FINANCIAL TIME SERIES PREDICTION USING ADAPTIVE INDICATOR IMAGE CNN-LSTM APPROACH

A PREPRINT

Andrew Christensen
Roy J. Carver Department of Biomedical Engineering
University of Iowa
Iowa City, IA 52240
andrew-christensen@uiowa.edu

Olivia Feldman
Roy J. Carver Department of Biomedical Engineering
University of Iowa
Iowa City, IA 52240
olivia-feldman@uiowa.edu

Jack R. Lynn
Roy J. Carver Department of Biomedical Engineering
University of Iowa
Iowa City, IA 52240
jacklynn@uiowa.edu

Ashwini Shivaprasad
Roy J. Carver Department of Biomedical Engineering
University of Iowa
Iowa City, IA 52240
ashwini-shivaprasad@uiowa.edu

May 16, 2021

ABSTRACT

Stock price data is time invariant, making it an excellent candidate for LSTM (Long Short Term Memory) networks, that store important trends using its memory function. The CNN (Convolutional Neural Networks) are best known for their image recognition capabilities. This property can be used to transform technical indicators, that are used for financial analysis,into images. This paper proposes a CNN-LSTM model that incorporates these two important properties of CNN and LSTM networks to predict stock prices for a given stock. This model will be compared to CNN and LSTM models for prediction accuracy and profit gain. According to the experimental results, although the accuracy of the CNN-LSTM network is the highest compared to the other two models, it performs quite poorly compared to the other two models for maximizing profit.

Keywords Deep Learning Stock-Market Prediction · Convolutional Neural Networks · Time Series to Image Processing Approach · Long Short-Term Memory Recurrent Neural Networks · CNN-LSTM Neural Networks

1 Introduction

Ever since the first exchange opened in the Netherlands in the early 17th century, stock markets have been a strong indicator of an economy's fitness. Therefore, the better traders predict the actions of stock markets, the better they can maximize their returns on investment (ROI).

Two frameworks—fundamental and technical analysis—form the major realms of smart trading. Fundamental analysis centers around the characteristics of an enterprise—size, revenue, expenses, assets, liabilities, etc.—to base financial investments. Technical analysis concerns itself only with patterns of price regardless of individual company character. [1] Two approaches—statistics and machine learning—form the basis of technical analysis. Statistical methods are based on the assumption of linearity and is used for finding explicit trends, like seasonal effects, cycles, and outliers. However, when analyzing financial time series data, statistical analysis fails to capture the complex, noisy, highly volatile, and nonlinear properties of stock prices. [2] To better handle these nonlinearities, machine learning is the natural progression toward more successful stock prediction. [3]

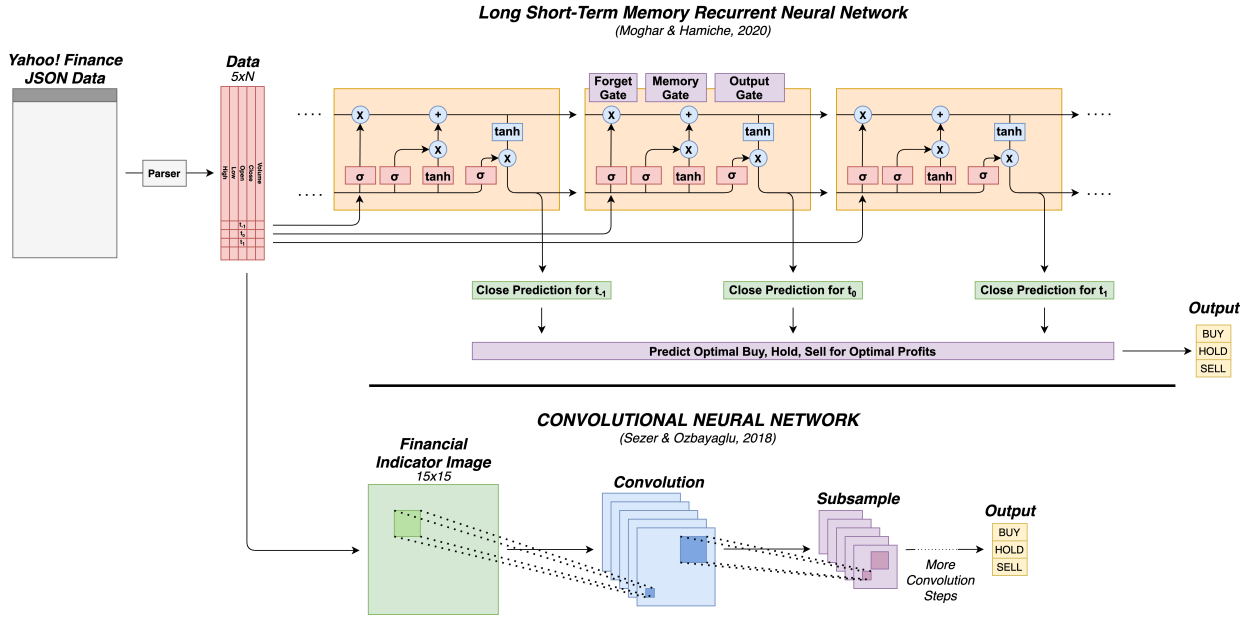

Figure 1: Process flows for CNN and LSTM deep neural networks. The CNN network converts data to technical indicator images and then convolution layers; it outputs buy, hold, or sell indicators. The LSTM network takes OHLCV data and passes them through LSTM layers; it outputs daily open prices, which is then passed through a buy-hold-sell predictor.

1.1 Existing Machine Learning Models Used for Financial Time Series

In 2016, Cavalcante et al. presented a holistic overview of current machine learning techniques used for financial time series, analyzing what trends they followed and what the future of financial time series prediction is. Artificial neural networks (ANN) were by far the most popular, specifically because of how well they handle nonlinearities, discontinuities, and self-adaptation. The majority of ANNs were multi-layer feed forward neural networks that were used to predict daily stock prices. For example, Dhar, Mukherjee, and Ghoshal predicted closing stock indices for the Indian Stock Exchange using a multilayer perceptron. [4] Despite their popularity, ANN's biggest issue is their necessity for high levels of optimization of neural network architecture and data preprocessing. The other most popular framework was support vector machines (SVM) primarily for their implementation of a risk function that minimizes empirical error. By virtue of SVM structure, the separation hyperplane between classifications could be a nonlinear function, again addressing the nonlinearity of stock activity. An example of SVM in practice comes from Chen, in which support vector regression (a subtype of SVM) was used to find the relationship between stock index price and financial indicators. [5] The biggest setback of SVM was that it was only a strong tool for finding general, long-term trends, so its utility for trading is limited. [6]

Despite the successes of traditional machine learning techniques, the future of financial time series prediction is deep learning, with its biggest setback being the lack of available research. [6] In an early attempt, Ding et al. used a convolutional neural network (CNN) and recent news and internet headlines to predict the short- and long-term impacts on stock price. [7] Fischer et al. and Kraus et al. used long short-term memory (LSTM) and deep neural networks to forecast S&P 500 stock trends. [8, 9] Shen et al. implemented a deep learning algorithm with stacked autoencoders and support vector regression to predict Foreign Exchange rate forecasting. [5] Finally, Tino et al. constructed recurrent neural networks (RNN) to predict German DAX and British FTSE. [10]

1.2 Long Short-Term Memory Recurrent Neural Networks

For much of its history, LSTM has been the preferred technique for financial time series primarily because of its ability to predict future potentials while also properly handling exploding/vanishing gradients common in other RNN. [11] The LSTM memory cell consists of three gates: the forget gate, input gate, and the output gate; these gates determine what information is important and uses them to update the cell state and hidden state. The forget gate combines the previous day's hidden state and current day's input with a sigmoid to determine what information is currently important; these values are then multiplied by the cell state vector, reducing what features are currently are unimportant. The memory

cell uses a multiplied combination of sigmoid and hyperbolic tangent (tanh) functions to update the current state of the cell. Finally, the output gate passes the resultant current state through a tanh function and multiplies it by the input data passed through a sigmoid in order to create a daily prediction as well as form the new hidden state. (See Figure 1.) The overall benefit of this structure is time-varying data that knows when to remove unimportant data as soon as it becomes irrelevant, which is ideal for stock prediction. [12, 13]

1.3 Convolutional Neural Networks

Convolutional neural networks (CNN) have historically been used for image processing and other spatial problems. For example, EfficientNet-B7, an image classification algorithm, achieved ImageNet Top 1 and Top 5 accuracies of 84.3% and 97.0%, respectively, using CNN. [14] However, Sezer and Ozbayaglu presented a new use of CNN to account for financial time series by forming 15x15 images using financial indicators—measures of stock quality based on daily opens, highs, lows, closes, and volumes (often referred to collectively as OHLCV). The major benefit of CNN in this context is that the convolutional layers are adept at picking up patterns in data, especially when combined with financial indicators. Then pooling layers can then pass these features through activation functions, such as ReLU or tanh, in order to determine what trends are important and to what degree. (See Figure 1.) The resultant product is a neural network that can pick up on both short-term and long-term trends, which is especially useful when trying to maximize return on investment (ROI). [11, 15]

Given the formations of both of these networks, Lu et al. proposed that a combination CNN and LSTM network would exploit the benefits of both types: time-dependent data processing from LSTM and key feature selection from CNN. [13] In this project, we will create and optimize a CNN-LSTM network for individual stock prediction with the goal of maximized ROI. This novel network will be compared again LSTM and CNN benchmarks to determine if the CNN-LSTM combination confers the advantages of both individual networks. After formation and optimization, the three neural networks will be compared by being trained using 10 years of QQQ historical data from Yahoo! Finance and then assessed based on ROI over a 1-year period; the primary output measures will training time, accuracies of, and ROI of the CNN, LSTM, and CNN-LSTM deep neural networks.

2 Problem Definition

In this project, we will compare three state-of-the-art deep neural networks used for financial stock predictions: (1) financial indicator-based convolutional neural networks proposed by Sezer and Ozbayaglu, [11] (2) the industry-standard long short-term memory recurrent neural network proposed by Moghar and Hamiche, [12] and (3) the novel combination CNN-LSTM model proposed by Lu et al but expanded upon using indicator-based images. [13] All three will be trained with the same financial dataset—QQQ Trust Series 1—from January 1, 2009 to December 31, 2019 (2768 periods) and will be evaluated using a 1-year test from January 1, 2020 to December 31, 2020 (252 periods). All will use buy-hold-sell implementation, in which financial trades will be made based on output nodes predicting if the user should buy, hold, or sell their stock. CNN, LSTM, and CNN-LSTM will be given a simulated $10,000 seed to invest over the test year and will be evaluated primarily for their abilities to handle variance and overall ROI. The only outside stipulations are that shares can be partial values (as in networks can have fractions of shares), single trades cannot exceed $3,000, assets in cash can never dip below $500, and models can never have fewer than 10 shares once that amount is bought. All of these stipulations were added to give the simulation a more realistic feel to actual stock market trading.

2.1 CNN-Specific Implementation

The input data will be converted from parsed Yahoo! Finance JSON data into 15x15 tensors, mimicking the structure of a typical image. The images-like tensors will represent 225 financial indicator instances derived from 15 indicator types—RSI, SR, WR, MFI, BB, ATR, KC, EMA, DEMA, CCI, ROC, MACD, EMV, FI, and OBV (See Appendix A.); variations of the same indicator will be used, just with different frame sizes (e.g., an EMA with a 10-day period versus a 20-day period). The 225 financial indicator instances are selected from a pool of 331, which will be normalized from a range of [0,1] among similar indicators and then selected using ANOVA. The total training set size is 2768 periods passed into 100 epochs, amounting to about 28 periods per epoch. CNN is then tested using the entire 2020 year, or about 252 working days, and is evaluated for buy-hold-sell prediction, which determine how the network will buy and sell stock. We will implement some deviations from the original paper [11] to optimize model performance. Some deviations we will incorporate include the type of indicators used, feature selection, and the normalization of the data. [15] We will also hyper-tune the model parameters to optimize model performance.

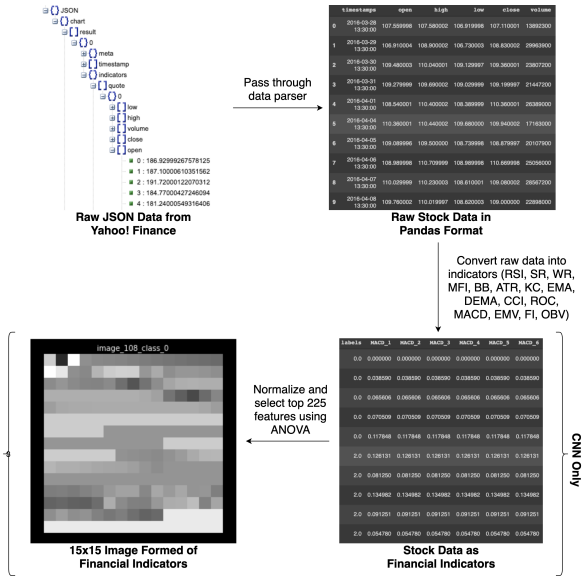

Figure 2: Data flow for LSTM and CNN networks. LSTM networks take in raw OHLCV stock data, while all CNN-based networks take in technical indicator images.

2.2 LSTM-Specific Implementation

Like CNN, the input data are pulled from Yahoo! Finance and will represent the same 2768 financial periods for the QQQ stock. However, the data are not converted into financial indicators but rather left kept as OHLCV data points and is passed in as 1x7 vectors formed of the same OHLCV values as well as difference and percent difference of close values. The LSTM network will similarly be trained for 2768 periods passed in 50 epochs, with about 55 periods per epoch. The neural network attempts to predict the following day's value and uses that data to determine the buy-hold-sell using a 10-day sliding prediction window. Finally, LSTM is then tested for the 2020 year, or about 252 periods, and is evaluated for its ROI.

2.3 CNN-LSTM-Specific Implementation

The CNN-LSTM model was inspired by a paper by Lu et al. [13] In our version, we will be combining a CNN model with an LSTM network to check its accuracy rate. The model first takes input JSON data from Yahoo! Finance API through Pandas dataframe. The model is again trained with data from January 2009 to December 2019 (2768 periods). The data has six columns, namely timestamps and OHLCV values. The data is put through a data normalization layer to obtain values between 0 and 1. Using TensorFlow's Sequential model, the data is processed. The input shape is of size (2002, 10, 15, 15, 3). The 10 corresponds to the time step used for grouping of the 15 by 15 images that are eventually feed into the LSTM layer. This is fed into two consecutive convolution layers with ReLU activation functions. The output vector is fed to a max pooling layer followed by a flattening layer. This output is then fed to the LSTM layer with a ReLU activation function. It contains 64 units within it and its output is fed to a dense layer with a softmax activation function. The output of the dense layer is of size (None, 3) and represents the probability distribution the model should buy, hold, or sell. The model uses a sparse categorical cross entropy loss function and an Adam optimizer when training. It has an approximate batch size of 64 and is run for a 100 epochs. The result is then back-propagated and run until the conditions are satisfied.

3 Data

All data has been pulled from Yahoo! Finance, which keeps real-time and historical stock data for several global stock exchanges, including NYSE, Nasdaq, Shanghai Stock Exchange, and HKEX. Stock values are updated in real or slightly delayed times from direct lines from all covered stock exchanges, with a refresh rate of 120 Hz. Yahoo! Finance stores daily financial data from 1970 to the current day; this project will only use daily OHLCV values.

The test set used in this project is QQQ Trust Series 1 (QQQ), which is a technology-centric ETF that includes differential share percentages of Apple, Microsoft, Amazon, Tesla, Facebook, Alphabet/Google, Nvidia, Adobe, and

PayPal. The data range used is from January 1, 2009 to December 31, 2020, representing 2768 periods. The incoming data is downloaded in JSON form, so a project-specific RESTful API was created to convert the data into Pandas Data Type tables. The chosen daily variables to be recorded are opening stock price ("open"), closing stock price ("close"), daily high value ("high"), daily low value ("low"), and stock trade volume ("volume"). All of these values can be pulled directly from Yahoo! Finance data. (See Figure 2.)

3.1 Converting Raw Data into Technical indicators

These daily market values are then converted into technical indicators (CNN and CNN-LSTM only) commonly used for stock prediction and are divided into four subcategories: momentum, volatility, trend, and volume. Momentum indicators determine the strength of stock by the rate its price raises or lowers; these indicators are particularly strong during bullish markets. [16] The four momentum indicators used in this project are relative strength index (RSI), stochastic oscillator (SR), Williams %R (WR), and money flow index (MFI). [17, 18, 19, 20] Volatility indicators determine the dispersion of stock values for a given period of time, with higher volatility connoting higher risk and lower volatility connoting lower risk. [21] The three volatility indicators used in this project are Bollinger Bands (BB), average true range (ATR), and Keltner channels (KC). [22, 17, 23] Trend indicators use momentum and direction to determine the likely path of stocks; most trend indicators are based on moving average, which flattens data over a given period of time, forming lagging prediction. [24] The five trend indicators used in this project are exponential moving average (EMA), double exponential moving average (DEMA), commodity channel index (CCI), price rate of change (ROC), and moving average convergence divergence (MACD). [25, 26, 27, 28] Volume indicators use how much asset trading occurs in a certain time to determine the strength of the current trends of a given stock; volume indicators are most commonly used to affirm or conflict the trends of other indicators. [29] The three volume indicators used in this project are ease of movement (EMV), force index (FI), and on-balance volume (OBV). [30, 31, 32] (See Appendix A.)

Collectively, raw daily stock data is converted into these indicator values and positioned into a 15x15 tensor composed 225 indicators. The structure imitates a typical image that would be passed into a CNN, meaning that indicator position is part of the optimization process. (See Figure 2.) The 225 indicators are chosen from a pool of 311 potential indicators formed from the 15 indicator types with various input ranges. The indicators are then normalized by indicator type and then selected using the ANOVA statistical model.

3.2 Forming the True Values

The goal the neural network is to determine when to buy, sell, and hold stocks in order to maximize profit. Therefore, the neural network is compared against "buy", "sell", and "hold" signals determined by local maxima and minima. Local minima are defined as prices that are lower than their 10 closest neighbors (5 before and 5 after); local minima are labelled as "buy" signals. Similarly, local maxima are defined as prices that are higher than their 10 closest neighbors; local maxima are labelled as "sell" signals. All other values are labelled as "hold" signals. These labels are compared against the predicted labels of the neural network to measure accuracy.

4 Methodology

All layers are designed using the TensorFlow Core v2.4.1 open-source deep learning library and implemented using Google Colaboratory's Nvidia K80 12GB 0.82GHz GPU processors. The six layer types used are from `tensorflow.keras.layers`: `Conv2D` (2D convolutional layers), `LSTM` (LSTM layers), `Dropout` (dropout layers), `Dense` (dense layers), `Flatten` (flatten layers), and `TimeDistributed` (time slicing layer). Each neural network is formed using TensorFlow's `Sequential` model creator, which sequential adds layers to the deep neural network.

4.1 Convolution Layers

Modeled after the human visual system, convolutional layers break down spatial data into smaller patterns that can describe trends. (See Figure 3A.) Convolutional layers are based off of the mathematical concept of convolution: spatially close data impact one another; therefore, trends in data should be derived from the combined effect data points have on one another, not just the data points themselves. Therefore, convolutional neural networks can be described using the generic mathematical convolution:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

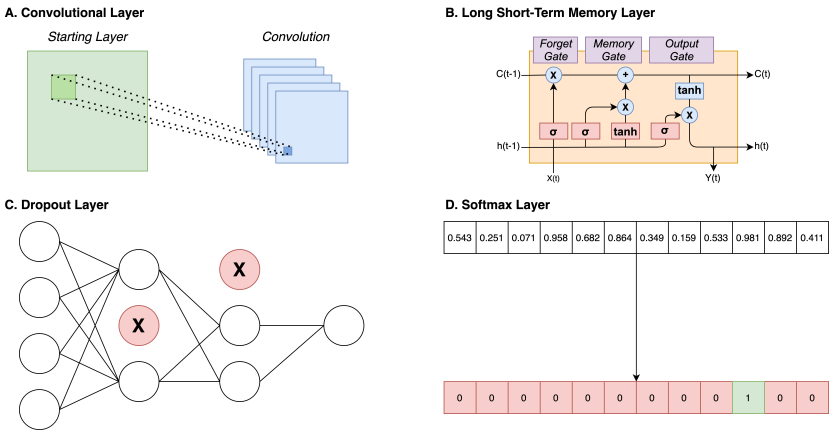

Figure 3: Four of the major types of neural network layers used in the CNN, LSTM, and CNN-LSTM deep neural networks: (A) convolutional layers, (B) LSTM layers, (C) dropout layers, and (D) softmax layers (a subtype of dense layers).

The output is composed of two interacting inputs taken from different locations, both separated by τ. By taking the integral over infinite space, the combined affect of both of these inputs are computed together for all space on a given input. Convolutional layers harness this convolution theorem to describe the combined effects of spatially-close inputs have on one other, allowing neural networks to not only find patterns in how individual points but also with overall trends subsets of data create.

Inputs for convolutional layers are described to be images because they are location-dependent. The output of convolutional layers are kernels, which are smaller images that extract features of the larger image by convolving around certain areas of the input image. Convolving is described as following:

$$f(X) = \sigma(W^T \cdot X + B)$$

In which X is an input of vector containing a subset of data taken from the image. W and B are trained weight vectors and intercepts, receptively, that are combined with X in order to form values that assess an internal data pattern "learned" by the neural network. Both W and B are trained using back-propagation and dictate the different patterns that the layer has extracted from the data. Finally, convolutional layers are passed through an activation function (σ) in order to drive individual points toward extreme states without destroying the derivative (and by extension back-propagation's gradient). Activation functions can vary based on the needs of layer, with sigmoid (σ), hyperbolic tangent ($tanh$), and rectified linear unit ($ReLU$) being those used in this project.

4.2 Long Short-Term Memory Layers

While convolutional layers extract spatial trends, LSTM layers observe trends over time and are especially helpful for time-dependent data. (See Figure 3B.) LSTM layers are made of two states: previous state (h) and cell state (C). Previous state is a generic feature of recurrent neural networks that holds the output from the LSTM cell one time-step earlier; this allows for LSTM cells to combine time-varying effects with the current data point when forming the output of the cell. Cell state is a unique feature of LSTM cells and is entirely internal; its primary function is to contribute to forget mechanisms, a tool used to determine when certain temporally old information should be remembered or forgotten. Cell states also provide uninterrupted gradient flow, allowing for better back-propagation during training.

LSTM cells are composed of three gates fundamental to handling temporal data: forget gate, memory gate, and the output gate.

$$C'(t) = \sigma(W_f \cdot [h(t-1), X(t)] + B_f)$$

The forget gate uses the previous state and the new input to determine what parts of the cell state should be remembered and what parts should be forgotten. The combined previous state and new input are passed through a sigmoid activation function to essentially turn off or on certain features, which alters the cell state when this vector is multiplied by the cell state. This mechanism all allows the cell state to remember only important features.

$$C(t) = C'(t) + \sigma(W_i \cdot [h(t-1), X(t) + B_i]) \cdot tanh(W_C \cdot [h(t-1), X(t)] + B_C)$$

The memory gate determines what the new cell state is going to be using the input and previous state information and the freshly altered cell state. Similar to the forget gate, the input data and previous use a sigmoid activation function to determine which features are turned on and off; this is combined with the same two features passed through a tanh activation function to form new values to add or subtract from the memory state. Finally, all of this data is added to the existing cell state, forming the new set of remembered features in the cell state.

$$Y(t) = \sigma(W_o \cdot [h(t-1), X(t) + B_o]) \cdot C(t)$$

The output gate determines its output data and the new previous state, both of which are the same value. The output is determined by finding the dot product between the sigmoid-activated input and previous state and is then added to the current cell state. This output, therefore, takes into consideration the new data input, current trends in the data through the previous state, and long-term features that the cell remembered through the cell state. This allows the output to not only be impacted by the new information and closely related temporal data but also information relating to overall motifs of the cell at that current time.

4.3 Dropout Layers

Dropout layers are primarily used for training more robust deep neural networks by randomly selecting certain nodes to be shut off for a training iteration. This mechanism adds to the stochasticity of network and prevents certain nodes from dominating output determination for a given layer, therefore spreading prediction more robustly across the entire network. This is especially beneficial for creating network that can handle a variety of situations and forces them to think broadly what patterns most impact the output. (See Figure 3C.)

4.4 Flatten and Dense Layers

Flatten and dense layers are both used to reshape data vectors to make them more accessible to outputs. Flatten layers reform vectors of higher dimensions to lower dimensions, and dense layers lower the number of variables in the current working vector. For example, if the output desired is a 3-long 1D vector deciding buy-hold-cell, flattening layers can first convert 2D layers into 1D. Then, dense layers combine the data until these three points are achieved. This is done using a couple of mechanism, like max pooling (choosing the largest point within a certain range of points) and softmax (making the largest point in the vector 1 and all others 0). (See Figure 3D.)

4.5 CNN Architecture

The CNN model implementation is described in Figure 4. The dataset is extracted using the method `getStockData`, which downloads JSON data from the Yahoo! Finance API and organises it in a Pandas dataframe. It contains two parameters, the ticker (chosen stock) and the range (defines the time range of the stock) and returns a Pandas dataframe that contains the OHLCV values and the timestamp of the ticker. The algorithm uses a 11 day sliding window to determine whether to buy, hold, or sell the stock. If the middle value of this window is the highest, the stock is labelled as sell; if it is the minimum, it is marked as buy. All other values are simply hold. This forms the true values that the CNN network is trained to.

The technical indicators used in this model are RSI, MACD, EMA, SO, WR, MFI, BB, ATR, KC, DEMA, CCI, EMV, FI, PRC, and OBV. The indicators form a 15x15 image that are input in to the CNN network, allowing for spatially-related data. The dataframe is divided into a training set and a test set. The training set includes data from January 1, 2009 to December 31, 2019 and the test set includes data from January 1, 2020 to December 31, 2020. The 225 financial indicator instances are selected from a pool of 331, which will be normalized from a range of [0,1] among similar indicators and then selected using ANOVA. The convolution network uses `Sequential` to build the model layer by layer. The model has two `Conv2D` layers. These are convolution layers that deal with input images which are seen as 2D matrices. The number of nodes in the first layer is 32 and the second layer is 64. The kernel size is the size of the filter matrix for convolution. Therefore a kernel size of 3 will show a 3x3 matrix for convolution. The activation function used is ReLU. Between the two Conv2D layers is a Dropout layer with a rate of 0.2. Next comes the MaxPooling layer which helps reduce the spatial size by extracting dominant features in the convolved feature. This is done to decrease the computational power required to process the data. After adding another dropout layer we flatten the entire image into a 1D column vector. An element wise ReLU activation followed by an element wise softmax activation is acheived by adding two Dense layers. The last Dense layer has an output size of 3, which indicates whether to buy, hold, or sell the stock. A dropout layer is introduced between the two dense layers to reduce overfitting and improve the generalisation of the model.

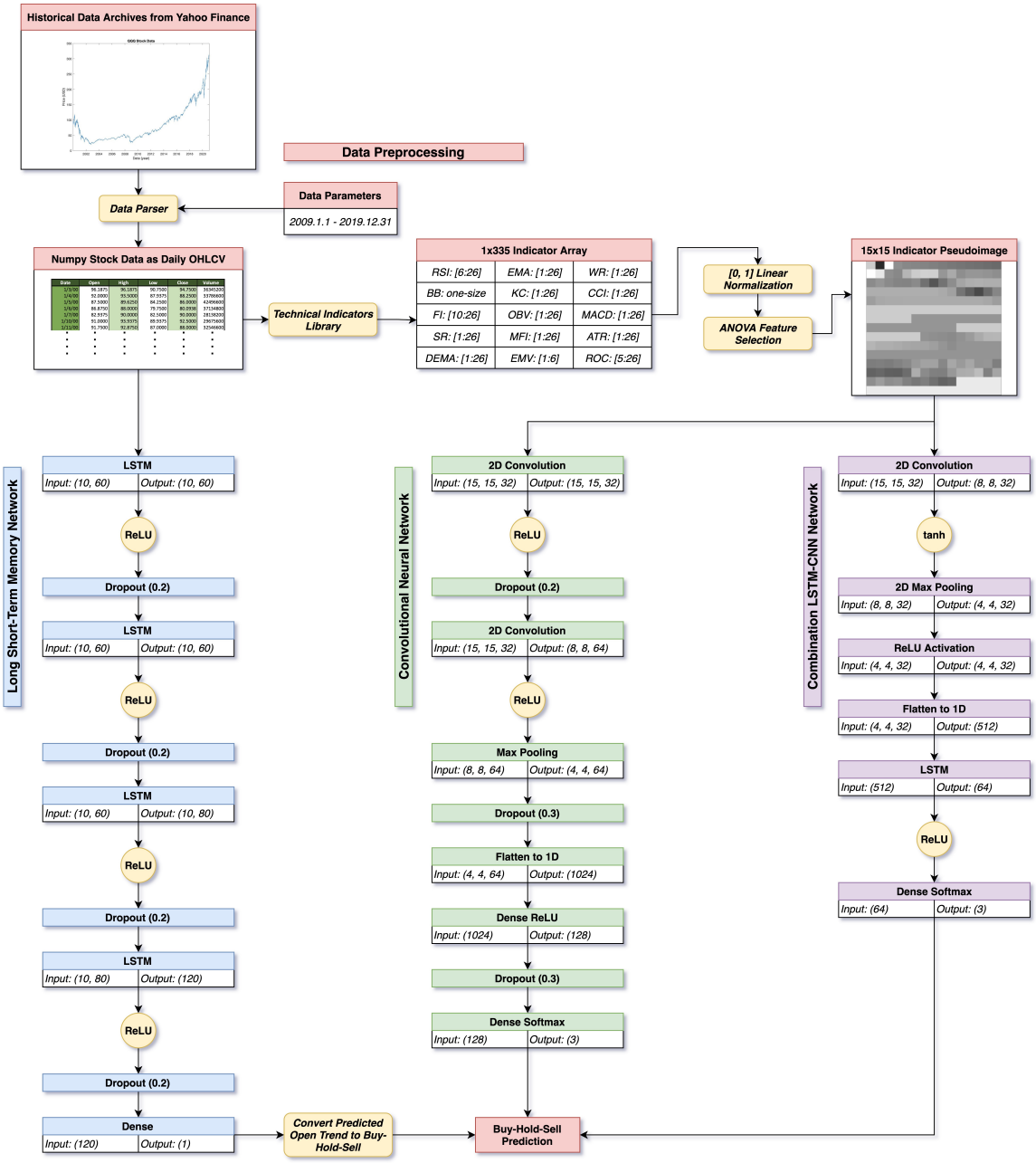

Figure 4: The overall flow for CNN, LSTM, and CNN-LSTM networks. All networks are from the same data extraction step, but only the CNN and CNN-LSTM networks get this raw OHLCV data converted into technical indicators. Then, for each network is a detailed flow of layers that form the network. Both the CNN and CNN-LSTM end with buy-hold-sell prediction, while the LSTM network must used an external method to get converted for prediction.

4.6 LSTM Architecture

The LSTM architecture contains four LSTM layers each followed by a dropout layer and lastly a dense layer. (See Figure 4.) The LSTM layer uses a unit parameter-a positive integer indicating the dimension of the output space-, an activation parameter that indicates the activation function-in this model, ReLU-, and a return sequence parameter that is set to return the last output in the output sequence. The training data consists of a panda's data frame that contains QQQ stock price data (OHLCV as well as difference and percent difference of close) from January 1, 2009 to December 31, 2020. This QQQ price stock data was obtained from the same Yahoo! Finance API used in the other two models. The training data is then is normalized to [0, 1] and then modified with a time step of 10. The time step is used to group the time series data before feeding the data into the model. The larger the time step value is, the more data points there are for the LSTM model to look back on to make a prediction from. The final shape of the training data is (2505, 10, 7) and (2505,). After the training data has been created it is then fed into the LSTM model and outputs a (2505, 1) array that predicts the close price for the next day from the given training data. The accuracy of the model is determined by comparing the change in the predicted close price to the change in the actual close price. If both the predicted price and actual price increase from their previous price, then that is counted as a correct prediction. A trading algorithm is used to evaluate the model at making a profit. This algorithm consists of finding the change in between the predicted current and previous price. If the percent change is greater than 1% then the model would expect the price to increase, shares are bought; if the change is less, then -1.5% then the model would expect the stock price to decrease, and shares are sold. The number of shares to be bought or sold is determined the current cash on hand and the actual previous price of the stock.

4.7 CNN-LSTM Architecture

The CNN-LSTM model implementation is described in Figure 4. This model has shares similarities with the CNN architecture previously described. The dataset is extracted using the method getStockData, which downloads JSON data from the Yahoo! Finance API and organizes it in a Pandas dataframe. It contains two parameters, the ticker (chosen stock) and the range (defines the time range of the stock) and returns a Pandas dataframe that contains the OHLCV values and the timestamp of the ticker. The algorithm uses a 11-day sliding window to determine whether to buy, hold, or sell the stock. If the middle value of this window is the highest, the stock is labelled as sell; if it is the minimum, it is marked as buy. All other values are simply hold. This forms the true values that the CNN network is trained to.

The technical indicators used in this model are RSI, MACD, EMA, SO, WR, MFI, BB, ATR, KC, DEMA, CCI, EMV, FI, PRC, and OBV. The indicators form a 15x15 image that are input into the CNN network, allowing for spatially related data. The dataframe is divided into a training set and a test set. The training set includes data from January 1, 2009, to December 31, 2019, and the test set includes data from January 1, 2020, to December 31, 2020. The 225 financial indicator instances are selected from a pool of 331, which will be normalized from a range of [0,1] among similar indicators and then selected using ANOVA. After feature selection, the images from the testing/training data are grouped together using a window size of 10 and incremented by one time step until all images have been grouped. The input for the model is of size (2002, 10, 15, 15, 3) The convolution network uses Sequential to build the model layer by layer. The model has two Conv2D layers. These are convolution layers that deal with input images which are seen as 2D matrices. The number of nodes in the first layer is 32 and the second layer is 64. The kernel size is a 3x3 matrix. The activation function for both layers is ReLU. To satisfy the temporal slicing requirements of LSTM, the Conv2D layers must both be wrapped in a TimeDistributed layer before being added to the model. Next comes the MaxPooling layer which helps reduce the spatial size by extracting dominant features in the convolved feature. This is done to decrease the computational power required to process the data. Before the MaxPooling layer is added to the model, it too must be wrapped in a TimeDistributed layer. A ReLU activation layer is then added to the model. After this activation layer, a Flatten layer wrapped in a TimeDistributed layer is added. A LSTM layer with 64 units and a ReLU activation layer is then added. Finally, a Dense layer with an output size of 3 and a softmax activation function is added. The three outputs are a probability distribution, which indicates whether to buy, hold, or sell the stock.

5 Results and Discussion

5.1 Training the Deep Neural Networks

In terms of training time, all networks performed comparably, with all finishing training in less than 3 minutes. By far, the fastest trained is the CNN network, which only took a total of 5.232s, with an average training time per epoch of 12.2ms. Both LSTM and CNN-LSTM are slower to training, at 1.040s/epoch and 1.098s/epoch, respectively. However, LSTM takes about half the time to train as CNN-LSTM, at 85.96s compared to 135.01s, primarily because the LSTM requires only half the amount of epochs to be sufficiently trained without being overfit. (See Figure 5A.) That being

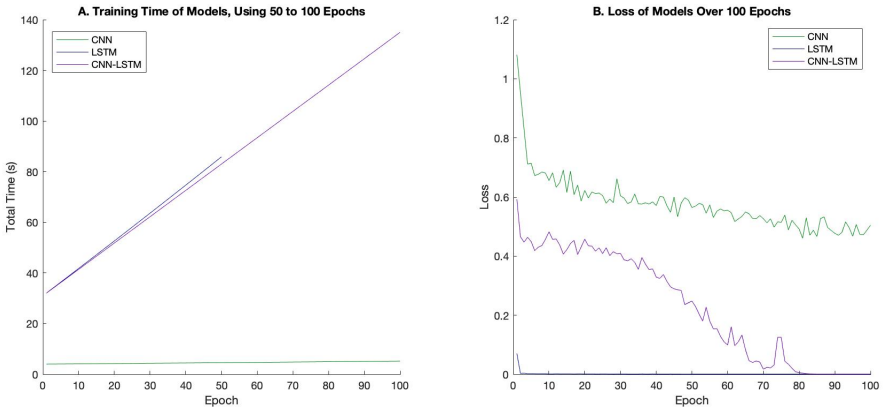

Figure 5: (A) Training time and (B) loss over 100 epochs for CNN and CNN-LSTM networks and 50 epochs for LSTM network.

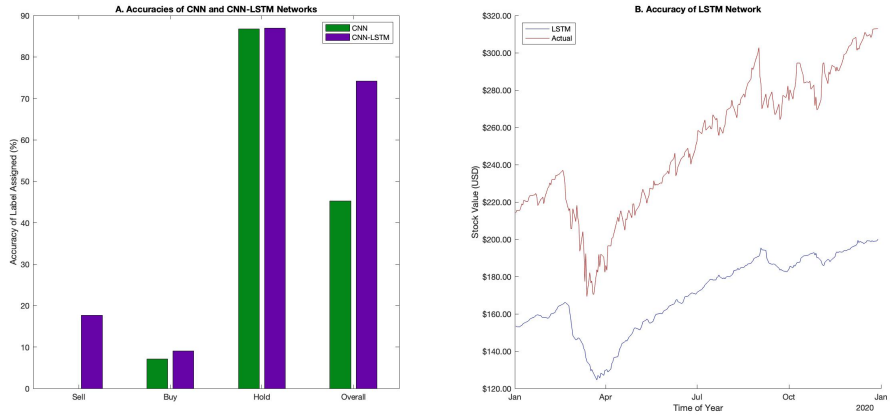

Figure 6: (A) Accuracy comparison between CNN and CNN-LSTM networks based on prediction of buy, hold, or sell markers. (B) Comparison of LSTM predicted stock value against real stock value.

said, the difference in training time is relatively inconsequential considering that all took a short time to train in order to produce serviceable networks.

Both LSTM and CNN-LSTM converge to loss values close to 0, ending 8.08e-4 and 1.34e-4, respectively, while the CNN network never comes close to converging to 0, achieving a final loss function 0.5045. (See Figure 5B.) Notably, LSTM converges much faster than CNN-LSTM, yet both models closeness to 0 indicate that both are sufficiently trained. CNN would benefit from being trained more to fully uncover the internal mechanisms of the QQQ stock, so any future uses of the CNN network would benefit from larger training datasets as compared to its LSTM and CNN-LSTM counterparts.

5.2 Accuracy

None of the networks particularly exceeded at accurately matching exactly the predicted times when stock should be bought, sold, or held. The most accurate network is CNN-LSTM with 74.206%, while the least accurate is CNN with 45.238%. The LSTM network performed comparably to LSTM 52.800%. While the CNN-LSTM network performed much better than the other two networks, none performed particularly well.

Similarly across the CNN and CNN-LSTM networks, both networks were particularly poor at predicting the more extreme indicators, namely buy and sell. (See Figure 6A.) For sell, the CNN and CNN-LSTM achieved accuracies of only 7.09% and 9.09%, respectively; for buy, the CNN and CNN-LSTM achieved accuracies of 0.00% and 17.65%, respectively. While certainly not ideal, these low accuracies do not necessarily confer that these networks do not

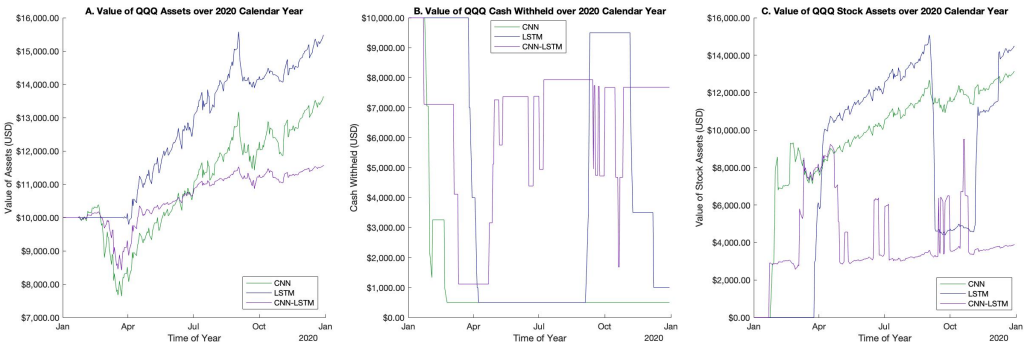

Figure 7: CNN, LSTM, and CNN-LSTM networks are given the 2020 calendar year to maximize the stock profit from QQQ. Each network is given a seed money of $10,000, and they can trade when they predict are the most optimal times. The networks are then compared for (A) overall growth of assets, (B) amount of assets in cash, and (C) amount of assets in stock.

successfully profit from a given stock. In many cases, both networks would predict buy or sell within a 5 day margin of what would be most ideal, and for the intents of maximizing profits, this usually does not confer substantial losses in profits. Both the CNN and CNN-LSTM performed relatively well when predicting when the stock to should be held, with both accuracies between 86% and 87%. This further supports that often these networks predict around the correct time to buy and sell and hold onto their stocks during major fluctuations in price. Overall, while neither network had outstanding accuracy, they both predict buy, hold, and sell markers close enough to peak and valley times that they are still successful in profiting of the stock.

LSTM network could not easily be compared to the CNN and CNN-LSTM networks because it conducts trades using the 1%/-1.5% incline/decline rule rather than using explicit buy, hold, or sell markers. Instead accuracy is better assessed by how well the LSTM network matches the trends of the actual data for QQQ during the 2020 calendar year. (See Figure 6B.) The most notable error is that the LSTM network consistently underestimates the value of stocks by around $100; however, this is not a major issue considering that the trading mechanism is based on changes in value rather then specific values themselves. In regards to matching general trends of the real data, the LSTM network excels at predicting the shape of the trends, with the biggest issues arising from matching more specific day to day trends. By being able to match the long-term trends, the LSTM network is particular keen at accurate investments for long-term trades but struggles more when day-by-day trading. Therefore, it would be more advisable to use the LSTM for long-term investments.

5.3 Practical Test of Profit

While accuracy is a relatively strong indicator how each of these models match the true values, the best indicator of the success of these networks are to compare how well they would fare when given seed money to trade stocks. Each network has been simulated to have $10,000 to invest in stocks using QQQ data from the 2020 calendar year. The LSTM network made the most money, achieving an ending portfolio value of $15,477.66, while CNN-LSTM performed by far the worst, ending with a portfolio value of $11,563.67. The CNN network performed firmly in between with $13,626.01. (See Figure 7.)

Not surprisingly, the LSTM network continues to perform well in stock-market price prediction and trading, not surprising considering that LSTM networks are keen at finding trends in time-dependent data. In particular, the LSTM networks has the unique ability of predict how the stock price will trend rather than attempt to determine a buy, hold, or sell marker. This simple change in output allowed the LSTM to use the unique method of withholding trades until the predicted change in price is greater than 1% or -1.5% (depending on direction), which made it overall more hesitant to trade its stocks. One of the biggest failures of the CNN-LSTM network, and to lesser extent the CNN network, is that traded out its stocks too often, mitigating potential profits. This key trading method made LSTM more adept at finding only the most ideal times to trade. As indicated by the LSTM network, the 1%/-1.5% trade method is most robust to big trade moments.

The CNN model had moderate success using the financial indicators, but its biggest drawback is that it was hesitant to trade its stock after it converted all of its available funds. By February in the simulation, the CNN network had nearly all of its assets in stock and did not sell beyond that point. This case expounds that the CNN network requires

far more training sets than the other models, as its accuracy at picking up buy and sell markers are highly inaccurate. Additionally, it is the only network not converge toward 0 loss while training, further supporting that the CNN requires a much larger training set. This should be a real consideration to those using this network that relatively new stocks or stocks with limited historical data would not perform well with CNN due to its inability to be fully trained.

By far, the biggest failure is the CNN-LSTM network, which only made about $1,500 despite the plentiful opportunities to profit. The biggest issue with the CNN-LSTM is simply that it traded far too often to gain any substantial profit. Whereas the LSTM network could find only the most robust trading opportunities, the CNN-LSTM was predicted far too often buy and sell markers, with low specificity with both markers. Unlike CNN, the CNN-LSTM network achieved substantial training as indicated by its loss converging to 0, so the CNN-LSTM model itself is not adept at predicting when to trade. Interestingly, further testing indicates that one of the major inhibitors in CNN-LSTM is the use of financial indicators instead raw stock data. In an outside test (not depicted), a CNN-LSTM network was created using a 1D vector of OHLCV data, as presented by Lu et al. [13] It predicted the stock price using the same method as the LSTM network and achieved the most profit out of any network: $16,311.02. This indicates that combining the CNN and LSTM networks actually a beneficial concept because the network can pick up on both relational and temporal data as provided by CNN and LSTM networks, respectively. The biggest difference the two CNN-LSTM networks is the use of financial indicators as the input values. This inclusion majorly hindered the CNN-LSTM network and prevented its potential success. Even in the CNN network, which had moderate success, the financial indicators, even when optimized into the images, not only did not confer any advantages but even hindered the success of the networks.

6 Conclusion

The paper compares the accuracy of three models CNN, LSTM and CNN-LSTM for predicting stock prices.The CNN and CNN-LSTM models predict whether to buy, hold, or sell the stock, whereas the LSTM model predicts the trend of the stock price by comparing it to a certain threshold.The input data for all three models included time-invariant 'QQQ' stock data from Yahoo! Finance. The CNN and the CNN-LSTM models are fed technical indicators images as input as well. Upon training the models, the model with the highest forecasting accuracy was shown to be the CNN-LSTM model, followed by the CNN model and the LSTM model. Although the CNN-LSTM network showed highest accuracy it performed relatively poor in terms of profit gained on a given principle. The biggest drawback of the given model was that it traded quite often, hence predicted 'buy','sell' markers with very low specificity. The LSTM model made the largest amount of profit and the CNN model came somewhere in the middle. Since stock price prediction is a time-dependent feature, the LSTM model could predict the price trend of the stock. The LSTM model gave a good profit due to its minimized trading frequency.The CNN-LSTM model, although had a high accuracy rate, which caused it to miss out on profits due to its high trading frequency. Thus, the LSTM model can be considered as a relevant reference to investors looking to maximize their gain. Further research on better implementation of technical indicators in the CNN and CNN-LSTM models to be more robust could lead to improved outcomes.

References

[1] John J. Murphy. *Technical analysis of the financial markets: a comprehensive guide to trading methods and applications*. New York Institute of Finance, 1999.

[2] Yain-Whar Si and Jiangling Yin. Obst-based segmentation approach to financial time series. *Engineering Applications of Artificial Intelligence*, 26(10):2581–2596, 2013.

[3] George S. Atsalakis and Kimon P. Valavanis. Surveying stock market forecasting techniques – part ii: Soft computing methods. *Expert Systems with Applications*, 36(3):5932–5941, 2009.

[4] S Dhar, T Mukherjee, and A K Ghoshal. Performance evaluation of neural network approach in financial prediction: Evidence from indian market. *International Conference on Communication and Computational Intelligence*, page 597–602, 2010.

[5] J Chen. Svm application of financial time series forecasting using empirical technical indicators. *International Conference on Information Networking and Automation*, 1, 2010.

[6] Rodolfo C Cavalcante, Rodrigo C Brasileiro, Victor L.F. Souza, Jarley P Nobrega, and Adriano L.I. Oliveira. Computational intelligence and financial markets: A survey and future directions. *Expert Systems with Applications*, 55:194–211, 2016.

[7] X Ding, Y Zhang, T Liu, and J Duan. Deep learning for event-driven stock prediction. *International Joint Conference on Artificial Intelligence*, 24:2327–2333, 2015.

[8] Thomas Fischer and Christopher Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669, 2018.

[9] Christopher Krauss, Xuan Anh Do, and Nicolas Huck. Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the sp 500. *European Journal of Operational Research*, 259(2):689–702, 2017.

[10] P. Tino, C. Schittenkopf, and G. Dorffner. Financial volatility trading using recurrent neural networks. *IEEE Transactions on Neural Networks*, 12(4):865–874, 2001.

[11] Omar Berat Sezer and Murat Ozbayoglu. Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach. *Applied Soft Computing*, page 1–29, 2018.

[12] Adil Moghar and Mhamed Hamiche. Stock market prediction using lstm recurrent neural network. *Procedia Computer Science*, 170:1168–1173, 2020.

[13] Wenjie Lu, Jiazheng Li, Yifan Li, Aijun Sun, and Jingyang Wang. A cnn-lstm-based model to forecast stock prices. *Complexity*, 2020:1–10, 2020.

[14] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *36 th International Conference on Machine Learning*, 2019.

[15] A Nayak. Stock buy/sell prediction using convolutional neural network. *Towards Data Science*, 2020.

[16] Troy Segal. Understanding momentum indicators and rsi, Sep 2020.

[17] J. Welles Wilder. *New Concepts in Technical Trading Systems*. Hunter Pub., 1978.

[18] George C. Lane. *Using stochastics, cycles R.S.I.: –to the moment of decision –*. G.C. Lane, published under the auspices of Investment Educators, 1986.

[19] Larry Williams. *How I Made One Million Dollars... Last Year... Trading Commodities*. Windsor Books, 3 edition, 1998.

[20] G Quong and A Quodack. Volume-weighted rsi: money flow. *Technical Analysis of Stocks Commodities*, 7(3):76–77, 1989.

[21] Justin Kuepper. Volatility, Mar 2021.

[22] John Bollinger. *Bollinger on Bollinger Bands*. McGraw-Hill Education, 1 edition, 2001.

[23] Chester Keltner. *How to Make Money In Commodities*. Keltner Statistical Service, 3 edition, 1960.

[24] Cory Mitchell. Trend trading: The 4 most common indicators, Nov 2020.

[25] Patrick Mulloy. Smoothing data with faster moving averages. *Technical Analysis of Stocks and Commodities*, 12, 1994.

[26] Donald Lambert. Commodities channel index. *Commodities*, 1980.

[27] Valeriy Zakamulin. Anatomy of trading rules with moving averages: Anatomy and performance of trading rules. *Market Timing with Moving Averages*, page 71–101, 2017.

[28] Gerald Appel and Edward Dobson. *Understanding MACD*. Traders Press, 2008.

[29] Cory Mitchell. Keltner channel definition, Mar 2021.

[30] Richard Arms. Ease of movement. *Technical Analysis of Stocks amp; Commodities*, 8(1), 1990.

[31] Alexander Elder. *Trading for a Living: Psychology, Trading Tactics, Money Management*. Wiley, 1 edition, 1993.

[32] Joseph Granville. *New Key to Stock Market Profits*. Prentice-Hall, 1 edition, 1963.

A Financial Indicators

(1)	RSI	(i.)	$AG = \frac{\sum_n \begin{cases} Close(t_0 - 1) - Close(t_0), if\ Close(t_0) < Close(t_0 - 1) \\ 0, else \end{cases}}{n}$
		(ii.)	$AL = \frac{\sum_n \begin{cases} Close(t_0) - Close(t_0 - 1), if\ Close(t_0) > Close(t_0 - 1) \\ 0, else \end{cases}}{n}$
		(iii.)	$RSI_1 = 100 - \left(\frac{100}{1 + \frac{AG}{AL}}\right)$
		(iv.)	$RSI_2 = 100 - \left(\frac{100}{1 + \frac{Avg(AG[t_0 - n: -1]) * (n-1) + AG(t_0)}{Avg(AL[t_0 - n: -1]) * (n-1) - AL(t_0)}}\right)$
(2)	SR		$SR = \left(\frac{Close(t_0) - Min\big((Low[t_0 - n: t_0])\big)}{Max\big(High([t_0 - n: t_0])\big) - Min\big((Low[t_0 - n: t_0])\big)}\right) * 100$
(3)	WR		$WR = \left(\frac{Max\big(High([t_0 - n: t_0])\big) - Close(t_0)}{Max\big(High([t_0 - n: t_0])\big) - Min\big((Low[t_0 - n: t_0])\big)}\right) * -100$
(4)	MFI	(i.)	$TP = \frac{High(t_0) + Low(t_0) + Close(t_0)}{3}$
		(ii.)	$MFR = \frac{Countif(TP[t_0 - n: t_0], TP_i > 0)}{Countif(TP[t_0 - n: t_0], TP_i < 0)}$
		(iii.)	$MFI = 100 - \frac{100}{1 + MFR}$
(5)	BB	(i.)	$MB = \frac{\sum_n Close_i}{n}$
		(ii.)	$UB = MB + 2\sigma$
		(iii.)	$LB = MB - 2\sigma$
(6)	ATR	(i.)	$TR = Max\left(\big(High(t_0) - Low(t_0)\big), \lvert High(t_0) - Close(t_0 - 1)\rvert, \lvert Low(t_0) - Close(t_0 - 1)\rvert\right)$
		(ii.)	$ATR = \frac{\sum_n ATR_i}{n}$
(7)	KC	(i.)	$KMB = EMA\big(ATR(n)\big)$
		(ii.)	$KUB = MB + 2ATR$
		(iii.)	$KLB = MB - 2ATR$
(8)	EMA	(i.)	$EMA_0 = \frac{\sum_n Close(t_i)}{n}$
		(ii.)	$EMA = \left(\frac{2}{n+1}\right)\big(Close(t_0) - EMA(t_0 - 1)\big) + EMA(T_0 - 1)$
(9)	DEMA		$DEMA = 2 * EMA(n) - EMA\big(EMA(n)\big)$
(10)	CCI	(i.)	$TP = \frac{High(t_0) + Low(t_0) + Close(t_0)}{3}$
		(ii.)	$CCI = \frac{TP - MA(TP[t_0 - n, t_0])}{0.015 * \sigma(TP[t_0 - n, t_0])}$
(11)	ROC		$ROC = \frac{Close(t_0) - Close(t_0 - n)}{Close(t_0 - n)}$
(12)	MACD		$MACD = EMA(12) - EMA(26)$
(13)	EMV		$EMV = \frac{\frac{High(t_0) + Low(t_0)}{2} - \frac{High(t_0 - 1) + Low(t_0)}{2}}{\frac{Volume(t_0)}{\big(High(t_0) - Low(t_0)\big) * 10^8}}$
(14)	FI		$FI = \big(Close(t_0) - Close(t_0 - n)\big) * Volume(t_0)$
(15)	OBV		$BV = OBV(t_0 - 1) + \begin{cases} Volume(t_0), if\ Close(t_0) > Close(t_0 - 1) \\ 0, if\ Close(t_0) = Close(t_0 - 1) \\ -Volume(t_0), if\ Close(t_0) < Close(t_0 - 1) \end{cases}$