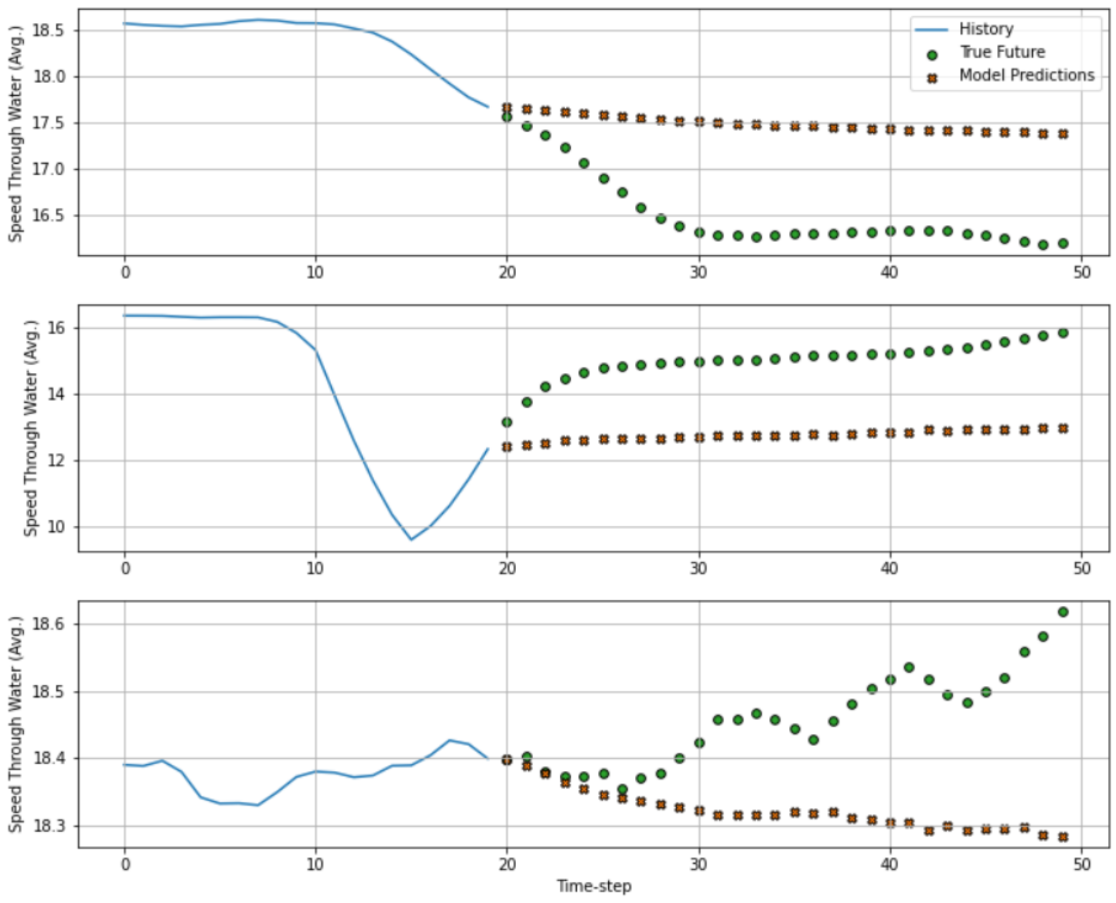


Figure 4.17: Indicative plots of the predictions using the simple linear network (20/30 configuration).

Using the MLP networks:

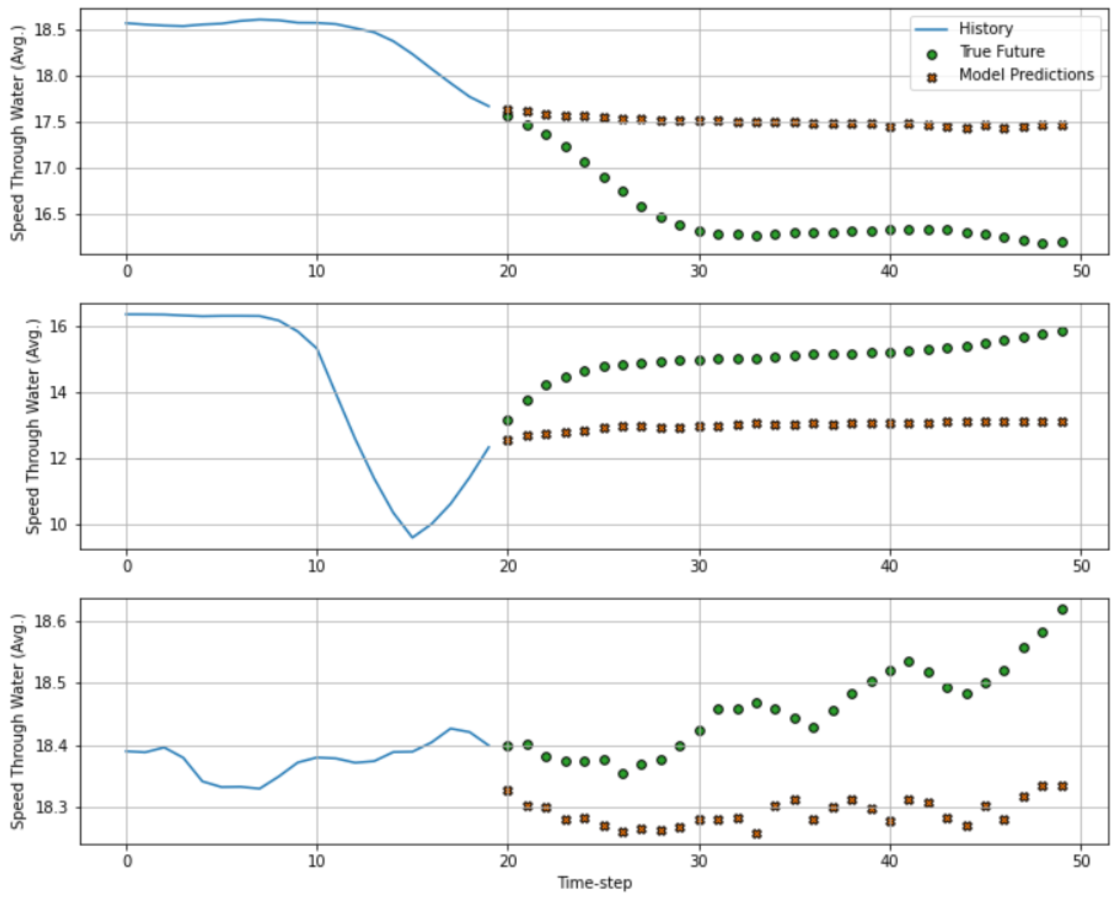


Figure 4.18: Indicative plots of the predictions using the MLP network (20/30 configuration).

Using the Convolution network:

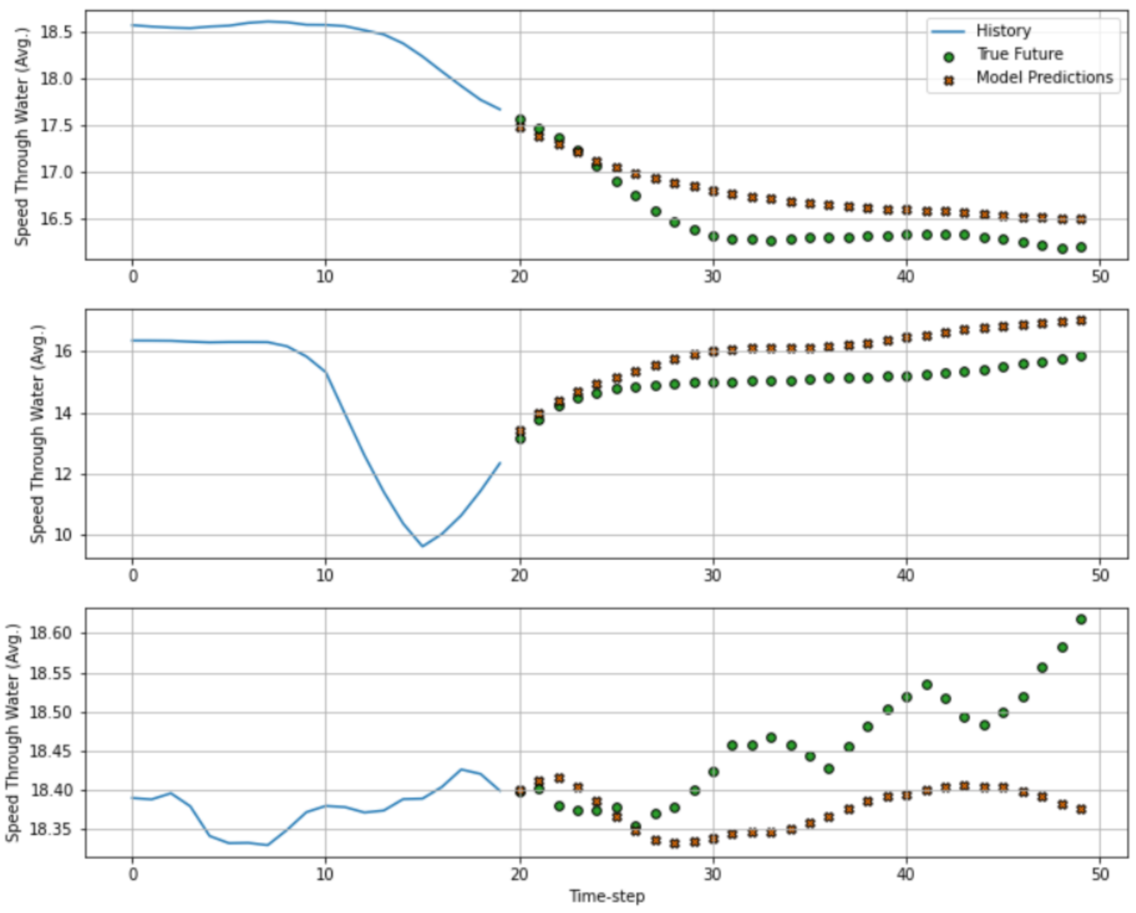


Figure 4.19: Indicative plots of the predictions using the Convolution network (20/30 configuration).

Using the Vanilla LSTM network:

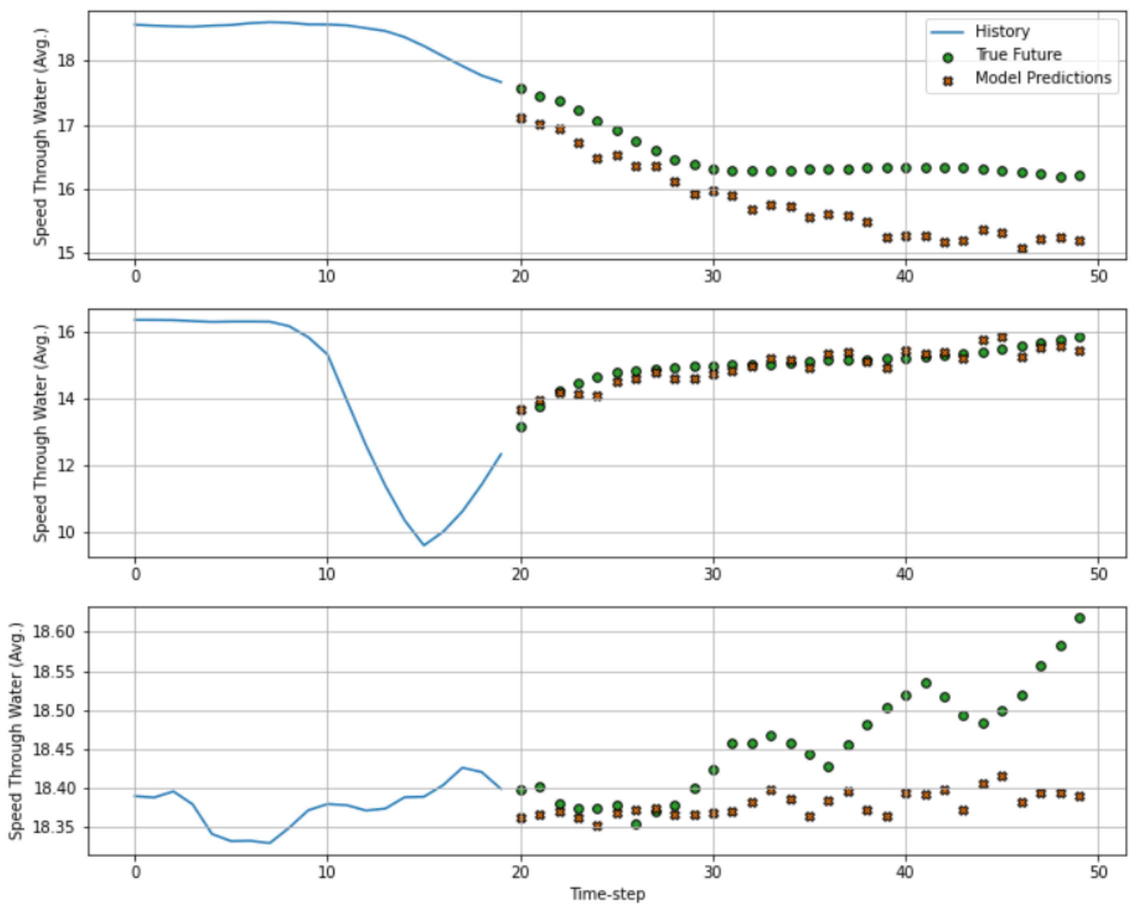


Figure 4.20: Indicative plots of the predictions using the Vanilla LSTM network (20/30 configuration).

Using the Stacked LSTM network:

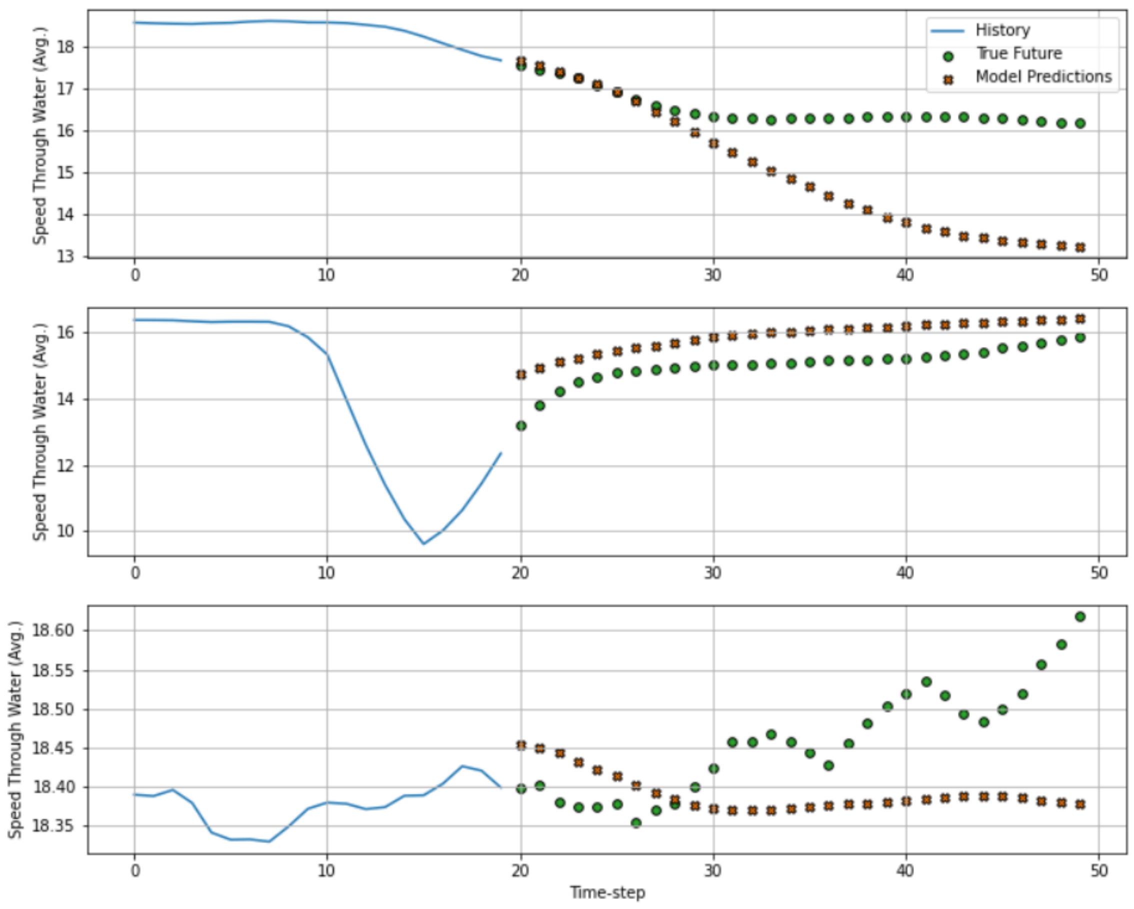


Figure 4.21: Indicative plots of the predictions using the Stacked LSTM network (20/30 configuration).

In comparison, the prediction mean absolute error is estimated as:

Table 4.3: Mean Absolute Percentage Error & Mean Absolute Error of 20/30 configuration

	MAPE %	MAE [kn]
Baseline	108.0	1.243
Linear	31.7	0.218
MLP	33.2	0.236
Convolution	28.6	0.273
LSTM	31.6	0.305
Stacked LSTM	28.9	0.219

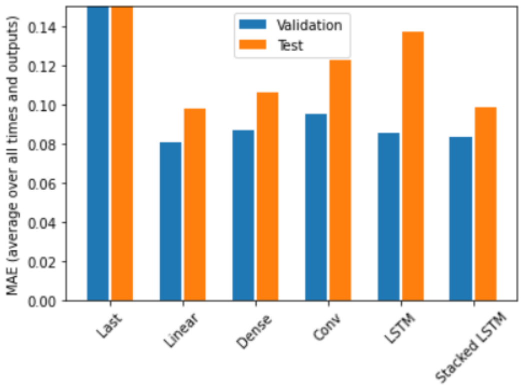


Figure 4.22: Performance comparison (20/30 configuration).

Again, lower accuracy in comparison with the 20/10 and 20/20 configurations is expected, while the greater generalization power of the Stacked LSTM is also apparent.

4.2 30 timesteps input

For the case of predicting 10 timesteps into the future, the input and label sets, together with the common sense baseline are shaped as follows:

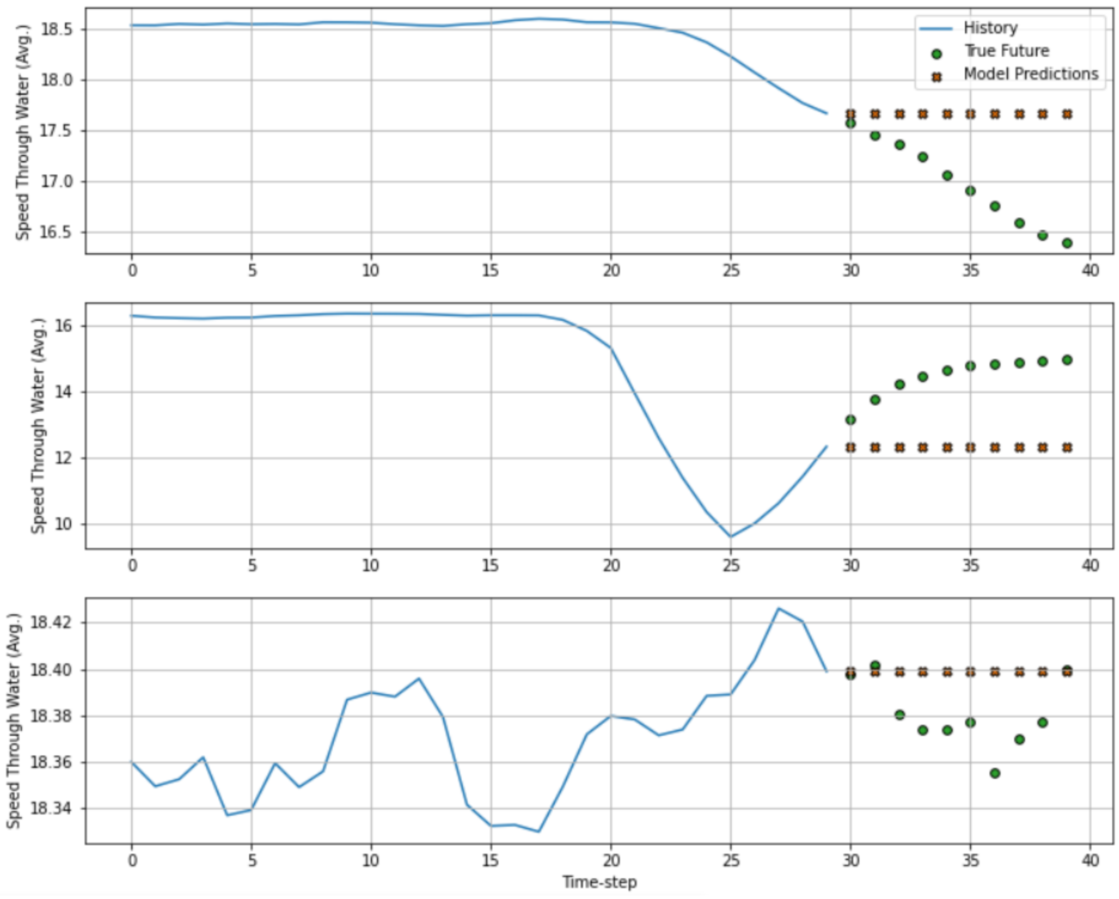


Figure 4.23: Indicative plots of the common sense baseline (30/10 configuration).

Using the simple linear system:

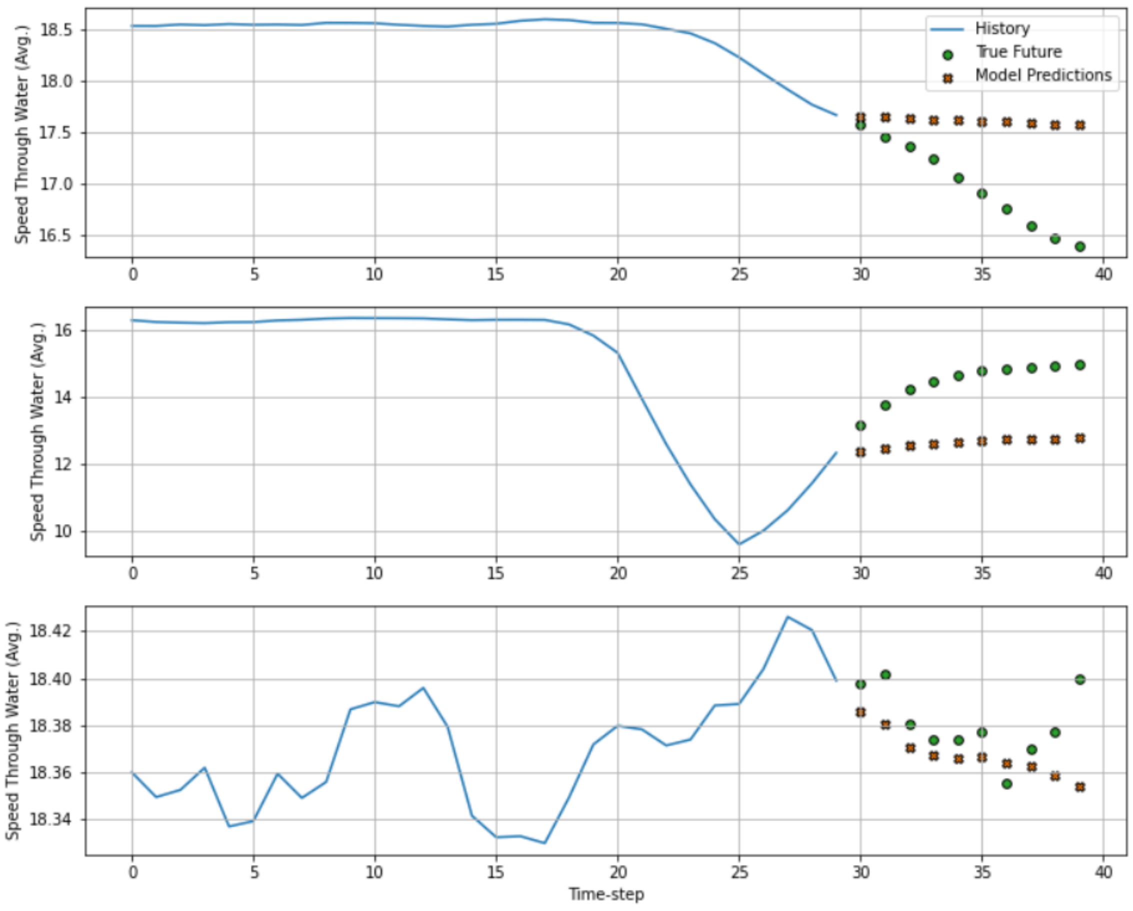


Figure 4.24: Indicative plots of the predictions using the simple linear network (30/10 configuration).

Using the MLP networks:

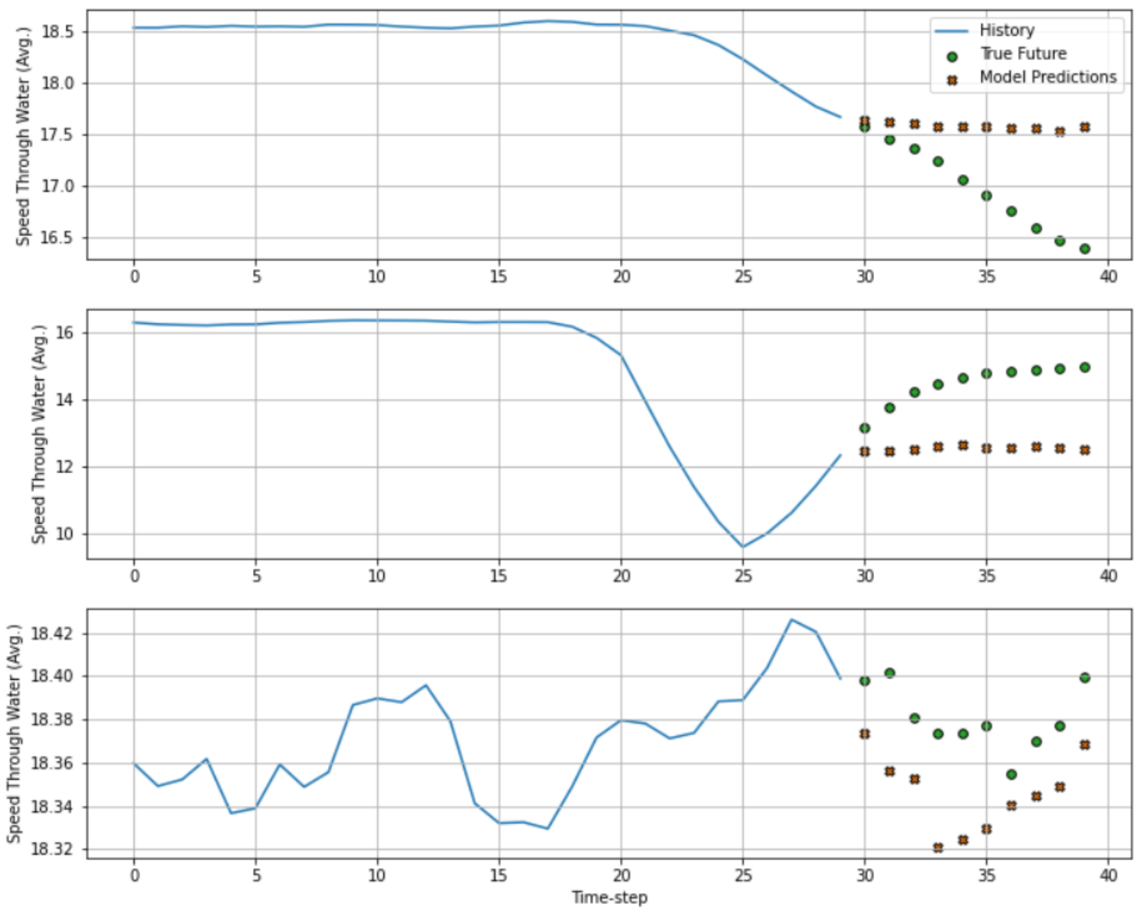


Figure 4.25: Indicative plots of the predictions using the MLP network (30/10 configuration).

Using the Convolution network:

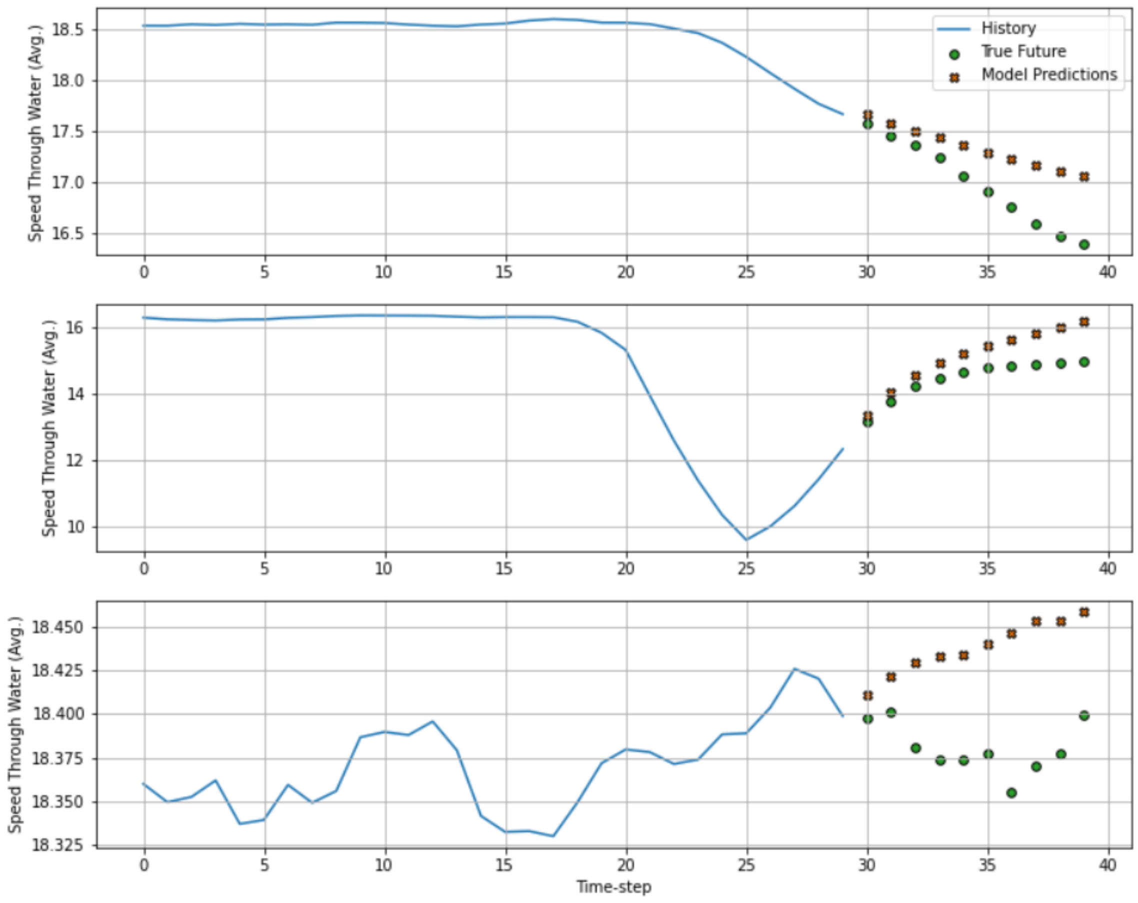


Figure 4.26: Indicative plots of the predictions using the Convolution network (30/10 configuration).

Using the Vanilla LSTM network:

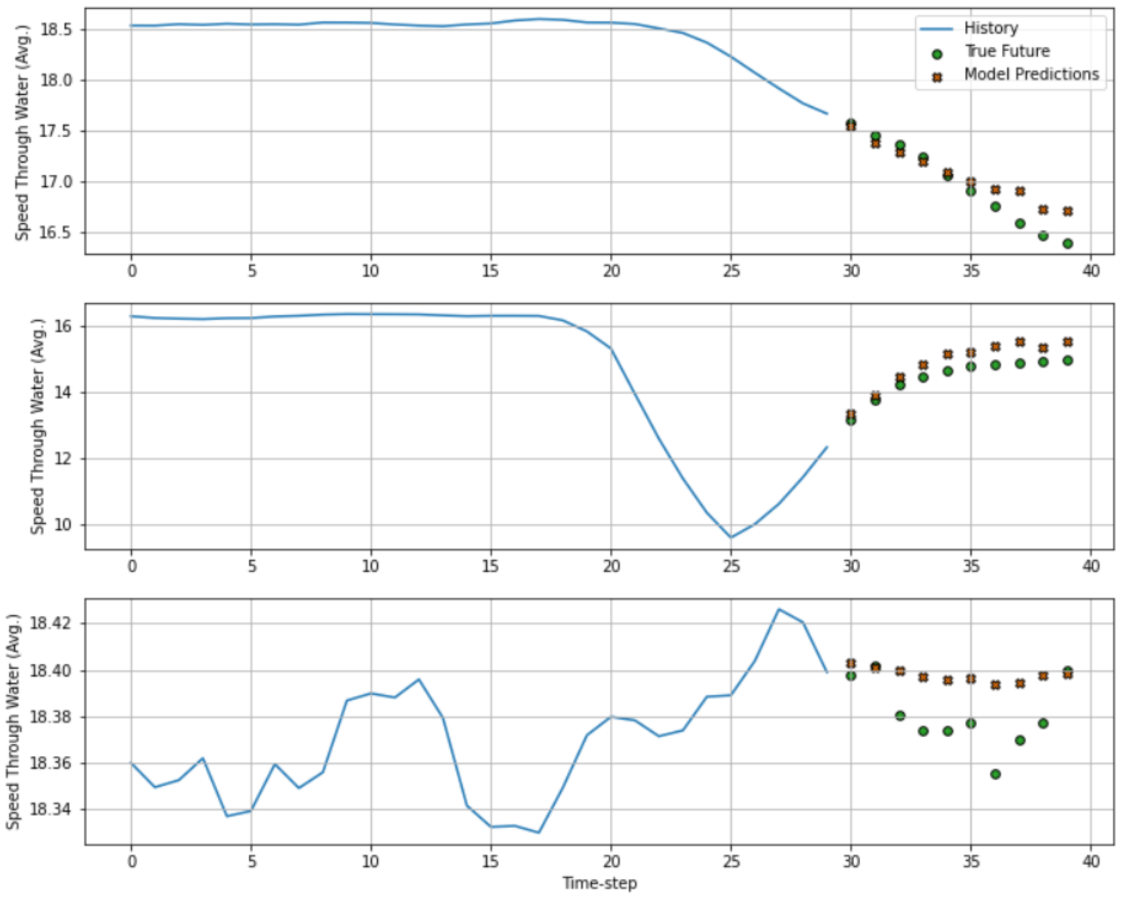


Figure 4.27: Indicative plots of the predictions using the Vanilla LSTM network (30/10 configuration).

Using the Stacked LSTM network:

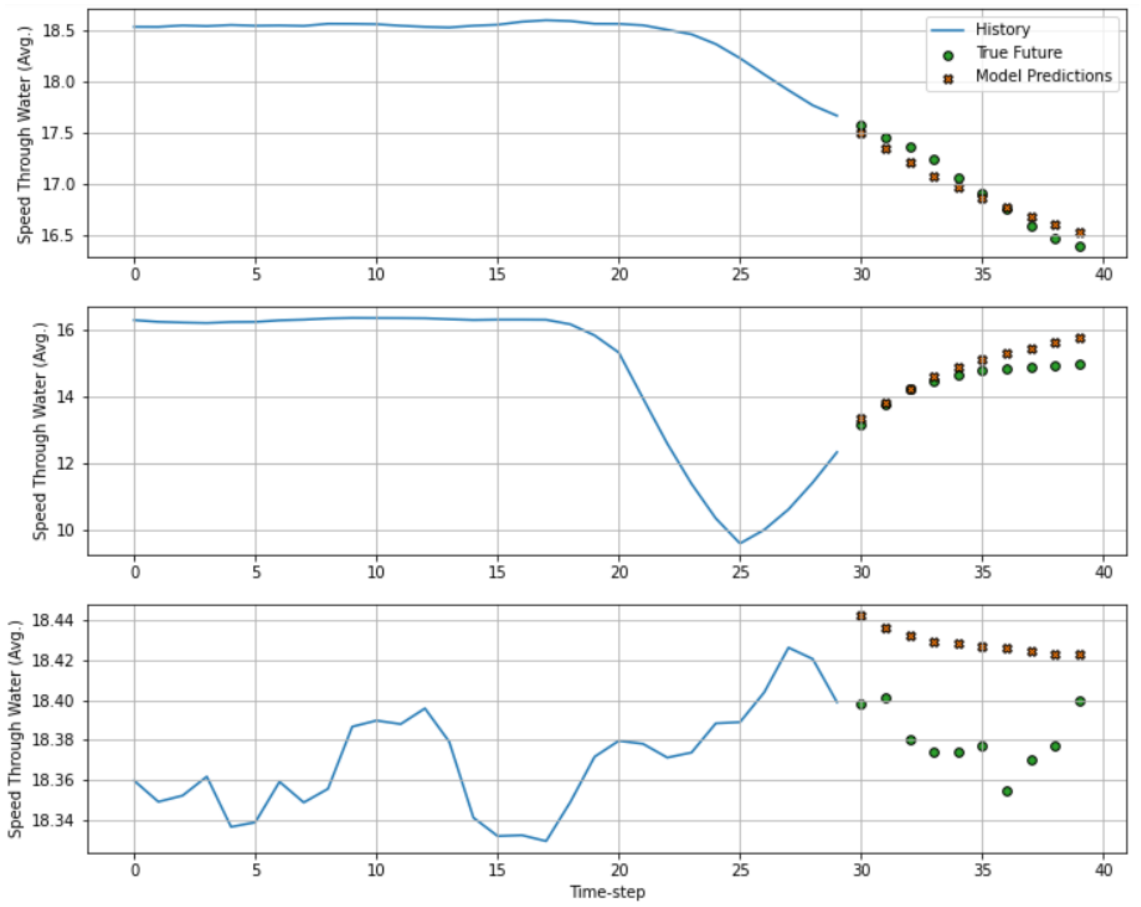


Figure 4.28: Indicative plots of the predictions using the Stacked LSTM network (30/10 configuration).

In comparison, the prediction mean absolute error is estimated as:

Table 4.4: Mean Absolute Percentage Error & Mean Absolute Error of 30/10 configuration

	MAPE %	MAE [kn]
Baseline	104.8	1.222
Linear	17.6	0.124
MLP	17.8	0.130
Convolution	17.7	0.157
LSTM	16.9	0.143
Stacked LSTM	16.2	0.131

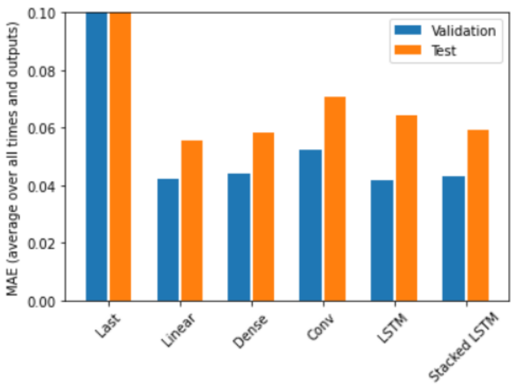


Figure 4.29: Performance comparison (30/10 configuration).

No significant change in accuracy for single input topologies, but slight improvement for CNN & LSTM observed, compared with the 20 timesteps input topologies.

For the case of predicting 20 timesteps into the future, the input and label sets, together with the common sense baseline are shaped as follows:

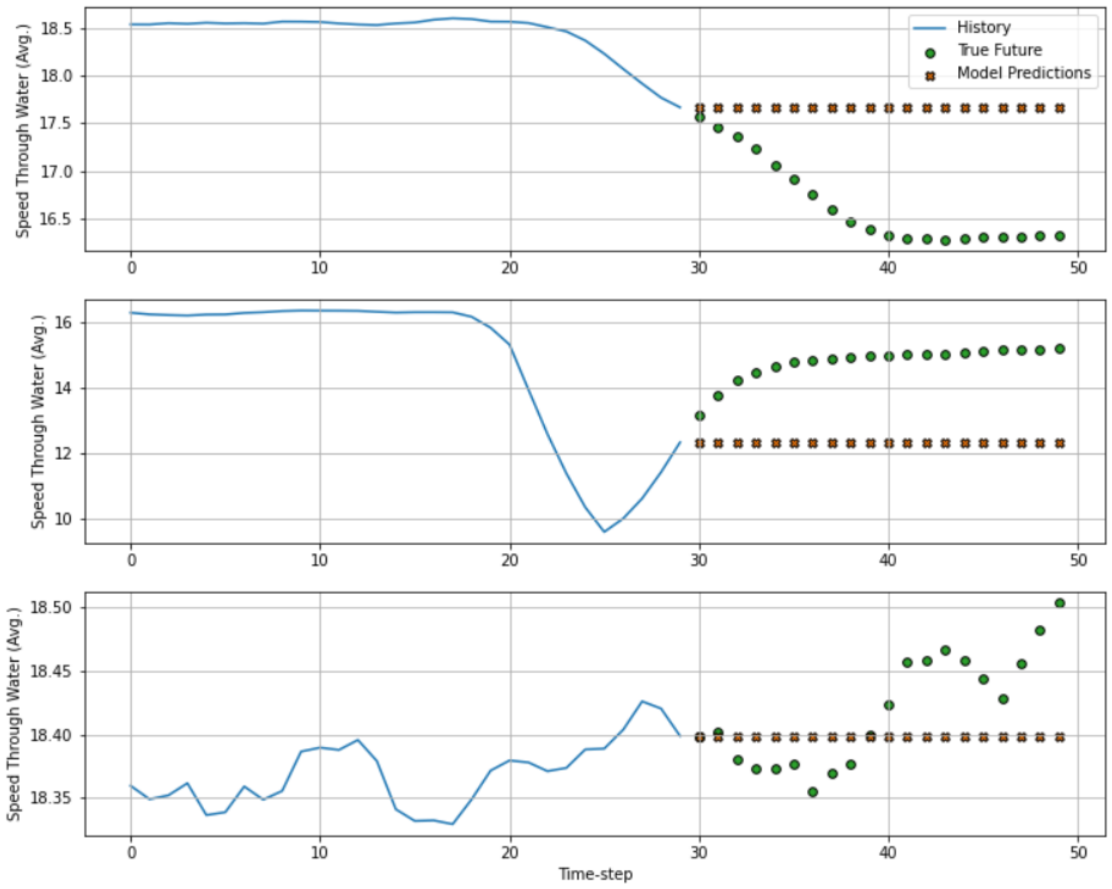


Figure 4.30: Indicative plots of the common sense baseline (30/20 configuration).

Using the simple linear system:

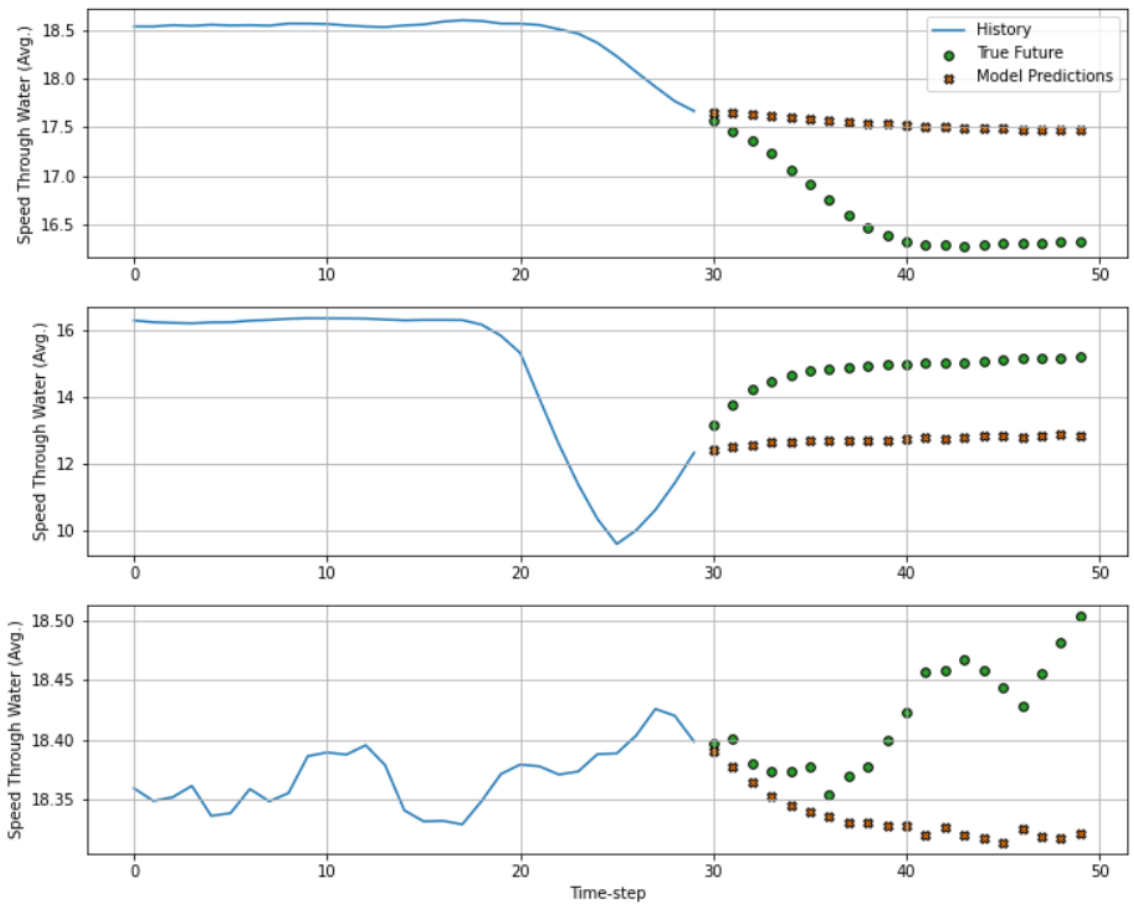


Figure 4.31: Indicative plots of the predictions using the simple linear network (30/20 configuration).

Using the MLP networks:

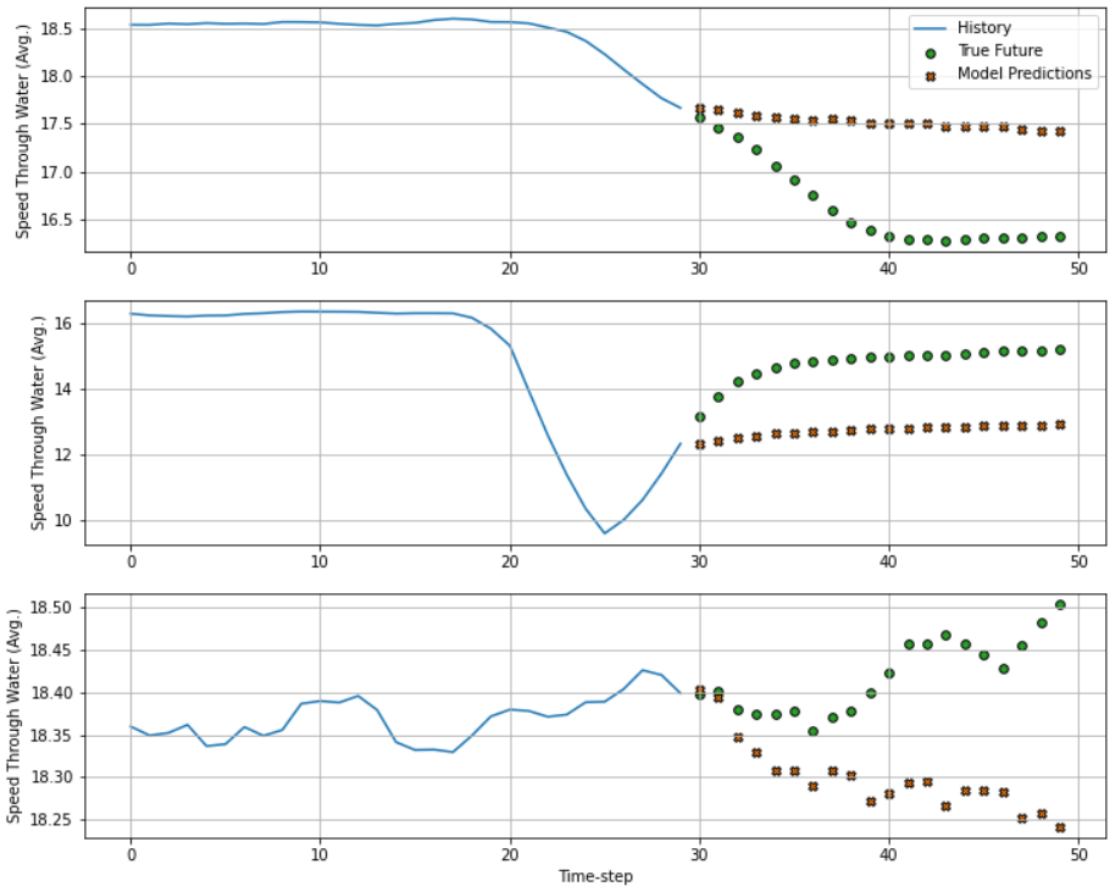


Figure 4.32: Indicative plots of the predictions using the MLP network (30/20 configuration).

Using the Convolution network:

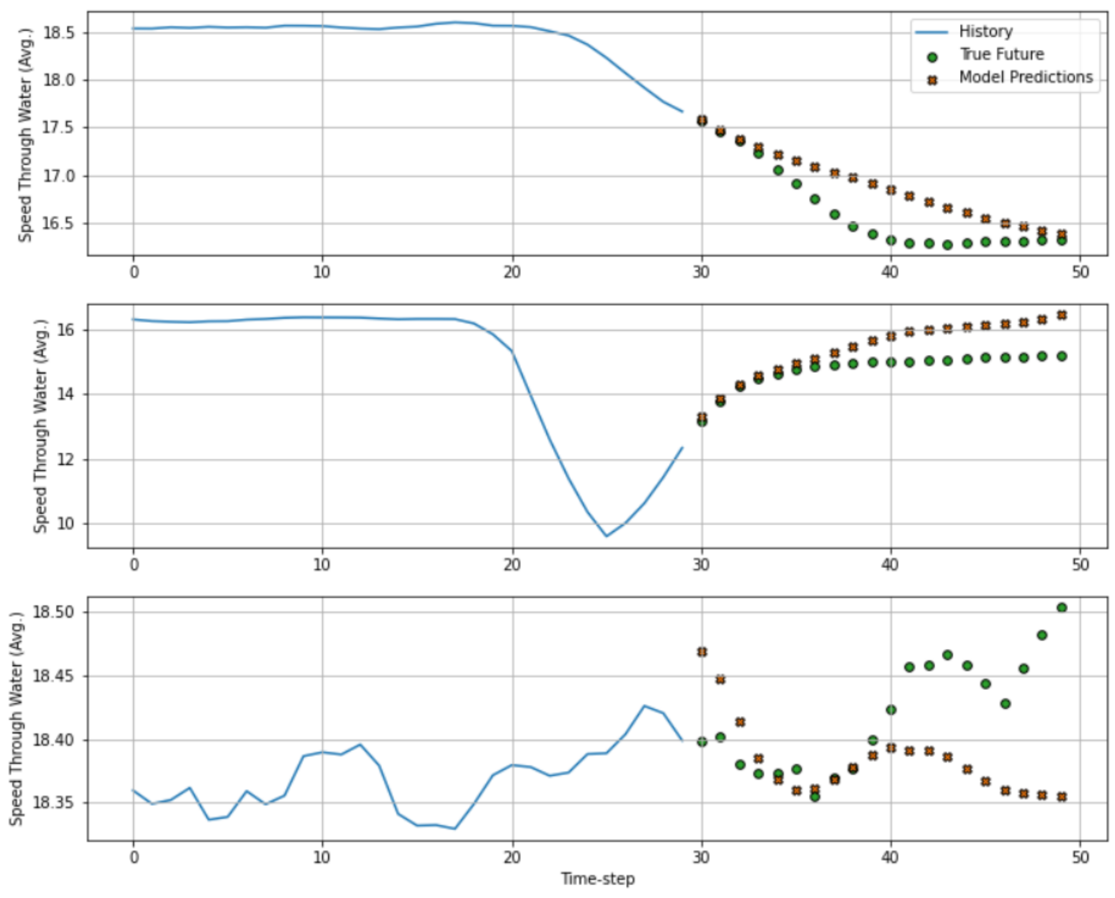


Figure 4.33: Indicative plots of the predictions using the Convolution network (30/20 configuration).

Using the Vanilla LSTM network:

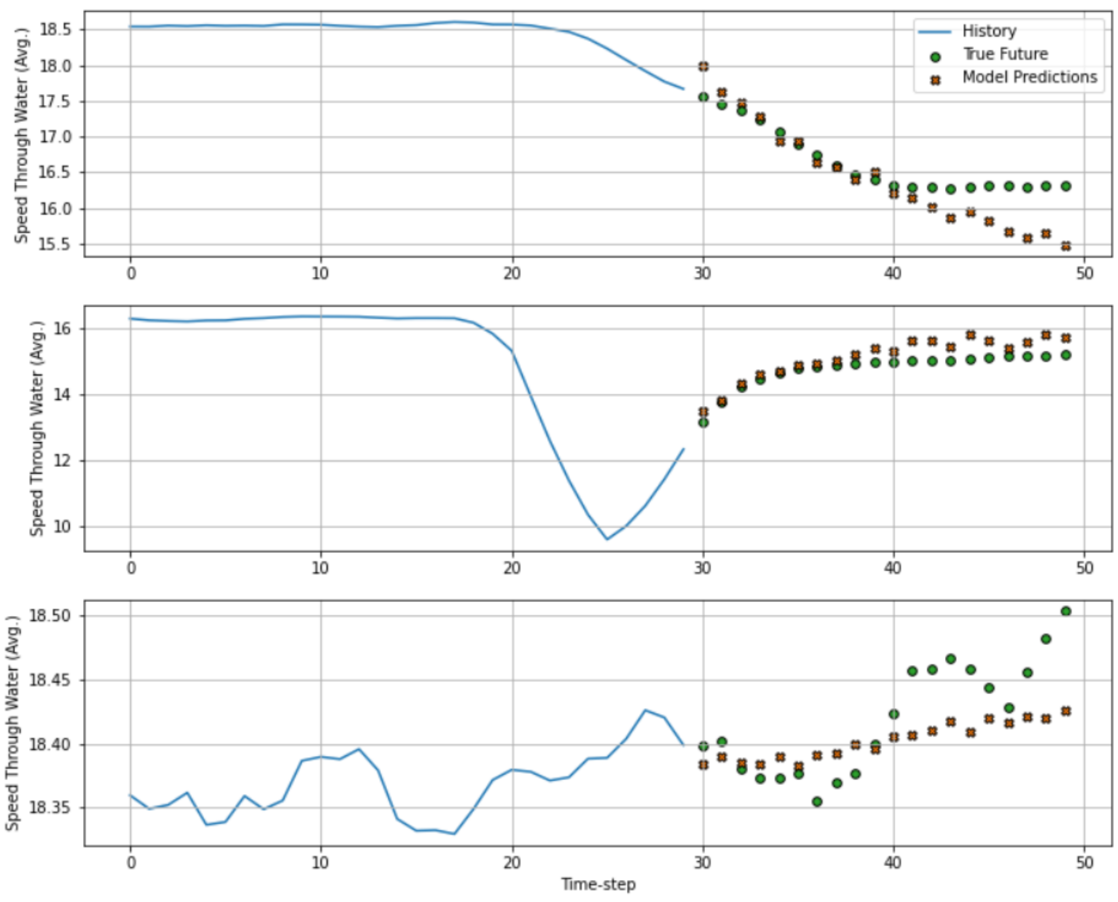


Figure 4.34: Indicative plots of the predictions using the Vanilla LSTM network (30/20 configuration).

Using the Stacked LSTM network:

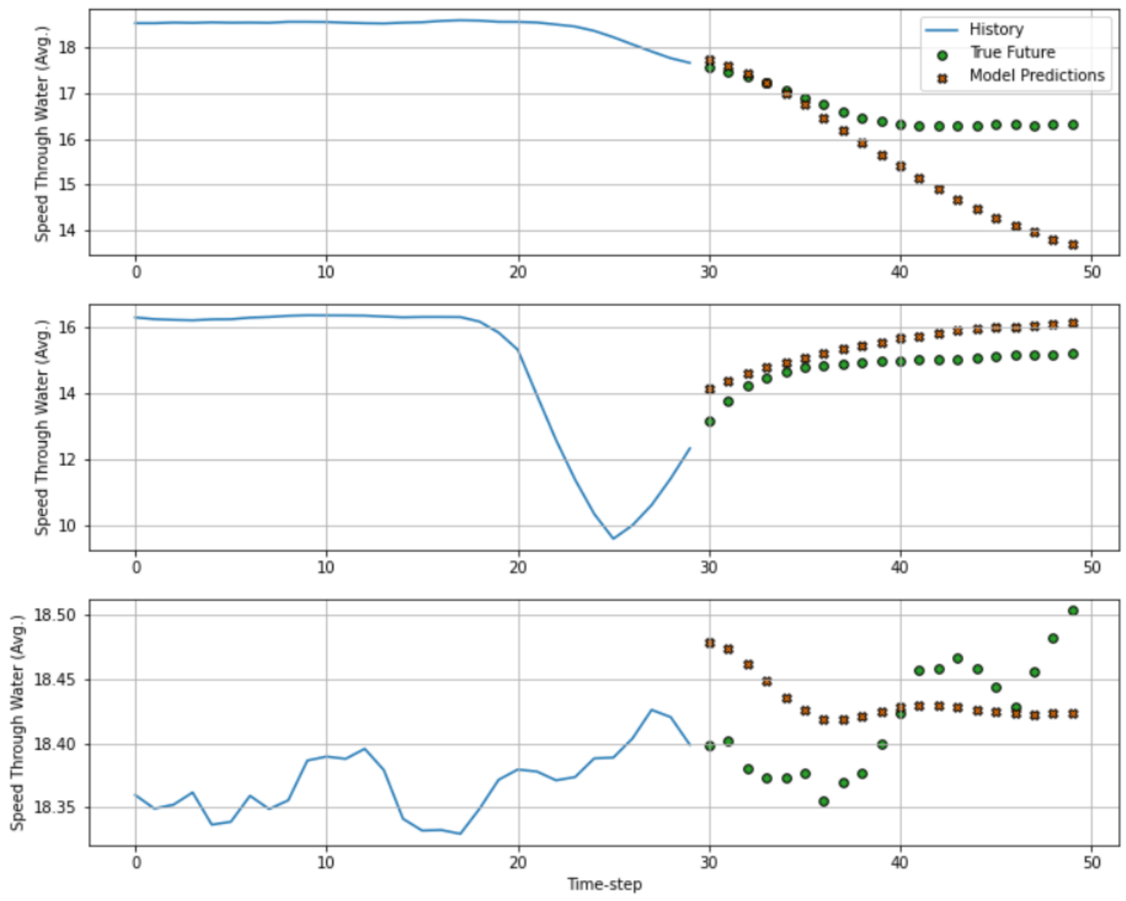


Figure 4.35: Indicative plots of the predictions using the Stacked LSTM network (30/20 configuration).

In comparison, the prediction mean absolute error is estimated as:

Table 4.5: Mean Absolute Percentage Error & Mean Absolute Error of 30/20 configuration

	MAPE %	MAE [kn]
Baseline	106.7	1.235
Linear	25.1	0.178
MLP	24.8	0.191
Convolution	23.5	0.221
LSTM	27.0	0.214
Stacked LSTM	28.0	0.209

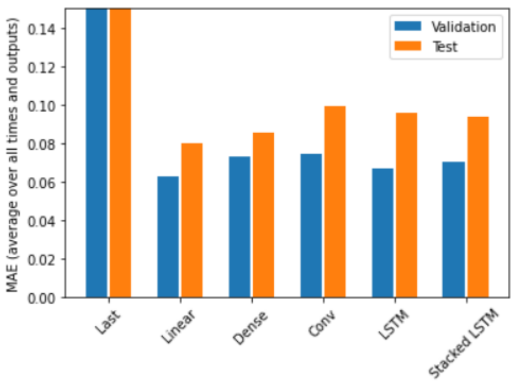


Figure 4.36: Performance comparison (30/20 configuration).

Once again, increasing the prediction horizon affects prediction accuracy. Greater generalization power observed in the MLP architecture.

For the case of predicting 30 timesteps into the future, the input and label sets, together with the common sense baseline are shaped as follows:

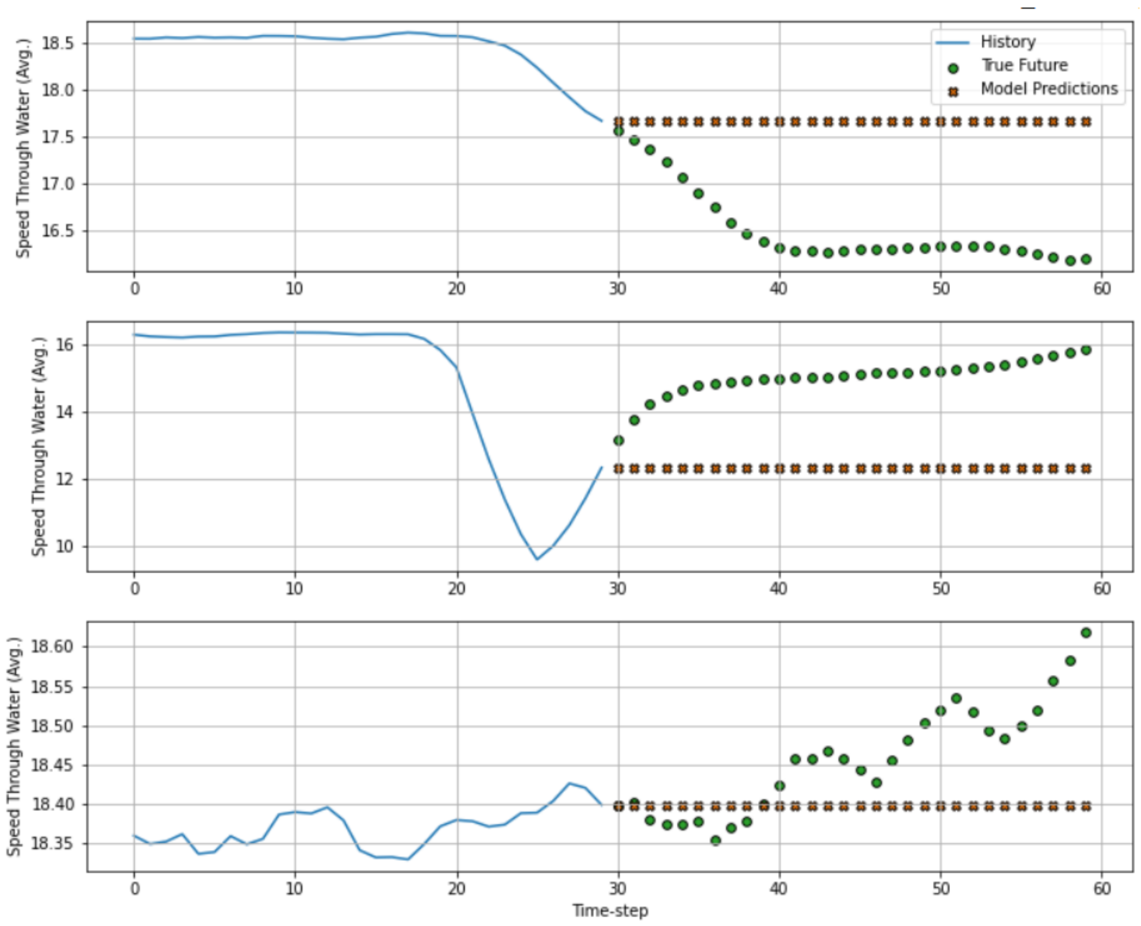


Figure 4.37: Indicative plots of the common sense baseline (30/30 configuration).

Using the simple linear system:

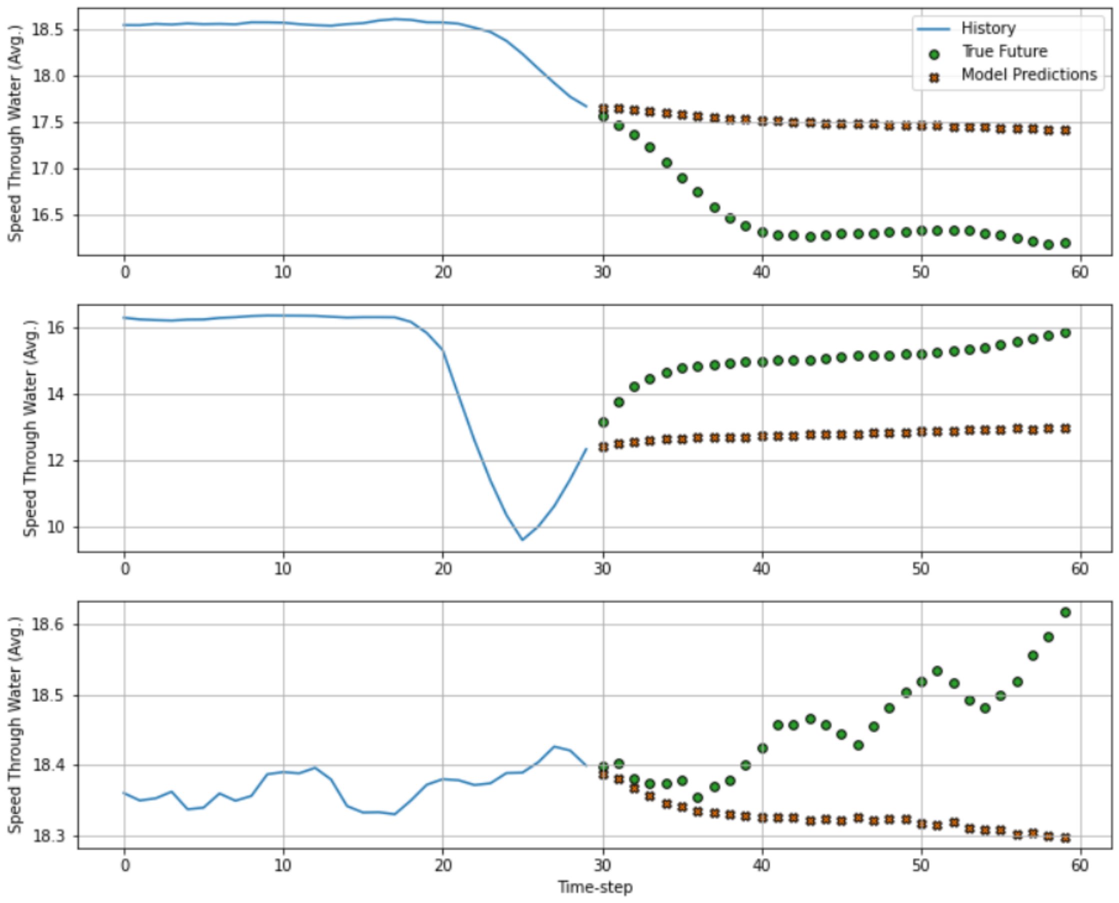


Figure 4.38: Indicative plots of the predictions using the simple linear network (30/30 configuration).

Using the MLP networks:

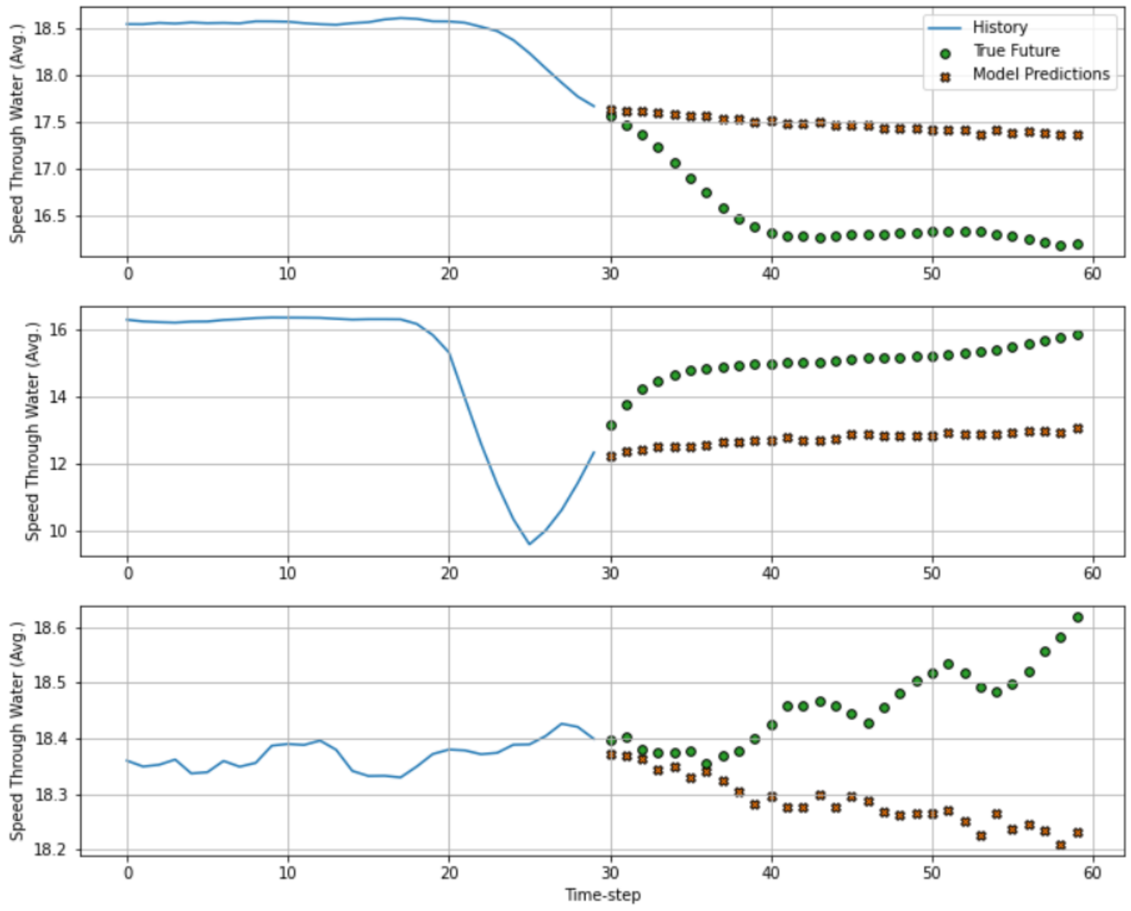


Figure 4.39: Indicative plots of the predictions using the MLP network (30/30 configuration).

Using the Convolution network:

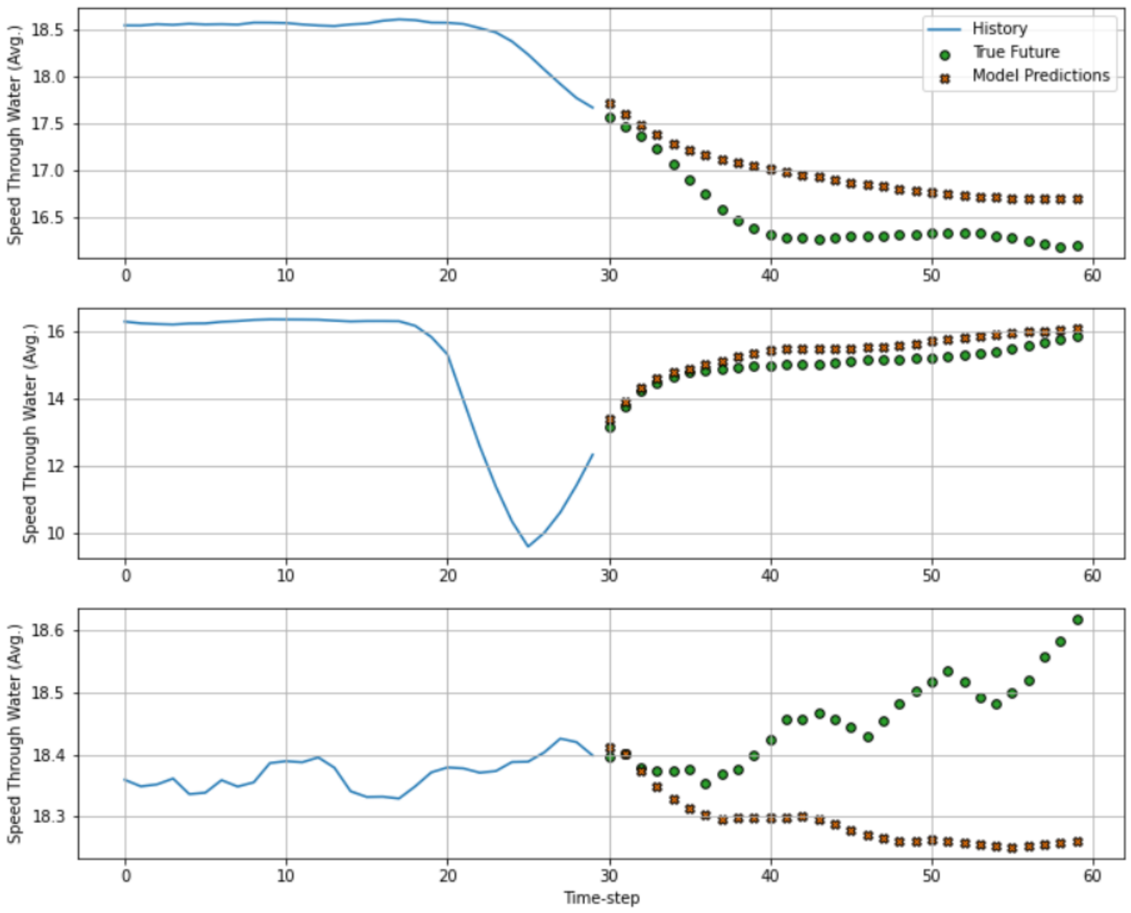


Figure 4.40: Indicative plots of the predictions using the Convolution network (30/30 configuration).

Using the Vanilla LSTM network:

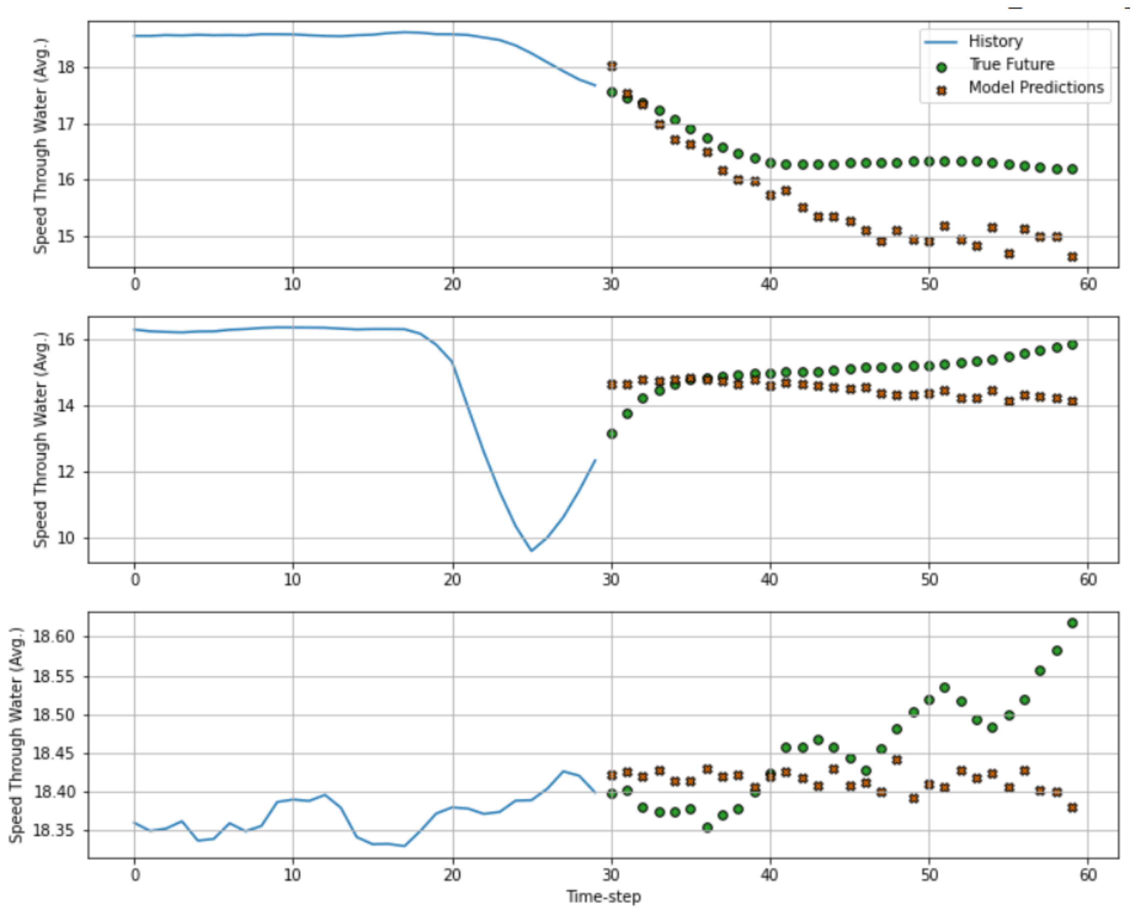


Figure 4.41: Indicative plots of the predictions using the Vanilla LSTM network (30/30 configuration).

Using the Stacked LSTM network:

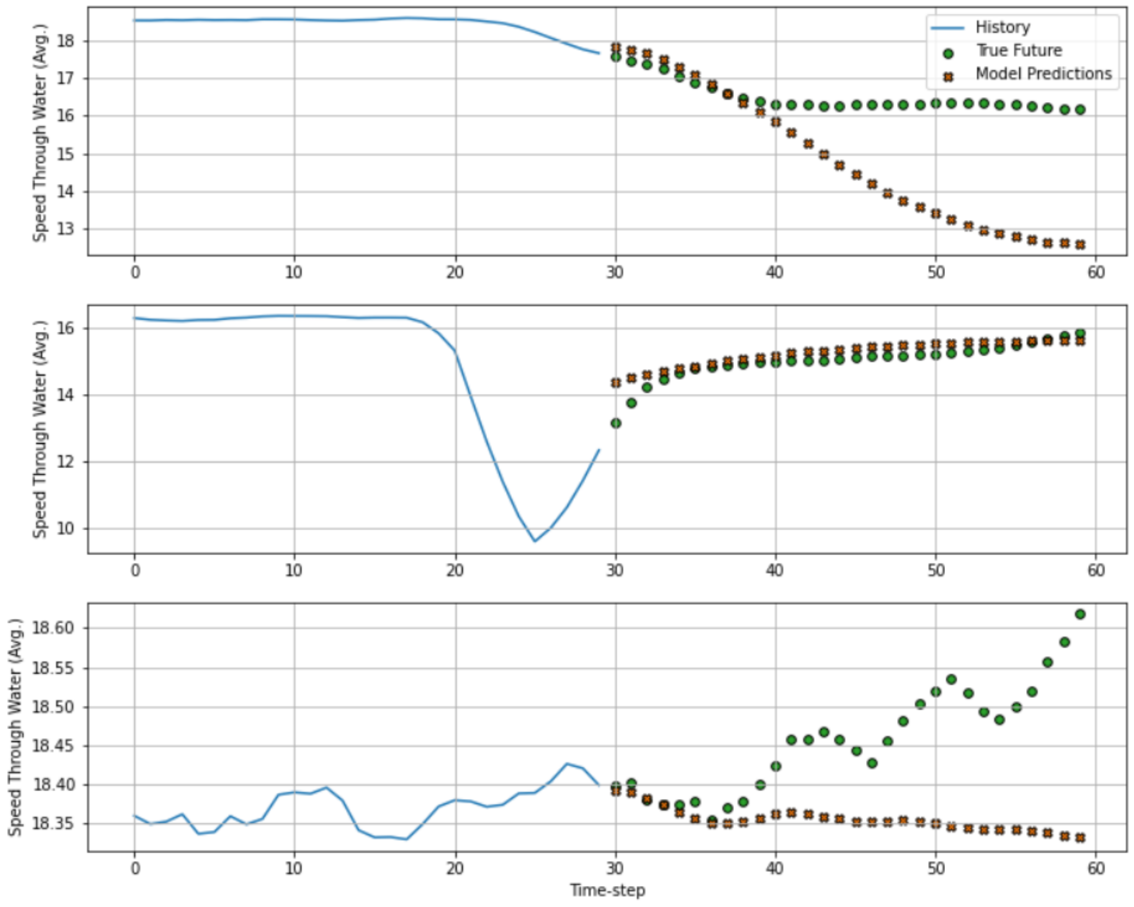


Figure 4.42: Indicative plots of the predictions using the Stacked LSTM network (30/30 configuration).

In comparison, the prediction mean absolute error is estimated as:

Table 4.6: Mean Absolute Percentage Error & Mean Absolute Error of 30/30 configuration

	MAPE %	MAE [kn]
Baseline	108.1	1.243
Linear	31.3	0.214
MLP	30.6	0.226
Convolution	28.7	0.308
LSTM	35.2	0.327
Stacked LSTM	33.2	0.286

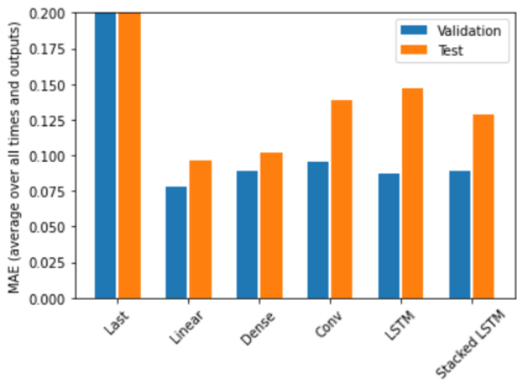


Figure 4.43: Performance comparison (30/30 configuration).

Even greater loss of accuracy is noted, together with the conservation of good behaviour by the MLP since the input window increased.

4.3 40 timesteps input

For the case of predicting 10 timesteps into the future, the input and label sets, together with the common sense baseline are shaped as follows:

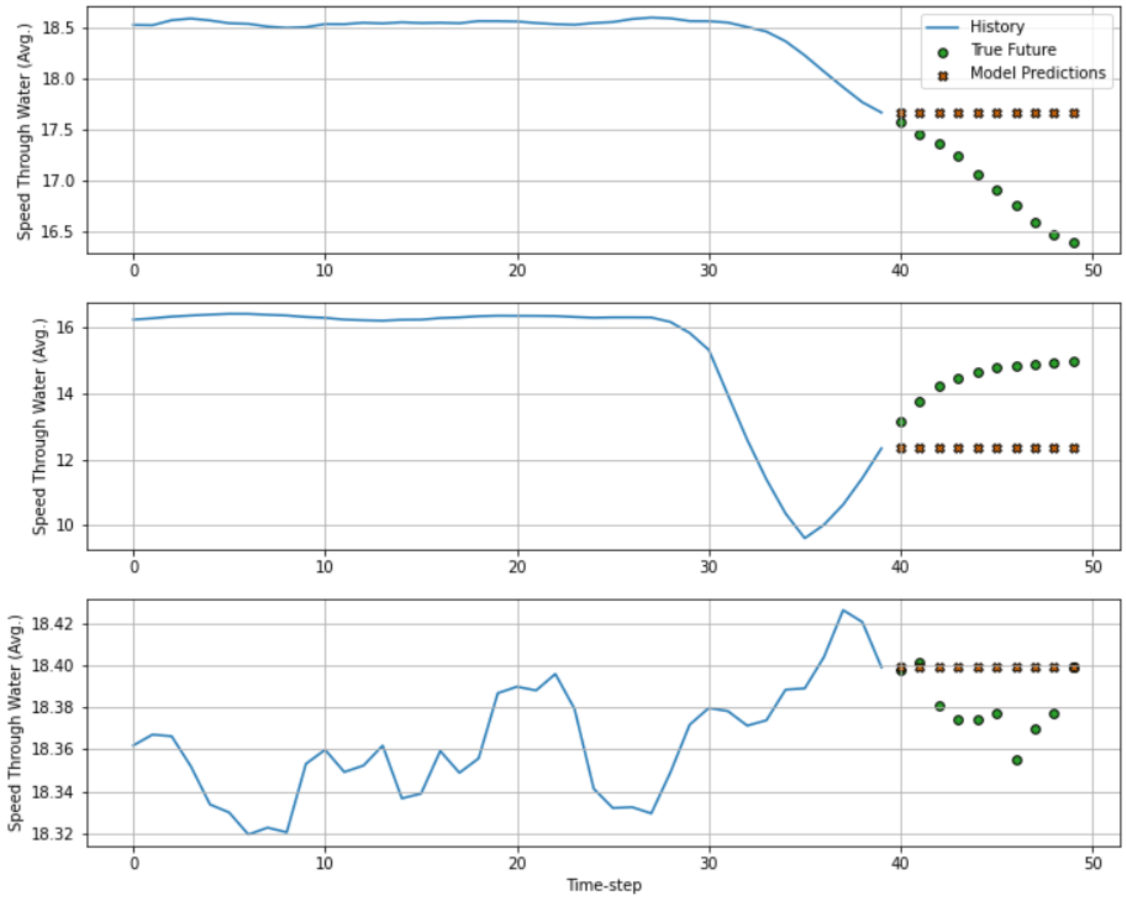


Figure 4.44: Indicative plots of the common sense baseline (40/10 configuration).

Using the simple linear system:

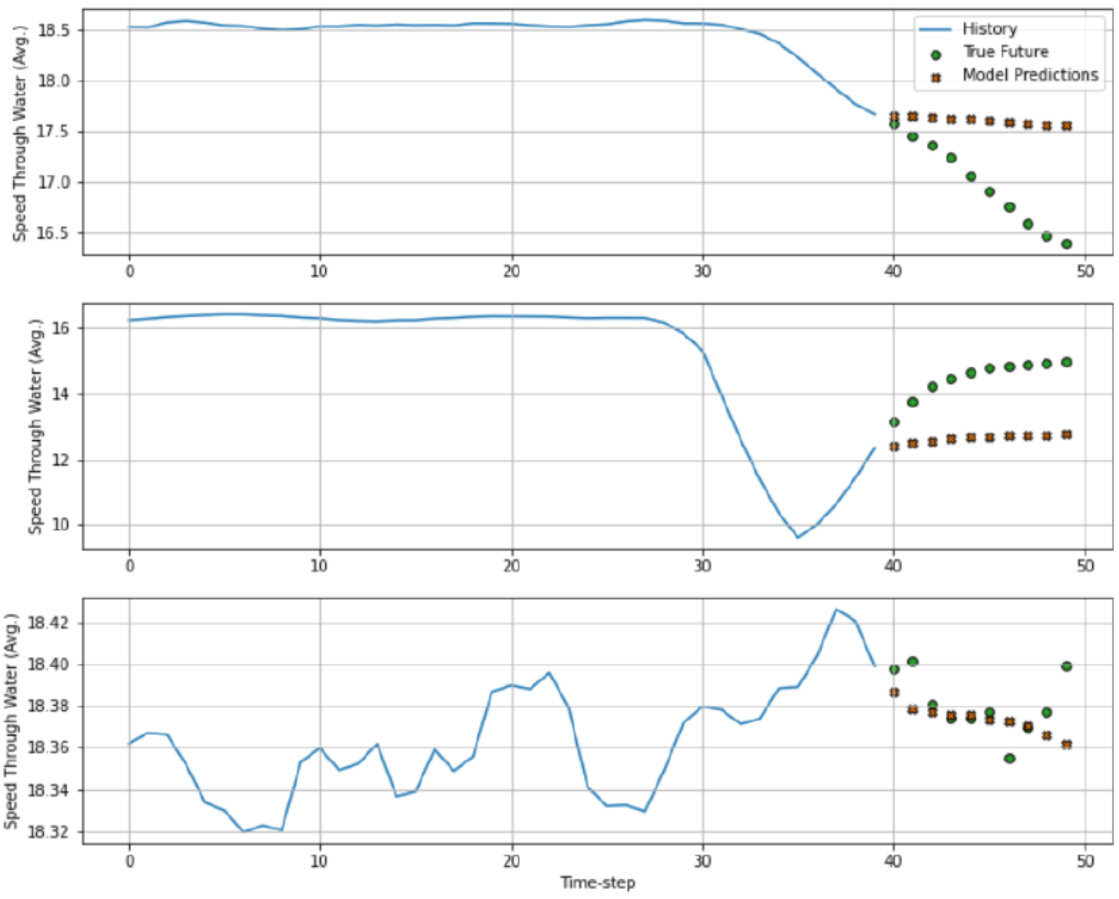


Figure 4.45: Indicative plots of the predictions using the simple linear network (40/10 configuration).

Using the MLP networks:

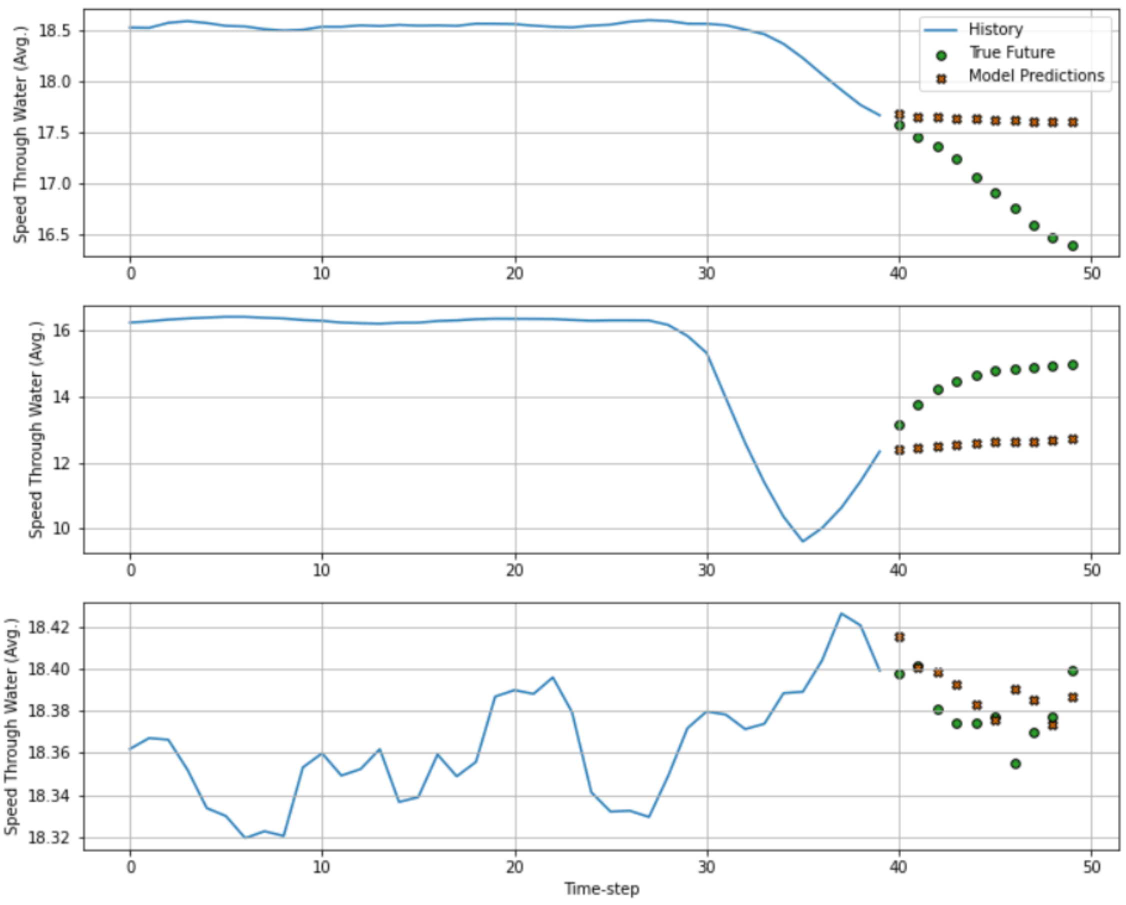


Figure 4.46: Indicative plots of the predictions using the MLP network (40/10 configuration).

Using the Convolution network:

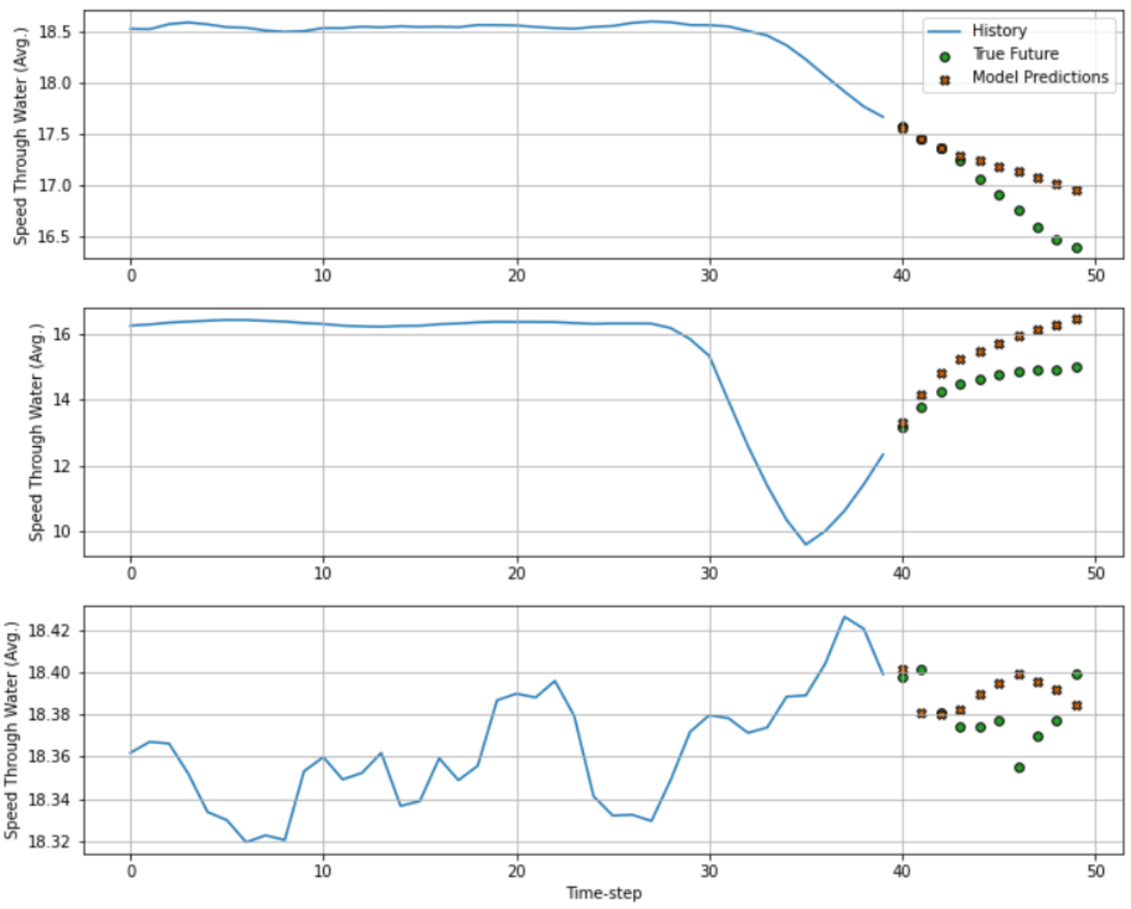


Figure 4.47: Indicative plots of the predictions using the Convolution network (40/10 configuration).

Using the Vanilla LSTM network:

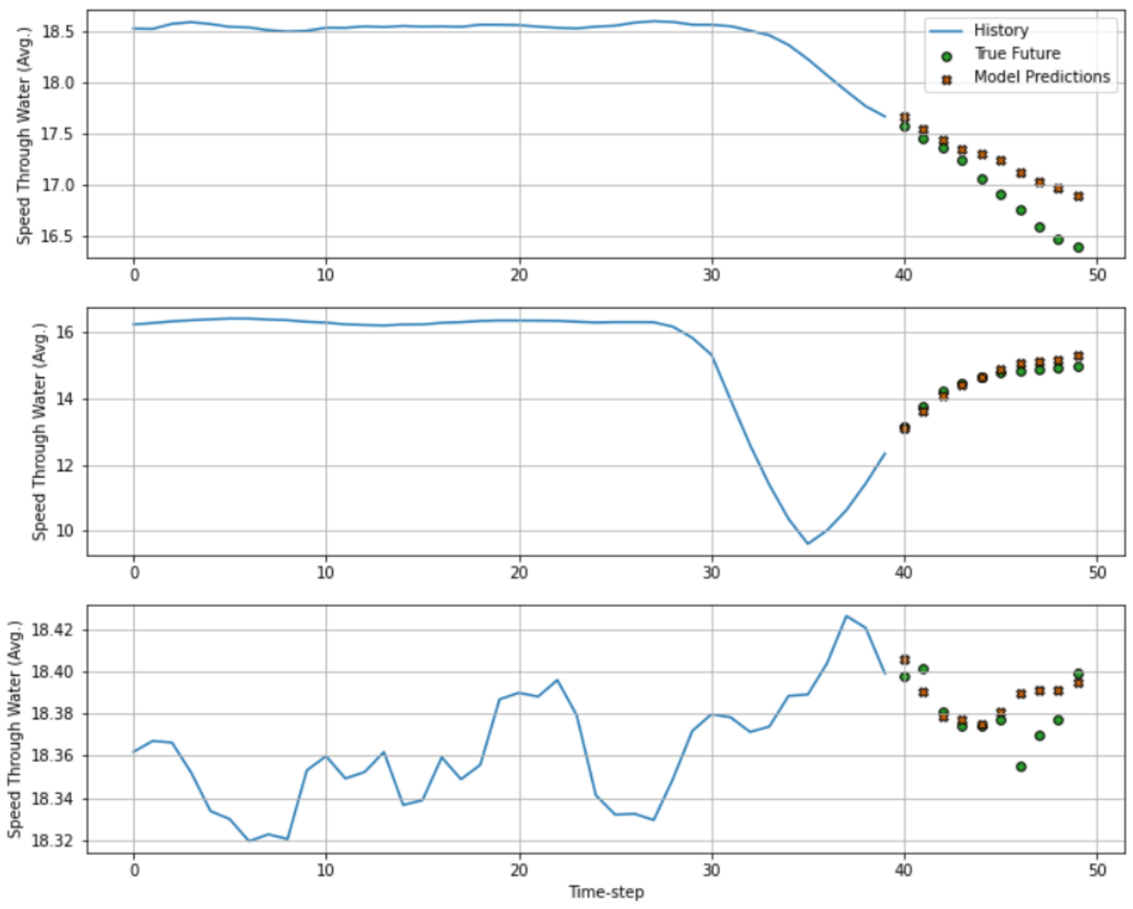


Figure 4.48: Indicative plots of the predictions using the Vanilla LSTM network (40/10 configuration).

Using the Stacked LSTM network:

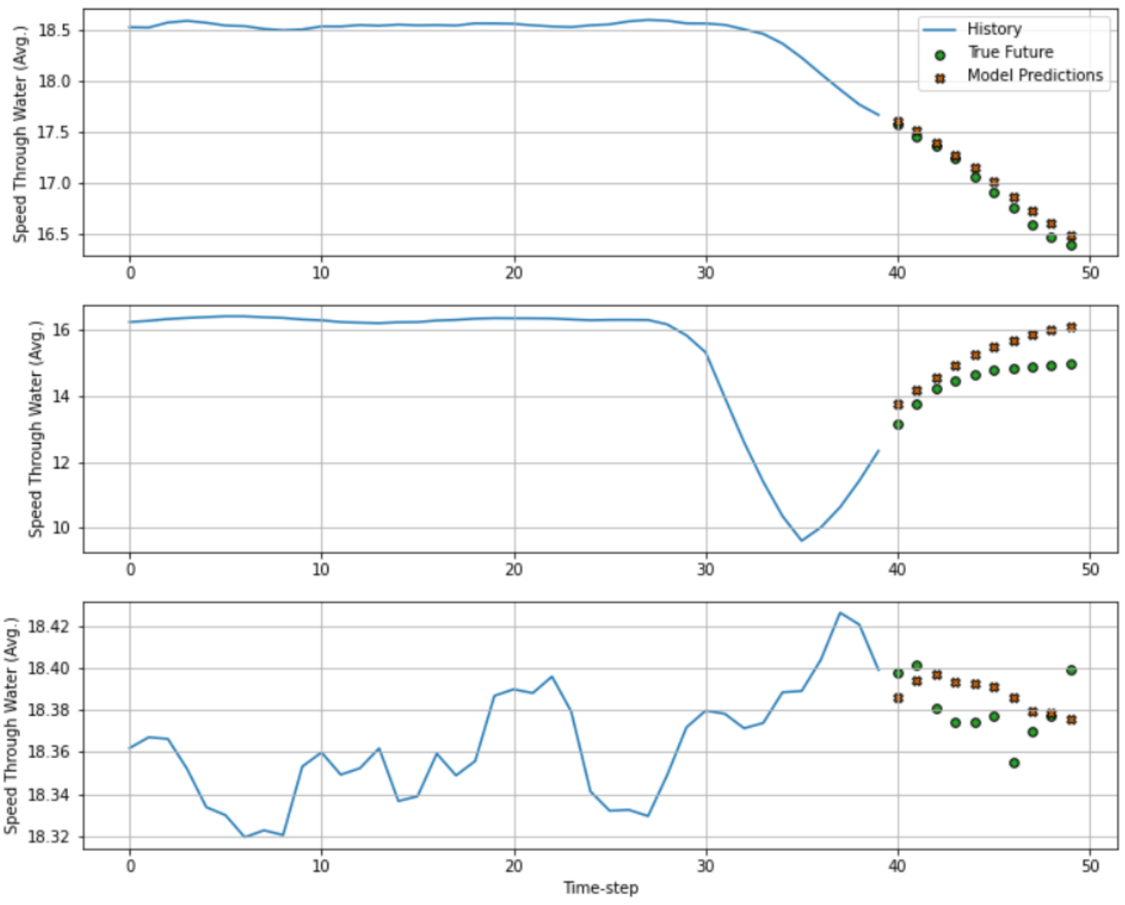


Figure 4.49: Indicative plots of the predictions using the Stacked LSTM network (40/10 configuration).

In comparison, the prediction mean absolute error is estimated as:

Table 4.7: Mean Absolute Percentage Error & Mean Absolute Error of 40/10 configuration

	MAPE %	MAE [kn]
Baseline	104.8	1.222
Linear	17.9	0.124
MLP	17.5	0.120
Convolution	17.4	0.162
LSTM	17.2	0.168
Stacked LSTM	16.9	0.122

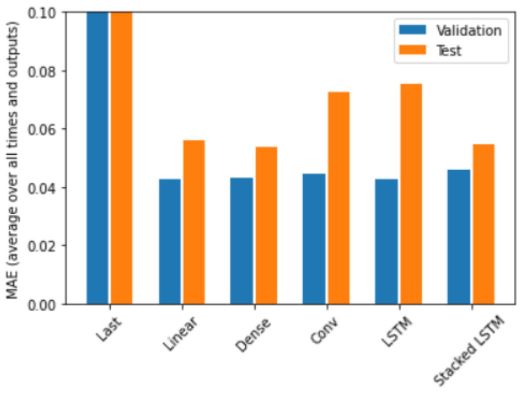


Figure 4.50: Performance comparison (40/10 configuration).

Stacked LSTM configuration shows its superiority in terms of generalization power when the prediction horizon is small. CNN and Vanilla LSTM seem to overfit.

For the case of predicting 20 timesteps into the future, the input and label sets, together with the common sense baseline are shaped as follows:

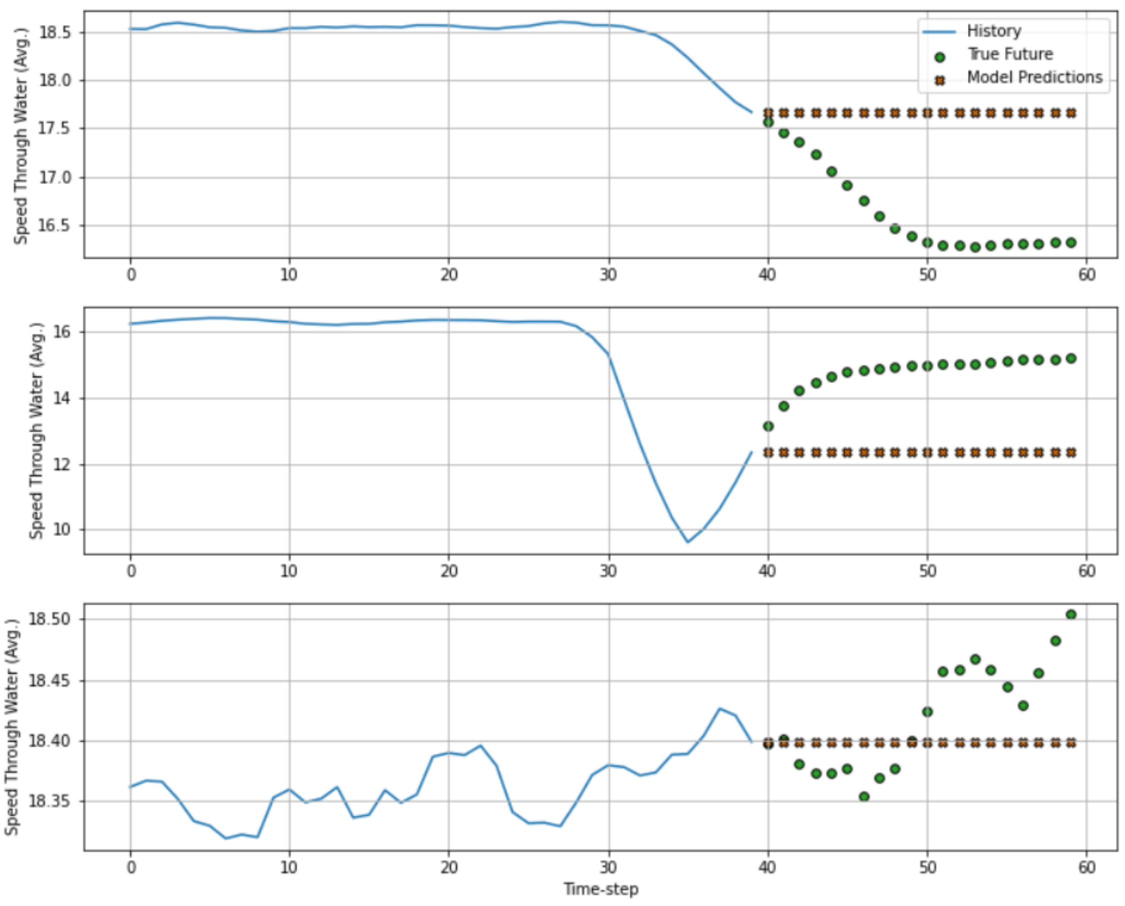


Figure 4.51: Indicative plots of the common sense baseline (40/20 configuration).

Using the simple linear system:

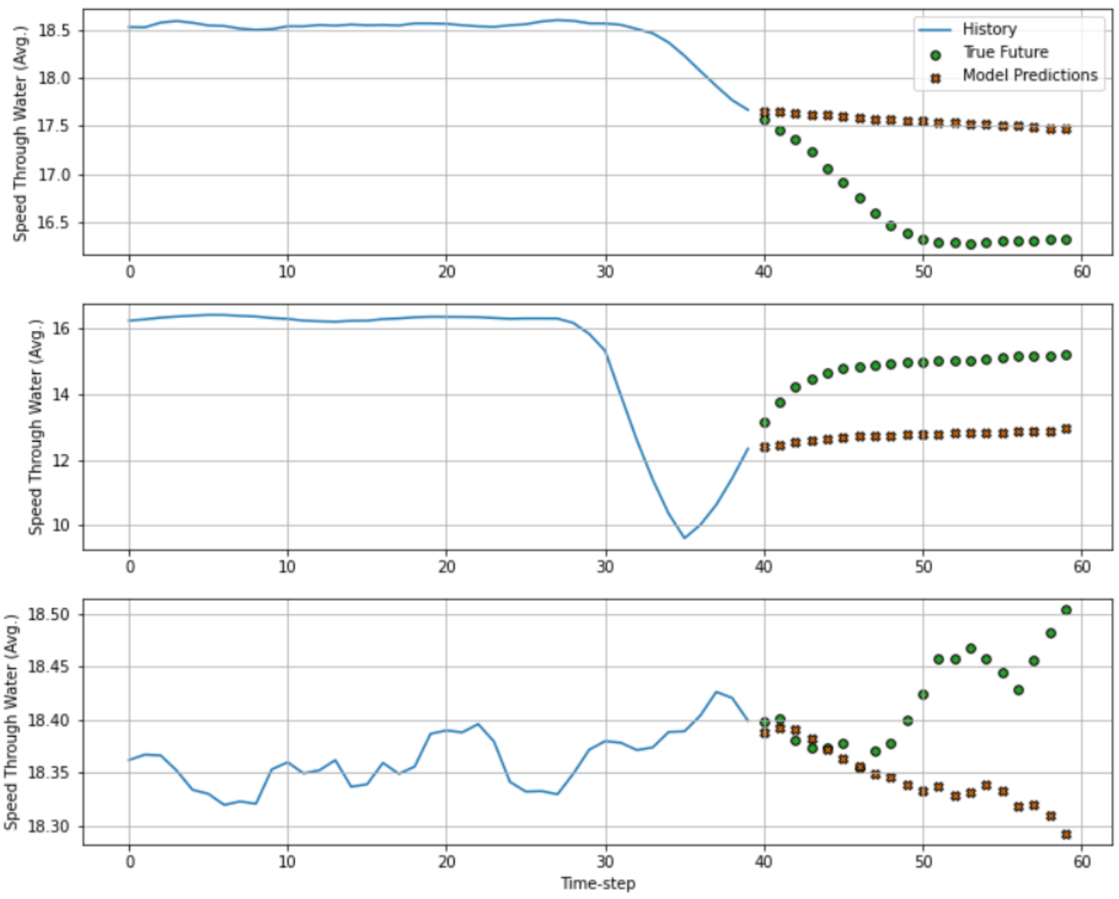


Figure 4.52: Indicative plots of the predictions using the simple linear network (40/20 configuration).

Using the MLP networks:

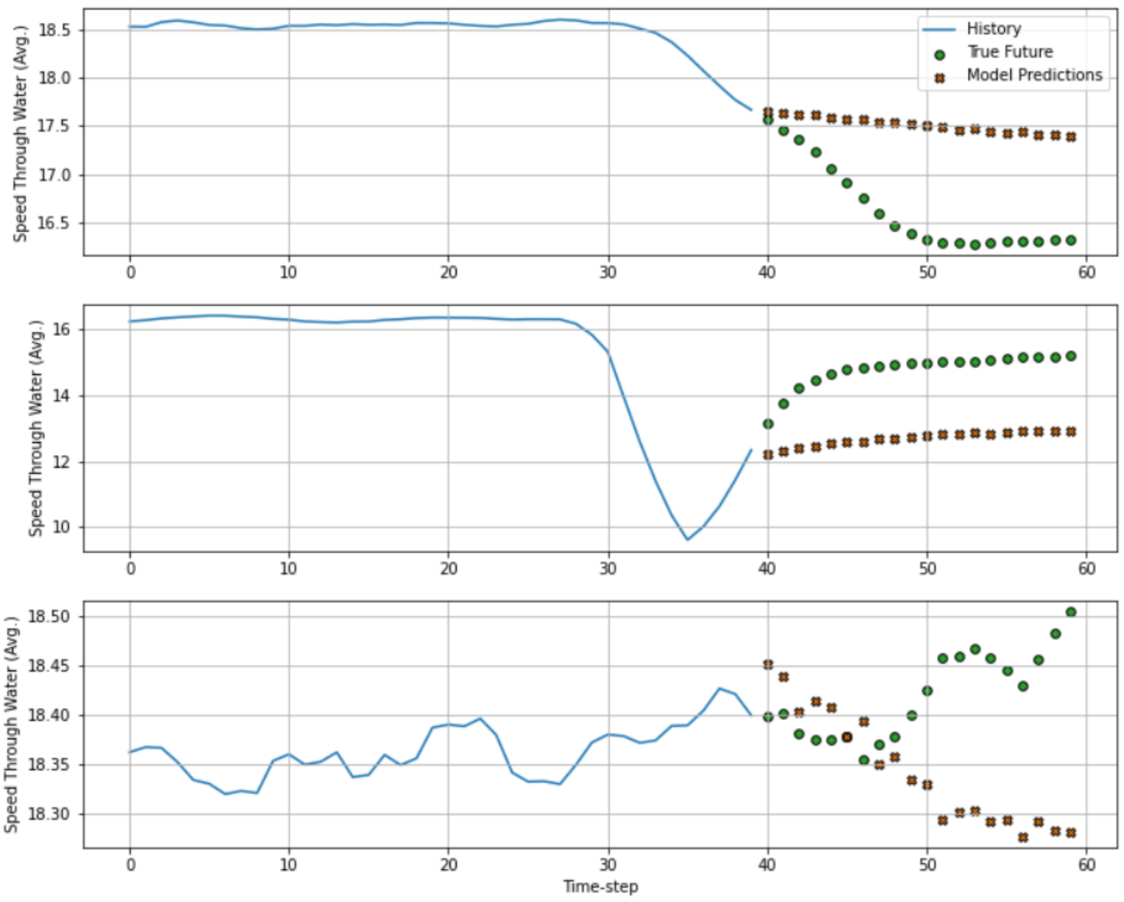


Figure 4.53: Indicative plots of the predictions using the MLP network (40/20 configuration).

Using the Convolution network:

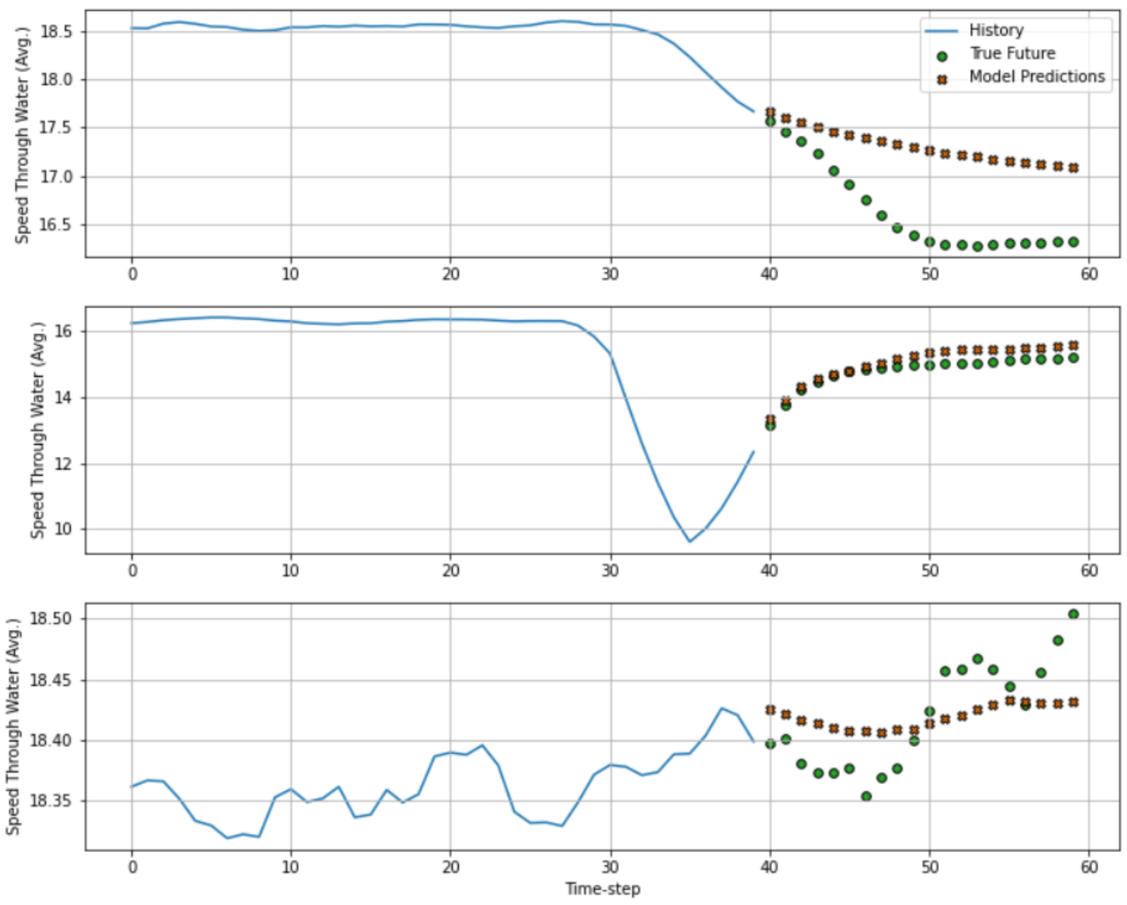


Figure 4.54: Indicative plots of the predictions using the Convolution network (40/20 configuration).

Using the Vanilla LSTM network:

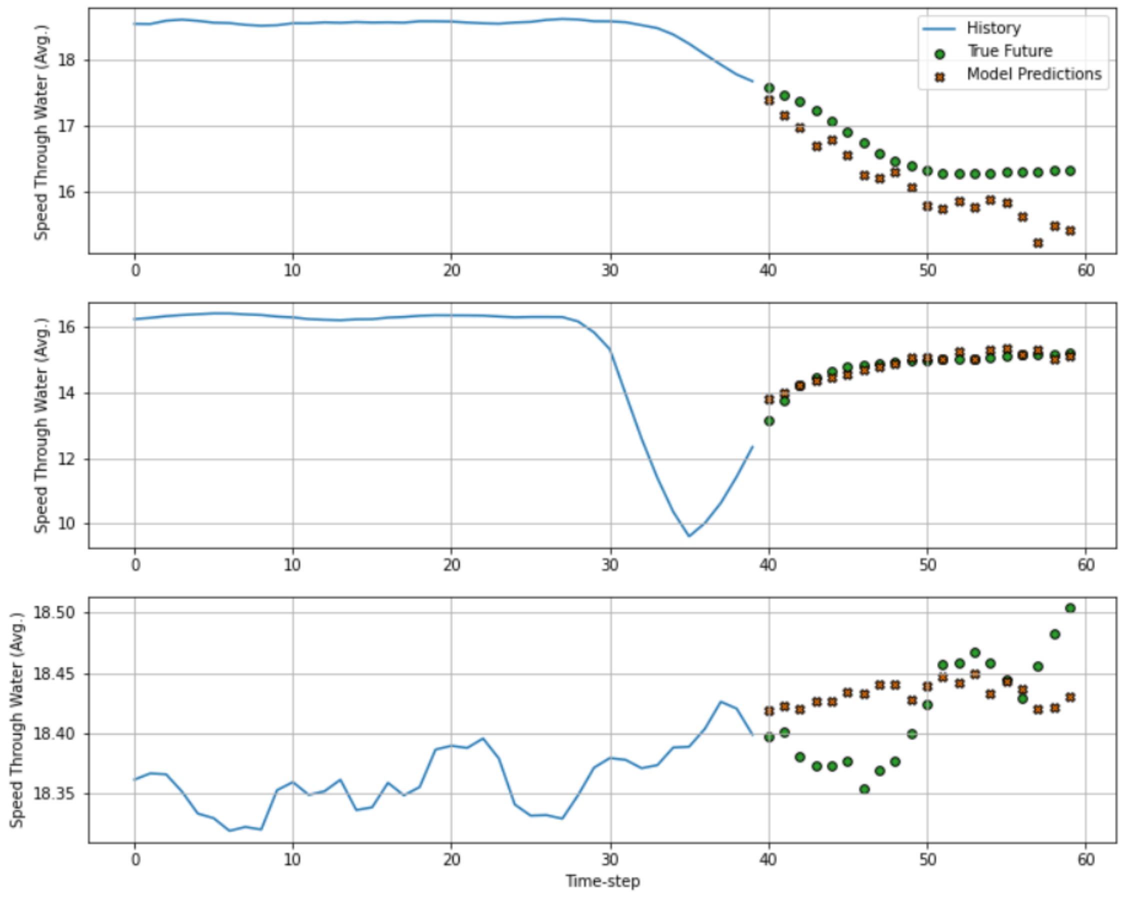


Figure 4.55: Indicative plots of the predictions using the Vanilla LSTM network (40/20 configuration).

Using the Stacked LSTM network:

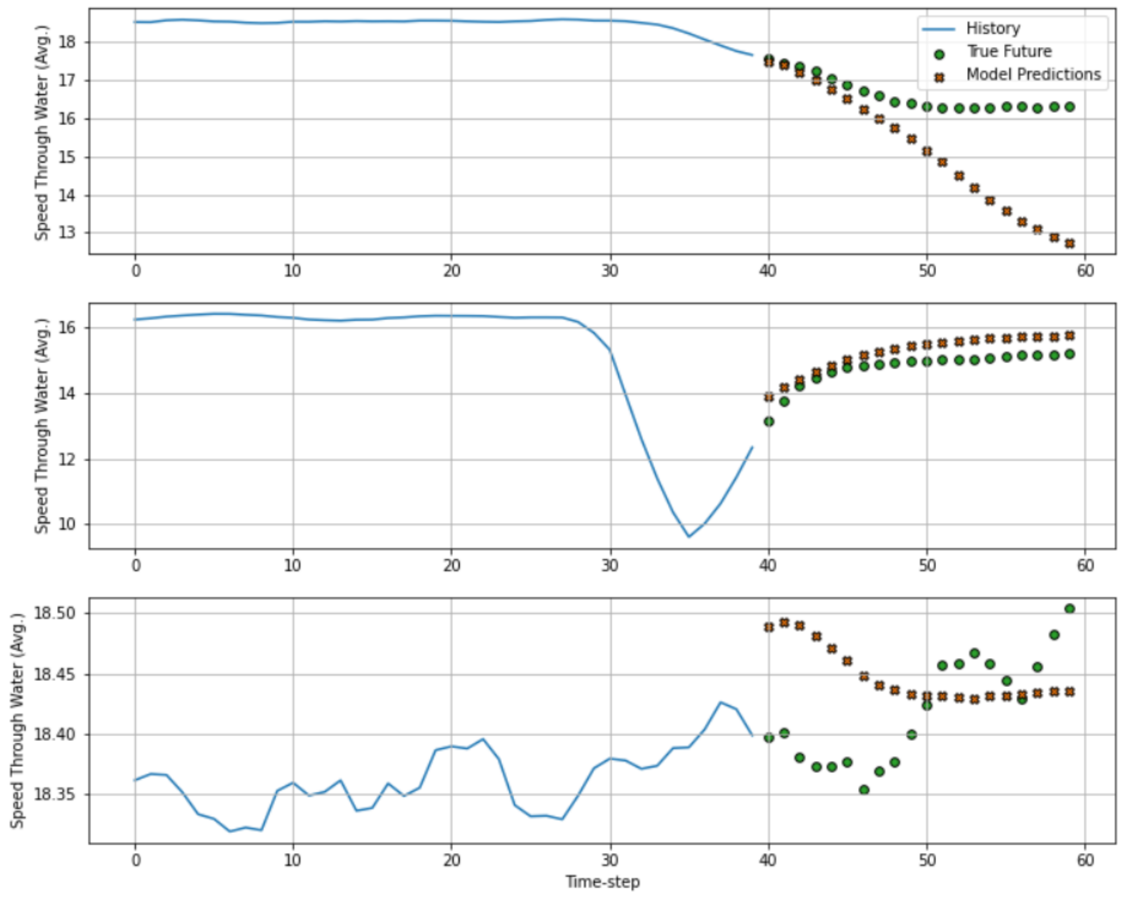


Figure 4.56: Indicative plots of the predictions using the Stacked LSTM network (40/20 configuration).

In comparison, the prediction mean absolute error is estimated as:

Table 4.8: Mean Absolute Percentage Error & Mean Absolute Error of 40/20 configuration

	MAPE %	MAE [kn]
Baseline	106.7	1.235
Linear	24.7	0.174
MLP	26.1	0.186
Convolution	24.7	0.230
LSTM	24.0	0.191
Stacked LSTM	29.0	0.209

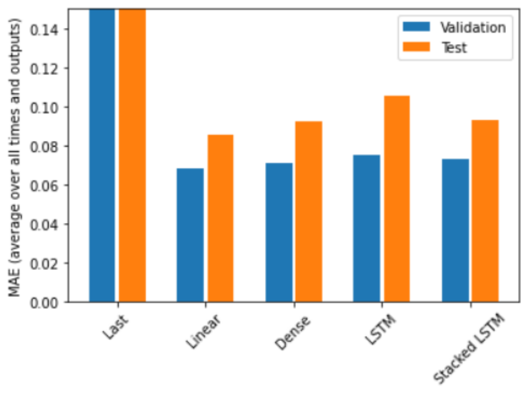


Figure 4.57: Performance comparison (40/20 configuration).

Similar behaviour observed for the Linear, Dense and Stacked LSTM networks.

For the case of predicting 30 timesteps into the future, the input and label sets, together with the common sense baseline are shaped as follows:

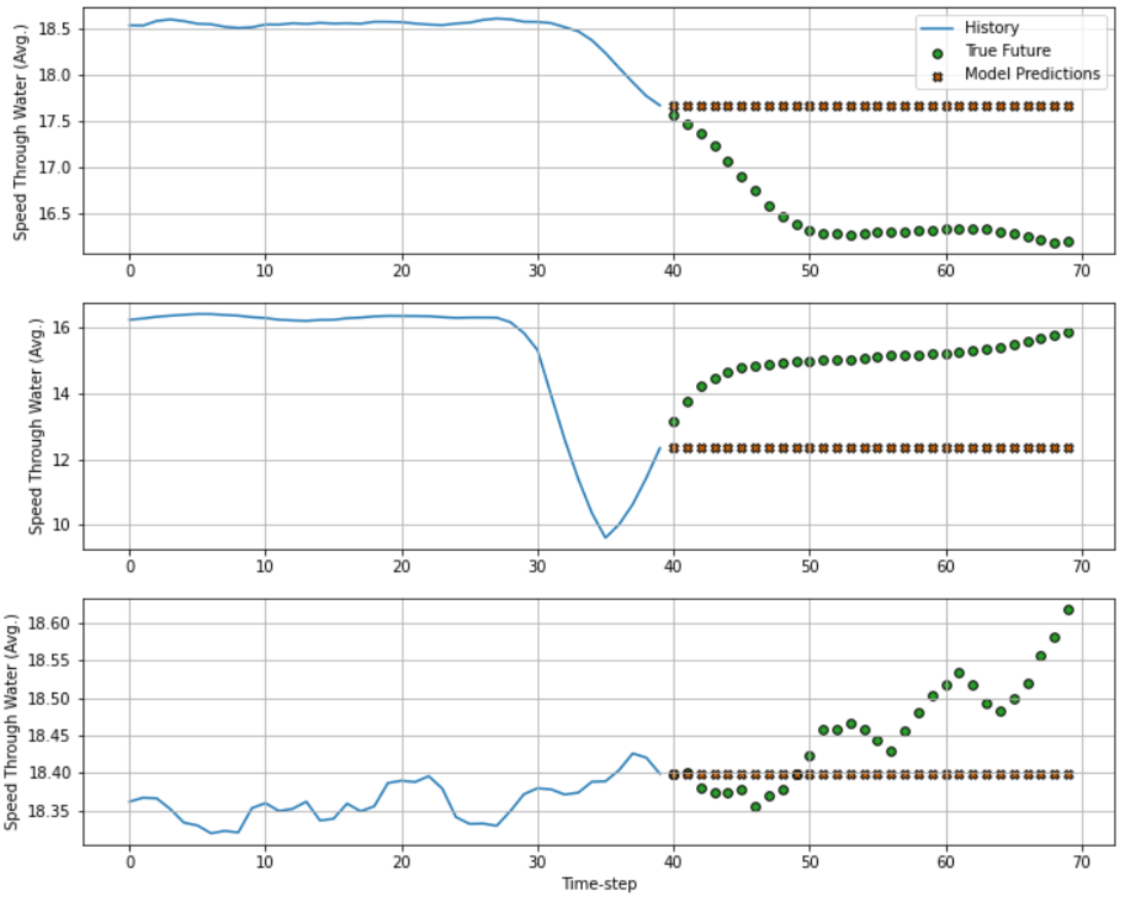


Figure 4.58: Indicative plots of the common sense baseline (40/30 configuration).

Using the simple linear system:

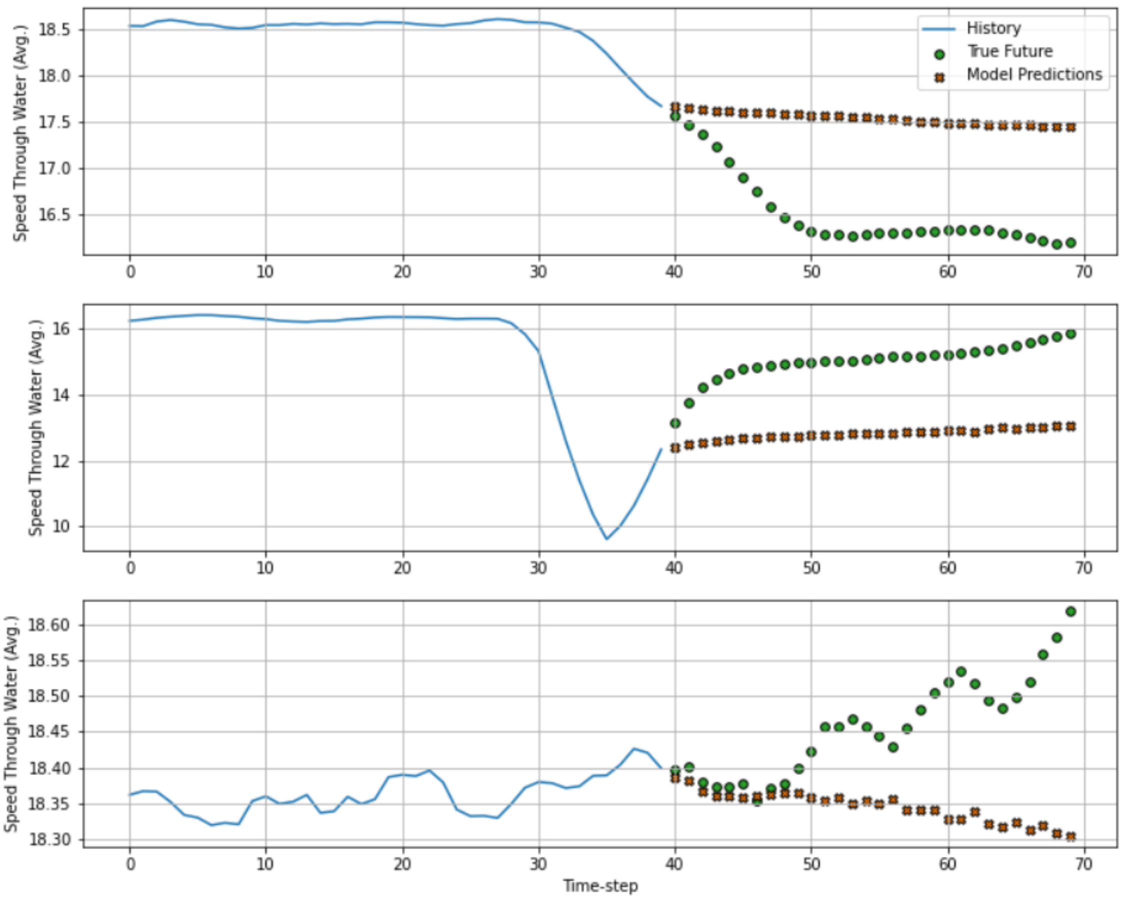


Figure 4.59: Indicative plots of the predictions using the simple linear network (40/30 configuration).

Using the MLP networks:

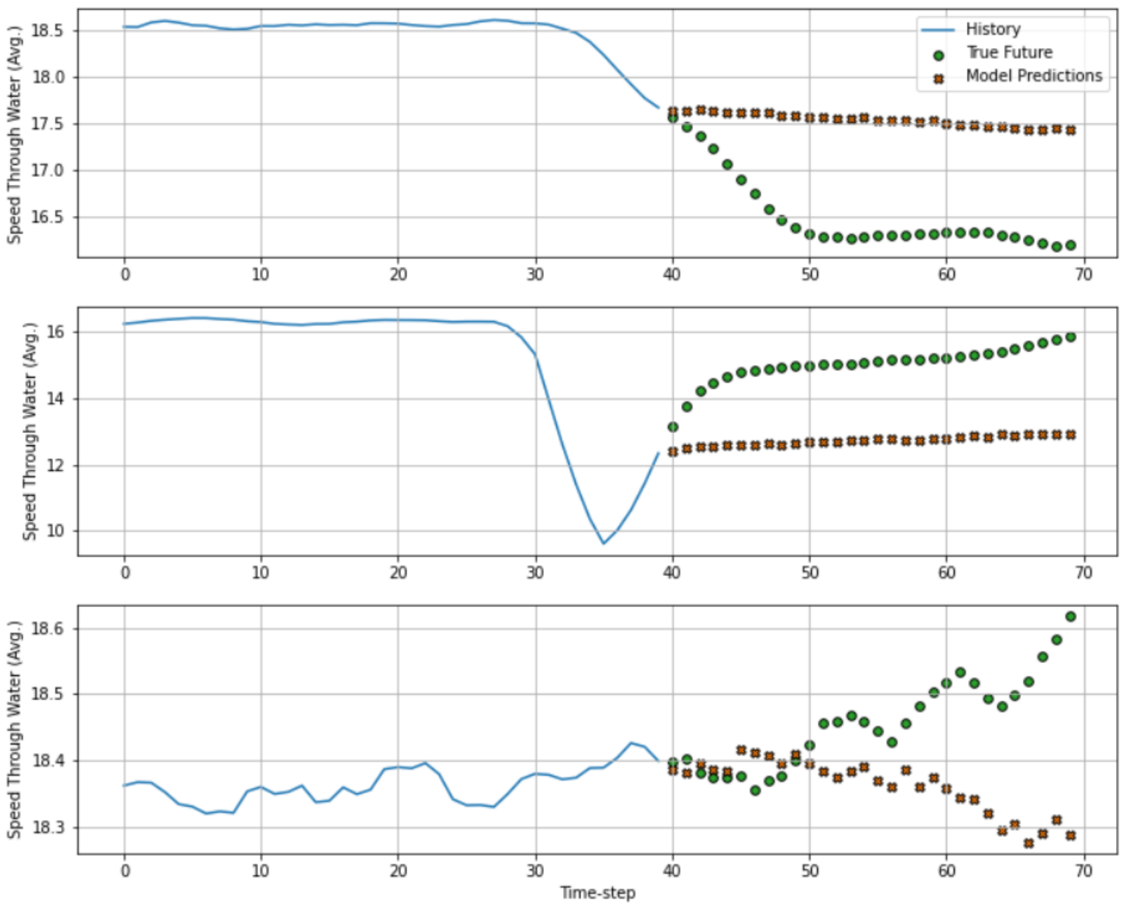


Figure 4.60: Indicative plots of the predictions using the MLP network (40/30 configuration).

Using the Convolution network:

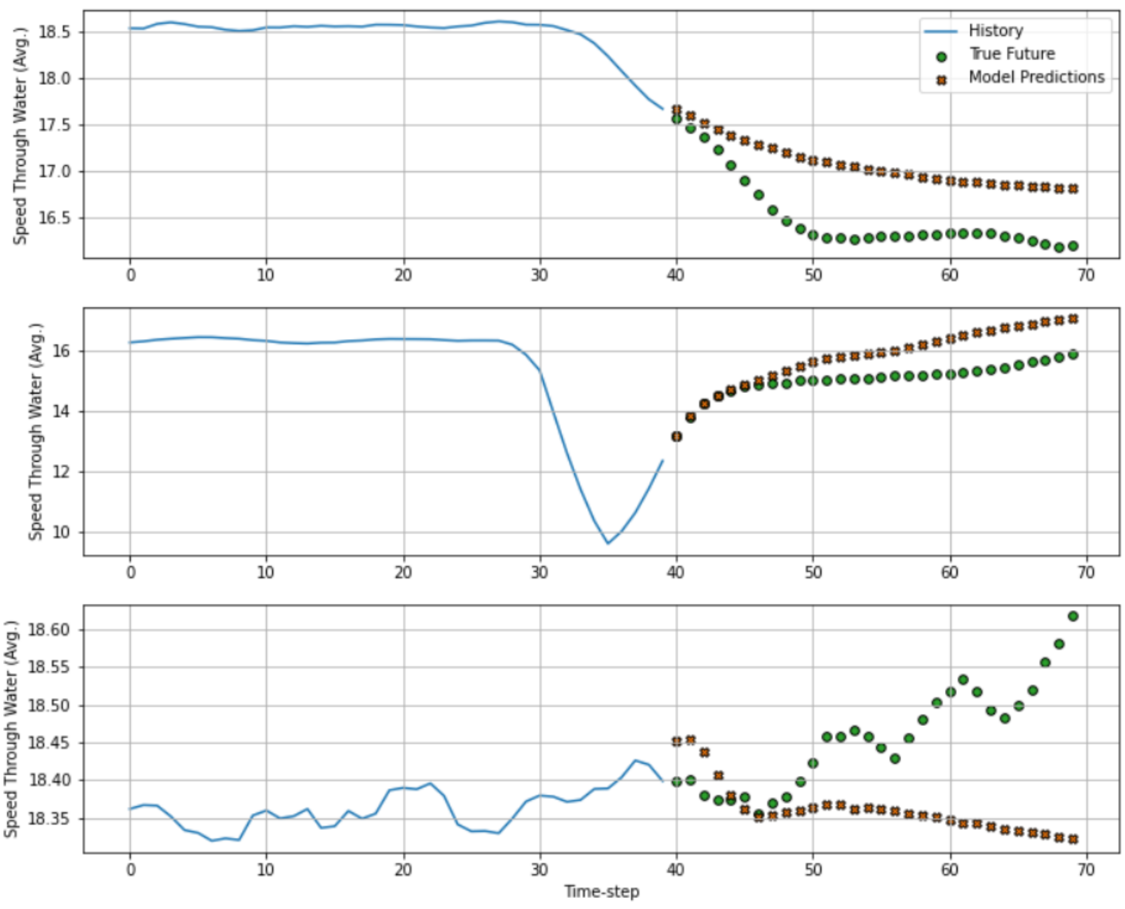


Figure 4.61: Indicative plots of the predictions using the Convolution network (40/30 configuration).

Using the Vanilla LSTM network:

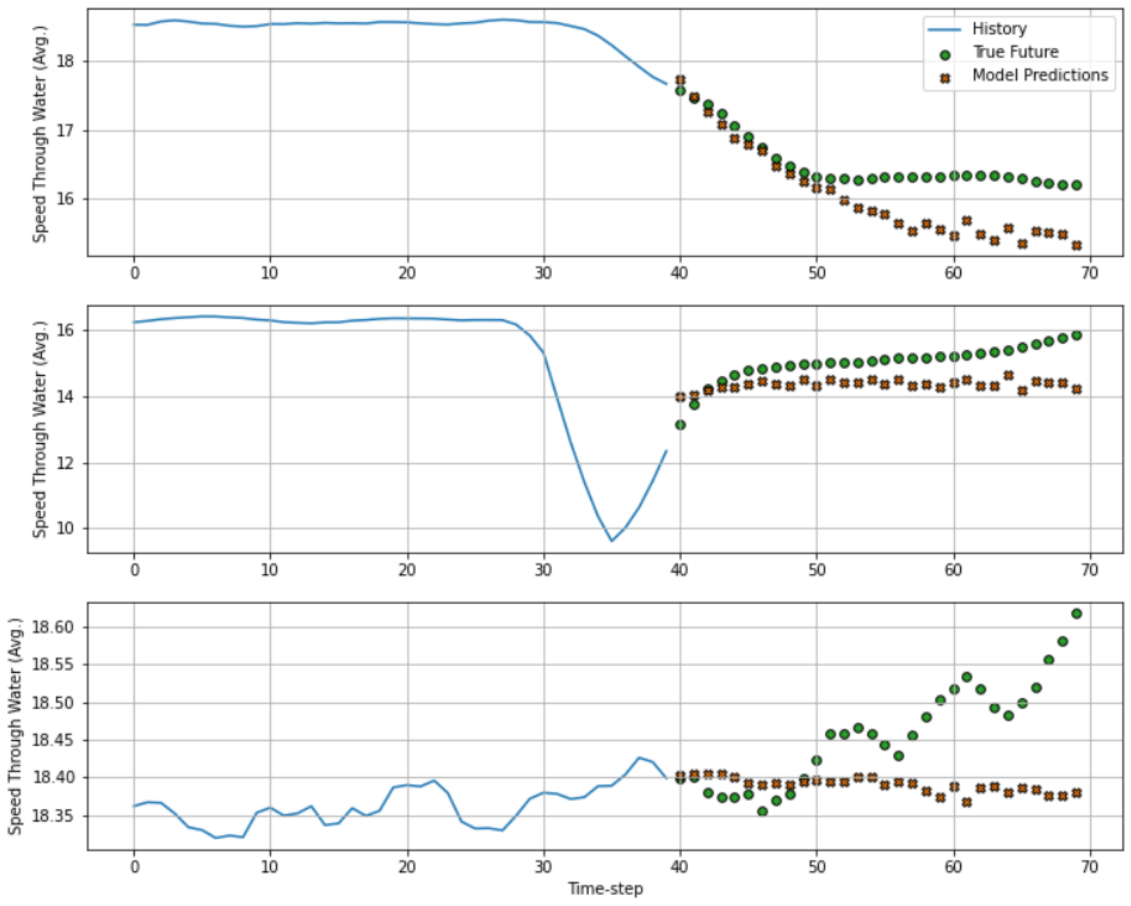


Figure 4.62: Indicative plots of the predictions using the Vanilla LSTM network (40/30 configuration).

Using the Stacked LSTM network:

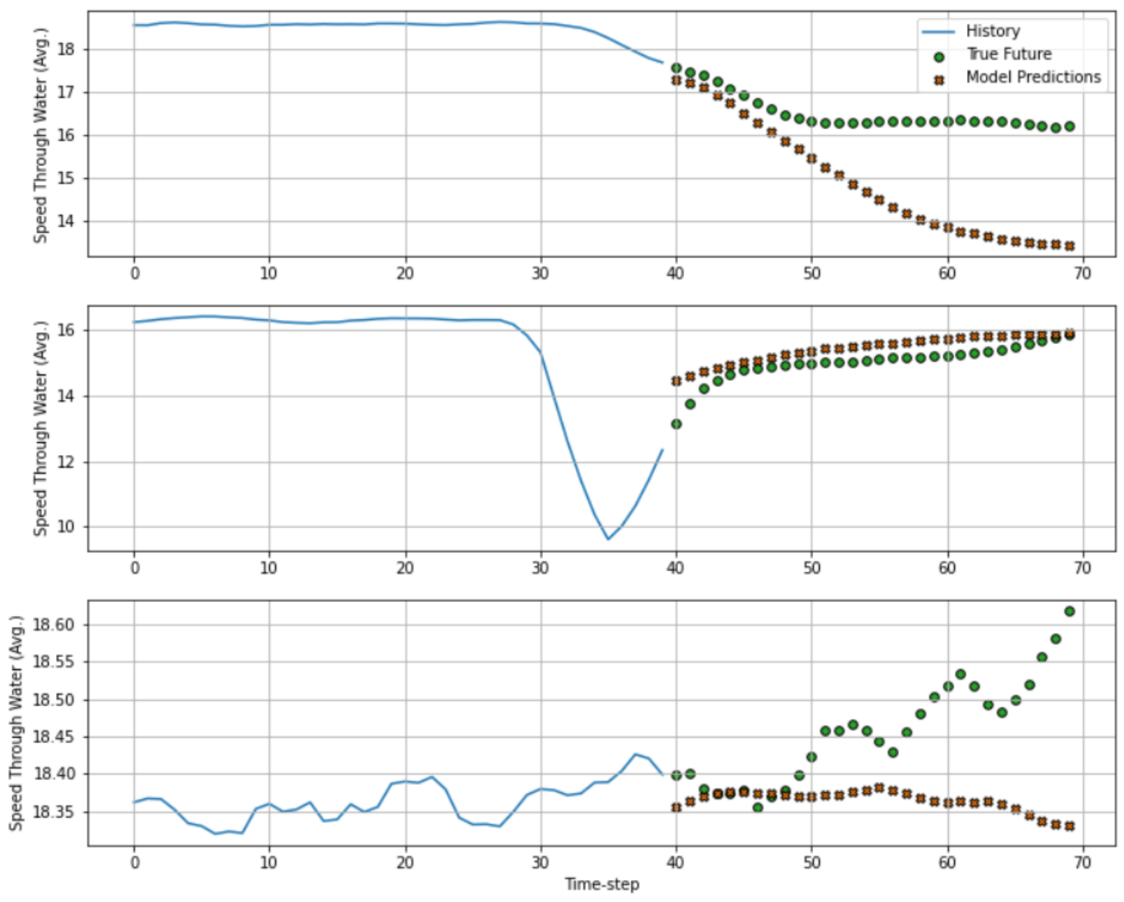


Figure 4.63: Indicative plots of the predictions using the Stacked LSTM network (40/30 configuration).

In comparison, the prediction mean absolute error is estimated as:

Table 4.9: Mean Absolute Percentage Error & Mean Absolute Error of 40/30 configuration

	MAPE %	MAE [kn]
Baseline	108.1	1.243
Linear	31.1	0.207
MLP	30.7	0.204
Convolution	29.6	0.292
LSTM	38.5	0.302
Stacked LSTM	34.0	0.270

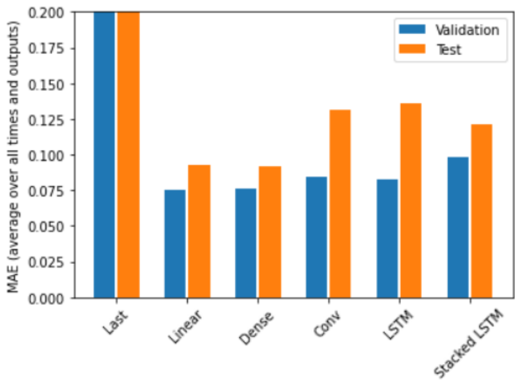


Figure 4.64: Performance comparison (40/30 configuration).

At the longest examined input window and prediction horizon the simpler configurations seem to handle predictions better. Sequence handling architectures display greater validation and testing error.

Chapter 5

Conclusions and Future Work

Conclusions

This thesis attempted to investigate the potential of utilizing neural networks in ship speed prediction. After several types and configurations of models were tested, sufficient accuracy and reliability were achieved from a practical point of view. All neural network architectures tested performed significantly better than the common sense baseline and managed to predict the vessel's speed with a mean absolute error under 0.5 kn. Based on the results, the more the prediction horizon extends to the future, prediction accuracy is decreased. On the other hand, providing more historical data as input did not seem to prove beneficial for the sequence handling networks - perhaps an outcome of limited computational power. Lastly, it should be highlighted that simple configurations performed notably well mainly because of their ability to follow the main underlying signal, despite only having access to the last input of the ship speed timeseries, while CNNs & LSTM networks that made more accurate local predictions, displayed better generalization characteristics but worse metrics perhaps due to their behaviour in the most sharply changing regions of the dataset.

Future Work

Based on the results, several suggestions can be made to further explore this research. The suggested future work is summarized in the next periods.

- The proposed model architectures should be trained and tested using more features, especially ones related to the environmental conditions, such as the significant wave height, wave period, the wind velocity and its direction relative to the ship, in accordance with other researchers' work. Significantly lower prediction error, below 10%, should be expected.

- A more thorough investigation of the possible network topologies and an automated, systematic hyperparameter tuning could yield better architectures and further investigate the models' behaviour on longer input and output windows.

- The possibility of employing autoregressive instead of single shot models could also be investigated. In autoregressive models the network decomposes the prediction into individual time steps and each model's output can be fed back into itself at each step, so that predictions can be made based on the previous one.

The behavior of the neural networks should, also, be tested as a part of a greater application, e.g. a weather routing system. A complete solution to the weather routing problem could combine information regarding the loading condition, its destination, the sea and weather state and help determine the optimal route.

Bibliography

[1] Masaru Tsujimoto and Hideo Orihara. Performance prediction of full-scale ship and analysis by means of on-board monitoring (part 1 ship performance prediction in actual seas). *Journal of Marine Science and Technology*, 24(1):16–33, 2019.

[2] Wengang Mao, Igor Rychlik, Jonas Wallin, and Gaute Storhaug. Statistical models for the speed prediction of a container ship. *Ocean engineering*, 126:152–162, 2016.

[3] Taizo Anan, Hiroyuki Higuchi, and Naoki Hamada. New artificial intelligence technology improving fuel efficiency and reducing co2 emissions of ships through use of operational big data. *Fujitsu Sci. Tech. J*, 53:23–28, 2017.

[4] Lúcia Moreira, Roberto Vettor, and Carlos Guedes Soares. Neural network approach for predicting ship speed and fuel consumption. *Journal of Marine Science and Engineering*, 9(2):119, 2021.

[5] Chollet Francois. *Deep Learning with Python, Second Edition.* Manning Publications.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[7] Aurélien Géron. *Hands-on machine learning with Scikit-Learn & TensorFlow: concepts, tools, and techniques to build intelligent systems.* OReilly Media, Inc., 2019.

[8] International Maritime Organization. Third IMO greenhouse gas study 2014. Technical report, 2014.

[9] Haiying Jia, Roar Adland, Vishnu Prakash, and Tristan Smith. Energy efficiency with the application of virtual arrival policy. *Transportation Research Part D: Transport and Environment*, 54:50–60, 2017.

[10] Roberto Vettor and C Guedes Soares. Development of a ship weather routing system. *Ocean Engineering*, 123:1–14, 2016.

[11] Miyeon Jeon, Yoojeong Noh, Yongwoo Shin, O-Kaung Lim, Inwon Lee, and Daeseung Cho. Prediction of ship fuel consumption by using an artificial neural network. *Journal of Mechanical Science and Technology*, 32(12):5785–5796, 2018.

[12] Chris Albon. *Machine learning with Python cookbook: practical solutions from preprocessing to deep learning.* OReilly®, 2018.

[13] J. Brownlee. *Introduction to Time Series Forecasting With Python: How to Prepare Data and Develop Models to Predict the Future.* Machine Learning Mastery, 2017.