Erasmus University Rotterdam
Erasmus School of Economics
Master Thesis Analytics and Operations Research in Logistics

Container Yard Planning Strategies Using (Convolutional) Neural Networks

E.M.C. van der Leeden (473936)

KONECRANES®

Supervisor:	R.S.H. Willemsen
Second assessor:	P.C. Bouman
Supervisors Konecranes:	C. Boer & S. van Dijk
Date final version:	30th November 2023

Abstract

Currently, all terminals base the container slot selection on penalty functions. However there are many patterns in the arrivals and departures of containers that provide learning possibilities for algorithms. In this paper we investigate how Artificial Intelligence can be used to improve the container slot selection in container terminals. We use a (convolutional) neural network and compare this with a simplified benchmark algorithm from Konecranes. We generate data to train the (convolutional) neural networks using a simulation, based on the benchmark algorithm. The same simulation is used to compare the performance of the algorithms. We use two scenarios for the simulation, one more simplified to test the abilities of the models, and a second that more closely resembles how terminals work in practice. We measure the performance of the methods as the average delay of the containers leaving the terminal. In the first simulation we find that the neural networks always perform worse than the benchmark algorithm. In the second simulation find that the convolutional neural network performs better than the benchmark in one of the cases. We also look at a convolutional neural network that is trained on extra data and see no performance difference compared to the original convolutional neural networks. We find that all (convolutional) neural networks perform better when predicting based on a short term delay compared to predicting delays longer ahead. Besides, we show that a good implementation of Artificial Intelligence has the potential to perform similar to or sometimes even better than the current benchmark implementation.

Contents

1 Introduction

During the 1980s, economies of scale facilitated the production of larger container ships which in turn led to more container traffic. This put more pressure on the container terminals. In 1980 the Panamax vessel, which was based on the size of the Panama canal, was introduced with a maximum capacity of 3,000 to 3,400 twenty-foot equivalent units (TEU), which is the standard measurement of container size. In 2019 the Megamax-24 vessel was introduced, which has a maximum capacity of 21,000 to 25,000 TEU (Rodrigue, 2020; Salido et al., 2012). These larger vessels can thus hold up to 8 times more than the older versions. The amount of containers per vessel is increasing, but the number of vessels is not decreasing. To be able to handle this increasing container load the terminals need to increase their productivity (Dekker et al., 2007; Saanen & Valkengoed, 2005; Wang & Meng, 2007; Zhen, 2014). Vessel operators often demand at least the same, or even higher service levels than before, such as service time. This increases the competition between different ports, and makes it even more important that the terminal can keep up its productivity. The application of modern technologies such as Artificial Intelligence (AI), cloud computing, and internet of things can help increase the productivity (Firooz et al., 2022).

To improve the productivity in a container terminal you can look at one of the three main parts: the quay cranes, the intra-terminal transport, and the container yard (Duinkerken et al., 2001). The quay cranes transport the containers between the container yard and the vessels that are berthing at the terminal. Within the terminal the containers are moved via intra-terminal transport to and from the container yard where the containers are stored. On the other side of the terminal the containers can arrive and leave via the gate. The container yard is thus the centre of the terminal that connects the other parts, and is often seen as the most important part of the container terminal (Guiliang & Lina, 2009; Tierney & Voß, 2016; Weerasinghe et al., 2023; A. Wong & Kozan, 2010).

Besides this central position, the container yard operations take up the most time in the terminal operations (Hirashima et al., 2005, 2006). Therefore, it is crucial to increase the efficiency and productivity in the container yard if you want to increase the efficiency and productivity in the entire terminal (Chen, 1999; Firooz et al., 2022; Guiliang & Lina, 2009).

One way to improve the productivity in the container yard is by applying a good algorithm for finding slots in the yard to place a container. These algorithms can often be improved with the use of Artificial Intelligence (AI) (Devanga et al., 2022; Fancello et al., 2011; Jin et al., 2023; Xie et al., 2022). The application of AI has advantages. For example, it requires less maintenance, because parameters no longer need to be manually updated, and it is continuously optimising. AI can also improve the situation by needing less employees and using equipment more efficiently, leading to less costs, and it can improve the quality of solutions (Sikorra et al., 2021). Besides, algorithms that incorporate AI aspects scale better to larger problems, can often provide better solutions, and can do this faster (Jin et al., 2023; Krizhevsky et al., 2017; Taddy, 2018).

In this paper we compare three algorithms for finding slots when placing a container in the yard. We first define a benchmark algorithm which is a penalty function. This is based on the algorithm as implemented by Konecranes, which is a competitive algorithm in the market.

The next algorithm is based on a regular neural network (NN), and the last is based on a convolutional neural network (CNN). We compare the algorithms by incorporating them within a simulation study.

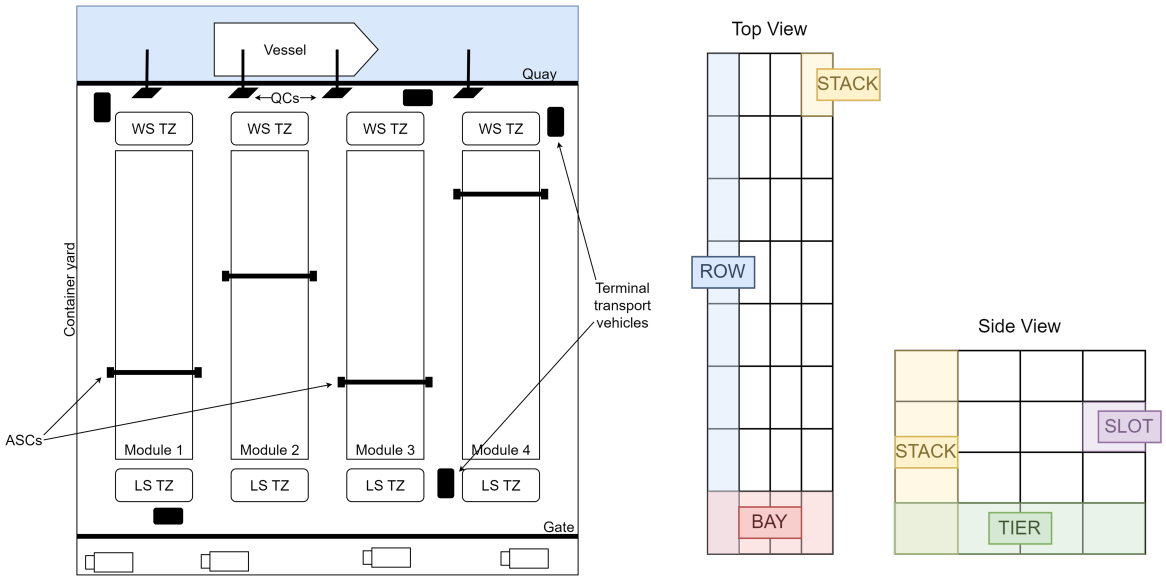
We find that for a more simple implementation of the simulation study the NNs are unable to provide results that are similar to or better than those provided by the benchmark algorithm. When looking at the more advanced simulation we find one CNN which is able to provide better results than the benchmark. Besides this, the CNNs are closer to the performance of the benchmark algorithm than the NNs were in the simple simulation. However, in most cases the benchmark algorithm still performs the best.

Applying AI to container yard problems is not yet implemented in practice and is therefore an interesting and innovative field of research. This paper, thus, provides an interesting contribution to the literature. We find that neural networks have potential to provide results at least similar to or even better than what is currently implemented. Our research thus provide a starting point for further research into the possibilities of neural networks, and show that a more detailed simulation study might improve the performance of the neural networks. Based on our findings Konecranes can already consider implementing a neural network for their yard slot selecting, however this neural network has to first be created and trained on actual data from a terminal. A neural network can also be used as part of the container slot selection process, instead of replacing the penalty function entirely.

The remainder of the paper is structured as follows. In Section 2 the problem and its assumptions are described in detail. In Section 3 we provide a more in depth overview of the previous research on this topic. The methods and approaches applied to the problem are mentioned in Section 4 and in Section 5 we provide the data for and the results from these methods. Lastly, we present a conclusion in Section 6.

2 Problem description

The container terminal consists of three main areas, namely the gate, the yard, and the quay, as presented in Figure 1a. In these areas we find the main types of logistics. There are the logistics on the waterside (WS) of the terminal along the quay. Here it has to be determined where and for how long the vessels berth. Besides this, decisions have to be made on the assignment of quay cranes that move the containers between the vessel and the quay. On the other side of the terminal there is the gate with its own logistical decisions. On this side the inland modalities arrive and leave. The inland modalities include trucks, rail, and barges. The last area is the yard, where the containers are stored. In this area logistic decisions have to be made on where to store the containers, how and when to rearrange the containers, and what equipment to assign to the movement of the containers (Anwar et al., 2019; Chen, 1999).



(a) Container terminal.

(b) Module.

Figure 1: Schematic overviews of container terminal components.

Containers leave the terminal either via the waterside or the landside (LS) (Chen, 1999; Weerasinghe et al., 2023). The container yard is split into several modules such that there is space for yard vehicles or stacking cranes to move throughout the yard (A. K.-s. Wong, 2008; Zhen, 2014). These are the horizontal transport vehicles that move containers to and from the modules, and the stacking cranes that move the containers within a module. An overview of a module, with the definitions of various measures that we use, based on the definitions from Konecranes, is shown in Figure 1b. The width and the length of a module determine how many rows and bays a module has respectively. This influences the amount of containers that fit in one module (Duinkerken et al., 2001). To define the dimensions of the yard we use the notation as shown in Table 1. The size of the yard is determined by the number of modules in the yard and the size of each module. The number of tiers defines how many containers can be stacked on top of each other.

Table 1: Notation for the dimensions of the yard.

Parameter	Explanation
m	Number of modules
r	Number of rows in a module
b	Number of bays in a module
h	Maximum number of tiers

One of the main components of a container terminal is the terminal operating system (TOS). This system controls many aspects of the terminal such as the vessel and yard planning, the equipment control, and the gate planning (Boer & Saanen, 2012). In this paper we focus on the improvement of the placement of containers in the yard given a specific layout. This yard planning strategy is also referred to as a stacking algorithm. This algorithm defines where the containers are placed in the container yard, but also how the containers are removed from the yard. It is important to do this efficiently since the containers have to be placed on the vessel in the correct order because they cannot be reshuffled once they are on the vessel (Dekker et al., 2007; Hirashima et al., 2005). Currently, no TOS implementation bases the container slot selection on more advanced methods such as Artificial Intelligence (AI), as it is always parameter based. Since the vessel arrivals follow a recurring schedule, meaning that the same vessels with similar content arrive on a weekly or monthly basis, there is the possibility to learn from this and implement this in an algorithm.

Assumptions

We base our model on the layout which is defined as the most important for this research by experts from Konecranes, due to verification purposes. This is a terminal which has rail mounted gantry (RMG) cranes in the yard, these are also referred to as automatic stacking cranes (ASCs). A terminal that uses RMG cranes tend to have a layout where the modules are placed perpendicular to the quay. There are many other options available for the cranes that are used in the yard, such as straddle carriers and rubber tired gantry cranes (Weerasinghe et al., 2023). Other options for the layout of the yard are also possible, such as modules parallel to the quay or other layouts based on the availability of space.

The layout of the terminal is shown in Figure 2. This figure shows how we assume the processes in the terminal to work and which parts we simplify. We assume that there are two modalities in the terminal, the vessels on the waterside and the trucks on the landside. We consider two possible routes through the terminal for containers. A route for export containers and a route for import containers. The export containers arrive at the terminal at the gate. Next, they are moved to the LS transfer zone (TZ). Then, they are placed in the module. When they leave the terminal they go from the module to the WS TZ, and then are placed on the vessel. We track the export containers until they are placed in the WS TZ. The import containers follow the same route as the export containers but in the opposite direction. The differences between the routes are that the import containers stop at the quay when they are removed from the vessel, and we track these until they are placed in the LS TZ.

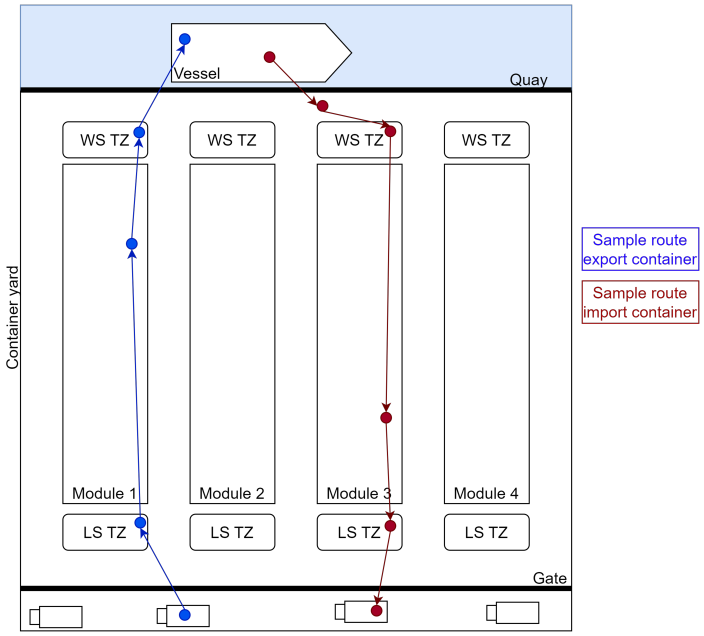


Figure 2: Terminal layout with sample paths for export and import containers.

We make the following assumptions on the arrival and departure processes of the containers. We assume the vessels arrive at the terminal using a recurring schedule. The vessels deviate from their expected arrival time according to a triangular distribution. According to experts from Konecranes this is representative of reality. A triangular distribution consists of three parameters, $Triangular(a, b, c)$, a represents the lower limit of the distribution, b the upper limit, and c the mode. This distribution then has the following probability density function,

$$f(x) = \begin{cases} \frac{2x}{c} & \text{if } a \leq x \leq c \\ \frac{2(1-x)}{(1-c)} & \text{if } c \leq x \leq b \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

The vessel arrivals and operations can take place during the whole day. The containers that need to leave with these vessels arrive before the expected arrival time of the vessel according to a triangular distribution. The arrivals at the gate only occur during working hours, which we assume to be the whole day, however in practice gate operations are often halted during the night. The containers that need to leave via the gate, stay in the yard according to a triangular distribution.

The common setup in practice is as follows. One module is serviced by two RMGs, one for the LS operations, and one for the WS operations. Each RMG can reach a predetermined number of rows on the corresponding side of the yard. These two cranes thus have an overlapping zone which they both can reach. However, using two RMGs like this does come with some extra restriction on their movement. Namely, (1) the RMGs are unable to cross each other, and (2) the containers can only be moved to the transfer zone on each of the sides by the corresponding crane. Thus, for this paper we start with a simpler implementation where there is one RMG that services the entire module. In this case the crane has more work but there are less restrictions.

Next, we implement a situation with two RMGs per module. In our case we let these RMGs cross, and they thus move over each other. This is a less common setup than one RMG on each side. In fact this is a unique setup that has only been implemented in the CTA terminal in Hamburg (van Valkengoed et al., 2004). However Saanen & Valkengoed (2005) find that the productiveness is similar as long as there is a similar level of demand on each side of the terminal.

Since the focus of this paper is on the yard we simplify the other operations in the terminal. We make the following assumptions on these aspects. First, the quay cranes (QCs) load and unload containers from the vessels in constant time. Second, we have unlinked interchange,this means that all terminal vehicles can operate completely independent of the other vehicles. We do not keep track of any of the other equipment in the terminal besides the RMGs. And lastly, since we do not keep track of any other terminal equipment besides RMGs, we assume that all movements that do not require RMGs or QCs can be performed immediately.

We assume two main simplifications in the container types. We assume that we only have import and export containers, and we only have container sizes 20ft and 40ft. To ensure proper placement we assume two restrictions on the placement of the containers. First, the slots in the yard are defined in TEU, thus 20' containers need one slot, and 40' containers need two consecutive slots in a row. Second, containers can only be placed directly on top of containers of the same size, or on empty ground slots.

We also define which container attributes are incorporated in the strategy. All container attributes that are incorporated in this paper with their corresponding possible values are presented in Table 2. Some of the attributes which are usually presented as a string of text are changed to a numeric value in our notation, for example $lineOperator$ and $outboundCarrierID$, this does not influence the performance of the algorithms. Using numeric values allows for easy extension of the model when more value options need to be added, and it allows for easy comparison between values. Some of the attributes are specific to import or export containers, for those attributes we set the value to 0 when the container does not have that transit direction.

Note that almost all container attributes have discrete values which represent the possible classes. The attribute 'edt', the estimated departure time, is the only one with a continuous value with an ordering. Besides, there are two attributes that can take on discrete values, but do have an ordering to them, these are 'weight' and 'length'. The 'weight' attribute has three classes from light to heavy. The actual weights of the categories are not relevant in this case, these can be defined by Konecranes themselves, as long as they have the light to heavy ordering. It should be defined such that a container belongs to a weight class if its weight is in between two boundaries. In an actual terminal the weight classes of the containers help define the order in which they are placed and removed from the yard, because the heaviest containers are loaded as low as possible on the vessels.

Table 2: Container attributes with their possible values and explanation.

Attribute	Values	Meaning
containerID	$\mathbb{N}$	Unique ID to identify each container.
inboundType	gate, vessel	The modality with which the container arrives at the terminal.
outboundType	gate, vessel	The modality with which the container leaves the terminal.
weight	1, 2, 3	The weight class of the container, with 1 the lightest, and 3 the heaviest.
lineOperator	$\mathbb{N}$	The owner/operator of the container.
nextDestination	A, B, C, D, E, F, G, H	The next destination of the container.
edt	$\mathbb{R}^+$	The time at which the container is expected to leave the terminal.
x	1,2,...,r	The row in which the container is currently located.
y	1,2,...,b	The bay in which the container is currently located.
z	1,2,...,h	The height at which the container is currently located.
module	1,2,...,m	The module in which the container is currently located.
transitDirection	import, export	The direction of the container through the terminal.
length	20, 40	The size of the container in ft.
outboundCarrierID	$\mathbb{N}$	The ID of the specific vessel that takes the export container, 0 if the container is import.
outboundServiceID	$\mathbb{N}$	The ID of the service line that takes the export container, 0 if the container is import.
outboundCarrierVisitID	$\mathbb{N}$	The ID of the actual visit of the vessel for an export container, 0 if the container is import.
inboundServiceID	$\mathbb{N}$	The ID of the service line that brought the import container, 0 if the container is export.
yardDirection	in, out	To track whether the container is moving (or just moved into) or out of the yard.

3 Literature review

A yard planning strategy is dependent on two main aspects, the amount of attributes that are considered when differentiating between the containers, and the objective that one tries to achieve with this strategy. Given the chosen information for these aspects a strategy can be defined. Next, an algorithm can be created to implement this strategy. Lastly, these algorithms and strategies are often compared in literature using a simulation study.

3.1 Container differentiation

Determining where to place a container is mainly based on the information that is available on this container and the other containers in the yard. The most important aspects of this information are the type, the size, the destination, and the weight of the containers. However, other aspects such as modality, shipping line, vessel, and customs information can also be used to determine the position of the container (Dekker et al., 2007; Firooz et al., 2022; K. H. Kim & Lee, 2006; Lee & Hsu, 2007).

The first aspect is the type of container, this refers to the type of cargo that it stores. The most basic type are the dry storage containers. Other types that have special requirements for placement are refrigerated containers (reefers), which need to be stored near an electricity outlet, and containers storing dangerous goods, which need to be placed at a certain distance from each other and certain other containers for safety reasons. There are many other types such as out of gauge, flat rack, open top, empty, and more (Firooz et al., 2022).

The second aspect is the container size. The most common container sizes are 20ft, 40ft, and 45ft. There are also other sizes available such as 10ft and 60ft, however these are much less common (Firooz et al., 2022). The ground slots in the yard are measured in twenty-foot equivalent units (TEU). Thus 20ft containers need one TEU and 40ft containers need two TEUs.

The third important aspect of a container is the destination of the container. This can be defined in detail by the port of discharge, which is the destination port of the container. However, it can also be defined roughly by how the container leaves the terminal, which is the transit direction of a container. For this second type of definition there are four main options, import, export, transit, and transhipment. Import containers arrive via vessel and leave the terminal via inland modalities such as rail, truck or barge. Export containers arrive via inland modalities and leave the terminal via deep sea or feeder vessel. If a container arrives and leaves via feeder or deep sea vessel it is called a transhipment container. Lastly, there are transit containers. These are moved inland across international borders (Firooz et al., 2022). Other destination types such as storage, restow, and through can also be defined.

A last component that we consider is the weight of the container. In general there is a limit to how much weight can be placed on a container, and thus it is not ideal to place heavier containers on top of lighter containers. The heaviest containers usually need to be loaded on the vessel before the lighter containers because these need to be placed on the bottom of the vessel. Due to this loading order the weight or weight class of a container is an important aspect for a stacking algorithm (A. K.-s. Wong, 2008).

Besides all the information on the container as mentioned above, a strategy also needs

information on the current situation in the yard. The current situation in the yard can, for example, contain information on all the containers currently in the yard, on how busy the transfer zones are, and on the current locations of the RMGs.

3.2 Objectives

There are various objectives that a stacking strategy can focus on. The choice of objective influences the placement and spread of containers in the yard. For example, some objectives may require to use many ground slots as possible, while others may stack the containers as high as possible.

Moving a container to a different slot after it has been placed in the yard is referred to as (re)shuffling or remarshalling. This is a move that is seen as unproductive since it does not add to the productiveness of the cranes. Productive crane moves are moves that are absolutely necessary, and thus consist of moves to and from the transfer zones. One of the objectives seen in yard planning is to minimise these remarshalling movements (K. H. Kim & Kim, 1999; Lee & Hsu, 2007; Salido et al., 2009).

Objectives can also be based on the usage of the equipment. This can be defined as the spread of the workload across the quay or yard cranes, or the utilisation of the cranes (Dekker et al., 2007; Duinkerken et al., 2001; K. H. Kim & Lee, 2006; Zhen, 2014). It can also be measured as the quay crane capacity and the yard throughput capacity. These are the number containers in TEU that can be handled over the quay, and the number of containers that the yard can handle and store in TEU per year respectively.

There are two other common objectives in literature when optimising the yard allocation or remarshalling processes. The first is to have a high density in the yard, and thus use as much as possible space in the yard. This is mainly important in terminals where the yard is relatively small (Sikorra et al., 2021). The second is based on minimising the time spent in the yard. This can be defined as the time needed to retrieve a container, or the time needed to handle a container in general (Dekker et al., 2007; Duinkerken et al., 2001; K. H. Kim & Lee, 2006; A. K.-s. Wong, 2008).

3.3 Yard planning strategies

In general, a classification into three stacking strategies can be made. These are preassigned stacks, dump-and-sort, and a dynamic strategy. Each of these strategies has their own advantages and disadvantages, an overview of which is presented in Table 3.

When using preassigned stacks, there are designated sections of the yard reserved for the containers that belong to a certain vessel. This is the same as the category stacking strategy as mentioned by Dekker et al. (2007). Some advantages of this strategy are that it is easy to execute and that it is possible to decrease the driving distances for the yard equipment. However, this strategy is not flexible and may require a lot of housekeeping moves to make the block empty for other vessels.

The second strategy is dump-and-sort. In this strategy the containers are placed in a dump stack when they arrive, and are later moved to a better location. This method is easy to implement during discharge operations, and allows the discharge operation to go fast. Disadvantages

Table 3: An overview of the advantages and disadvantages of the three main classes of strategies.

Strategy	Advantages	Disadvantages
Preassigned stacks	Easy to execute. Possible to decrease driving distances.	Not flexible. May require a lot of housekeeping moves.
Dump-and-sort	Easy to implement during discharge operations. Fast discharge operations.	Requires many housekeeping moves. Requires quite some planning
Dynamic	Good use of entire yard. High flexibility. Can avoid housekeeping moves.	More effort to implement. May require more equipment. May require larger travel times in the yard.

of this method are that it requires many housekeeping moves to move the containers to their correct location, and it requires quite some planning. A strategy that is similar to this is mentioned by Chen (1999), who calls it premarshalling.

The last strategy is a dynamic strategy. Here, the slots in the yard are chosen based on the container aspects and the situation in the yard. Then the container is moved to the best slot. What makes a slot the best compared to the others can be determined separately for each implementation. This method has many advantages, it can make good use of the entire yard and spread the load, it has a high flexibility, and it can avoid housekeeping moves. There are two main disadvantages of this method. The first is that due to the location selection in the yard there may be larger travel times to the slots in the yard, which may also require more equipment to perform all the moves. The second disadvantage is that it requires more effort to implement for the people in the yard and it needs more advanced software. Chen (1999) refers to this dynamic strategy as the sort-and-store strategy.

Some papers define their own split in stacking strategies. Dekker et al. (2007) and Sikorra et al. (2021) define the distinction between category and residence time stacking. In the first the containers are assigned to categories and containers from the same category are stacked on top of each other. In the second a container is stacked on top of another when the departure time of that container is earlier than all the containers below it. Steenken et al. (2004) define storage planning and scattered stacking. Storage planning is similar to the preassigned stacks strategy. In the scattered stacking method the containers are stochastically spread through a specific part of the yard based on the berthing place of the vessel.

Duinkerken et al. (2001) split the assignment of a container to a slot in two steps. In the first step, called the stacking method, they find the row for the container. In the second step, the stacking strategy, they find the best slot within the selected row. They mention two options for the stacking method, selecting a row randomly, and selecting the row based on the quay crane. They mention four options for the stacking strategy: random, levelling, closest position, and maximum remaining stack capacity.

There are some general aspects that planning strategies try to implement, because they are known to be effective. First, the two main transit directions of containers each have a preferred stacking height. For the export containers it is efficient to stack them as high as possible. This is because these containers tend to have information available on their departure. For the import containers holds the opposite. You want to stack these as low as possible because their

departure from the yard can be unpredictable (Chen, 1999). This is because trucks do not have predetermined arrival times at the terminal, and it is not known exactly which container they will pick up. The empty containers and the reefers also have a preferred stacking height. The empty containers can be stacked very high because it usually does not matter exactly which one is selected as long as they belong to the same company. The reefers cannot be stacked very high because they need to be accessible for checks and they need access to power.

Another basic aspect is about the side of the yard in which the containers are placed. It seems to be effective to place export container close to the waterside and import containers close to the landside because these are the sides on which these containers leave the yard. Besides this, there are often specific stacks available for the reefers, and specific stacks available for the empty containers.

3.4 Algorithms

To improve the placement of the containers in the yard we need to know more than the strategy on which we base the placement of containers. We want to choose an algorithm that can implement a strategy. In general, the algorithms can be split into 'classic' operations research (OR) algorithms and Artificial Intelligence (AI) algorithms. OR is a scientific method for decision making, where it is sought to find the best way to operate a system, usually under circumstances of scarce resources (Winston, 2004). Some examples of these algorithms are genetic algorithm (GA) and ant colony optimisation (ACO). AI is based on the opportunities of computers, and often does not contain explicit instructions, some examples of these methods are Q-learning and neural networks (NNs).

There are advantages to applying AI to problems. Some of these advantages are less maintenance, improved quality due to continuous optimisation, the possibility to save costs, and the possible need for less employees (Sikorra et al., 2021). Many papers have therefore looked into the application of AI to container terminal processes (B. Kim et al., 2020; Salido et al., 2009; Sikorra et al., 2021; Wang & Meng, 2007).

AI can be defined as enabling systems to mimic human behaviour. The idea is to artificially implement techniques in a system such that it can learn from and interpret data in the same way as humans do (Anwar et al., 2019; Firooz et al., 2022; Taddy, 2018). In general some main approaches to AI are machine learning and more specifically deep learning. Anwar et al. (2019) mention some possible AI methods that can be applied to container terminals, these are multi-agent system (MAS), (artificial) neural network (NN), decision support system (DSS), search algorithms, and automation.

3.4.1 Operations research algorithms

operations research (OR) algorithms are algorithms that are commonly applied in the context of OR. Examples of these for container terminals are genetic algorithm (GA) and ant colony optimisation (ACO) (Weerasinghe et al., 2023; A. K.-s. Wong, 2008).

Genetic Algorithm

The idea of GA is based on natural selection as is seen in nature. First, create a population

of entities, which represent solutions, where there is some diversity between the chromosomes, which define the values of attributes for an entity. Second, select a set of parents from this population based on a certain fitness value. Next, create children who are a mixture of the chromosomes of the parents, and mutate the chromosomes of the children in some way. Lastly, continue the second and third step with this new population until a certain stopping criterion is achieved, for example when the change in optimal solution value is less than ϵ. Guiliang & Lina (2009) create a method that combines a GA with simulation. They create a simulation model, and then use the output of this model to optimise their algorithm. Then they run the simulation again. They continue this process until a stopping criterion is reached. They specifically apply this technique to the problem of optimally allocating storage space in container freight stations. They find that this approach improves the system in many aspects, such as decreasing errors and improved operation efficiency.

Ant Colony Optimisation

ACO is based on the behaviour of ants. The idea is to find the best path along certain nodes from a starting to an end point. The agents (based on the ants) walk along a path and leave pheromones on this path. The other agents will then select the path with the most pheromones. Wang & Meng (2007) provide an algorithm that is based on both GA and ACO. They use a GA to find the best resource allocation in a container terminal. Next, they use ACO to optimise the scheduling plan based on the allocation plan that was found with GA. They test their algorithm on a tugboat allocation problem and find that it can improve the tugboat utilization rate.

3.4.2 Machine learning algorithms

Machine learning can be split in three main categories: supervised learning, unsupervised learning, and reinforcement learning (RL) (Belsare et al., 2022; Morales & Escalante, 2022). In supervised learning the outcome of the problem is known and we try to find how the output can be determined based on a certain input (B. Kim et al., 2020). In unsupervised learning the output is not known and we try to find patterns in the data (Hastie et al., 2009). In RL an objective is optimised where the actions that should be taken are learned via interactions with the environment (Morales & Escalante, 2022). The most relevant for our problem are supervised and reinforcement learning.

There are advantages and disadvantages to both reinforcement and supervised learning. An RL method allows for constant updating, this means that the algorithm will be based on the most recent data set which might provide better predictions. However, this also means that the algorithm constantly has to run and that it will be influenced by abnormalities in the data set. A supervised learning algorithm may need to be updated by manually adding new data every time period, however this will also provide a more robust solution against abnormalities since these can easily be excluded from the data set.

Reinforcement learning

RL is the most popular method in the context of container terminals. The technique for RL can be described with the following steps. First the environment is observed, then a certain strategy

is used to undertake an action. After performing this action a reward or penalty is observed, and based on this outcome the decision making strategy is adjusted. This process is continued until a strategy is achieved that satisfies the given stopping criteria (Belsare et al., 2022; Fechter et al., 2019; B. Kim et al., 2020; Xie et al., 2022).

RL does not need any information on the environment at first, and therefore it is appropriate to apply to a variety of problems for which the environment is unknown but where learning is possible (Fechter et al., 2019). RL techniques work especially well for problems that can be defined as a Markov decision process (MDP) (Devanga et al., 2022; Fechter et al., 2019). In an MDP you move between states with certain probabilities, which are based on the current state and the action chosen in that state. Sikorra et al. (2021) and Xie et al. (2022) agree that RL would be an appropriate method for stacking problems. Sikorra et al. (2021) tested their method on slot allocation in a container terminal and find that it learns well for a small storage block.

The most common method to train an RL algorithm is by using simulation. These types of algorithms need a lot of data to make good estimations. In general an original data set can be used, but because it is much cheaper and faster to simulate situations and create data this way, simulations are more popular (Belsare et al., 2022; Devanga et al., 2022; Taddy, 2018; Xie et al., 2022).

Q-learning

A specific technique derived from reinforcement learning is Q-learning (QL), this is a model-free technique that can learn rewards on its own (Devanga et al., 2022). When applying QL, a Q-table is defined which stores the Q-values. A Q-value is the expected sum of future rewards when a certain action is performed in a certain state, they are defined for every combination of state and action (Devanga et al., 2022; Sikorra et al., 2021). For example, for Hirashima et al. (2005) this represents the value for each combination of a layout and the movement of a container. The function that determines this sum of future rewards is called a Q-function.

The main advantage of using QL is that it is an effective learning method when applied to an unknown environment (Hirashima et al., 2005, 2006). The main disadvantage of the method is the amount of storage that is required. The amount of values that need to be stored in the Q-table can become increasingly large, making the method not applicable to all types of problems (Hirashima et al., 2005; Sikorra et al., 2021).

Various papers have provided an implementation of a QL algorithm for the container yard. Sikorra et al. (2021) define a method for container slot allocation in the yard which is based on simulation and a deep Q-network. Hirashima et al. (2005) and Hirashima et al. (2006) provide a Q-learning algorithm for placing containers on their desired positions in the yard. They split the process in two steps. In the first step the desired position of a container is determined, and in the second step the steps needed to make this position available are determined. For each of these steps they define separate Q-tables to which the other step can reference when needed. Hirashima et al. (2005) and Hirashima et al. (2006) both find that this method is able to learn well and provide better results than conventional methods.

Some researchers looked into improving QL algorithms. Since the main problem with this

method is the storage space, Hirashima et al. (2005) provide an improvement to decrease this. They adjust the method such that there is a limit to how long Q-values should be stored. Hirashima et al. (2006) provide a different solution to this problem by only storing Q-values that are referred to during the learning process. Besides this, they also define a binary tree method to store the Q-values more efficiently. Both Hirashima et al. (2005) and Hirashima et al. (2006) find that QL with these adjustments can be applied to larger problems and provide better solutions for these larger problems.

Neural Network

Neural networks (NNs) are a machine learning technique that learns from data in a similar way as the human brain. It is a part of deep learning that uses nodes and a deeply layered structure to achieve this. The key to the effectiveness of NNs is the recursive combination of linear combinations and non-linear transformations (Srivastava et al., 2014; Taddy, 2018).

The main advantage of a NN is that it is known to produce universal function approximations, which means that for any continuous function it can provide a function approximation that is accurate up to some level ϵ (DeVore et al., 2021; Srivastava et al., 2014). Besides this, NNs are also known to perform well under circumstances, such as problems with a lot of dimensions (DeVore et al., 2021; Krizhevsky et al., 2017), problems with non-convex objective functions (Kingma & Ba, 2014), and problems with a lot of noise (Hastie et al., 2009; Krizhevsky et al., 2017). Lastly, NNs have the advantage that they are not very dependent on human experience, and can solve problems at high speeds (Jin et al., 2023).

Because a NN can approximate nearly every function to any desired level of accuracy, it is important to prevent overfitting (Srivastava et al., 2014; Taddy, 2018). This occurs when the model starts to fit the training data too well and is therefore no longer able to generalise to other data (Hastie et al., 2009). Three options to prevent overfitting are: early stopping, which uses a validation set to check when the model starts to overfit; regularisation, which adds a penalty to the loss function; and dropout, which randomly drops a subset of nodes (Srivastava et al., 2014; Taddy, 2018).

NNs can also be used in other algorithms. For example, deep learning is based on NNs, these algorithms have a very large structure, and specifically contain a lot of hidden layers. NNs can also be applied in RL settings (Belsare et al., 2022). An example of this is a double deep Q-network. In this method the Q-learning function is approximated by NNs (Devanga et al., 2022; Taddy, 2018).

3.5 Simulation models for container yards

One of the most popular techniques to validate the effectiveness and correctness of an algorithm for a container terminal is by using a simulation (Anwar et al., 2019). For example, Fechter et al. (2019) test their stacking problem algorithm using a simulation based on the SimSharp framework, and Hirashima et al. (2006) provide a Q-learning algorithm for marshalling plans that they test using simulation. Duinkerken et al. (2001) provide a complete simulation model for the terminal that integrates multiple aspects of a container terminal. Xie et al. (2022) also use a simulation based on discrete event simulation (DES) to run their experiments to compare

various NNs.

In some cases the optimisation process is combined with the simulation process, also known as simulation-optimisation. An example is provided by Guiliang & Lina (2009), who test their optimisation algorithm based on GA and simulation using the platform Arena. Furthermore, Devanga et al. (2022) combine a simulation with their RL algorithm.

Overall, the validation of algorithms for container terminals commonly involves simulation techniques. Where some studies also integrate the optimization processes, showing the versatility of using simulations.

4 Methodology

Containers have a specific time at which they need to be picked up and leave the yard. We optimize the time it takes to retrieve the containers. Thus, our main objective is to minimize the delay experienced by containers when they are picked up from the yard, this was defined as the most relevant for Konecranes by their experts.

To tackle this problem we execute a quantitative research based on the following steps. We set up three algorithms, namely a benchmark algorithm, a regular neural network (NN), and a convolutional neural network (CNN). Next, we implement a simulation, which we use in combination with the benchmark algorithm to generate data for the NNs. Lastly, we also use this simulation to compare the algorithms.

4.1 Benchmark algorithm

To compare whether our algorithm produces better results we define a benchmark algorithm, which is based on a penalty function from Konecranes. In our research we use a simplified version of this function which is based on the same concepts. The penalty function $f(\cdot)$ defines the value of each slot, which is equal to the sum of penalty values corresponding to a set of relevant attributes. Using Algorithm 1 we can then find the best slot in a module. The idea is to go over all slots and select the slot that has lowest penalty value. Note that we only check the slots in the yard where a container can be placed, thus it has to be placed on a ground slot or on top of another container. We find the best module for a container using Algorithm 2.

Algorithm 1 Finding best slot in a specific module using benchmark algorithm.

Define $z^* = \infty$: lowest penalty value, and $x^* = 0$: best slot.
Define S: set of available slots.
Define c: the container that we try to place.
while $S \neq \emptyset$ **do**
    Select $x \in S$.
    Compute $z = f(x, c)$.
    if $z < z^*$ **then**
        Update best solution, $z^* = z$ and $x^* = x$.
    end if
    Update set S, $S = S \setminus \{x\}$.
end while
Return x^*.

4.2 Machine learning algorithms

We base our algorithm on a supervised learning method. We use this because these algorithms are easily applicable in container terminals and there is more control over which data is used in the determination of the method. Specifically we use neural networks (NNs). We use these because they are able to approximate nearly any function, and we have no information on what the function for our problems looks like.

Algorithm 2 Finding best module using benchmark algorithm.

Define $z^* = \infty$: lowest penalty value, and $m^* = 0$: best module.
Define M: set of all modules.
while $M \neq \emptyset$ **do**
    Select $m \in M$.
    Compute z, the penalty value of the best slot, using Algorithm 1.
    if $z < z^*$ **then**
        Update best solution, $z^* = z$ and $m^* = m$.
    end if
    Update set M, $M = M \setminus \{m\}$.
end while
Return m^*.

4.2.1 Input data

The input data for the NN contains information on the status of the yard after a container has just been placed. The values that the NN tries to predict are based on the delay of the containers. Note that we input the data per module, as we assume that all modules operate identically.

The exact data that is used as input for the NN is based on stack information, which contain values for attributes for each stack in the yard. We thus do not present individual container information to the model, since this can create data sets of extreme sizes. We only use attributes that contain information that cannot be retrieved from other attributes, and that provide container specific information. We also do not use attributes that are not reliable in practice according.

Because the NN needs numeric input variables, and because we want to ensure that there is no specific ordering in the categorical variables we use one-hot encoding on the categorical attributes. One-hot encoding transforms the value into an array of binary values, where at most one column contains a 1, and the others contain a 0 (Seger, 2018).

The delay of a container is based on the time it is expected to leave the yard compared to the actual time it leaves the yard. A performance measure can be defined as the effect a placement has on the delay of the container that has just been placed, the delay of all containers, or the delay of any other subset of containers. We define the performance measure as the average delay of the containers that leave the module in which the container is placed in the λ hour(s) after the container is placed because we want to minimise the delay of all containers in the yard, and the various stacks may influence each other due to reshuffles. We look at $\lambda = 1, 24$, and 168 (one hour, one day, and one week respectively). The results may change in steps, which may be interesting for other values of λ, however we do not look at those in this paper. All data for the NN is generated using a simulation model where the decisions are made based on the benchmark algorithm. The idea is that the NN learns which actions from the benchmark algorithm are effective.

4.2.2 Neural network

A feed forward neural network (NN), a NN with information flow in one direction and no cycles, with a single hidden layer can be represented by,

$$\hat{f}(x^*) = \beta_0^{(2)} + \sum_{j=1}^{M} \beta_j^{(2)} h(a_j(x^*)). \tag{2}$$

In this function the numerical input of the model is represented by x^*, the $\beta^{(2)}$'s are the weights, and the number of nodes in the hidden layer is represented by M. Next, we have the activation function defined by $h(\cdot)$, and the activation defined by,

$$a_j(x^*) = \beta_{j0}^{(1)} + \sum_{i=1}^{p} \beta_{ji}^{(1)} x_i^*. \tag{3}$$

Here p is the number of inputs, and the $\beta^{(1)}$'s are the weights that transform the input data. This shows that a neural network is based on taking a weighted sum of inputs and transforming this using an activation function.

If the model has more hidden layers this same principle can be applied where the output of the nodes from the previous layers, as defined by $\hat{f}(\cdot)$, are the input for the next layer. A transformation of the weighted sum of these inputs then provides the output of this next layer. This is equal to applying (2) with its own new weights, with the outputs of the previous layers as x^*, which could have a different number of nodes M (DeVore et al., 2021; Hastie et al., 2009).

We have to decide on how many hidden layers, L, and what those look like. Each layer is defined by its number of hidden units, M, and the activation function, $h(x)$. Some examples of activation functions are the identity function, sigmoid function, tanh function, and the rectified linear unit (ReLU) function, which are shown in (4), (5), (6), and (7) respectively (DeVore et al., 2021; Hastie et al., 2009; Zhang et al., 2023).

$$h(x) = x \tag{4}$$

$$h(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

$$h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{6}$$

$$h(x) = \max(0, x) \tag{7}$$

To define the hyperparameters we use the tuner as integrated in Keras (O'Malley et al., 2019), with the tuning options as in Table 4. Note that the number of nodes and the activation function can be different for each layer. The final layer is the output layer, in which we want to generate the output to the model. We have a regression type model and, thus, want a single numerical output. We use the recommended activation function for this layer which is the identity activation function as in (4) (DeVore et al., 2021; Hastie et al., 2009; Jin et al., 2023).

The tuner is used to search through the hyper parameter space to find the best values (Chollet et al., 2015). First, we select the random search tuner, this allows the tuner to randomly go

Table 4: Hyperparameter tuning options for the whole model and per layer i.

Hyperparameter	Values
N	4, 5, 6
M_i	4, 8, 12, ..., 256
$h_i(x)$	ReLU, tanh, sigmoid
α	0.01, 0.001, 0.0001

through the possible options and combinations of the hyperparameters. The objective that we focus on is the accuracy on the validation set, this is equal to the amount of correctly predicted values in the validation set divided by the total number of observations in the validation set. A score is predicted correctly when the predicted value is equal to the actual value. While mainly relevant for classification problems, this score will provide an indication of whether the model can predict any of the values correctly. Next, the maximum number of different settings that can be tried, which are called trials, is equal to 10. Lastly, each trial can consist of 2 executions, the number of times the model is build from scratch given these parameters, and each of these executions consist of 2 training epochs, which are the number of batches of data on which this model is trained.

After the tuner finds the best parameter settings we build the model using these settings and train the model on the test and the validation set combined. To prevent overfitting in this last step, we implement early stopping with a patience of 20, which means that the algorithm stops training if there is no improvement for during last 20 training epochs. After the training has stopped we select the best parameter values that were found for the final model.

As optimisation method we use Adam as defined by Kingma & Ba (2014). This is a version of the stochastic gradient descent (SGD) method with a faster convergence rate than SGD. Kingma & Ba (2014) find a lot of advantages for their method compared to other methods, such as the computational efficiency, the small memory requirements, and the efficiency when working with a large data set and/or a large amount of parameters. We only tune the learning rate, α, as defined in Table 4, this value approximately bounds the magnitude of the steps for the parameters. For the other hyperparameters we use the standard values as recommended by Kingma & Ba (2014), which can be found in their paper.

Next we define the loss function based on which the model will optimise the parameters, as this is used for defining the penalty for errors in the prediction. As recommended by Hastie et al. (2009) we use the sum of squared errors function. If we define the actual output as y, the predicted output as $\hat{f}(x)$, and the number of observations as N, we can define this loss function as,

$$L(y, \hat{f}(x)) = \sum_{i=1}^{N} (y_i - \hat{f}(x_i))^2. \tag{8}$$

We apply the following methods to prevent overfitting. First, we apply L2 regularization, with default parameter value 0.01. This means that we add the term $0.01 \cdot \sum_{j=1}^{p} \beta_j^2$ to the loss function, which causes the weights to stay lower and thus helps minimize the amount of nodes that are active in the model (Chollet et al., 2015; Hastie et al., 2009; Kingma & Ba, 2014). Besides, we apply early stopping by using a validation set. The validation set ensures that the

model stops the fitting when the error on the validation set starts increasing while the error on the training set is decreasing. From the total data set, 70% is used for the training of the model, 15% is used for the validation, and the last 15% is used for testing (Hastie et al., 2009). The test set is used for the comparison of different models.

4.2.3 Convolutional neural network

We use the neural network as described in Section 4.2.2 as the starting point for the convolutional neural network (CNN). We provide the input data for this neural network in 3D-form and add a convolutional layer before the other layers in the NN. The data is presented in 3D-form because this allows us to keep the yard dimensions and the locations of stacks relative to each other.

The convolutional layer will be a 2D-convolutional layer, which has two parameters that are fixed a priori. The first is the input shape, which has to be used if the convolutional layer is the first layer in the model and defines the shape of the input for the model. We set this to (r, b, S), with r the number of rows in one module, b the number of bays in one module, and S the number of data values in one stack. Next, we set the kernel size to $(2, 2)$. This defines the height and the width of the convolution window, which is the size of a matrix of values that the model looks at as one observation to find connections between surrounding data points. We use these values because we aspect that the slots directly next to each other are the most related. The convolutional layer also has two parameters which we tune, these are presented in Table 5. The first hyperparameter is the number of filters, F, this defines the number of output channels from the convolutional layer. The second is the activation function for the convolutional layer, $h_C(x)$. To ensure we can use the output of the convolutional layer in this rest of the model we add a flattening layer, this turns the data into a one-dimensional array.

Table 5: Additional hyperparameter tuning options for the CNN.

Hyperparameter	Values
F	2, 4, 6, ..., 30
$h_C(x)$	ReLU, tanh, sigmoid

4.3 Simulation

We use simulation to validate and test our methods, as has been done by Duinkerken et al. (2001), Hirashima et al. (2006), and Xie et al. (2022). Our simulation represents a simplified version of a container terminal in which we only account for import and export containers. We use coordinates to define the locations of the slots within a module. We simplify the transfer zones by assuming that containers are located in one spot. The coordinates of the transfer zones on the landside and the waterside are the same for each module and are $(x, y, z) = (\lceil \frac{r+1}{2} \rceil, b+1, 1)$ and $(x, y, z) = (\lceil \frac{r+1}{2} \rceil, 0, 1)$ respectively.

Whether a vessel can berth is determined by the number of quay crane (QC)s that are available on the quay, and the number of QCs that the vessel requires. If there are enough QCs available the vessel berths, otherwise it waits 60 seconds before trying again. We do not account for whether the QCs are next to each other, we only look at the amount that is available. We

assume that a vessel berths exactly in the middle of all modules.

For the arrivals at the gate, thus the land side, we use a method that differentiates between the busy and the calm hours. This means that if $a\%$ of gate arrivals occur during the busy period and $(1-a)\%$ during the calm period, we implement this as follows. Every time we generate a gate arrival, we check during which time period this moment falls, and we accept this time with the respective probabilities a and $(1-a)\%$. If the time is not accepted, we generate a new time and repeat the process.

We assume the following about the rail mounted gantry (RMG) cranes. The starting position of the first RMG in coordinates is $(x, y, z) = (\lceil \frac{r+1}{2} \rceil, \lceil \frac{b+1}{2} \rceil, h)$, and of the second RMG is $(x, y, z) = (\lceil \frac{r+1}{2} \rceil, \lceil \frac{b+1}{2} \rceil, h+1)$. If an RMG has placed a container and there are no other jobs to perform it will wait at its current spot. The travelled distance when performing a move is calculated as the Manhattan distance. This distance from point (x_1, y_1, z_1) to point (x_2, y_2, z_2) can be calculated as,

$$distance = |x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2|. \tag{9}$$

In general, container yards are filled for 60-80%, thus we use this as an indication for the amount of arrivals, and how far we fill the yard before starting the simulation. We use a specific method to fill the yard before starting the actual simulation, as is also done by for example Dekker et al. (2007). For filling the yard we let ships arrive that only drop off containers, which are immediately placed in the yard. The slots where these containers are placed are based on the algorithm that is implemented in the simulation.

We implement the model as a discrete event simulation (DES) using the free package Salabim in Python 3.11.5 (van der Ham, 2018). An overview of the steps in each of the events is shown in Appendix A. We initialise the simulation by planning the arrival of the vessels, and with this the arrivals of the containers that need to be loaded on those vessels. When these containers arrive at the gate the best module is selected and they are moved to the LS transferzone corresponding to this module. When in the LS transferzone, the best slot in the module is selected and the container is moved there. At the expected departure time of these containers, they are moved to the WS transferzone, and they leave the simulation. Next, with every vessel arrival we unload containers from this vessel. These then arrive at the quay. Upon arrival at the quay the best module is selected and they are moved to the WS transferzone corresponding to this module. At the quay the expected departure time for this container is also determined. When in the WS transferzone the best slot in the module is found and the container is moved there. When their departure time comes around they are removed from the yard and placed in the LS transferzone and they leave the simulation.

We make the following note on the reshuffling of a container. When trying to remove a container from the yard it is checked if all slots above this container are empty. If there is a container in the highest slot, this container gets reshuffled, and we plan the departure of the original container again. Continuing until the container is actually removed. When we reshuffle a container we plan the departure from its new slot at either its actual departure time, or the current time, whichever is the latest. Also note that when we try to remove a container from a slot we are not allowed to place any containers in this stack until the container is removed.

5 Results

In this section we present the results of our research. We first elaborate on the data we use, then we describe and compare the results we find with the neural networks for two versions of our simulation.

5.1 Algorithm and simulation set-up

5.1.1 Benchmark algorithm

We define the benchmark algorithm based on a penalty function. The used attributes and their corresponding penalty values are presented in Table 6. The information in this table is defined based on information from experts at Konecranes. The penalty values for 'empty ground slot', 'workload', and 'TZ occupancy' are pre-tuned to ensure the containers are distributed over all the modules. The penalties are split in three categories, (1) penalties that are relevant for all containers, (2) penalties that are relevant if the container has transit direction export, and (3) penalties that are relevant if the container has transit direction import. Note that the penalties for 'workload' and 'TZ occupancy' are module specific, and thus may influence the selection of a module, but do not influence which slot is the best within a module.

5.1.2 Neural networks

Next, we define the input data for the neural networks, which is based on the stack information. The specific attributes that are used are presented in Table 7. The categorical attributes 'transit direction', 'outbound service ID', 'outbound carrier ID', 'next destination', 'line operator', and 'inbound service ID' are transformed using one-hot encoding, see explanation in Section 4.2.1. An example of the one-hot encoding for the attribute 'transit direction' is shown in Table 8. Some attributes are not included in this information because they provide no additional information or are not reliable in practice. An example of an attribute that is not reliable in practice is 'edt'. In our implementation the estimated departure time of a container corresponds with the actual departure time of a container, and in our case we define this value before the container is placed in the yard. However, in practice there are many deviations possible from the expected departure time, and it is not known in advance.

Table 6: The penalty values and their definitions.

Attribute	Penalty Value	Definition
ALL CONTAINERS		
Transit direction	400	When the transit direction of the container differs from any of the containers below.
Empty ground slot	70	When this slot is an empty ground slot.
Workload	$7500 \cdot c_{fill}$	Based on how filled the yard is, with c_{fill} the percentage of the module that is filled with containers.
TZ occupancy	$0.2 \cdot c_{wait}^3$	Based on the occupancy of the transferzone, with c_{wait} the percentage of the total waiting containers that are waiting for this module.
Outbound service	50	When the outbound service of the container differs from the container below.
Outbound vessel visit	50	When the outbound vessel visit of the container differs from the container below.
Outbound vessel	50	When the outbound vessel of the container differs from the container below.
Next destination	50	When the next destination of the container differs from the container below.
Lower than max weight	$75 \cdot c_{max}$	When the weight class of the container is lower than the maximum weight class of the stack, where c_{max} is the class difference to the maximum weight.
Lower than top weight	$75 \cdot c_{top}$	When the weight class of the container is lower than the weight class of the container below, where c_{top} is the class difference to the top container weight.
Higher than max weight	$10 \cdot c_{max}$	When the weight class of the container is higher than the maximum weight class of the stack, where c_{max} is the class difference to the maximum weight.
Higher than top weight	$50 \cdot c_{top}$	When the weight class of the container is higher than the weight class of the container below, where c_{top} is the class difference to the top container weight.
EXPORT CONTAINERS		
Tier index	$6 - z$	For every tier, where the tier of the slot is given by z.
Bay index	y	For every bay, the number of bays from the water side, where the bay of the slot is given by y.
IMPORT CONTAINER		
Tier index	$10 \cdot z$	For every tier, where the tier of the slot is given by z.
Bay index	$47 - y$	For every bay, the number of bays from the land side, where the bay of the slot is given by y.

Note that these penalty values are specific for our terminal layout and dimensions.

Table 7: Stack information attributes.

Attribute	Definition
Length	The length of the containers in the stack, 0 if empty.
Transit direction	The transit direction of the containers in the stack, 'mixed' if it contains both 'import' and 'export' containers, 'none' if empty.
Outbound service ID	Outbound service ID of the top container in the stack, -1 if empty.
Outbound carrier visit ID	Outbound carrier visit ID of the top container in the stack, -1 if empty.
Outbound carrier ID	Outbound carrier ID of the top container in the stack, -1 if empty.
Next destination	Next destination of the top container in the stack, 'none' if empty.
Tier	The lowest empty tier in the stack, 1 if empty.
Max weight	The weight class of the heaviest container in the stack, 0 if empty.
Top weight	The weight of the top container in the stack, 0 if empty.
Line operator	The line operator of the top container in the stack, -1 if empty.
Inbound service ID	The inbound service ID of the top container in the stack, -1 if empty.

Note that for most attributes the value for an empty stack is equal to -1, because the value 0 is already reserved for the case that the transit direction does not correspond to the transit direction for which this attribute is defined.

Table 8: Example of one-hot encoding for the attribute 'transit direction'.

Value	One-hot encoded array
import	[1, 0, 0]
export	[0, 1, 0]
mixed	[0, 0, 1]
none	[0, 0, 0]

5.1.3 Simulation

To compare the different algorithms we implement a simulation. We create three data cases, *small*, *medium*, and *large*, on which we compare the algorithms. For the simulation we first define the parameters, sets, and distributions that are the same in all simulations. These are presented in Table 9. All parameters are assigned a value based on information from experts at Konecranes. First, s_{rmg} defines the speed at which the RMGs can move through the yard. The distance these travel is based on the dimensions of the containers. Standard 20ft containers have a width of 2.44 meters, a length of 6.1 meters, and a height of 2.59 meters (*20-foot Container - Dimensions, Measurements and Weight*, 2023). Next, s_{qtz} defines the speed at which containers are moved from the quay to the transfer zones.

Next, we define 5 sets for the simulation. These define the possible values that containers can have for certain attributes. We define the set W for the attribute 'weight', and the set δ for the attribute 'length', for distributions across these two sets can differ per vessel. Next we have

Table 9: The parameters, sets, and distributions that are the same for all simulations, with their corresponding values. All distances are in meters, times in seconds, and speeds in m/sec.

Parameter	Abbreviation	Value
Bays	b	46
Rows	r	8
Tiers	h	5
Number of modules case *small*	m_{small}	1
Number of modules case *medium*	m_{medium}	3
Number of modules case *large*	m_{large}	15
RMG speed	s_{rmg}	3.25
Q-TS Speed	s_{qts}	8
QC move time for case i	t_{qc}	$120/N_{QC,i}$
Sets		
Weight classes	W	{1, 2, 3}
Container length	δ	{20, 40}
Next destination 1	D_1	{A, B}
Next destination 2	D_2	{A, B, C, D}
Next destination 3	D_3	{A, B, C, D, E, F, G, H}
Distributions		
Export arrival	F_{EA}	Triangular(86400, 1209600, 475200)
Import departure	F_{ID}	Triangular(86400, 1209600, 475200)
Vessel arrival	F_{VA}	Triangular(-43200, 432000, 43200)

the sets D_1, D_2 and D_3 for the attribute 'next destination'. The distributions for these sets are always as presented in Table 10. Lastly, we define the distributions that are the same for all simulations. Export containers arrive a certain amount of time before the expected arrival of the vessel, the first distribution, F_{EA}, provides the distribution for this time in seconds. The second distribution, F_{ID}, determines the amount of time an import container stays in the yard before it is picked up by a truck. The last distribution, F_{VA}, provides the deviation from the expected arrival time for the vessel. This means that the vessels arrive somewhere between 43200 seconds (half a day) before, to 432000 seconds (5 days) after their expected arrival time.

Table 10: The distributions of the containers across the various destinations for each destination set.

Destination set	Distribution
D_1	(0.5, 0.5)
D_2	(0.25, 0.25, 0.25, 0.25)
D_3	(0.2, 0.2, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1)

Besides the predefined parameters we have parameters which are adjusted per simulation, see Table 11. We first have the number of RMGs per module, N_{rmg}, and the number QCs at the quay, $N_{QC,i}$. The number of QCs is different for each data case. Next, we have the number of seconds which a container can be delayed in its removal before we actually count it as a delay, t_{delay}. The fourth parameter is the percentage of the yard that we fill with containers before we start running the simulation, ρ. This thus provides a base level on the number of containers that are always in the yard. Next is the time required to move a container between the vessel and the

quay using the QCs in seconds, denoted by t_{qc}. The parameters τ_l and τ_u, define which numbers of load and unload containers should be used respectively. The parameter V_{qc} defines which values of the number of required QCs per vessel should be used. The last three parameters are related to the distributions that container arrivals at the gate follow. The gate is busy between θ_{start} and θ_{end}, during the other hours it is calm. The percentage of arrivals at the gate that occur during these busy hours is defined by γ.

Table 11: The parameters that are adjusted per simulation.

Parameter	Abbreviation
Number of RMGs per module	N_{rmg}
Number of QCs at the quay per data case i	$N_{QC,i}$
Allowed delay	t_{delay}
Start filling percentage of yard	ρ
Time to move a container with a QC	t_{qc}
Load containers	τ_l
Unload containers	τ_u
Required quay cranes by vessel	V_{qc}
Start time busy hours at the gate	θ_{start}
End time busy hours at the gate	θ_{end}
Busy percentage at the gate	γ

For the containers arriving via vessel we use a recurring vessel schedule. This schedule is created in correspondence with the simulation department from Konecranes. These schedules are presented in Table 12. The distributions are presented in the same order as the sets are presented in Table 9. For each arriving vessel we determine a list of containers with their attributes that need to be unloaded into the yard, and a list of containers that need to be loaded onto the vessels. These container attributes are retrieved from sets and their corresponding distributions. To fill up the yard we use the available data on the vessels.

We need at least a certain amount of data before we can use it to train the NN. We try various run lengths for the simulation, which produce differently sized data sets, to investigate the influence of the amount of data on the algorithms. We define three different combinations of time periods for the simulation run. Each time period consists of two parameters, ϵ the time in weeks to warm up the simulation, and η the time in weeks, during which we measure statistics. The total run time then consists of $\epsilon + \eta$ weeks. The combinations of (ϵ, η) that we use are $(1, 4)$, $(4, 20)$ and $(4, 52)$. During the warm-up period of the simulation we see more containers arrive than leave the yard, because the containers that need to leave the yard have not arrived yet. Therefore, we expect that a longer warm-up period will ensure that the arrival and departure processes have stabilized.

Table 12: The vessel schedules and the corresponding distributions for all data cases.

Vessel Line	Arrival Day	ETA [1]	Load Containers (TEU) $\tau_l = 1$	Load Containers (TEU) $\tau_l = 2$	Unload Containers (TEU) $\tau_u = 1$	Unload Containers (TEU) $\tau_u = 2$	Length distribution	Weight Distribution	Destination set	QCs per vessel $V_{qc} = 1$	QCs per vessel $V_{qc} = 2$
data case = *small*											
1	1	08:00	50	90	60	110	(0.3, 0.7)	(0.5, 0.3, 0.2)	D_1	1	1
2	3	12:00	50	80	50	75	(0.35, 0.65)	(0.3, 0.5, 0.2)	D_1	1	1
3	5	08:00	60	90	30	60	(0.3, 0.7)	(0.5, 0.3, 0.2)	D_1	1	1
4	7	12:00	55	95	50	85	(0.4, 0.6)	(0.5, 0.3, 0.2)	D_1	1	1
data case = *medium*											
1	1	08:00	80	135	100	165	(0.3, 0.7)	(0.5, 0.3, 0.2)	D_1	1	2
2	2	16:00	100	170	75	125	(0.4, 0.6)	(0.2, 0.7, 0.1)	D_2	1	1
3	3	12:00	70	125	80	145	(0.35, 0.65)	(0.3, 0.5, 0.2)	D_1	1	1
4	4	19:00	90	145	105	170	(0.45, 0.55)	(0.6, 0.2, 0.2)	D_1	1	2
5	5	08:00	110	185	80	135	(0.3, 0.7)	(0.5, 0.3, 0.2)	D_2	1	2
6	6	22:00	75	180	95	160	(0.35, 0.65)	(0.5, 0.3, 0.2)	D_1	1	1
7	7	12:00	95	160	80	135	(0.4, 0.6)	(0.5, 0.3, 0.2)	D_2	1	1
data case = *large*											
1	1	08:00	85	260	90	275	(0.4, 0.6)	(0.5, 0.3, 0.2)	D_2	1	1
2	3	12:00	1450	2260	1340	2090	(0.35, 0.65)	(0.3, 0.5, 0.2)	D_3	1	8
3	5	08:00	65	215	90	300	(0.3, 0.7)	(0.5, 0.3, 0.2)	D_2	1	1
4	6	10:00	1350	2130	1500	2370	(0.35, 0.65)	(0.3, 0.5, 0.2)	D_3	1	8
5	7	15:00	70	235	85	285	(0.4, 0.6)	(0.6, 0.2, 0.2)	D_2	1	1

[1] Estimated Time of Arrival (ETA).

5.2 Simple simulation

Parameter settings

We start with a simple implementation of the simulation, the parameter settings for this simulation are presented in Table 13. First of all, we define that there is one QC, and each vessel needs exactly one QC to be serviced. Next, we set that the yard is serviced by one RMG, and a container counts as delayed if it needs more than 900 seconds (15 minutes) to leave the yard. The start percentage to which the yard is filled is set to 25%, due to this already decreased amount of space in the yard we use the sets with a smaller amount of load and unload containers. Lastly, we assume that the gate is equally busy all throughout the day.

Table 13: Parameter values for the simple simulation.

Parameter	Value	Parameter	value	Parameter	Value
$N_{QC,small}$	1	N_{rmg}	1	θ_{start}	00:00
$N_{QC,medium}$	1	t_{delay}	900	θ_{end}	23:59
$N_{QC,large}$	1	ρ	25%	γ	100
V_{qc}	1	τ_l	1	τ_u	1

Training neural network

We first run this simulation using the penalty function to generate the input data for the NNs. The hyperparameters from tuning the neural networks are presented in Table 21 in Appendix B. The accuracy scores on the test sets for all data cases are presented in Table 14. Note that for some cases there is no accuracy score because there is no training data available. This occurred because the simulation using the benchmark algorithm was unable to finish due to it not being able to find a slot for a container at a certain point and we do not allow the simulation to continue if a container cannot be placed.

The test accuracy of the neural network is greater than 0.00% for 14 out of the 21 cases for which training data was available. Another interesting observation is that the accuracy score of the NN is decreasing as the λ increasing, which means that the models are less accurate in predicting the delay of containers over a long period of time. We also see no specific influence from the length of the data set on the accuracy score. Though in all cases the combination $(\epsilon, \eta) = (1, 4)$ has the lowest accuracy score, the difference with the other lengths is small, and the effect of going from $\eta = 20$ to $\eta = 52$ can be either positive or negative.

Comparison using simulation

Since our main goal is not to train a NN that can copy the benchmark algorithm, but to find a NN that learns from the decisions of the benchmark algorithm to find which actions provide the least delay for removing containers from the terminal, we compare the two algorithms using a simulation study. We only run the simulation for the NNs that have an accuracy score greater than 0.00%, because we expect the algorithm to not have any additional value the score is 0.00%.

The results from these simulations are presented in Table 15 and Table 16. Note that we have no results for the data case *large*, because of various reasons. Namely, memory issues, being unable to find a slot for a container at some point, and having no training data for the NN.

Table 14: The accuracy scores of the neural network on the test set based on the simple simulation.

Data case	λ	ϵ	η	Accuracy (%)
small	1	1	4	64.69
small	1	4	20	70.78
small	1	4	52	66.18
small	24	1	4	25.85
small	24	4	20	25.63
small	24	4	52	0.00
small	168	1	4	0.00
small	168	4	20	0.00
small	168	4	52	0.00
medium	1	1	4	69.23
medium	1	4	20	71.87
medium	1	4	52	72.39
medium	24	1	4	36.24
medium	24	4	20	41.80
medium	24	4	52	42.88
medium	168	1	4	0.00
medium	168	4	20	0.00
medium	168	4	52	0.00
large	1	1	4	68.93
large	1	4	20	- [1]
large	1	4	52	- [1]
large	24	1	4	58.85
large	24	4	20	- [1]
large	24	4	52	- [1]
large	168	1	4	4.81
large	168	4	20	- [1]
large	168	4	52	- [1]

[1] No training data available.

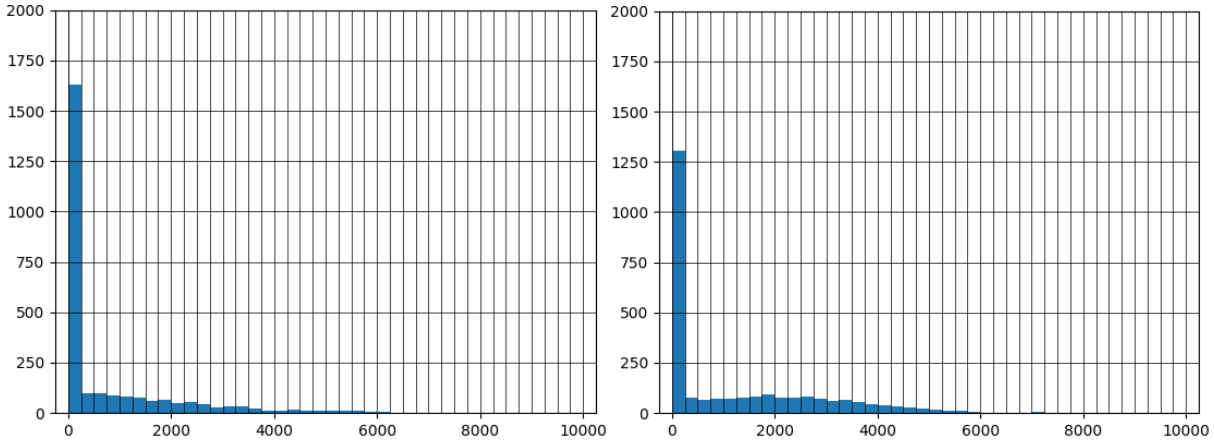
Since we focus mostly on the delay of containers we first look at the delay statistics from the simulation (Table 15). The first important observation is that the average delay of containers leaving the yard is always lower using the benchmark algorithm than using the NN. Another statistic in which it is clearly visible that the benchmark still performs better than the NN is the 50% score. This indicates that 50% of delays are between the minimum and this value, for the simulation using the benchmark algorithm this value is always 0, whereas this value goes up to 1140 seconds for the simulation using a NN. The case for which the NN best approaches the performance of the benchmark algorithm is for the data case *medium* with $\lambda = 1$, $\epsilon = 1$, and $\eta = 4$. For this case the mean is less than 500 seconds higher, the 50% score is 213 seconds, and the maximum is just over 3000 seconds higher.

To get some more insight in the distribution of the delays we look at a histogram of delays of containers leaving the yard for the case in which the NN performs the closest to the benchmark algorithm. In Figure 3a we see the distribution of these delays when using the benchmark algorithm. Here we mainly see the peak at a delay of 0 to 250 seconds. After this, the amount of observations per bin decreases as the delay increases, with the values at the end having so little observations they are not even visible. In Figure 3b we see this distribution when using the NN. Here we see the same shape as for the benchmark algorithm. The main difference is the increase

Table 15: Statistics on the delay per containers in seconds based on the given performance measure λ, generated using the simple simulation.

Data Case	λ	ϵ	η	Algorithm	Mean	St. dev.	Min	50%	Max
small	1	1	4	Benchmark	686	925	0	0	5522
				NN	2319	2757	0	814	11255
small	1	4	20	Benchmark	675	953	0	0	5208
				NN	2613	3067	0	904	11486
small	1	4	52	Benchmark	649	949	0	0	6718
				NN	2354	2861	0	516	14152
small	24	1	4	Benchmark	686	925	0	0	5522
				NN	2543	2869	0	1140	10096
small	24	4	20	Benchmark	675	953	0	0	5208
				NN	2407	2861	0	746	10941
small	24	4	52	Benchmark	649	949	0	0	6718
				NN	- [1]	- [1]	- [1]	- [1]	- [1]
small	168	1	4	Benchmark	686	925	0	0	5522
				NN	- [1]	- [1]	- [1]	- [1]	- [1]
small	168	4	20	Benchmark	675	953	0	0	5208
				NN	- [1]	- [1]	- [1]	- [1]	- [1]
small	168	4	52	Benchmark	649	949	0	0	6718
				NN	- [1]	- [1]	- [1]	- [1]	- [1]
medium	1	1	4	Benchmark	762	1328	0	0	7031
				NN	1259	1675	0	213	10231
medium	1	4	20	Benchmark	970	1635	0	0	7686
				NN	4258	5222	0	1090	20558
medium	1	4	52	Benchmark	1110	1757	0	0	7805
				NN	- [2]	- [2]	- [2]	- [2]	- [2]
medium	24	1	4	Benchmark	762	1328	0	0	7031
				NN	3439	4727	0	960	19588
medium	24	4	20	Benchmark	970	1635	0	0	7686
				NN	- [3]	- [3]	- [3]	- [3]	- [3]
medium	24	4	52	Benchmark	1110	1757	0	0	7805
				NN	- [3]	- [3]	- [3]	- [3]	- [3]
medium	168	1	4	Benchmark	762	1328	0	0	7031
				NN	- [1]	- [1]	- [1]	- [1]	- [1]
medium	168	4	20	Benchmark	970	1635	0	0	7686
				NN	- [1]	- [1]	- [1]	- [1]	- [1]
medium	168	4	52	Benchmark	1110	1757	0	0	7805
				NN	- [1]	- [1]	- [1]	- [1]	- [1]
large	1	1	4	Benchmark	- [3]	- [3]	- [3]	- [3]	- [3]
				NN	- [2]	- [2]	- [2]	- [2]	- [2]
large	1	4	20	Benchmark	- [3]	- [3]	- [3]	- [3]	- [3]
				NN	- [4]	- [4]	- [4]	- [4]	- [4]
large	1	4	52	Benchmark	- [3]	- [3]	- [3]	- [3]	- [3]
				NN	- [4]	- [4]	- [4]	- [4]	- [4]
large	24	1	4	Benchmark	- [3]	- [3]	- [3]	- [3]	- [3]
				NN	- [2]	- [2]	- [2]	- [2]	- [2]
large	24	4	20	Benchmark	- [3]	- [3]	- [3]	- [3]	- [3]
				NN	- [4]	- [4]	- [4]	- [4]	- [4]
large	24	4	52	Benchmark	- [3]	- [3]	- [3]	- [3]	- [3]
				NN	- [4]	- [4]	- [4]	- [4]	- [4]
large	168	1	4	Benchmark	- [3]	- [3]	- [3]	- [3]	- [3]
				NN	- [2]	- [2]	- [2]	- [2]	- [2]
large	168	4	20	Benchmark	- [3]	- [3]	- [3]	- [3]	- [3]
				NN	- [4]	- [4]	- [4]	- [4]	- [4]
large	168	4	52	Benchmark	- [3]	- [3]	- [3]	- [3]	- [3]
				NN	- [4]	- [4]	- [4]	- [4]	- [4]

[1] NN did not provide accuracy score above 0.00%.
[2] Unable to finish simulation due to memory error.
[3] Unable to finish simulation because algorithm was unable to find a slot at some point.
[4] There was no training data available for the NN.



(a) Distribution using the benchmark algorithm.

(b) Distribution using the NN.

Figure 3: Histogram of the delays of containers leaving the yard in seconds for the data case *medium*, with $\lambda = 1$, $\epsilon = 1$ and $\eta = 4$. Based on the simple simulation.

of the amount of values on the x-axis, showing that, though it occurs in very little cases, larger delays occur than with the benchmark algorithm. The visible bars for both algorithms end just before 6000 seconds delay.

Next, we look at the statistics on the arrivals and the departures of the containers as in Table 16. In general the number of arrivals and departures, for both the import and the export containers, are similar for the benchmark algorithm and the NN. The most interesting statistic in here is the number of reshuffles, since this is closely related to the delay. The number of reshuffles required by the NN is always higher than for the benchmark algorithm. This difference increases significantly when the length of the simulation is increased. For example for the data case *small* with $\lambda = 1$, the difference goes from just under 500 moves, to just under 3000 moves, to finally over 8000 moves.

Summary simple simulation

To conclude, for the simple simulation we find that the benchmark algorithm always performs better. The accuracy scores of the neural networks were quite high, however the average delays for the neural networks are always much larger than for the benchmark algorithms. For the neural networks this higher average is mainly caused by some large outliers. When looking at the number of reshuffles, we find that these are also much larger for the neural networks.

Table 16: Results from simple simulation, with statistics on the arrivals and departures of containers.

Data case	λ	ϵ	η	Algorithm	Import Arrivals	Import Departures	Export Arrivals	Export Departures	Reshuffles
small	1	1	4	Benchmark	431	340	492	516	1195
				NN	435	347	492	516	1630
small	1	4	20	Benchmark	2341	2341	2623	2635	7416
				NN	2301	2302	2623	2635	10206
small	1	4	52	Benchmark	6051	6056	6771	6799	18301
				NN	6060	6060	6771	6799	26491
small	24	1	4	Benchmark	431	340	492	516	1195
				NN	436	347	492	516	1803
small	24	4	20	Benchmark	2341	2341	2623	2635	7416
				NN	2291	2297	2623	2635	10097
small	24	4	52	Benchmark	6051	6056	6771	6799	18301
				NN	- [1]	- [1]	- [1]	- [1]	- [1]
small	168	1	4	Benchmark	431	340	492	516	1195
				NN	- [1]	- [1]	- [1]	- [1]	- [1]
small	168	4	20	Benchmark	2341	2341	2623	2635	7416
				NN	- [1]	- [1]	- [1]	- [1]	- [1]
small	168	4	52	Benchmark	6051	6056	6771	6799	18301
				NN	- [1]	- [1]	- [1]	- [1]	- [1]
medium	1	1	4	Benchmark	1418	1066	1471	1531	3850
				NN	1404	1054	1471	1531	5134
medium	1	4	20	Benchmark	7626	7506	7589	7639	24651
				NN	7606	7486	7589	7639	29362
medium	1	4	52	Benchmark	19610	19678	19840	19897	65460
				NN	- [2]	- [2]	- [2]	- [2]	- [2]
medium	24	1	4	Benchmark	1418	1066	1471	1531	3850
				NN	1394	1051	1471	1531	5252
medium	24	4	20	Benchmark	7626	7506	7589	7639	24651
				NN	- [3]	- [3]	- [3]	- [3]	- [3]
medium	24	4	52	Benchmark	19610	19678	19840	19897	65460
				NN	- [3]	- [3]	- [3]	- [3]	- [3]
medium	168	1	4	Benchmark	1418	1066	1471	1531	3850
				NN	- [1]	- [1]	- [1]	- [1]	- [1]
medium	168	4	20	Benchmark	7626	7506	7589	7639	24651
				NN	- [1]	- [1]	- [1]	- [1]	- [1]
medium	168	4	52	Benchmark	19610	19678	19840	19897	65460
				NN	- [1]	- [1]	- [1]	- [1]	- [1]
large	1	1	4	Benchmark	- [3]	- [3]	- [3]	- [3]	- [3]
				NN	- [2]	- [2]	- [2]	- [2]	- [2]
large	1	4	20	Benchmark	- [3]	- [3]	- [3]	- [3]	- [3]
				NN	- [4]	- [4]	- [4]	- [4]	- [4]
large	1	4	52	Benchmark	- [3]	- [3]	- [3]	- [3]	- [3]
				NN	- [4]	- [4]	- [4]	- [4]	- [4]
large	24	1	4	Benchmark	- [3]	- [3]	- [3]	- [3]	- [3]
				NN	- [2]	- [2]	- [2]	- [2]	- [2]
large	24	4	20	Benchmark	- [3]	- [3]	- [3]	- [3]	- [3]
				NN	- [4]	- [4]	- [4]	- [4]	- [4]
large	24	4	52	Benchmark	- [3]	- [3]	- [3]	- [3]	- [3]
				NN	- [4]	- [4]	- [4]	- [4]	- [4]
large	168	1	4	Benchmark	- [3]	- [3]	- [3]	- [3]	- [3]
				NN	- [2]	- [2]	- [2]	- [2]	- [2]
large	168	4	20	Benchmark	- [3]	- [3]	- [3]	- [3]	- [3]
				NN	- [4]	- [4]	- [4]	- [4]	- [4]
large	168	4	52	Benchmark	- [3]	- [3]	- [3]	- [3]	- [3]
				NN	- [4]	- [4]	- [4]	- [4]	- [4]

[1] NN did not provide accuracy score above 0.00%.
[2] Unable to finish simulation due to memory error.
[3] Unable to finish simulation because algorithm was unable to find a slot at some point.
[4] There was no training data available for the NN.

5.3 Advanced simulation

Parameter settings

Next, we implement a more advanced version of the simulation for which the parameter values are presented in Table 17. In this version we implement two RMGs per module, and allow there to be more QCs, which differ per data case. We also use that vessels may need a different number of QCs to berth. To get a more detailed insight in the delay of containers, we count a container as delayed when it needs more than 0 seconds to leave the yard. For a more realistic view of the containers in the yard, we do not fill the yard prior to the start of the simulation, but to get a similar filling percentage during the simulation we use the situations with more load and unload containers. Lastly, we implement busy hours at the gate during the day, with much less activity during the night.

Table 17: Parameter values for the advanced simulation.

Parameter	Value	Parameter	value	Parameter	Value
$N_{QC,small}$	1	N_{rmg}	2	θ_{start}	08:00
$N_{QC,medium}$	2	t_{delay}	0	θ_{end}	20:00
$N_{QC,large}$	8	ρ	0%	γ	98
V_{qc}	2	τ_l	2	τ_u	2

Training convolutional neural network

For the advanced simulation we use the same steps as for the simple simulation. We run the simulation using the penalty function to generate input to train the convolutional neural network (CNN). The advanced simulation using the benchmark algorithm was unable to finish for all combinations with data case *large* because it was unable to find a slot for a container at some point, thus we no longer look at this case. The hyperparameters of the CNN are presented in Table 22 in Appendix C. The accuracy scores on the test sets for the CNNs are presented in Table 18. Note that in some cases we do not have an accuracy score, this is because of memory issues. We only find accuracy scores greater than 0.00% when $\lambda = 1$. This shows that these NNs are still unable to predict very far ahead.

Extra training convolutional neural network

Since the accuracy scores are in this case still quite low, we try to improve these by adding extra data to train the CNNs that have an accuracy score greater than 0.00%. Thus, we run a simulation with the CNNs to generate the extra data. The CNNs can then learn from their previous decisions, this would be similar as to how it could be used in reality by Konecranes. The accuracy scores on the test sets for these extra trained CNNs are also presented in Table 18. For four out of five cases these extra trained CNNs still provide an accuracy score greater than 0.00%. However, it should be noted that these accuracy scores are the same as the accuracy scores from the original CNNs. In the small data case with $\lambda = 1$, $\epsilon = 1$, and $\eta = 4$ we find an accuracy score of 0.00% while the original CNN for this case had an accuracy score of nearly 15%. This means that the CNN has started to overfit by adjusting the parameters to better fit the training data, but thus no longer fitting the test data. Lastly, we see no indication that the

Table 18: The accuracy scores of the convolutional neural networks on the test sets based on the advanced simulation.

				Accuracy (%)		New Accuracy (%)	
Data case	λ	ϵ	η	CNN	Extra trained CNN	CNN	Extra trained CNN
small	1	1	4	14.61	0.00	58.70	0.00
small	1	4	20	16.19	16.19	14.21	14.21
small	1	4	52	16.10	16.10	15.69	15.69
small	24	1	4	0.00	-[2]		
small	24	4	20	0.00	-[2]		
small	24	4	52	0.00	-[2]		
small	168	1	4	0.00	-[2]		
small	168	4	20	0.00	-[2]		
small	168	4	52	0.00	-[2]		
medium	1	1	4	16.26	16.26	28.37	28.37
medium	1	4	20	15.28	15.28	14.23	14.23
medium	1	4	52	-[1]			
medium	24	1	4	0.00	-[2]		
medium	24	4	20	0.00	-[2]		
medium	24	4	52	-[1]			
medium	168	1	4	0.00	-[2]		
medium	168	4	20	0.00	-[2]		
medium	168	4	52	-[1]			

[1] CNN is unable to provide an accuracy score due to a memory error during the training process.
[2] CNN is not trained further because the original CNN has an accuracy score of 0.00%.

size of the data set influences the accuracy of the model.

The test sets on which we determine these original accuracy scores are from the data generated in simulations using the benchmark algorithm. Because the CNN would be used for the entire simulation, it is relevant that the CNNs can still predict the delays accurately in simulation using a CNN. Thus, we also look at the accuracy scores on new test sets which are selected from data that is generated using the original CNN. These new accuracy scores are presented in the column 'New Accuracy' in Table 18. We see the same patterns with these accuracy scores as with the original accuracy scores. The accuracy score on this new set is larger than the accuracy score on the original set in 2 cases, and is smaller in 2 cases. This shows that based on the accuracy scores there is no clear improvement in the CNNs by providing it with extra training data.

Comparison using simulation

Next, we also compare the CNN and the extra trained CNN with the benchmark algorithm using a simulation study. We again only run the simulations for the CNNs that have an accuracy score greater than 0.00%. The results from these simulations are presented in Table 19 and Table 20.

Since the delay of containers is still the most important results from the simulation, we analyse these first. The statistics on this are presented in Table 19. For the *small* data case the benchmark algorithm still always has the lowest mean. However, for the *medium* data case the CNNs have a lower mean for $\lambda = 1$, $\epsilon = 4$, and $\eta = 20$. And for the data case *medium* with $\lambda = 1$, $\epsilon = 1$, and $\eta = 4$ the results from the (extra trained) CNN simulation and benchmark simulation are about the same.

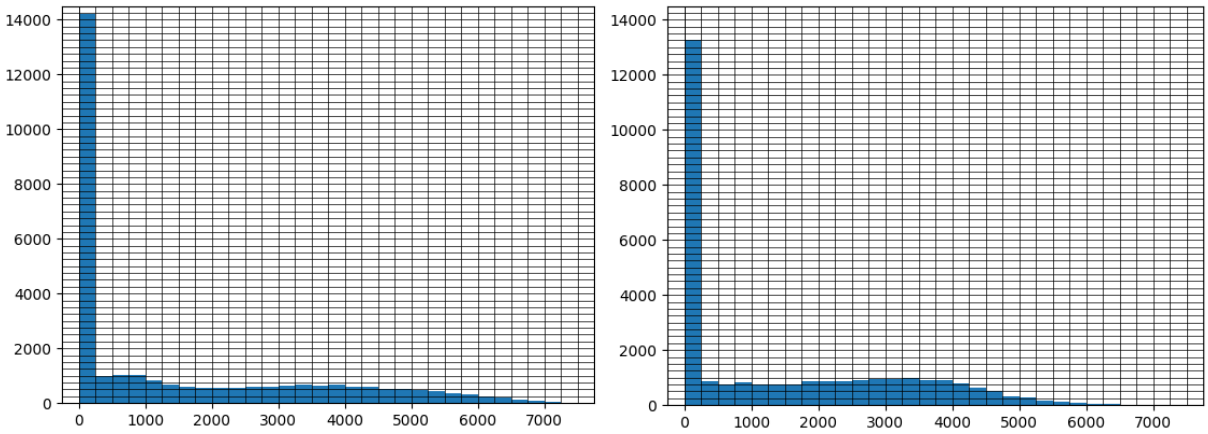
Table 19: Statistics on the delay of containers in seconds based on the given performance measure value λ, generated using the advanced simulation.

Data Case	λ	ϵ	η	Algorithm	Mean	St. dev.	Min	50%	Max
small	1	1	4	Benchmark	933	992	0	531	4100
				CNN	2395	2420	0	1639	8432
				Extra trained CNN	2011	2052	0	1410	6525
small	1	4	20	Benchmark	884	1051	0	299	6587
				CNN	1812	2116	0	637	8476
				Extra trained CNN	2399	2673	0	1068	11202
small	1	4	52	Benchmark	873	1053	0	271	7073
				CNN	2394	2681	0	1042	11989
				Extra trained CNN	2394	2681	0	1042	11989
small	24	1	4	Benchmark	933	992	0	531	4100
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
small	24	4	20	Benchmark	884	1051	0	299	6587
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
small	24	4	52	Benchmark	873	1053	0	271	7073
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
small	168	1	4	Benchmark	933	992	0	531	4100
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
small	168	4	20	Benchmark	884	1051	0	299	6587
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
small	168	4	52	Benchmark	873	1053	0	271	7073
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
medium	1	1	4	Benchmark	1692	1948	0	788	7338
				CNN	1694	1700	0	1231	7347
				Extra trained CNN	1694	1700	0	1231	7347
medium	1	4	20	Benchmark	1517	1927	0	368	7622
				CNN	1485	1661	0	656	7669
				Extra trained CNN	1485	1661	0	656	7669
medium	1	4	52	Benchmark	1505	1934	0	329	7959
				CNN	-[2]	-[2]	-[2]	-[2]	-[2]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
medium	24	1	4	Benchmark	1692	1948	0	788	7338
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
medium	24	4	20	Benchmark	1517	1927	0	368	7622
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
medium	24	4	52	Benchmark	1505	1934	0	329	7959
				CNN	-[2]	-[2]	-[2]	-[2]	-[2]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
medium	168	1	4	Benchmark	1692	1948	0	788	7338
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
medium	168	4	20	Benchmark	1517	1927	0	368	7622
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
medium	168	4	52	Benchmark	1505	1934	0	329	7959
				CNN	-[2]	-[2]	-[2]	-[2]	-[2]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]

[1] NN did not provide accuracy score above 0.00%.
[2] Unable to finish simulation due to memory error.
[3] Not further looked into because simulation using CNN did not provide results.

The CNN and the extra trained CNN mostly provide the same results using a simulation. This is to be expected as the accuracy scores between the two models are also mostly the same, thus it is expected that the models have not changed. For the data case *small* with $\lambda = 1$, $\epsilon = 1$, and $\eta = 4$ we do expect to see different results between the CNN and the extra trained CNN. This is reflected in the outcomes of the simulation, where the extra trained CNN performs better. For the data case *small* with $\lambda = 1$, $\epsilon = 4$, and $\eta = 20$ the simulation results between the CNN and the extra trained CNN are also different, which means that the model has changed, though it is still able to provide the same accuracy score. However, in this case the extra trained CNN performs worse than the original CNN, with all delay statistics being larger.

Again we present the distributions of the delays of containers leaving the yard to get a more in depth analysis of the differences between the benchmark algorithm and the CNNs. We do this for the data case in which the CNNs seem to perform better than the benchmark algorithm, which is for data case *medium* with $\lambda = 1$, $\epsilon = 4$ and $\eta = 20$. In Figure 4a we present the histogram of these delays when using the benchmark algorithm. Most of the observations are between 0 and 250 seconds delayed. The next bin only has a very small amount of observations compared to the first, and the amount of observations per bin slowly decreases after this. In Figure 4b we present the histogram of the delays when using the (extra trained) CNN. Note that the extra trained and the regular CNN provide the same results for this data case and are thus represented with the same histogram. The largest peak of observations is still in the first bin, however the number of observations in this bin is lower. We see a similar pattern in the rest of the histogram as for the benchmark. However, the amount of observations in the later bins is lower, and the amount of observations in the bins between about 1500 and 4000 seconds is higher.



(a) Distribution using the benchmark algorithm. (b) Distribution using the (extra trained) CNN.

Figure 4: Histograms of the delays of containers leaving the yard in seconds for the data case *medium*, with $\lambda = 1$, $\epsilon = 4$ and $\eta = 20$. Based on the advanced simulation.

Next, we look at the statistics on the arrivals and departures of the containers as presented in Table 20. The most important statistic here is the number of reshuffles. Again, for three data cases the number of reshuffles is the lowest for the benchmark algorithm, in one case it is the lowest for the CNN, and in one case for the extra trained CNN. This provides the same analysis as based on the results presented in Table 19.

Summary advanced simulation

In conclusion, for the more advanced simulation we find that the convolutional neural networks have a performance that more closely resembles the performance of the benchmark algorithm. The accuracy scores of the CNNs are lower than those of the NNs in the simple simulation, and we find no improvement in training the CNN on extra data. The average delay of the containers is lower for one of the CNN models compared to the benchmark, and in a different case the average is about the same between the two models. We also find that the distribution of the delays of containers for the CNNs and the benchmarks are similar. Lastly, we also find that the amount of reshuffles for the two algorithms do not differ as much anymore as for the simple simulation.

Table 20: Results from advanced simulation, with statistics on the arrivals and departures of containers.

Data case	λ	ϵ	η	Algorithm	Import Arrivals	Import Departures	Export Arrivals	Export Departures	Reshuffles
small	1	1	4	Benchmark	746	582	824	858	2498
				CNN	743	582	824	858	3042
				Extra trained CNN	743	580	824	858	2168
small	1	4	20	Benchmark	4002	3973	4283	4308	14544
				CNN	4015	3989	4283	4308	14333
				Extra trained CNN	4012	3987	4283	4308	17635
small	1	4	52	Benchmark	10376	10299	11128	11160	37660
				CNN	10366	10299	11128	11160	45149
				Extra trained CNN	10366	10299	11128	11160	45149
small	24	1	4	Benchmark	746	582	824	858	2498
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
small	24	4	20	Benchmark	4002	3973	4283	4308	14544
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
small	24	4	52	Benchmark	10376	10299	11128	11160	37660
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
small	168	1	4	Benchmark	746	582	824	858	2498
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
small	168	4	20	Benchmark	4002	3973	4283	4308	14544
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
small	168	4	52	Benchmark	10376	10299	11128	11160	37660
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
medium	1	1	4	Benchmark	2457	1764	2646	2710	7934
				CNN	2426	1722	2646	2710	9465
				Extra trained CNN	2426	1722	2646	2710	9465
medium	1	4	20	Benchmark	12870	12677	13357	13470	47720
				CNN	12833	12583	13357	13470	55165
				Extra trained CNN	12833	12583	13357	13470	55165
medium	1	4	52	Benchmark	32848	32932	35011	35091	123932
				CNN	-[2]	-[2]	-[2]	-[2]	-[2]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
medium	24	1	4	Benchmark	2457	1764	2646	2710	7934
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
medium	24	4	20	Benchmark	12870	12677	13357	13470	47720
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
medium	24	4	52	Benchmark	32848	32932	35011	35091	123932
				CNN	-[2]	-[2]	-[2]	-[2]	-[2]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
medium	168	1	4	Benchmark	2457	1764	2646	2710	7934
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
medium	168	4	20	Benchmark	12870	12677	13357	13470	47720
				CNN	-[1]	-[1]	-[1]	-[1]	-[1]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]
medium	168	4	52	Benchmark	32848	32932	35011	35091	123932
				CNN	-[2]	-[2]	-[2]	-[2]	-[2]
				Extra trained CNN	-[3]	-[3]	-[3]	-[3]	-[3]

[1] NN did not provide accuracy score above 0.00%.
[2] Unable to finish simulation due to memory error.
[3] Not further looked into because simulation using CNN did not provide results.

6 Conclusion

In this paper we looked at how yard planning strategies can be improved using Artificial Intelligence (AI). We investigated this by comparing a benchmark algorithm as implemented and provided by Konecranes with three neural networks (NNs). The first NN is a regular NN. The second NN is a convolutional NN (CNN), this means that the data can be provided as an 3D-array and the NN contains a convolutional layer to deal with this data. The last model is the same as the original CNN, however it has been trained on extra data. The comparisons between the models were made using two simulation models of a container terminal.

In the first simulation we compared the benchmark algorithm with the regular NN. We find that in 1/3 of the cases the NNs have an accuracy of 0%, meaning that it is unable to predict the delay. The accuracy scores of the NNs were the highest when predicting the delay of containers leaving in the next hour. When comparing the algorithms in a simulation we find that the average delay is always lower for the benchmark algorithm. The number of reshuffles is always larger for the NN. When the time period increases the difference between the number of reshuffles between the two algorithms increases.

Next, we adjust the simulation to be more realistic. We use this simulation to compare the two versions of the CNN with the benchmark algorithm. When training the CNN we find that it often has a low accuracy score. We tried to improve this score by providing it with extra data, however this mostly did not affect the accuracy. We compared the algorithms using a simulation and find that the average delay is mostly lower for the benchmark algorithm. However, there are some cases in which the CNN is able to perform the same, or slightly better than the benchmark. For the extra trained CNN we do not find a lot of differences compared to the regular CNN, because the model parameters mostly did not change. Based on the number of reshuffles we come to the same conclusions.

We find only one case in which a NN performs better than the benchmark algorithm. This is for the CNN in the second simulation. Though the CNN only performs better in one case, we do find results close to the benchmark for this same combination of a more realistic simulation and a CNN. Our results show that although based on our current implementation a NN cannot yet replace the effectiveness of the benchmark, with possibly a more realistic simulation, a CNN may be able learn from the benchmark algorithm to eventually perform better.

Implementing Artificial Intelligence techniques into container slot selection in container yards is a relatively new subject for which there is not a lot of research available yet. Our paper provides new insights into the possibilities of neural networks for these problems. We find that a convolutional neural network combined with a more advanced simulation is able to provide results similar to the benchmark, which shows that the data from container terminals contains patterns that neural networks can learn from.

Our current results show that applying neural networks in practice is a possibility. A neural network can be used in container yard planning, however this model has to be created and trained on the actual data from a terminal, and it will be terminal specific. Besides, the performance of the neural networks in this paper are not always as efficient as the benchmark yet, and thus more research is needed before neural networks can be truly effectively implemented. It should be noted that the benchmark algorithm, as currently implemented by Konecranes, has been

used for a lot of years and is thus based on a lot of knowledge and experience. Therefore, it is not surprising that it has this level of performance.

As this paper provides a good starting point for further research into this subject we provide some options to improve the performance of our neural networks, or at least provide a more extensive analysis of the possibilities.

First, our current implementation of the simulation misses a lot of aspects that might make it more realistic. Providing the simulation with a more realistic setup and implementation might help the neural networks perform better as well. When the simulation is more realistic, there is more data for the neural network to learn from. This data will also be more realistic, meaning that it contains more of the patterns that can be observed in actual container terminals.

One way to make the simulation more realistic is by adding more uncertainty. All parts of a container terminal have an uncertain aspect, most of which are not implemented in this paper. Next, the simulation becomes closer to the situation in practice if more container types are implemented. In this paper we only looked at import and export containers, however types such as empty, reefers, and containers storing dangerous goods might provide their own patterns for a neural network to learn from. A last example of an addition to the simulation, is the addition of aspects which were left out or simplified in our version. Some examples of these aspects are the loading and unloading of containers from vessels in batches, picking up containers in order of weight, and having a limited amount of space in the transfer zones.

A second extension, which is proposed to improve the performance of the neural networks, is to generate more data sets to help the neural networks in their training process. These data sets can be provided by various algorithms, such that they include a more diverse set of data points.

Lastly, it should be noted that our current implementation is based on a specific type of yard, with a specific corresponding layout. It would be interesting to see how a change in layout, or a change in equipment would influence the performance of the algorithms. For example, how would the average delay be affected if we use rubber tired gantry cranes, or if we use a layout that is parallel, instead of perpendicular, to the quay.

References

20-foot container - dimensions, measurements and weight. (2023). https://www.icontainers.com/help/20-foot-container/#:~:text=Dimensions%20of%20the%2020%2Dfoot%20container&text=The%20dimensions%20of%20a%2020,wide%20x%207%27%2010%E2%80%9D%20high. (Accessed: 14-11-2023)

Anwar, M., Henesey, L., & Casalicchio, E. (2019). Digitalization in container terminal logistics: A literature review. In *27th annual conference of international association of maritime economists, athens* (pp. 1–25).

Belsare, S., Badilla, E. D., & Dehghanimohammadabadi, M. (2022). Reinforcement learning with discrete event simulation: The premise, reality, and promise. In *2022 winter simulation conference (wsc)* (pp. 2724–2735).

Boer, C., & Saanen, Y. (2012). Improving container terminal efficiency through emulation. *Journal of Simulation*, *6*, 267–278.

Chen, T. (1999). Yard operations in the container terminal-a study in the 'unproductive moves'. *Maritime Policy & Management*, *26*(1), 27–38.

Chollet, F., et al. (2015). *Keras.* https://keras.io. (Accessed: 03-08-2023)

Dekker, R., Voogd, P., & Van Asperen, E. (2007). Advanced methods for container stacking. *Container Terminals and Cargo Systems: Design, Operations Management, and Logistics Control Issues*, 131–154.

Devanga, A., Badilla, E. D., & Dehghanimohammadabadi, M. (2022). Applied reinforcement learning for decision making in industrial simulation environments. In *2022 winter simulation conference (wsc)* (pp. 2819–2829).

DeVore, R., Hanin, B., & Petrova, G. (2021). Neural network approximation. *Acta Numerica*, *30*, 327–444.

Duinkerken, M. B., Evers, J. J., & Ottjes, J. A. (2001). A simulation model for integrating quay transport and stacking policies on automated container terminals. In *Proceedings of the 15th european simulation multiconference* (pp. 909–916).

Fancello, G., Pani, C., Pisano, M., Serra, P., Zuddas, P., & Fadda, P. (2011). Prediction of arrival times and human resources allocation for container terminal. *Maritime Economics & Logistics*, *13*, 142–173.

Fechter, J., Beham, A., Wagner, S., & Affenzeller, M. (2019). Approximate q-learning for stacking problems with continuous production and retrieval. *Applied Artificial Intelligence*, *33*(1), 68–86.

Firooz, K. H., Lee, M., & Tavakoli, M. (2022, 12). Arrangement and placement of containers in a container terminal.
doi: 10.13140/RG.2.2.16346.62404

Guiliang, Z., & Lina, M. (2009). The module design of intelligent allocating storage space in container freight station. In *2009 first international workshop on database technology and applications* (pp. 333–336).

Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction* (Vol. 2). Springer.

Hirashima, Y., Furuya, O., Takeda, K., Deng, M., & Inoue, A. (2005). A new method for marshaling plan using a reinforcement learning considering desired layout of containers in port terminals. *IFAC Proceedings Volumes*, *38*(1), 318–323.

Hirashima, Y., Ishikawa, N., & Takeda, K. (2006). A new reinforcement learning for group-based marshaling plan considering desired layout of containers in port terminals. In *2006 ieee international conference on networking, sensing and control* (pp. 670–675).

Jin, X., Duan, Z., Song, W., & Li, Q. (2023). Container stacking optimization based on deep reinforcement learning. *Engineering Applications of Artificial Intelligence*, *123*, 106508.

Kim, B., Jeong, Y., & Shin, J. G. (2020). Spatial arrangement using deep reinforcement learning to minimise rearrangement in ship block stockyards. *International journal of production research*, *58*(16), 5062–5076.

Kim, K. H., & Kim, H. B. (1999). Segregating space allocation models for container inventories in port container terminals. *International Journal of Production Economics*, *59*(1-3), 415–423.

Kim, K. H., & Lee, J.-S. (2006). Satisfying constraints for locating export containers in port container terminals. In *Computational science and its applications-iccsa 2006: International conference, glasgow, uk, may 8-11, 2006, proceedings, part iii 6* (pp. 564–573).

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv:1412.6980*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90.

Lee, Y., & Hsu, N.-Y. (2007). An optimization model for the container pre-marshalling problem. *Computers & operations research*, *34*(11), 3295–3313.

Morales, E. F., & Escalante, H. J. (2022). A brief introduction to supervised, unsupervised, and reinforcement learning. In *Biosignal processing and classification using computational learning and intelligence* (pp. 111–129). Elsevier.

O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., Invernizzi, L., et al. (2019). *Keras-tuner.* `https://github.com/keras-team/keras-tuner`.

Rodrigue, J.-P. (2020). *Evolution of containerships.* Retrieved 02-05-2023, from `https://transportgeography.org/contents/chapter5/maritime-transportation/evolution-containerships-classes/`

Saanen, Y. A., & Valkengoed, M. V. (2005). Comparison of three automated stacking alternatives by means of simulation. In *Proceedings of the winter simulation conference, 2005.* (p. 1567-1576).

Salido, M. A., Rodriguez-Molins, M., & Barber, F. (2012). A decision support system for managing combinatorial problems in container terminals. *Knowledge-Based Systems*, *29*, 63–74.

Salido, M. A., Sapena, O., Rodriguez, M., & Barber, F. (2009). A planning tool for minimizing reshuffles in container terminals. In *2009 21st ieee international conference on tools with artificial intelligence* (pp. 567–571).

Seger, C. (2018). *An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing.* DiVA.

Sikorra, J., Bohlken, W., & Goes, M. (2021). Allocation of container slots based on machine learning. *Hhla. De*, 11–15.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929–1958.

Steenken, D., Voß, S., & Stahlbock, R. (2004). Container terminal operation and operations research-a classification and literature review. *OR spectrum*, *26*, 3–49.

Taddy, M. (2018). The technological elements of artificial intelligence. In *The economics of artificial intelligence: An agenda* (pp. 61–87). University of Chicago Press.

Tierney, K., & Voß, S. (2016). Solving the robust container pre-marshalling problem. In *Computational logistics: 7th international conference, iccl 2016, lisbon, portugal, september 7-9, 2016, proceedings 7* (pp. 131–145).

van der Ham, R. (2018). salabim: discrete event simulation and animation in python. *Journal of Open Source Software*, *3*(27), 767. Retrieved from `https://doi.org/10.21105/joss.00767` doi: 10.21105/joss.00767

van Valkengoed, M. P., Bierhuizen, I. Y. A. S. P., Nederland, T. N. T., & Vis, S. B. D. I. (2004). How passing cranes influence stack operations in a container terminal.

Wang, S., & Meng, B. (2007). Resource allocation and scheduling problem based on genetic algorithm and ant colony optimization. In *Advances in knowledge discovery and data mining: 11th pacific-asia conference, pakdd 2007, nanjing, china, may 22-25, 2007. proceedings 11* (pp. 879–886).

Weerasinghe, B. A., Perera, H. N., & Bai, X. (2023). Optimizing container terminal operations: a systematic review of operations research applications. *Maritime Economics & Logistics*, 1–35.

Winston, W. L. (2004). *Operations research: applications and algorithm.* Thomson Learning, Inc.

Wong, A., & Kozan, E. (2010). Optimization of container process at seaport terminals. *Journal of the Operational Research Society*, *61*, 658–665.

Wong, A. K.-s. (2008). *Optimisation of container process at multimodal container terminals* (Unpublished doctoral dissertation). Queensland University of Technology.

Xie, S., Zhang, T., & Rose, O. (2022). Real-time scheduling based on simulation and deep reinforcement learning with featured action space. In *2022 winter simulation conference (wsc)* (pp. 1731–1739).

Zhang, C., Wang, Q., & Yuan, G. (2023). Novel models and algorithms for location assignment for outbound containers in container terminals. *European Journal of Operational Research*, *308*(2), 722–737.

Zhen, L. (2014). Storage allocation in transshipment hubs under uncertainties. *International Journal of Production Research*, *52*(1), 72–88.

A Simulation algorithms

Event 3 Initialisation

1: **for** every vessel line **do**
2: Create a vessel with containers.
3: Determine arrival time of vessel and plan vessel arrival, Event 4.
4: Determine a list of containers that need to be loaded onto the vessel.
5: **for** Every container that needs to be loaded on the vessel **do**
6: Determine arrival time and plan gate arrival, Event 9.
7: **end for**
8: **end for**

Event 4 Vessel Arrival

1: **for** every container on the vessel **do**
2: Determine arrival time on quay, and plan quay arrival, Event 5.
3: **end for**

Event 5 Quay Arrival

1: Determine best module for the container using algorithm.
2: Determine arrival time and plan arrival at WS transferzone, Event 6.

Event 6 WS Transferzone Arrival

1: **if** ASC for this module is available **then**
2: Determine best slot in module for container using algorithm.
3: Determine arrival time and plan container placed, Event 11.
4: Determine departure time for container.
5: Plan import departure, Event 8.
6: **else**
7: Put the container in the movement queue for this module.
8: **end if**

Event 7 Export Departure

1: **if** ASC for this module is available **then**
2: Determine arrival time at WS transferzone.
3: Plan container placed, Event 11.
4: **else**
5: Put container in the movement queue for this module.
6: **end if**

Event 8 Import Departure

```
if ASC for this module is available then
    Determine arrival time at LS transferzone.
    Plan container placed, Event 11.
else
    Put container in the movement queue for this module.
end if
```

Event 9 Gate Arrival

```
Determine best module for the container using algorithm.
Determine arrival time and plan arrival at LS transferzone, Event 10.
```

Event 10 LS Transferzone Arrival

```
if ASC for this module is available then
    Determine best slot in module for container using algorithm.
    Determine arrival time and plan container placed, Event 11.
else
    Put the container in the movement queue for this module.
end if
```

Event 11 Container Placed

```
if queue for this ASC is not empty then
    if first container in queue needs to go in the yard then
        Determine best slot in module for first container in queue using algorithm.
        Determine arrival time and plan container placed, Event 11.
        Remove this container from the queue.
        if container is import then
            Determine departure time for container.
            Plan import departure, Event 8.
        else if container is export then
            Determine departure time for container.
            Plan export departure, Event 7.
        end if
    else if first container in queue is export removal then
        Determine arrival time at WS transferzone.
        Plan container placed, Event 11.
    else if first container in queue is import removal then
        Determine arrival time at LS transferzone.
        Plan container placed, Event 11.
    end if
else
    Make the ASC idle.
end if
```

B Hyper parameters of tuned neural network

Table 21: The hyperparameters found when tuning the neural network, for the data cases for which training data was available.

Data Case	λ	ϵ	η	N	M_0	$h_0(x)$	M_1	$h_1(x)$	M_2	$h_2(x)$	M_3	$h_3(x)$	M_4	$h_4(x)$	M_5	$h_5(x)$	α
small	1	1	4	4	12	relu	140	relu	52	relu	232	sigmoid					0,01
small	1	4	20	5	56	sigmoid	48	sigmoid	12	relu	12	relu	216	sigmoid			0,01
small	1	4	52	6	76	sigmoid	140	sigmoid	84	tanh	32	sigmoid	4	relu	4	relu	0,001
small	24	1	4	6	52	relu	28	tanh	80	sigmoid	60	sigmoid	224	relu	208	sigmoid	0,01
small	24	4	20	5	232	relu	144	tanh	136	tanh	192	relu	144	tanh			0,01
small	24	4	52	4	36	relu	132	sigmoid	248	sigmoid	224	relu					0,01
small	168	1	4	6	104	tanh	64	tanh	144	relu	136	tanh	60	sigmoid	4	relu	0,0001
small	168	4	20	5	48	sigmoid	232	relu	248	sigmoid	184	tanh	4	relu			0,0001
small	168	4	52	6	168	relu	164	relu	36	tanh	196	tanh	4	relu	4	relu	0,01
medium	1	1	4	5	164	tanh	120	sigmoid	232	relu	80	relu	20	tanh			0,01
medium	1	4	20	6	24	sigmoid	168	relu	240	sigmoid	164	relu	4	relu	4	relu	0,001
medium	1	4	52	5	208	sigmoid	148	tanh	160	tanh	252	tanh	4	relu			0,01
medium	24	1	4	4	208	tanh	148	tanh	148	sigmoid	204	tanh					0,01
medium	24	4	20	4	152	relu	144	relu	44	relu	20	tanh					0,0001
medium	24	4	52	4	84	relu	52	relu	196	relu	116	tanh					0,0001
medium	168	1	4	5	208	relu	20	sigmoid	116	tanh	188	tanh	4	relu			0,0001
medium	168	4	20	4	100	sigmoid	168	tanh	8	sigmoid	188	relu					0,01
medium	168	4	52	4	88	tanh	240	relu	208	relu	40	tanh					0,0001
large	1	1	4	6	152	sigmoid	48	relu	52	relu	44	sigmoid	116	tanh	192	tanh	0,01
large	24	1	4	4	72	relu	56	tanh	64	relu	100	tanh					0,01
large	168	1	4	4	152	tanh	172	sigmoid	12	relu	16	tanh					0,01

C Hyper parameters of convolutional neural network

Table 22: The hyperparameters found when tuning the convolutional neural network, for the data cases for which training data was available.

Data Case	λ	ϵ	η	N	M_0	$h_0(x)$	M_1	$h_1(x)$	M_2	$h_2(x)$	M_3	$h_3(x)$	M_4	$h_4(x)$	M_5	$h_5(x)$	α	F	$h_C(x)$
small	1	1	4	4	108	tanh	48	tanh	92	sigmoid	168	tanh					0.01	30	sigmoid
small	1	4	20	4	40	relu	132	relu	244	relu	32	sigmoid					0.01	22	tanh
small	1	4	52	4	12	sigmoid	224	tanh	92	relu	236	tanh					0.01	20	tanh
small	24	1	4	5	72	tanh	128	relu	8	tanh	184	sigmoid	244	tanh			0.01	24	tanh
small	24	4	20	4	8	sigmoid	72	relu	244	relu	64	relu					0.001	16	tanh
small	24	4	52	6	164	tanh	44	tanh	40	sigmoid	124	tanh	4	relu	4	relu	0.0001	16	relu
small	168	1	4	5	160	relu	84	sigmoid	64	sigmoid	236	relu	96	sigmoid			0.01	6	sigmoid
small	168	4	20	5	144	sigmoid	24	sigmoid	88	relu	40	tanh	4	relu			0.01	26	tanh
small	168	4	52	4	188	relu	204	sigmoid	84	sigmoid	116	relu					0.01	20	relu
medium	1	1	4	5	236	tanh	132	relu	248	relu	204	tanh	88	tanh			0.01	12	tanh
medium	1	4	20	6	104	sigmoid	4	relu	64	relu	32	relu	164	tanh	12	tanh	0.01	6	sigmoid
medium	24	1	4	6	188	tanh	144	relu	36	sigmoid	188	tanh	4	relu	4	relu	0.0001	2	tanh
medium	24	4	20	4	48	tanh	196	relu	144	sigmoid	108	relu					0.01	8	tanh
medium	168	1	4	5	136	sigmoid	184	relu	192	relu	208	tanh	236	sigmoid			0.01	26	relu
medium	168	4	20	5	8	sigmoid	188	relu	76	relu	64	tanh	4	relu			0.001	4	relu

D Code description

The code required to create our results consists of 10 files, which are all written in Python.

mainPenaltyFunctionVersion1.py: Is used to run the simple version of the simulation based on the penalty function. Thus, this file is used to generate the input data for the neural networks, and for the comparison in the simulation study.

mainPenaltyFunctionVersion2.py: Is used to run the advanced version of the simulation based on the penalty function. Thus, this file is used to generate the input data for the convolutional neural networks, and for the comparison in the simulation study.

mainNNSimulation.py: Is used to run the simple version of the simulation based on the neural networks. Thus, is used for the comparison in the simulation study.

mainNNSimulationVersion2.py: Is used to run the advanced version of the simulation based on the convolutional neural networks. Thus, is used to generate extra training data for the convolutional neural networks, and is used for the comparison in the simulation study.

neuralNetworkVersion1.py: Is used to train the neural networks with the data that is generated with the simple version of the simulation based on the benchmark algorithm.

neuralNetworkVersion2.py: Is used to train the convolutional neural networks with the data that is generated with the advanced version of the simulation based on the benchmark algorithm.

extraTrainingCNN.py: Is used to train the convolutional neural networks on the extra training data, given a starting model.

comparisonCNN.py: Is used to compare the convolutional neural network with the extra trained convolutional neural network on a new set of test data.

dataExportInformation.py: Is a supportive file for all simulations that provides methods on how to transform the data such that it can be added to the data files that are generated in the simulations.

penaltyValues.py: Is a supportive file for the simulations based on the penalty function. Provides the values of the various penalties, and supportive functions on how to calculate the penalties.