IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 24, 2019, accepted November 11, 2019, date of publication November 14, 2019,
date of current version December 2, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2953542

Global Stock Market Prediction Based on Stock Chart Images Using Deep Q-Network

JINHO LEE, RAEHYUN KIM, YOOKYUNG KOH, AND JAEWOO KANG
Department of Computer Science and Engineering, Korea University, Seoul 02841, South Korea

Corresponding author: Jaewoo Kang (kangj@korea.ac.kr)

This work was supported by the National Research Foundation of Korea under Grant NRF-2017R1A2A1A17069645 and Grant NRF-2017M3C4A7065887.

ABSTRACT We applied Deep Q-Network with a Convolutional Neural Network function approximator, which takes stock chart images as input for making global stock market predictions. Our model not only yields profit in the stock market of the country whose data was used for training our model but also generally yields profit in global stock markets. We trained our model only on US stock market data and tested it on the stock market data of 31 different countries over 12 years. The portfolios constructed based on our model's output generally yield about 0.1 to 1.0 percent return per transaction prior to transaction costs in the stock markets of 31 countries. The results show that some patterns in stock chart images indicate the same stock price movements across global stock markets. Moreover, the results show that future stock prices can be predicted even if the model is trained and tested on data from different countries. The model can be trained on the data of relatively large and liquid markets (e.g., US) and tested on the data of small markets. The results demonstrate that artificial intelligence based stock price forecasting models can be used in relatively small markets (emerging countries) even though small markets do not have a sufficient amount of data for training.

INDEX TERMS Artificial intelligence, finance, neural networks, stock markets.

I. INTRODUCTION

Predicting future stock prices has always been a controversial research topic. In "Efficient Capital Markets [1]," Eugene Fama argued that the stock market is highly efficient and the price always fully reflects all available information, which is referred to as the Efficient Market Hypothesis (EMH). He also maintained that technical analysis or fundamental analysis (or any analysis) would not yield any consistent over-average profit to investors. However, not all researchers agreed with EMH [2]–[4]. Some classical works in financial economics maintained that stock markets have anomalies and are profitable, which is inconsistent with EMH [5]–[10]. Also, there was some criticism of the argument that stock markets are inefficient and profitable. Studies that asserted stock markets are profitable and have anomalies were criticized for not considering all transaction costs [11]–[13]. EMH is still the subject of intense debate. Many following studies focused on demonstrating the profitability of stock markets. Some works have used technical analysis, which involves studying past stock prices and volumes, to predict future stock prices and demonstrated its profitability [14], [15]. Other studies, especially in the computer science field, have focused on discovering non-conventional signals that may help to predict the stock markets. Studies focused on discovering non-conventional signals have analyzed Web data such as Social Networking Service (SNS) messages [17], investors' sentiments [18], news articles [19], or search engine queries [20], [21]. These studies found that investors' sentiments from SNS platforms and search query frequency data provide useful information for predicting future stock prices.

One of the other approaches to predicting future stock prices in the computer science field is to build artificial intelligence based models that use machine learning techniques such as Neural Network (NN) [22] or Reinforcement Learning (RL) [23]. NN and RL are currently among the most commonly used machine learning methods. Many state-of-the-art methods in various domains such as natural language processing, image classification, and speech recognition are based upon Convolutional Neural Network (CNN) or Recurrent Neural Network (RNN) models. RL is also adopted in many

The associate editor coordinating the review of this manuscript and approving it for publication was Andrei Muller.

domains such as robotics and game playing. The goal of RL is to train an agent to choose the optimal action given the current state. But unlike supervised learning where exact answers are given to a model, an RL agent is trained to maximize cumulative rewards in the training process. A brief overview of recent studies that used artificial intelligence models for stock prediction is provided below. A more extensive review of the recent studies can be found in [24], [25].

Takeuchi and Lee [26] applied a feedforward neural network to a basic momentum strategy [7] to enhance prediction performance. The authors reported that the performance of their proposed model on individual US stock market prediction was better than that of the original momentum strategy. Krauss *et al.* [27] analyzed the performance of deep neural networks, gradient boosted trees, random forests, and ensembles of these methods in predicting the future prices of S&P500 stocks. Fischer and Krauss [28] found that Long Short Term Memory (LSTM) [29] networks outperformed deep neural networks and random forests. Deng *et al.* [30] proposed an RNN based RL model, and a training algorithm which addresses the gradient vanishing problem. The authors validated their model on Chinese stock index futures data, commodity futures data, and S&P500 index data. All the above mentioned works used historical price data as input. However, other previous works directly used stock chart images as input. Tsai and Quan [31] used candlestick chart images to predict the Dow Jones Industrial Average Index. The authors used a model based on a content image retrieval technique to automatically extract features from candlestick chart images. Guo *et al.* [32] proposed a CNN based model that predicts price movements based on candlestick charts. All of these studies have demonstrated that among various input variables, such artificial intelligence based models efficiently capture complex non-linear patterns associated with future returns. But most of the previous works mainly focused on building a high performance model optimized on a limited number of securities or composite indexes only in a single country using various input variables such as price, volume, and technical and other financial indicators. In their works, it was not determined whether the signals or patterns that were found to yield profit in a given country would yield profit in other countries. No previous work could design an NN model for emerging countries that usually do not have enough data for training complex NN models.

In our work, we mainly focus on finding patterns that generally yield a profit not only in a stock market of the single country whose data is used for training our model but also in global stock markets. For example, let us assume that our model learned some unique patterns from the training data of a single country, and the patterns indicate that the stock price will sharply go up. Then, we need to show that these unique patterns consistently indicate the same future stock price movement (that the stock price will rise) not only in the stock market of the country in which our model was trained but also in many others. Interestingly, the results show that the investment activities of people from different countries and cultures tend to be similar for certain price/volume patterns.

We adopt the framework of Deep Q-Network (DQN) [33], which solves the instability problem which is caused by using nonlinear function approximators with Q-learning [34]. It uses the following two methods to stabilize the training process: experience replay and parameter freezing. We use the same methods in our training process. Our model takes chart images of an individual company as input and chooses one action among Long, Neutral, or Short every day. It receives a positive or negative reward based on its action and the subsequent price change of a company. Our model is trained to select the action that will yield maximum cumulative rewards given chart images.

We use the framework of DQN as opposed to conventional supervised learning to effectively train our NN model using RL for stock prediction. Using RL (Q-learning) instead of conventional supervised learning for the stock prediction problem has various advantages. First, our model is trained using rewards. Since we are dealing with the stock price prediction problem, assigning binary labels (e.g., True or False) to actions is insufficient. For example, if a model decides to take a Long action, it is desirable to receive a 10.0 reward for a $+10\%$ subsequent price change and a 1.5 reward for $+1.5\%$. Only receiving True for both cases does not give any distinction between the two cases. Second, RL uses cumulative rewards, not just immediate rewards, to train an agent. In most stock price prediction problems, supervised learning models are trained to predict the price (or price change) of the next time step based on the information of the current time step. In supervised learning, it is quite difficult to consider the time steps following the next time step. But RL can efficiently handle this problem by maximizing cumulative rewards using information from not only the next time step but from all subsequent time steps. Finally, in Q-learning, a trained model can make use of an action value, which is the expected cumulative rewards of the corresponding action. So when training is done, our model not only knows which action to take but also can predict the amount of profit the action will yield, which enables us to distinguish strong patterns from weak ones. We use CNN as the function approximator so that our model can take chart images as input. Since few previous works used chart images as input in stock prediction [24], [25], we believe conducting further studies that directly use chart images as input is necessary. Also, CNN is known to obtain good performance in numerous tasks [22] and can be trained much faster than RNN or LSTM.

We conducted numerous experiments on global stock markets. For this work, only five years (Jan.2001-Dec.2005) of US individual stock data are used for training and our model is tested on the stock market data of 31 countries over 12 years (Jan.2006-Dec.2017) after the training period. We only used the US stock data instead of the stock data of all other countries to train our model to show that our model not only yields profit in the stock market of the country whose data is

used for training but also in other global markets. The main contributions of our paper are as follows. First, as shown by the experimental results, we found that there are some patterns in stock chart images, which not only yield profit in a single country but in most of the other counties as well. Second, unlike most of the previous works, our model does not need to be trained and tested on data from the same market or country. For example, it is possible to use US stock market data for training an NN model for Spain or Taiwan. This may help an artificial intelligence based stock price prediction models to be more widely used in emerging markets, some of which are inefficient and have an insufficient amount of data for training models. As the results show, even though our model is trained only on the US individual stock data, it generally yields a considerable amount of profit in other countries. Finally, to the best of our knowledge, our artificial intelligence based model, which is trained on the data of only a single country, is the first to obtain numerous testing results on global stock markets.

II. BACKGROUND

A. CONVOLUTIONAL NEURAL NETWORK

Deep learning and NNs are currently the most widely used machine learning methods for classifying highly non-linear patterns. CNN is one of the NN architectures that was successfully applied to image classification problems. Many state-of-the-art image classification models are based upon CNN architecture. Such models usually take 2D images as input with three color channels. The input is passed to multiple hidden layers. Typically, each hidden layer consists of convolutional layers followed by non-linearity and pooling layers. But in the last one or two hidden layers, usually fully connected (FC) layers are used with a softmax function. The final output is usually a one-hot vector that corresponds to the label of the input image. Note that in our work, we use CNN as a function approximator in the Q-learning algorithm.

B. Q-LEARNING

Q-learning is one of the most common RL algorithms. The goal of all RL algorithms is to enable an agent to learn optimal policies, or in other words, train an agent so that it is capable of choosing the action that would give maximum cumulative rewards in a given state. In Q-learning, an agent does not directly learn optimal policies; instead, an agent is trained to acquire the optimal action value which is the expected cumulative rewards of each action given the current state. So when training is done, the optimal policy of an agent is simply a greedy policy where an agent chooses the action with the maximum action value given the state. To obtain the optimal action value, an agent should iteratively update the action value using the Bellman Equation. An agent chooses action given the current state following behavior policy and observes reward and next state. Usually, in Q-learning, the ϵ -greedy policy is used as a behavior policy, where an agent either chooses a random action with probability ϵ or acts greedily.

C. DEEP Q-NETWORK

When state representation is simple, the original Q-learning algorithm is proven to converge at optimal behavior. But like in Deep Q-Network (DQN), if the current state is very complex and cannot be represented in table lookup form, one can use the function approximator to efficiently represent the state. The function approximator could be any type of function that maps raw state representations to actions. In our case, CNN is used as the function approximator for mapping a state representation (stock chart image) to a Long, Neutral, or Short action. But naively implementing a non-linear function approximator such as NN is known to be ineffective in real practice because the training process is unstable. DQN addresses this problem using the following two methods: experience replay and parameter freezing. Experience replay is a way to reduce correlations in the sequence of data by storing the latest $\boldsymbol{M}$ experiences (input data) in the memory buffer and sampling random batches from the memory buffer at every iteration to take the gradient step. The parameter freezing method temporarily freezes the target parameters during training. To reduce correlations with the target, two sets of parameters are maintained and the target network parameters are updated periodically.

III. METHOD

A. OVERVIEW

In this subsection, a brief overview of our model is provided. Fig. 1 illustrates how our CNN reads input and outputs action values for an individual company. The term action value refers to the expected cumulative rewards of an action. Fig. 1 (a) shows the architecture of our CNN. Fig. 1 (b) illustrates example of a $\boldsymbol{W}$ by $\boldsymbol{W}$ chart image at time $\boldsymbol{t}$ and time $\boldsymbol{t+1}$. For example, if $\boldsymbol{W}$ equals 8 as shown in this figure, our CNN reads input as an 8 by 8 matrix with all elements filled with 0 or 1. A single column in this matrix represents a single day. Elements filled with the color black corresponds to 1; otherwise, they are all 0. The top part of the matrix represents the relative value of the closing price and the lower half represents the relative value of the volume. Two rows in the middle of the chart are empty (has zero value) to help our CNN to distinguish price from volume. Fig. 1 (c) shows sequential chart of 39 consecutive days. In this figure, all price volume data are min-max normalized over 39 days for visualization. In other words, for price data, the highest price in 39 days is listed in the first row, and the lowest price is provided in the third row; however, this is only for visualization purposes. In our actual experiments, input data are min-max normalized over $\boldsymbol{W}$ days (horizontal size of chart), and are not min-max normalized over the entire experimental period.

As shown in Fig. 1, our CNN takes a $\boldsymbol{W}$ by $\boldsymbol{W}$ chart image as input at each time step $\boldsymbol{t}$, which shows the daily closing price and volume data of a single company over the last $\boldsymbol{W}$ days. At time $\boldsymbol{t}$, our CNN outputs two vectors with a length of 3:$\boldsymbol{\rho}$ and $\boldsymbol{\eta}$. Based on these vectors, the action (Long, Neutral, or Short) to take at time $\boldsymbol{t}$ is decided. Likewise,

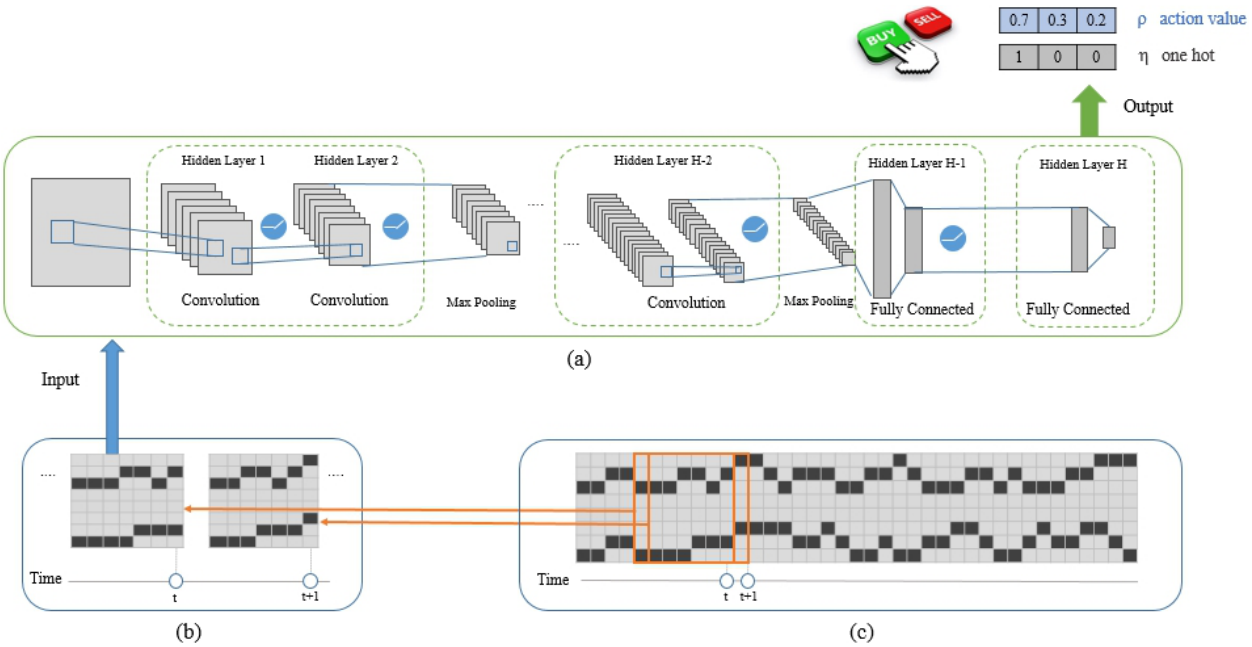

FIGURE 1. Overview of how our CNN reads an input chart of a single company at a specific time point (time t) and outputs the two vectors ρ and η.

at time $t + 1$ (or the next day), our CNN in Fig. 1 reads a stock chart image at time $t + 1$ and decides which action to take at time $t + 1$. The action value vector ρ represents an action value which is the expected cumulative rewards of an action (Long, Neutral, or Short). One hot vector η is marked as 1 in the same index where ρ has the maximum action value; otherwise, it is marked as 0. Each element of the vectors represents Long, Neutral, or Short action respectively. Thus, for example, the value of $\rho[3]$ at time t denotes the expected cumulative rewards if our CNN takes the Short action at time t. For simplicity, we standardized the index of all vectors in this paper to start from one. To sum up, the way in which our CNN operates is simple. It reads a chart at time t and chooses the action which has the maximum action value. At time $t + 1$, it receives reward based on the action at time t and the price change from time t to $t + 1$. It takes action at time $t + 1$ in the same way it does at time t.

B. NETWORK ARCHITECTURE

Our CNN takes $32 \times 32 \times 1$ as input. The input has only 1 channel because it does not need to be colored. The exact architecture of our CNN is as follows. Our CNN has six hidden layers. Thereby, H equals 6 in Fig. 1 (a). The first four hidden layers are convolutional layers followed by a Rectifier non-Linearity Unit (ReLU) and the last two hidden layers are FC layers. In the FC layers, ReLU is implemented only after the fifth layer. Each of the first four hidden layers consists of 16 filters of size $5 \times 5 \times 1$, 16 filters of size $5 \times 5 \times 16$, 32 filters of size $5 \times 5 \times 16$, and 32 filters of size $5 \times 5 \times 32$, respectively, all with stride 1, zero padding and followed by ReLU. Right after the second and fourth hidden layers, a max-pooling layer with a 2×2 filter and stride 2 is applied. The last two hidden layers are FC layers with 2048×32 and 32×3 parameters, respectively, followed by ReLU except for the final layer. The batch normalization [35] layer is added in every layer right before ReLU. The parameters are initialized using Xavier initialization [36]. The softmax function is not implemented since the output of our CNN is an action value, not a probability distribution between 0 and 1.

C. DATA DESCRIPTION

We collected daily closing price and volume data from Yahoo Finance. But Yahoo Finance does not provide the list of companies that can be download from the web site, we obtained the list of companies of roughly 40 countries including most of the developed markets from http://investexcel.net/all-yahoo-finance-stock-tickers/. Only for US, we used the list of companies of Russell 3000 index (The first half of 2018). We downloaded the adjusted closing price data to reflect events such as stock splits. Countries that did not have enough valid data were excluded. The data of 30 countries collected over 12 years and data of one country (US) collected over 17 years were downloaded. In each country, we also eliminated companies with noisy data. First, we eliminated companies that had no price data. Second, we also eliminated companies that had an excessive number of days with zero volume (we eliminated the companies that had zero volume for more than 25% of the entire testing period). Strictly speaking, many days of zero volume may not be considered as noise because a company's stocks may not be traded on some days or in some cases, stocks may be suspended for trading for a certain period. But stocks that have been

suspended for more than 25% of the entire testing period are definitely abnormal, and may indicate that the data of the given company is erroneous. Thus, excluding such companies does not undermine the validity of our work.

After downloading and eliminating noisy data, the entire dataset is divided into the training set and test set. The training set contains only US market data collected over a five-year period (Jan.2001-Dec.2005) from approximately 1500 companies that are included in the Russell 3000 Index (The first half of 2018). Only about half of the companies listed in the Russel 3000 index in the first half of 2018 had data from Jan. 2001 to Dec. 2005. Approximately 80% of the training set is actually used for training our model and about 20% is used for optimizing the hyperparameters. The test set contains data from 31 countries including the US, which was collected over a 12-year period (Jan.2006-Dec.2017). The test set is further divided into four-year intervals as follows: (2006-2010), (2010-2014), (2014-2018). Every four years, the top liquid N companies are selected from each country for the experiment. Values of 3000, 500, and 100 are initially assigned to N for US, developed countries, and emerging countries, respectively. The values are selected based on market capitalization and the number of available companies in each country. All the available companies were used if the number of valid companies were less than initial N value. In further experiments, we also tested our model on the data of the most liquid N_L companies from each country. Values of 1000, 200, and 40 are assigned to N_L for the US, developed countries, and emerging countries, respectively. The top N and N_L liquid companies are selected every four years based on the data collected over last 30 business days prior to the first day in each test set. For example, the top N and N_L liquid companies from Jan.2006 to Dec. 2009 were selected based on data collected from Nov. 15, 2005 to Dec. 31, 2005. Not all companies have all 12 years of data. The companies listed in Jan. 2010 have data starting from Jan. 2010. So companies that were listed in the exchange market for the entire four-year period were used for that testing period. In other words, all companies used in the testing period (Jan.2006-Dec.2009) were listed before Jan.2006 (strictly speaking, Nov. 15, 2005) and were still listed in the exchange market after Dec.2009.

In our experiments, we used daily closing price and volume data downloaded from Yahoo Finance, and we converted the raw data to input data as follows. A single input (corresponds to a single day of one company) consists of two parts: input chart S_t^c and scalar value L_t^c. The superscript c and subscript t indicate company c and time t, respectively. The input chart S_t^c is a W by W matrix in which all elements are either 0 or 1. The W by W matrix consists of the last W days of closing price and volume data of a single company. For example, the input chart of company c at time t contains closing price and volume data from time $t - W + 1$ to time t of company c. Like mentioned earlier, when closing price and volume are included in a chart, the values of closing price and volume are min-max normalized over last W days. The scalar value L_t^c represents the price change in percentage from time t to time $t + 1$. In other words, L_t^c is simply the daily return of company c from time t to time $t + 1$. In Fig. 1, chart S_t^c is shown as the only input to our CNN because the scalar value L_t^c is used with our CNN output to calculate reward. But in the actual training and test procedure, our CNN receives a W by W matrix of company c on time t as input S_t^c and outputs an action based on S_t^c. The reward for this action is calculated using the scalar value L_t^c. Equation (1) calculates L_t^c where Prc_t^c indicates the closing price of company c at time t.

$$L_t^c = 100 \times (Prc_{t+1}^c - Prc_t^c)/Prc_t^c \tag{1}$$

While generating L_t^c, we applied two simple methods for training our model. First, we bounded the values of L_t^c between -20% and 20% to prevent excessive rewards from noisy data. Though we tried to remove noisy data, there may still be some noisy data. Since (1) involves division, the value of L_t^c can easily change when an extremely small or potentially incorrect value is assigned to the closing price. By bounding the values of L_t^c, we could minimize the impact of such undesirable cases. In addition, we conducted more experiments with less tight bounds (50%, 100%) but there was no notable change in the results. Second, for the training set (not the test set), we neutralized the daily return L_t^c to address the data imbalance problem. In other words, the daily return averaged over the entire training set is subtracted from each daily return L_t^c. If we sample stock market data for a long period, the data usually becomes imbalanced because the market tends to go up. So the data usually has more positive values than negative values. Although the degree of imbalance in the stock market data is not that significant, we found that neutralizing the imbalance improves our training process. Table 1 summarizes the information about the dataset used in our experiments. The number of available companies from each country is listed in column Com#. The number of companies actually used in our experiments is listed in columns N and N_L. The total number of data used in our experiments is listed in column Data#. The column Avg lists the average daily return (in percentage) of the buy and hold portfolios of a given period. As shown in the first row of Table 1, the return average of the training set is 0 because we neutralized the training set. The column Std lists the standard deviations of daily returns. The column ExcessRate lists the percentage of data with the absolute value of L_t^c, which originally had a value larger than 20% before bounding.

Our training and test sets are in the form of a matrix. For example, the training set consists of $N \times T$ data points where N is $\approx$1500 and T is $\approx$1000 (number of business days in four years which is 80% of entire training set). The test set is formatted in the same way with different values of N and T.

D. TRAINING PROCESS

The standard Q-learning algorithm is based on the Bellman equation, and iteratively updates its action value based on the assumption that if an action value is optimal then it satisfies the Bellman equation. The Bellman equation defines the relationship between the current action value $Q(s, a)$

TABLE 1. Data statistics from 31 countries. The first row lists the training set that contains data collected over a 5-year period (2001-2006) from the US, and all other rows list the test set collected over a 12-year period (2006-2018).

Symbol	Country	Period	Com#	N	N_L	Data#	Avg	Std	ExcessRate
US	United States	2001-2006	2,792	1,534	-	1,876,082	0	2.9313	0.0021
US	United States	2006-2018	2,792	2,061	1,000	6,019,248	0.0537	2.7078	0.0016
AUS	Australia	2006-2018	960	485	200	1,421,920	0.0404	3.9144	0.0090
CAN	Canada	2006-2018	2,059	500	200	1,456,500	0.0134	3.427	0.0062
CHI	China	2006-2018	854	500	200	1,410,000	0.0976	3.0847	0.0004
FRA	France	2006-2018	3,605	500	200	1,484,000	0.0209	2.4629	0.0013
GER	Germany	2006-2018	4,639	500	200	1,475,000	0.0333	2.4497	0.0011
HK	Hong Kong	2006-2018	1,674	500	200	1,433,000	0.0257	3.2623	0.0033
IND	India	2006-2018	4,595	500	200	1,431,500	0.0521	2.8909	0.0008
KOR	South Korea	2006-2018	1,493	500	200	1,440,000	0.0438	3.2587	0.0006
SWI	Switzerland	2006-2018	463	169	169	495,736	0.027	2.3183	0.0033
TAI	Taiwan	2006-2018	1,484	475	200	1,360,534	0.0293	2.4316	0.0004
UK	United Kingdom	2006-2018	1,243	500	200	1,466,500	0.021	2.7824	0.0022
BRA	Brazil	2006-2018	392	94	40	270,327	0.0345	3.1094	0.0044
DEN	Denmark	2006-2018	99	85	40	246,942	0.026	2.6541	0.0024
FIN	Finland	2006-2018	103	75	40	218,495	0.0204	2.8638	0.0034
GRE	Greece	2006-2018	73	62	40	181,693	0.0188	3.8062	0.0076
MAL	Malaysia	2006-2018	764	100	40	288,400	0.0264	2.8811	0.0052
NET	Holland	2006-2018	98	69	40	206,174	0.0248	2.7585	0.0053
NOR	Norway	2006-2018	130	82	40	239,590	0.0139	3.3363	0.0038
SIG	Singapore	2006-2018	315	99	40	289,844	0.0304	2.6781	0.0046
SPA	Spain	2006-2018	136	87	40	259,116	0.0017	2.6124	0.0016
SWD	Sweden	2006-2018	488	100	40	292,600	0.029	2.4377	0.0014
TUR	Turkey	2006-2018	384	100	40	299,700	0.0506	2.6988	0.0009
AUR	Austria	2006-2018	38	30	30	85,717	0.0328	2.3076	0.0016
BEL	Belgium	2006-2018	96	80	40	238,650	0.022	2.2977	0.0022
IDO	Indonesia	2006-2018	274	100	40	285,400	0.0585	3.1439	0.004
IRL	Ireland	2006-2018	27	18	18	54,058	0.0381	3.0084	0.0029
ISR	Israel	2006-2018	224	100	40	285,100	0.0152	2.5997	0.0028
ITL	Italy	2006-2018	284	100	40	294,800	0.0051	2.4948	0.0012
POR	Qatar	2006-2018	30	25	25	74,364	0.0035	2.8408	0.0015
TAL	Thailand	2006-2018	399	100	40	285,000	0.0494	2.6075	0.0025

and the subsequent action value $\boldsymbol{Q(s', a')}$. The loss function is derived from this equation. Our training process uses the following two methods of DQN: experience replay and parameter freezing. Our loss function is defined in (2). We use the Adam optimizer [37] to perform a gradient step on $\boldsymbol{Loss(\theta)}$ with respect to parameters $\boldsymbol{\theta}$. For better understanding, the batch size is omitted in (2) so the loss function can be interpreted as loss calculated from a single experience.

$$Loss(\theta) = [r + \gamma \max_{a'} Q(s', a'; \theta^*) - Q(s, a; \theta)]^2 \quad (2)$$

where $\boldsymbol{s}$, $\boldsymbol{a}$, $\boldsymbol{r}$, $\boldsymbol{s'}$, and $\boldsymbol{a'}$ refer to current state, action, reward, subsequent state, and subsequent action, respectively, and $\boldsymbol{\gamma}$ denotes the discount factor. New symbols are used to maintain consistency with the standard Q-learning algorithm used in previous works. In Fig. 1, input chart and output action at time $\boldsymbol{t}$ correspond to state $\boldsymbol{s}$ and action $\boldsymbol{a}$, respectively. Likewise, input chart and output action at time $\boldsymbol{t+1}$ also refer to subsequent state $\boldsymbol{s'}$ and action $\boldsymbol{a'}$, respectively. As mentioned earlier, output $\boldsymbol{\rho}$ in Fig. 1, which is the output of our CNN, is the action value vector of each element which corresponds to each Long, Neutral, or Short action. The term $\boldsymbol{Q(s, a; \theta)}$ is a scalar value that represents the action value of action $\boldsymbol{a}$ given state $\boldsymbol{s}$ using our CNN parameterized by $\boldsymbol{\theta}$. Thus, given state $\boldsymbol{s}$, if action $\boldsymbol{a}$ is Short, then $\boldsymbol{Q(s, a; \theta)}$ exactly corresponds to output $\boldsymbol{\rho[3]}$ from our CNN parameterized by $\boldsymbol{\theta}$. Here, the network parameters $\boldsymbol{\theta}$ and target network parameters $\boldsymbol{\theta^*}$ are maintained throughout the training process to implement the parameter freezing method. Both $\boldsymbol{\theta}$ and $\boldsymbol{\theta^*}$ are randomly initialized with the same value in the beginning of the training stage. In the original version of the parameter freezing method, the optimizer performs a gradient step on $\boldsymbol{Loss(\theta)}$ with respect to the network parameters $\boldsymbol{\theta}$ at every iteration, and no gradient step is performed

with respect to the target network parameters θ^*. Target network parameters θ^* are only updated at every C iteration by copying parameters θ to θ^*.

Although our training algorithm is based on the standard Q-learning algorithm, our algorithm differs in the following ways. Unlike the standard Q-learning algorithm, our algorithm needs information about the previous action to calculate the current reward. Reward r_t^c is calculated as below. Superscript c and subscript t are added in (3) and denote company c and time t, respectively.

$$r_t^c = a_t^c \times L_t^c - P \times |a_t^c - a_{t-1}^c| \tag{3}$$

where r_t^c, L_t^c and a_t^c are reward, next day return, and action of company c at time t, respectively. Scalar value L_t^c in (3) and that in (1) are exactly the same term. Also, P denotes the transaction penalty. Our model assigns a value of 1, 0 or -1 to a_t^c for Long, Neutral, or Short actions respectively, for company c at time t. Thus, we can interpret the first term on the right side of (3) as the earned profit by choosing action given state. The second term on the right side of (3) refers to transaction costs when the model changes position at time t. Without some penalty, the model could change positions too frequently, which would incur high transaction costs in real practice. Equation 3 indicates the model needs to know the previous action a_{t-1}^c to calculate the current reward. The previous action a_{t-1}^c given the previous state is also chosen by implementing the ϵ -greedy policy. Unlike in the standard Q-learning method, in our method, the next state is not affected by the current action. Thus, when performing experience replay, our training algorithm needs to obtain the previous state and implement the ϵ -greedy policy to obtain the previous action.

Next, we modified the experience replay introduced in the previous work. First, our model not only samples random batches from the memory buffer to take a gradient step on the loss function but it also randomly generates an experience at every iteration to store it in the memory buffer. Second, our model updates parameters θ every B iteration, and not every iteration like the original version. In other words, our model stores an experience in the memory buffer at every iteration, updates the network parameters θ at every B iteration by taking a gradient step on the loss function, and updates the target network parameters θ^* at every $B \times C$ iteration by copying θ to θ^*. We modified the original version of experience replay to prevent our model from updating parameters θ for too many iterations with experiences generated from only a few companies. As mentioned earlier, we use 80% of our entire training set to actually train our model; our training set contains data on approximately 1500 companies, which was collected over 1000 days (total ≈1,500,000). The original version of experience replay generates experiences and stores them in the memory buffer by the order of input sequence (one company at a time). Assuming that the size of the memory buffer is 1000, the memory buffer has experiences from only one or two companies over the entire training period. It will take approximately 1000 iterations to observe an experience generated from a new company. Randomly generating experiences and taking a gradient step at every B iteration are done to help our model use many experiences uniformly generated from the entire training set.

The training algorithm generates experience e_b at b th iteration and stores it in the memory buffer. Experience is simply a tuple of the current state, action, reward, and subsequent state, i.e., (S, A, R and S'). The algorithm first randomly selects a data index (c and t) from the training set. Next, the ϵ -greedy policy (either randomly selects an action with probability ϵ or acts greedily) is used as the behavior policy on state s_{t-1}^c and s_t^c to obtain the previous action a_{t-1}^c and the current action a_t^c. When implementing the behavior policy, value ϵ is initialized to 1 and gradually decremented until it reaches the minimum value ϵ_m. Rewards are calculated based on previous and current actions and L_t^c, then the tuple (S, A, R and S') is assigned to experience e_b. The experience e_b is stored in the memory buffer. At every B iteration, the minibatch of size β is randomly sampled from the memory buffer and used to calculate $Loss$. A gradient step is taken to minimize $Loss$ with respect to parameters θ. The target network parameters θ^* are updated every $B \times C$ iteration. The full training algorithm is stated in Algorithm 1. Also, the list of hyperparameters mentioned in this paper is provided in Table 2.

E. SOURCE CODE AVAILABILITY

Our source code used in our experiments is available at https://github.com/lee-jinho/DQN-global-stock-market-prediction/. Since the converted data (chart image data) that was used as input data in our experiments is too large to be uploaded to the online repository, only the sample data is available. The entire dataset can be provided by the corresponding author upon request.

IV. EXPERIMENTS

A. PORTFOLIO CONSTRUCTION

Our CNN introduced in the previous section basically takes a single chart image as input and outputs an action value for a single company at time t. But in experiments, we have to deal with more than one company. To deal with more than one company, we constructed a length N portfolio vector α which satisfies $\sum_{c=1}^{N} |\alpha[c]| = 1.0$ based on the output vectors of our CNN. N is the total number of companies. At time t, our CNN produces the predictions for N companies instead of one. The portfolio is constructed based on the following N outputs: ρ_c and η_c where $1 \leq c \leq N$. Thus, the portfolio for N companies is reconstructed every day as done for a single company. The portfolio weight assigned to company c ($\alpha[c]$) represents the portion of the total asset that should be invested into company c. Vectors ρ_c and η_c also represent an action value and one hot vector for company c, respectively. Note that vector α can have a negative value, which means a Short position was taken on the company. For example, assuming that the total asset is 1.0, $\alpha[c] = -0.008$ indicates that our CNN is taking a 0.008 Short position on company c at time t. There may be multiple ways to assign weights to each

Algorithm 1 Training Algorithm

Initialize memory buffer to capacity M
Initialize network parameters θ
Initialize target network parameters $\theta^* = \theta$
Initialize $\epsilon = 1$
for all $b = 1$,*maxiter* **do**
  $c \leftarrow$ randomly chosen index
  $t \leftarrow$ randomly chosen index
  With probability ϵ, $a^c_{t-1} \leftarrow$ random action
  otherwise $a^c_{t-1} \leftarrow \max_a Q(s^c_{t-1}, a; \theta)$
  With probability ϵ, $a^c_t \leftarrow$ random action
  otherwise $a^c_t \leftarrow \max_a Q(s^c_t, a; \theta)$
  S $\leftarrow s^c_t$
  A $\leftarrow a^c_t$
  R $\leftarrow a^c_t \times L^c_t$ - P $\times |a^c_t$ - $a^c_{t-1}|$
  S' $\leftarrow s^c_{t+1}$
  Set $e_b \leftarrow$ {S,A,R,S'}
  Store experience e_b in memory buffer
  if $\epsilon > \epsilon_m$ **then**
    $\epsilon \leftarrow \epsilon \times 0.999999$
  end if
  if Memory buffer is full **then**
    Delete the oldest experience from the memory buffer
  end if
  if b % B == 0 **then**
    Random sample minibatch of size β from memory buffer
    Loss $\leftarrow 0$
    for all *k in minibatch* **do**
      Set $S_k, A_k, R_k, S'_k \leftarrow$ from e_k
      $Loss \leftarrow Loss + [R_k + \gamma \max_a Q(S'_k, a; \theta^*) - Q(S_k, A_k; \theta)]^2$
    end for
    $Loss \leftarrow Loss/\beta$
    Perform gradient step to minimize Loss with respect to the parameters θ
  end if
  if b % (B × C) == 0 **then**
    $\theta^* \leftarrow \theta$
  end if
end for

company even if we use the same output of our CNN. In our experiments, we used two methods that were used in previous works and real practice for creating portfolios. We conducted various experiments with the portfolios we created, which are described in the following subsections.

B. MARKET NEUTRAL PORTFOLIO

First, we use the market neutral portfolio which takes the same number of Long and Short positions every day. In other words, the portfolio satisfies $\sum_{c=1}^{N} \boldsymbol{\alpha_{n[c]}} = \mathbf{0}$ every day. The term ''neutralize'' could be interpreted as making the average zero. The subscript $\boldsymbol{n}$ is added to represent the market neutral portfolio. Because the market neutral portfolio takes the same number of Long and Short positions every day, its return has a much lower correlation with the average market return than the return of the non-neutral portfolio; hence, the portfolio is relatively free from overall market risk. In our experiments, we measured the correlation between our market neutral portfolio and the average market return. The results are shown in Table 3.

Algorithm 2 Market Neutral Portfolio

Initialize $\alpha_n \leftarrow 0$
for all c **do**
  if $\eta_c[1] == 1$ **then**
    $\alpha_n[c] \leftarrow 1$
  else if $\eta_c[3] == 1$ **then**
    $\alpha_n[c] \leftarrow -1$
  end if
end for
$\mu_n \leftarrow \frac{1}{N} \sum_{c=1}^{N} \alpha_n[c]$
$\alpha_n[c] \leftarrow \alpha_n[c] - \mu_n$ for all c
$\Sigma_n \leftarrow \sum_{c=1}^{N} |\alpha_n[c]|$
$\alpha_n[c] \leftarrow \alpha_n[c]$ / Σ_n for all c

One hot vector $\boldsymbol{\eta_c}$ is used when constructing the market neutral portfolio vector $\boldsymbol{\alpha_n}$. Steps to create the market neutral portfolio are described in Algorithm 2. First, for company $\boldsymbol{c}$, a scalar value of 1, 0, or -1 is assigned to $\boldsymbol{\alpha_n[c]}$ for Long, Neutral, and Short actions, respectively, based on vector $\boldsymbol{\eta_c}$. Then, the mean of vector $(\boldsymbol{\mu_n} = \frac{\mathbf{1}}{N} \sum_{c=1}^{N} \boldsymbol{\alpha_{n[c]}})$ is subtracted from each element of the vector $\boldsymbol{\alpha_n}$. Finally, each element of $\boldsymbol{\alpha_n}$ is divided by the sum of the absolute value of the vector's element $(\boldsymbol{\Sigma_n} = \sum_{c=1}^{N} |\boldsymbol{\alpha_{n[c]}}|)$ to make sure that the portfolio satisfies $\sum_{c=1}^{N} |\boldsymbol{\alpha_{n[c]}}| = \mathbf{1.0}$ every day.

To evaluate the performance of our market neutral portfolio, we conducted various types of experiments. The annual return and the return per transaction prior to transaction costs are reported in Table 3. Also, the risk of our market neutral portfolio is measured using the Sharpe ratio [38], [39] and the maximum drawdown [40], both of which are reported in Table 4. The Sharpe ratio is a commonly used measure and is calculated by dividing the mean of the excess return by the standard deviation of the excess return. We used the US 13-week Treasury bill for measuring the excess return of our market neutral portfolio. The maximum drawdown measures the portion of an asset that could be lost during an entire testing period. So the maximum drawdown is simply the loss from a peak to a trough of a portfolio. The exact formula can be found in [38]–[40].

These results in Table 3 and Table 4 show that our approach generally obtains good performance in most of the stock markets worldwide during most of the testing periods. Except for a few testing periods, our market neutral portfolio generally yields ≈ a 10 to 100 percent annual return

TABLE 2. List of hyperparameters mentioned in the paper. Hyperparameter optimization is done using 20% of the training set.

Hyperparameter	Description	Value
maxiter	The maximum number of iterations	5,000,000
learning rate	The learning rate used by Adam optimizer	0.00001
ϵ_m	The minimum value of ϵ	0.1
W	Horizontal and vertical size of input matrix	32
M	The capacity of the memory buffer	1,000
B	The update interval of parameters θ	10
C	The update interval of parameters θ^*	1,000
P	The transaction penalty while training	0.05
γ	The discount factor	0.99
β	The batch size	32

TABLE 3. Results of the market neutral portfolio on a 4-year interval prior to transaction costs. The column Acc lists the prediction accuracy averaged over an entire testing period (2006-2018). The column TR# lists the average number of transactions of each company per year. The per TR columns list the return per transaction in percentage. The Annual columns list the annual returns in percentage. The correlation between the returns of our market neutral portfolio and the average market returns are shown in the Cor columns. The last row lists the overall average of 31 countries.

			Period 2006 - 2010			Period 2010 - 2014			Period 2014 - 2018		
Symbol	Acc	TR#	per TR	Annual	Cor	per TR	Annual	Cor	per TR	Annual	Cor
US	0.503	89.75	0.21	20.64	-0.006	0.10	9.39	0.157	0.05	4.11	0.197
AUS	0.53	87.67	0.34	32.38	-0.001	0.19	18.13	0.059	0.12	11.35	0.071
CAN	0.52	79.83	0.9	106.54	0.111	0.85	100.39	0.286	0.72	75.99	0.21
CHI	0.503	84.38	0.92	120.95	-0.054	0.41	43.94	0.066	0.16	15.2	0.002
FRA	0.51	88.04	0.07	5.97	0.122	0	0.25	0.14	-0.02	-2.03	0.118
GER	0.51	89.17	0.22	21.65	-0.001	0.17	15.93	0.076	0.12	10.94	0.068
HK	0.527	83.62	0.52	55.25	-0.063	0.27	25.97	0.119	0.19	18.28	0.342
IND	0.5	86.88	0.09	8.61	0.32	0.02	1.37	0.308	0.01	0.88	0.165
KOR	0.517	84.96	0.06	5.45	0.367	0.05	4.51	0.286	0.07	6.01	0.347
SWI	0.523	84.0	0.51	54.75	0.063	0.4	37.93	-0.206	0.35	35.4	0.081
TAI	0.517	88.04	0.14	13.93	0.175	0.03	2.58	0.318	0.03	2.84	0.354
UK	0.51	81.46	0.16	13.72	-0.204	0.22	20.38	-0.058	0.12	10.83	0.136
BRA	0.513	87.5	0.86	110.43	-0.03	0.23	22.91	0.126	0.12	11.45	-0.011
DEN	0.523	88.12	0.52	58.33	0.073	0.59	68.27	0	0.26	26.16	0.108
FIN	0.533	90.21	0.84	113.96	0.082	0.74	95.47	0.005	0.44	47.03	0.082
GRE	0.527	87.29	0.37	39.79	0.122	0.79	103.85	0.14	0.83	95.87	0.171
MAL	0.53	84.92	0.71	84.43	0.163	0.45	47.15	0.151	0.42	42.64	0.217
NET	0.52	91.12	0.5	55.95	-0.035	0.58	68.36	0.1	0.32	33	0.061
NOR	0.523	89.12	0.42	45.13	0.051	0.59	69.08	0.053	0.3	30.32	0.135
SIG	0.527	86.33	1.05	147.61	0.113	0.32	32.59	0.099	0.29	28.52	0.181
SPA	0.517	88.96	0.16	14.87	0.096	0.36	36.53	-0.035	0.2	18.1	0.032
SWD	0.503	91.67	0.19	18.84	0.162	0.11	10.79	0.086	0.07	6.18	0.044
TUR	0.51	94.12	0.09	8.53	0.182	0.06	5.87	0.2	0.05	4.62	0.03
AUR	0.507	93.46	0.29	32.29	0.151	0.21	22.18	0.091	0.11	10.92	0.118
BEL	0.523	92.67	0.43	49.99	0.156	0.4	47.29	0.167	0.26	27.46	0.15
IDO	0.53	84.83	0.58	65.25	0.118	0.24	24.74	0.195	0.14	13.12	0.134
IRL	0.523	96.79	0.46	56.31	0.042	0.54	70.09	0.038	0.33	36.83	0.094
ISR	0.5	87.29	0.22	21.42	0.047	0.09	9.19	0.117	-0.06	-5.8	0.084
ITL	0.51	93.96	0.13	12.7	0.11	0.18	18.38	0.062	0.13	13.02	0.142
POR	0.527	96.21	0.18	18.56	0.134	0.55	72.03	0.09	0.48	56.82	0.039
TAL	0.523	89.04	0.34	36.74	0.17	0.12	11.85	0.162	0.22	23.32	0.072
Average	0.517	88.43	0.4	46.81	0.088	0.32	36.04	0.11	0.22	22.88	0.128

prior to transaction costs or has a Sharpe ratio of ≈2.0 to 10.0 in developed countries as well as in emerging countries. But in some countries where the number of candidate companies is very small, such as AUR and IRL, the maximum drawdown of our market neutral portfolio is larger than 10%.

TABLE 4. The SR columns list the annualized Sharpe ratios of our market neutral portfolio. The MDD columns list the maximum drawdowns of our market neutral portfolio. The maximum portfolio weight assigned to one company for each 4-year period is also listed in the MW columns.

	Period 2006 - 2010			Period 2010 - 2014			Period 2014 - 2018		
Symbol	SR	MDD (%)	MW	SR	MDD (%)	MW	SR	MDD (%)	MW
US	6.68	-1.2	0.0021	5.6	-1.25	0.0032	2.62	-1.3	0.002
AUS	6.73	-1.33	0.0082	6.65	-0.91	0.0107	5.53	-1.23	0.0114
CAN	19.41	-0.71	0.0068	22.04	-0.51	0.0067	19.16	-0.56	0.0052
CHI	17.36	-0.89	0.0051	11.58	-0.98	0.0063	5.08	-2.07	0.0064
FRA	1.23	-3.77	0.0132	-0.01	-4.18	0.0086	-0.82	-16.02	0.0078
GER	9.06	-0.89	0.0088	6.85	-0.97	0.0127	5.57	-0.88	0.01
HK	11.92	-1.21	0.0064	9.76	-1.58	0.0075	5.78	-4.64	0.0137
IND	1.99	-4.18	0.0122	0.43	-5.55	0.0088	0.13	-4.26	0.0082
KOR	0.75	-3.78	0.0065	1.9	-1.98	0.0074	2.59	-1.72	0.0065
SWI	10.17	-1.08	0.0203	9.9	-0.73	0.0304	10.27	-1.94	0.0192
TAI	3.1	-2.27	0.0091	1.4	-2.39	0.0091	1.69	-2.02	0.0073
UK	3.61	-2.72	0.0052	7.33	-0.73	0.0067	4.21	-2.17	0.0079
BRA	9.65	-2.12	0.0538	4.75	-1.88	0.0417	2.50	-3.35	0.0379
DEN	6.81	-3.44	0.0551	10.07	-1.16	0.042	6.01	-1.62	0.0648
FIN	10.63	-1.85	0.0738	11.54	-0.96	0.0693	8.14	-2.39	0.0451
GRE	4.41	-3.35	0.1318	7.83	-3.05	0.0881	7.61	-3.43	0.0714
MAL	9.57	-2.25	0.0403	9.25	-1.78	0.0354	7.91	-4.15	0.0283
NET	6.81	-2.44	0.082	9.03	-2.76	0.0788	6.10	-2.12	0.1305
NOR	5.04	-3.49	0.0712	8.04	-2.88	0.0724	4.83	-2.92	0.0389
SIG	13.2	-1.97	0.0471	8.42	-1.1	0.0431	7.71	-1.47	0.0354
SPA	2.36	-4.08	0.1173	6.56	-3.1	0.0383	3.72	-3.59	0.0541
SWD	3.1	-3.94	0.0653	2.86	-2.31	0.1011	1.90	-4.7	0.0517
TUR	1.13	-5.88	0.079	1.5	-7.25	0.0639	1.05	-5.65	0.0475
AUR	2.68	-13.89	0.2619	3.2	-5.11	0.339	1.71	-3.98	0.21
BEL	5.96	-4.44	0.0737	8.39	-1.58	0.0807	6.32	-1.26	0.0434
IDO	6.60	-5.39	0.0431	4.45	-2.05	0.0442	2.46	-6.82	0.0345
IRL	2.88	-11.18	0.5	4.17	-9.11	0.5002	3.12	-9.36	0.3426
ISR	2.79	-7.55	0.0398	1.69	-6.33	0.0646	-1.62	-22.26	0.0417
ITL	2.27	-4.38	0.0517	3.81	-2.48	0.0773	3.09	-3.02	0.0799
POR	1.68	-7.31	0.5	5.31	-8.88	0.2605	4.41	-6.9	0.1591
TAL	5.06	-1.93	0.0465	2.75	-5.83	0.0653	4.83	-2.55	0.0507

Also, the absolute value of the maximum portfolio weight assigned to one company during the testing period (the columns MW) is reported in Table 4. For example, the value 0.0021 in the first row of the column MW during the period of 2006-2010 indicates that the largest absolute value of the portfolio weight assigned to one company during the period of 2006-2010 is 0.0021, assuming that the total asset is 1.0. Thus, about 0.21 %, at most, of the total asset is assigned to one company during the 2006-2010 period in the US. As shown in the column MW in Table 4, except for a few countries with a small number of candidate companies, the market neutral portfolio distributes the asset to most of the candidate companies in most of the countries during most of the testing period. In other words, the profit of the market neutral portfolio does not come from a small number of extremely well performing companies.

Although our model is not designed for classification problems, we also calculated the prediction accuracy of our market neutral portfolio for further understanding. For simplicity, the action Neutral is ignored and prediction accuracy is calculated based on only Long or Short actions taken. Only when the next day's return is positive and the current action is Long or when the next day's return is negative and the current action is Short, the action is considered correct. Then the prediction accuracy is calculated by dividing the number of correct actions by the total number of correct and wrong actions. The results are provided in the column Acc of Table 3. As the results show, the prediction accuracy is slightly higher than 0.5 in most cases, even when the portfolio yields a considerably high annual return. These results indicate that our model focuses on the patterns that yield relatively high rewards, rather than on all input patterns that yield relatively low rewards.

C. TOP/BOTTOM K PORTFOLIO

Next, we use the top/bottom K portfolio, which takes a position only when the signal is strong. In other words, our CNN takes a Long position for the top $K\%$ of companies, a Short position for the bottom $K\%$ of companies, and a Neutral position for the others each day based on vector $\boldsymbol{\rho_c}$, which is another output of our CNN. Note that this portfolio also satisfies $\sum_{c=1}^{N} \boldsymbol{\alpha_{s[c]}} = \mathbf{0}$ (market neutral). The difference is that the top/bottom K portfolio distributes its asset to only $2 \times K\%$ companies. To construct this portfolio, first, subtract $\boldsymbol{\rho_c}[\mathbf{3}]$ from $\boldsymbol{\rho_c}[\mathbf{1}]$ and use this value to decide which company to take position. Note that each element of vector $\boldsymbol{\rho_c}$ represents the action value of corresponding action. If we take a closer look, the value $(\boldsymbol{\rho_c}[\mathbf{1}] - \boldsymbol{\rho_c}[\mathbf{3}])$ is the difference between the expected cumulative return of the Long action and the expected cumulative return of the Short action of company $\boldsymbol{c}$ at time $\boldsymbol{t}$. Intuitively, this value indicates how much the stock price of company $\boldsymbol{c}$ will increase at time $\boldsymbol{t+1}$. Based on this value $(\boldsymbol{\rho_c}[\mathbf{1}] - \boldsymbol{\rho_c}[\mathbf{3}])$, a value of 1.0 is assigned to $\boldsymbol{\alpha_s}[\boldsymbol{c}]$ for the top $K\%$ of companies (which have a bigger value) and -1.0 is assigned to $\boldsymbol{\alpha_s}[\boldsymbol{c}]$ for the bottom $K\%$ of companies (which have a smaller value). As done in the market neutral portfolio above, we divide each element of $\boldsymbol{\alpha_s}$ by the sum of the absolute value of the element of $\boldsymbol{\alpha_s}$ ($\boldsymbol{\Sigma_s} = \sum_{c=1}^{N} |\boldsymbol{\alpha_{s[c]}}|$) and use this as the top/bottom K portfolio. The subscript $\boldsymbol{s}$ is added to represent the top/bottom K portfolio. Steps to create the top/bottom K portfolio are described in Algorithm 3.

Algorithm 3 Top/Bottom K Portfolio

```
Initialize α_s ← 0
for all c do
    if ρ_c[1] − ρ_c[3] is in the top K% then
        α_s[c] ← 1
    else if ρ_c[1] − ρ_c[3] is in the bottom K% then
        α_s[c] ← −1
    end if
end for
Σ_s ← Σ_{c=1}^{N} |α_s[c]|
α_s[c] ← α_s[c] / Σ_s for all c
```

The main purpose of testing the performance of the top/bottom K portfolio is as follows. We used the Q-learning algorithm for training, which uses an action value that corresponds to the expected cumulative rewards of an action. In this sense, a larger action value of the Long (Short) position should indicate more profit the model will receive if the model takes a Long (Short) position. So if our CNN is trained properly, the top/bottom K portfolio that takes a position based on the subtracted value $(\boldsymbol{\rho_c}[\mathbf{1}] - \boldsymbol{\rho_c}[\mathbf{3}])$ should yield more profit than the market neutral portfolio that distributes asset to all companies. Thus, by this test, we are able to show that our CNN is not only capable of choosing the best action among Long, Neutral, and Short actions, but also can assign higher values to more profitable actions.

Table 5 compares the overall performance of the top/bottom K portfolio with the average annual market return. The results generally show that when the portfolio distributes more of its asset to a smaller number of companies that have larger action value, the annual return increases. Although the transaction costs are not included in the results, the results show that the annual return is much higher than the average market return in most countries for the majority of the time period. The average tendency is also shown in Fig. 2. Fig. 2 (a) shows the results of countries, with relatively large market capitalization, where an initial $\boldsymbol{N}$ value of 3000 or 500 is used in the experiments (12 countries total). Fig. 2 (b) shows the results of other countries (19 countries total), which were obtained by our model which used an initial $\boldsymbol{N}$ value of 100. The results clearly show that decreasing $\boldsymbol{K}$ increases the annual return.

D. STATISTICAL TESTS

For statistical tests, we constructed one random market neutral portfolio and three random top/bottom K portfolios. We compared the performance of random portfolios with that of our portfolios to verify the statistical significance of our results. Since the number of companies and the type of portfolio affect the standard deviation of the return on a portfolio, each of the four portfolios (neutral, $\boldsymbol{K} = 20, 10,$ and 5) was tested in the US, developed, and emerging countries (initial $\boldsymbol{N} = 3000, 500,$ and 100) over the entire testing period. 10,000 simulations were conducted for each experiment, and the mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma}$ of the annual return were calculated. Random portfolios were created as follows. A value between -1 and 1 was randomly selected, neutralized, and divided by the sum of absolute values to construct the random market neutral portfolio $\boldsymbol{\alpha_n}$ using the same method mentioned in the Market Neutral Portfolio section. So $\boldsymbol{\alpha_n}$ is a randomly weighted portfolio which satisfies $\sum_{c=1}^{N} \boldsymbol{\alpha_{n[c]}} = \mathbf{0}$ and $\sum_{c=1}^{N} |\boldsymbol{\alpha_{n[c]}}| = \mathbf{1.0}$. The random top/bottom K portfolios $\boldsymbol{\alpha_s}$ were also generated in a similar way. $K\%$ of randomly selected companies took a Long position and the other $K\%$ of randomly selected companies took a Short position. Like the portfolios of our model, the random market neutral portfolio and the three random top/bottom K portfolios were also reconstructed every day.

The return of a portfolio is generally assumed to be normally distributed, so we calculated Z-scores for the statistical test. The results are provided in Table 6. In Table 6, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ denote the mean and standard deviation of the annual return of the random portfolios, respectively. Since all the four random portfolios (neutral, $\boldsymbol{K} = 20, 10, 5$) are actually market neutral, $\boldsymbol{\mu}$ is zero. The standard deviation $\boldsymbol{\sigma}$ tends to increase as $\boldsymbol{K}$ and $\boldsymbol{N}$ become smaller; $\boldsymbol{K}$ and $\boldsymbol{N}$ are smaller when the portfolio distributes its asset to a smaller number of companies. The columns $\overline{\boldsymbol{\mu}}$ and Z-score list the average annual returns of our portfolios and Z-scores, respectively. As the results show, in most cases, the average annual returns of our portfolios are usually more than 30-$\boldsymbol{\sigma}$ away from the average annual returns of random portfolios.

TABLE 5. Comparison of the average annual return of the top/bottom K portfolios (K = 5, 10, 20) with the average annual market return (the columns avg) prior to transaction costs. The average annual market return is the return of the buy and hold portfolio with asset uniformly distributed to N companies.

	Annual Return (%) 2006 - 2010				Annual Return (%) 2010 - 2014				Annual Return (%) 2014 - 2018			
Symbol	K=5	K=10	K=20	avg	K=5	K=10	K=20	avg	K=5	K=10	K=20	avg
US	73.27	50.6	33.83	7.22	33.95	24.44	16.41	21.8	12.91	9.66	6.31	11.25
AUS	97.21	76.58	57.77	13.27	44.95	36.01	27.53	7.27	26.38	22.42	16.69	10.24
CAN	530.01	365.83	228.41	5.72	531.17	349.42	214.83	2.82	343.17	239.42	153.84	1.61
CHI	652.78	427.44	257.74	52.79	158.47	114.31	79.19	0.44	48.86	36.33	25.28	20.35
FRA	8.72	8.15	6.56	-1.07	-2.99	-2.28	-1.12	8	-10.34	-6.46	-3.29	8.88
GER	67.91	52.69	38.88	2.03	53.39	38.83	26.07	12.71	36.08	26.63	18.88	10.41
HK	226.4	155.79	101.14	17.29	82.71	61.36	43.95	-0.07	58.37	44.98	32.38	2.19
IND	30.48	19.76	14.07	19.62	5.15	4.27	2.61	-4.27	1.56	1.56	2.05	24.23
KOR	20.87	15.25	9.58	13.16	10.15	9.27	7.67	9.63	16.59	12.38	9.62	10.28
SWI	235.31	156.86	101.19	2.37	109.87	84.32	67.53	7.52	129.31	94.43	62.09	10.24
TAI	37.51	32.9	25.33	21.08	2.65	4.2	3.54	1.18	10.16	6.96	3.62	1.8
UK	45.76	33.12	24.18	-0.62	61.68	47.43	35.22	12.35	29.23	25	18.22	4.21
BRA	526.32	375.28	229.75	21.67	96.84	67.06	38.36	-0.22	29.38	21.72	18.47	6.99
DEN	334.31	189.9	111.45	-3.56	274.01	197.13	131.86	8.72	87.98	62.96	45.46	12.75
FIN	550.63	386.36	235.23	0.54	524.55	306.07	199.99	4.99	148.24	126.2	84.45	9.2
GRE	108.82	95.54	66.36	-1.81	365.98	292.18	188.92	5.73	444.16	350.02	226.39	10.26
MAL	427.61	303.35	166.39	9.52	192.24	133.81	89.49	11.3	158.57	109.59	77.21	-0.8
NET	186.72	143.99	93.57	1.64	241.76	169.39	123.16	4.79	77.5	71.12	51.36	11.48
NOR	187.83	110.95	81.31	5.62	261.36	188.23	133.27	6.12	111.11	64.78	51.85	0
SIG	1,021.09	622.37	333.39	16.99	116.24	85.06	57.06	6.51	96.97	73.18	51.12	-0.37
SPA	40.29	25.69	22.17	-3.98	125.39	97.02	64.82	0.58	45.88	32.87	26.69	3.42
SWD	57.75	45.08	36.44	3.52	30.29	21.48	15.96	9.21	19.5	15.79	11.25	9.2
TUR	21.68	19.18	16.26	9.82	15.63	12.69	10.14	9.59	2.59	6.98	6.53	18.75
AUR	101.34	80.69	52.04	4.12	44.42	38.15	31.16	10.79	56.89	30.42	19.84	8.93
BEL	225.81	142.14	92.87	-2.59	166.36	125.16	88.44	8.69	87.67	69.87	48.87	9.44
IDO	241.51	206.58	131.32	23.93	68.8	55.79	43.15	18.02	27.97	35.05	24.8	2.18
IRL	178.87	178.87	92.51	3.07	397.81	397.81	165.08	19.68	125.96	102.13	54.36	5.27
ISR	59.85	41.45	31.96	8.3	11.67	11.17	9.09	1.94	-7.28	-6.94	-4.34	1.15
ITL	31.77	24.07	18.17	-8.14	60.28	43.62	32.39	4	40.85	29.08	27.12	8
POR	26.46	41.66	34.75	-2.07	277.57	194.36	127.78	0.95	136.47	139.39	79.77	2.92
TAL	117.63	93.61	65.71	4.86	46.26	38.12	21.45	21.57	63.75	51.14	38.7	11.04
Average	208.79	145.86	90.66	7.88	142.21	104.71	67.58	7.49	79.24	61.25	41.47	7.92

E. CONSIDERING TRANSACTION COSTS

In this subsection, we describe the experiments where transaction costs are considered. In real practice, various types of transaction costs are applied when an asset is reallocated. Brokerage fees, transaction taxes (in some countries), or bid-ask spreads are some of the different types of transaction costs. However, since we are using the daily closing price data of 31 countries, considering all types of transaction costs is infeasible. Therefore, we conducted two experiments. In the first experiment, we assumed that the transaction cost is a certain percentage of the transaction amount. In the second experiment, we tested our portfolios on the most liquid N_L companies and compared the results with that of the previous experiments which used the most liquid N companies for testing our portfolios. Values of 1000, 200 and 40 are assigned to N_L for US, developed countries, and emerging countries, respectively. Table 1 shows the exact number of companies (N and N_L) from each country used in our experiments.

Before discussing the results, the transaction cost should be clearly defined. We define the transaction cost as the cost of a buy/sell transaction. For example, if a share is bought at \$100, and it is sold at \$110, a profit of \$9.9 instead of \$10 will be earned if the transaction cost is 0.1%. Note that transaction cost is different from the value P in Equation 3. Transaction penalty P works like transaction cost, but P is a fixed value applied only in training. The transaction cost in the experiments is denoted as ζ.

Table 7 summarizes the results of the first experiment. The column Type lists the types of portfolios (Neutral, $K = 20$, 10, 5). All 31 countries are categorized into three groups US, Dv and Em based on their market capitalization, where

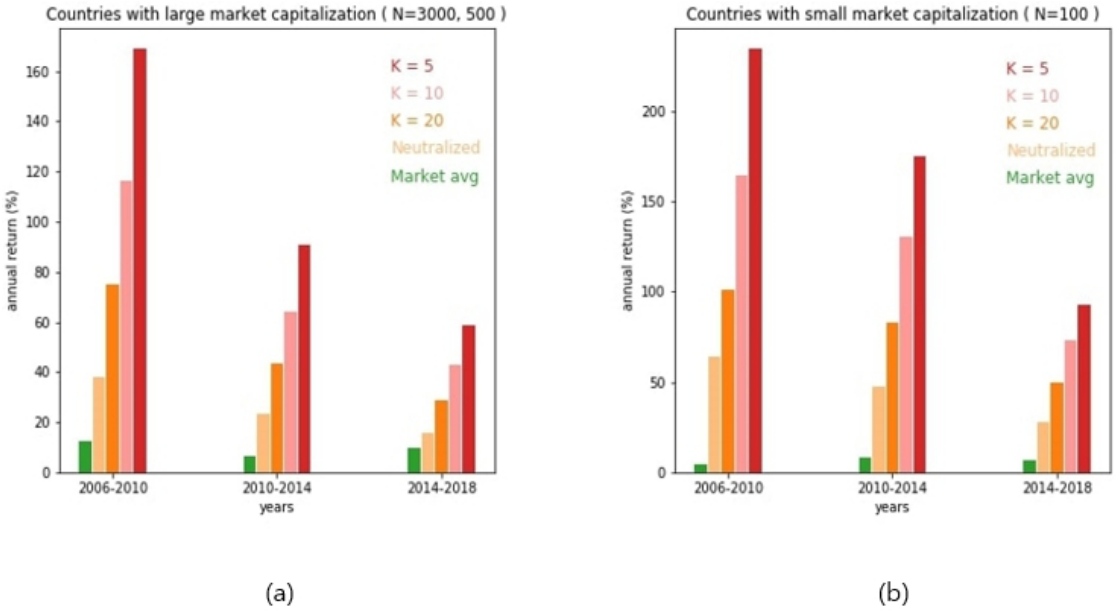

FIGURE 2. Comparison of the average annual returns of the market neutral portfolio and top/bottom K portfolio with the annual market average return.

TABLE 6. Statistical results of the random portfolios compared to our market neutral (NEU), top/bottom K portfolios. The results are calculated and averaged over the entire testing period (12 years).

	US				Developed Countries				Emerging Countries			
Portfolio	μ	σ	$\overline{\mu}$	Z-score	μ	σ	$\overline{\mu}$	Z-score	μ	σ	$\overline{\mu}$	Z-score
NEU	≈ 0	0.299	11.38	38.06	≈ 0	0.982	27.16	27.66	≈ 0	1.160	46.56	40.14
K=20	≈ 0	0.405	18.85	46.54	≈ 0	1.355	51.86	38.27	≈ 0	1.596	77.6	48.62
K=10	≈ 0	0.574	28.23	49.18	≈ 0	1.904	78.64	41.30	≈ 0	2.210	122.57	55.46
K=5	≈ 0	0.811	40.04	49.37	≈ 0	2.696	112.1	41.58	≈ 0	3.149	166.95	53.02

TABLE 7. The average annual returns of four types of portfolios (neutral, K = 20, 10, and 5) after transaction costs (ζ) are applied. Countries are divided into three groups (US, Dv, and Em) based on their market capitalization. The first row lists the testing periods.

		Return (06-10)			Return (10-14)			Return (14-18)			Return (06-18)		
Type	Group	ζ=0.1	ζ=0.2	ζ=0.3	ζ=0.1	ζ=0.2	ζ=0.3	ζ=0.1	ζ=0.2	ζ=0.3	ζ=0.1	ζ=0.2	ζ=0.3
NEU	US	10.38	1.03	-7.47	-0.4	-9.3	-17.39	-3.99	-11.37	-18.09	2	-6.55	-14.32
	Dv	28.53	18.1	8.56	14.57	5.34	-3.1	7.35	-1.35	-9.31	16.82	7.36	-1.28
	Em	39.07	27.15	16.28	31.48	20.07	9.69	16.48	6.58	-2.45	29.01	17.93	7.84
K=20	US	18.94	5.69	-6.11	3.67	-7.68	-17.79	-4.98	-15.05	-24.02	5.88	-5.68	-15.97
	Dv	59.67	42.77	27.72	30.74	17.04	4.84	17.07	4.79	-6.17	35.83	21.53	8.8
	Em	78.88	59.57	42.41	62.34	44.31	28.34	33.39	19.09	6.38	58.2	40.99	25.71
K=10	US	30.58	13.2	-1.89	7.99	-6.32	-18.74	-4.4	-16.62	-27.25	11.39	-3.25	-15.96
	Dv	94.17	69.76	48.51	47.25	29.21	13.47	27.75	12.02	-1.69	56.39	37	20.1
	Em	130	100.1	74.31	99.2	72.49	49.49	50.73	31.51	14.85	93.31	68.03	46.22
K=5	US	47.33	25.22	6.37	14.36	-2.35	-16.62	-4.06	-18.55	-30.91	19.21	1.44	-13.72
	Dv	138.3	104.8	76.22	69.19	46.05	26.17	40.71	21.88	5.73	82.73	57.58	36.04
	Em	185	143.4	108	133.1	98.06	68.44	64.31	40.5	20.25	127.47	93.99	65.56

the US, Dv, and Em groups have 1, 11, and 19 countries, respectively. The abbreviations Dv and Em correspond to developed and emerging countries, respectively. All the values in Table 7 are annual returns (in percentage) averaged over each testing period and Group. The transaction cost in percentage is denoted by ζ. For example, the value of 28.53 in the upper left corner is the annual return of the market neutral portfolio, averaged over 11 developed countries from 2006 to

TABLE 8. Comparison of the average annual returns of four types of portfolios (neutral, K = 20, 10 and 5). The columns N list average annual return of each portfolio tested on the most liquid N companies. The columns N_L list average annual return of each portfolio tested on the most liquid N_L companies. The first row lists the testing periods. The transaction cost is not applied.

		Return (06-10)		Return (10-14)		Return (14-18)		Return (06-18)	
Type	Group	N	N_L	N	N_L	N	N_L	N	N_L
NEU	US	20.64	7.48	9.39	0.48	4.11	-0.05	11.38	2.64
	Dv	39.93	18.76	24.67	10.8	16.88	8.82	27.16	12.79
	Em	52.16	32.29	44.03	27.97	27.35	17.97	41.18	26.08
K=20	US	33.83	8.97	16.41	2.49	6.31	-0.65	18.85	3.6
	Dv	78.62	31.17	46.09	17.04	30.85	13.87	51.85	20.69
	Em	100.6	54.25	82.71	48.11	49.47	28.69	77.6	43.68
K=10	US	50.6	11.75	24.44	2.54	9.66	-0.82	28.23	4.49
	Dv	122.2	45.92	67.92	21.74	45.79	19.07	78.64	28.91
	Em	164.6	87.37	130.2	80.38	72.91	44.8	122.57	70.85
K=5	US	73.27	16.72	33.95	4.71	12.91	-1.37	40.04	6.69
	Dv	177.5	63.05	96.11	27.48	62.67	24.93	112.11	38.49
	Em	234	126.9	174.6	104.3	92.32	58.07	166.98	96.43

2010 when a transaction cost of 0.1% is applied. As the results show, the returns of all the portfolios in Table 6 decrease as the transaction cost increases. In addition, the column Return(06-18) shows that the market neutral portfolio did not make a profit in the US ($\zeta = 0.2$) or in developed countries ($\zeta = 0.3$). But the top/bottom K portfolios yield a considerable amount of return in developed and emerging countries after a transaction cost of $\zeta = 0.3$ is applied. The top/bottom $K = 5$ portfolio yields more than the average market return in 23 countries when a transaction cost of $\zeta = 0.2$ is applied, and yields more than the average market return in 17 countries when a transaction cost of $\zeta = 0.3$ is applied.

The second experiment was conducted since the more liquid companies are relatively easy to trade at the desired time and price. Four types of portfolios were tested on the more liquid companies (N_L). Table 8 compares the results of using a top liquid N_L companies (columns N_L) with the results of using the original top liquid N companies (columns N). The data listed in columns N of Table 8 are obtained from Tables 3 and 5. The columns Type and Group in Table 8 list the types of portfolios and the categories of countries, respectively. Table 8 shows that the annual return decreases when the asset is allocated to only more liquid companies. The annual return of the portfolios may decrease more if the portfolios allocate the asset to even more liquid companies. The results show that it is more difficult for portfolios of more liquid companies to yield profit. But as the column Return(06-18) shows, our portfolio still yielded profits in the US, developed countries, and emerging countries. Our top/bottom $K = 5$ portfolio yields more profit than the market average in 28 countries.

F. COMPARISON WITH OTHER BASELINES

In this subsection, we compare our model with NN based and technical analysis based baselines. The NN baselines are as follows: FC network, CNN, and LSTM network all of which are trained using conventional supervised learning. We use FC network, CNN, and LSTM network as our baselines for two reasons. First, recent state-of-the-art studies [27], [28] have applied different types of NNs to the stock prediction problem and obtained good performance. Second, in the Introduction section, we stated that RL is more suitable for the stock prediction problem; hence, comparing the performance of FC network, CNN, and LSTM each trained using a conventional supervised learning method with that of our model is necessary. We also use the momentum strategy and mean average convergence and divergence (MACD), both of which are based on conventional technical analysis, as our baselines. A brief overview of the training process and architectures of the NN baselines is provided below.

We used the same data used in our previous experiments for training and testing FC network, CNN, and LSTM network. Five years (Jan. 2001-Dec. 2005) of US data are used for training (80%) and hyper-parameter tuning (20%). All other data is used for testing. FC network, CNN, and LSTM network are trained using supervised learning. Based on the next day's return, the training data is equally divided into the following classes: Long, Neutral, or Short. The entire training set is sorted based on the next day's return. The top 33.3% of training data with high returns are labeled as Long, the bottom 33.3% of training data with low returns are labeled as Short, and the others are labeled as Neutral. In the testing stage, evaluation metrics such as precision or recall, which are widely used in classification problems, were not used to evaluate the performance of the NN baselines. To compare the performance of the NN baselines with that of our model, we constructed a market neutral portfolio using the output of the NN baselines. Since the NN baselines are trained to classify the current input (closing price and volume data of the past W days) as Long, Neutral, or Short, the predicted label of the current input can be considered as the action that has to be taken at the current time.

TABLE 9. Performance comparison of our model and other baselines. The first row lists the testing periods. The column Methods lists the baselines and our model. MM_{12}, MACD, FCN, CNN, LSTM, and Ours denote momentum strategy, mean average convergence and divergence, fully connected network, convolutional neural network, LSTM network, and our model, respectively. All values are annual returns averaged over each period and Group. Group Avg in the column Group denotes the annual returns averaged over all 31 countries.

		Return (06-10)		Return (10-14)		Return (14-18)		Return (06-18)	
Group	Methods	ζ=0	ζ=0.1	ζ=0	ζ=0.1	ζ=0	ζ=0.1	ζ=0	ζ=0.1
US	MM_{12}	-13.19	-13.37	-0.42	-0.61	-1.76	-1.94	-5.12	-5.31
	MACD	-9.32	-11.17	1.23	-0.56	-2.62	-4.4	-3.57	-5.38
	FCN	15.73	-0.32	7.72	-7.47	4.93	-9.38	9.46	-5.72
	CNN	15.25	1.03	7.79	-5.89	4.52	-8.43	9.19	-4.43
	LSTM	3.55	-4.12	4.63	-3.18	3.6	-0.28	3.93	-2.53
	Ours	20.64	10.38	9.39	-0.4	4.11	-3.99	11.38	2
Dv	MM_{12}	-3.16	-3.34	9.8	9.62	3.99	3.81	3.54	3.36
	MACD	-0.39	-2.18	1.46	-0.31	-3.23	-5.03	-0.72	-2.51
	FCN	32.72	14.78	22	5.54	14.62	-0.73	23.11	6.53
	CNN	29.09	13.42	16.14	2.12	11.14	-2.29	18.79	4.42
	LSTM	-0.99	-5.17	4.17	-0.7	4.47	0.97	2.55	-1.63
	Ours	39.93	28.53	24.67	14.57	16.88	7.35	27.16	16.82
Em	MM_{12}	-4.36	-4.55	6.04	5.86	5.79	5.62	2.49	2.31
	MACD	-6.01	-7.85	-1.66	-3.5	-7.16	-9	-4.94	-6.78
	FCN	40.4	20.82	36.7	17.57	23.19	6.14	33.43	14.84
	CNN	32.15	15.5	27.23	11.13	17.2	2.47	25.53	9.7
	LSTM	-0.9	-5.59	1.91	-3.69	2.62	-1.44	1.21	-3.57
	Ours	52.16	39.07	44.03	31.48	27.35	16.48	41.18	29.01
Avg	MM_{12}	-4.22	-4.41	7.17	6.99	4.91	4.73	2.62	2.44
	MACD	-4.12	-5.95	-0.46	-2.27	-5.62	-7.44	-3.4	-5.22
	FCN	36.88	17.99	30.55	12.49	19.56	3.2	28.99	11.23
	CNN	30.52	14.3	22.67	7.38	14.64	0.43	22.61	7.37
	LSTM	-0.79	-5.39	2.8	-2.61	3.31	-0.55	1.77	-2.85
	Ours	46.8	34.4	36.04	24.45	22.89	12.58	35.24	23.81

We can construct a portfolio and evaluate the performance of the portfolio using the same method discussed in the previous Portfolio Construction and Market Neutral Portfolio sections.

The network architectures of FC network, CNN, and LSTM network are as follows. The FC network consists of four FC layers with 64×128, 128×64, 64×32, and 32×3, parameters respectively. Only the first three FC layers are followed by a batch normalization layer and ReLU. A softmax function is implemented after the final FC layer. A vector with a length of $2 \times \boldsymbol{W}$ is inputted to the FC network; the inputted vector consists of the min-max normalized closing price and volume data over the last $\boldsymbol{W}$ days. The LSTM network consists of one LSTM layer which is followed by a fully connected layer. The LSTM layer consists of 64 hidden units. The FC layer takes the last hidden vector from the LSTM layer (vector with a length of 64) as input and outputs a vector with a length of 3. A softmax function is implemented after the FC layer. The shape of the matrix inputted to the LSTM network is 2 by W, where each column corresponds to each time step of the LSTM network. Therefore, the min-max normalized closing price and volume data of each day is input at each corresponding time step of the LSTM network. The network architecture and the shape of the matrix inputted to the CNN are the same as those of our model, except that in the CNN, the softmax function is implemented after the final FC layer.

The conventional momentum strategy, which is introduced in [7], is a simple strategy of buying winners and selling losers. This strategy is used to rank all candidate companies based on past returns; it takes a Long position on the top 10% of companies and takes a Short position on the bottom 10% of companies. We tested the strategy in two different settings. In the first setting, companies are ranked based on the last 3 months, and the asset is reallocated every month. In the second setting, companies are ranked based on the last 12 months and the asset is reallocated every 3 months. Higher performance was obtained in the second setting, and only the results from the second setting are reported in Table 9. MACD [41] is one of the most commonly employed technical indicators and consists of the MACD line and the signal line. The MACD line is calculated by subtracting a 26 day-period moving average from a 12 day-period moving average. The signal line is a 9 day-period moving average of the MACD line. The most common MACD strategy used for trading stocks involves buying when the MACD line crosses above

TABLE 10. Average annual returns of portfolios that were created using different types of network architectures, which are listed in the column Architecture. The first row lists the testing periods. The annual returns are calculated using the market neutral portfolio and without considering transaction cost.

		Return (06-10)			Return (10-14)			Return (14-18)			Return (06-18)		
W	Architecture	US	Dv	Em	US	Dv	Em	US	Dv	Em	US	Dv	Em
32	$\boldsymbol{L_6 16 \times 32}$	20.64	39.39	52.16	9.39	24.67	44.03	4.11	16.88	27.35	11.38	27.16	41.18
32	L_6 8×16	11.49	19.11	17.5	5.21	11.98	17.18	3.81	7.48	11.79	6.84	12.86	15.49
32	L_6 32×64	11.16	21.32	24.66	6.07	12.9	20.42	3.13	9.63	16.01	6.79	14.62	20.36
32	L_4 16×32	19.45	34.07	40.26	10.48	19.86	35.6	4.58	11.76	19.57	11.5	21.9	31.81
32	L_4 32×64	20.31	28.76	40.55	8.03	18.44	36.09	3.16	11.83	21.31	10.5	19.68	32.65
16	L_6 16×32	20.45	34.73	42.9	10.74	20.85	38.05	4.98	15.38	25.83	12.06	23.65	35.59
8	L_6 16×32	17.17	40.19	51.84	9.56	25.69	43.38	4.97	17.48	28.29	10.57	27.79	41.17

the signal line and selling when the MACD line crosses below the signal line.

Table 9 summarizes the results of our experiments. The results of NN baselines and ours are obtained using the market neutral portfolio. Each baseline is tested with ($\zeta = 0.1$) and without ($\zeta = 0$) transaction cost and the resulting annual return is averaged over each group of countries (US, developed, and emerging) as done in the previous subsections. As the results show, the performance of our model is better than that of the NN baselines trained using supervised learning and conventional technical analysis. There are two main findings in this experiment. First, the performance of FC network and CNN is fairly high before transaction cost is included. But as the columns $\zeta = 0.1$ indicate, the profit significantly decreases when transaction cost ($\zeta = 0.1$) is applied. If a naive supervised learning method for classification problems is implemented, it could be difficult to control the number of transactions, and the model could be trained to excessively change positions. Second, although the performance of the NN baselines is affected by the number of layers, the model architecture, and other hyper-parameters, the experimental results show that RL is generally more suitable and efficient than conventional supervised learning for solving stock prediction problems.

G. ROBUSTNESS VERIFICATION

In this subsection, we test the robustness of our model. The market neutral portfolio was created and tested on the same data used in our previous experiments using different network architectures or different sizes of input ($W = 16, 8$). The results are shown in Table 10. The column W lists the input sizes. The column Architecture lists the symbol indicating different network architectures used in this experiment. Like in previous subsections, all values are averaged over the corresponding group of countries (US, developed and emerging). The exact description of the network architectures is provided below. $\boldsymbol{L_6 16 \times 32}$ is the architecture of our model, which is described in the Network Architecture section.

$\boldsymbol{L_6 8 \times 16}$ and $\boldsymbol{L_6 32 \times 64}$ each consist of four convolutional layers and are followed by two FC layers. The four convolutional layers of $\boldsymbol{L_6 8 \times 16}$ consist of 8 filters of size $5 \times 5 \times 1$, 8 filters of size $5 \times 5 \times 8$, 16 filters of size $5 \times 5 \times 8$, and 16 filters of size $5 \times 5 \times 16$, respectively, and are followed by two FC layers with 1024×16 and 16×3 parameters, respectively. The four convolutional layers of $\boldsymbol{L_6 32 \times 64}$ consist of 32 filters of size $5 \times 5 \times 1$, 32 filters of size $5 \times 5 \times 32$, 64 filters of size $5 \times 5 \times 32$, and 64 filters of size $5 \times 5 \times 64$, respectively, and are followed by two FC layers with 4096×16 and 16×3 parameters, respectively.

$\boldsymbol{L_4 16 \times 32}$ and $\boldsymbol{L_4 32 \times 64}$ each consist of two convolutional layers and are followed by two FC layers. The two convolutional layers of $\boldsymbol{L_4 16 \times 32}$ consist of 16 filters of size $5 \times 5 \times 1$ and 32 filters of size $5 \times 5 \times 16$, respectively, and are followed by two FC layers with 2048×32 and 32×3 parameters, respectively. The two convolutional layers of $\boldsymbol{L_4 32 \times 64}$ consist of 32 filters of size $5 \times 5 \times 1$ and 64 filters of size $5 \times 5 \times 32$, respectively, and are followed by two FC layers with 4096×64 and 64×3 parameters, respectively.

As the results in Table 10 show, changing the network architecture affects the overall performance of models. Also, naively increasing the number of parameters or layers may degrade the performance of our model. Even though the results in Table 10 show that changing the network architecture or input size does not drastically affect performance, using a suitable architecture and input size still helps to improve performance.

V. DISCUSSION

In this work, we applied DQN for making global stock market predictions. Unlike the previous works, we focused mainly on discovering patterns from stock chart images, which could yield a profit not only in a country whose stock data was used for training our model but also in other countries. We showed that the investment activities of people from different countries and cultures tend to be similar for certain price and volume patterns. The results also showed that the RL based DQN method is more suitable than conventional supervised learning methods for stock prediction problems.

We conducted further experiments to bridge the gap between our results from the experiment in the ideal circumstances and real practice. We conducted the experiments

where transaction costs were applied or the asset was distributed to a more liquid universe (N_L). Although increasing the transaction cost reduced the profit generated by our portfolios, our portfolios generally yielded a considerable amount of profit in many countries. Still, our work has some limitations. By increasing transaction cost, our result shows that our portfolios did not make a profit in some well developed countries such as the US. The profit also decreased when our portfolios distributed the asset to only a more liquid universe (N_L). Also, we were not able to consider bid-ask spreads or calculate the actual amount of asset could be managed, since we only have daily scale data.

VI. CONCLUSION

We conducted numerous experiments to determine whether our model trained on certain patterns in stock charts from a single country can make a profit not only in the given country but generally in all other countries. As our results show, our model trained in only the US market, also performed well or even better in many other markets for the 12-year testing period. Based on this observation, artificial intelligence and machine learning stock price forecasting studies, which have been conducted in only a single country so far, can be employed in global stock markets. In other words, if the model structure, input feature, and training procedure are satisfactory, the model does not have to be trained and tested in the same market. To the best of our knowledge, our artificial intelligence based model, which is trained on the data of only a single country, is the first to obtain numerous testing results on global stock markets.

REFERENCES

[1] E. F. Fama, "Efficient capital markets: A review of theory and empirical work," *J. Finance*, vol. 25, no. 2, pp. 383–417, May 1970.
[2] B. G. Malkiel, "The efficient market hypothesis and its critics," *J. Econ. Perspect.*, vol. 17, no. 1, pp. 59–82, 2003.
[3] N. Barberis and R. Thaler, "A survey of behavioral finance," *Handbook Econ. Finance*, vol. 1, pp. 1053–1128, Jan. 2003.
[4] J. H. Kim, A. Shamsuddin, and K. P. Lim, "Stock return predictability and the adaptive markets hypothesis: Evidence from century-long U.S. data," *J. Empirical Finance*, vol. 18, pp. 868–879, Dec. 2011.
[5] W. F. M. De Bondt and R. Thaler, "Does the stock market overreact?" *J. Finance*, vol. 40, no. 3, pp. 793–805, Jul. 1985.
[6] N. Jegadeesh, "Evidence of predictable behavior of security returns," *J. Finance*, vol. 45, no. 3, pp. 881–898, Jul. 1990.
[7] N. Jegadeesh and S. Titman, "Returns to buying winners and selling losers: Implications for stock market efficiency," *J. Finance*, vol. 48, no. 1, pp. 65–91, Mar. 1993.
[8] L. K. C. Chan, N. Jegadeesh, and J. Lakonishok, "Momentum strategies," *J. Finance*, vol. 51, no. 5, pp. 1681–1713, Dec. 1996.
[9] J. S. Abarbanell and B. J. Bushee, "Fundamental analysis, future earnings, and stock prices," *J. Accounting Res.*, vol. 35, no. 1, pp. 1–24, 1997.
[10] G. W. Schwert, "Anomalies and market efficiency," *Handbook Econ. Finance*, vol. 1, pp. 939–974, Jan. 2003.
[11] M. C. Jensen, "Some anomalous evidence regarding market efficiency," *J. Financial Econ.*, vol. 6, pp. 95–101, Sep. 1978.
[12] S. M. Phillips and C. W. Smith, Jr., "Trading costs for listed options: The implications for market efficiency," *J. Financial Econ.*, vol. 8, no. 2, pp. 179–201, Jun. 1980.
[13] H. R. Stoll and R. E. Whaley, "Transaction costs and the small firm effect," *J. Financial Econ.*, vol. 12, no. 1, pp. 57–79, Jun. 1983.
[14] A. W. Lo, H. Mamaysky, and J. Wang, "Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation," *J. Finance*, vol. 55, no. 4, pp. 1705–1765, Aug. 2000.
[15] C.-H. Park and S. H. Irwin, "What do we know about the profitability of technical analysis?" *J. Econ. Surv.*, vol. 21, no. 4, pp. 786–826, 2007.
[16] A. J. Patton, "Are 'market neutral' hedge funds really market neutral?" *Rev. Financial Stud.*, vol. 22, no. 7, pp. 2495–2530, Jul. 2008.
[17] J. Bollen, H. Mao, and X. Zeng, "Twitter mood predicts the stock market," *J. Comput. Sci.*, vol. 2, no. 1, pp. 1–8, Mar. 2011.
[18] D. Huang, F. Jiang, J. Tu, and G. Zhou, "Investor sentiment aligned: A powerful predictor of stock returns," *Rev. Financial Stud.*, vol. 28, no. 3, pp. 791–837, Mar. 2014.
[19] R. P. Schumaker and H. Chen, "Textual analysis of stock market prediction using breaking financial news: The azfin text system," *ACM Trans. Inf. Syst.*, vol. 27, no. 2, p. 12, Feb. 2009.
[20] Z. Da, J. Engelberg, and P. Gao, "In search of attention," *J. Finance*, vol. 66, no. 5, pp. 1461–1499, Oct. 2011.
[21] T. Preis, H. S. Moat, and H. E. Stanley, "Quantifying trading behavior in financial markets using Google trends," *Sci. Rep.*, vol. 3, Apr. 2013, Art. no. 1684.
[22] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.
[23] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Reading, MA, USA: MIT Press, 1998.
[24] G. Atsalakis and K. P. Valavanis, "Surveying stock market forecasting techniques—Part II: Soft computing methods," *Expert Syst. Appl.*, vol. 36, no. 3, pp. 5932–5941, Apr. 2009.
[25] R. C. Cavalcante, R. C. Brasileiro, V. L. F. Souza, J. P. Nobrega, and A. L. I. Oliveira, "Computational intelligence and financial markets: A survey and future directions," *Expert. Syst. Appl.*, vol. 55, pp. 194–211, Aug. 2016.
[26] L. Takeuchi and Y. Y. A. Lee, "Applying deep learning to enhance momentum trading strategies in stocks," Stanford Univ., Stanford, CA, USA, Working paper, Dec. 2013.
[27] C. Krauss, N. A. Do, and N. Huck, "Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the s&p 500," *Eur. J. Oper. Res.*, vol. 259, no. 2, pp. 689–702, Jun. 2017.
[28] T. Fischer and C. Krauss, "Deep learning with long short-term memory networks for financial market predictions," *Eur. J. Oper. Res.*, vol. 270, no. 2, pp. 654–669, 2018.
[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[30] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 653–664, Mar. 2016.
[31] C.-F. Tsai and Z.-Y. Quan, "Stock prediction by searching for similarities in candlestick charts," *ACM Trans. Manage. Inf. Syst.*, vol. 5, no. 2, p. 9, Jul. 2014.
[32] S.-J. Guo, F.-C. Hsu, and C.-C. Hung, "Deep candlestick predictor: A framework toward forecasting the price movement from candlestick charts," in *Proc. 9th Int. Symp. Parallel Architectures, Algorithms Program.*, pp. 219–226, Dec. 2018.
[33] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.
[34] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.
[35] S. Loffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," Feb. 2015, *arXiv:1502.03167*. [Online]. Available: https://arxiv.org/abs/1502.03167
[36] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. Int. Conf. Artif. Intell. Statist.*, May 2010, pp. 249–256.
[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," Dec. 2014, *arXiv:1412.6980*. [Online]. Available: https://arxiv.org/abs/1412.6980
[38] W. F. Sharpe, "Mutual fund performance," *J. Bus.*, vol. 39, no. 1, pp. 119–138, Jan. 1966.
[39] W. F. Sharpe, "The Sharpe ratio," *J. Portfolio Manage.*, vol. 21, no. 1, pp. 49–58, 1994.
[40] M. Magdon-Ismail, A. F. Atiya, A. Pratap, and Y. S. Abu-Mostafa, "On the maximum drawdown of a Brownian motion," *J. Appl. Probab.*, vol. 41, no. 1, pp. 147–161, Mar. 2004.
[41] G. Appel and E. Dobson, *Understanding MACD*. Greenville, SC, USA: Traders Press, 2008.

JINHO LEE received the B.S. degree in computer science and engineering from Korea University, Seoul, South Korea, in 2012, where he is currently pursuing the Ph.D. degree in computer science and engineering. His research focus on applying state-of-the-art machine learning algorithms to financial time series problems, such as stock market prediction problem.

RAEHYUN KIM received the B.S. degree from the Business School, Korea University, Seoul, South Korea, in 2018, where he is currently pursuing the Ph.D. degree in computer science and engineering. His research interests focus on stock market prediction and recommendation systems.

YOOKYUNG KOH received the B.S. degree in mathematics from Korea University, Seoul, South Korea, in 2017, where she is currently pursuing the M.S. degree in computer science and engineering. Her current research interests include deep learning applied to social media and recommendation systems.

JAEWOO KANG received the B.S. degree in computer science from Korea University, Seoul, South Korea, in 1994, the M.S. degree in computer science from the University of Colorado at Boulder, CO, USA, in 1996, and the Ph.D. degree in computer science from the University of Wisconsin–Madison, WI, USA, in 2003.

From 1996 to 1997, he was a Technical Staff Member with AT&T Labs Research, Florham Park, NJ, USA. From 1997 to 1998, he was a Technical Staff Member with Savera Systems Inc., Murray Hill, NJ, USA. From 2000 to 2001, he was the CTO and a co-founder of WISEngine Inc., Santa Clara, CA, USA, and Seoul. From 2003 to 2006, he was an Assistant Professor with the Department of Computer Science, North Carolina State University, Raleigh, NC, USA. Since 2006, he has been a Professor with the Department of Computer Science, Korea University. He also serves as the Department Head for the Interdisciplinary Graduate Program in Bioinformatics with Korea University. He is jointly appointed as a Professor with the Department of Medicine, School of Medicine, Korea University.