Artificial Neural Network Based Empty Container Fleet Forecast

Joan Meseguer Llopis[1(✉)], Salvador Furio Prunonosa[1], Miguel Llop Chabrera[1], and Francisco J. Cubas Rodriguez[2]

[1] R&D Projects, Fundacion Valenciaport, Valencia, Spain
{jmeseguer,sfurio,mllop}@fundacion.valenciaport.com
[2] Cosco Shipping Lines Spain, SA, Valencia, Spain
javier.cubas@coscospain.com

Abstract. Global trade imbalances and poor, partial and unreliable information about available equipment make the coordination of empty containers a very challenging issue for shipping lines. The cancellation of transport operations once started or the extraordinary repositioning of containers are some of the problems faced by the local shipping agencies. In this paper, we selected the Artificial Neural Networks technique to predict the reception and withdrawal of empty containers in depots to forecast their future stock. To train the predictive models we used the different messages generated along the containers' shipment journey together with the temporal data related to these events. The evaluation of the models with the test dataset confirmed the possibility of using ANN to predict the number of empty containers in depots.

Keywords: Artificial Neural Networks · Empty Container Depots · Container Flow Prediction · Forecasting

1 Introduction

The management of the fleet of empty containers carried out by the local agencies of large shipping companies is an activity difficult to optimize. Among the different reasons, the level of uncertainty in the operations of delivery and reception of containers and the difficulty to predict them stands out [1]. This uncertainty leads to repetition of problems that cause extra costs and discontent of customers. Repeated lack of equipment in certain places (i.e. container depots) and times when attending export operations, and the extraordinary movements of repositioned equipment due to bad decisions are some of these problems. According to a study by the Boston Consulting Group the repositioning of empty containers costs the shipping industry between $15 and $20 billion a year. The same study estimates that 33% of these costs are derived from company inefficiencies and poor management decisions [2].

Nowadays, shipping agencies do not have systems to forecast the availability of empty containers. They keep track of the movements of containers and the ordered transport operations while maintaining an updated stock, which is contrasted with the

L. Rutkowski et al. (Eds.): ICAISC 2021, LNAI 12854, pp. 111–123, 2021.
https://doi.org/10.1007/978-3-030-87986-0_10

information received from empty container depots. Therefore, shipping agencies make decisions based on the current stock of containers and the experience of the equipment control department. However, they would make better decisions with the support of good predictions of the stock of containers using the information available.

With the help of artificial intelligence, this work aims to improve the effectiveness in short-term and mid-term planning operations of the Spanish agency of COSCO Shipping Lines, the world's third-largest shipping company in terms of container traffic. It is expected an improvement in the quality of service level to clients by avoiding situations of lack of availability of empty containers thanks to the quality of the forecasts.

The real-time prediction of the empty container delivery and demand in certain inland depots is a complex task. Therefore, previous work has been concentrated in optimizing the container transport operations by reducing the empty container movements [3] and in optimizing the repositioning of empty containers using various techniques such as evolutionary algorithms [4], policy-based decision making [5], and Multi-Agent Reinforcement Learning techniques [6].

In this paper, we selected Artificial Neural Networks (ANN) as it is used to cope with complex forecasting problems in applications with a high level of uncertainty that need to take into account several variables at a time. We wanted to verify if ANN models can be effectively used to make short/mid-term predictions of the delivery and withdrawal of empty containers from the warehouses. To train our ANN models, we used the date and time related parameters, and the different events generated along the containers' shipment journey as input variables. In this paper, we show the results for the most frequently used container type (i.e. 20 feet containers) by the local agencies. The same approach can be used for the other types.

The paper is structured as follows: in Sect. 2, the empty container management process is described and the different data sources are identified. In Sect. 3, we provide the Exploratory Data Analysis of the variables used to fit our models. The data preprocessing steps are presented in Sect. 4. In Sect. 5, we describe the ANN architecture of our solution and the method used to build our models. Results of the evaluation and its further discussion are presented in Sect. 6. Section 7 provides the main conclusions resulting from this work and proposes some guidelines for future work.

2 Empty Container Management Process

Generally, a container can have 3 states: empty, full and in maintenance. A depot is a warehouse used by the shipping and logistic companies to keep their empty containers until it is time for reloading of goods. In the process of export of goods, the empty container (EC) leaves a warehouse to be loaded at a client, transported to the port of origin, loaded on a vessel and transported to the destination country. In an import process, the loaded container arrives at the port of destination, is unloaded at the port terminal and is delivered to the carrier that takes it to the goods' destination. The emptied container is finally taken to the EC depot selected by the shipping agent.

The flow of ECs in a depot is generally determined by the delivery of ECs by the carriers to the depot (i.e. confirmed entries) and the delivery of ECs by the depot to the carriers (i.e. confirmed exits). These operations are also known as Acceptance and

Release of ECs. The number of confirmed entries and the number of confirmed exits per day are the target variables that we want to predict. These two variables will help the shipping agents to know the actual stock of ECs in the near future.

Two different transport orders lead to the entries and exits of ECs in a depot:

Empty Acceptance Order (EAO): it is the order generated for the delivery of the EC by the carrier to the depot in an import of goods (i.e. a new entry)
Empty Release Order (ERO): it is the order generated for the delivery of the EC by the depot to the truck driver in the export of goods. (i.e. a new exit)

Each of the above orders can have two states: generation and confirmed. The order generation serves as a statement of intent to inform the various agents involved in shipping operations that such EC acceptance or release will occur soon. The confirmation message is sent once the operation has been carried out. These events are:

Empty Acceptance Order Generation (EAOG): it is the event triggered when a new EAO is generated.
Empty Release Order Generation (EROG): it is the event triggered when a new ERO is generated.
Full Release Confirmation (FRC): it is the event generated when the port terminal delivers the full container to the carrier (i.e. a new EC about to enter at the depot).
Empty Acceptance Confirmation (EAC): it is the message generated in the moment when the carrier delivers the EC to the depot.
Empty Release Confirmation (ERC): it is the message generated in the moment when the EC is withdrawn from the warehouse by the truck driver.

A history of such events has been the main data source for our training and validation datasets. More precisely, EAOG and FRC events have been the source of data for the input variables to predict the quantity of EAC while the number of ERC was forecasted using the EROG events.

In addition to the above events, the temporal information associated with these events has also been taken into account as input variables for the models. We use the month, the day of the week and the day of the month as additional predictors.

All the above order and confirmation messages are exchanged between the shipping agent and the different transport actors (i.e. port terminals, carriers and depots) through its internal fleet management system. These messages are also stored in an internal database. The dataset used for the training and validation of our models is built from three years-long history of messages extracted from the database of Cosco's Spanish agency. A total of 163.441 messages generated over 1154 days has been used.

3 Exploratory Data Analysis

This section describes the Exploratory Data Analysis (EDA) process realized before the model training phase. The EDA refers to the important process of conducting initial research on the data to discover patterns, detect outliers, check correlations between

variables, and test assumptions. In this paper, we show the most relevant results due to lack of space.

Our EDA has been performed to the numeric variables: EAOG, EROG, FRC, EAC and ERC. These variables refer to the number of messages registered in a given day. The parameters EAOG, EROG, FRC are the predictors for our predictive models, whereas EAC and ERC are the target variables to predict.

The tools used in the EDA phase are: a) Pandas [7] Python library for the management and analysis of our different data structures; b) Seaborn [8] Python library for the statistical data visualization of our dataset; and c) Jupyter Notebook [9] as our web-based development environment to run the above libraries and to build and train our models in the following phase.

A summary of the descriptive statistics for all the numeric variables in the dataset is shown in Table 1.

Table 1. Input and Output variables statistics.

Statistics	EAOG	EROG	FRC	EAC	ERC
Max	87.0	71.0	53.0	53.0	83.0
Mean	8.01	7.32	7.99	7.99	7.31
Median	4.0	4.0	5.00	5.0	5.0
Std. dev	10.53	9.31	9.31	9.3	8.86
Skewness	1.98	1.99	1.43	1.42	2.05
Kurtosis	5.92	5.80	2.12	2.18	7.55

As we can observe from the above table, all variables have similar statistical values which indicate that they all follow a similar behavior. All variables present relatively high maximum values which may show the presence of outliers if we look at the values of their variance. These outliers can also be seen in Fig. 1. As can be seen in the boxplot on the left, all variables have a zero value as the minimum. This is normal as there are many days (especially the weekends) where no transport orders nor confirmations occur. The quantity per day of EAOG, EROG and ERC have a high value of kurtosis which indicates a higher degree of concentration around the mean and a sharper distribution of their observations [10]. A lower kurtosis in FRC and EAC variables shows a more normal distribution of their recorded quantities. As for the bias (i.e. skewness), the values between 1.5 and 2 indicate a slight bias towards the upper side of the mean, also visible in the boxplots' upper quartile of Fig. 1.

As it can be observed from the above boxplot, all variables have their whisker (i.e. 1.5 * [Q3 − Q1]) between 30 and 35. The number of EAOG per day has the biggest outliers. This could be also noted from its higher deviation from the rest.

Last but not least, it is worth analyzing the time lapse between the container's EAOG/EROG and its final EAC/ERC at the depot. For this purpose, the boxplot on the right-hand side of Fig. 1 is shown. As it can be noted, the orders usually take from 0

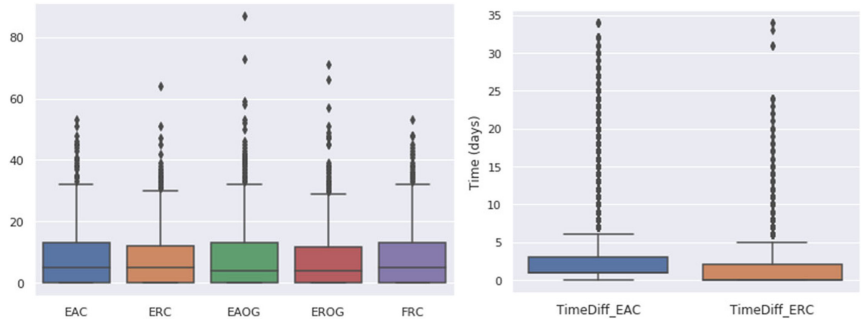

Fig. 1. Number of events per day (left) and time lapse between order and confirmation (right)

to 7 days to be completed. As it will be shown in Sect. 4, this is relevant when representing the input data for the prediction models.

Now we analyze the existing correlations between the different numeric variables considered. We run the *heatmap()* method of the Python Seaborn tool to obtain the correlation matrix for the entire dataset. As for the strength of the relationship, a value of ± 1 indicates a perfect degree of association between the two variables. And a value of 0 means a weak correlation between variables [11].

The matrix showed a correlation 0.63 between EAC and the EAOG. This correlation is between moderate and strong with a positive linear relationship as expected (the more orders the more acceptances). A correlation of 0.86 between confirmed acceptances and the number of full container releases (i.e. FRC) is also strong. Both correlations can be confirmed as their Pearson's p-values obtained are 1.75E-127 and 0.0 respectively. A value of 0.71 between ERC and EROG reflects a strong correlation between these two variables as well. This is also confirmed with a p-value of 2.0E-175. The p-values have been calculated using Python's Scipy library [12].

4 Data Preprocessing

To train the models with data of good quality a historical data preprocessing of the three years' messages from the database is required. As mentioned in Sect. 2.1, before the EC is delivered to the depot a EAO is issued first followed by a FRC. Similarly, before an EC withdrawal from the depot takes place, a ERO is generated. Each row in our database corresponds to a registry of timestamps for the events generated within each container import or export operation that took place along these 3 years of data. An extract example of this database is shown in Table 2.

Table 2. Transport operation database.

Op. Id	Imp./Exp	EAOG date	EROG date	FRC date	EAC date	ERC date
7739	Exp	–	11/05/19	–	–	12/05/19
7740	Imp	13/05/19	–	13/05/19	14/05/19	–

Moreover, the analysis done in Fig. 1 highlights the importance of considering the event to event elapsed time in the input data for the predictive models. To do this, we quantify the number of messages in each of the 7 days before the day when the prediction is made (from now on the PD day). These are the ECs that have not yet been delivered or withdrawn.

For the prediction of the quantity of EAC, we build two arrays as input data to the model. In Function 1, the steps to build these arrays are shown. The first one (EAOGs) is an array of length 7 where the first value corresponds to the number of EAOG issued the day before the prediction day (i.e. PD-1), the second value is the number of EAOG two days before the PD, and so forth until the last value. The last value of this array refers to the number of EAOG (and not confirmed yet) equal or older than 7 days before the PD (i.e. d $\leq$ PD-7). The second array (FRCs) of length 5, corresponds to the number of FRC messages for the days PD-1, PD-2, PD-3, PD-4 and d $\leq$ PD-5. As for the forecast of ERC, an array of length 7 (EROGs), with the number of EROG in the past days before the PD, is also computed. A new array is built for every day from the start date (StartDate) until the end date (EndDate) of the whole history of registries. At the end, it is obtained a matrix where each row represents a new observation in the training dataset. The arrays EAOGs, FRCs and EROGs are appended as new rows in the matrixes [EAOG], [FRC] and [EROG] respectively.

Function 1 Build Prediction Input and Output Arrays

```
Input: StartDate, EndDate
Output: [EAOG], [FRC], [EROG], [EAC], [ERC]
1:  let [EAOG], [FRC], [EROG], EAOGs, FRCs, EROGs = new Array []
2:  let [EAC], [ERC], EACs, ERCs = new Array []
3:  Let currentDay = StartDate
4:  while currentDay ≤ EndDate do
5:      for d in range(0,6):
6:          dayBack = currentDay - timedelta(days= d)
7:          let N_EAOG_orders = COUNT registries WHERE EAOG_Date = dayBack
            and EAC_Date > currentDay
8:          let N_EROG_orders = COUNT registries WHERE EROG_Date = dayBack
            and ERC_Date > currentDay
9:          EAOGs.append(N_EAOG_orders); EROGs.append(N_EROG_orders)
10:     for d in range(0,4):
11:         dayBack = currentDay - timedelta(days= d)
12:         let N_FRC_orders = COUNT registries WHERE FRC_Date = dayBack and
            EAC_Date > currentDay
13:         FRCs.append(N_FRC_orders)
14:     for d in range(0,6):
15:         dayForec = currentDay + timedelta(days= d)
16:         let N_conf_EAC = COUNT registries WHERE EAC_Date = dayForec
17:         let N_conf_ERC = COUNT registries WHERE ERC_Date = dayForec
18:         EACs.append(N_conf_EAC); ERCs.append(N_conf_ERC)
19:     [EAOG].append(EAOGs); [FRC].append(FRCs); [EROG].append(EROGs)
20:     [EAC].append(EACs); [ERC].append(ERCs)
21:     currentDay = currentDate + timedelta(days=1)
22: end while
23: return [EAOG], [FRC], [EROG], [EAC], [ERC]
```

In regards to the output data for the model's training dataset, a similar approach is followed. In this case, the third *for* loop of Function 1 collects the number of deliveries and withdrawals confirmed in the following 7 days from the prediction day (i.e. EACs and ERCs arrays). In each *while* loop iteration, each EACs and ERCs array is added to the matrixes [EAC] and [ERC] respectively.

Besides the above matrixes, another one ([Temp]) with the temporal parameters is also created. Each row in this temporal matrix contains an array with 3 elements: day of the month, day of the week and the month. From the above functions, these values are taken as follows: [*currentDay.Day*, *currentDay.DayWeek, currentDay.Month*].

Before proceeding to the models' training step, we first need to merge all above matrixes into a single dataset using Python's Numpy [13] and Pandas libraries (lines 2–4 in Function 2), split the whole dataset into a train set and a test set, and finally, normalize these data partitions. The code implemented to perform these steps is described in Function 2. This function is a reduced version of the one used which only shows the steps for the dataset to train the model that forecast the number of EACs. In lines 5–7 we remove the rows with outliers greater than the 99% percentile of the dataset's values.

To find out the best configuration parameters (a.k.a hyperparameters) that control the learning process of our models, the whole set of observations is split into a train set and

a test set. The former is used to initially fit the model while the latter is used to evaluate the predictions done by the fitted model with the true values from this partition. A good rule of thumb is to divide the whole dataset into 80% train set and 20% test set [14]. In our case, we made a random split of our observations (see line 8 in Function 2) using the Panda's *sample*() function of our dataset *class.*

The next step is to normalize the train and test input datasets for the model fitting process. The data normalization process is used to train models with homogeneous data and without outliers. This is a well-known procedure that considerably improves the performance of predictive models [15]. In our case, the min-max technique has been used (lines 12–13 in Function 2), which leverages the Pandas' DataFrame *describe()* function [7] that returns the *min* and *max* values for each column in our dataset.

Function 2 Get Train and Test Data

Input: [EAC], [EAOG], [FRC], [Temp]
Output: x_train, x_test, y_train, y_test

```
1:  let dataset = [EAC]
2:  dataset = column_stack((dataset, [EAOG]))
3:  dataset = column_stack((dataset, [FRC].))
4:  dataset = column_stack((dataset, [Temp]))
5:  q = dataset.quantile(0.99)
6:  dataset = dataset[dataset < q]
7:  dataset = dataset.dropna()
8:  train_dataset = dataset.sample(frac=0.80,ra dom_state=0)
9:  test_dataset = dataset.drop(train_dataset.index)
10: x_train, x_test, y_train, y_test =  split_to_x_and_y(train_dataset, test_da-
    taset)
11: train_stats = dataset.describe()
12: x_train = (x_train - train_stats['min']) / (train_stats['max']-train_stats['min'])
13: x_test = (x_test - train_stats['min']) / (train_stats['max']- train_stats['min'])
14: return x_train, x_test, y_train, y_test
```

5 Methodology: ANN for the EC Fleet Forecast

ANN basically performs like a human brain. A neural network is a dense parallel-distributed processor composed of simple computing nodes, also known as neurons [16]. This network is made of groups or layers of interconnected nodes. Each node is an artificial neuron and contains a function (i.e. activation function) that aggregates and correlates the weighted input parameters by weights (w_{ij}) into an output parameter that is transmitted to the next node. ANNs are trained by an optimization process (e.g. Gradient Descent Algorithm [17]) which is an iterative task aiming to find the value of the network weights that minimizes a loss function. This loss function is used to calculate the model's error. In regression problems, like the one described in this work, it is common to use

the mean squared error as the loss function (1).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\tilde{Y}_i - Y_i\right)^2 \qquad MAE = \frac{1}{n}\sum_{i=1}^{n}\left|\tilde{Y}_i - Y_i\right| \tag{1}$$

In our approach, we use a multi-layer neural network composed of an input layer (i.e. input data), N hidden layers and an output layer. In this work, we had to find the value of N and the number of neurons in each layer that provided a higher performance in the predictions. The output layer consists of 7 nodes; each one returns the number of delivered/withdrawn ECs for each of the following 7 days' prediction. As we aim to forecast numeric values greater than the unit, the Relu activation function is used [16]. This architecture is visually described in Fig. 2.

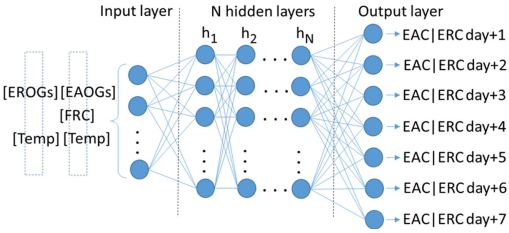

Fig. 2. ANN architecture for ECF forecast

The input layer is a concatenation of the normalized arrays [EAOG], [FRC] and [Temp] for the 7-day prediction, and [EROG] with [Temp] for the ERC forecast.

The architecture of Fig. 2, is built using Python's Tensorflow [17] and Keras [18] libraries. We made an iterative function (see Function 3) that automatically builds a NN following the architecture of Fig. 2. The input parameter "*layers*" is an array with N (i.e. the number of layers) values, each one referring to the number of nodes in each layer. The function *Sequential()* instantiates a model structure with a plain stack of layers. In steps 2 and 4, the *add()* function of the model class is used to add the layers with the number of nodes given by the *layers* array. Finally, the function *compile()* configures the model for training, which in this case we have indicated the MSE as loss function and the Mean Absolute Error (MAE) with the MSE as metrics for evaluating the model in the training and testing phases.

Function 3 Build NN model

Input: layers
Output: x_train, x_test, y_train, y_test

```
1:  model = keras.Sequential()
2:  model.add(layers.Dense(architecture[0],activation='relu',input_shape=[layers(0)]))
3:  for d in range(1, len[layers]):
4:      model.add(keras.layers.Dense(architecture[i],activation = 'relu'))
5:  model.compile(loss='mse', metrics=['mse','mae'])
6:  return model
```

6 Results: EC Fleet Forecast Evaluation

Now, the two neural networks to predict the number of EAC and ERC are trained with the datasets obtained from Function 2. As mention before, we had to train several models with a different number of layers and neurons in each layer that approximates us to a NN architecture that performs better in terms of MAE and MSE metrics. To do so, Function 3 was run for all the possible combinations of the possible values in the 3 hidden layers in Eq. (2). A total of 294 NN architectures was tried. As a heuristic widely adopted by the community, we also used powers of 2 number of nodes in each hidden layer.

$$\begin{aligned} layers &= [l_1,\ l_2,\ l_3] \\ l_1 &= \left(2^4,\ 2^5,\ 2^6,\ 2^7,\ 2^8,\ 2^9,\ 2^{10}\right) \\ l_2,\ l_3 &= \left(0, 2^4,\ 2^5,\ 2^6,\ 2^7,\ 2^8,\ 2^9,\ 2^{10}\right) \end{aligned} \tag{2}$$

To train the model, we used the *fit()* function of our model instance created in Function 3, which returns the MSE and MAE values for each iteration on the test data set. In each run, we used a learning rate decay [19] approach to progressively decrease the learning rate to avoid weights' oscillation and speed up the learning of our model. We fit our model using 500 iterations in each run. The results are shown in Fig. 3. As we can observe, the minimum values of MAE and MSE obtained for the EAC forecast are 2.2 and 16.6 respectively. These values correspond to the [1024,518,128,7] architecture of the NN. As for the ERC forecast (see Fig. 4), the found architecture is [1024,256,32,7] whose MSE and MAE are 3.04 and 38.8 respectively.

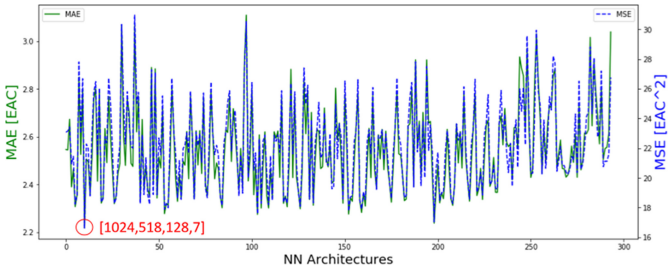

Fig. 3. EAC NN architecture selection

We have used the *predict()* function of our model to compute the predictions for the test dataset. The scatter plots of the true and predicted values for the EAC's model are shown in Fig. 5. In this figure, we can see that our model predicts reasonably well.

Also in Fig. 5 the scatter plots of the true and predicted values for the ERC's model are presented.

In this case, the plotted points on the first and second day are slightly farther from the true line than in the EAC forecast. One reason for this could be the lack of an event previous to the actual withdrawal of the container from the depot similar to the FRC event in the import operations.

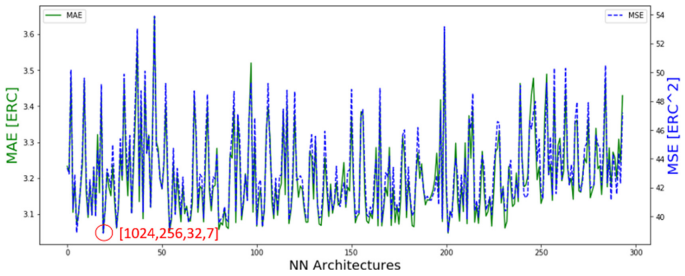

Fig. 4. ERC NN architecture selection

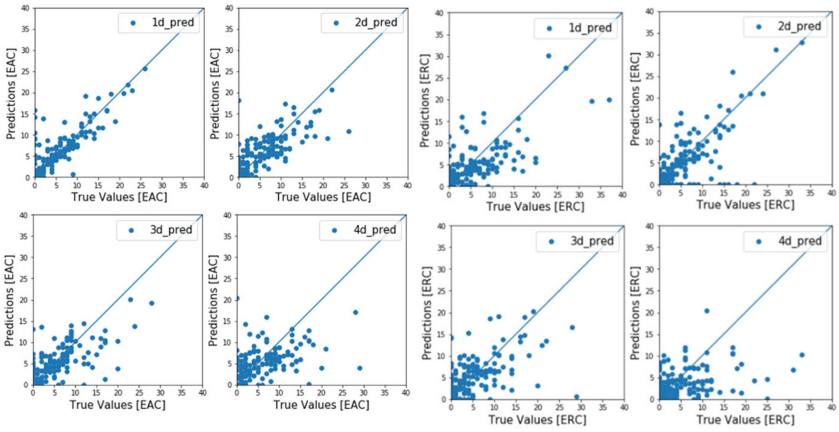

Fig. 5. Scatter plot for EAC and ERC 4-day predictions evaluation

To see the performance of our model for the 7 days of forecast we use the boxplot of the MAE for each forecast sample (see Fig. 6 Left and Right). As it can be observed, from days 3 to 6, the forecasted error is noticeably higher than the first 3 days. In Sect. 3.1, we saw that most of the transport orders (75% approx.) are carried out within 0 to 4 days. The rest of the orders introduces more uncertainty for longer-term predictions. Nevertheless, shipping agents still can make decisions based on the predictions of days 3–6 by aggregating the forecasted values.

The above results show the possibility to use our method to forecast the flow of ECs in depots. The future stock would be calculated by adding and subtracting these forecasted values to the actual amounts in each depot on the day when the forecast is computed. These stock predictions can then be effectively used to support shipping agents to make decisions in selecting the depot from which a EC should be withdrawn to attend an export operation as well as the depot to which the EC should be delivered to correct in advance a predicted lack of equipment. A direct consequence of this would be a reduction of the extraordinary displacements of trucks between depots. This, in turn, contributes to a CO2 reduction too.

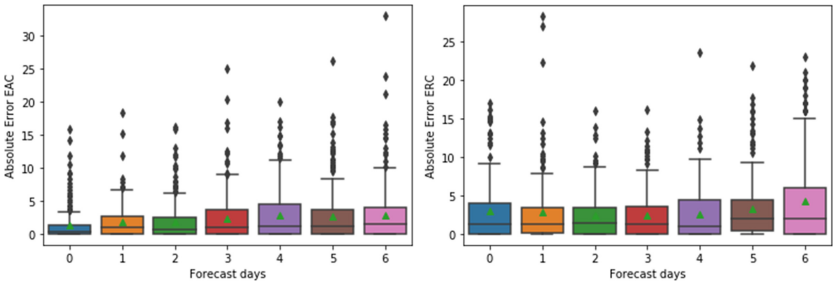

Fig. 6. Absolute 7-day EAC forecast error (left) and Absolute ERC forecast error (right)

7 Conclusion and Future Work

This paper presents a 7-day empty container delivery and withdrawal forecast approach for the empty container depots. In this work, we successfully demonstrated the possibility to forecast the stock of containers using Artificial Neural Networks. In the current scenario where freight transport is constantly increasing, our solution is able to support shipping agencies in taking greener and cost-effective decisions in the containers' release and pick-up operations, taking into account the forecasted stock in the different locations. In our test data, we are able to forecast EAC and ERC events for the next 7 days in an absolute error of 2.2 and 3.2 containers respectively. We have also checked the predictions' errors for each forecasted day. We observed that our approach makes more accurate predictions for the first three days whose mean absolute errors are 1.24, 1.79 and 1.86 containers respectively for the EAC. The error of our predictions increases for the following days as one could expect as the uncertainty also increases. Hence, future work could involve the usage of additional parameters such as customer class, weather variables or location of clients. These variables may affect the midterm forecast of the number of EAC and ERC. If these parameters will also be taken into account, the result of forecasting can be further improved. Other further work includes the usage of other techniques such as Recurrent Neural Networks [16] which is also used for time series prediction.

References

1. Sanchez-Rodrigues, V., Potter, A., Naim, M.M.: The impact of logistics uncertainty on sustainable transport operations. Int. J. Logist. Manage. (2010)
2. Sanders, U., Roeloffs, C., Schlingmeier, J., Riedl, J.: Think Outside Your Boxes: Solving the Global Container-Repositioning Puzzle. bgc.perspectives, The Boston Consulting Group, Inc. (2015)
3. Furio, S., Andres, C., Adenso-Diaz, B., Lozano, S.: Optimization of empty container movements using street-turn: application to Valencia hinterland Comput. Ind. Eng. **66**(4), 909–917 (2013)
4. Wong, E., Yeung, H., Lau, H.: Immunity-based hybrid evolutionary algorithm for multi-objective optimization in global container repositioning. Eng. Appl. Artif. Intell. **22**(6), 842–854 (2009)

5. Won Young, Y., Yu Mi, L., Yong Seok, C.: Optimal inventory control of empty containers in inland transportation system. Int. J. Prod. Econ. **133**(1), 451–457 (2011)
6. Luo, Q., Huang, X.: Multi-agent reinforcement learning for empty container repositioning. In: IEEE 9th International Conference on Software Engineering and Service Science. IEEE, Beijing (2018)
7. Python Data Analysis Library. https://pandas.pydata.org/docs/ (2020)
8. Seaborn v0.11.0. Statistical data visualization. https://seaborn.pydata.org (2020)
9. Jupyter Notebook. https://jupyter.org/documentation (2020)
10. Jain, V.K.: Data Science and Analytics (with Python, R and SPSS Programming). Khanna Publishing House (2018)
11. Phillips, D., Romano, F., Vo T. H., P., Czygan, M., Layton, R., Raschka, S.: Python: Real-World Data Science. Packt Publishing Ltd. (2016)
12. SciPy v1.5.3. Python-based ecosystem of open-source software for mathematics, science, and engineering. https://www.scipy.org/ (2020)
13. NumPy v1.19.0. The fundamental package for scientific computing with Python. https://numpy.org/doc/stable/ (2020)
14. EMC: Data Science & Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data. EMC Education Services (2015)
15. Nawi, N. M., Atomi, W. H., Rehman, M. Z.: The effect of data pre-processing on optimized training of artificial neural networks. In: 4th ICEEI Conference 2013. Selangor (2013)
16. Aggarwal, C.C.: Neural Networks and Deep Learning: A Textbook. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94463-0
17. TensorFlow v2.1.0. An end-to-end open source machine learning platform. Google Brain Team. https://www.tensorflow.org/versions/r2.1/api_docs/python/tf (2020)
18. Keras v2.3.1. Keras: The Python deep learning API. Google Brain Team. https://keras.io/getting_started/intro_to_keras_for_researchers (2020)
19. Smith, L.N.: A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. US Naval Research Laboratory Technical Report, Washington DC (2018)