Stock Price Prediction based on Convolutional Neural Networks

by

Zhenrui Cheng

Matriculation Number 367964

A thesis submitted to

Technische Universität Berlin
School IV - Electrical Engineering and Computer Science
DAI-Labor

Bachelor's Thesis

June 15, 2023

Supervised by:
Prof. Dr.-Ing. habil. Dr. h.c. Sahin Albayrak

Assistant supervisor:
Prof. Marc Toussaint
Dr.- Ing. Stefan Fricke

Eidestattliche Erklärung / Statutory Declaration

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed.

Berlin, June 15, 2023 Zhenrui Cheng

Acknowledgments

I would like to thank Professor Albayrak, Professor Toussaint, and Dr. Fricke. Thank you for giving me this opportunity to write my undergraduate thesis. Thank you for your careful supervision of my thesis. I don't know what words to use to express my deep gratitude to you. It may be a trivial thing to you. But it means a great deal to me. This is an important turning point in my life. In the past few years, I have endured physical illness and psychological disorders. Especially during the pandemic, severe depression made me lose my confidence to continue living for a while. Thanks to the kind people who helped me. Especially Professor Albayrak, Professor Toussaint, and Dr. Fricke. Your great help has given me courage and hope. It has shown me that there are many meaningful things to experience in life. I will have the confidence to meet the challenges of life. Thank you to all of you who have helped me during this difficult and dark time. I will be forever grateful to you all.

Abstract

Stock trading is one of the very common forms of investment today. Investors are looking to make a profit from their stock investments. Traditional methods of investment analysis are no longer suitable for investors who lack professional skills and knowledge. Quantitative investment analysis methods are becoming increasingly popular among investors compared to traditional investment analysis methods. At the heart of the quantitative investment analysis approach is artificial intelligence technology. With the development of artificial intelligence technology, various machine learning algorithms are applied to the field of quantitative investment analysis. Common machine learning algorithms are back propagation (BP) neural network, recurrent neural network, convolutional neural network and their derivatives.

Convolutional neural network war developed in the 1960s. It has powerful data feature extraction capabilities. In recent years, thanks to the rapid development of GPU technology, convolutional neural network is not only widely used in vision, but also applied to the processing of big data, such as the analysis of large amounts of financial data in quantitative investment.

Here we try to make a prediction of the short-term price trend of stocks by constructing a suitable convolutional neural network model. In this thesis, the following issues are studied in depth.

First, 10 stocks from the Shanghai stock market with up to 10 years of historical data are selected to construct the data set by a specific method. The convolutional neural network model is trained with this data set and parameter tuning is performed.

Second, the constructed convolutional neural network model is compared with the random forest model and the long and short term memory network model for testing. Compare the advantages and disadvantages of various artificial intelligence algorithms.

Finally, the Bayesian ridge regression model is obtained by model ensemble.

By comparing the prediction results of different models, we can know that the convolutional neural network model is slightly better than the long and short-term memory network model and substantially better than the random forest model. The Bayesian ridge regression model is an optimization of the prediction results of the above three models. It gives satisfactory results for stock price forecasts.

Keywords: convolutional neural network, random forest, long and short-term memory network, model ensemble, Bayesian ridge regression

Zusammenfassung

Der Aktienhandel ist eine der gängigsten Formen der Geldanlage. Die Anleger wollen mit ihren Aktieninvestitionen einen Gewinn erzielen. Traditionelle Methoden der Investitionsanalyse sind für Anleger, denen es an professionellen Fähigkeiten und Kenntnissen mangelt, nicht mehr geeignet. Quantitative Investmentanalysemethoden werden bei Anlegern immer beliebter als herkömmliche Investmentanalysemethoden. Im Mittelpunkt des quantitativen Investmentanalyseansatzes steht die Technik der künstlichen Intelligenz. Mit der Entwicklung der Technologie der künstlichen Intelligenz werden verschiedene Algorithmen des maschinellen Lernens auf dem Gebiet der quantitativen Investitionsanalyse eingesetzt. Gängige Algorithmen des maschinellen Lernens sind Backpropagation (BP) neuronales Netz, rekurrentes neuronales Netz, convolutionales neuronales Netz und ihre Derivate.

Convolutionales neuronales Netz wurde in den 1960er Jahren entwickelt. Es verfügt über leistungsstarke Fähigkeiten zur Extraktion von Datenmerkmalen. In den letzten Jahren wurde dank der raschen Entwicklung der GPU-Technologie Convolutionales neuronales Netz nicht nur in der Bildverarbeitung eingesetzt, sondern auch bei der Verarbeitung großer Datenmengen, z.B. bei der Analyse großer Mengen von Finanzdaten für quantitative Investitionen.

Hier versuchen wir, eine Vorhersage der kurzfristigen Kursentwicklung von Aktien zu machen, indem wir ein geeignetes convolutionales neuronales Netzmodell konstruieren. In dieser Arbeit werden die folgenden Fragen eingehend untersucht.

Zunächst werden 10 Aktien des Shanghaier Aktienmarktes mit bis zu 10 Jahren historischer Daten ausgewählt, um den Datensatz nach einer bestimmten Methode zu erstellen. Mit diesem Datensatz wird das convolutionale neuronale Netzmodell trainiert und eine Parameteroptimierung durchgeführt.

Zweitens wird das konstruierte convolutionale neuronale Netzmodell mit dem Random-Forest-Modell und dem Lang- und Kurzzeitgedächtnis Netzmodell verglichen und getestet. Wir analysieren die Vor- und Nachteile der verschiedenen Algorithmen der künstlichen Intelligenz.

Schließlich wird das Bayes'sche Ridge-Regressionsmodell durch ein Modell-Ensemble ermittelt.

Die Untersuchung der Vorhersageergebnisse der verschiedenen Modelle zeigt, dass das convolutionale neuronale Netzmodell etwas besser ist als das Modell des Lang- und Kurzzeitgedächtnisses und deutlich besser als das Random Forest Modell. Das Bayes'sche Ridge-Regressionsmodell ist eine Optimierung der Vorhersageergebnisse der drei oben genannten Modelle. Es liefert zufriedenstellende Ergebnisse für Aktienkursprognosen.

Schlüsselwörter: Convolutionales neuronales Netz, Random Forest, Long Short-Term Memory, Modell-Ensemble, Bayessche Ridge-Regression

Contents

1 Introduction

Buying stocks is a convenient and quick way to invest your assets. More and more investors are looking to earn income through stock investments. But due to the lack of understanding of the capital markets and the necessary professional skills, especially risk control awareness, they often get the opposite result. A simple and scientific trading strategy is key in order for investors to make reasonable profits in the stock market.

In the stock market, in the short term, stock prices are in a state of random fluctuation. Nevertheless, when analyzed over a longer-term time period, stock price changes have a certain trend.

Traditional investment analysis methods analyze stock price trends from both fundamental and technical aspects of the stock. The main indicators used in fundamental analysis are price-to-earnings ratio (P/E), return on equity (ROE), price-to-book ratio (PBR), debt-to-equity ratio (D/E), etc. The technical analysis method focuses on the trend and channel analysis of the charts generated during the stock trading process. Traditional stock analysis methods are not suitable for amateur investors who lack professional skills and knowledge. And as the amount of information to be analyzed expands dramatically, the probability of errors in these analysis methods increases.

1.1 Research Background

With the geometric change in computing speed of computer CPU and GPU chips in accordance with Moore's Law and the rapid development of software algorithms, investment analysis by computer has become a popular method, known as quantitative investment. It is a new way to implement investment strategies based on mathematical models, fully utilizing computer technology as an aid. And it has been the most important investment analysis method today besides fundamental analysis and technical analysis.

In recent years, with the development of artificial intelligence technology, various machine learning algorithms have emerged, and their applications have gradually penetrated into every industry, including, of course, the field of quantitative investment analysis. Common machine learning algorithms are back propagation (BP) neural networks, recurrent neural networks, convolutional neural networks and their derivatives. Back propagation (BP) neural networks are mostly used in electrical engineering, chemical engineering and other fields. Recurrent neural networks are well adapted to language models, machine translation, text similarity comparison, speech recognition, etc. with the introduction of attention mechanism. Its derivatives, long and short-term memory networks, are also used in quantitative stock selection.

The prototype of convolutional neural networks was proposed in the 1960s. As a result

of its nearly demanding hardware requirements, it was not widely used in the early stage. Nonetheless, its powerful feature extraction ability and classification effect are well suited for visual information processing. Such as image recognition, target tracking, environmental monitoring and other aspects. In recent past, the rapid development of GPU technology and the leap forward in parallel processing capability have made its application in visual information processing gradually popular. But its application in the field of financial analysis is still in the initial stage. Financial data has a wide variety, broad coverage and large data scale. The convolutional neural network can filter out the effective feature values by eliminating a large amount of redundant data through the core convolution operation. Following the learning for effective feature values instead of all data, it can reduce the risk of over fitting in the first place, while improving the training efficiency. In addition the layout structure of convolutional neural networks is closer to biological neural networks compared to other artificial neural networks. It simplifies the complexity of the network using weight sharing techniques and improves the generalization of the model. This also gives it better compatibility for different periods and types of stock data. Based on such considerations, if there is a pattern in the process of stock price changes, then convolutional neural networks are certainly a good tool to discover such patterns.

1.2 Research Purpose and Relevance

The learner builds mathematical models by performing deep learning on large amounts of data. The utility model does quantitative timing for the targets selected by the investor and provides corresponding buy and sell time points for the investor to refer to.

Quantitative trading mainly includes quantitative stock selection, quantitative timing, commodity futures arbitrage, stock index futures arbitrage, options arbitrage and many other aspects. Quantitative timing is studied in this paper. This is because it plays a very important role in the stock trading process.

On the one hand, amateur investors do not have much time to spend on researching investments and watching the market. And inside the short term high frequency trading, the opportunity is fleeting. It is not easy to pinpoint the right time to buy and sell. After selecting a target stock, investors often incur unnecessary losses by choosing the wrong time to trade and making a reverse move. The combination of accurate mathematical models and appropriate trading strategies enables them to select the right time points for automated trading. As an aid to decision making, it undoubtedly provides greater certainty than spontaneous trading by investors, as well as great convenience.

On the other hand, investors have limited expertise. They have a natural disadvantage compared to neural networks trained through deep learning with big data. Quantitative trading using neural networks can improve the timeliness and accuracy of trading, reduce investor losses and increase the return on investment.

A proper trading strategy can also overcome the emotional weaknesses that investors have during the trading process. It also avoids chasing the ups and downs. Everything is done according to the established strategy and there is a greater probability of gaining excess profit margin.

Because of the reproducibility of its strategies and models, quantitative investment makes

a lot of sense for many public and private equity institutions. Often an excellent portfolio can bring huge economic benefits and social impact to investment institutions. Concurrently quantitative investment has a strong stability, saving a lot of human and material expenditure.

From another point of view, trying to apply convolutional neural networks, which have had great success in other fields, to financial investment analysis can also contribute some experience to its practice in financial prediction.

1.3 Research Status

Stock investment analysis and forecasting involves a great deal of financial knowledge, mathematical knowledge and the integrated application of computer science. It is also influenced by news, unexpected events, natural environment and other factors. So far, although the technology has progressed and the accuracy rate has increased, there is still no technology that can achieve satisfactory results in terms of reliability, stability and versatility. The main artificial intelligence methods currently used for stock and futures market analysis are data mining, stochastic processes, wavelet analysis, support vector machines, fractal theory, recurrent neural networks, and long and short-term memory networks.

The earliest application of neural networks to stock forecasting was in the late 1980s. White[1], a famous IBM engineer, first tried to apply back propagation (BP) neural network to predict the daily return of the company's stock. However, unfortunately, after learning from a large number of training samples, the results were found to be unsatisfactory. It was a long way from commercial application. Afterwards, through analysis, he concluded that the problem was mainly related to the failure of the neural network to converge to the extreme value. Later Kimoto T, Asakawa K and Yoda M [2] (1990) used neural network techniques to construct a model. Forecasting and regression testing of index trends on the Tokyo Stock Exchange had yielded good results. Bada and Kozaki [3] (1992) also operated a neural network model as a basis for forecasting some stock trends in Japan. The network model made use of 15 factors related to stock prices as input layer and it also had two hidden layers. This model had proved to be a good predictor for stocks that follow trends. Fama and French [4] (1993) proposed the famous three-factor model. They argued that market capitalization, book-to-market ratio, and price-to-earnings ratio in the model could better explain the disparity in returns of different stocks than Beta. The forward neural network model of Gencay [5] (1996) worked the moving average price of the Dow Jones Industrial Average as an input layer to predict the index between 1967 and 1988. The experimental data revealed that the neural network provided better forecasts than linear regression. Carhart [6] (1997) added momentum factor to the Fama-French three-factor model. He believed that in addition to size, value and market factors, momentum also had a significant impact on the long-term performance of securities. Leo and Adele [7] (2001) first implemented algorithmically the random forest concept proposed by Tin Kam Ho of Bell Labs in 1995. This pushed the development of artificial intelligence to a new level. G. Peter Zhang [8] (2003) compared the predictive power of ANN and ARIMA models on a time-ordered data set. The experiments demonstrated that the neural network ANN outperformed the regression model in performing nonlinear processing. Meanwhile, the organic combination of the two, complementing each other's strengths, had significantly improved the prediction accuracy. Qing Cao [9] (2005) et al. studied the Fama-French's three-factor model with neural networks using the Chinese securities market as a sample, and the

results proved that the neural network model predicted much better than the three-factor model. Ozbayoglu A.M [10] (2008) experimentally compared the accuracy of neural networks in predicting the securities market over Bayesian models. Haifan Liu and Jun Wang [11] (2011) researched the Chinese stock market with back propagation (BP) neural networks and wavelet analysis. They found that the former outperformed the latter in predictions regardless of the size of the sample. Mohammad Umair [12] (2016) and others combined back propagation (BP) neural networks with ARMA model to analyze and predict the Dow Jones index using an integrated model formed by multilayer perceptron and achieved more satisfactory results. In the same year, Moghaddam A.H [13] (2016) et al. used feedforward neural networks to predict the daily turnover rate of NASDAQ stocks. Abe M and Nakayama H [14] (2018) tested a variety of neural networks. They found that deep neural networks were effective in improving the accuracy of prediction compared to traditional neural networks.

Neocognitron is prototype convolutional neural network, which was first proposed by Kunihiko Fukushima [15] in 1980. His creation was inspired by the fact that two primary visual cortical cells [S] and [C] arranged in cascade order can then be used for pattern recognition. In the late 1980s, Yann LeCun [16], the father of convolutional neural network, produced his research on "backpropagation for handwritten postal code recognition". This is the first time a convolutional neural network has been employed in a commercial system. LeCun [17] (1998) again published a paper proposing one of the classical convolutional neural network models: LeNet-5. And it has gained initial application in handwriting recognition. The development of convolutional neural network models was slow for some time afterwards by reason of the huge amount of data and enormous computing power required for training. It was not until 2012 that GPU technology advanced exponentially, allowing the ability of parallel computing to grow exponentially. Krizhevsky et al [18] proposed the Alex Net model in due course. This model was a deep convolutional neural network containing 5 convolutional layers and 3 fully connected layers. It also introduced new elements such as data normalization, ReLu activation function, Dropout method. And these new elements were applied to image classification in the ImageNet. This Alex Net model had received a extensive response. It successfully pushed convolutional neural networks into the public eye. Its appearance had led to the rapid development and application of convolutional neural networks. Persio [19] (2016) et al. attempted several neural networks, MLP, LSTM, and CNN, using the previous 30 days' returns as input to predict the rise and fall of stocks. Eventually LSTM, CNN achieved better results. Sezer [20] (2018) utilized CNN, LSTM, and DMLP simultaneously for stock price prediction, and his experiments indicated that CNN significantly outperformed other models.

Ruiqi Sun [21] (2016) trained LSTM to learn past stock market data and make predictions of stock prices. In the same year, Shuyan Wang et al [22] applied the data of 200 stocks in the previous month as a sample and used the random forest algorithm to predict the stock prices in the latter month with good results. Xiao Zhang and Zengxin Wei [23] (2018) build a random forest model to predict the price trend of stocks based on the historical price, volume and other information of stocks. The backtest results were found to have some reliability. Qiao Ruoyu [24] (2019) verified the optimization of neural network models including recurrent neural network, long and short-term memory network, and convolutional neural network based on indices, and added an attention mechanism to correspond different weights to different time periods. This attempt explored direction for the optimization of neural network for prediction of financial markets. Chen, Xiangyi [25] (2018) employed convolutional neural network for

prediction analysis of CSI 300 index. And the parameters were tuned to process in order to achieve the best results. Xie Qi [26] (2019) et al. used Bagging method to generate 8 neural networks by constructing 6-layer independent LSTM networks and tested the CSI Composite Index, CSI 300 Index by doing so. The accuracy rate was 57.67Hongrui Zhao et al [27] (2021) introduced attention mechanism in a hybrid model of long and short-term memory network and convolutional neural network. In such manner, the effectiveness of the neural network model for feature analysis extraction was improved.

From the above research analysis, it is easy to see that various deep learning algorithms have been tried many times in the field of finance and have achieved certain results. However, they mainly focus on regression prediction of a single index or quantitative stock selection (classification problem). In contrast, research on quantitative timing using convolutional neural network is relatively rare. Also hyperparameters in CNN have very obvious consequences on the effectiveness of model training. This thesis will investigate each of the above issues.

1.4 Research Contents

Select appropriate transaction data as input factors. Construct an appropriate convolutional neural network learner. The data factors are put into training the learner to obtain the prediction model. The model is expected to forecast future stock prices. The detailed process is described below.

First, get long period stock trading data, such as opening price, high price, low price, closing price, volume, turnover and other indicators. After removing the outliers from these trading data, then they are treated as input factors for the convolutional neural network. Stock trading is in a disorderly state in the short term and is highly random. However, the training of the learner requires a large amount of stable and reliable high-quality data, so the influence from short-term noise has to be weakened as much as possible.

Second, the hyperparameters of the convolutional neural network, such as the number of convolutional layers, convolutional kernel size, number of iterations, learning rate, batch size, activation function, hidden layer units, etc. , have a significant impact on the speed of model operation, resource consumption, and accuracy of results. Therefore, it needs to be determined by research trials. This has a crucial role and significance to the effectiveness of the model.

Third, the model's learner is trained with a pre-processed input data set. There are two main categories of machine learning methods: supervised learning and unsupervised learning. Supervised learning finds intrinsic connections between input and output data by learning algorithmically from a portion of the available data (often called the training set). Further the corresponding function mapping between the two is established. Then the answers to some relevant unknown questions can be predicted more accurately based on this function mapping. Unsupervised learning does not contain a given output. Instead, it directly models and analyzes the input data. For instance, a clustering algorithm is a type of unsupervised learning. What is usually used for financial data analysis is supervised learning. In this study, with a large amount of data a neural network learner is trained and a mapping model is built. Then the future price of the stock can be predicted with the help of the model. It's a regression problem.

Fourth, in this experiment, a comparison test between convolutional neural network and

other deep learning algorithms will be conducted to verify the effectiveness of convolutional neural network in financial prediction.

1.5 Research Methods

Daily data for selected stocks for a total of 10 years between January 2013 and December 2022 became the original data set. 50% of them are used as training set to train the learner. The other 50% is marked as the validation set for model testing.

Based on previous experience, adjust the important hyperparameters of the learner one by one. Combine with actual measurements until the accuracy, operating efficiency, etc. are optimal.

This thesis is structured as follows:

The first chapter is the introduction. It mainly introduces the research background, status, contents and methods of this thesis.

Chapter 2 provides general knowledge related to stock trading. Focuses on the relevant data factors for stock trading and their role in technical analysis. The data factors directly related to stock prices are then selected as the data set for the model.

Chapter 3 is the theoretical foundation of machine learning algorithms. From the evolution of machine learning algorithms to the analysis of the advantages and disadvantages of current mainstream algorithms. Then convolutional neural network is introduced systematically. The basic theory of convolutional neural network, it's components and training methods are introduced as key parts.

Chapter 4 is the key to this thesis. It specifically shows the construction of convolutional neural networks. The first step is to determine the samples for the training and test sets and perform data pre-processing. The second is the choice of hyperparameters for the convolutional neural network. The last step explains the model building and the detailed process of training and testing.

Chapter 5 is a comparison of multiple deep learning algorithms. Several deep learning algorithms are trained using the same data set, and then the effectiveness of each model is compared using validation metrics.

Chapter 6 is a summary. The superiority of the current convolutional neural network for stock price regression prediction is illustrated. It also points out the problems of existing models and looks into the future research directions.

2 Introduction to Stock Trading

People have been trading stocks for a long time. Stocks and stock markets existed in the ancient Roman Republic more than 2,000 years ago. The establishment of the modern stock market system is credited to the Netherlands. On 20 March 1602, the Dutch East India Company, based in Amsterdam, issued shares for the first time, with 1,143 people subscribing, among them Germans, Belgians and Luxembourgers. In the following centuries, stock exchanges were established everywhere. As stock trading developed, the corresponding institutions and laws became increasingly sophisticated. Essentially, stocks are a product of the continuing evolution of the commodity economy and productivity. The development of stock trading has played a huge role in the world economy.

2.1 Common Data and Factors in Stock Trading

Each stock constantly generates various data as it is traded. Some of the data are generated in real time during the transaction. Such as transaction times, prices, volumes. Some of the data are processed by computer analysis to facilitate the quantification of trades and analysis of stocks. For example, technical indicators such as turnover rate, volume ratios, K-line charts, MACD, dynamic P/E, KDJ, etc.

The trading price of stocks, as an indicator that is closely related to every investor, the slightest change in it touches everyone's nerves. without a doubt it is also the main focus of our stock investment analysis. There are 4 important components of the daily share price: the opening price, the high price, the low price and the closing price, which are the basic building blocks of a K chart. As shown in figure 2.1, the red and green parts form the K-line entity, while the black lines are the upper and lower shadows of the K-line. The whole graph is a complete visual representation of the change in stock price over a certain period of time.

During the opening period, the price of a stock basically keeps changing constantly. So what causes this change? Stocks are also essentially commodity. The price of a commodity is a monetized expression of its value. The value of a stock is actually determined by the fundamentals of the company issuing the stock. It refers to aspects such as the value of the business, profitability, debt ratios, growth, etc. Another major factor that causes fluctuations in commodity prices in market economy is the balance between supply and demand. When supply exceeds demand, commodity prices fall back. When the supply is less than the demand, then the price increase is inevitable. Then specifically in the trading of stocks, the role of volume becomes crucial. Throughout stock trading, any time there is a high price fluctuation, it is accompanied by a sharp increase in volume. How do we determine the size of a stock's volume? There are two commonly used indicators. One is the turnover rate and the other is the volume ratio. The turnover rate can be calculated by multiplying the total number of transactions reached in a given period of time for a given stock by the inverse of the total

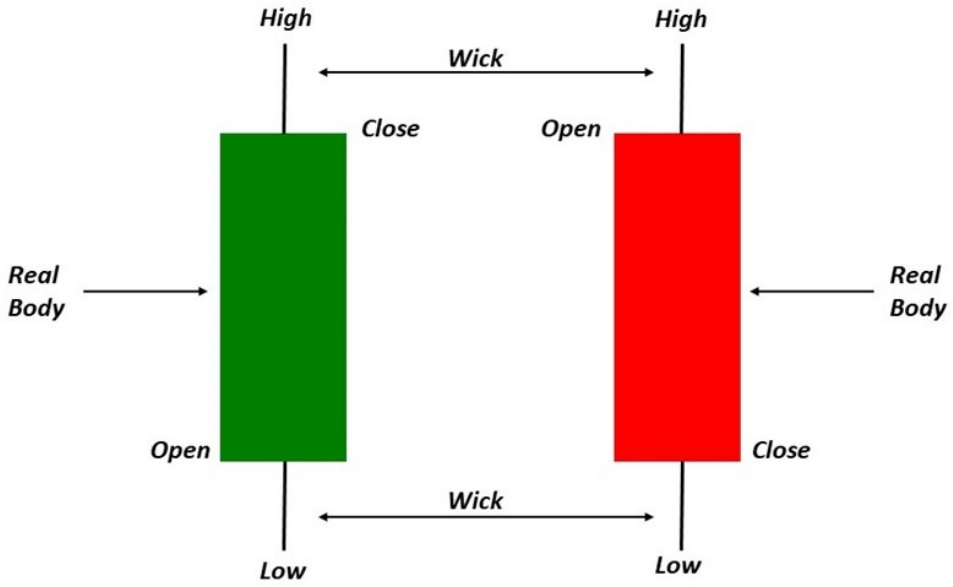

Figure 2.1: K-line Chart

number of issues for that stock. If a stock has a trading volume of 10 million shares in one day, the total number of shares in issue is 100 million. Then the stock's turnover rate for that day is 10%. The turnover rate is one of the main indicators of stock trading volume. The volume ratio refers to the ratio of the total number of transactions on the day to the average number of recent transactions, and is an indicator to measure the relative transaction volume. It makes mention of the ratio of the average trading volume per minute after the stock market opens to the average trading volume per minute in the past 5 trading days. This is a relative value. This ratio makes it possible to determine how active a stock has been trading recently. It also has a significant impact on the immediate price of the stock.

P/E and PBR are also common indicators in stock trading. P/E is the price to earnings ratio of a stock and is calculated by dividing the price of a stock by its earnings per share. PBR is a stock's price-to-book ratio. It is calculated by dividing the share price by the net value per share. These two reference indicators can reflect to some extent the reasonableness of the stock's valuation.

2.2 Input Factors Selection

There are numerous data factors involved in stock trading. Some quantitative fund teams use up to thousands of metrics, with different methods of classification. In the traditional way, they can be divided into fundamental and technical factors. In quantitative investing, momentum, sentiment, value, quality and risk factors are introduced to define different indicators. Among these thousands of data factors, many of them are very highly correlated. For example, using factors such as PE, PB, PCF and dividend yield have roughly the same effect. Since the model in this thesis only predicts the short-term trend of stock prices and is limited by the computational power of the experimental platform, only the factors that are highly correlated

with the prediction results are selected as inputs to the convolutional neural network model. This input contains the most basic volume and price information, i.e. the daily opening, closing, high and low prices, volume and turnover of the stock. An example is shown in table 2.1 below.

Date	Opening Price	Closing Price	High Price	Low Price	Volume	Value
2013/1/4	0.48	0.48	0.48	0.48	0	0
2013/1/7	0.53	0.53	0.53	0.52	224192365	118755634
2013/1/8	0.55	0.56	0.56	0.54	159586816	87654091
2013/1/9	0.56	0.55	0.56	0.54	70273828	38566631
2013/1/10	0.55	0.55	0.55	0.54	93751591	51149989
2013/1/11	0.55	0.55	0.58	0.54	67055318	37511970
2013/1/14	0.55	0.57	0.57	0.55	70601320	40043600
2013/1/15	0.57	0.55	0.57	0.55	45354228	25219877
2013/1/16	0.55	0.55	0.56	0.55	43810715	24299319
2013/1/17	0.55	0.55	0.55	0.54	20885871	11407922

Table 2.1: Input Factors (Example)

2.3 Summary of This Chapter

This chapter introduces some important factors related to stock trading. According to the purpose of this experiment and the capability of the experimental hardware platform, data factors directly related to stock prices will be selected as input data for the experiment.

3 Introduction to Machine Learning Algorithms

Machine learning is an important approach to achieve artificial intelligence, a major branch of artificial intelligence. It studies how computers can simulate human learning behavior to acquire new knowledge or skills and reorganize the existing knowledge structure to continuously improve itself. Machine learning has been widely used in areas such as data mining, computer vision, natural language processing, biometric recognition, search engines, speech and handwriting recognition, and robotics. Today, machine learning is changing our world. The following is a brief review of the development stages of machine learning and an introduction to several classical machine learning algorithms. The main part of this is the introduction of convolutional neural network. It is the core algorithm of this thesis.

3.1 The Evolution of Machine Learning Algorithms

So far, the development of AI has actually gone through three main stages. The first stage is the period of reasoning that began roughly in the late 1950s, represented by the "logical theorists" created by Herbert A. Simon and others in 1955. At that time this program gave perfect proofs of almost all 52 theorems in the work "Principia Mathematica" by mathematicians Russell and Whitehead [28]. In the second phase, starting from the mid-1970s, artificial intelligence entered the "knowledge phase". After the first phase of application, people began to realize that it was quite tedious to teach the machine what they had learned from their own experience. So attempts were started to let machines learn knowledge by themselves. This approach involved building a corresponding knowledge base of experts in each field and using it as a basis for analytical reasoning to assist in decision making. One such representative was N. J. Nilson's "Learning Machine". But there are still many problems with this approach. All information had to be recorded and stored. This information would be retrieved during the query. Such a large amount of storage space is occupied. And this stored content was not universal and scalable. If a little change occurred, they may not match. This was not the kind of artificial intelligence we want but rote learning. So from the 1980s artificial intelligence gradually entered the era of machine learning.

Barr and Feigenbaum, in their handbook of artificial intelligence, classified machine learning into "mechanical learning", "didactic learning", "analogical learning", and "inductive learning" [29]. Among them, the mechanical learning was only to save the input information as it was and to take it out when it would be queried, which was equivalent to the second stage of the development of artificial intelligence. The third stage of AI, which started in the 1980s, was "learning from examples". This was fundamentally different from the almost search-like approach of the second stage. The most typical example is the decision tree algorithm

based on logic learning. This algorithm is easy to understand and implement. It runs quickly and efficiently. It can better support the processing of continuous and missing values, but is prone to overfitting. The random forest algorithm, which emerged subsequently, uses a random sampling of sample features to build multiple decision trees to avoid this problem. However, it still does not work well in the case of more noisy data. Moreover, the random forest algorithm model is less interpretable compared to a single decision tree. By the mid-1990s neural network-based machine learning algorithms were gradually developed into the mainstream.

3.2 Types of Machine Learning Algorithms and Introduction

3.2.1 Random Forest

Tim Kam Ho of Bell Labs introduced the concept of random forest in 1995. It is figuratively named random forest because it consists of multiple decision trees. Each decision tree in a random forest is independent and unrelated. The judgments of each tree are collected and counted when a new sample is input. The judgments of the more numerous ones are used as the result output. The generation of each decision tree in the forest is detailed in the following method [30]. (a) Assume a total of N input samples as the training set, whose number of features is denoted by M. (b) A certain number of samples are randomly selected from N samples to make a training set for each decision tree. (c) m randomly selected nodes from M features as decision trees need to satisfy m«M; (d) Each family of decision trees will not be pruned.

Random forest has very high accuracy. Good adaptability to a large number of features of the data. It can handle large amount of data efficiently. Also the accurate estimation of missing data and the self-balancing error method make it have a place in machine learning algorithms even after the widespread popularity of many neural networks.

3.2.2 Support Vector Machine

Support vector machines are hyperplanes that are separated by features in a certain sample space, and this hyperplane is guaranteed to maximize the feature interval. In short a support vector machine is actually a two-class classifier [31] as shown in figure 3.1.

Support vector machines can support linear as well as nonlinear classification very well. If the input sample data is linearly divisible, then we use the maximum classification interval as the basic principle to find the two parameters w, b in the classification function.

$$f(x) = w * x + b \tag{3.1}$$

Assuming that the original sample data is linearly indistinguishable, in order to find a way to differentiate the sample features need to be mapped to a higher dimensional space. After that, we search the higher dimensional space for the plane that can distinguish the features. The consequent problem is that as the dimensionality increases, the exponentially growing

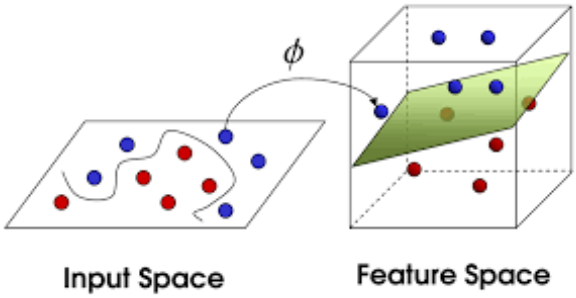

Figure 3.1: SVM Diagram

computational workload overwhelms the computer. The kernel function was then introduced to solve this problem. The inner product of vectors in the new space can be calculated using kernel functions. This effectively reduces the amount of operations caused by the dimensional explosion. Support vector machines are now widely used to play an important role in text classification, graphics classes, handwriting recognition, gene sequence analysis, etc.

3.2.3 Recurrent Neural Network

Compared with the independence of previous neural network, recurrent neural network ist better at processing sequential information with certain correlation before and after [32]. Early neural networks existed in isolation. When we input a variable x, the hidden layer would calculate the corresponding output y based on x and the mapping relationship. The recurrent neural network also uses the values of the previous layer as the weights of the current input for the association calculation, as shown in figure 3.2.

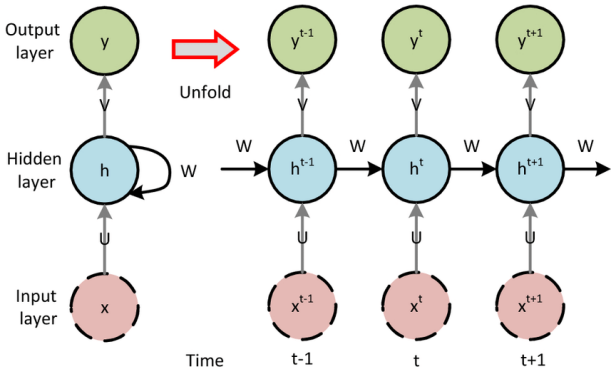

Figure 3.2: Recurrent Neural Networks Diagram

The lowermost layer x^t represents the input at time t, and the layer above x^t is the hidden layer. The predicted output at time t is denoted by o, and the actual value of the sample is y^t. From the above figure, we can see that it is not only h^t and x^t that determine o^t, but also the W passed from the upper layer. And the loss function L^t is calculated from y^t and o^t.

The biggest weakness of recurrent neural networks in applications is the gradient explosion and vanishing gradient. The worst result of gradient explosion is that the model fails to

converge. Gradient disappearance tends to occur when the network level is very deep. In particular, when Sigmoid is used as the activation function, the derivative is close to zero when the input is large or small. At this point the gradient keeps decreasing exponentially, infinitely close to zero, which causes the model to become slow to converge. The weight parameter cannot be updated. There are reliable methods to deal with gradient explosion. For example, gradient clipping is a good solution for gradient explosion. But how to solve the gradient disappearance is the key research direction of almost all deep learning methods including RNN.

3.2.4 Long Short-Term Memory

Long Short-Term Memory network is an extension of recurrent neural network. It improves and extends for the weaknesses of recurrent neural networks. S Hochreiter and J Schmidhuber designed the long and short-term memory network in 1997. The problem of gradient explosion and gradient disappearance caused by BPTT (back-propagation through time) in RNN is achieved by replacing the original hidden units of RNN with CEC (constant error carrousel) units. As shown in figure 3.3:

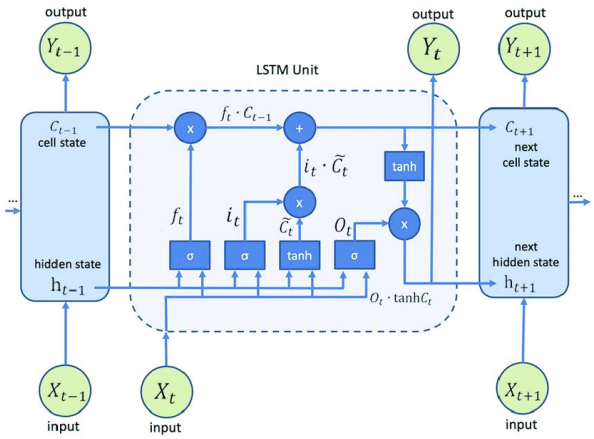

Figure 3.3: Long Short-Term Memory Network Cell Structure

From the above figure, we can see that the LSTM cell structure introduces a gate structure (Gate) based on RNN, and its complexity is much higher than the latter. The gating structure contains the forget gate f_t, the input gate i_t, and the output gate o_t. f_t controls the information to be discarded in c_{t-1} of the previous state to solve the problem of memory capacity saturation.

LSTM can effectively alleviate the gradient vanishing problem in RNN. In addition, although RNN networks can theoretically establish dependencies for time intervals of arbitrary length, it is a challenge for recurrent neural networks if the output x_t at moment t depends on the input x_{t-m} before the interval m moments, when the value of m is relatively large. This is known as the long time dependence problem in RNN. In contrast, the hidden state h in LSTM contains historical information and can be treated as memory. The memory unit C in the hidden state captures and retains a significant portion of the key information, and thus can effectively overcome the problem of long-time dependence [33].

3.3 Overview of Convolutional Neural Network and Components

Convolutional neural network is the late start type of many deep learning algorithms. It uses multiple layers of perceptrons to detect analyze and acquire features in the data. Back in the 1960s biologists Thuibal and Wiesel studied the visual cortex of the cat in depth. It was found that some of the cells in it formed a complex structure. The presence of this structure makes it very sensitive to visual input space. They named it the receptive field. The multilayer perceptron is the development of the receptive field. The essence of a convolutional neural network is a multilayer perceptron with reduced number of weights and model complexity [34]. The advantage is that it is possible to use images directly as input to the neural network. This eliminates the tedious process of image feature extraction and data reconstruction in traditional algorithms. CNN can also automatically extract various features such as color, texture, topology, etc. from images. The advantages of CNN over other algorithms for 2D image processing are self-evident. In the early days, it was not popular due to the limitation of computing power. Nowadays, thanks to the rapid development of GPU chips, the powerful parallel processing of GPU chips can provide good support for the development of CNN. Therefore it has grown by leaps and bounds in recent years. It has been well used in various image processing, image recognition, and target tracking algorithms.

3.3.1 Structure of Convolutional Neural Network

The main components of a convolutional neural network are Input Layer, Convolutional Layer, ReLU Layer, Pooling Layer, and Full Connection Layer, as shown in figure 3.4.

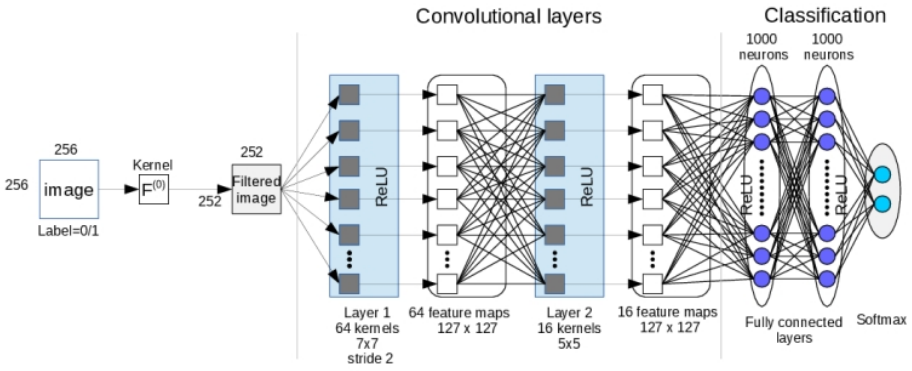

Figure 3.4: Convolutional Neural Network Structure Diagram

In the above diagram we see that the input layer is mainly used to accept data entered by the user. Other layers are highlighted below.

3.3.2 Convolutional Layer

2D Convolution

As the name implies the convolutional layer is the most basic and core part of the convolutional neural network. The principle is to extract the features of the data by doing convolutional operations with the input data and filters (also called convolutional kernels, filters and weight matrices).

Table 3.1 shows a 6*6*1 grayscale image as the input to the convolutional neural network.

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

Table 3.1: Convolutional Neural Network Input Data

Table 3.2 shows a 3*3 filter.

1	0	-1
1	0	-1
1	0	-1

Table 3.2: Convolutional Neural Network Filter

The filter is dot product with the input image, and then all the products are added together to obtain the values in the upper left corner of table 3.3. And so on by translating the filter one step to the right, we will again get nine dot products, adding up the nine numbers to get the second value in the first row of table 3.3. After the first row is calculated, pan down one cell and calculate the second row, and finally get the 4*4 matrix in table 3.3 as the result.

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	16

Table 3.3: Convolution Results

Suppose we use an n*n image as input and a filter of size f*f for one convolution operation. Then we will get a (n-f+1)*(n-f+1) matrix. Such an operation has two problems [35].

(a) The image size will become smaller after each convolution operation. This can be a significant loss for deep neural networks that require multiple convolutions.

(b) Too little information is retained about the edges of the input image. Pixels at the edges are lost with each convolution operation. But the pixels in the middle will always be there,

being convolved many times. This actually loses most of the information of the image edges. And the middle part is repeatedly used easily to cause overfitting.

To solve the above problem, we introduce Padding method for convolutional neural networks. Suppose we fill one pixel along the edge of the 6*6 image (usually filled with 0), so the 6*6 image becomes an 8*8 image. After this image is convolved with a 3*3 filter, a new 6*6 image will be obtained, with the same size as the original input image. This mitigates the loss of image edge information from the convolution operation.

Because of the above operations, the convolutional computation is divided into two categories. One is called Valid convolution, which does not incorporate the Padding operation. So the image size is reduced proportionally after each convolution calculation. The other is called Same convolution. Because the Padding method mentioned above is used. So the size of the input and output images can be kept the same. Equation 3-8 gives the pixels that need to be filled in order to keep the image size consistent.

$$p = (f-1)/2 \tag{3.2}$$

where p is the pixel to be filled and f is the edge length of the input image. When f is an odd number, normal padding is possible. When f is even, only asymmetric padding can be used.

As we discussed earlier, after each convolution operation, the filter is panned horizontally and vertically by one pixel before the next calculation. This one-pixel shift is called the convolutional step, which is usually denoted by the letter s. As one of the hyperparameters of the convolutional neural network, the step value can be set artificially. After introducing the concepts of padding and step size, we can conclude that the output image size of the convolutional neural network is: Equation 3-9

$$\lfloor (n+2p-f)/s+1 \rfloor * \lfloor (n+2p-f)/s+1 \rfloor \tag{3.3}$$

where n is the input image edge length, p is the padding pixel, f is the filter edge length, and s is the convolution step size.

3D Convolution

The previous discussion of 2D convolutional networks used a 6*6*1 grayscale image. When we need to use an RGB color image, then the input becomes a 6*6*3 three-dimensional matrix. Where 3 represents the three primary RGB color channels. Then assuming that the 3-channel filter is still used, the new filter is a 3*3*3 matrix.

As shown in figure 3.5, since the input and filtering become three levels, the dot product yields as many as 27 numbers, which are added together to obtain one value for the 4*4*1 output on the right. And only one filter is used here. When we need to detect more than one feature of the target (suppose n), we can use n filters. The final output will be a 4*4*n 3D cube. Usually n is referred to as the depth of the cube, also called as the number of channels.

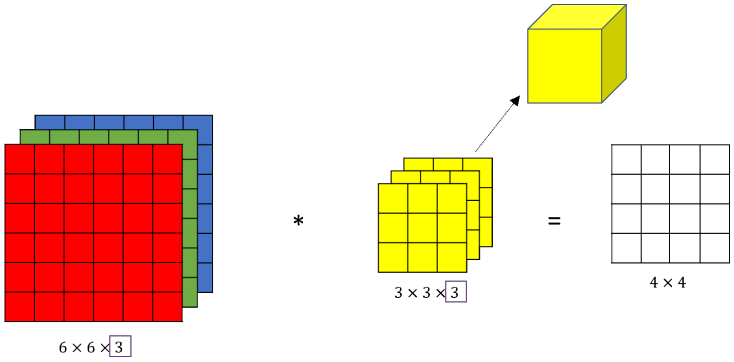

Figure 3.5: 3D Convolution Example

3.3.3 Pooling Layer

The pooling layer is an important part of a convolutional neural network. It is usually located between two convolutional layers. Its role is to filter features and redundant information and compress the number of features. Pooling layer not only reduces the computation and hardware resource consumption, speeds up the operation, but also prevents overfitting [35].

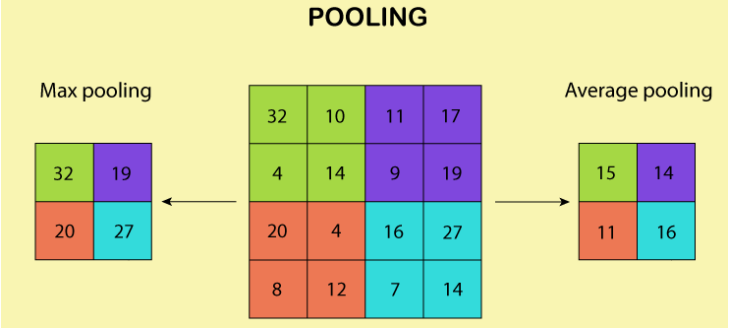

Figure 3.6: Pooling Method Diagram

As shown in the figure 3.6, a 4*4 input matrix is shown on the left. The execution pooling is a 2*2 matrix which has a step size of 2. When maximum pooling is performed, the maximum value from the area covered by the 2*2 matrix is selected as the output to form the result matrix on the left. If averaging pooling is performed, the result on the right is obtained by averaging the 4 numbers in the coverage of the 2*2 matrix. Usually maximum pooling is used more often. Its essence is to preserve the maximum value of some local feature. In addition, pooling does not reduce the number of channels relative to convolutional computation.

3.3.4 Fully Connected Layer

The fully connected layer is essentially a classifier. It takes the feature values extracted from the convolution and pooling layers and computes a weighted sum of them to output the probability of each feature. This has the advantage of reducing the impact of feature location distribution on classification. It is called a fully connected layer because each neuron in this layer is connected to all nodes in the upper layer, as shown in the figure 3.7 below.

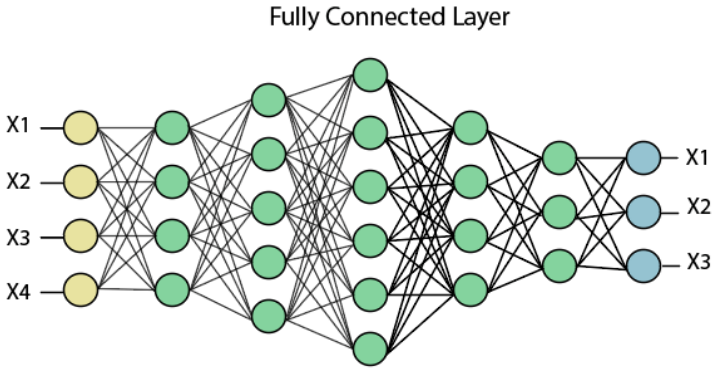

Figure 3.7: Fully Connected Layer Diagram

3.3.5 Cost Function

The cost function is an important concept that is involved in all neural networks. Similar to it is the loss function. The loss function reflects the loss value of a single sample, i.e., it calculates the difference between the predicted value and the true value of a single sample. The cost function, on the other hand, is oriented to the entire training set and calculates the average of the errors of the entire training set. The key value gradient in the gradient descent algorithm can be obtained by taking the partial derivative of each parameter in the function. Therefore, for a qualified cost function, two basic requirements must be satisfied. One is that the effect of the model can be evaluated correctly. The second is that it can be differentiated for the parameters. After a lot of practice, the most widely used cost function for regression problems is the mean squared error (Mean squared error), whose mathematical expression is as follows:

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^i - y^{(i)})^2 = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \tag{3.4}$$

where m is the total number of training samples. θ is the parameter and $h_\theta(x)$ is the predicted value through h_θ. From the above equation, we can see that the mean square error squares the difference between the predicted value and the true value, increasing the penalty of the error and making it more sensitive to the error. And our goal is to find out when the function $J(\theta)$ obtains the minimum value parameter of θ, which is the optimal solution of the model.

3.3.6 Training Process

The network is first initialized using pre-defined weights. In forward propagation, the input data is convolved and pooled to obtain the feature vectors. Then it is imported to the fully connected layer and processed to obtain the classification results. When the result matches the expectation, the result can be output directly. Otherwise, it enters the backpropagation process. In the backpropagation process, the error between the predicted value and the actual value is first calculated using the cost function. Then the error is returned to the previous layer. The difference value of each layer is thus iteratively calculated. The weights of each layer are dynamically updated according to the difference value. After that, it re-enters the forward propagation as shown in the figure 3.8. This cycle is repeated until the matching weights are

found, so that the error between the output value and the actual value is within the appropriate range.

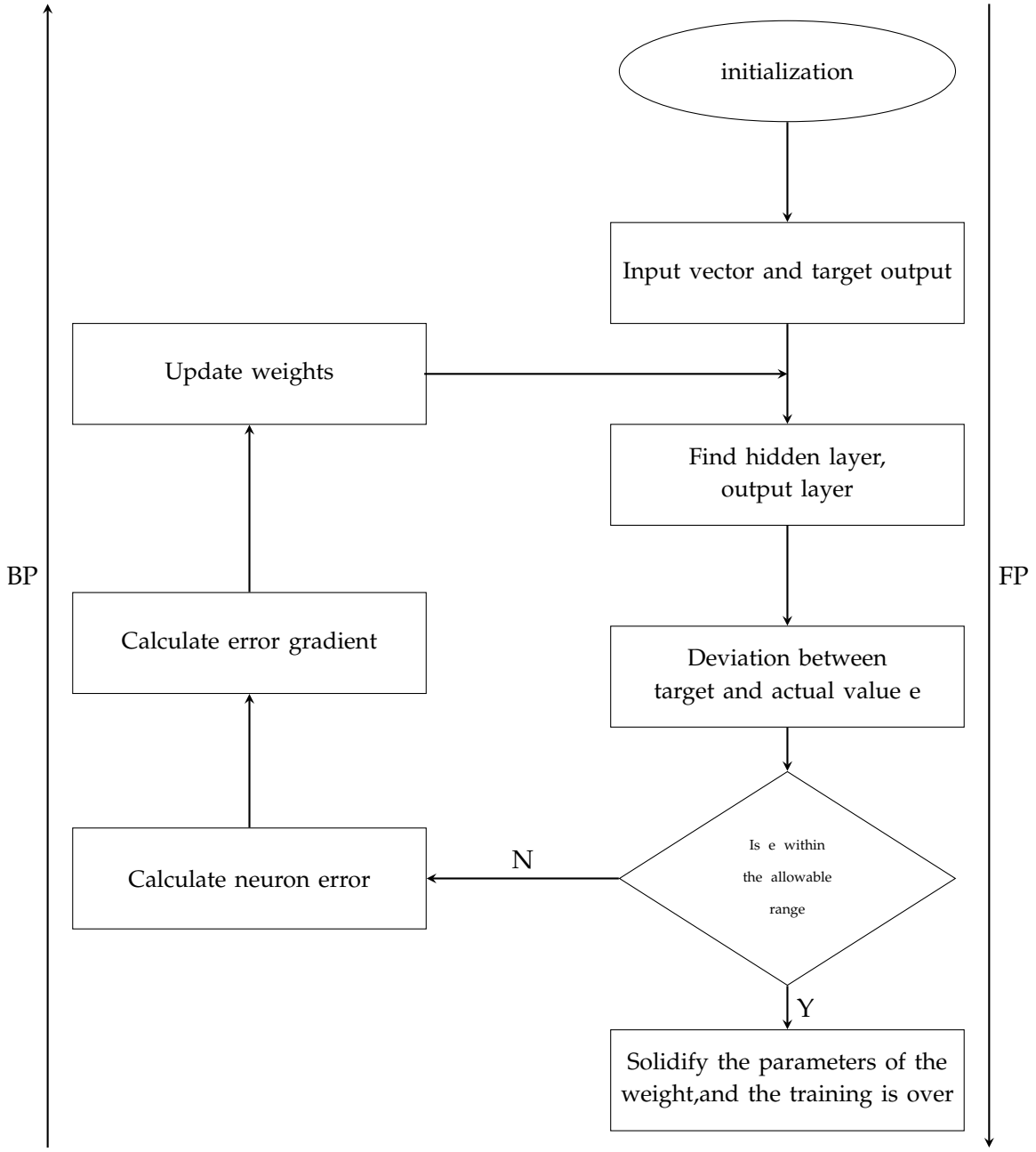

Figure 3.8: Convolutional Neural Network Training Process

3.3.7 Classical Convolutional Neural Networks

LeNet-5 convolutional neural network structure proposed by LeCun et al [17] in 1998. It was mainly applied to automatic handwritten digit recognition and achieved good results. The

neural network has a total of 7 layers. The first layer is a convolutional layer that uses six 5*5 filters with a step size of 1 to get an output of 28*28*6. Then use the average pooling with width 2 and step size 2 to get an image of 14*14*6. It is exactly half the size of the original image. Then continue the convolution using 16 filters of 5*5 to get an image of 10*10*16. Another pooling to get a 5*5*16 image. The next two fully connected layers have a total of 400 nodes in them, where each node contains 120 neurons. Finally a classifier is used to get the results. As shown in figure 3.9.

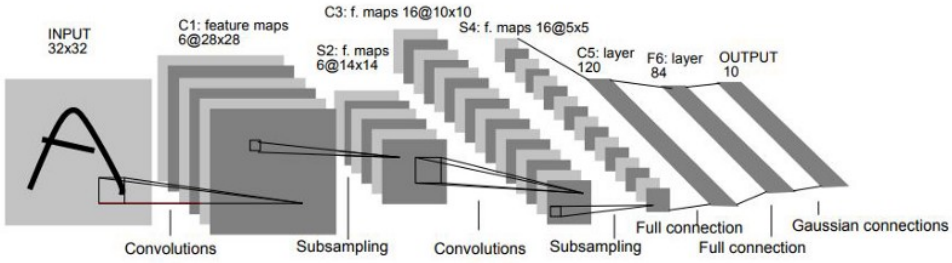

Figure 3.9: LeNet-5 Convolutional Neural Network

This approach of alternating convolution and pooling layers, then outputting features to the fully connected layer and finally obtaining the result is still widely used today.

Hinton and his student Alex Krizhevsky [18] proposed AlexNet in 2012. This network uses an image of 227*227*3 as input. Firstly, 96 11*11 filters with a step size of 4 are applied to the input for Valid convolution operation to obtain a 55*55*96 image, which is only one quarter of the original size. Next, a max pooling with a step size of 2 is performed using a 3*3 matrix, and the output is 27*27*96. Then this result is filled and Same convolution is performed to get 27*27*256. The max pooling is performed again and the image becomes 13*13*256. Continue to do three more Same convolutions and one max pooling, and the final result is 6*6*256. This result is expanded into 9216 units, weighted by the fully connected layer, and finally classified by the softmax function, and the whole process is carried out as shown in figure 3.10.

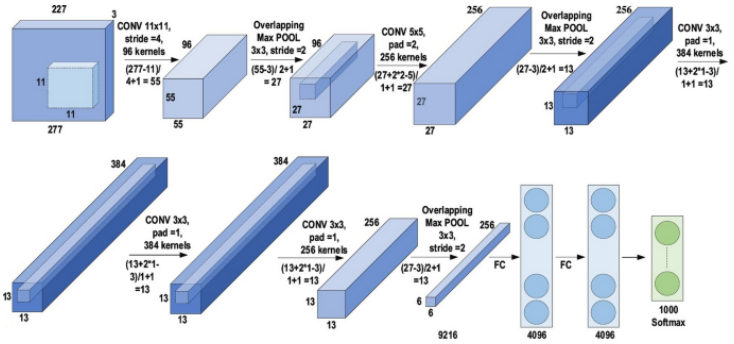

Figure 3.10: AlexNet Detailed Structure Diagram

Compared with LeNet, the whole structure of AlexNet has many similarities, but the internal structure is much more complex. It contains about 60 million parameters. In contrast, the former consists of only about 60,000 parameters. The immediate benefit is that AlexNet's ability to identify similar modules has been greatly improved. In addition, thanks to the change of the

activation function to ReLU, the processing of nonlinear relations has been improved.

3.4 Summary of This Chapter

This chapter introduces the development of its learning algorithms and the principles of common machine learning algorithms at this stage. Random forest is a classical algorithm. Compared with decision tree, it has qualitatively improved its accuracy. And because of its simple structure, it is easy to implement. Random forest has strong resistance to fitting and can handle most of the problems in linear and nonlinear, so it is used until now. SVM is more resistant to attacks because of its unique inter-sample distance. It is mostly used in binary classification problems. The popular neural network in recent years was actually proposed a long time ago, but its accuracy depends on the training of a large amount of data. Limited by the amount of data available at that time and the arithmetic power of the hardware platform, it was not widely promoted. It has the advantage of being able to exist many intermediate layers. This plays an important role in the analysis of correlations between input features. Currently it is mostly used in AI translation, reading comprehension, speech recognition, etc. The long and short-term memory network replaces the original hidden units with CEC units based on recurrent neural networks and adds a gating structure. This gives the long short-term memory network the ability to overcome the long time dependence problem. It also alleviates the gradient explosion and gradient disappearance. The overall structure of convolutional neural networks are more similar to biological neural network. They can directly perform feature extraction on the input image data, thus avoiding the tedious data reconstruction. Convolutional neural networks are now widely used in image analysis, speech image recognition, target tracking, etc.

4 Construction and Training of Convolutional Neural Network Model

This experiment is carried out on a development platform with an Intel CPU [1] and Ubuntu operating system [2]. The Pycham IDE combined with an extensive library of third party Python tools [3] allows us to efficiently process multi-dimensional matrices, construct and train multi-layer neural networks, analyse large amounts of financial data and plot statistical graphs.

4.1 Data Pre-processing

After obtaining the relevant raw data, first we have to filter and correct the missing data and abnormal data.

Abnormal values, also called outliers. Usually they are the values in the dataset that deviate from the large and unreasonable ones. For example, the height of a person is 3 meters, or the weight of a mouse is 20 kg. These values clearly do not make sense. From the concept of normal distribution, we know that the probability that the data are distributed in the interval (μ - 3 σ,μ + 3 σ) (where μ is the mean and σ is the standard deviation) is 0.9974, and the probability that the data are outside of 3σ is only 0.0026. Therefore, we can consider it as anomalous data. To avoid outliers interfering with model training, we remove them directly from the data set. There may be some missing items in our data set due to some lost items in the original data itself or some data lost during the transfer process. This is handled in the same way as outliers, and we remove them before the model is trained. Since this part of the data is a very small percentage, it will not affect the training of data set.

There is a difference in the fundamental unit of the raw data. For instance, data describing a person's physical appearance includes height, weight, etc. They are data of different scale. Length units include meters, decimeters, centimeters, etc. To avoid these differences from affecting the validity of data analysis, it is necessary to standardize the data. Data normalization is the process of converting input into pure data without units. Then it is scaled to a certain proportion. Eventually, all data fall into a specific range. This method is often used in the comparison and evaluation of financial data indicators. Two common normalization techniques may be useful:

(a)Linear Normalization or Min-Max Normalization

[1] CPU is Intel Core i7-4710MQ 2.5GHz 8-core processor, 16GB RAM, GPU is Intel® HD Graphics 4600 with 1GB Graphics Memory.

[2] Ubuntu 20.04.2 LTS 64-bit

[3] Numpy scientific computing base library,Scikit-learn machine learning library,Tensorflow machine learning framework,Keras neural network library,Pandas,Matplotlib,etc.

The input raw data $x_1, x_2, x_3, ..., x_n$ are linearly transformed according to the following rules:

$$y_i = \frac{x_i - \min\limits_{1 \le j \le n}(x_i)}{\max\limits_{1 \le j \le n}(x_i) - \min\limits_{1 \le j \le n}(x_i)} \tag{4.1}$$

The transformed output data are all mapped to the [0,1] interval. Linear normalization is suitable when the features are more or less uniformly distributed within a fixed range.

(b).Z-score normalization

The input raw data $x_1, x_2, x_3, ..., x_n$ are transformed according to the following rules:

$$\mu = \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4.2}$$

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{4.3}$$

$$y_i = \frac{x_i - \bar{\mu}}{\sigma} \tag{4.4}$$

The output new series $y_1, y_2, y_3, ..., y_n$ without fundamental unit, with mean 0 and variance 1. Z-score normalization is suitable when the feature distribution does not contain extreme outliers.

The best normalization techniques are the ones that work empirically. Therefore, we will try some new normalization techniques that work well on data feature distribution in this experiment.

4.2 Targets Selection

The test targets for this thesis were selected from the CSI 300 index, which currently has more than 4,000 stocks in it. In order to verify that the model can be applied to different types of stocks, some typical stocks of representative industries will be selected as target stocks for testing. The selection will be based on the following principles:

(a) The stock is representative of the industry to which it belongs. The main business of the company to which the stock belongs must be highly relevant to the industry. Select a maximum of one stock in an industry.

(b) Stocks that have been listed for more than 10 years are selected. In this way, enough raw data can be obtained to train the neural network model.

(c) Stocks with more volatile stock prices are preferred if conditions a,b are met. Active stocks usually offer more opportunities to investors.

(d) Exclude unconventional stocks of ST and other types.

To verify the generality of the model, in addition to the index stocks, the CSI 300 index is also used as the target. The details are shown in table 4.1.

Number	Stock Code	Stock Name	Industry	Reason for Selection
1	000300	CSI300	Index	Representative Index
2	600109	Sinolink Securities	Financial Industry	Securities Trading Company, longer established and active
3	600598	Beidahuang Group	Agriculture	Typical agricultural stock
4	600660	Fuyao Group	Manufacturing	Manufacturing representative, largest manufacturer of automotive safety glass
5	600808	Masteel Group	Mining Industry	Representative of cyclical stocks, one of the major steel companies
6	600327	Grand Orient	Retail Industry	Retail Industry Representative
7	600171	Shanghai Belling	High Tech Industry	One of the earliest listed chip companies
8	600618	Shanghai Chlor-Alkali Chemical	Chemical Industry	The largest chlor-alkali chemical company in China
9	300003	Lepu Medical	Healthcare Industry	One of the first stocks to be listed after the establishment of GEM
10	000651	Gree Electric	Household Appliances Industry	One of the major stocks in the home appliance industry

Table 4.1: Test Targets

The 10 stocks listed in the table above are in different industries. They are in line with the principles proposed above and are highly representative.

4.3 Data Construction

In this thesis, a total of 10 years of daily stock data between January 1, 2013 and December 31, 2022 are used as the raw dataset. 80% of the raw data set is used to build the training set and 20% of the data is used as the test set. Because of the limitation of the total amount of data, a special validation set is not set up but a test set is used instead.

Using 30 days as a window period, the volume and price information of the stock for the previous 30 days is used to predict the price of the stock 5 days later. As shown in figure 4.1, the window is rolled in order of 1 day. The number of training and testing data is thus the total number of samples minus the window period.

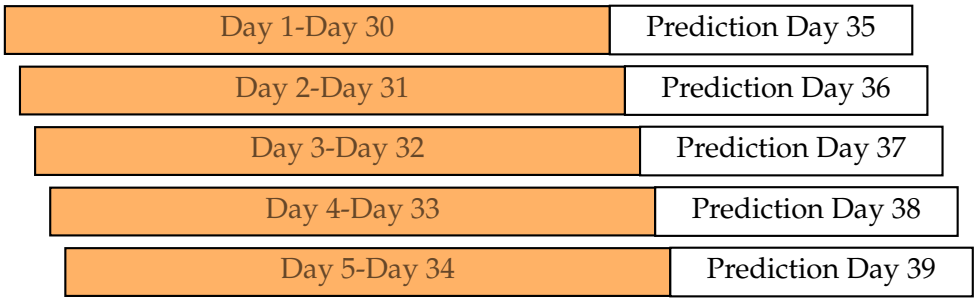

Figure 4.1: Rolling Construction Dataset

4.4 Model Construction

The convolutional neural network model in this thesis has a convolutional layer as the core, supplemented by some pooling layers, a flatten layer, a long short-term memory network, a Dropout layer, and a fully connected layer. The network structure is shown in table 4.2.

As seen in the table above, the entire network uses 3 convolutional layers with all filters of size 2*2. Each convolutional layer is immediately followed by a pooling layer for max pooling. This structure is inspired from the LeNet-5 classical network. The maximum value in each pooling matrix is chosen to represent a feature as a way to avoid possible overfitting during the training process. It also reduces the dimensionality of the feature data and reduces the complexity of the computation. It is of interest that as the network deepens, the length and width of each layer gradually decreases after each maximum pooling, but the number of filters

Layer Name	Filter Size	Filter Number	Activation Function/Pooling	Padding	Step Size	Probability	Nodes Number/Vector Length
Convolutional	2*2	32	ReLU	Same	2*2	-	-
Pooling	2*2	-	Max	Valid	2*2	-	-
Convolutional	2*2	32	ReLU	Same	2*2	-	-
Pooling	2*2	-	Max	Valid	2*2	-	-
Convolutional	2*2	32	ReLU	Same	2*2	-	-
Pooling	2*2	-	Max	Valid	2*2	-	-
Flatten	-	-	-	-	-	-	-
LSTM	-	-	tanh	-	-	-	128
Dropout	-	-	-	-	-	0.2	-
Fully Connected	-	-	-	-	-	-	1

Table 4.2: Convolutional Neural Network Initial Parameters

is incremental. This is because the number of filters determines the number of features to be extracted from the input data. And this is highly correlated with the accuracy of the prediction. Therefore, in modern multilayer convolutional network designs, the number of filters is usually increased layer by layer.

The flatten layer is located between the convolutional layer and the fully connected layer. The data is convolved and turned from two-dimensional to three-dimensional output, while the fully-connected layer can only accept two-dimensional data. The flatten layer is to flatten (i.e. Height*Width*Channel) the multi-dimensional data (Height,Width,Channel) generated by the convolutional layer into data that can be processed by the fully connected layer.

A layer of long short-term memory network is added to the convolutional neural network structure. The output of the convolutional neural network is used as the input of the long short-term memory network for training again. Convolutional neural network can effectively extract the features in the data, but there is almost no analysis of the dependency relationship between data in time series. The long short-term memory network can make up for this deficiency and form a good match.

The long short-term memory network is followed by the Dropout layer. The role of the Dropout layer is to temporarily discard the units in the neural network with a set probability. Its purpose is to prevent the model weights, etc. generated during the training process from being overly consistent with the training data. This avoids the phenomenon of having a fairly high accuracy on the training set, but a far cry when tested on the test set. Since the Dropout discarded data is random, the remaining neurons vary for different batches. It is equivalent to training diverse mini-grids, and thus can effectively prevent overfitting.

Finally, the prediction results are output using fully connected layer. figure 4.2 shows the structure of the convolutional neural network.

The initial values of the other hyperparameters in the model are shown in table 4.3.

Learning Rate	BatchSize	Epoch	Train Ratio	Activation Function	Optimizer	Cost Function
0.1	64	10	80%	ReLU	Adam	Mean_Squared_Error

Table 4.3: Model Hyperparameters Initial Values

Where epoch is the process where the neural network is trained once completely using the training set and returned. However, one pass of the sample data in the neural network is not enough, and several training sessions are required to achieve a good fit. Too many epochs can cause overfitting of the data. Therefore an appropriate number of epochs needs to be chosen.

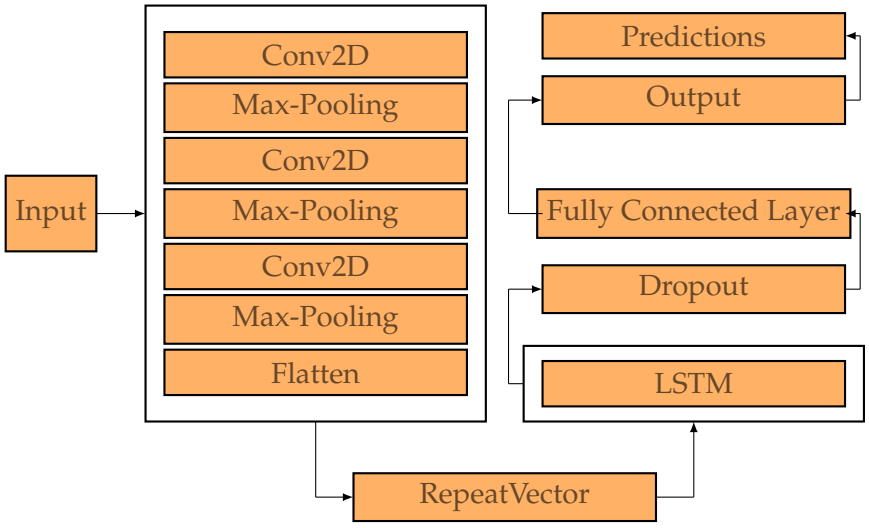

Figure 4.2: Convolutional Neural Network Structure

Due to the limitation of the number of stock data, 80% of the data are used for training, and the test set is used instead of the validation set.

4.5 Evaluation Metrics

After the model has been trained, we need to evaluate the model effect. Since the data of the same stock for the previous 30 days is used here to predict the closing price after 5 days, it is a regression problem. The mean squared error, mean absolute error, and coefficient of determination R^2 are proposed to be used to evaluate this regression algorithm. These three indicators are analyzed and explained below.

4.5.1 Mean Squared Error

$$MSE = \frac{1}{m}\sum_{k=1}^{n}(y_i - \hat{y_i})^2 \tag{4.5}$$

where y_i is the true value and $\hat{y_i}$ is the predicted value. This formula is actually the loss function of linear regression. It is a direct indication of the difference between the true value and the predicted value. The smaller the value is, the smaller the error is. However, in this formula, it squares the difference in order to ensure that the resulting value is positive and derivable. This causes problems with the base units. When the mean squared error is greater than 1, it amplifies the error, and when it is less than 1, it reduces the error.

4.5.2 Mean Absolute Error

$$MAE = \frac{1}{m}\sum_{k=1}^{n}\left|y_i - \hat{y_i}\right| \tag{4.6}$$

As can be seen from the above equation, mean absolute error makes improvements on the

basis of mean squared error. Instead of using the square operation, the error result is positive when the absolute value is used. So it reflects the real error value without amplification, and like MSE its value is as small as possible.

4.5.3 Coefficient of Determination R^2

The two metrics above suffer from the same problem, with no upper and lower bounds on the results. Thus Coefficient of Determination R^2 is introduced.

$$R^2 = 1 - \frac{\sum_i (\hat{y_i} - y_i)^2}{\sum_i (\bar{y} - y_i)^2} \tag{4.7}$$

As can be seen from this formula, the numerator is sum of the squares of the differences between predicted and true values, similar to the MSE. The denominator is sum of squares of the difference between mean and true value, similar to the variance. In this way the results are effectively unified between [0,1]. If the result is 0, it means that the model fit is almost non-existent. A result of 1 indicates a very good fit of the model.

4.5.4 Classification Accuracy Score

It is very difficult to make accurate regression prediction of stock prices. Therefore there is one more important parameter in stock investment analysis. It is the prediction of the trend of whether the closing price of a stock will go up or down. This is a classification problem. Since this model is a regression prediction of the closing price of the stock, a simple processing of the prediction results is needed to derive the accuracy of the classification.

The previous section mentions that it is using the previous 30 days of stock data to predict the closing price 5 days later. Then the closing price of the day minus the closing price 5 days before, if the result is positive, the stock is up in the 5-day period, marked as 1. If the result is negative, the stock price is down, marked as 0. Do the same with the predicted stock price. The value of classification accuracy is then calculated using the classification accuracy function of the sklearn machine learning library. From this, the accuracy of the model in predicting the up and down trend of the stock can be obtained.

4.6 Hyperparameter Tuning

In order to avoid the interference of too much noise in the parameter debugging process which affects the generalization ability of the model, we want the sample data to be relatively stable. In the experimental targets, stock 600598 is suitable as a sample for tuning the hyperparameters of the model. After satisfactory results are achieved, the optimal hyperparameter values are then applied to other stocks for testing.

4.6.1 Learning Rate and Epoch

The Learning Rate is one of the most important hyperparameter to tune for Neural network to achieve better performance. When training deep neural networks, it is often useful to reduce learning rate as the training progresses. This can be done by using pre-defined learning rate schedules or adaptive learning rate methods. The learning rate determines the step size for gradient adjustment of the network weights at each iteration of the backpropagation process to reach the optimal solution of the loss function. If the learning rate is too small, it will not only make the gradient descent process very slow and affect the training efficiency, but also may cause the network to fall into local minimization. If it is too large, then the weight parameters are updated faster and may oscillate repeatedly around the optimal solution and fail to converge, and may miss the optimal solution. Therefore, an appropriate learning rate has an important impact on the final results of the network.

Epoch is the process by which a training set is trained once completely through a neural network and returned. However, one pass of the sample data in the neural network is not enough and multiple passes are needed to fit the training data. But too many epochs can cause overfitting of the data. Therefore, an appropriate number of epochs needs to be chosen.

The two hyperparameters are put together for tuning because they have inverse correlation. When the learning rate is set larger, the gradient descent is fast and the epoch needed to reach the optimum is relatively small. When the learning rate is small, the gradient descent process becomes slower. In order to obtain the optimal solution, it is necessary to increase the number of epochs.

Constant Learning Rate

Constant learning rate is a method to manually stabilize the learning rate of a model at a certain value based on past experience. Table 4.4 below shows the test results for some constant learning rate values.

Number	Learning Rate	epoch	Test Set Cost Function Loss	Validation Set Cost Function Loss	Mean Absolute Error	Mean Squared Error	Determination Coefficient	Classification Accuracy Score
1	0.1	20	0.0244	0.0101	0.2350	1.0132	-0.4	0.51
2	0.05	25	0.0261	0.0169	0.1211	1.6963	-1.35	0.51
3	0.005	30	0.0018	0.0083	0.1610	0.8183	0.89	0.83
4	0.002	50	0.0009	0.0028	0.0613	0.1677	0.89	0.88
5	0.001	60	0.0021	0.0059	0.0856	0.8842	0.88	0.84
6	0.0005	80	0.0027	0.0092	0.0689	0.7986	0.88	0.82
7	0.0003	100	0.0030	0.0102	0.0765	1.0011	0.86	0.83
8	0.0001	150	0.0058	0.0099	0.0769	0.9555	0.87	0.83
9	0.00005	200	0.0182	0.0110	0.0648	0.8816	0.89	0.79

Table 4.4: Learning Rate and Epoch Constant Parameter Tuning

It can be clearly seen from the table that the 1st and 2nd experiments have a fast gradient descent because of the large learning rate parameter. Therefore the global optimal solution is obviously missed. The results are not satisfactory for either the regression or classification problems. By the 4th time, the regression test results and classification test results were optimal when the learning rate is 0.002 and the epoch number is 50. Starting from the 5th time, while the loss of the cost function for the test set continues to decrease, the loss value of the cost function for the validation set starts to pick up. This means that the neural network begins to overfit to the training set at this point. In several later experiments, the validation set cost function

climbs in oscillation. And the three evaluated parameters regarding the regression problem remain oscillatory. It indicates that the regression prediction is not sensitive to overfitting. However, the indicator about the accuracy of the classification problem continues to decrease, indicating that overfitting has an impact on the classification problem.

Dynamic Learning Rate

The dynamic learning rate is conceived based on the need for different gradient descent at different stages of training. At the beginning of training, the position is still far from the bottom of the loss function surface at this point. Increasing the learning rate at this point helps to move quickly towards the bottom of the surface, thus increasing the training efficiency. When the position is close to the bottom of the surface, decreasing the learning rate can avoid missing the lowest point of the surface (i.e., the optimal solution) by moving too fast.

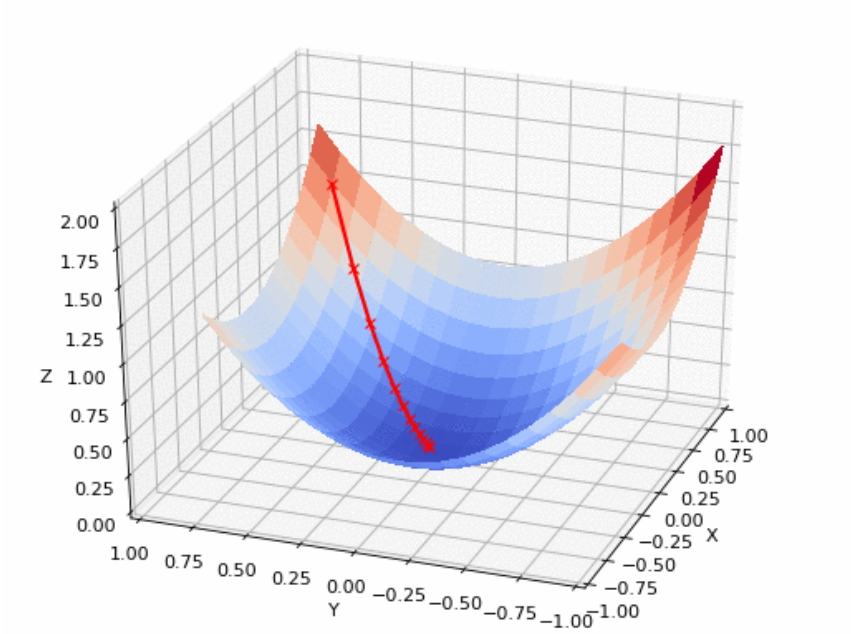

Figure 4.3: Convolutional Neural Network Gradient Descent

So we try to do dynamic adjustment of the learning rate according to the number of training epochs. We will take the initial value of the learning rate as the base, and then reduce the learning rate to the original value * adjustment ratio every adjustment frequency times. The epoch is set to 60.

Number	Initial Learning Rate	Adjustment frequency	Adjustment Ratio	Test Set Cost Function Loss	Validation Set Cost Function Loss	Mean Absolute Error	Mean Squared Error	Determination Coefficient	Classification Accuracy Score
1	0.1	10	0.6	0.0042	0.0093	0.0946	0.0022	0.69	0.51
2	0.05	10	0.6	0.0039	0.0055	0.0692	0.0022	0.87	0.77
3	0.006	10	0.7	0.0022	0.0002	0.0123	0.0022	0.91	0.86
4	0.002	15	0.8	0.0033	0.0048	0.0293	0.0022	0.90	0.84
5	0.001	15	0.8	0.0132	0.0136	0.0824	0.0022	0.89	0.84
6	0.0005	20	0.9	0.0201	0.0167	0.0821	0.0022	0.89	0.82

Table 4.5: Learning Rate and Epoch dynamic Parameter Tuning

As can be seen from the table 4.5 above, at larger learning rates, the dynamic learning rate is significantly better than the constant learning rate because of the constant contraction.

The initial learning rate at the 3rd and 4th experiments is close to the neighborhood of the optimal point in the constant learning rate test, and achieve the best results. In the 5th and 6th experiments, the learning rate is so small that the test falls into local minimization.

From the above experiments, we have verified the conclusion that learning rate is negatively correlated with epoch. And it has been found that an appropriate combination of learning rate and epoch has a decisive effect on the training results. Based on the above test results, we decided to use the 3rd set of values in the dynamic learning rate experiment as the parameters of learning rate and epoch for this model.

4.6.2 Batch Size

Batch Size is the number of samples to be trained at a time by the convolutional neural network. It is also one of the important hyperparameters of convolutional neural networks. Because the number of training samples is usually large and the computer memory is limited to hold all the data at the same time, Batch Size is needed to limit the amount of data to be trained at one time. However, the value of Batch Size should also not be too small. A too small sample size is prone to large variance due to noise effects. so that the model falls into a local optimal solution. Table 4.6 shows the test results for different Batch Size.

Number	Batch Size	Test Set Cost Function Loss	Validation Set Cost Function Loss	Mean Absolute Error	Mean Squared Error	Determination Coefficient	Classification Accuracy Score
1	32	0.0031	0.0043	0.0592	0.0030	0.89	0.84
2	64	0.0033	0.0037	0.0588	0.0049	0.89	0.86
3	67	0.0029	0.0038	0.0601	0.0038	0.93	0.92
4	70	0.0019	0.0059	0.0662	0.0047	0.91	0.91
5	128	0.0011	0.0055	0.0699	0.0056	0.89	0.86
6	160	0.0038	0.0070	0.0557	0.0060	0.90	0.88
7	200	0.0032	0.0061	0.0203	0.0039	0.90	0.81
8	260	0.0015	0.0022	0.0781	0.0030	0.88	0.88

Table 4.6: Batch Size Parameter Tuning

The above table reflects that as the Batch Size gradually increases, the loss of the validation set cost function tends to gradually decrease, the mean square error, the average absolute error decreases, and the coefficient of determination and accuracy increase. The increase in Batch Size not only reduces the number of iterations required for an epoch, but also speeds up the processing of data. Also the increase in the number of samples per treatment makes the direction of gradient descent more accurate. However, when the Batch Size is large enough, continuing to increase its value will bring the opposite effect. This is because the data variation increases between each Batch, causing the respective generated gradient values to cancel each other out and make it hard to correct. This causes the model to be difficult to converge. The 8th set of experimental data demonstrates the above issues. In general, Batch Size plays a role in stock prediction, but the effect is smaller than that of learning rate and epoch. In this test, we choose the 3th set of data, i.e. 67, as the Batch Size of the model.

4.6.3 Activiation Function

The activation function, as one of the hyperparameters of the convolutional neural network, plays a crucial role in the whole network. Among the given sample data, many of them are

nonlinear. In contrast, the computation of an ordinary neural network is linear. After each layer of neurons is computed, an activation function is superimposed, which allows the neural network model to fit the data better. In this way, the linear function is made nonlinear. The common activation functions are Sigmoid, Tanh, ReLU (Rectified Linear Unit), Leaky ReLU, etc. As shown in figure 4.4:

where Sigmoid is defined as follows:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{4.8}$$

It is one of the most commonly used activation functions for early convolutional networks. It can map the output to the interval of (0,1). However, the gradient of the sigmoid function is problematic in some cases. During the backpropagation process of deep learning, the gradient of the sigmoid function is used to update the weights and biases of the neural network. For very low and very high input values, the gradient vanishes to zero. This prevents the neural network weights and biases from being updated, and thus the model cannot continue learning.

The Tanh hyperbolic tangent function is defined as follows:

$$tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{4.9}$$

Tanh is also similar to sigmoid but better. The tanh function takes values in the range (-1 to 1), which effectively alleviates the problem of gradient disappearance. At the same time, the convergence speed is accelerated because the value interval is "zero-centered". The definition of ReLU is:

$$ReLU(x) = max(0, x) \tag{4.10}$$

It has a constant derivative 1. Thus the problem of gradient disappearance in deep networks is completely solved. In ReLU, the gradient descent is equal to the product of the weights, so there is also the possibility of gradient explosion. Also ReLU is particularly unfriendly to negative numbers. It will force negative inputs to zero. This removes a noise, but causes the model to lose useful features because these weights are not updated. Usually this phenomenon is called the "Dead Neuron". To solve this problem, the Leaky ReLU is introduced, which is defined as follows:

$$y_i = \begin{cases} x_i & if \quad x_i \geq 0 \\ \frac{x_i}{a_i} & x_i < 0 \quad a_i \in (1, +\infty) \end{cases} \tag{4.11}$$

As can be seen from the equation, Leaky ReLU no longer forces a negative number to zero. Instead, it is given a very small number as a way to solve the "Dead Neuron" problem. The following are images of several commonly used activation functions.

Here are the tests of these activation functions on the model. Since the model performs 3 convolution operations, it is necessary to determine the types of the 3 activation functions.

The above table 4.7 reflects that TanH, ReLU performs better in this model. In contrast, Leaky

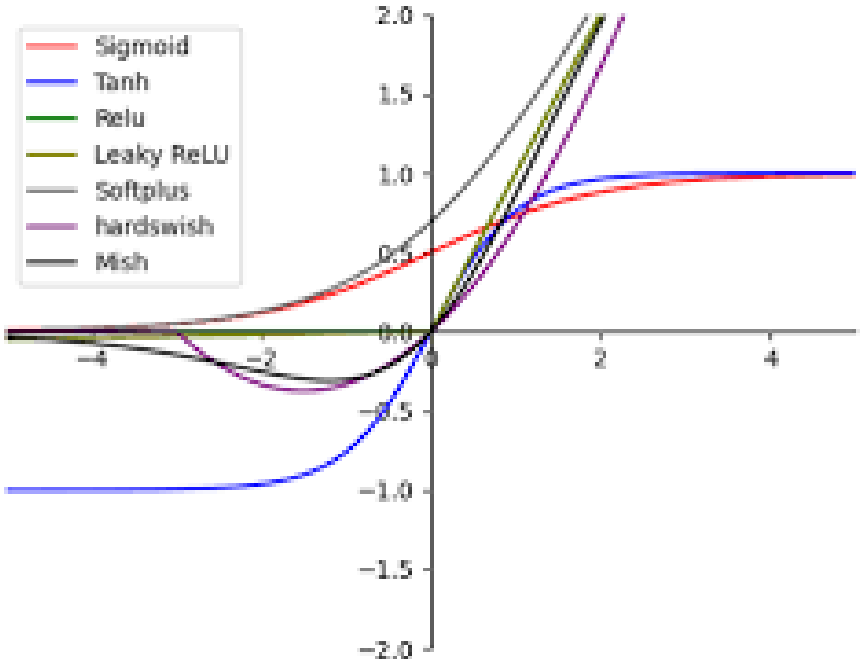

Figure 4.4: Comparison of Common Activation Functions

Number	1.Activation Function	2.Activation Function	3.Activation Function	Test Set Cost Function Loss	Validation Set Cost Function Loss	Mean Absolute Error	Mean Squared Error	Determination Coefficient	Classification Accuracy Score
1	Sigmoid	Sigmoid	Sigmoid	0.0081	0.0093	0.0621	0.0087	0.89	0.87
2	TanH	TanH	TanH	0.0081	0.0088	0.0580	0.0069	0.90	0.88
3	ReLU	ReLU	ReLU	0.0051	0.0072	0.0401	0.0060	0.92	0.90
4	Leaky ReLU	Leaky ReLU	Leaky ReLU	0.0093	0.0091	0.0733	0.0074	0.90	0.84
5	TanH	ReLU	Leaky ReLU	0.0077	0.0072	0.0593	0.0074	0.90	0.86
6	TanH	TanH	TanH	0.0075	0.0068	0.0634	0.0087	0.89	0.86
7	TanH	ReLU	ReLU	0.0098	0.0088	0.0724	0.0089	0.88	0.83

Table 4.7: Activation Function Tuning

ReLU, a modified version of ReLU, does not test as well. This is affected by the data distribution in this thesis, where data close to 0 or less than 0 are almost non-existent. We continued to try to use different activation functions in 3 different convolutional layers and the results are not satisfactory. Therefore, ReLU is finally chosen as the activation function for this model.

4.6.4 Convolution Kernel Size and Step Size(Stride)

The convolution kernel is also known as a filter. It is actually a weight matrix that is used in the convolution operation to extract the eigenvalues of the data. The size of the convolution kernel is usually small. The more common sizes are 1*1, 2*2, 3*3, 4*4, and 5*5. However, 1*1 convolution does not enhance the receptive field, so it is not considered. We test the other commonly used sizes one by one.

The distance that the convolution kernel moves in this way is called the convolution step. The size of the step affects the accuracy of feature extraction. For example, a convolutional kernel of size 3 with a step size of 1, then there will be overlap in the receptive field of adjacent

steps. If the step size is 3, the overlap will disappear. With a step size of 4, a gap of size 1 will appear in the middle. In short, the smaller the step length, the higher the accuracy of extracting features, and vice versa, the lower. And the step size is also related to the size of the convolution kernel.

Number	Convolutional Kernel Size	Convolutional Step Size	Test Set Cost Function Loss	Validation Set Cost Function Loss	Mean Absolute Error	Mean Squared Error	Determination Coefficient	Classification Accuracy Score
1	2*2	2	0.0011	0.0021	0.029	0.0021	0.90	0.89
2	3*3	2	0.0034	0.0031	0.098	0.0076	0.89	0.84
3	4*4	2	0.0048	0.0060	0.079	0.0101	0.88	0.81
4	5*5	2	0.0039	0.0058	0.062	0.0083	0.88	0.87

Table 4.8: Convolutional Kernel Size and Step Size Tuning

The test results in above table 4.8 show that the convolution kernel size and step size have a small effect on this model. We choose the 1st set of data as the parameters of the model.

4.6.5 Number of Filters

The number of filters is also commonly referred to as the number of channels. The number of channels is determined by the number of filters in this layer after each layer of convolution is completed, except for the initial value. It extracts the feature values of the data layer by layer in the model. And the number of channels can be regarded as the width of the model. It can make the number of features extracted from each layer of the model richer. But when the number of channels of the model reaches a certain size, the improvement of the model effect is very limited by increasing its number. At the same time, the increase in the number of channels also leads to an exponential increase in computation. Therefore, the setting of channels needs to be carefully considered.

Number	1.Channel Number	2.Channel Number	3.Channel Number	Test Set Cost Function Loss	Validation Set Cost Function Loss	Mean Absolute Error	Mean Squared Error	Determination Coefficient	Classification Accuracy Score
1	32	32	32	0.0062	0.0053	0.0658	0.0062	0.88	0.84
2	64	64	64	0.0030	0.0034	0.0492	0.0030	0.92	0.89
3	128	128	128	0.0035	0.0041	0.0622	0.0079	0.89	0.84
4	256	256	256	0.0052	0.0055	0.0654	0.0071	0.90	0.88
5	32	64	128	0.0099	0.0121	0.0158	0.0072	0.90	0.84
6	64	128	256	0.0200	0.0181	0.0269	0.0093	0.87	0.83
7	32	64	256	0.0039	0.0010	0.0359	0.0002	0.86	0.82
8	64	64	128	0.0033	0.0039	0.0632	0.0082	0.89	0.84

Table 4.9: Number of Filters Tuning

The first 4 items in table 4.9 use the same number of channels for the 3 convolutional layers of the neural network, with the number of channels increasing in sequence. It can be seen that as the number of channels increases, the prediction results and evaluation values move in a better direction. Due to hardware constraints, the maximum number of channels tested in this thesis is 256. In the last 4 items of the test, different number of channels are used in different convolutional layers to optimize the performance, but the expected results are not achieved. Balancing efficiency and effectiveness, the 2nd set of data is chosen as the final parameter for the number of channels.

4.6.6 Max Pooling

The pooling layer of this experimental model uses max pooling. This is because max pooling extracts the maximum value in the area s a feature and discards other values. This discards as much redundant information as possible, reduces computation, and prevents overfitting The parameters of max pooling are mainly related to the pooling size and step size and padding mode. In the previous test, we defaulted to a pooling with size of 2, step size of 1, and valid padding. In the following, we test to find the optimal parameters.

Number	Pooling Size	Step Size	Padding	Test Set Cost Function Loss	Validation Set Cost Function Loss	Mean Absolute Error	Mean Squared Error	Determination Coefficient	Classification Accuracy Score
1	2*2	1*1	valid	0.0030	0.0031	0.0591	0.0021	0.92	0.90
2	2*2	2*2	valid	Error	Error	Error	Error	Error	Error
3	2*2	2*2	valid+same	0.0058	0.0099	0.0630	0.0083	0.88	0.86
4	2*2	2*2	same	0.0097	0.0264	0.0888	0.0054	0.89	0.85
5	4*4	2*2	same	0.0157	0.0375	0.0283	0.0081	0.88	0.82

Table 4.10: Pooling Parameter Tuning

From test results table 4.10 can be seen, when the pooling size is 2*2, the step size is 2*2 and continues to increase, the program runs with an error when using valid padding. After debugging, the pooling process eliminated a large amount of data, resulting in too little data for the later training. After changing the filling method from valid to same, the program can run normally. From the above table we can find that items 1, 3 have low data pooling size and smaller step size, thus discarding less data and thus higher correct rate. The losses in the validation set are all significantly higher than those in the training set. Looking at the data in items 4 and 5, the increase in pooling size and step size discards more redundant data, bringing the obvious effect that the loss of the validation set is smaller than that of the test set. The significant effect of pooling size on model overfitting is well demonstrated. By comparison, the data of item 1 is used as the max pooling parameters of the model.

4.6.7 Dropout Probability

The Dropout probability in Tensorflow machine learning library is the probability that a neuron will be retained. That is, the smaller the probability set, the more neurons will stop working, the better the prevention of overfitting, but the more features will be discarded. We test the effect of this probability on the prediction results in ascending order.

Number	Dropout Probability	Test Set Cost Function Loss	Validation Set Cost Function Loss	Mean Absolute Error	Mean Squared Error	Determination Coefficient	Classification Accuracy Score
1	0.2	0.0016	0.0025	0.0841	0.0029	0.90	0.85
2	0.3	0.0008	0.0018	0.0556	0.0035	0.88	0.87
3	0.5	0.0010	0.0012	0.0532	0.0025	0.91	0.87
4	0.6	0.0010	0.0011	0.0682	0.0028	0.89	0.87
5	0.7	0.0014	0.0012	0.0971	0.0036	0.90	0.84

Table 4.11: Dropout Tuning

As can be seen in table 4.11, as the probability becomes larger, the number of retained neurons increases and the prediction accuracy tends to rise. This is caused by making the probability of retained neurons larger and the discarded data smaller. However, as the probability becomes larger, the risk of overfitting also gradually increases.

Dropout is implemented by dropping some neurons randomly according to probability. Because it is random, the neurons discarded are not the same each time, thus leading to some possible variability in the results of each run of the model with the same parameter settings and input data. Based on the above test, the 3rd data is used as the Dropout probability of the model.

4.6.8 Optimizer

The function of optimizer is to find more suitable parameters to reduce the loss of the cost function in the continuous iteration of the model. It has been shown to play an important role in the training process of neural networks. A proper optimizer can make the neural network converge quickly and with high accuracy. The common optimizers can be divided into two categories, one is the adaptive learning rate algorithm. Such as Adaptive gradient algorithm (Adagrad), Adadelta, RootMean Square Prop (RMSProp), Adaptive Moment Estimation (Adam), etc. Another one is gradient descent. The typical ones are batch gradient descent (BGD), stochastic gradient descent (SGD) and mini-batch gradient descent (MBGD). In the following equation, θ is the initial parameter, η is the learning rate, δ is the small constant, β, β_1, β_2 is the decay rate, r represents the gradient accumulation, J is the objective function to be optimized, $\in$ is the step size, and $\odot$ denotes the element-by-element multiplication of two vectors.

BGD is calculated using the full data of the training set for gradient descent.

$$\theta = \theta - \eta \nabla_\theta J(\theta) \tag{4.12}$$

It is extremely inefficient and can be very slow when encountering large amounts of data.

SGD is improved for the shortcomings of BGD. Each update is performed for a single sample. For the number of training samples of hundreds of thousands or millions, the optimal solution of θ may already be found with only some of the samples updated.

$$\theta = \theta - \eta \nabla_\theta J(\theta;\ x^{(i)}; y^{(i)}) \tag{4.13}$$

However, because SGD is updated for a single sample, it is much more affected by noise than BGD. It is possible that the direction of each iteration is not the global optimal direction and may even fall into a local optimum.

And MBGD is computed using n small batch samples. It also overcomes the disadvantages of slow speed of BGD and the possibility of SGD to fall into local minimization.

$$\theta = \theta - \eta \nabla_\theta J(\theta;\ x^{(i:i+n)}; y^{(i:i+n)}) \tag{4.14}$$

MBGD is very strict about the learning rate. When the learning rate is too small, the model will converge slowly and inefficiently. If the learning rate is too large, it will again cause the model to oscillate repeatedly near the lowest point and eventually it will be difficult to converge.

Adagrad keeps track of the sum of gradient squared and uses that to adapt the gradient in

different directions.

$$\theta = \theta + \mathrm{d}\theta \tag{4.15}$$

where:

$$\mathrm{d}\theta = -\frac{n}{\delta + \sqrt{r}} \odot g \tag{4.16}$$

$$r = r + g \odot g \tag{4.17}$$

$$g = \frac{1}{m} \times \frac{\mathrm{d}(\sum_{i=1}^{m} L(f(x^i;\theta), y^i))}{\mathrm{d}\theta} \tag{4.18}$$

Its advantage is that it can make different updates to the high-frequency and low-frequency parameters separately, reducing the artificial learning rate adjustment. However, the learning rate becomes smaller and smaller as the denominator keeps accumulating. This will affect the speed of model convergence when the data volume is large.

Adadelta improves Adagrad by replacing the denominator with the decaying average of the gradient squares for the sum of squares of all gradients. This solves the problem that the gradient in Adagrad tends to zero.

RMSProp can also solve the problem of decreasing Adagrad learning rate. It has the following equation:

Update parameters:

$$\theta = \theta + \mathrm{d}\theta \tag{4.19}$$

Calculate $\mathrm{d}\theta$:

$$\mathrm{d}\theta = -\frac{n}{\delta + \sqrt{r}} \odot g \tag{4.20}$$

Cumulative squared gradients:

$$r = \beta r + (1 - \beta) g \odot g \tag{4.21}$$

Calculate gradient:

$$g = \frac{1}{m} \times \frac{\mathrm{d}(\sum_{i=1}^{m} L(f(x^i;\theta), y^i))}{\mathrm{d}\theta} \tag{4.22}$$

As we know from the above equations, it differs from Adagrad only in the cumulative squared gradient. It adds a decay coefficient β to Adagrad to control the number of previous gradients, thus preventing the learning rate from converging to zero.

Adam is a widely used optimization algorithm today. It calculates the adaptive learning rate

for each parameter. Its principle is close to a combination of RMSProp and Momentum.

Calculate gradient:

$$g = \frac{1}{m} \times \frac{\mathrm{d}(\sum_{i=1}^{m} L(f(x^i;\theta), y^i))}{\mathrm{d}\theta} \tag{4.23}$$

$$t = t + 1 \tag{4.24}$$

Update first-order moment estimates:

$$s = \beta_1 s + (1 - \beta_1) g \tag{4.25}$$

Update second-order moment estimates:

$$r = \beta_2 r + (1 - \beta_2) g \odot g \tag{4.26}$$

Correct first-order moment deviations:

$$\hat{s} = \frac{s}{(1 - \beta_1^t)} \tag{4.27}$$

Correct second-order moment deviations:

$$\hat{r} = \frac{r}{(1 - \beta_2^t)} \tag{4.28}$$

Calculate $\mathrm{d}\theta$:

$$\mathrm{d}\theta = - \in \frac{\hat{s}}{\delta + \sqrt{\hat{r}}} \odot g \tag{4.29}$$

Update parameters:

$$\theta = \theta + \mathrm{d}\theta \tag{4.30}$$

The above equations illustrate that Adam combines RMSProp and Momentum, correcting for their biases. Adam is more adapted to sparse data. In the following, we apply several common optimizers to the model for test.

Number	Optimizer	Test Set Cost Function Loss	Validation Set Cost Function Loss	Mean Absolute Error	Mean Squared Error	Determination Coefficient	Classification Accuracy Score
1	Adagrad	0.0023	0.0010	0.0226	0.58	-0.41	0.41
2	Adadelta	0.0048	0.0052	0.0464	0.69	-6.30	0.51
3	RMSprop	0.0008	0.0014	0.0071	1.45	0.81	0.79
4	Adam	0.0001	0.0008	0.0288	0.0082	0.89	0.87
5	Nadam	0.0017	0.0007	0.0213	0.0077	0.89	0.82
6	SGD	0.0036	0.0102	0.0256	0.824	-0.42	0.38

Table 4.12: Optimizer

The differentiation of the test data shows that the optimizer plays a rather important role in the prediction effectiveness of the model. As shown in table 4.12: RMSProp and its derivative types Adam, Nadam are well adapted to this model. SGD is a typical algorithm of gradient descent, which is sensitive to noise. The stock data is highly volatile. Therefore SGD test result in this model is not satisfactory. Finally Adam is chosen as the optimizer of the model.

4.6.9 Short Summary

After a series of analysis, testing, and comparison, we have a deeper understanding of the hyperparameters of the convolutional neural network. The optimal hyperparameters of this model are also screened out. As shown in table 4.13.

Initial Learning Rate	Adjustment Frequency	Adjustment Rate	epoch	Batch Size	Activation Function	Convolution Kernel Size	Convolution Kernel Step Size	Filter Number	Max Pooling Size	Max Pooling Step Size	Dropout Probability	Optimizer
0.006	10	0.7	70	67	ReLU	2*2	2*2	64	2*2	1*1	0.5	Adam

Table 4.13: Convolutional Neural Network Hyperparameter Setting

4.7 Model Actual Test

Based on the previous part of this chapter, we perform actual test on the constructed model to verify the model effect. Since 600109 has been used as a sample in the hyperparameter tuning, the next test is only for the remaining 9 stocks. The data set is constructed according to 80% training set, 20% validation set, and 30-day window period. The size of the data training matrix is 2000*30*6 and the size of the validation matrix is 400*30*6. The following are the training and validation results.

(a) 600598 Beidahuang Group

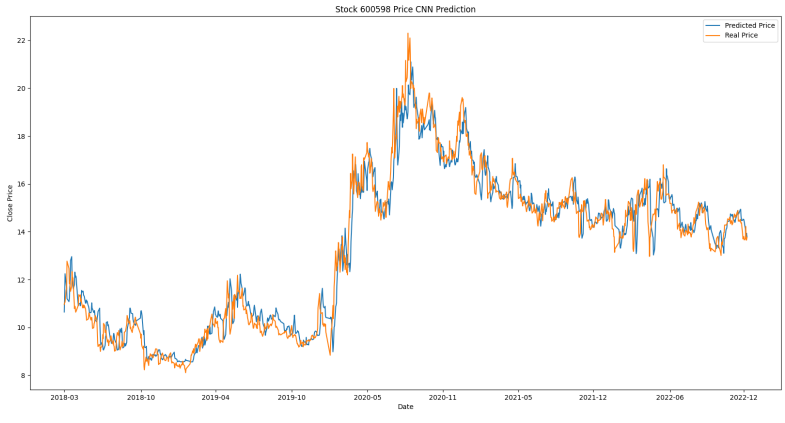

Figure 4.5: Stock 600598 Price Prediction Result

The test data show that the model has satisfactory prediction results in both regression prediction and classification prediction. As shown in figure 4.5 and table 4.14.

Stock Code	Mean Absolute Error	Mean Squared Error	Coefficient of Determination	Classification Accuracy Score
600598	0.0336	0.0023	0.95	0.94

Table 4.14: 600598 Prediction Results

(b) 600660 Fuyao Group

Figure 4.6: Stock 600660 Price Prediction Result

Stock Code	Mean Absolute Error	Mean Squared Error	Coefficient of Determination	Classification Accuracy Score
600660	0.0295	0.0019	0.96	0.91

Table 4.15: 600660 Prediction Results

During the training process using this stock data, the learner had sufficient convergence. Therefore, there is not much difference between the prediction result and the real price of the stock. As shown in figure 4.6 and table 4.15.

(c) 600808 Masteel Group

Stock Code	Mean Absolute Error	Mean Squared Error	Coefficient of Determination	Classification Accuracy Score
600808	0.0313	0.0022	0.91	0.93

Table 4.16: 600808 Prediction Results

This neural network model has a good result for the price prediction of this stock. The fit between the predicted result curve and the true price curve is good. This is a typical cyclical stock. Its price trend is positively correlated with macroeconomic changes. That is, the trend is highly similar to that of the CSI 300 index. The price of this stock fluctuates in a relatively small range. Therefore, the convolutional neural network has good prediction results for it. As shown in figure 4.7 and table 4.16.

(d) 600327 Grand Orient

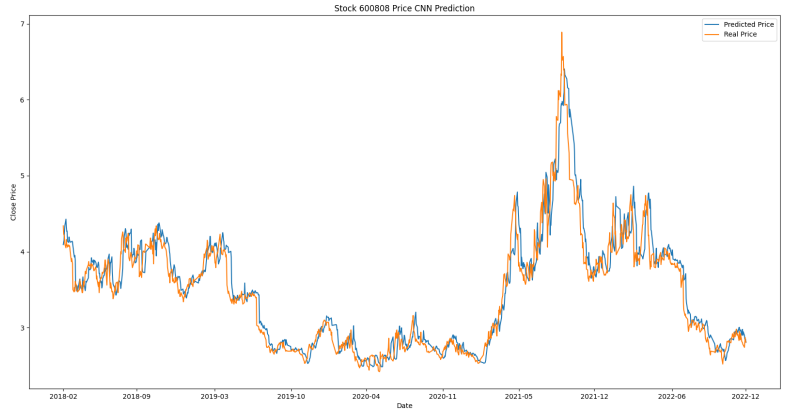

Figure 4.7: Stock 600808 Price Prediction Result

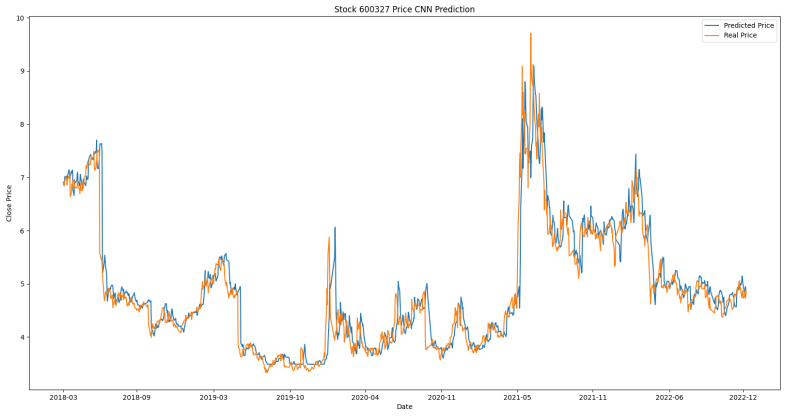

Figure 4.8: Stock 600327 Price Prediction Result

Stock Code	Mean Absolute Error	Mean Squared Error	Coefficient of Determination	Classification Accuracy Score
600327	0.0356	0.0037	0.88	0.93

Table 4.17: 600327 Prediction Results

As a representative of the retail industry, Grand Orient's data is also predicted by the model with good results. As shown in figure 4.8 and table 4.17.

(e) 600171 Shanghai Belling

Stock Code	Mean Absolute Error	Mean Squared Error	Coefficient of Determination	Classification Accuracy Score
600171	0.0306	0.0020	0.94	0.93

Table 4.18: 600171 Prediction Results

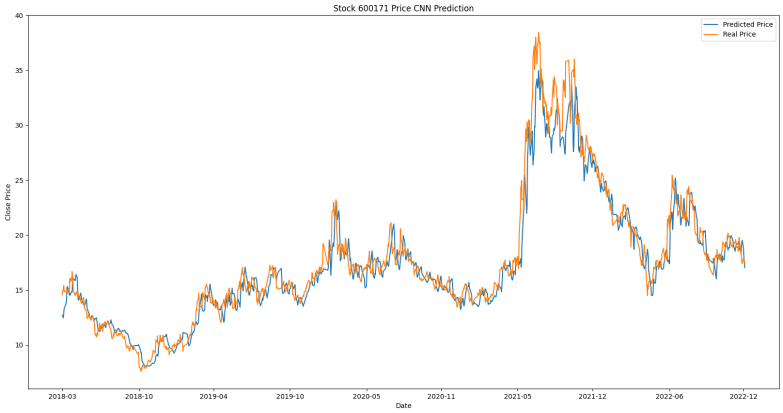

Figure 4.9: Stock 600171 Price Prediction Result

600171 is a typical growth stock. It has a large frequency and magnitude of price changes. From the model training results, the prediction is relatively not very satisfactory. As shown in figure 4.9 and table 4.18.

(f) 600618 Shanghai Chlor-Alkali Chemical

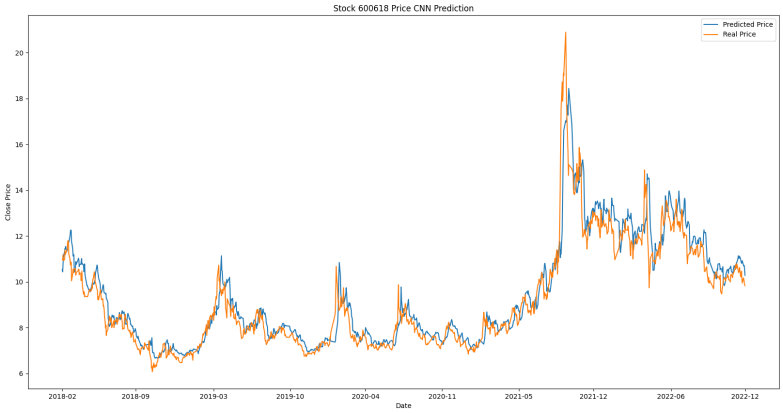

Figure 4.10: Stock 600618 Price Prediction Result

Stock Code	Mean Absolute Error	Mean Squared Error	Coefficient of Determination	Classification Accuracy Score
600618	0.0288	0.0024	0.89	0.93

Table 4.19: 600618 Prediction Results

The situation here is similar to that of the last stock. When the stock price fluctuates drastically, the convolutional neural network model does not have good predictions. As shown in figure 4.10 and table 4.19.

(g) 300003 Lepu Medical

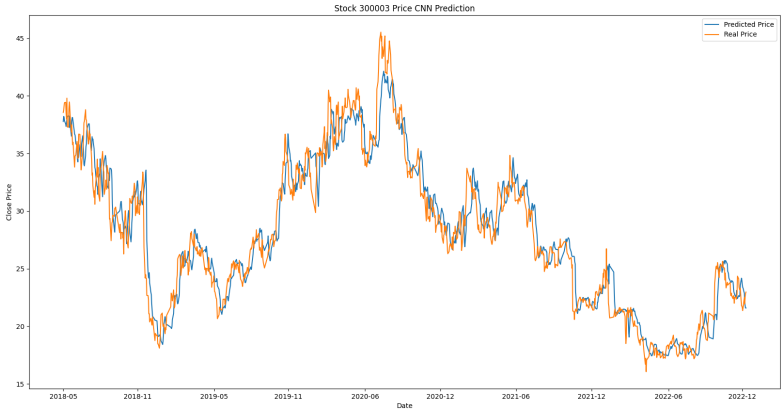

Figure 4.11: Stock 300003 Price Prediction Result

Stock Code	Mean Absolute Error	Mean Squared Error	Coefficient of Determination	Classification Accuracy Score
300003	0.0450	0.0037	0.92	0.90

Table 4.20: 300003 Prediction Results

The chart shows that the trend of the predicted and true values are basically the same. However, the difference between the two values is obvious. As shown in figure 4.11 and table 4.20.

(h) 000651 Gree Electric

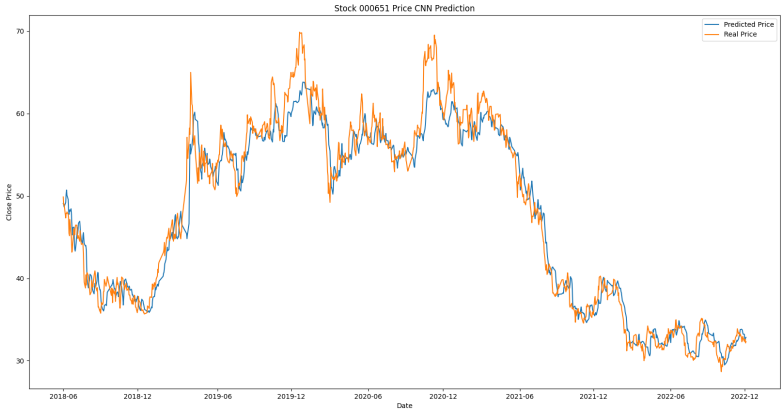

Figure 4.12: Stock 000651 Price Prediction Result

The forecast result for 000651 is similar to that of 600171. What these two have in common is the high frequency and amplitude of their price curves. This indicates that the model's ability to fit this type of stock is still lacking. As shown in figure 4.12 and table 4.21.

(i) 000300 CSI 300 Index

Stock Code	Mean Absolute Error	Mean Squared Error	Coefficient of Determination	Classification Accuracy Score
000651	0.0404	0.0030	0.96	0.92

Table 4.21: 000651 Prediction Results

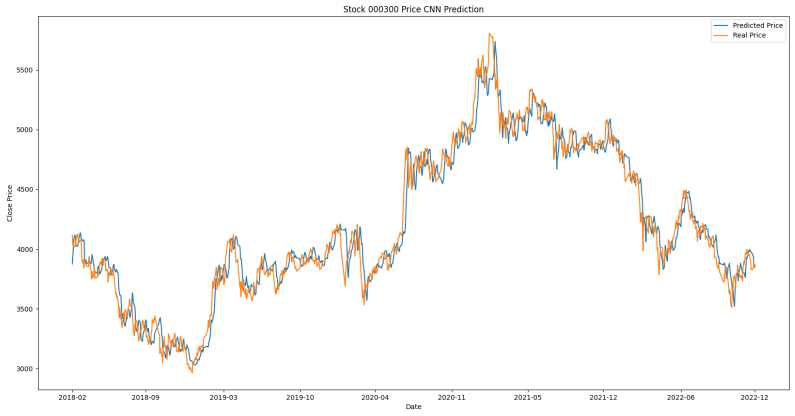

Figure 4.13: Stock 000300 Price Prediction Result

Stock Code	Mean Absolute Error	Mean Squared Error	Coefficient of Determination	Classification Accuracy Score
000300	0.0324	0.0019	0.96	0.94

Table 4.22: 000300 Prediction Results

000300 is the index of CSI 300. It is also the only index target in this thesis. The prediction of it can test the prediction ability of this model for the index. The values of the coefficient of determination and classification accuracy in the table are good. And the values of mean absolute error and mean square error are good as as well. As shown in figure 4.13 and table 4.22.

4.8 Summary of This Chapter

In this chapter, the software and hardware development environments for convolutional neural networks are first introduced. The data pre-processing is then described. Target stocks are selected according to certain rules, network models are constructed, evaluation indicators are set up, and training and testing data are constructed. Then the important hyperparameters of the convolutional neural network model are tested and compared one by one. The optimal values are selected as the model parameter values. Finally, the model is used to perform a real test on the target stock.

By comparing different types of hyperparameters, we found that several parameters, such as learning rate, activation function, and optimizer, have significant effects on the prediction of the model.

In addition we can observe the accuracy of regression prediction and classification prediction

from the data. Regression prediction we can see based on MAE, MSE, R^2 and other metrics. The most intuitive is the graph of prediction results. As we can see from the figure, the trend of the prediction result can be basically consistent with the actual trend. However, as a regression prediction, the prediction effect on the fluctuation magnitude is still lacking. As for the classification prediction, we can find from the classification prediction accuracy that the room for improvement has been very limited.

Finally, we found that the Dropout layer, introduced to prevent overfitting, has a certain probability of negative impact when training the model. Sometimes the trained model is significantly lower than the average in terms of accuracy. It is caused by the random deletion of neurons by Dropout.

5 Convolutional Neural Network Compared with Other Models

In order to evaluate convolutional neural network more objectively and comprehensively, and to understand its effectiveness on stock price prediction, this chapter will use other common neural networks to compare with it. Also, to ensure the validity of the results, we will perform parameter tuning for all the learners involved in the comparison.

5.1 Random Forest

The important parameters of random forest are: $n\_estimators$, max_depth, $max\_features$ $min\_samples\_leaf$, min_sample_split etc., where

$n\_estimators$ is the most important parameter of the random forest. It refers to the number of submodels contained in the model. That is, the number of trees in the forest. The larger its value, the better the prediction of the model. However, this is accompanied by a dramatic increase in memory consumption and training time, as well as the risk of overfitting. At the same time, the marginal benefit is diminishing after a certain point.

max_depth determines the maximum depth of each tree. Branches that exceed the depth will be cropped. It is closely related to the complexity of the individual submodels. We strike a balance between the accuracy and complexity of the model by adjusting its size.

$min\_samples\_leaf$ refers to the minimum number of samples that should be included on the self-node. It can reduce overfitting during training.

min_sample_split Parameter that tells the decision tree in a random forest the minimum required number of observations in any given node to split it. Default = 2

$max\_features$ Maximum number of features, limiting the maximum available features to limit the splitting of high-dimensional data. Usually set to "auto".

For decision tree-based models, the greater the number of trees, the greater the depth, and the larger the branches, the higher the complexity of the model. However, a model with too much complexity will be overfitted and too little will be underfitted. Both will affect the error value of the generalized prediction. So a balance needs to be found in the middle. As shown in figure 5.1.

To determine the optimal values of the above parameters, we also use 600109 as the input sample. The above 4 parameters except $max_features$ are tuned one by one by the grid search method. The grid search method is essentially an exhaustive method. That is, the parameters are tested one by one in a certain step within the range of possible values of the parameters to find the parameter that leads to the highest prediction accuracy of the validation set.[36]

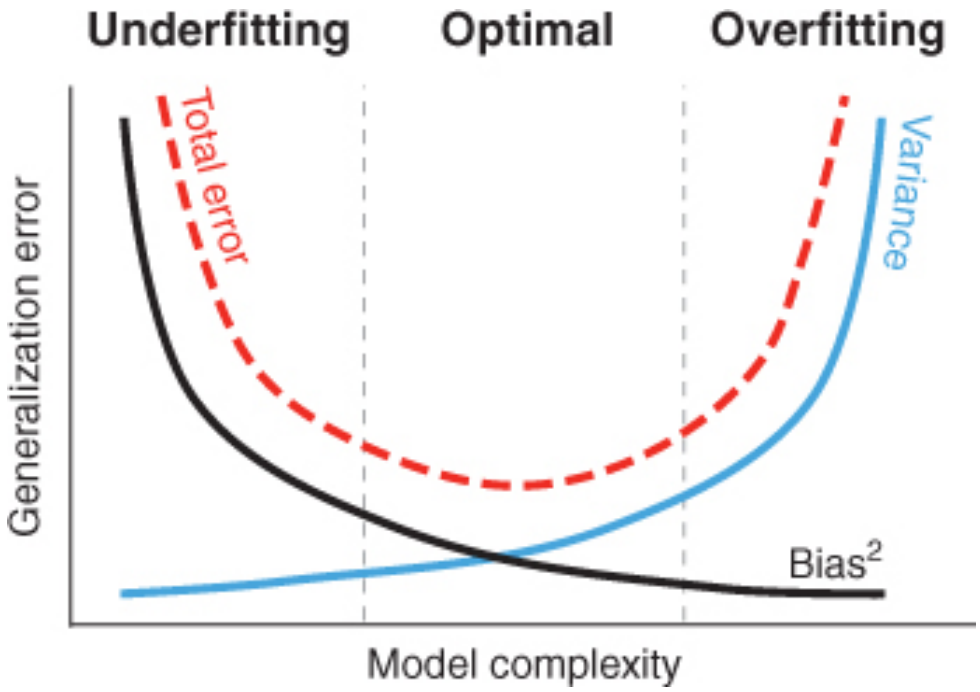

Figure 5.1: Complexity versus generalization error graph

n_estimators	Mean Absolute Error	Mean Square ErrorMSE	Coefficient of Determination R^2
20	0.0883	0.0052	0.92
40	0.0843	0.0045	0.91
60	0.0549	0.0044	0.92
80	0.0764	0.0051	0.91
100	0.0533	0.0040	0.92
120	0.0637	0.0061	0.92
140	0.0655	0.0056	0.92
160	0.0853	0.0077	0.92
180	0.0888	0.0091	0.92
200	0.0954	0.0082	0.91

Table 5.1: Random Forest *n_estimators* Parameter Tuning

It can be seen from table 5.1 that there is little difference in model performance as the number of decision trees increases. and stabilizes when the number of decision trees is around 100. This is caused by the small total amount of input data. Considering the impact of the number of decision trees on the model performance, we finally set the model parameter *n_estimators* to 100.

As can be seen from the table 5.2, the three evaluation metrics improve significantly as the number of *max_depth* increases. After reaching 5, they tend to stabilize, so the value of *max_depth* is determined as 5.

Due to the small number of samples, the change of *min_samples_leaf* has almost no effect on the prediction results. As shown in table 5.3. Taking the training efficiency into consideration, it is set to 2.

max_depth	Mean Absolute Error	Mean Square Error	Coefficient of Determination R^2
2	0.0514	0.0026	0.91
3	0.0477	0.0097	0.92
4	0.0458	0.0057	0.90
5	0.0643	0.0061	0.90
6	0.0671	0.0092	0.89
7	0.0983	0.0033	0.90

Table 5.2: Random Forest *max_depth* Parameter Tuning

min_samples_leaf	Mean Absolute Error	Mean Square Error	Coefficient of Determination R^2
2	0.0393	0.0022	0.92
3	0.0487	0.0034	0.92
4	0.0545	0.0046	0.91
5	0.0578	0.0044	0.91
6	0.0592	0.0035	0.91
7	0.0432	0.0033	0.91

Table 5.3: Random Forest *min_samples_leaf* Parameter Tuning

min_samples_split	Mean Absolute Error	Mean Square Error	Coefficient of Determination R^2
2	0.0134	0.0026	0.92
3	0.0240	0.0022	0.90
4	0.0233	0.0027	0.90
5	0.0213	0.0033	0.91
6	0.0353	0.0053	0.91
7	0.0321	0.0041	0.92

Table 5.4: Random Forest *min_samples_split* Parameter Tuning

Similar to *min_samples_leaf*, *min_samples_split* also has little effect on the prediction results of this model. Set it to 2. As shown in table 5.4.

Combining the above tests, the parameters for the construction of the random forest are shown in below table 5.5.

n_estimators	*max_depth*	*min_samples_leaf*	*min_samples_split*	*max_features*
100	3	2	2	auto

Table 5.5: Random Forest Parameters

5.2 Long Short-Term Memory Network

Long Short-Term Memory Network is more widely used in stock forecasting. Because it belongs to the derived category of recurrent neural networks. It is a sequential processing of data according to time series, which has an inherent advantage for time-varying stock

information. Further, it basically solves the gradient explosion and gradient disappearance of recurrent neural networks, and it has some memory effect. Therefore, it has a strong advantage over other deep learning methods for stock prediction. The long short-term memory network constructed in this thesis consists of LSTM, Dropout, and fully connected layers. The structure is shown in figure 5.2.

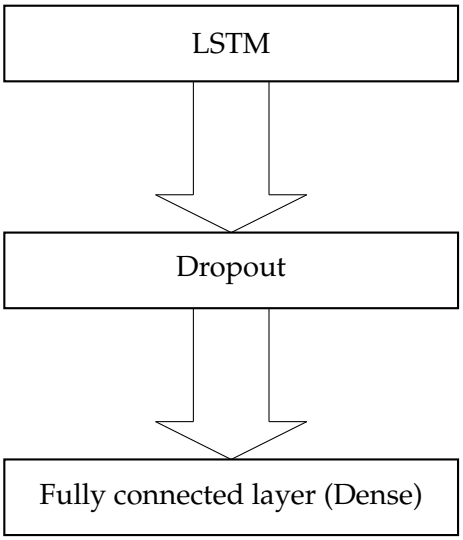

Figure 5.2: Long Short-Term Memory Network Structure

Long Short-Term Memory Networks belong to the same deep learning algorithm as convolutional neural networks. Therefore, they have similar hyperparameters, such as: learning rate, epoch, *batch_size*, activation function, and optimization function. However, the former has a unique *num_units* parameter, which represents the number of hidden neurons per feedforward network layer, i.e., the dimensionality of the output layer. Its size plays an important role in the prediction effect of the network. In the following, the hyperparameters of the long short-term memory network are also tuned using 600109 as input.

Number	Learning Rate	Epoch	Mean Absolute Error	Mean Squared Error	Coefficient of Determination R^2	Classification Accuracy Score
1	0.1	20	0.0452	0.0059	0.92	0.87
2	0.05	25	0.0531	0.0039	0.92	0.87
3	0.005	30	0.0557	0.0038	0.90	0.87
4	0.002	50	0.0654	0.0033	0.92	0.85
5	0.001	60	0.0478	0.0033	0.91	0.83
6	0.0004	70	0.0357	0.0029	0.92	0.89
7	0.0003	100	0.0352	0.0038	0.91	0.79
8	0.0001	150	0.0453	0.0039	0.92	0.79
9	0.00005	200	0.0383	0.0020	0.91	0.76

Table 5.6: Long Short-Term Memory Network Learning Rate Tuning

From the above table 5.6, it can be seen that as the learning rate decreases and the epoch increases, the metrics of the regression test show a stable trend, but the classification accuracy has decreased significantly. This is because the amount of input data in this thesis is small and too many iterations will lead to overfitting the model. Based on the above data, we determined a learning rate of 0.005 and an epoch of 30.

A larger Batch Size can theoretically increase the stability of the model. However, according

Number	BatchSize	Mean Absolute Error	Mean Squared Error	Coefficient of Determination R^2	Classification Accuracy Score
1	32	0.0571	0.0089	0.90	0.87
2	64	0.0628	0.0074	0.91	0.88
3	70	0.0524	0.0064	0.93	0.90
4	128	0.0531	0.0065	0.91	0.88
5	160	0.0540	0.0076	0.92	0.84
6	200	0.0518	0.0088	0.90	0.85

Table 5.7: Long Short-Term Memory Network Batch Size Tuning

to the data in table 5.7, the accuracy of classification prediction decreases significantly when the Batch Size increases. This indicates that when Batch Size is 32 to reach a better level. In the process of continuing to increase its value, it causes the model to fail to jump out of the local optimal solution, making its generalization ability poor.

Number	Units	Mean Absolute Error	Mean Squared Error	Coefficient of Determination R^2	Classification Accuracy Score
1	32	0.0301	0.0059	0.90	0.87
2	64	0.0288	0.0065	0.92	0.91
3	128	0.0254	0.0062	0.91	0.90
4	256	0.0198	0.0036	0.95	0.93

Table 5.8: Long Short-Term Memory Network Units Tuning

From table 5.8, it can be seen that the Units hyperparameter has a significant effect on the long short-term memory network. In this model, it tends to be stable when it is greater than 64, so 64 is chosen as the model parameter.

Number	Dropout	Mean Absolute Error	Mean Squared Error	Coefficient of Determination R^2	Classification Accuracy Score
1	0.2	0.0411	0.0028	0.92	0.90
2	0.3	0.0358	0.0023	0.93	0.91
3	0.5	0.0452	0.0027	0.91	0.88
4	0.6	0.0354	0.0058	0.92	0.88
5	0.7	0.0551	0.0075	0.88	0.87

Table 5.9: Long Short-Term Memory Network Dropout Tuning

As the Dropout probability increases, the number of deleted neurons increases, and the prediction of the model oscillates down, apparently due to the effect of overfitting. Here we choose 0.2 as the value of the model parameter Dropout. As shown in figure 5.9.

Number	Activation Function	Mean Absolute Error	Mean Squared Error	Coefficient of Determination R^2	Classification Accuracy Score
1	TanH	0.0212	0.0018	0.92	0.91
2	ReLU	0.0263	0.0034	0.92	0.87
3	Leaky ReLU	0.0583	0.0052	0.90	0.85

Table 5.10: Long Short-Term Memory Network Activation Function Tuning

The data in table 5.10 reflect that the use of TanH as the activation function in long short-term memory networks is significantly better than other activation functions.

Combining the above tests, the hyperparameters of the Long Short-Term Memory Network are determined as follows in table 5.11:

Learning Rate	Epoch	BatchSize	Units	Dropout	Activation Function
0.004	70	32	256	0.3	TanH

Table 5.11: Long Short-Term Memory Network Hyperparameter Table

5.3 Model Ensemble

In the previous section, we constructed convolutional neural networks, random forests, and long short-term memory networks for regression prediction, respectively. In order to improve the generalization ability of the model prediction and optimize the overall performance of the model, we try to integrate the three previous models through certain strategies. The ensemble of models is used to alleviate the overfitting or underfitting of individual models and improve the overall prediction effect.

In 1990, Hansen et al. proposed ensemble learning of neural networks, treating each neural network as a weak classifier, combining the prediction results of multiple neural networks into the final output, and optimizing the parameters of neural networks using cross-validation methods, Hansen's experimental results showed that ensemble learning of neural networks could improve the generalization ability of neural networks. [37].

The general idea of multi-model ensemble is to train some weak models and then aggregate these models as components to combine them into a strong model [38]. In general ensemble methods are divided into two main categories. One category is homogeneous model ensemble, which refers to aggregated multiple models based on the same type of base learner. The other category is heterogeneous ensemble, which is just the opposite of the former. In the following, we briefly examine three common ensemble methods.

(a)Bagging(bootstrap aggregating) In 1996 Leo Breiman proposed the Bagging algorithm.[39]. The entire training set is randomly sampled for T times using the automatic sampling method. Each sampling acquires m samples. Then T models are obtained by training T learners with T sample sets. These T models are then used to form a strong model through a strategy [40]. As shown in figure 5.3.

The focus of Bagging is on the self-sampling method. It is a sampling with put-back. When using a training set containing m samples, we randomly collect one sample from the entire sample set and put it into the training set. Put this sample back again, and repeat this process m times. For each sample the probability of being picked each time is $\frac{1}{m}$, and the probability of not being picked on the contrary is $1 - \frac{1}{m}$. Then the probability that the same sample is not picked m times is $(1 - \frac{1}{m})^m$. That is, when the capacity of the samples is large enough, there will be $\lim_{x\to\infty}(1 - \frac{1}{m})^m \to \frac{1}{e} \approx 0.368$ samples in the training set that are not picked. This part of the data is called out-of-bag data, and they are not involved in the training of the model, which improves the generalization ability of the model. It should be noted that Bagging belongs to one of the homogeneous ensembles. It is a relatively simple integration strategy for solving classification problems that can be judged using the voting method, while the regression problems use arithmetic average.

(b)Boosting

Boosting uses the entire training set to train each learner in turn. Then model ensemble is

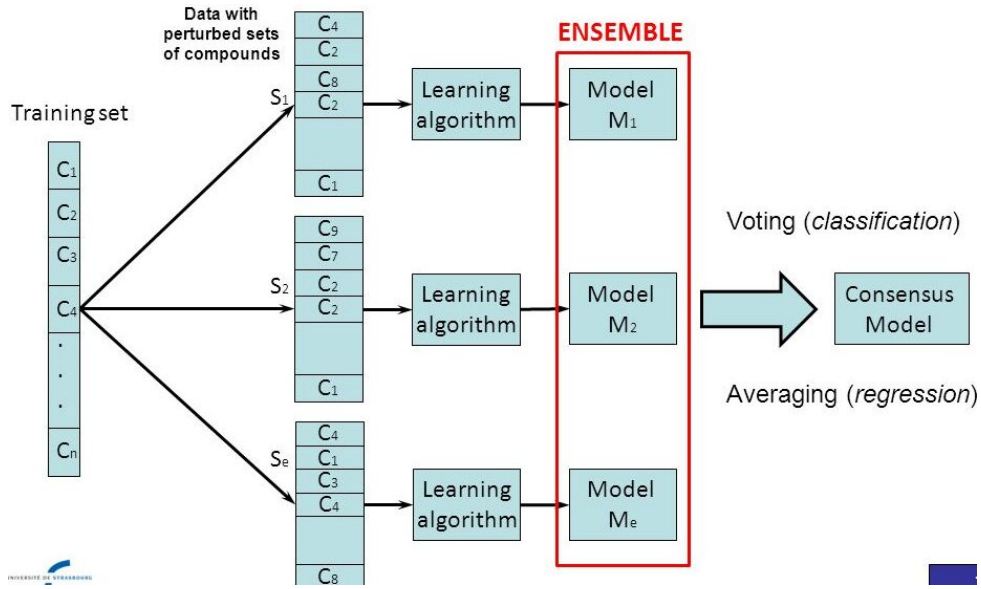

Figure 5.3: Bagging Ensemble Method

done according to some strategy. After each training, it automatically adjusts the weights of the samples according to the training effect. This is done sequentially until the T learners are trained. As shown in figure 5.4.

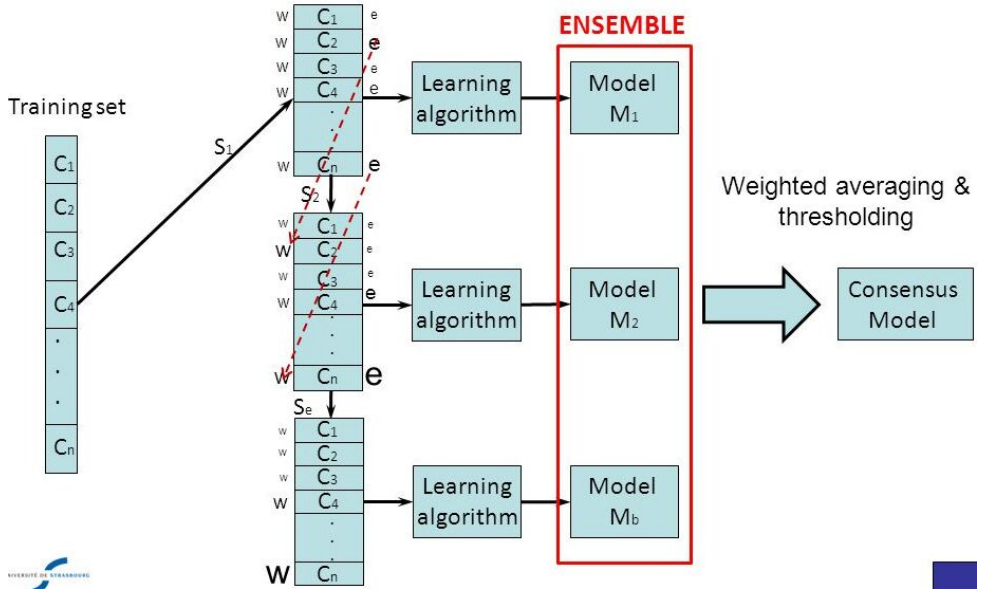

Figure 5.4: Boosting Ensemble Method

Boosting is also applicable to homogeneous ensemble. The representative algorithms include AdaBoost algorithm and boosting tree algorithm.

(c)Stacking

Stacking is also called the stacking method. It is a layered ensemble framework. It usually

consists of two or more layers of learners. Multiple base learners form its first layer, and the input to each learner is the same original training set data. The latter layer can be one or more learners with the input as the output of the previous layer. figure 5.5 shows the Stacking framework as an example of the learner in this thesis.

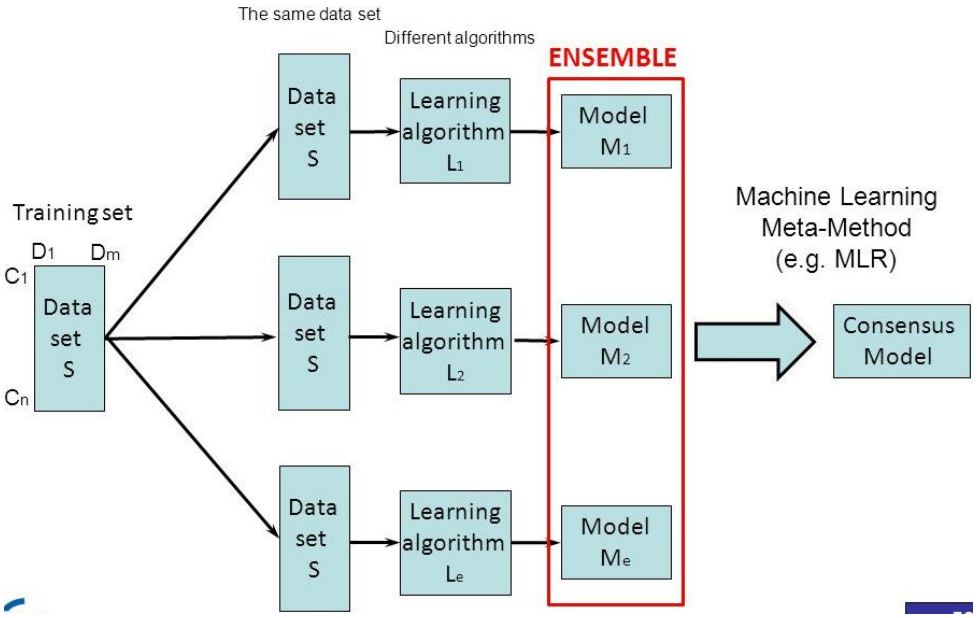

Figure 5.5: Stacking Ensemble Method

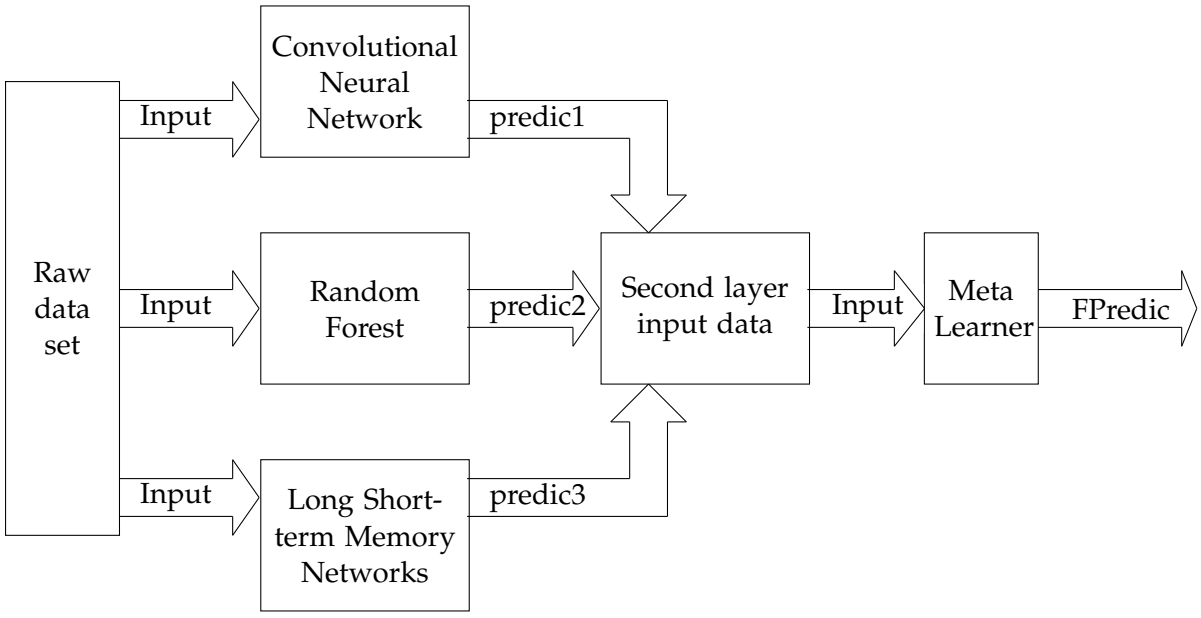

Figure 5.6: Stacking Framework

As you can see from the figure 5.6, Stacking uses the output of the first layer as the second layer training data, which is used to train the second layer learner (also called meta-learner)

to finally produce the results. The essence is to train the second layer learner so that it can automatically assign weights to the answers given by the first layer model. We can also find that the structure of the Stacking integration framework is extremely similar to that of a neural network. And its effectiveness depends mainly on the extraction of features by front layer learning. Therefore it is more suitable for heterogeneous ensemble, so that more different features can be obtained.

The above analysis shows that for the three different types of existing models, it is obvious that the Stacking model ensemble approach is better.

So how does the second layer of learners get selected? It is argued that numerous learners in the first layer already make extensive use of complex nonlinear transformations, so the complexity can be controlled by using linear transformations for the learners in the second layer. However, from the model in this thesis, the matrix consisting of the outputs of the three existing models may be closely correlated and ultimately not correctly predicted. In this way, the least squares method used in linear regression can make the results unstable. That is, the output results will have a large effect if there is noise interference in the input variables. To solve this problem, a penalty term (also called a regularization term) is introduced in ridge regression through the objective function as a way to limit the size of the model parameter values and thus reduce the sensitivity of the model to noise in the input parameters. However, in ridge regression, it is necessary to manually set the α hyperparameters and then evaluate the advantages and disadvantages of α, which causes inconvenience in use. Thus, we consider applying Bayesian ridge regression to construct a meta learner.

In Bayesian ridge regression, the conjugate prior is therefore chosen to be performed in order to reduce the complexity of the computation [41]. First assume that the likelihood function obeys a normal distribution $N(\omega^t x_i, \beta^{-1})$ and the prior obeys a normal distribution $N(m, \alpha^{-1})$. Based on the self-conjugating property of the normal distribution, it can be assumed that the posterior distribution will also be normal. According to Bayes' theorem, the parameters of the posterior normal distribution can be expressed as follows:

$$\mu = \sigma(\alpha m + \beta x^T y) = \sigma(\alpha m + \beta x^T x \omega^*) \tag{5.1}$$

$$y_i = \sigma^{-1} = \alpha + \beta x^T x \tag{5.2}$$

From the above equation, it can be seen that the mean of the posterior distribution μ is composed of the prior parameter α and the likelihood parameter β together. The precision of the prior determines the weight of the posterior mean. In Bayesian regression, the posterior of the previous sample is used as the prior for the next estimation. Essentially, a Gaussian prior with a mean that can be 0 is adde. We then let the mean m = 0, and using the maximum a posteriori estimate, omitting the constant term yields the Bayesian ridge regression equation:

$$argmin_w \sum_{i=1}^{m} \frac{\beta}{2}(y_i - \omega^T x_i)^2 + \sum_j^d \frac{\alpha}{2}(\omega_j)^2 \tag{5.3}$$

It can be seen that Bayesian ridge regression uses a step wise updating of the prior as opposed to ordinary ridge regression. And the standard deviation of the normal distribution cannot be infinite. Therefore the hyperparameter α cannot be 0. Therefore, Bayesian

ridge regression is not likely to degenerate into least squares regression like ordinary ridge regression, which has high robustness.

5.4 Model Comparison Tests

We use each of the above 4 models to forecast 10 stocks representing different sectors in order to compare their actual results.

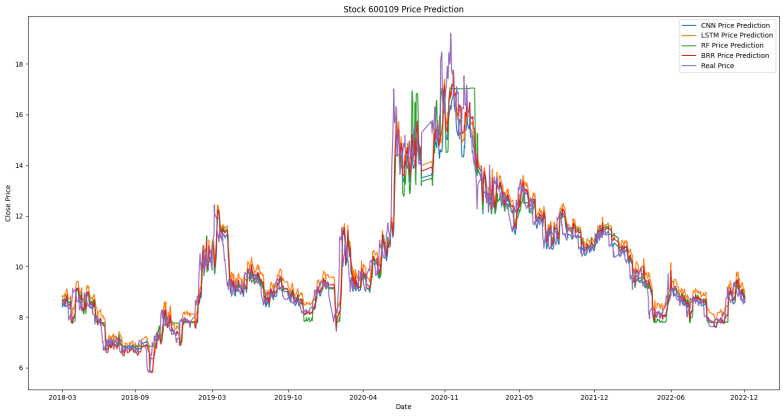

Figure 5.7: Stock 600109 Price Prediction

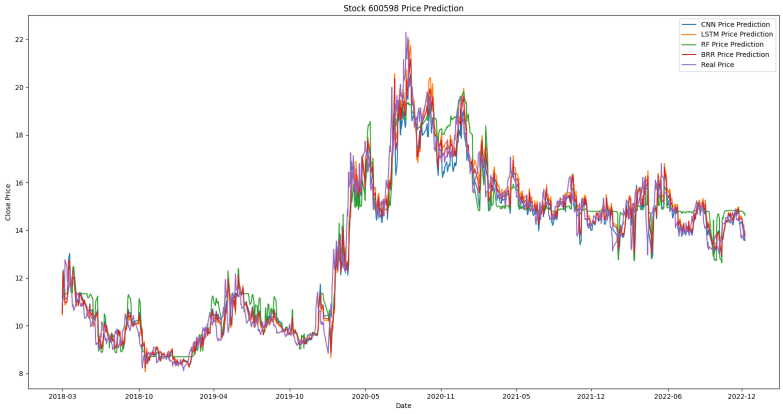

Figure 5.8: Stock 600598 Price Prediction

Analyzing theses graphs, it is easy to see that the model integration, which draws on the strengths of many learners, achieves excellent results in both regression and classification predictions, except for the prediction results of 600660 and 000651 stocks, which are relatively weak due to the influence of random forest errors. The overall effect of random forest is relatively the worst among the 4 prediction models. In particular, the prediction error is especially obvious when the frequency of input data fluctuation is large. The convolutional

Figure 5.9: Stock 600660 Price Prediction

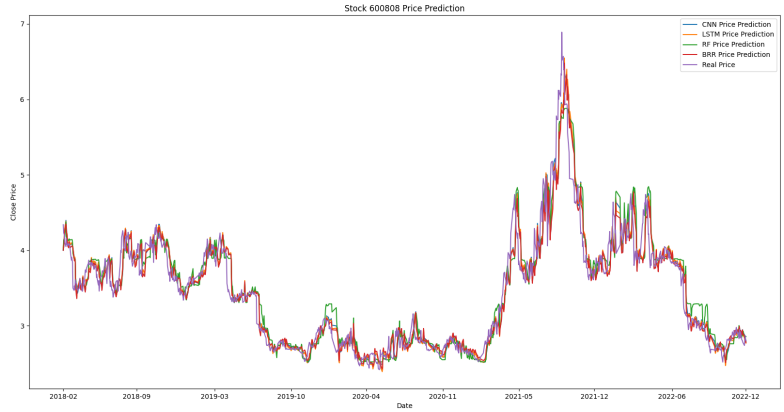

Figure 5.10: Stock 600808 Price Prediction

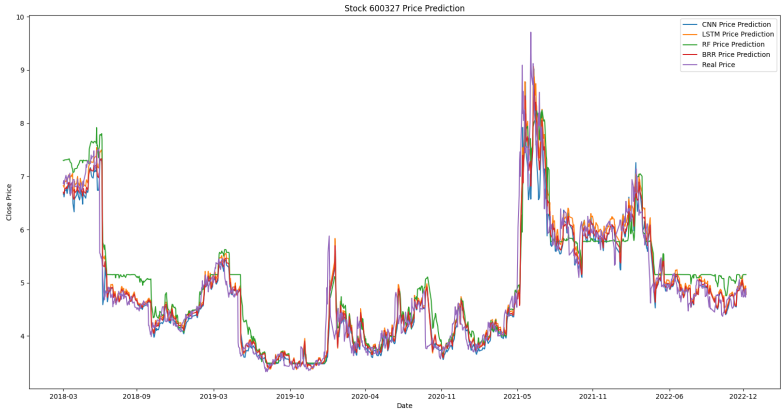

Figure 5.11: Stock 600327 Price Prediction

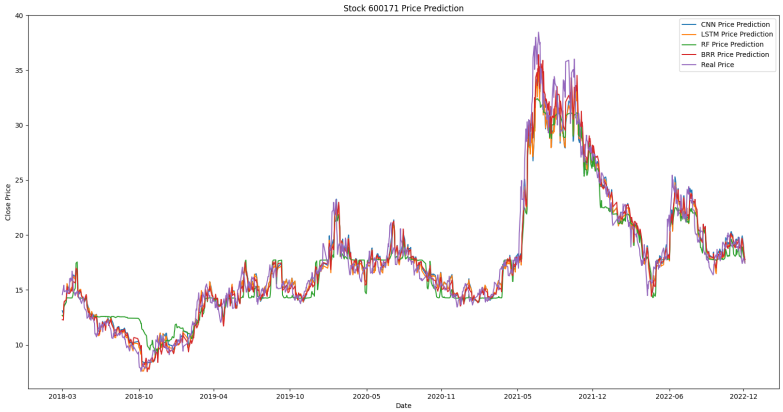

Figure 5.12: Stock 600171 Price Prediction

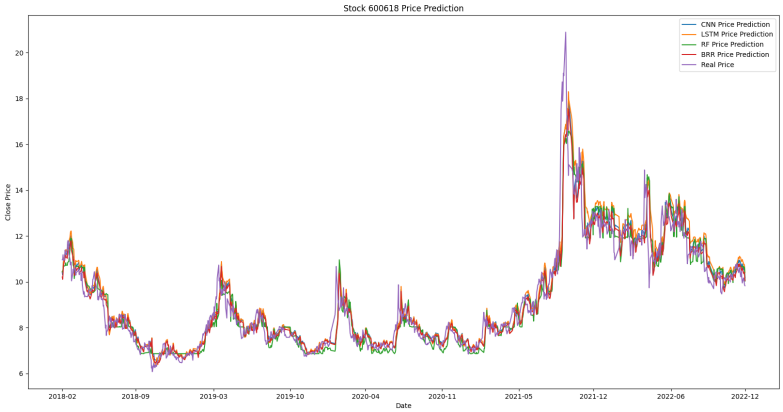

Figure 5.13: Stock 600618 Price Prediction

Figure 5.14: Stock 300003 Price Prediction

Figure 5.15: Stock 000651 Price Prediction

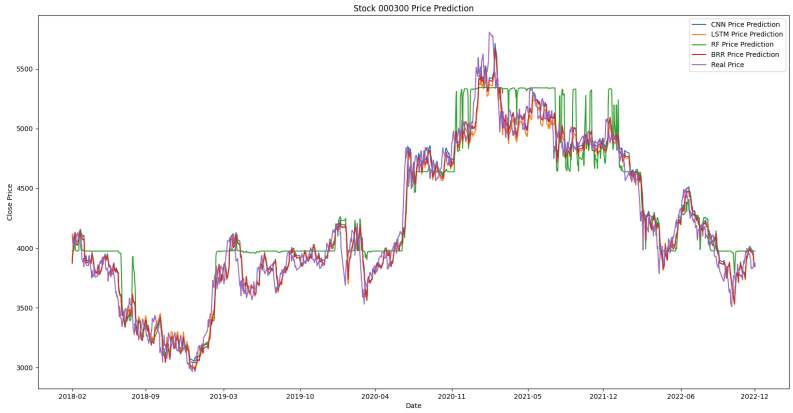

Figure 5.16: CSI Index 000300 Price Prediction

Models	CNN				RF				LSTM				Model Ensemble			
Stock/Metrics	MAE	MSE	R^2	ACC	MAE	MSE	R^2	ACC	MAE	MSE	R^2	ACC	MAE	MSE	R^2	ACC
600109	0.0322	0.0025	0.9630	0.9480	0.0365	0.0031	0.9265	0.8938	0.0326	0.0026	0.9518	0.9401	0.0324	0.0023	0.9768	0.9488
600598	0.0336	0.0023	0.9535	0.9410	0.0366	0.0031	0.9298	0.8795	0.0354	0.0023	0.9535	0.9306	0.0333	0.0022	0.9560	0.9460
600660	0.0295	0.0019	0.9684	0.9160	0.0483	0.0042	0.9306	0.8136	0.0306	0.0020	0.9665	0.8901	0.0295	0.0018	0.9991	0.9169
600808	0.0313	0.0022	0.9158	0.9376	0.0366	0.0028	0.8933	0.8813	0.0323	0.0023	0.9132	0.9197	0.0314	0.0022	0.9167	0.9410
600327	0.0356	0.0037	0.8878	0.9340	0.0491	0.0052	0.8375	0.8247	0.0360	0.0037	0.8848	0.9194	0.0345	0.0035	0.8904	0.9356
600171	0.0306	0.0020	0.9446	0.9337	0.0384	0.0028	0.9250	0.8343	0.0307	0.0021	0.9439	0.9102	0.0300	0.0018	0.9500	0.9398
600618	0.0288	0.0024	0.8934	0.9385	0.0317	0.0027	0.8767	0.8897	0.0291	0.0024	0.8880	0.9307	0.0280	0.0023	0.8936	0.9470
300003	0.0450	0.0037	0.9238	0.9030	0.0501	0.0043	0.9108	0.8150	0.0453	0.0037	0.9226	0.8958	0.0450	0.0036	0.9251	0.9054
000651	0.0404	0.0030	0.9619	0.9286	0.0718	0.0145	0.8013	0.8004	0.0397	0.0029	0.9613	0.9194	0.0386	0.0027	0.9838	0.9421
000300	0.0324	0.0019	0.9617	0.9487	0.0520	0.0044	0.9102	0.8479	0.0331	0.0019	0.9612	0.9248	0.0325	0.0018	0.9821	0.9625

Table 5.12: Multiple Model Prediction Effects

neural network model and the long and short-term memory network model have the most stable predictions. Excluding the uncertainty caused by the random factor in Dropout, the former is relatively better in regression prediction because of the advantage of convolutional neural network in fitting the price graph, while the latter is slightly better in classification

prediction.

In this experiment, we chose the convolutional neural network algorithm to build the stock price prediction model. The reason for this is not only due to the graphs and charts of the prediction results from various models, but also due to the following factors.

First, the long and short-term memory network algorithm is very good at processing time series data such as financial data. There has been a great deal of research using it focusing on related problems. As described in the introduction section of this thesis, the typical application scenarios of convolutional neural networks are image processing, speech recognition, video analysis, etc. It is an important tool for computer vision and artificial intelligence. The use of convolutional neural network algorithms to study and analyze financial data problems is still a rarely addressed area. Here we have made a brave attempt to do this. Fortunately, the results obtained are satisfactory.

Secondly, the long and short-term memory model is not as good as the convolutional neural network model for fitting the prices of stocks like 600660, 600109 and 600327 which have continuous and sharp price changes price changes. In particular, the two evaluation metrics, classification accuracy score and determination coefficient, illustrate this point very well.

Furthermore, in the comparison test of these models, we have tuned their parameters to different degrees by labeling more training data, changing the batchsize, modifying the loss function, changing the dropout strategy, and so on. In this process, the convolutional neural network performed the best. Often a slight tuning can improve the generalization ability of the model very well. In contrast, the long and short-term memory network model and the random forest model did not perform so well. It takes several parameter tuning to slightly improve their generalization ability.

Finally there is an important point. In this experiment we only selected data that are closely correlated with stock prices for the study. We dropped other factors that are less correlated with stock prices, such as stock turnover, volume and P/E ratio. For possible future extensions of the prediction model, we chose a convolutional neural network algorithm to implement this experiment.

5.5 Summary of This Chapter

In this chapter, we first perform parameter tuning for random forest, long and short-term memory networks, and test their practical effects. Subsequently, Bayesian ridge regression models were constructed by model integration using the outputs of convolutional neural networks, random forests, and long- and short-term memory networks as raw data. Finally, by comparing the prediction effects of the four models, it is demonstrated that convolutional neural networks have certain advantages in regression prediction. It should be noted that although the model integration is better for prediction, it is not a stand-alone algorithm. It relies on the prediction results of other models. Essentially, it is an integration and correction of the prediction results of multiple models.

6 Summary and Prospect

This thesis begins by introducing the background of the study, and then explores the purpose and relevance of the study. The next summarizes the research methods and directions in this field. Then a brief overview of common data factors for stock trading and the fundamentals of the common AI learning algorithms available today is presented.

6.1 Research Summary

The focus of this study is on the selection of the stock and input factors, the construction of the convolutional neural network model and the testing process of its hyperparameters. Nine stocks representing different sectors and the CSI 300 index are chosen as the test benchmark in order to test the generalization ability of the model. After testing, the model was found to have satisfactory prediction results. It is a better fit for stock price trends in most industries. In particular, the price movements of representative stocks such as financial, retail and chemicals are best fitted. For stocks in other sectors it is basically possible to correctly determine the trend of their price changes. However, there is some error in the model's regression prediction results, especially when stock prices fluctuate sharply in one direction.

The important hyperparameters of convolutional neural network are studied and tested one by one. Five parameters such as the learning rate, the size and step size of the convolution kernel, the number of filters and the optimizer are found to have a significant impact on the prediction performance of the model.

To further validate the effectiveness of convolutional neural networks, we introduced random forests, long and short-term memory networks, and Bayesian ridge regression as comparison tests. The test data show that both the convolutional neural network and the long and short-term memory network are stable in their predictions and far better than the random forest. According to the key metric R^2 for regression prediction, the convolutional neural network has a clear advantage in regression prediction, while the long and short-term memory network has a slight lead in classification prediction. Bayesian ridge regression, which integrates the strengths of all the techniques, is superior in both regression and classification.

The feasibility and effectiveness of convolutional neural network in quantitative timing is well illustrated by modeling analysis comparison. For investors and investment institutions, trading in this way will undoubtedly help to increase the win rate and increase the certainty and stability of trading. It also substantially increases the efficiency of the investment.

6.2 Shortcomings and Prospects

There are some shortcomings in the research of this thesis.

First, in the process of forecasting, the model is usually more accurate for stocks with upward or downward trending prices. And the probability of deviation is higher for stocks that are in a wide range of oscillations for a long period of time.

Second, the introduction of Dropout layer alleviates the possible overfitting phenomenon during the training process. However, due to the uncertainty brought by its implementation mechanism, the learner does not get consistent results during each training session.

In subsequent studies, we can try to improve the existing situation by working on the following areas.

The first is to expand the data sources. The training data used in this thesis for each stock is just over 2000 items. This is less than adequate for the learning of neural networks. In future studies, a switch to higher frequency data could be considered. For example, replace the daily data with 120-minute, 60-minute data. This allows more data to be acquired for more adequate training.

Secondly, the weaknesses such as the poor effect of wide oscillations in forecasting should be caused by the limitations of the model itself. Then the model integration we discuss in Chapter 5 will have a chance to be the superior choice. There are certain flaws inherent to each model. We compare the results of each model and then perform secondary training to complement the strengths and weaknesses, which is certainly very effective in compensating for the weaknesses of a single model.

Finally, if the quantitative timing studied in this thesis can be effectively combined with quantitative stock selection, the probability of achieving excess returns will be further increased.

List of Tables

List of Figures

Bibliography

[1] H. L. White, "Economic prediction using neural networks: the case of ibm daily stock returns," *IEEE 1988 International Conference on Neural Networks*, pp. 451–458 vol.2, 1988.

[2] T. Kimoto, K. Asakawa, M. Yoda, and M. Takeoka, "Stock market prediction system with modular neural networks," *1990 IJCNN International Joint Conference on Neural Networks*, pp. 1–6 vol.1, 1990.

[3] N. Baba and M. Kozaki, "An intelligent forecasting system of stock price using neural networks," in *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, vol. 1, 1992, pp. 371–377 vol.1.

[4] E. Fama and K. French, "Common risk factors in the returns on stocks and bonds," *Journal of Financial Economics*, vol. 33, no. 1, pp. 3–56, 1993. [Online]. Available: https://EconPapers.repec.org/RePEc:eee:jfinec:v:33:y:1993:i:1:p:3-56

[5] R. Gencay, "Non-linear prediction of security returns with moving average rules," *Journal of Forecasting*, vol. 15, pp. 165–174, 1996.

[6] M. M. Carhart, "On persistence in mutual fund performance," *The Journal of Finance*, vol. 52, no. 1, pp. 57–82, 1997. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6261.1997.tb03808.x

[7] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, p. 5–32, oct 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[8] G. P. Zhang, "Time series forecasting using a hybrid arima and neural network model," *Neurocomputing*, vol. 50, pp. 159–175, 2003.

[9] Q. Cao, K. B. Leggio, and M. J. Schniederjans, "A comparison between fama and french's model and artificial neural networks in predicting the chinese stock market," *Computers and Operations Research*, vol. 32, no. 10, pp. 2499–2512, 2005, applications of Neural Networks. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S030505480400067X

[10] A. M. Ozbayoglu and I. Bahadir, "Comparison of Bayesian Estimation and Neural Network Model in Stock Market Trading," in *Intelligent Engineering Systems through Artificial Neural Networks Volume 18*. ASME Press, 01 2008. [Online]. Available: https://doi.org/10.1115/1.802823.paper73

[11] H. Liu and J. Wang, "Integrating independent component analysis and principal component analysis with neural network to predict chinese stock market," *Mathematical Problems in Engineering*, vol. 2011, pp. 1–15, 2011.

[12] M. U. Yaqub and M. S. Al-Ahmadi, "Application of combined arma-neural network models to predict stock prices," *Proceedings of the The 3rd Multidisciplinary International Social Networks Conference on SocialInformatics 2016, Data Science 2016*, 2016.

[13] A. H. Moghaddam, M. H. Moghaddam, and M. Esfandyari, "Stock market index prediction using artificial neural network," *Journal of Economics, Finance and Administrative Science*, vol. 21, no. 41, pp. 89–93, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2077188616300245

[14] M. Abe and H. Nakayama, "Deep learning for forecasting stock returns in the cross-section," in *Advances in Knowledge Discovery and Data Mining: 22nd Pacific-Asia Conference, PAKDD 2018, Melbourne, VIC, Australia, June 3-6, 2018, Proceedings, Part I 22*. Springer, 2018, pp. 273–284.

[15] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: A neural network model for a mechanism of visual pattern recognition," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, pp. 826–834, 1983.

[16] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, pp. 541–551, 1989.

[17] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[18] K. Alex, S. Ilya, and E. Geoffrey, "Imagenet classification with deep convolutional neural networks," in *[Proceedings 2012] ICNEPS International Conference on Neural Information Processing System*, 2012, pp. 1–5. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[19] L. Di Persio and O. Honchar, "Artificial neural networks architectures for stock price prediction: Comparisons and applications," *International journal of circuits, systems and signal processing*, vol. 10, no. 2016, pp. 403–413, 2016.

[20] O. Sezer and M. Ozbayoglu, "Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach," *Applied Soft Computing*, vol. 70, 04 2018.

[21] S. Ruiqi, "A study of lstm neural network-based price trend prediction model for u. s. stock indexes," diplomathesis, Capital University of Economics and Business, 2016.

[22] W. Shuyan, C. Zhengfeng, and C. Mingzhi, "Research on application of random forests in the quantitative stock selection model," *Operations Research and Management Science*, vol. 25, no. 3, pp. 163–168, 2016.

[23] Z. Xiao and W. Zengxin, "Application of random forest in stock trend forecasting," *China Management Informationization*, vol. 21, no. 3, pp. 163–168, 2018.

[24] Q. Ruoyu, "Stock prediction model based on neural network," *Operations Research and Management Science*, vol. 28, no. 10, pp. 132–140, 2019.

[25] C. Xiangyi, "Research on csi 300 index prediction method based on convolutional neural network," diplomathesis, Beijing University of Posts and Telecommunications, 2018.

[26] X. Qi, C. Gengguo, and X. Xu, "Research based on stock predicting model of neural networks ensemble learning," *Computer Engineering and Applications*, vol. 55, no. 8, pp. 238–243, 2019.

[27] Z. QiHongrui and X. Lei, "Research on stock forecasting based on lstm-cnn-cbam model," *Computer Engineering and Applications*, vol. 57, no. 3, pp. 203–207, 2021.

[28] P. McCorduck, *Machines Who Think (2Nd Ed.)*. A. K. Peters, 2004.

[29] A. M. Andrew, "The handbook of artificial intelligence, vols. 1 and 2 by avron barr and edward a.

feigenbaum; vol. 3 by paul r. cohen and edward a. feigenbaum, pitman, london. vol. 1 published 1981, vols. 2 and 3 in 1982. vol. 2, 428 pp., vol. 3, 639 pp. (respective prices £15.00, £22.50, £28.50)." *Robotica*, vol. 1, no. 2, p. 110–110, 1983.

[30] J. Ali, R. Khan, N. Ahmad, and I. Maqsood, "Random forests and decision trees," *International Journal of Computer Science Issues (IJCSI)*, vol. 9, no. 5, p. 272, 2012.

[31] D. A. Pisner and D. M. Schnyer, "Chapter 6 - support vector machine," in *Machine Learning*, A. Mechelli and S. Vieira, Eds. Academic Press, 2020, pp. 101–121. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128157398000067

[32] A. Caterini and D. Chang, *Deep Neural Networks in a Mathematical Framework*, ser. SpringerBriefs in Computer Science. Springer International Publishing, 2018. [Online]. Available: https://books.google.de/books?id=aM5SDwAAQBAJ

[33] A. Graves, *Long Short-Term Memory*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 37–45. [Online]. Available: https://doi.org/10.1007/978-3-642-24797-2_4

[34] S. Indolia, A. K. Goswami, S. Mishra, and P. Asopa, "Conceptual understanding of convolutional neural network- a deep learning approach," *Procedia Computer Science*, vol. 132, pp. 679–688, 2018, international Conference on Computational Intelligence and Data Science. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050918308019

[35] J. Wu, "Introduction to convolutional neural networks," *National Key Lab for Novel Software Technology. Nanjing University. China*, vol. 5, no. 23, p. 495, 2017.

[36] D. Belete and M. D H, "Grid search in hyperparameter optimization of machine learning models for prediction of hiv/aids test results," *International Journal of Computers and Applications*, vol. 44, pp. 1–12, 09 2021.

[37] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE transactions on pattern analysis and machine intelligence*, vol. 12, no. 10, pp. 993–1001, 1990.

[38] T. G. Dietterich, "Ensemble methods in machine learning," in *Multiple Classifier Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 1–15.

[39] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 2004.

[40] B. Efron and R. Tibshirani, "Improvements on cross-validation: the 632+ bootstrap method," *Journal of the American Statistical Association*, vol. 92, no. 438, pp. 548–560, 1997.

[41] A. G. Assaf, M. Tsionas, and A. Tasiopoulos, "Diagnosing and correcting the effects of multicollinearity: Bayesian implications of ridge regression," *Tourism Management*, vol. 71, pp. 1–8, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0261517718302164

Appendices

Appendix 1 Model Training Log

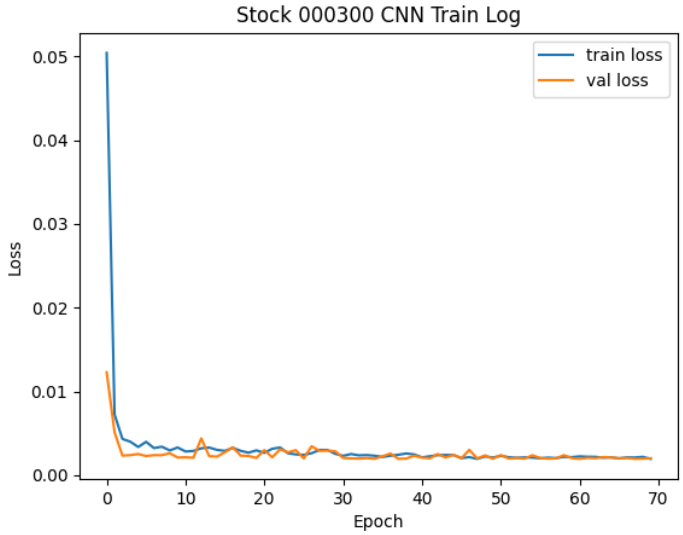

Figure 1: Stock 000300 CNN Train Log

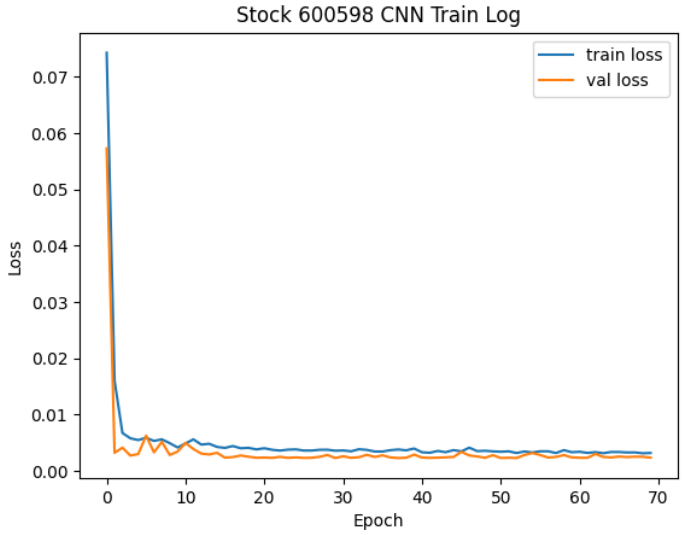

Figure 2: Stock 600598 CNN Train Log

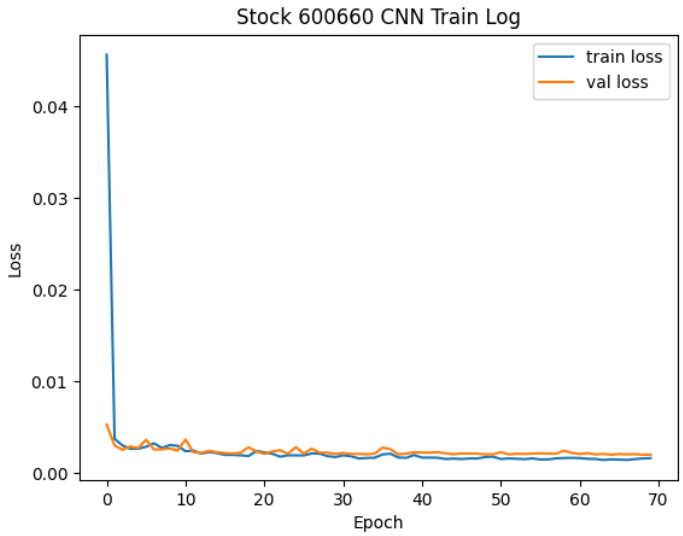

Figure 3: Stock 600660 CNN Train Log

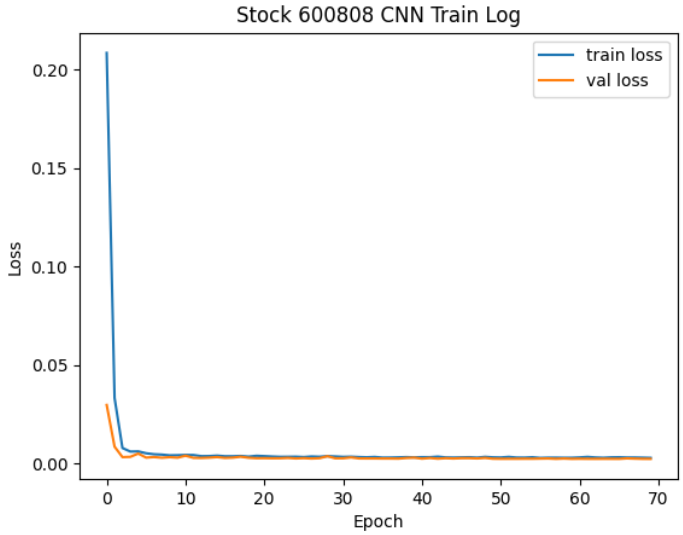

Figure 4: Stock 600808 CNN Train Log

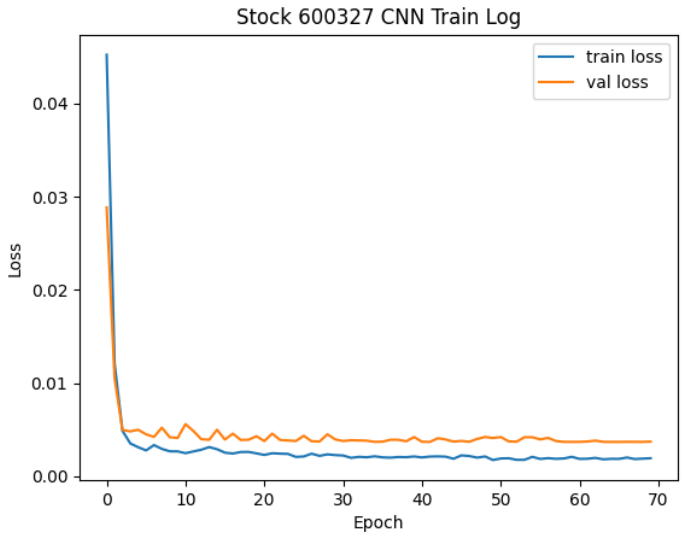

Figure 5: Stock 600327 CNN Train Log

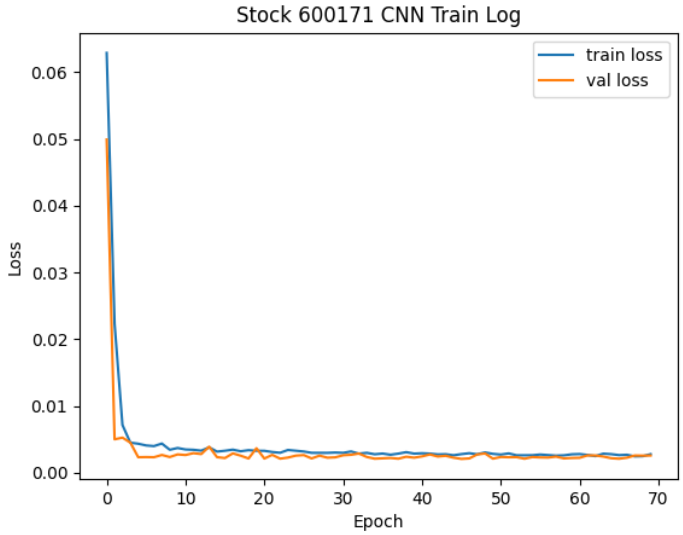

Figure 6: Stock 600171 CNN Train Log

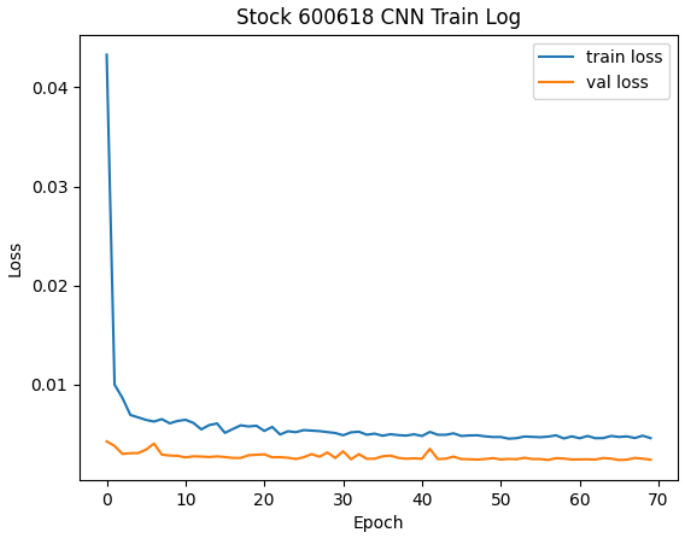

Figure 7: Stock 600618 CNN Train Log

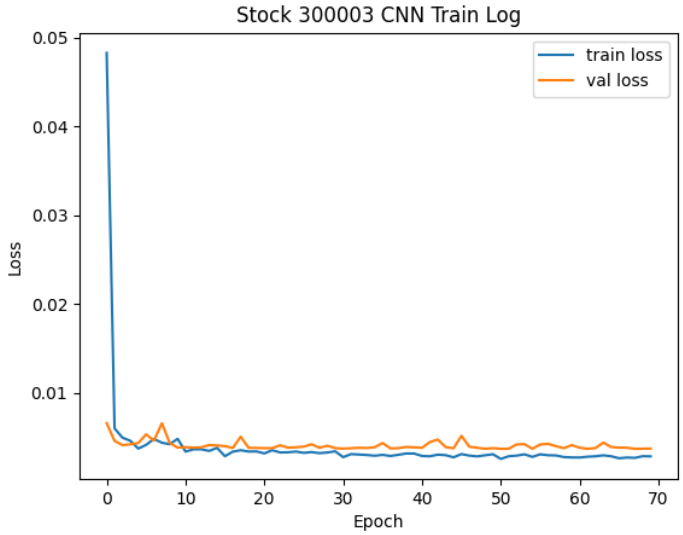

Figure 8: Stock 300003 CNN Train Log

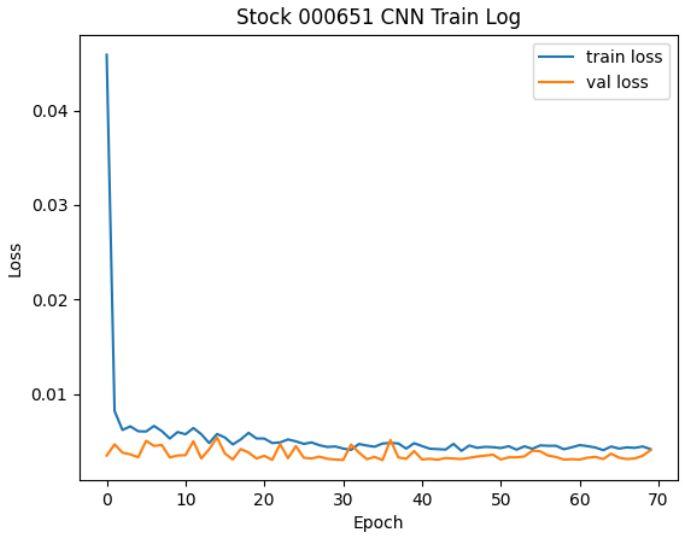

Figure 9: Stock 000651 CNN Train Log

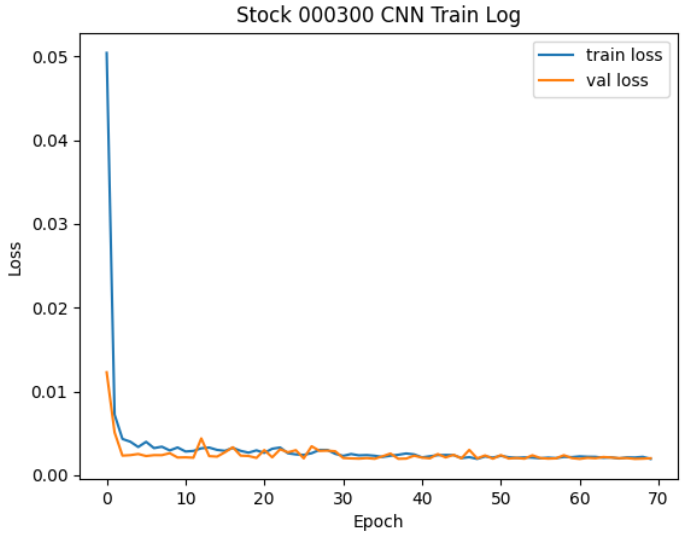

Figure 10: Stock 000300 CNN Train Log

Appendix 2 Part Of Code

```
import os
import math
import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import keras.backend as bk
import tensorflow as tf
import warnings
from datetime import datetime
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import BayesianRidge
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from keras.models import Sequential
from keras import optimizers
from keras.callbacks import LearningRateScheduler
from keras.layers import *
from sklearn.metrics import accuracy_score
# Predict future pre_window days price
pre_window = 5
# Stock Code
sk_code = '600660.XSHG'
# Get data start date
start_date = '2013-1-1'
# Get data end date
end_date = '2022-12-31'
# Training set ratio
train_ratio = 0.5
# Window time
prepare_window = 30
cnn_batch_size = 67
cnn_epoch = 70
cnn_learning_rate = 0.006
cnn_act_fun = 'relu'
cnn_ken_size = [2, 2]
cnn_str = [2, 2]
filter_num = 64
dropout_prob = 0.5
mp_size = [2, 2]
mp_str = [1, 1]
```

```
padding_p = 'same'
lstm_learning_rate = 0.004
lstm_epoch = 70
lstm_batch_size = 32
lstm_unit = 256
lstm_dropout = 0.3
# Import Data
dates = data[['Date']]
data = data[['Open', 'High', 'Low', 'Close']]
df = np.array(data)
dates = np.array(dates)
print(dates)
print(type(dates))
print(df.shape)
# Data normalization and construction
ts = math.floor(len(df)*train_ratio)
dates = dates[ts + prepare_window + pre_window - 1:]
date = [datetime.strptime(d, '%Y-%m-%d').date() for d in datest]
sx = MinMaxScaler(feature_range=(0, 1))
sy = MinMaxScaler(feature_range=(0, 1))
x_train_sca = sx.fit_transform(x_train)
x_test_sca = sx.fit_transform(x_test)
y_train_sca = sy.fit_transform(np.array(y_train).reshape(-1, 1))
y_test_sca = sy.fit_transform(np.array(y_test).reshape(-1, 1))
x = []
y = []
xt = []
yt = []
yt_a = []
# Construct the training set according to the 30-day prediction method
for i in range(len(x_train_sca) - prepare_window - pre_window + 1):
x.append(x_train_sca[i:i + prepare_window])
y.append(y_train_sca[i + prepare_window + pre_window - 1][-1])
x = np.array(x)
y = np.array(y)
x = np.reshape(x, (x.shape[0], x.shape[1], x.shape[2]))
for i in range(len(x_test_sca) - prepare_window - pre_window + 1):
xt.append(x_test_sca[i:i + prepare_window])
yt.append(y_test_sca[i + prepare_window + pre_window - 1][-1])
xt = np.array(xt)
print(xt.shape)
yt = np.array(yt)
print(yt.shape)
xt = np.reshape(xt, (xt.shape[0], xt.shape[1], xt.shape[2]))
#Construct convolutional neural network model, train and validate
model = Sequential()
model.add(Reshape((x.shape[1], x.shape[2], 1)))
model.add(Conv2D(kernel_size=cnn_ken_size, strides=cnn_str,
filters=filter_num, activation=cnn_act_fun, padding='same'))
model.add(Conv2D(kernel_size=cnn_ken_size, strides=cnn_str,
filters=filter_num, activation=cnn_act_fun, padding='same'))
model.add(MaxPooling2D(pool_size=mp_size, strides=mp_str, padding=padding_p))
```

```
model.add(Conv2D(kernel_size=cnn_ken_size, strides=cnn_str,
filters=filter_num,
activation=cnn_act_fun, padding='same'))
model.add(MaxPooling2D(pool_size=mp_size, strides=mp_str, padding=padding_p))
model.add(Flatten())
model.add(Reshape((-1, 1)))
model.add(LSTM(128))
model.add(Dropout(dropout_prob))
model.add(Dense(1))
opt = tf.keras.optimizers.Adam(learning_rate=cnn_learning_rate)
model.compile(optimizer=opt, loss='mean_squared_error')
history = model.fit(x, y, batch_size=cnn_batch_size, epochs=cnn_epoch,
validation_data=(xt, yt), shuffle=True, callbacks=[reduce_lr])
predictions_c = model.predict(xt
# Construct long and short term memory network model, training and validation
model1 = Sequential()
model1.add(LSTM(lstm_unit, activation='tanh', return_sequences=False))
model1.add(Dropout(lstm_dropout))
model1.add(Dense(1))
opt_l = tf.keras.optimizers.Adam(learning_rate=lstm_learning_rate)
model1.compile(optimizer=opt_l, loss='mean_squared_error')
model1.fit(x, y, batch_size=lstm_batch_size, epochs=lstm_epoch,
validation_data=(xt, yt),       shuffle=True)
predictions_l = model1.predict(xt)
# Construct random forest model, train and validate
x = x.reshape(x.shape[0], x.shape[1]*x.shape[2])
xt = xt.reshape(xt.shape[0], xt.shape[1]*xt.shape[2])
model2 = RandomForestRegressor(n_estimators=100000, min_samples_split=2,
max_depth=3, min_samples_leaf=2,max_features='auto')
model2.fit(x, y)
predictions_r = model2.predict(xt)
# Construct Bayesian ridge regression network model, training and validation
predictions_r = np.array(predictions_r)
predictions_r = np.expand_dims(predictions_r, axis=-1)
yt = np.array(yt)
xb = []
xb.append(predictions_c)
xb.append(predictions_l)
xb.append(predictions_r)
xb = np.array(xb)
xb = xb.swapaxes(0, 1)
xb = xb.reshape(xb.shape[0], xb.shape[1]*xb.shape[2])
model3 = BayesianRidge()
model3.fit(xb, yt)
predictions_b = model3.predict(xb)
# Plot  Price Predictions
yt = sy.inverse_transform(yt.reshape(-1,1))
predictions_c = sy.inverse_transform(predictions_c.reshape(-1, 1))
predictions_l = sy.inverse_transform(predictions_l.reshape(-1, 1))
predictions_r = sy.inverse_transform(predictions_r.reshape(-1, 1))
predictions_b = sy.inverse_transform(predictions_b.reshape(-1, 1))
plt.figure(figsize=(20, 10))
```

```
148 plt.plot(date, predictions_c, label='CNN Price Prediction')
149 plt.plot(date, predictions_l, label='LSTM Price Prediction')
150 plt.plot(date, predictions_r, label='RF Price Prediction')
151 plt.plot(date, predictions_b, label='BRR Price Prediction')
152 plt.plot(date, yt, label='Real Price')
153 plt.title('Stock 000300 Price Prediction')
154 plt.xlabel('Date')
155 plt.ylabel('Close Price')
156 plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
157 plt.gca().xaxis.set_major_locator(mdates.MonthLocator())
158 ds = math.floor(len(date)/9)
159 plt.xticks(date[::ds])
160 plt.legend()
161 plt.savefig('./'+ str(600109) + '.png')
162 plt.show()
```