mathematics

MDPI

Article

A Comparative Study of Bitcoin Price Prediction Using Deep Learning

Suhwan Ji, Jongmin Kim and Hyeonseung Im *

Department of Computer Science, Kangwon National University, Chuncheon-si, Gangwon-do 24341, Korea; shji@kangwon.ac.kr (S.J.); jongmin.kim@kangwon.ac.kr (J.K.)

* Correspondence: hsim@kangwon.ac.kr; Tel.: +82-33-250-8441

Received: 12 July 2019; Accepted: 23 September 2019; Published: 25 September 2019

Abstract: Bitcoin has recently received a lot of attention from the media and the public due to its recent price surge and crash. Correspondingly, many researchers have investigated various factors that affect the Bitcoin price and the patterns behind its fluctuations, in particular, using various machine learning methods. In this paper, we study and compare various state-of-the-art deep learning methods such as a deep neural network (DNN), a long short-term memory (LSTM) model, a convolutional neural network, a deep residual network, and their combinations for Bitcoin price prediction. Experimental results showed that although LSTM-based prediction models slightly outperformed the other prediction models for Bitcoin price prediction (regression), DNN-based models performed the best for price ups and downs prediction (classification). In addition, a simple profitability analysis showed that classification models were more effective than regression models for algorithmic trading. Overall, the performances of the proposed deep learning-based prediction models were comparable.

Keywords: bitcoin; blockchain; cryptocurrency; deep learning; predictive model; time series analysis

1. Introduction

Bitcoin [1] has recently received a lot of attention from the media and the public due to its recent price surge and crash. Figure 1 shows the Bitcoin daily prices from 29 November 2011 to 31 December 2018 on Bitstamp (https://www.bitstamp.net/), which is the longest-running cryptocurrency exchange. On Bitstamp, the Bitcoin price reached the highest price (19,187.78 USD) on 16 December 2017 and has fallen up to 3179.54 USD (16.57% of the highest price) on 15 December 2018. Then, it has again increased with some fluctuations since April 2019. Although the Bitcoin price seems to follow a random walk [2], some recurring patterns seem to exist in the price fluctuations when considering the log value of the Bitcoin price, as shown in Figure 1.

As Bitcoin has been considered to be a financial asset and is traded through many cryptocurrency exchanges like a stock market, many researchers have investigated various factors that affect the Bitcoin price and the patterns behind its fluctuations using various analytical and experimental methods; for example, see the works by the authors of [3,4] and references therein. In particular, due to the recent advances in machine learning, many deep learning-based prediction models for the Bitcoin price have been proposed [5–11].

Although so far several deep learning methods were studied and compared for the Bitcoin price prediction, most previous work considered only a few deep learning methods, mostly based on a deep neural network (DNN) or a recurrent neural network (RNN) [12]. For example, a convolutional neural network (CNN) [13,14] and its variants, such as a deep residual network (ResNet) [15], have gained little attention for the Bitcoin price prediction, even though they were shown to be very effective for many applications, including long sequence data analysis [16]. Moreover, most previous work addressed only

a regression problem, where the prediction model predicts the next Bitcoin price based on the previous prices, but not a classification problem, where the prediction model predicts if the next price will go up or down with respect to the previous prices. More precisely, for a regression problem, the performance of prediction models is often measured in terms of the root-mean-square error (RMSE) or the mean absolute percentage error (MAPE) between the predicted values and the actual values, but a low RMSE or MAPE value does not necessarily mean that the prediction model is indeed effective; for instance, for Bitcoin trading, as it might not perform well for a classification problem, as shown in Section 4.

In this paper, we study and compare various state-of-the-art deep learning methods, such as DNNs, long short-term memory (LSTM) models [17], CNNs, ResNets, a combination of CNNs and RNNs (CRNN) [18], and their ensemble models for Bitcoin price prediction. In particular, we developed both regression and classification models by exploiting the Bitcoin blockchain information and compared their prediction performance under various settings. Experimental results showed that although LSTM-based prediction models slightly outperformed the other prediction models for regression problems, DNN-based prediction models performed the best for classification problems. In addition, to determine the applicability of the proposed prediction models to algorithmic trading, we compared the profitability of the proposed models by using a simple trading strategy. More specifically, for regression models, if the predicted price is higher than or equal to the current price, then we buy Bitcoin with all funds or hold it if we already spent all funds. Otherwise, we sell all Bitcoin or wait if we did not buy Bitcoin yet. Similarly, for classification models, we buy or hold if the prediction model predicts a price rise and otherwise sell or wait. The analysis result showed that classification models were more effective than regression models. Overall, the performance of the deep learning-based prediction models studied in this paper was comparable to each other.

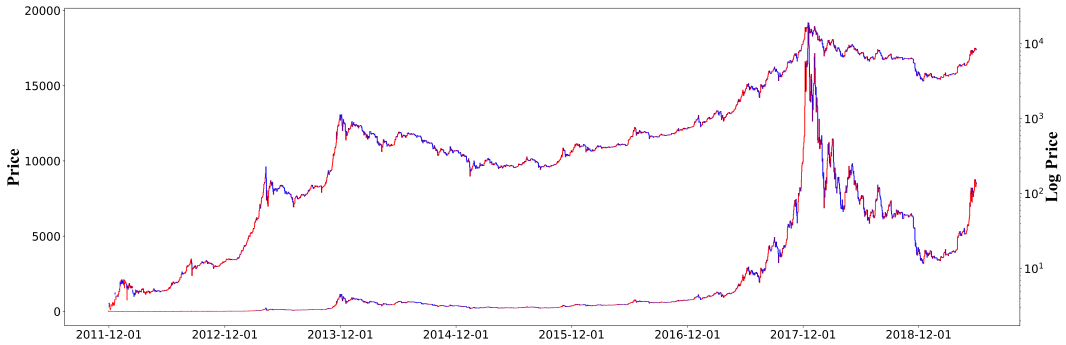


Figure 1. Bitcoin daily prices on Bitstamp (USD) from 29 November 2011 to 31 December 2018. The upper line shows log prices, whereas the lower line shows plain prices. Some recurring patterns seem to exist when considering the log value of the Bitcoin price.

1.1. Related Work

This section briefly reviews previous studies on Bitcoin price prediction using deep learning. For other statistical analysis, we refer the reader to the works by the authors of [3,4] and references therein. McNally et al. [5] proposed two prediction models based on recurrent neural networks (RNNs) and long short-term memory (LSTM), and compared them with an autoregressive integrated moving average (ARIMA) model [19], which is a traditionally widely used time series forecasting model. They developed classification models using the Bitcoin price information, which predict if the next Bitcoin price will go up or down based on the previous prices. In the work by the authors of [5], the RNN and LSTM models were shown to be better than the ARIMA model. In addition to the price information, Saad and Mohaisen [6] analyzed the Bitcoin blockchain information, such as the number of Bitcoin wallets and unique addresses, block mining difficulty, hash rate, etc., and used those features that are highly correlated to the Bitcoin price to build prediction models. They studied several regression models based on linear regression, random forests [20], gradient boosting [21], and neural networks. In addition to the blockchain information, Jang and Lee [7] further considered macroeconomic factors such as S&P 500, Euro stoxx 50, DOW 30, and NASDAQ, and exchange rates between major fiat currencies. They studied three prediction

models based on a Bayesian neural network (BNN) ([22], Chapter 5.7), linear regression, and support vector regression (SVR) [23], and showed that the BNN model outperformed the other two prediction models. In their follow-up study, Jang et al. [10] proposed a rolling window LSTM model and showed that it outperformed the prediction models based on linear regression, SVR, neural networks, and LSTM. Similarly, Shintate and Pichl [11] proposed a deep learning-based random sampling model and showed that it outperformed LSTM-based models. Meanwhile, Kim et al. [24] and Li et al. [25] considered social data to predict the Bitcoin price fluctuations. However, none of the work mentioned above considered CNN-based prediction models and various combinations of deep learning models. Moreover, unlike the previous studies, we provide in-depth comparisons between various deep learning models both for regression and classification problems, and show that the best prediction model for regression is not necessarily the best for classification and vice versa.

1.2. Organization of the Paper

Section 2 analyzes various features of the Bitcoin blockchain using the Spearman rank correlation analysis [26], and discusses those features that we used to build prediction models. Section 3 presents various prediction models based on deep neural networks (DNNs), LSTMs, CNNs, ResNets, CRNNs, and their ensemble models for Bitcoin price prediction. Section 4 presents experimental results under various settings. In particular, both regression and classification results of the proposed deep learning models are reported and analyzed. Finally, Section 5 concludes and discusses future work.

2. Datasets and Feature Engineering

In this study, we used the Bitstamp Bitcoin market (USD) time series data, which can be obtained from https://bitcoincharts.com/charts/. In particular, we used Bitcoin's prices from 29 November 2011 to 31 December 2018 (for a total of 2590 days), which are shown in Figure 1. During this period, the lowest price was 2.72 USD (29 November 2011) and the highest price was 19,187.78 USD (16 December 2017), which is 7054.33 times higher than the lowest price. Moreover, the Bitcoin blockchain information and its various statistics were also exploited, which can be obtained from https://www.blockchain.com/charts. Among various features, we considered the 29 features of the Bitcoin blockchain, listed in Table 1. The values of each feature were measured every day or converted to daily average values for this study.

Among the features listed in Table 1, we did not consider the features related to the Bitcoin memory pool such as mempool-count, mempool-growth, mempool-size, and trans-per-sec because their information is available only from 25 April 2016. Then, to find the most useful features, a correlation analysis between the rest of the features and the Bitstamp Bitcoin price was performed. In particular, we calculated the Spearman rank correlation coefficients [26] as shown in Figure 2 and Table 2. The Spearman rank correlation coefficient is a nonparametric measurement correlation, which can be used to evaluate the monotonic relationship between two variables. The Spearman correlation between two variables is the same as the Pearson correlation [27], i.e., linear correlation, between the ranked values of the two variables. A positive Spearman correlation coefficient between two variables, X and Y, indicates that when X increases (decreases), Y also tends to increase (decrease). In contrast, a negative coefficient indicates that when X increases (decreases), Y tends to decrease (increase). A Spearman correlation of zero indicates that there is no correlation between X and Y.

In this study, we excluded the features with a correlation coefficient less than 0.75, such as cost-per-trans-pct, est-trans-vol, med-cfm-time, output-vol, and trans-fees, as they are not so much related to the Bitcoin price. We also excluded the features with a correlation coefficient greater than 0.95, such as market-cap, market-price, and miners-revenue. Indeed, the trend in market-cap and market-price is almost the same as in the Bitstamp Bitcoin price. Note that market-price is the average Bitcoin USD market price across major Bitcoin exchanges, and market-cap is defined as market-price times the amount of Bitcoin supply in circulation. In addition, miners-revenue is directly proportional to the Bitcoin price and should be considered as a lagging indicator. Figure 3 shows the trends of some

features which are highly correlated or not correlated with the Bitcoin price. For example, the trend of trade-vol shown in Figure 3b (having a Spearman correlation of 0.8977) is very similar to that of the Bitstamp Bitcoin price shown in Figure 3a. In contrast, the trend of est-trans-vol shown in Figure 3e (having a Spearman correlation of −0.0595) has nothing to do with the Bitcoin price. In total, 18 features (including the Bitcoin price) were used to develop prediction models and to predict the next Bitcoin price. Among 18 features, the range of the values of difficulty, est-trans-vol-usd, hash-rate, my-wallets, trade-vol, and trans-fees-usd is very large. For these features, the difference between the minimum and the maximum values is from 2700 times to 8,000,000 times. Therefore, we used the log values of these features in the experiment.

Table 1. Bitcoin blockchain features.

Feature	Description
avg-block-size	The 24 h average block size in MB.
blockchain-size	The total size of all block headers and transactions.
cost-per-trans	Miners revenue divided by the number of transactions.
cost-per-trans-pct	Miners revenue as percentage of the transaction volume.
difficulty	A relative measure of difficulty in finding a new block.
est-trans-vol	The estimated value of transactions on the Bitcoin blockchain in BTC.
est-trans-vol-usd	The estimated USD value of transactions.
hash-rate	The estimated number of tera hashes per second the Bitcoin network is performing.
market-cap	The total USD value of Bitcoin supply in circulation.
market-price	The average USD market price across major Bitcoin exchanges.
med-cfm-time	The median time for a transaction to be accepted into a mined block.
mempool-count	The number of transactions waiting to be confirmed.
mempool-growth	The rate of the memory pool (mempool) growth per second.
mempool-size	The aggregate size of transactions waiting to be confirmed.
miners-revenue	The total value of Coinbase block rewards and transaction fees paid to miners.
my-wallets	The total number of blockchain wallets created.
n-trans	The number of daily confirmed Bitcoin transactions.
n-trans-excl-100	The total number of transactions per day excluding the chains longer than 100.
n-trans-excl-popular	The total number of transactions, excluding those involving any of the network's 100 most popular addresses.
n-trans-per-block	The average number of transactions per block.
n-trans-total	The total number of transactions.
n-unique-addr	The total number of unique addresses used on the Bitcoin blockchain.
output-val	The total value of all transaction outputs per day.
total-bitcoins	The total number of Bitcoins that have already been mined.
trade-vol	The total USD value of trading volume on major Bitcoin exchanges.
trans-fees	The total BTC value of all transaction fees paid to miners.
trans-fees-usd	The total USD value of all transaction fees paid to miners.
trans-per-sec	The number of Bitcoin transactions added to the mempool per second.
utxo-count	The number of unspent Bitcoin transactions outputs.

Table 2. Spearman correlation coefficients between the Bitcoin price and various features.

Bitcoin Feature	Coefficient	Bitcoin Feature	Coefficient
avg-block-size	0.8496	n-trans	0.8162
blockchain-size	0.9030	n-trans-excl-100	0.8469
cost-per-trans	0.7542	n-trans-excl-popular	0.8233
cost-per-trans-pct	−0.4874	n-trans-per-block	0.8070
difficulty	0.9033	n-trans-total	0.9030
est-trans-vol	−0.0595	n-unique-addr	0.8655
est-trans-vol-usd	0.9436	output-vol	0.1537
hash-rate	0.9033	total-bitcoins	0.9030
market-cap	0.9975	trade-vol	0.8977
market-price	0.9997	trans-fees	0.4227
med-cfm-time	0.0895	trans-fees-usd	0.9390
miners-revenue	0.9692	utxo-count	0.9067
my-wallets	0.9030		

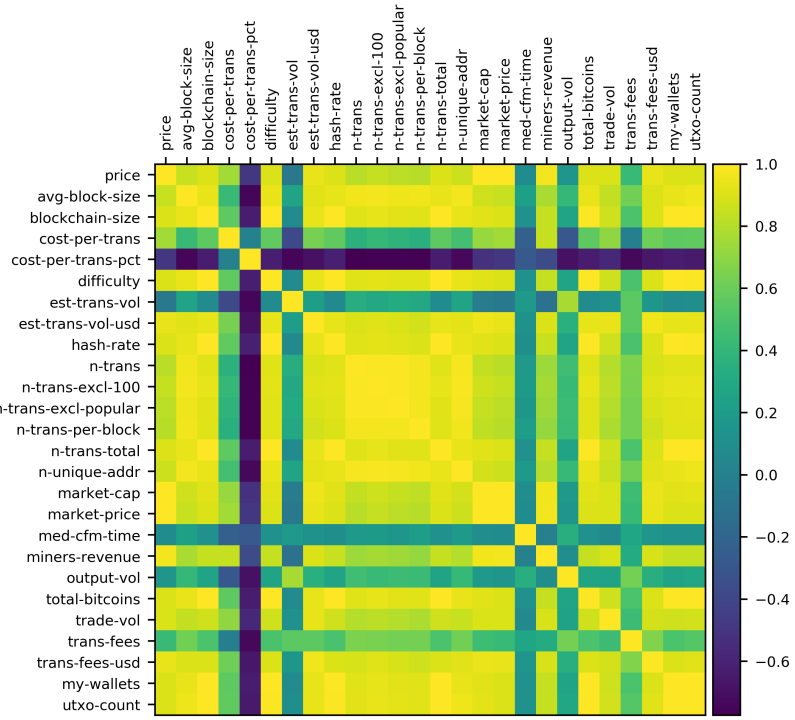
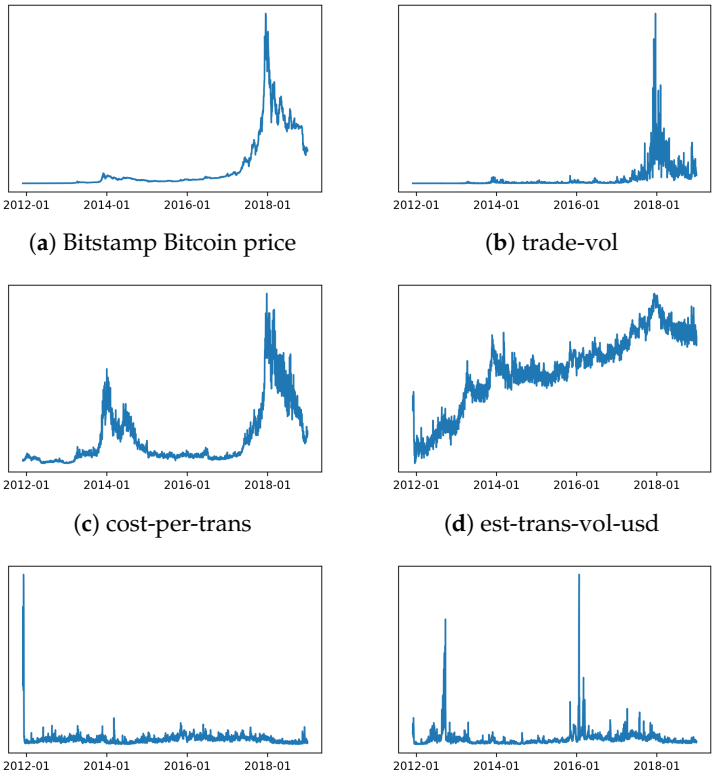


Figure 2. Spearman rank correlation coefficient matrix between Bitcoin blockchain features.



(**a**) Bitstamp Bitcoin price (**b**) trade-vol

(**c**) cost-per-trans (**d**) est-trans-vol-usd

(**e**) est-trans-vol (**f**) output-vol

Figure 3. Changes in the values of selected features.

3. Methods

This section describes data preprocessing and proposes various deep learning-based prediction models for Bitcoin price regression and classification problems. Table 3 summarizes the notation used in this paper.

Table 3. List of notations.

Notation	Definition
S	A raw dataset
$S[i:j]$	The sequence from $S[i]$ to $S[j]$
X	A data window of size m, $\{x_1, \ldots, x_m\}$
$X[i]$	The i-th data point in X
$X[i][j]$	The value of the j-th dimension of the i-th data point in X

3.1. Data Preparation

In our study, the raw dataset S consisted of 2590 days Bitcoin data (from 29 November 2011 to 31 December 2018). Each data point, p, is an 18-dimensional real-valued vector where each dimension stores a daily value of one of the Bitcoin blockchain features discussed in Section 2. To predict Bitcoin prices, sequences of previous data were exploited. When the size of each sequence was m, a total of $2590 - m + 1$ sequence data were used to train and test various deep learning-based prediction models, that is, from $S[1 : m]$ to $S[2590 - m + 1 : 2590]$, as shown in Figure 4. The sequence size was experimentally determined as explained in Section 4, and the first 80% of the sequence data were used as the training data and the rest as the test data. Given a sequence $S[i : i + m - 1]$, for a regression problem, we predicted the Bitcoin price of the $(i + m)$-th day, and for a classification problem, we predicted whether the price would go up or down in the $(i + m)$-th day with respect to the $(i + m - 1)$-th day. In other words, by analyzing previous Bitcoin prices including today's price, we predict tomorrow's price for the regression case and predict if tomorrow's price will go up or down with respect to today's price for the classification case, as it is what most investors or traders would be interested in.

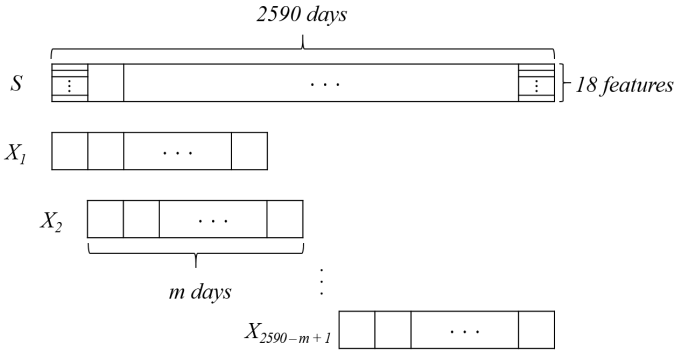


Figure 4. Data preprocessing. A total of $2590 - m + 1$ sequence data are generated from the whole dataset of 2590 days if m consecutive days are analyzed to predict the next Bitcoin price.

Each sequence X was normalized in the following way. For all i and j,

$$Y[i][j] = \frac{X[i][j]}{X[0][j]} - 1$$

Then, the normalized sequences Ys were used instead of the original sequences Xs to build and test the prediction models. This normalization method is more effective in capturing local trends than the usual minmax normalization method. Assuming that max_j and min_j are, respectively, the maximum

and the minimum value of the j-th dimension of the raw data points, the "minmax normalization" method translates each data point as follows,

$$S'[i][j] = \frac{S[i][j]-min_j}{max_j-min_j}$$

That is, every value is normalized into a value between 0 and 1. As the highest price is 7054.33 times higher than the lowest price in our dataset, if the usual minmax normalization method is used, the Bitcoin prices hardly change in most sequence data, and thus the prediction accuracy is degraded. Experimental results also confirmed that the first value-based normalization was more effective than the minmax normalization.

3.2. Prediction Models

3.2.1. Deep Neural Networks

The first prediction model was based on a deep neural network (DNN). A DNN usually consists of an input layer, several hidden layers, and an output layer as shown in Figure 5.

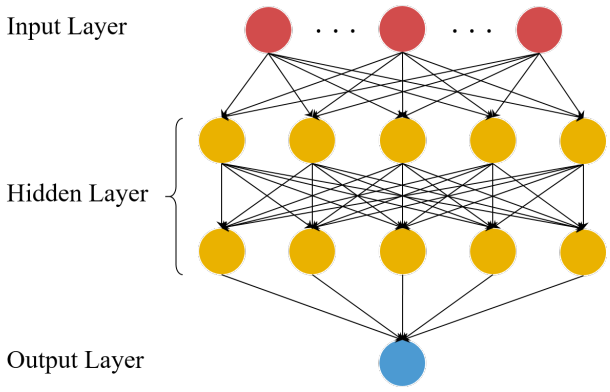


Figure 5. An example of fully-connected deep neural network (DNN) model.

As we use 18 features to predict the Bitcoin price, the input layer consists of 18 nodes, i.e., each input node corresponds to each feature. Moreover, if we analyze m days data at once, then each input node takes a column vector of size m as input. In addition, in our DNN prediction model, every node in one layer is connected to every node in the next layer, i.e., fully-connected. That is, each node in the hidden layer takes input vectors $x_1, \ldots, x_n$ from the previous layer, where n is the number of nodes in the previous layer and the size of each vector is m. Its output h is then defined as

$$h = \sigma_h(w_1 \odot x_1 + \cdots + w_n \odot x_n + b)$$

where the weight vectors $w_1, \ldots, w_n$ and the bias b are parameters to be learned using training data. In addition, $\odot$ is an element-wise multiplication (Hadamard product) and σ_h is a nonlinear function, such as the logistic sigmoid, defined by the activation function

$$\sigma(x) = \frac{1}{1+\exp(-x)}$$

and the rectified linear unit (ReLU) [28], defined as $g(x) = \max(0, x)$. σ_h is applied element-wise, and thus the size of h is also m.

In the output layer, the set of input vectors are flattened into a single vector and then translated into a single value y using a dot product between the flattened vector and a weight vector. For a regression problem, the final value y is used as the predicted Bitcoin price. For a classification problem,

y is further translated into a value between 0 and 1 using a sigmoid function and then rounded off. In particular, 1 means the rise of the price and 0 means the fall or no change.

If we represent the whole DNN model as a function f, the parameters of the model are optimized using the back-propagation algorithm [12] and training data so that the following mean squared error can be minimized:

$$\frac{1}{N}\sum_{i=1}^{N}(f(X_i) - y_i^{true})^2$$

where $f(X_i)$ is a predicted value based on an input sequence X_i and y_i^{true} is the true Bitcoin price at the day after the day corresponding to $X_i[m]$ for regression. For classification, y_i^{true} is 1 if the true price is higher than the price at $X_i[m]$ and otherwise 0.

3.2.2. Recurrent Neural Networks and Long Short-Term Memory

For time series data, it is well known that a recurrent neural network (RNN) [12] model and a long short-term memory (LSTM) [17] model are effective. Although a DNN model can be used to predict time series data, it does not exploit the temporal relationship between the data points in the input sequence. To exploit the temporal relationship, both RNN and LSTM models use an internal state (memory). Figure 6 shows example RNN and LSTM models for the Bitcoin price prediction.

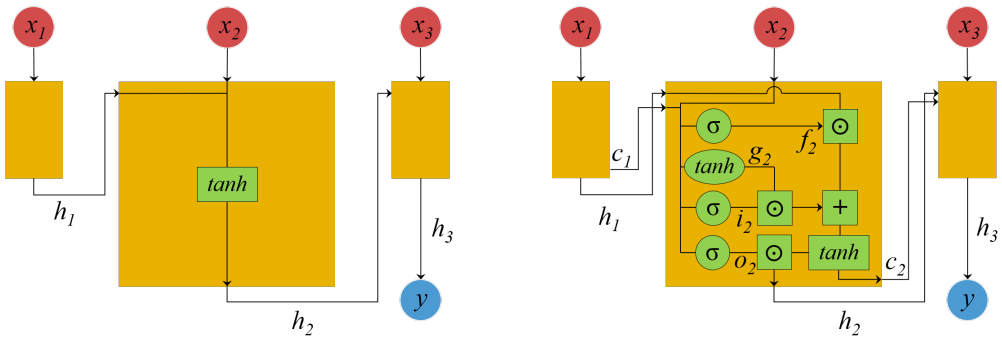


Figure 6. Recurrent neural network (RNN) model (**left**) and long short-term memory (LSTM) model (**right**).

To illustrate, suppose that we use three days data to predict the price. Then, the input layer of an RNN model consists of three nodes each of which takes the whole data of a single day, i.e., a vector of 18 features. Given x_t as an input, each hidden state h_t is then computed as follows:

$$h_t = tanh(W_h x_t + U_h h_{t-1} + b_h)$$

where W_h, U_h, and b_h are parameters to be learned and $tanh$ is a hyperbolic tangent function. The main difference from DNNs is that RNNs use not only the input x_t, but also the previous hidden state h_{t-1}, to compute the current hidden state h_t, thus exploiting the temporal dependency between the input sequence data. Assuming that we use three days sequence data, the final output y for regression is computed as follows,

$$y = w_y h_3 + b_y$$

where w_y and b_y are also parameters to be learned. One drawback of RNNs is that when the length of the input sequence is large, the long-term information is rarely considered due to the so-called vanishing gradient problem.

LSTM avoids the vanishing gradient problem mainly by using several gate structures, namely input, output, and forget gates, together with a cell unit (the memory part of an LSTM unit),

and additive connections between cell states. For an LSTM unit at time t, its hidden state h_t, i.e., the output vector of the LSTM unit, is computed as follows,

$$
\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
g_t &= tanh(W_g x_t + U_g h_{t-1} + b_g) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot tanh(c_t)
\end{aligned}
$$

where W, U, and b are parameters to be learned and σ is a sigmoid function. f_t, i_t, and o_t, respectively, correspond to the forget, input, and output gates, and c_t is the cell state vector. Thanks to the cell state, the long-term dependencies between the data points in the input sequence is maintained well and LSTMs can be applied to long sequence data. In the experiment, only LSTM-based prediction models were considered.

3.2.3. Convolutional Neural Networks

A convolutional neural network (CNN) [13,14] has many applications in image analysis and classification problems. Recently, it is shown to be also effective for sequence data analysis [16], and thus we also developed a CNN-based prediction model. Normally, a CNN consists of a series of convolution layers, ReLU layers, pooling layers, and fully-connected layers, where a convolution layer convolves the input with a dot product. In this work, we developed a simple CNN model consisting of a single 2D convolution layer (Conv2D) as shown in Figure 7. (A more sophisticated CNN model is discussed in the next section.) The input is an $m \times 18$ matrix, where m is the number of days to be consulted for the prediction and 18 is the number of the features. A total of 36 2D convolution filters of size 3×18 are used for convolution, where single real values are extracted from consecutive three days data through each filter. That is, 3×18 feature values are translated into a single value and thus each filter produces a real-valued vector of size $m - 2$, which is then applied to an element-wise ReLU activation function. Then, the 36 output vectors produced by the convolution filters are flattened into a single vector of size $(m - 2) \times 36$, which is then translated into a single prediction value through a fully connected layer. The number of filters was determined experimentally. Moreover, unlike other image analysis applications, simply adding more convolution layers together with pooling layers did not improve the performance of CNN models for Bitcoin price prediction, and therefore we present only a simple CNN model.

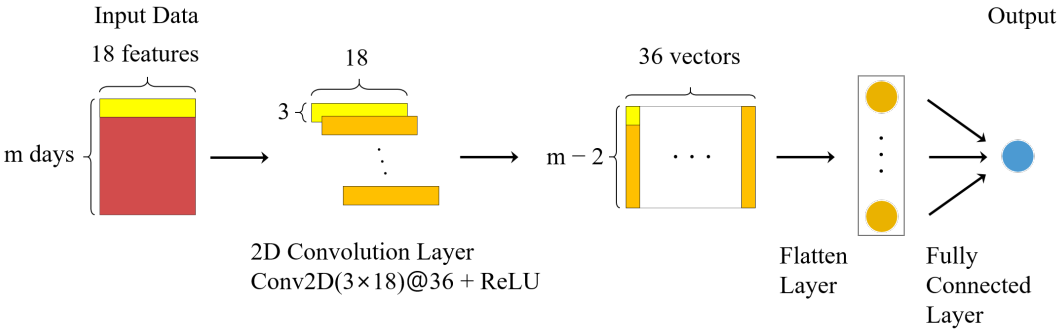
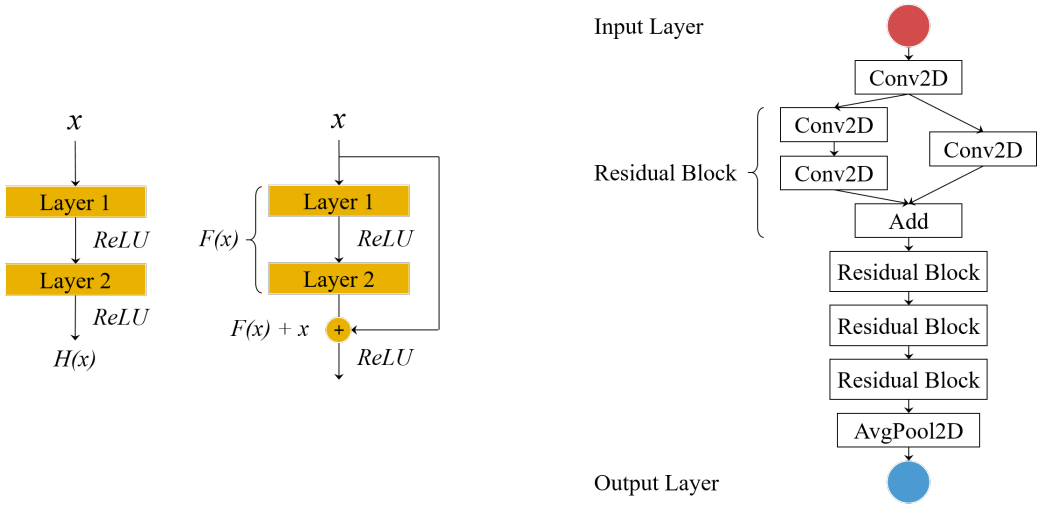


Figure 7. Our CNN model. It consists of a single 2D convolution layer where 36 filters of size 3×18 are used for convolution. An $m \times 18$ input matrix is translated into an $(m - 2) \times 36$ matrix by the Conv2D layer.

3.2.4. Deep Residual Networks

Although stacking many layers allows a CNN model to encode a very complex function from the input to the output, it is difficult to train such a model due to the vanishing gradient problem as in RNNs. A deep residual network (ResNet) [15] resolves this problem by introducing a so-called "identity shortcut connection" that skips one or more layers. Figure 8a shows stacked nonlinear layers without/with a shortcut connection. Assuming that $H(x)$ is the desired underlying mapping, residual learning lets the stacked nonlinear layers fit a residual mapping of $H(x) - x$. The original mapping is then recast to $F(x) + x$. In the work by the authors of [15], it is shown that the residual mapping is easier to optimize than the original. By simply adding identity shortcut connections, deep convolutional networks can easily be optimized and produce a good result. Figure 8b shows our ResNet model used in the experiment. It consists of an input layer, one 2D convolution layer, four residual blocks, one average pooling layer, and an output layer. A residual block implements a shortcut connection between two convolution layers. For ResNet, we use convolution layers different from the one used for CNN. More precisely, the first Conv2D for input data uses 36 convolution filters of size 4×4. Then, each Conv2D in the four residual blocks uses filters of size 2×2. The number of filters used in each Conv2D of the first, second, third, and last residual blocks is 36, 72, 144, and 288, respectively. Although it is not shown in the figure, we apply batch normalization [29] and the ReLU activation function after each Conv2D in each residual block. After applying 2D average pooling of size 3×3, we flatten the resultant 288 vectors of size 1×1 into a one-dimensional vector. It is then translated into a single prediction value through a fully connected layer as in the CNN model.



(**a**) Plain learning (left) and residual learning (right) (**b**) Our ResNet model

Figure 8. Our ResNet model. It uses four residual blocks which implement shortcut connections between two convolution layers.

3.2.5. Combinations of CNNs and RNNs

As CNNs are effective for structural analysis, whereas RNNs are effective for temporal analysis, we developed a prediction model called CRNN by combining CNN and LSTM models so as to integrate their strengths [18]. Figure 9 shows the structure of our CRNN model. The same input data are fed into both the CNN and LSTM models discussed in the previous sections. In particular, for the CNN part, after applying 2D convolution, we also apply batch normalization and flattening. As for the LSTM part, we additionally apply dropout [30]. The results of CNN and LSTM are then combined into a single vector, which is then translated into a single prediction value through a fully connected layer. As a combining operator, we tested element-wise addition, subtraction, multiplication, average, maximum and minimum operators, and a concatenation operator. In the experiment, the subtraction

operator was used for regression problems while the concatenation operator was used for classification problems because they showed better performance.

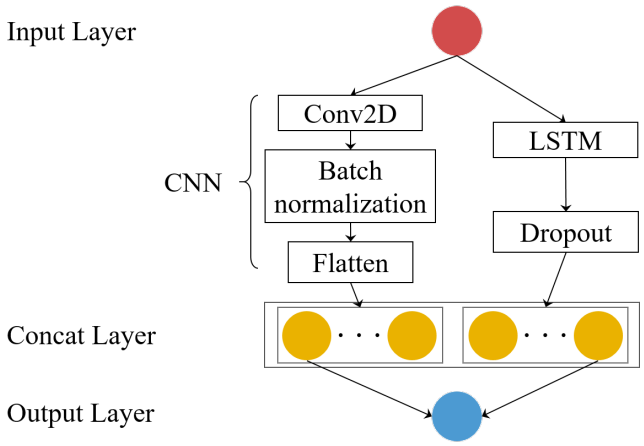


Figure 9. Our CRNN model (a combination of CNNs and RNNs) that combines the results of CNN and LSTM models using a concatenation operator. The Conv2D block represents the 2D convolution layer in Figure 7 and the LSTM block represents the LSTM model discussed in Secion 3.2.2.

3.2.6. Ensemble Models

The last model we considered is a stacking ensemble model [31]. The main idea behind ensemble learning is that by combining multiple weak models, one may produce a strong model with better predictive performance. Our stacking model consists of two levels as shown in Figure 10. The first level consists of three base learners, namely, DNN, LSTM, and CNN, which are already discussed in the previous sections. The second level consists of a single DNN model as a meta learner, whose main purpose is to find an optimal combination of the base learners. To train the stacking model, we divided the training data into two disjoint sets. The first dataset was then used to train the base learners. Next, each base learner was tested on the second dataset to make predictions. Lastly, the meta learner was trained to produce final outputs based on the predicted values of the base learners as inputs.

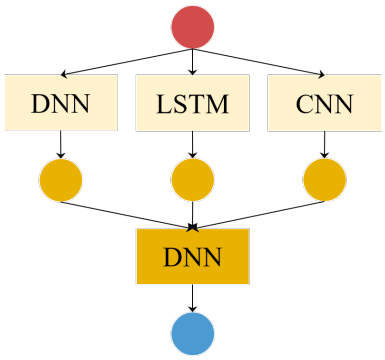


Figure 10. Stacking ensemble model: DNN, LSTM, and CNN are used as base learners at the first level and another DNN as a meta learner at the second level.

4. Experimental Results

This section experimentally evaluates the performance of the DNN, LSTM, CNN, ResNet, CRNN, and Ensemble models. As a baseline method, we also compared support vector machine (SVM) prediction models [23]. We used the SVR class for regression and the SVC class for classification, which are provided in the Python Scikit-learn machine learning library (https://scikit-learn.org/stable/). Three kernel options, namely, "linear", "poly (polynomial)", and "rbf (radial basis function)", were tested. The performance of SVMs with the polynomial kernel was not so good while the

performance with the linear kernel and the RBF kernel was comparable. Therefore, in this section, we report only the performance of SVMs with the RBF kernel. In addition, we also implemented linear and logistic regression models and gated recurrent unit (GRU) models. GRU is a simpler variation of LSTM that uses fewer parameters but exhibits better performance than LSTM on certain small datasets [32]. In the experiments, however, the performance of linear and logistic regression models was similar to or worse than that of SVM models, whereas the performance of GRU models was comparable to that of LSTM models. Therefore, we omitted the experimental results of prediction models based on linear and logistic regression and GRUs.

Every model was trained using the training data, and their performance was tested using the test data. In particular, we compared their prediction performance with respect to the mean absolute percentage error (MAPE) for regression analysis, which is defined by

$$\frac{100\%}{N}\sum_{i=1}^{N}\left|\frac{y_i^{true}-y_i^{pred}}{y_i^{true}}\right|$$

where N is the total number of predictions, and y_i^{true} and y_i^{pred} are the actual value and the predicted value, respectively. For classification analysis, we compared the accuracy of the proposed models, which is defined by the number of correct predictions divided by the total number of predictions. The seven prediction models were compared under various conditions as follows.

- The first set of experiments varied the size, m, of the input sequence. We used $m = 5, 10, 20, 50$, and 100. For other experiments below, m was set to 20 for regression problems and 50 for classification problems.
- In the second set of experiments, for six blockchain features such as difficulty, est-trans-vol-usd, hash-rate, my-wallets, trade-vol, and trans-fees-usd, where the difference between the minimum and the maximum values is very large, we compared the performance of using their log values with the performance of using their plain values. For other experiments, the default experimental setting was to use their log values.
- Different data split methods were also compared. The default split method was to use the first 80% of the entire data as training data and the rest as test data (i.e., sequential partitioning).
- In addition, different normalization methods were compared. The default normalization method was the first value-based normalization discussed in Section 3.1.
- Finally, we compared the profitability of the proposed models using a simple trading strategy: buy Bitcoin with all funds or hold it if the model predicts a price rise or otherwise sell all Bitcoin or wait.

All experiments were conducted on a workstation equipped with an Intel® Core™ i7-6700 3.4 GHz CPU, 8 GB of main memory, and an NVIDIA GeForce GTX 1080 GPU with 8 GB memory. The host operating system was the Windows 10 and all prediction models were implemented using Python 3 and the Keras 2.1.6 deep learning library (https://keras.io). For every deep learning-based prediction model, the number of epochs was 200 and the batch size was set to 64. All measurements were averaged over 10 sample runs.

4.1. Effect of the Sequence Size

4.1.1. Effect of the Sequence Size on Regression

We experimentally determined the input sequence size, m, i.e., the number of previous data consulted to predict the Bitcoin price. Table 4 shows the effects of m on the regression performance of each prediction model with respect to the MAPE. In the table, the underlined numbers denote the best value for the given window size and the boldface numbers denote the best value for the given prediction model. All prediction models exhibited the best performance when $m = 5$ (the lower the MAPE, the better the performance). Indeed, for every model, the MAPE increases with m. The reason

is that when the input sequence size is small, most prediction models were trained to predict a price that is very close to the previous day's price, thus minimizing the MAPE. Interestingly, in terms of the MAPE, a simple shift model is the best, which simply uses the previous day's price for the prediction. In the entire data, the rate of price change of more than 5% is 17.72%, and thus it is indeed an effective policy to predict a price closed to the previous price for regression problems. However, this is not the case for classification problems, as explained in Section 4.1.2.

In this experiment, when $m = 5$, DNN showed the best performance, whereas when $m = 10, 20$, LSTM was the best. Meanwhile, when $m = 50, 100$, SVM showed the best performance. Moreover, SVM was the least sensitive to the sequence size. When $m = 5, 10, 20$, the performance of all prediction models was comparable to each other. However, when m was large, CNN did not seem to work very well, and consequently the performance of ResNet, CRNN, and Ensemble decreased greatly. For the rest of experiments, m was set to 20 for regression problems unless otherwise specified. Although all prediction models performed the best when $m = 5$, we chose to use $m = 20$, because this setting clearly contrasted the performance of the proposed models and all models were relatively well-trained under this setting. Another reason not to use $m = 5$ is that when $m = 5$, most prediction models behaved like a shift model and thus their distinct characteristics seemed to disappear.

Table 4. Effect of the sequence size m on regression (MAPE, %). The log values of the major features, the sequential partitioning, and the first value-based normalization were used.

Size (m)	DNN	LSTM	CNN	ResNet	CRNN	Ensemble	SVM
5	**3.61**	**3.79**	**4.27**	**4.95**	**4.12**	**4.02**	**4.75**
10	4.00	3.96	4.88	7.12	4.26	4.80	4.88
20	4.81	4.46	7.93	8.96	5.90	6.19	5.19
50	10.88	6.68	20.00	16.91	10.11	11.45	6.34
100	21.44	27.75	115.22	52.10	39.13	48.87	12.77

Figure 11 shows the prediction results of DNN with $m = 5, 10, 20, 50, 100$. Note that when $m = 100$, the predicted prices are totally different from the actual prices. In contrast, when $m = 5, 10, 20$, the predicted prices are quite close to the actual prices. In other words, if the MAPE value is less than 5, then it means that the model produced a good prediction result. Figure 12 shows the prediction results of all prediction models with $m = 20$. Although there were some deviations from the actual prices (in particular, when the Bitcoin price was the highest), all prediction models quite successfully predicted the Bitcoin prices.

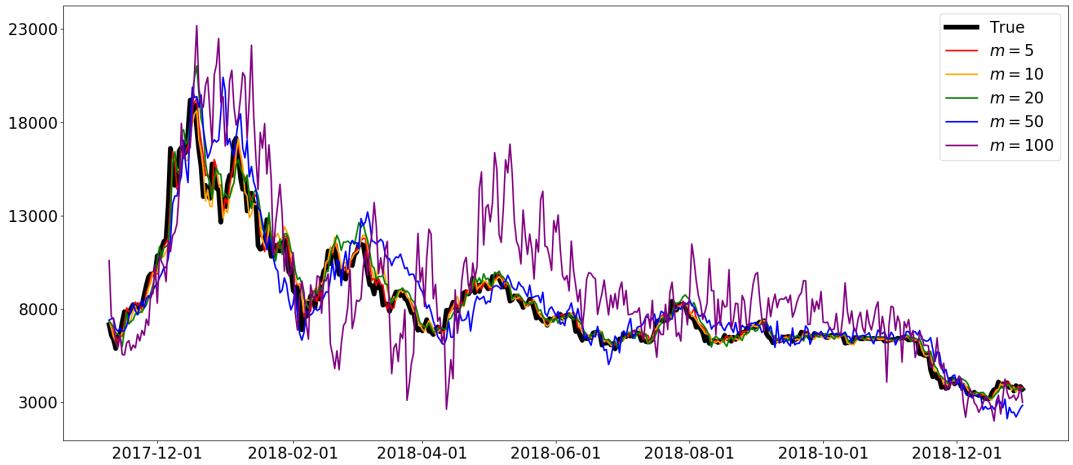


Figure 11. Prediction results of the DNN model with $m = 5, 10, 20, 50, 100$. The log values of the major features, the sequential partitioning, and the first value-based normalization were used.

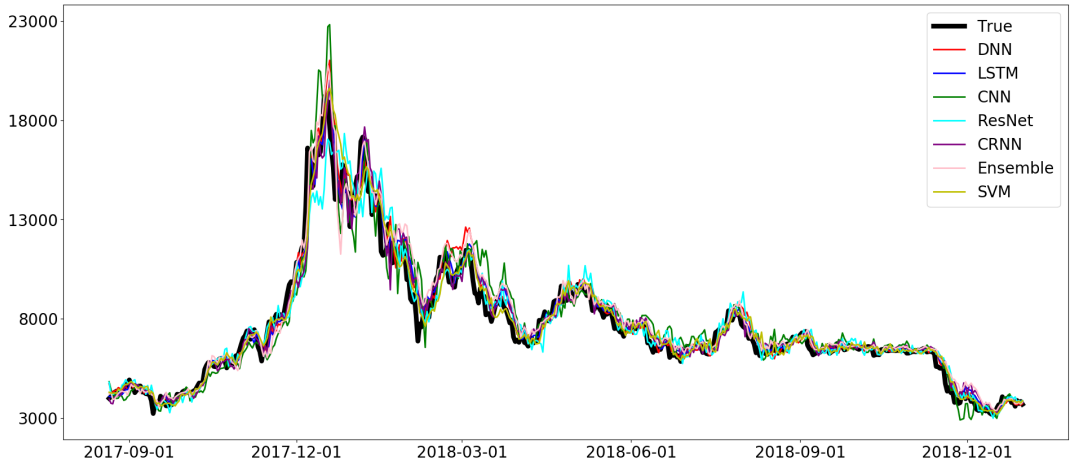


Figure 12. Prediction results of the proposed models with $m = 20$. The log values of the major features, the sequential partitioning, and the first value-based normalization were used.

Figure 13 shows the part of the results in Figures 11 and 12 from 1 December 2017 to 1 April 2018, that is, a period of high price fluctuation. The left figure shows the prediction results of DNN with $m = 5, 20$. The prediction graph with $m = 5$ is similar to the actual price graph shifted by 5 days forward, with slight deviations. In contrast, DNN with $m = 20$ often overestimates or underestimates the Bitcoin price. The right figure compares the prediction results of DNN and LSTM when $m = 20$. Although the difference between the MAPE values of the two models is small, i.e., 0.35, the prediction results of LSTM are mostly much closer to the actual prices.

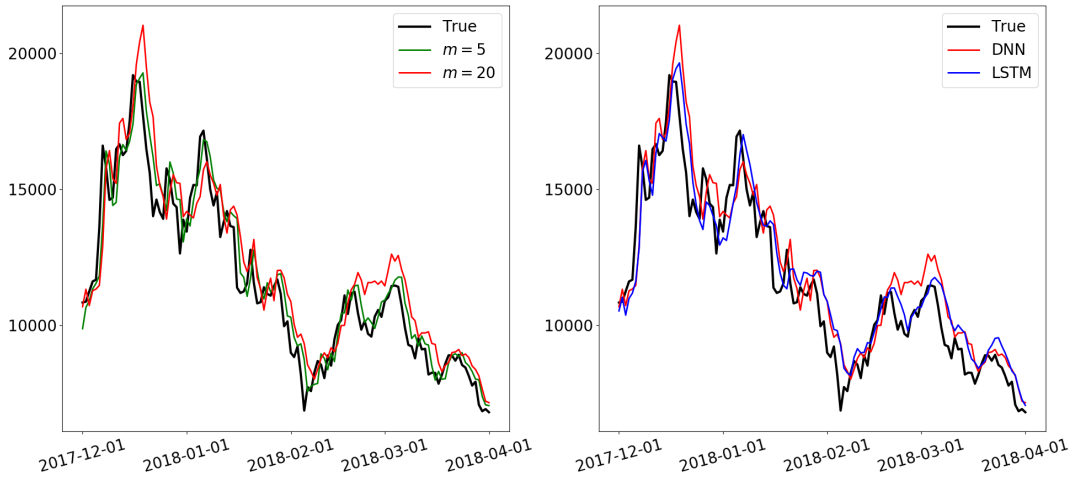


Figure 13. Prediction results of DNN with $m = 5, 20$ (**left**) and prediction results of DNN and LSTM with $m = 20$ (**right**) from 1 December 2017 to 1 April 2018, excerpted from Figures 11 and 12.

4.1.2. Effect of the Sequence Size on Classification

Table 5 shows the effect of the input sequence size m on the classification performance of each prediction model with respect to accuracy. For classification problems, we also tested two baseline prediction methods, namely, Base and Random. The Base model always predicts the rise of the price, and thus its accuracy is the price rise rate in the whole test data. As the total number of sequence data slightly varies depending on the size of the sequence, the accuracy of Base also slightly varies. The Random model randomly chooses between the rise and the fall of the price. In clear contrast to the regression result given in Table 4, most deep learning-based prediction models exhibited poor performance when $m = 5, 10, 20$; in these settings, Base or Random rather showed a better result. Note that for regression, the smaller the sequence size, the better the performance of the deep learning models. In contrast, for classification, it seems that more data are needed for better performance.

This is partly because when m is small, every prediction model tends to behave like a shift model, but it means that when the price falls after rises (or vice versa), the model tends to make a wrong prediction. Another difference from the regression results was that the performance of LSTM was not particularly good and mostly similar to the performance of other models. When $m = 50$, where most prediction models were relatively well trained, DNN, CNN, CRNN, and Ensemble outperformed LSTM. Meanwhile, in contrast to the regression results, SVM was not trained very well for all cases and behaved like Base. As all deep learning models except ResNet exhibited the best performance when $m = 50$, for the rest of experiments, m was set to 50 for classification problems.

Table 5. Effect of the sequence size m on classification (accuracy). The log values of the major features, the sequential partitioning, and the first value-based normalization were used.

Size (m)	DNN	LSTM	CNN	ResNet	CRNN	Ensemble	SVM	Base	Random
5	49.16%	50.43%	50.14%	**50.88%**	50.02%	49.14%	**50.88%**	**50.88%**	**51.38%**
10	48.19%	50.22%	48.74%	50.51%	48.37%	49.43%	50.79%	50.79%	50.62%
20	50.64%	49.70%	50.78%	49.84%	48.86%	51.02%	50.80%	50.80%	49.68%
50	**53.06%**	**50.94%**	**52.48%**	49.83%	**51.52%**	**52.02%**	50.85%	50.85%	50.98%
100	48.16%	48.89%	48.49%	50.00%	48.59%	49.40%	50.00%	50.00%	48.61%

Table 6 compares the proposed prediction models using various measures such as precision, recall, specificity, and F1 score when $m = 50$. Specifically, the precision is the percentage of the correctly predicted price ups out of the total price ups predicted by the model. The recall is the percentage of the correctly predicted price ups out of the total actual price ups, and thus the recall of the Base model is 100%. The specificity is the percentage of the correctly predicted price downs out of the total actual price downs, and therefore the specificity of Base is 0%. Finally, the F1 score is the harmonic mean of the precision and the recall. In terms of the accuracy, precision, and specificity, DNN showed the best result, whereas in terms of the recall and the F1 score, DNN and Ensemble showed a relatively good result. Overall, although DNN was slightly better than other models, the performance of all proposed models except for ResNet was comparable to each other and there was no clear winner. One reason for the relatively poor performance of ResNet is the lack of data for training. As ResNet uses many layers, it requires a huge amount of data, but the amount of data available on the day scale is insufficient for training ResNet.

Table 6. Model evaluation under various measures when $m = 50$. The log values of the major features, the sequential partitioning, and the first value-based normalization were used.

Measure	DNN	LSTM	CNN	ResNet	CRNN	Ensemble	SVM	Base	Random
Accuracy	53.06%	50.94%	52.48%	49.83%	51.52%	52.02%	50.85%	50.85%	50.98%
Precision	52.90%	51.40%	52.70%	20.40%	52.00%	51.89%	51.00%	51.00%	51.80%
Recall	69.70%	66.70%	66.80%	40.00%	63.50%	75.22%	100.00%	100.00%	51.20%
Specificity	53.70%	50.00%	52.30%	29.40%	50.80%	52.33%	0.00%	0.00%	50.20%
F1 score	60.00%	57.90%	58.50%	26.80%	56.90%	61.33%	67.00%	67.00%	51.50%

4.2. Effect of Using Log Values

This section presents the effect of using the log values of the six blockchain features: difficulty, est-trans-vol-usd, hash-rate, my-wallets, trade-vol, and trans-fees-usd. Tables 7 and 8 show the results on regression and classification, respectively. In both cases, for all prediction models except for ResNet and CRNN, using the log values was better than using the plain values. In particular, in Table 8, when the plain values were used, most prediction models were not trained well, and thus exhibited poor performance. As the difference between the maximum and the minimum plain values for the aforementioned six features is too high, e.g., up to eight million times, it seems that the prediction models were trained to be more sensitive to the changes of the six features than the rest 12 features.

By using their log values, the prediction models were trained to consider other features as important factors as well, thus improving the prediction accuracy.

Table 7. Effect of using log values on regression (MAPE, %). The sequence size $m = 20$. The sequential partitioning and the first value-based normalization were used.

	DNN	LSTM	CNN	ResNet	CRNN	Ensemble	SVM
Log values	**4.81**	**4.46**	**7.93**	8.96	**5.90**	**6.19**	**5.19**
Plain values	5.48	5.21	10.13	**8.94**	7.50	6.40	7.02

Table 8. Effect of using log values on classification (accuracy). The sequence size is $m = 50$. The sequential partitioning and the first value-based normalization were used.

	DNN	LSTM	CNN	ResNet	CRNN	Ensemble	SVM	Base	Random
Log values	**53.06%**	**50.94%**	**52.48%**	49.83%	51.52%	**52.02%**	50.85%	50.85%	**50.98%**
Plain values	50.85%	50.53%	49.83%	**50.81%**	**52.22%**	49.74%	50.85%	50.85%	49.76%

4.3. Effect of Data Split Methods

This section presents the effect of data split methods on the performance of the proposed models. We compared three methods: sequential and random partitioning, and 5-fold cross-validation (CV). The usual sequential partitioning is the default split method used in our experiments, and it uses the first 80% data as training data and the rest 20% data as test data. In this case, the training data contain only the information up to 31 July 2017, and thus the price surge and collapse and abrupt changes from November 2017 to February 2018 are not reflected in the prediction models. In contrast, the random partitioning randomly chooses the training data from the entire sequence data, and thus the recent large fluctuations are also partly reflected in the prediction models. Lastly, 5-fold CV divides the whole dataset into five equal-sized subsets. Then, for each prediction method, five models are trained and tested using five distinct pairs of training and test data, where the former consists of four subsets and the latter the remaining subset. The prediction performance is then averaged. By using 5-fold CV, we may avoid overfitting and sampling bias.

Tables 9 shows the regression performance, where all the proposed models performed better when using the random partitioning. Note that LSTM showed the best result for all cases. The results of 5-fold CV are similar to those of using the sequential partitioning. In contrast to the regression result, the random partitioning is not necessarily a better choice for classification as shown in Table 10. For classification, what matters is having a sufficient amount of patterns of price ups and downs in the training data. As both sequential and random partitioning generated a similar amount of ups and downs patterns in the training data, the classification performance under the two methods was comparable. Meanwhile, in the results of 5-fold CV, Base and SVM significantly outperformed the other deep learning models except for DNN. The reason is that the percentage of price rises during the period from 20 April 2016 to 31 July 2017 (the fourth subset of the five CV subsets) is 60.47%, which was the prediction accuracy of Base and SVM, whereas most deep learning models achieved a much lower accuracy. If we exclude that period, the prediction accuracy of Base and SVM becomes 52.39% and 52.08%, respectively, which is lower than the prediction accuracy 53.57% of DNN under the same condition.

Table 9. Effect of data split methods on regression (MAPE, %). The sequence size $m = 20$. The log values of the major features and the first value-based normalization were used.

	DNN	LSTM	CNN	ResNet	CRNN	Ensemble	SVM
sequential	4.81	4.46	7.93	8.96	5.90	6.19	5.19
random	**3.65**	**3.52**	**4.93**	**5.26**	**5.20**	**4.16**	**4.23**
5-fold CV	4.83	4.09	6.80	9.75	7.12	5.18	5.04

Table 10. Effect of data split methods on classification (accuracy). The sequence size $m = 50$. The log values of the major features and the first value-based normalization were used.

	DNN	LSTM	CNN	ResNet	CRNN	Ensemble	SVM	Base	Random
sequential	53.06%	50.94%	**52.48%**	49.83%	51.52%	52.02%	50.85%	50.85%	**50.98%**
random	52.46%	**52.24%**	52.17%	49.76%	**52.40%**	**52.09%**	50.97%	51.18%	50.63%
5-fold CV	**53.60%**	51.82%	51.07%	**51.92%**	51.06%	51.14%	**53.76%**	**54.00%**	50.68%

4.4. Effect of Normalization Methods

This section presents the effect of normalization methods on the performance of the proposed models. We compared two normalization methods using the sequential partitioning: the first value-based normalization and the usual minmax normalization discussed in Section 3.1. Tables 11 and 12 show the results on regression and classification, respectively. In Table 11, for all prediction models, using the first value-based normalization was better than using the minmax normalization. By using the first value-based normalization, local fluctuations were better reflected in the prediction models, and thus the prediction accuracy was improved. In contrast, when using the minmax normalization, the Bitcoin price fluctuations in the training data (until 31 July 2017) were relatively small, and thus the prediction accuracy was not good for the test data containing large price fluctuations. However, for classification, it was not always better to use the first value-based normalization as shown in Table 12. The reason is along the same lines as Table 10. Figure 14 shows the prediction results of DNN with two normalization methods. Although the predicted prices under the first value-based normalization were very close to the actual prices, the predicted prices under the minmax normalization were quite deviated from the actual prices.

Table 11. Effect of normalization methods on regression (MAPE, %). The sequence size $m = 20$. The log values of the major features and the sequential partitioning were used.

	DNN	LSTM	CNN	ResNet	CRNN	Ensemble	SVM
first value	**4.81**	**4.46**	**7.93**	**8.96**	**5.90**	**6.19**	**5.19**
minmax	14.23	14.18	157.00	92.68	35.73	22.34	34.44

Table 12. Effect of normalization methods on classification (accuracy). The sequence size $m = 20$. The log values of the major features and the sequential partitioning were used.

	DNN	LSTM	CNN	ResNet	CRNN	Ensemble	SVM	Base	Random
first value	**53.06%**	50.94%	**52.48%**	49.83%	**51.52%**	**52.02%**	50.85%	50.85%	**50.98%**
minmax	51.71%	**51.28%**	49.83%	**50.85%**	49.36%	49.44%	50.85%	50.85%	49.82%

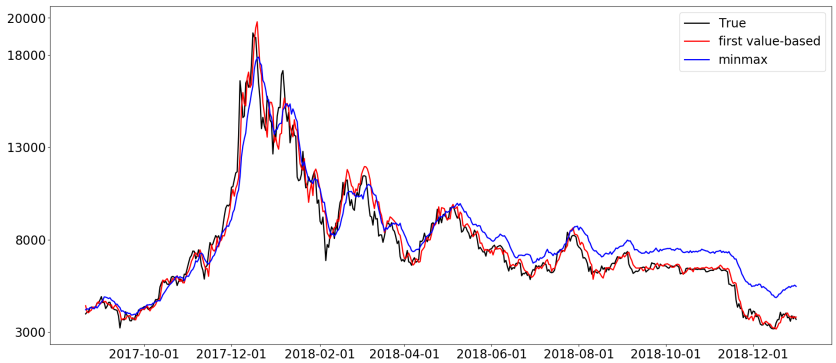


Figure 14. Prediction results of the DNN model with different normalization methods. The sequence size $m = 20$. The log values of the major features and the sequential partitioning were used.

4.5. Profitability of the Proposed Models

This section presents the results of a profitability analysis using a simple trading strategy. More specifically, for each regression model, we bought Bitcoin with all funds if the predicted price was higher than the current price and sold all Bitcoin if the predicted price was lower than the current price. Similarly, for each classification model, we bought Bitcoin if the prediction model predicted a price rise and sold it otherwise. Table 13 shows the investment results during the period from 20 September 2017 to 31 December 2018, with the initial budget of 10,000 USD. For simplicity, we did not consider trading fees. In the beginning, for classification models, the previous 50-days information from 1 August 2017 to 19 September 2017, which was not used during the model training, was used to predict the price on 20 September 2017. As for regression models, the previous 20-days information from 31 August 2017 to 19 September 2017 was used to predict the price on 20 September 2017. The Bitcoin price was 3874.46 USD on 20 September 2017 and 3693.30 USD on 31 December 2018. Interestingly, every deep learning-based classification model made a little profit, whereas every regression model made a negative return. Moreover, all deep learning-based regression models even underperformed Base, which corresponds to a simple buy-and-hold strategy, and Random, which corresponds to a strategy that randomly buys and sells. Meanwhile, as in the previous results, DNN was the best for classification-based trading, whereas LSTM was the best for regression-based trading among deep learning methods. In addition, although SVM behaved like Base for classification-based trading, for regression-based trading, it outperformed every deep learning models. Overall, classification models were more effective than regression models for algorithmic trading, but it is still premature to apply such models in practice.

Table 13. Results of a profitability analysis. $m = 20$ was used for regression models and $m = 50$ for classification models. The log values of the major features, the sequential partitioning, and the first value-based normalization were used. For every model, the initial budget was 10,000 USD and the final value of investment is shown in the table.

	DNN	**LSTM**	**CNN**	**ResNet**	**CRNN**	**Ensemble**	**SVM**	**Base**	**Random**
regression	6755.55	8806.72	6616.87	7608.35	8102.71	5772.99	**9842.95**	–	–
classification	**10877.07**	**10359.42**	**10422.19**	**10619.98**	**10315.18**	**10432.44**	9532.43	**9532.43**	**9918.70**

5. Discussion and Conclusions

In this study, we have developed and compared various deep learning-based Bitcoin price prediction models using Bitcoin blockchain information. More specifically, we tested the state-of-the-art deep learning models such as deep neural networks (DNN), long short-term memory (LSTM) models, convolutional neural networks (CNN), deep residual networks (ResNet), and their combinations. We addressed both regression and classification problems, where the former predicts the future Bitcoin price, and the latter predicts whether or not the future price will go up or down. For regression problems, LSTM slightly outperformed the other models, whereas for classification problems, DNN slightly outperformed the other models unlike the previous literature on Bitcoin price prediction. Although CNN and ResNet are known to be very effective in many applications, including sequence data analysis, their performance was not particularly good for Bitcoin price prediction. Overall, there was no clear winner and the performance of all deep learning models studied in this work was comparable to each other. In addition, although deep learning models seem to predict the Bitcoin price very well in terms of the regression analysis, it is still premature to solely use such models for algorithmic Bitcoin trading.

There are several research directions for future work. First, it would be interesting to consider other recent deep learning models, such as transformer networks [33] and few-shot/one-shot learning [34], to build prediction models. Moreover, to improve the accuracy of the prediction models, it would also be interesting to consider other factors such as stock market indices and major currency exchange rates as in the works by the authors of [7,10]. We may also consider price fluctuations of major Altcoins such

as Ethereum, Ripple, Litecoin, Bitcoin Cash, or EOS as in the work by the authors of [9]. Lastly, news and social media sentiment analysis could also be incorporated to improve the prediction accuracy as in the works by the authors of [24,25].

Author Contributions: S.J. developed and implemented the algorithms and conducted the experiments. S.J. and H.I wrote the initial draft. J.K. and H.I. verified the results, supervised this research, and revised the paper.

Funding: This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2019R1F1A1063272) and by a 2015 Research Grant from Kangwon National University.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. Technical Report. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 12 July 2019).
2. Cootner, P.H. *The Random Character of Stock Market Prices*; MIT Press: Cambridge, MA, USA, 1964.
3. Alessandretti, L.; ElBahrawy, A.; Aiello, L.M.; Baronchelli, A. Anticipating Cryptocurrency Prices Using Machine Learning. *Complexity* **2018**, *2018*, 8983590:1–8983590:16. [CrossRef]
4. Corbet, S.; Lucey, B.; Urquhart, A.; Yarovaya, L. Cryptocurrencies as a financial asset: A systematic analysis. *Int. Rev. Financ. Anal.* **2019**, *62*, 182–199. [CrossRef]
5. McNally, S.; Roche, J.; Caton, S. Predicting the Price of Bitcoin Using Machine Learning. In Proceedings of the 2018 26th Euromicro International Confence on Parallel, Distributed and Network-Based Processing (PDP), Cambridge, UK, 21–23 March 2018; pp. 339–343.
6. Saad, M.; Mohaisen, A. Towards characterizing blockchain-based cryptocurrencies for highly-accurate predictions. In Proceedings of the IEEE INFOCOM 2018—IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Honolulu, HI, USA, 16 April 2018; pp. 704–709.
7. Jang, H.; Lee, J. An Empirical Study on Modeling and Prediction of Bitcoin Prices with Bayesian Neural Networks Based on Blockchain Information. *IEEE Access* **2018**, *6*, 5427–5437. [CrossRef]
8. Nakano, M.; Takahashi, A.; Takahashi, S. Bitcoin technical trading with artificial neural network. *Phys. A Stat. Mech. Appl.* **2018**, *510*, 587–609. [CrossRef]
9. Rebane, J.; Karlsson, I.; Denic, S.; Papapetrou, P. Seq2Seq RNNs and ARIMA models for Cryptocurrency Prediction: A Comparative Study. In Proceedings of the KDD Data Science in Fintech Workshop, London, UK, 20 August 2018.
10. Jang, H.; Lee, J.; Ko, H.; Lee, W. Predicting Bitcoin Prices by Using Rolling Window LSTM model. In Proceedings of the KDD Data Science in Fintech Workshop, London, UK, 20 August 2018.
11. Shintate, T.; Pichl, L. Trend Prediction Classification for High Frequency Bitcoin Time Series with Deep Learning. *J. Risk Financ. Manag.* **2019**, *12*, 17. [CrossRef]
12. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
13. Lecun, Y. *Generalization and Network Design Strategies*; Technical Report CRG-TR-89-4; Department of Computer Science, University of Toronto: Toronto, ON, Canada, 1989.
14. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
15. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 770–778.
16. Dauphin, Y.N.; Fan, A.; Auli, M.; Grangier, D. Language Modeling with Gated Convolutional Networks. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; Volume 70, pp. 933–941.
17. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
18. Wang, J.; Yang, Y.; Mao, J.; Huang, Z.; Huang, C.; Xu, W. CNN-RNN: A Unified Framework for Multi-label Image Classification. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 26 June–1 July 2016; pp. 2285–2294.

19. Box, G.; Jenkins, G. *Time Series Analysis: Forecasting and Control*; Holden-Day: San Francisco, CA, USA, 1976.
20. Ho, T.K. Random decision forests. In Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, Canada, 14–16 August 1995; Volume 1, pp. 278–282.
21. Freund, Y.; Schapire, R.E. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *J. Comput. Syst. Sci.* **1997**, *55*, 119–139. [CrossRef]
22. Bishop, C.M. *Pattern Recognition and Machine Learning*; Springer: Berlin/Heidelberg, Germany, 2006.
23. Cortes, C.; Vapnik, V. Support-vector networks. *Mach. Learn.* **1995**, *20*, 273–297. [CrossRef]
24. Kim, Y.B.; Kim, J.G.; Kim, W.; Im, J.H.; Kim, T.H.; Kang, S.J.; Kim, C.H. Predicting Fluctuations in Cryptocurrency Transactions Based on User Comments and Replies. *PLoS ONE* **2016**, *11*. [CrossRef]
25. Li, T.R.; Chamrajnagar, A.S.; Fong, X.R.; Rizik, N.R.; Fu, F. Sentiment-Based Prediction of Alternative Cryptocurrency Price Fluctuations Using Gradient Boosting Tree Model. *arXiv* **2018**, arXiv:1805.00558.
26. Spearman, C. The Proof and Measurement of Association between Two Things. *Am. J. Psychol.* **1904**, *15*, 72–101. [CrossRef]
27. Pearson, K. Notes on the history of correlation. *Biometrika* **1920**, *13*, 25–45. [CrossRef]
28. Jarrett, K.; Kavukcuoglu, K.; Ranzato, M.; LeCun, Y. What is the best multi-stage architecture for object recognition? In Proceedings of the IEEE 12th International Conference on Computer Vision, Kyoto, Japan, 27 September–4 October 2009; pp. 2146–2153.
29. Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 6–11 July 2015; Volume 37, pp. 448–456.
30. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
31. Wolpert, D.H. Stacked generalization. *Neural Netw.* **1992**, *5*, 241–259. [CrossRef]
32. Cho, K.; van Merrienboer, B.; Gülçehre, Ç.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, Doha, Qata, 25–29 October 2014; pp. 1724–1734.
33. Jaderberg, M.; Simonyan, K.; Zisserman, A.; Kavukcuoglu, K. Spatial Transformer Networks. In Proceedings of the 28th International Conference on Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; Volume 2, pp. 2017–2025.
34. Koch, G.; Zemel, R.; Salakhutdinov, R. Siamese neural networks for one-shot image recognition. In Proceedings of the ICML 2015 Deep Learning Workshop, Lille, France, 10–11 July 2015.