Yun-Cheng Tsai*, Jun-Hao Chen and Jun-Jie Wang

Predict Forex Trend via Convolutional Neural Networks

https://doi.org/10.1515/jisys-2018-0074
Received January 31, 2018.

Abstract: Deep learning is an effective approach to solving image recognition problems. People draw intuitive conclusions from trading charts. This study uses the characteristics of deep learning to train computers in imitating this kind of intuition in the context of trading charts. The main goal of our approach is combining the time-series modeling and convolutional neural networks (CNNs) to build a trading model. We propose three steps to build the trading model. First, we preprocess the input data from quantitative data to images. Second, we use a CNN, which is a type of deep learning, to train our trading model. Third, we evaluate the model's performance in terms of the accuracy of classification. The experimental results show that if the strat egy is clear enough to make the images obviously distinguishable the CNN model can predict the prices of a financial asset. Hence, our approach can help devise trading strategies and help clients automatically obtain personalized trading strategies.

Keywords: Deep learning, convolutional neural network (CNN), geometric Brownian motion (GBM), Forex (FX), trading strategies.

1 Introduction

Human beings are visual animals. The eyes are the most compact structure of all the sensory organs, and the visual intelligence of the human brain is rich in content. Exercise, behavior, and thinking activities use visual sensory data as their greatest source of information. The more flexible and talented we become, the more we rely on visual intelligence. What general business and decision-makers desire after analysis is not the data itself but the value. Therefore, it is important that data analyses be intuitive; in this way, the visual ization of financial data can be more easily accepted: they can "see the story" and thus interpret the data more easily.

Although visualization analysis can benefit decision-makers, many traditional statistical or machine learning methods for predicting currency movements use quantitative models. These methods do not con sider visualization. We attempt to make good use of the advantages of visualization and comprehensively enhance the efficiency of intelligence analysis. For example, most traders use charts to analyze and predict currency movement trends, which carry obvious economic benefits. However, in this visualization, analysis is artificial. We intend to teach machines to achieve visualization like a human brain. We then hope to use the machine to visually analyze huge financial data.

Convolutional neural networks (CNNs) are widely used in pattern and image recognition problems. In these applications, the best possible correction detection rates (CDRs) have been achieved using CNNs. For example, CNNs have achieved a CDR of 99.77% using the modified National Institute of Standards and Technology database of handwritten digits, a CDR of 97.47% with the New York University Object Recognition Benchmark data set of 3D objects, and a CDR of 97.6% on more than 5600 images of more than 10 objects. CNNs not only give the best performance compared to other detection algorithms but also outperform humans in

***Corresponding author: Yun-Cheng Tsai,** Center for General Education, National Taiwan University, Taipei, Taiwan, e-mail: pecutsai@ntu.edu.tw **Jun-Hao Chen:** Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan **Jun-Jie Wang:** Department of Computer Science and Information Engineering, National Taipei University, New Taipei City, Taiwan

cases such as classifying objects into fine-grained categories such as the particular breed of dogs or species of birds. The two main reasons for choosing a CNN model to predict currency movements are as follows: 1. CNN models are good at detecting patterns in images such as lines. We expect that this property can also be used to detect the trend of trading charts.

2. CNNs can detect relationships among images that humans cannot find easily. The structure of neural networks can help detect complicated relationships among features.

CNN is a graph-based model that is different from quantitative models. People do not need to consider all possible features that affect currency movements using quantitative models alone.

Compared to a quantitative model, a CNN model contains many filters that are similar to the eyes of a human being and can extract the features of images. In general, the literature on financial time-series fore casting relevant to CNN models is still scarce because CNN models are much more commonly applied in classi fication problems. Recently published articles relevant to CNN models for time-series forecasting. Mittelman proposed an undecimated convolutional network for time-series modeling based on the undecimated wavelet transform [7]. Binkowski et al. proposed to use an autoregressive-type weighting system for forecasting finan cial time-series, where the weights are allowed to be data dependent by learning them through a CNN model [1]. Borovykh et al. presented a method for conditional time-series forecasting based on an adaptation of the CNN architecture [2].

As the convolution layer goes deeper, a CNN model can also extract more detailed features from the image, just like human visualization. Predicting currency movement trends is a time-series problem. Many people look for the Holy Grail of prediction, which in fact does not exist. We cannot predict the future in the real world; however, we can define the small world to evaluate our prediction approach. To realize the idea, we use a geometric Brownian motion (GBM) to model the currency movements. We believe that these prices follow, at least approximately, as a subset of real-world rules that we can derive from the historical data and our knowledge of prices.

Eun and Shim investigated the international transmission mechanism of stock market movement using the forecast errors of the nine-market vector autoregression model [5]. They decomposed the stock market returns into 5-, 10-, and 20-day ahead forecasts and analyzed the correlation of nine international stock mar kets. In our study, we chose to use the same time periods in our moving average feature extracting method. Although the model is different, the concept of forecasting current daily data by the past returns is the same. Hence, the three steps involved are as follows:

1. Before training, preprocess the training data from quantitative data to images. Our input images include price, moving average 5 (MA5), moving average 10 (MA10), and moving average 20 information (MA20).
2. Use a CNN to train our trading models.
3. Evaluate the models in terms of the accuracy of classification.

When we control our small world, we use the CNN model to classify the weekly currency movements by sep arating price series into three groups: rising trend, downtrend, and nonmovement groups. The remainder of this paper is organized as follows. A review of the literature is given in the next section. In Section 3, we present our methodology. Then, a description of the empirical data employed in our study is provided in Section 4. Section 5 presents the conclusion of our study.

2 Preliminary

We used a graph-based model to train a predictive model rather than common quantitative methods such as recurrent neural networks (RNNs). In other words, we wanted to model the thoughts of people rather than the rule-based decisions, which can be clearly stated by the people. Research on using CNNs to predict financial asset prices is limited; most researchers prefer the quantitative-based models. However, there are still some researchers attempting to study it.

Di Persio and Honchar [4] tried to compare different artificial neural network approaches to predict stock market indices in classification-based models. They compared three common neural network models, namely, multilayer perceptron, CNN, and long short-term memory (LSTM). They found that a novel architec ture based on a combination of wavelets and CNNs reaches an 83% accuracy rate on foreign exchange rates, outperforming the RNNs by 4%. Distinct from our work, Di Persio and Honchar [4] designed their CNN archi tecture using a 1D convolution layer and a 1D pooling layer. The 1D convolution layer considers only the price data, which means that this convolution layer still captures the quantitative information.

Similar to our work, Ashwin Siripurapu used convolution networks to predict movements in stock prices with a series of time-series pixel images. The input images to the model are the graphs of high and low prices for a 30-min window of time. The input graphs to the model are saved in an RGB color space to highlight the different lines of the stock prices. Siripurapu used three kinds of input images. For the first input, he used only the high and low prices. For the second one, he added the volume data together with the high and low prices. For the third one, he used the correlation feature representation of the top 10 com panies share of the Standard & Poor's 500 index basket. In the experiment, Siripurapu used two different architectures of conventional networks, which are called full and reduced models. The full model had five pairs of convolution-ReLU-pooling layers and was further connected to a fully connected layer. The reduced model reduced the pooling layers in the first two pairs. Although the performance does not exceed 0 for an out-of-sample R^2 People like to think intuitively when viewing trading charts; many of them cannot clearly explain how to make their decisions and how machines better perform. We focus on this and try using pixel images as it still gives us many ideas for using pixel images as the input data to a CNN model.

inputs to enable the computer to refine the features from it. However, beyond learning, we want to teach the computer to simulate and thus predict the behavior of people as they trade on the trading charts; that is, make a model that can learn the trading strategies of the people. Define S_t as the price of the financial asset at time t. The risk-neutralized version of stock price's log-normal diffusion process is

$$dS_t = r\, S_t\, dt + ÿ\, S_t\, dW_t, \tag{1}$$

where r is the risk-free rate, $ÿ$ is the constant volatility price process of the financial asset, and the random variable W_t is a standard Brownian motion [3]. S_t is said to follow a GBM process because it satisfies the above stochastic differential equation. For an initial value S_0, Eq. (1) has the analytic solution:

$$S_t = S_0 \exp(ÿ\, r\, ÿ\, \frac{2p}{2})ÿ\, t + ÿ\, W_t.$$

From Eq. (1), it has the following discrete solution [8]:

$$X_t = X_{tÿ1} + (ÿ\, r\, ÿ\, \frac{2p}{2})ÿ\, ÿt + ÿ\, ÿ\, \overline{ÿt}\, B_t, \tag{2}$$

where X_t ÿ $ln(S_t)$ is the log price, $ÿt$ ÿ T/n is the length of a time step in the time interval [0, T] divided into n subintervals, B_t ÿ N(0, 1) is i.i.d. normal random variable, and $ÿ$ is the annualized constant volatility.

The CNN is one of the best graph-based models in recent years. Many new architectures of CNNs con stantly appeared very fast, but the most original architecture was proposed by K. Fukushima in 1980. Fukushima proposed a model called "neocognitron", which is generally seen as the model that inspires the CNN on the computation side [6]. Neocognitron is a neural network designed to simulate the human visual cortex. It consists of two types of layers called the feature extractor layers and the structured connection lay ers. The feature extractor layers, also called S-layers, simulate the cell in the primary visual cortex, helping human beings to perform feature extraction. The structured connection layers, also called C-layers, simulate the complex cell in the higher pathway of the visual cortex, providing the model with its shifted invariant property.

Inspired by the neocognitron and the concept of back propagation, the most generally classic modern CNN, LeNet, was proposed by LeCun et al. in 1990. The potential of the modern convolution architecture can be seen in LeNet (1990), consisting of a convolution layer, a subsampling layer, and a full connection layer [9]. As the concept of rectified linear unit (ReLU) and dropout was presented in recent years, a new convolution based model (AlexNet), proposed by Hinton and Alex Krizhevsky, appeared and beat the previous champion of the ImageNet Challenge, with more than 15 million labeled high-resolution images and roughly 22,000 categories. There are three main differences between LeNet and AlexNet as follows:

1. The ReLU is used as the activation function in AlexNet. It introduces a nonlinearity transform after convolution, which helps the computer to simulate human vision more accurately. The ReLU is also a nonsaturating activation function and is several times faster than tanh and sigmoid units in computation.
2. A new regularization technique called dropout was introduced to AlexNet to avoid overfitting with much less computation. The dropout technique randomly drops some neurons with a particular probability, and the dropped neurons are not involved in forward and backward computations.
3. Owing to the technological progress in recent years, AlexNet was supported by a more efficient GPU than LeNet (1980). This means that a larger data set and more epochs can be trailed in the training process.

With the success of AlexNet, many researchers have been motivated to participate in this kind of research, inventing architectures with deeper structures and modified convolution such as VGG and GoogleNet. These developments continually improve CNNs in the field of computer vision.

The two most important components of CNNs are the convolution layer and the pooling layer. The convo lution layer implements the convolution operation (see Figure 1), which extracts image features by computing the inner product of an input image matrix and a kernel matrix; the number of channels of the input image and kernel matrix must be the same. For example, if the input image is an RGB color space, then the depth of the kernel matrix must be three; otherwise, the kernel matrix cannot capture the information between different color spaces. Another important component is the pooling layer (see Figure 2), also called the sub sampling layer, which is mainly in charge of simpler tasks. The pooling layer will only retain part of the data after the convolution layer, which reduces the number of large features extracted by the convolution layer and makes the retained features more refined.

Only with these two components can the convolution model be used to imitate human vision. In practi cal applications, the CNN model usually combines the convolution layer and the pooling layer together. This

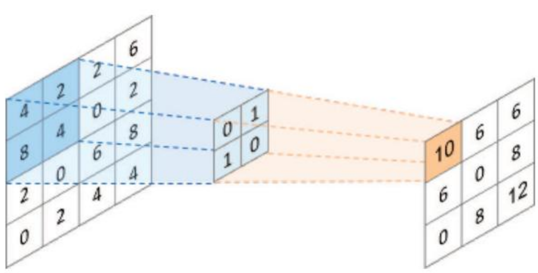

Figure 1: Convolution Operation.

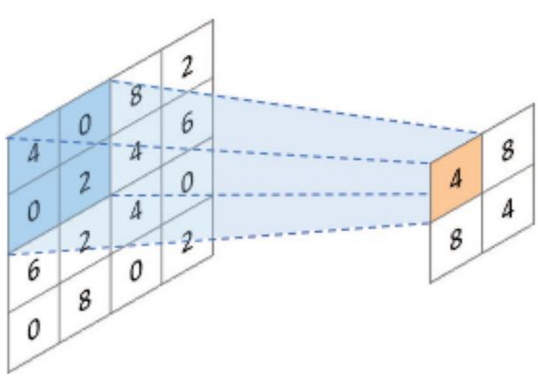

Figure 2: Pooling Operation.

is because the convolution layer often extracts a great number of features, and most of the features may be noise, which could lead to model learning in the wrong direction. This is the so-called model overfitting prob lem. Furthermore, the fully connected layers are usually connected at the end of the sequence. The function of the fully connected layer is to organize the extracted features, which were processed by the convolution and pooling layers. The correlation between the extracted features is learned in this layer.

Although the pooling layer can reduce the occurrence of overfitting after convolution, it is inappropri ate to use after the fully connected layer. Another widely known regularization technique called dropout is designed to solve this issue. The dropout technique randomly drops some neurons with a specific probability, and the dropped neurons are not involved in forward and backward computations. This idea directly limits the model's learning; the model can only update its parameters subject to the remaining neurons in each epoch.

Next, we introduce how to generate the data and how to design the architecture in the first workflow. The input data that we provide the computer with is a pixel image drawing from time I to I + N, where index I represents the beginning of each image and index N represents the total length of the historical data we want the computer to see. After the first image is generated, the beginning of the time sequence will advance and keep generating the new images until a specific number of images has been created, meaning that the time will move from I and $I + N$ to time $I + 1$ and $I + N + 1$ and proceed as thus until M images have been collected. Then, because we assume that increasing and decreasing patterns exist in the foreign exchange, we label the images through time $I + N + 1$, which is out of the time region of each generated image. Figure 3 depicts the process of generating and labeling data in detail.

After the data are collected, we supervise the model as it learns how to classify the images into three categories: buy, sell, and not taking any action. We expect the model to predict what kind of images will rise or fall in the future; in other words, learning the data from time I to $I + N$ and predicting the outcomes at time $I + N + 1$. Different from the typical image recognition problems with the CNN model, applications in finance need to make some modifications. Financial data have time-series characteristics, which cannot be captured by the convolution model. For this reason, our first workflow combines the concept of moving windows into the CNN model (see Figure 4).

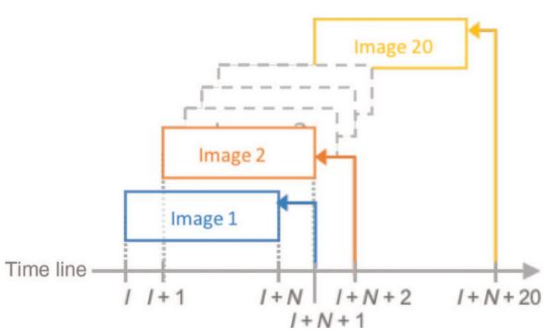

Figure 3: Process of Creating and Labeling Data in Workflow 1.

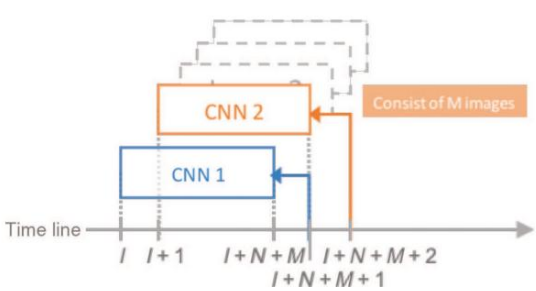

Figure 4: Process of Combining the Moving Windows into the Convolution Model in Workflow 1.

To consider the time-series properties of the financial data, the single CNN model needs to be modified. It is intuitive to think of training the new CNN model in different time regions; in more detail, we use day *I* to *I* + *N* + 20 to generate data and train a convolution model. After the first run, we move to the next time window and train a new convolution model. This process continues to run until all the predictions have been made. There are two main advantages of this process: the different CNN models can capture different features in the particular time interval and this also prevents the CNN models from using noisy features from a long time ago.

For example, we may use days 1–20 to make the data and labels and then train a CNN model to predict the outcome on day 21. In the second run, we use days 2–21 to generate the new images and labels and train a new convolution model again to predict the outcome on day 22 and so forth. In terms of the architecture of the convolution model, we first intend to try some simpler models, which only consist of two or three pairs of convolution and pooling layers before using the famous AlexNet model. This is because the images we want the computer to learn are simple sets of one to four closed price line plots, including high, low, and the moving average. They are not as complex as the ImageNet Challenge. All the architectures we used are shown in Figure 5, where Conv, Pool, and FC are the convolutional layer, pooling layer, and fully connected layer, respectively.

1. Architecture 1: Conv + Conv + Pool + FC 2.
Architecture 2: (Conv + Pool) * 2 + FC 3. Architecture 3:
Conv * 4 + Pool + Conv + Pool + FC

In the first architecture, we used two convolution layers, further connected to a pooling layer and a fully connected layer. In the second architecture, we used two pairs of convolution and pooling layers and a fully connected layer, which is similar to the architecture of LeNet (see Figure 6). We expect that these two simple architectures can enable the computer to learn the simple structure from the input images. In the third archi tecture, we designed a deeper architecture consisting of more convolution layers. We used this architecture because we tried to solve the underfitting problem from the model; simple architecture was not sufficient to learn features from input images.

The results of these experiments do not fit the expectation; whether simple or complex, the architectures do not fit the convolution model well. The experimental procedures are illustrated in detail in Section 4.
Another architecture that is widely used in our second workflow is the well-known AlexNet model. The AlexNet model appeared in 2012, beat the previous champion, and became the state-of-the-art model in the ImageNet Challenge, which has more than 15 million labeled high-resolution images and roughly 22,000 cat egories. The AlexNet model has a deeper structure than LeNet, containing five convolutional layers, three fully connected layers, and a softmax layer. To prevent the model from overfitting, the AlexNet model also uses a new regularization method called dropout and data augmentation, which horizontally flips the image or performs random cropping. The AlexNet model also uses the ReLU as the activation function, which is

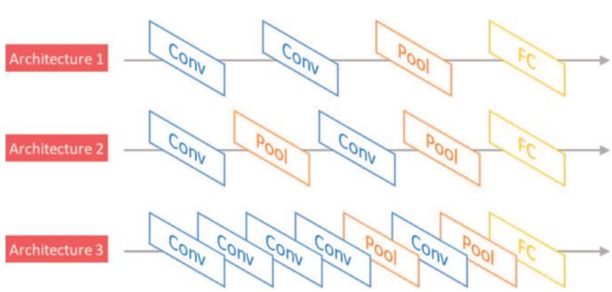

Figure 5: Three Architectures in Workflow 1.

Figure 6: Classic LeNet Model.

a nonsaturating activation function and is several times faster than tanh and sigmoid units. With these improvements and excellent GPU support, the AlexNet model has become one of the most powerful models today.

3 Methodology

In this section, we introduce the architectures we used in our experiments and justify our decision for using these workflows. We also illustrate some data preprocessing techniques used to generate our inputs. The deep learning frameworks used in each workflow are the Python Keras module and NVIDIA DIGITS with the Caffe back-end. All the convolution models in both workflows were trained for 30 epochs and were speeded up by the GTX TITAN GPU. We also tried to observe the result of different epochs, even up to 4000 epochs, but the overfitting almost significantly occurs at about 50–100 epochs. The workflows are as follows.

3.1 Workflow 1

In the first workflow, we used the real-world exchange rates of Japanese yen from 2010 to 2011. We designed three kinds of convolution architectures and expected one of these architectures to fit the real-world data well. The overview of the raw data is shown in Figure 7, and the first workflow is enumerated in detail as follows:

1. Transform the quantitative price data to image data using the Python Matplotlib module and create classification-based labels that consist of buy, sell, and not taking any action.
2. Create the three architectures of the CNN model using the Python Keras deep learning module. Each of the architectures will be experimented independently.
3. Train the CNN model and tweak the parameters to maximize accuracy. The number of epochs used for training ranges from 30 to 100.
4. Evaluate the model with a confusion matrix for currency performance.
5. Repeat the above steps until the best model is found.

3.2 Workflow 2

Because the performance of workflow 1 was not as good as expected, we switched to using simulation data from the GBM. We simulated 90 days of foreign exchange rate data repeatedly, for 100 times, with a 1% yearly return and 25% yearly standard errors. We believed that these prices approximately followed a subset of the

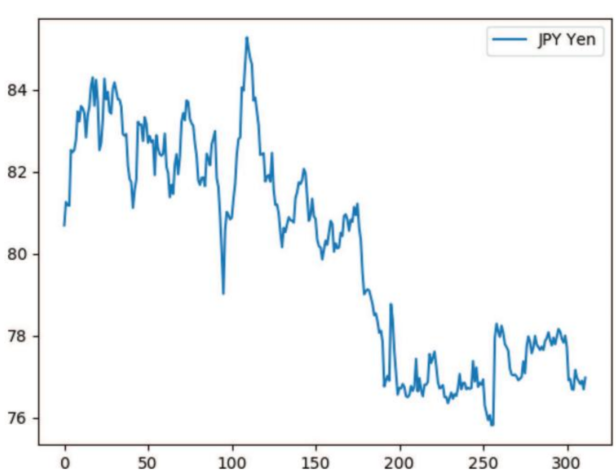

Figure 7: Exchange Rates of Japanese Yen from November 9, 2010 to January 13, 2011.

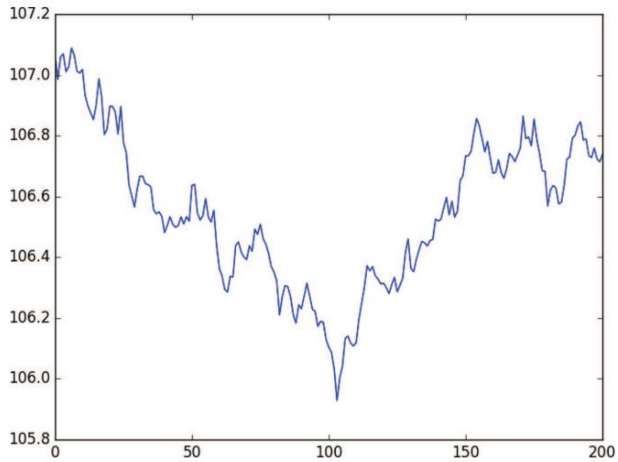

Figure 8: One of the Simulation Data Generated by the GBM Process.

real-world data; therefore, we expected the new architecture to fit well in the subset of the real world. One of the simulated data is shown in Figure 8, and the second workflow is enumerated in detail as follows: 1.

Transform the quantitative price data to image data using the Python Matplotlib module and create classification-based labels that consist of buy, sell, and not taking any action.

2. Create the AlexNet architecture of the CNN model using NVIDIA DIGITS with the Caffe back-end. NVIDIA DIGITS is a lightweight tool especially good at presenting the training process in real time.
3. Train the AlexNet model and tweak the parameters to maximize accuracy. The number of epochs used for training is 50.
4. Evaluate the model with a confusion matrix for currency performance.
5. Repeat the above steps until the best model is found.

The structure of workflow 2 is almost the same as that of workflow 1. The main difference in workflow 2 is the way data are labeled; in workflow 1, the same strategy is used to label all inputs, but many kinds of strategies are used in workflow 2. In workflow 2, we used the AlexNet model with its default parameters and tweaked only the epochs and the different kinds of input images. The strategies we used in workflow 2 are listed below:

1. Use every 20-day period as an image and the following 5 days as holding days; that is, we use days 1–20 as the input image and use day 25 to label day 20. If the price on day 25 is greater than day 20 by at least 1%, then we will buy on day 20 and sell on day 25. If the price on day 25 is less than day 20 by at least 1%, then we will sell on day 20 and buy on day 25. Otherwise, no action will be taken.
2. In this case, we tried to use the moving average as our strategy. Because we wanted the inputs to be more distinguishable by the model, the rule we used was that if MA5 is greater than moving average 7 (MA7) by at least 1% and MA7 is greater than MA10 by at least 1% on day 9, then we will buy on day 6 and sell on day 9. If MA5 is less than MA7 by at least 1% and MA7 is less than MA10 by at least 1% on day 9, then we will sell on day 6 and buy on day 9. Otherwise, no action will be taken.
3. Furthermore, we also simulated both open and closed prices and plotted it with the MA5, MA10, and MA20 lines. We used every 15-day period as the image and the following 5 days as the holding period. The strategy used here is that if the opening price on day 20 is greater than the closing price on day 15 by at least 2%, then we will buy on day 15 and sell on day 20. If the opening price on day 20 is less than the closing price on day 15 by at least 1%, then we will sell on day 15 and buy on day 20.

4 Experimental Results

First, we introduce three ways to preprocess the image data; second, we discuss problems we encountered in the experimental procedure and illustrate how to solve them. The preprocess frameworks we used are the Python Matplotlib module and Python Pillow module. The following are the three ways in which we preprocess our images:

1. Crop the images without the information of the x-axis and y-axis. This is because we want our input data to be as clean as possible.
2. Use the RGB color space to capture the information of moving average lines. Different colors will be given to each moving average line, so the moving average lines will be represented in the different channels.
3. Invert the color space to highlight only the lines in the image. The background will become black, which means the value of each background pixel is zero.

The moving average lines we used are MA5, MA7, MA10, and MA20. We used moving average lines to simulate our inputs and increase similarity to the trading charts. We also rescaled the images to different sizes, for example, 100 × 150 or 300 × 400. We also tried to set the different y-axes in the same scale. The image of the moving average lines is shown in Figure 9. Black is the price line, red is the MA5 line, blue is the MA10 line, and green is the MA20 line. There are still many different permutations and combinations of the price and moving average lines. The inverted one is shown in Figure 10.

4.1 Workflow 1

In workflow 1, we tried three architectures. The default time region is 20; in each region, we used every 5-day period to create the image data and used the next day to label the input images. Each architecture used the framework of the moving windows and predicted 100 times. The three architectures we tried are as follows:

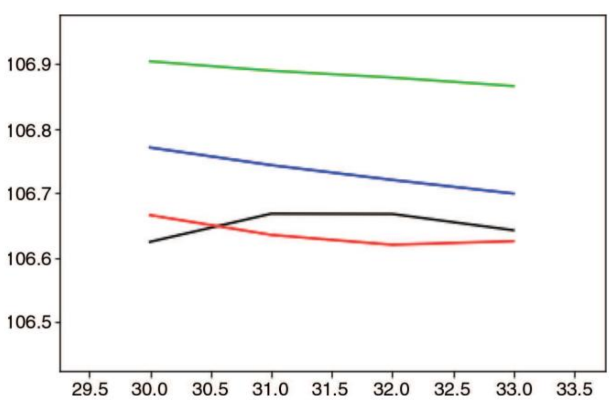

Figure 9: Image Data with Price and Moving Average Lines Without Resizing.

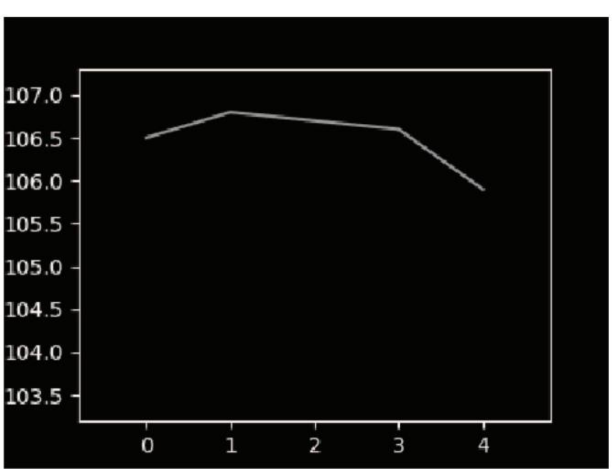

Figure 10: Image Data Only with Price Line Preprocessed with Inversion.

1. Architecture 1: Conv + Conv + Pool + FC
2. Architecture 2: (Conv + Pool) * 2 + FC 3.
Architecture 3: Conv * 4 + Pool + Conv + Pool + FC

We used the first two architectures (architectures 1 and 2) because we expected that a simple model could solve our problem; however, the results were not good. Therefore, we next used a deeper structure with archi tecture 3; we added more convolution layers and filters in the first two layers to help the model extract more detailed information. We hoped that a more complex architecture would help solve this problem. Unfortu nately, neither the simple nor the complex architecture worked well. The complex one did not improve the per formance of classification. The experimental results of each architecture are shown below. For architecture 1, we carried out three experiments. We inverted all input images and resized them to 100 × 150.

We used different parameters in each experiment as follows:
1. In the first experiment, we used a kernel size of 30 × 40, with 5 kernels and 128 fully connected units. The pooling layer we used was MaxPooling 2 × 2 and the time region used was 20, which means using 20-day historical information to predict the action for the next day.
2. In the second experiment, we used a kernel size of 30 × 40, with 10 kernels and 128 fully connected units. The pooling layer we used was MaxPooling 2 × 2 and the time region used was 20.
3. In the third experiment, we used a kernel size of 30 × 40, with 5 kernels and 128 fully connected units. The pooling layer we used was MaxPooling 2 × 2, but this time we used 30 days as our time region, which means using 30-day historical information to predict the action for the next day.

The results of the three experiments are described in Figures 11–13, respectively. There is no significant improvement between parameters; the model often predicts the action to be doing nothing.

The parameters used in the second experiment are the same as those used in the first experiment; only the architecture of the model is different. The performance of the second architecture is also poor, with the model once again giving the prediction of taking no action often. One result is shown in Figure 14.

True value \ Predict value	−1	0	1
−1	1	18	0
0	3	59	1
1	1	16	1

Kernel_size = 30 × 40
Kernel_num = 5
Maxpooling = 2 × 2
Region = 20

Figure 11: Confusion Matrix of Experiment 1 in Architecture 1.

True value \ Predict value	−1	0	1
−1	1	17	0
0	6	46	11
1	3	13	2

Kernel_size = 30 × 40
Kernel_num = 10
Maxpooling = 2 × 2
Region = 20

Figure 12: Confusion Matrix of Experiment 2 in Architecture 1.

True value \ Predict value	−1	0	1
−1	1	21	0
0	6	49	7
1	1	15	0

Kernel_size = 30 × 40
Kernel_num = 5
Maxpooling = 2 × 2
Region = 30

Figure 13: Confusion Matrix of Experiment 3 in Architecture 1.

True value \ Predict value	−1	0	1
−1	1	18	1
0	4	52	7
1	2	14	1

Kernel_size = 30 × 40
Kernel_num = 5
Maxpooling = 2 × 2
Region = 20

Figure 14: Confusion Matrix of Experiment 1 in Architecture 2.

We made some changes to the architectures because we obtained poor performance with the architec tures and experiments above: we added two more convolution layers and an additional pooling layer to make the model deeper and more complex. With the new, more complex architecture, we designed three kinds of experiments. The parameters of each experiment are almost the same; the only difference between the three experiments is the number of kernels. This is because we expected more filters would capture more features of the image. In experiments 1–3, the number of kernels is designated as 5, 10, and 20. The results of each experiment are described in Figures 15–17.

From the results of the three architectures, we can clearly see that none of the experiments yielded good performance. Additionally, each model is unstable due to overfitting. This is because the number of input images is too small to train the convolution model; if the time region is 20 and if we use every 5-day period to create an image, we only have 16 images of training data.

The convolution model can fit the given 16 images training data well but cannot recognize images with many differences to the training data. The only way to obtain more real-world training data is to extend the time region; in finance, however, older information does not help predict future data. Additional data would only increase the occurrence of noise, meaning we cannot simply extend the time region to collect more training data; an alternative approach is required.

True value \ Predict value	−1	0	1
−1	1	18	0
0	4	50	9
1	1	17	0

Kernel_size = 5 × 5
Kernel_num = 5
Maxpooling = 2 × 2
Region = 20

Figure 15: Confusion Matrix of Experiment 1 in Architecture 3.

True value \ Predict value	−1	0	1
−1	2	17	0
0	8	47	8
1	3	15	0

Kernel_size = 5 × 5
Kernel_num = 10
Maxpooling = 2 × 2
Region = 20

Figure 16: Confusion Matrix of Experiment 2 in Architecture 3.

True value \ Predict value	−1	0	1
−1	1	18	0
0	8	40	15
1	3	13	2

Kernel_size = 5 × 5
Kernel_num = 20
Maxpooling = 2 × 2
Region = 20

Figure 17: Confusion Matrix of Experiment 3 in Architecture 3.

4.2 Workflow 2

Before addressing the real-world data, we wanted to fit the model with the simulated data. This is because the simulated data can give us sufficient data with little noise. In addition, simulated data accurately represent a subset of the real-world data and therefore may be easier to fit. If we can fit the small world well, the convolu tion model can learn strategies from it. We used a mean of 1% and a standard error of 25% to simulate 90-day data; we simulated it many times to generate enough data for the convolution model. The three experiments, trained with the simulated data, are introduced in detail as follows.

In experiment 1, we used every 20-day period to create an image and the following 5 days as the holding days; that is, we may use days 1–20 as the input image and day 25 to label day 20. If the price on day 25 is larger than day 20 by at least 1%, then we will buy on day 20 and sell on day 25. If the price on day 25 is smaller than day 20 by at least 1%, then we will sell on day 20 and buy on day 25. Otherwise, no action will be taken. The images of the three different classes are shown in Figures 18–20. We can clearly see that each class cannot be easily distinguished by humans; this also makes it difficult for the convolution model to recognize the pattern of each class. In the training process of this case, which is shown in Figure 21, the loss of the training data and the validation data was not decreasing. The overfitting problem also occurred after the 100th epoch.

This time, the accuracy of the simple convolution model is better than the moving average one. The model predicts better in label 1 and ÿ1, but there are still many regions in which it could be improved. Figures 22 and 23 show the confusion matrix of the training and testing data.

Inspired by experiment 1, we tried to use the moving average as our strategy. Because we wanted the inputs to be more distinguishable by the model, the rule we used was that if MA5 is greater than MA7 by at least 1% and MA7 is greater than MA10 by at least 1% on day 9, then we will buy on day 6 and sell on day 9.

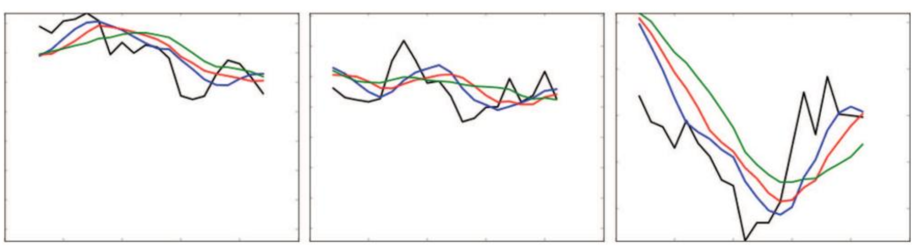

Figure 18: Experiment 1 with Label 1.

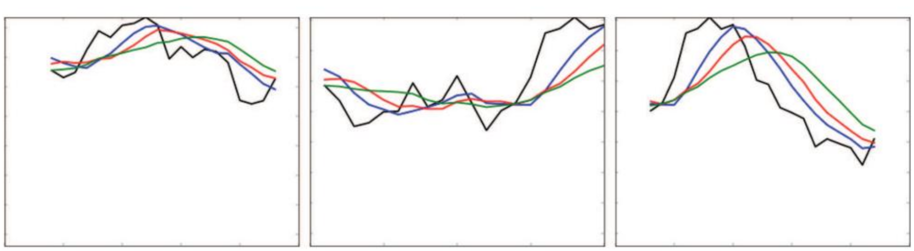

Figure 19: Experiment 1 with Label ÿ1.

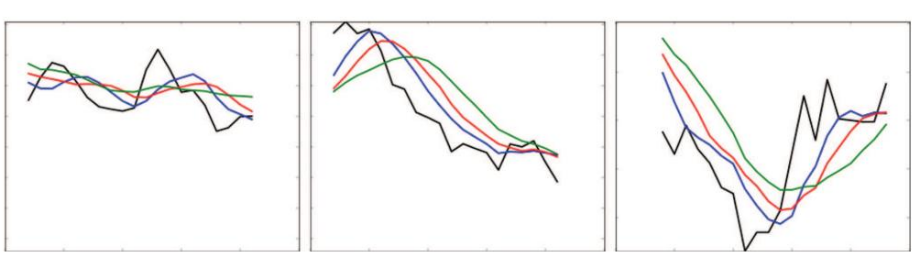

Figure 20: Experiment 1 with Label 0.

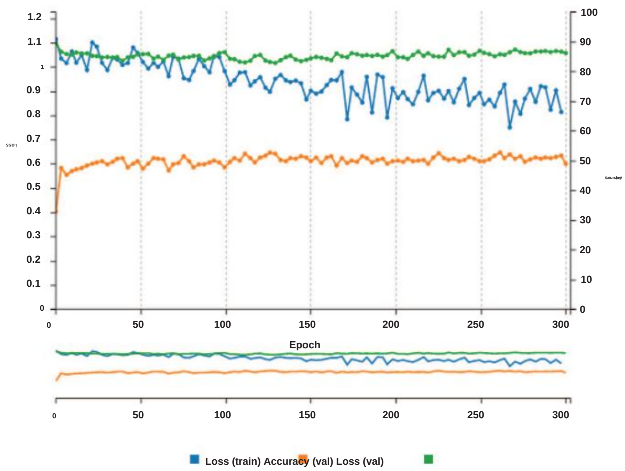

Figure 21: Training Process of Experiment 1.

	−1	0	1	Per-class accuracy
−1	272	24	50	78.61%
0	134	63	63	24.23%
1	81	50	173	56.91%

Figure 22: Training Confusion Matrix of Experiment 1.

	−1	0	1	Per-class accuracy
−1	136	17	19	79.07%
0	66	27	37	20.77%
1	54	17	81	53.29%

Figure 23: Testing Confusion Matrix of Experiment 1.

If MA5 is less than MA7 by at least 1% and MA7 is less than MA10 by at least 1% on day 9, then we will sell on day 6 and buy on day 9. Otherwise, no action will be taken.

The three kinds of labeled images are shown in Figures 24–26, and we can see that the pattern is more sig nificant in the buying (1) and selling (ÿ1) labels now. This makes it easier for the convolution model to detect the difference between the strategies. After the trials of experiment 3, we achieved an accuracy rate of 82%, which is a significant improvement over experiments 1 and 2. We also scaled the images to the maximum and minimum of the prices and the moving average; this yielded an 80% accuracy rate.

The training process is shown in Figure 27. In the experiment, we used 25% of the data for validation and 25% for testing. The accuracy rate increased to 82% in the 70th epoch. The problem of overfitting does not occur, which can be explained by the loss of the training data and the validation data. The confusion matrix of the training data and the testing data are shown in Figures 28 and 29. From the result, we can see that the

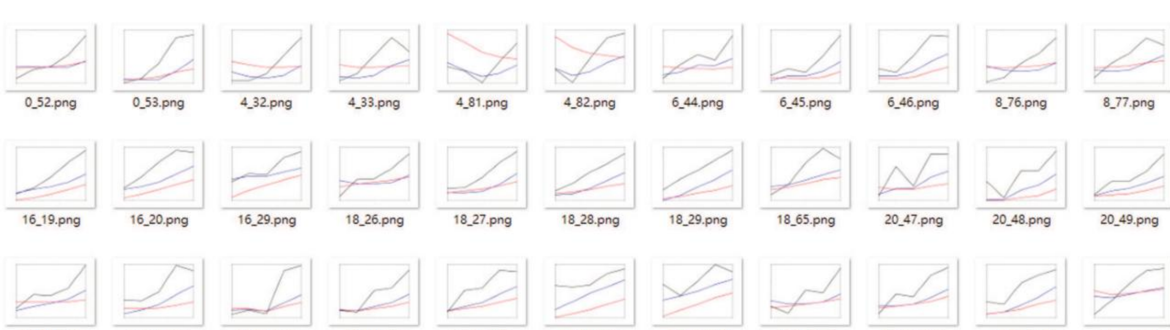

Figure 24: Experiment 2 with Label 1.

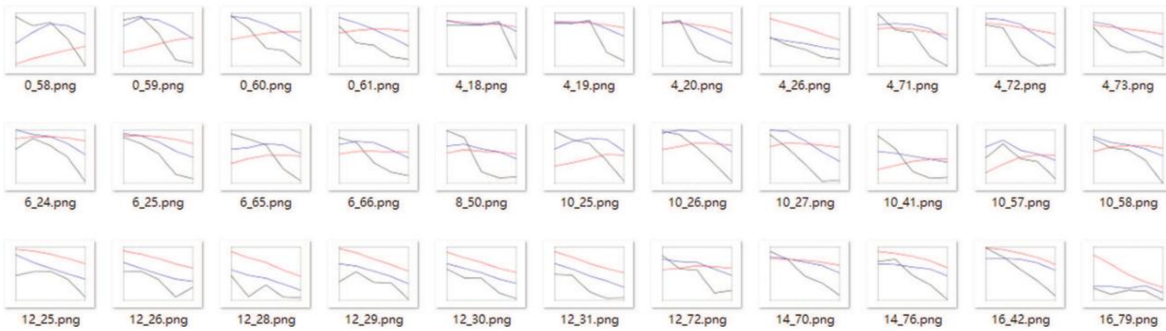

Figure 25: Experiment 2 with Label ÿ1.

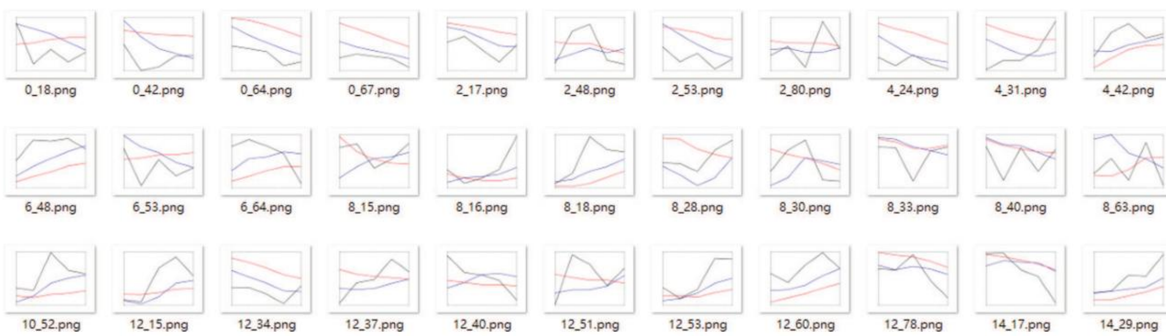

Figure 26: Experiment 2 with Label 0.

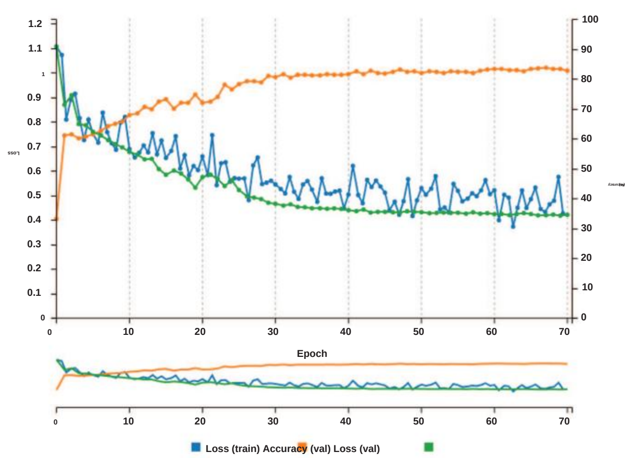

Figure 27: Training Process of Experiment 3.

	−1	0	1	Per-class accuracy
−1	347	19	0	94.81%
0	64	206	63	61.86%
1	0	20	286	93.46%

Figure 28: Training Confusion Matrix of Experiment 3.

	−1	0	1	Per-class accuracy
−1	169	14	0	92.35%
0	22	111	33	66.87%
1	0	17	136	88.89%

Figure 29: Testing Confusion Matrix of Experiment 3.

accuracy of each class is not significantly impacted by the overfitting problem. The accuracy of the testing data is only slightly lower than the training data.

The experimental results of the images scaled to the maximum and minimum of the prices and moving average are as follows. Figures 30–32 show the images classified by the moving average strategy. The results of this case achieved an accuracy rate of 82%, which is better compared to the earlier rate.

We also simulated both open and closed prices and plotted them with the MA5, MA10, and MA20 lines. We used every 15-day period to create an image and the following 5 days as the holding days. If the opening price on day 20 is greater than the closing price on day 15 by at least 2%, then we will buy on day 15 and sell

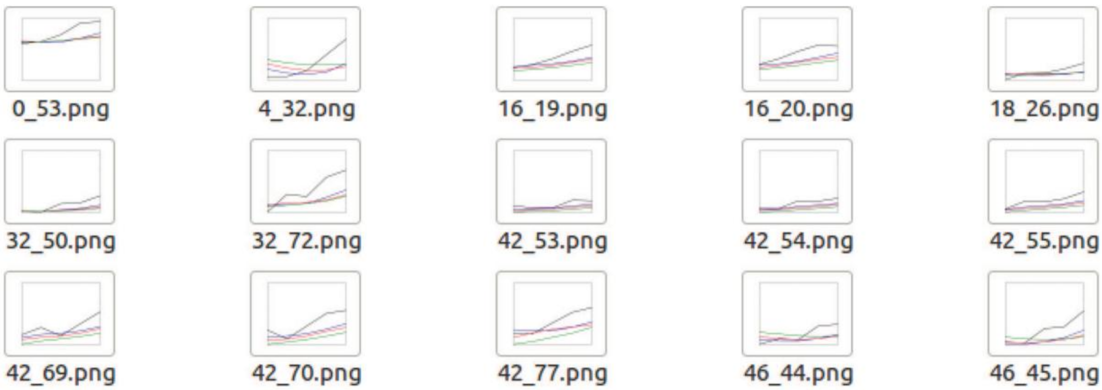

Figure 30: Experiment 2 with Scaling with Label 1.

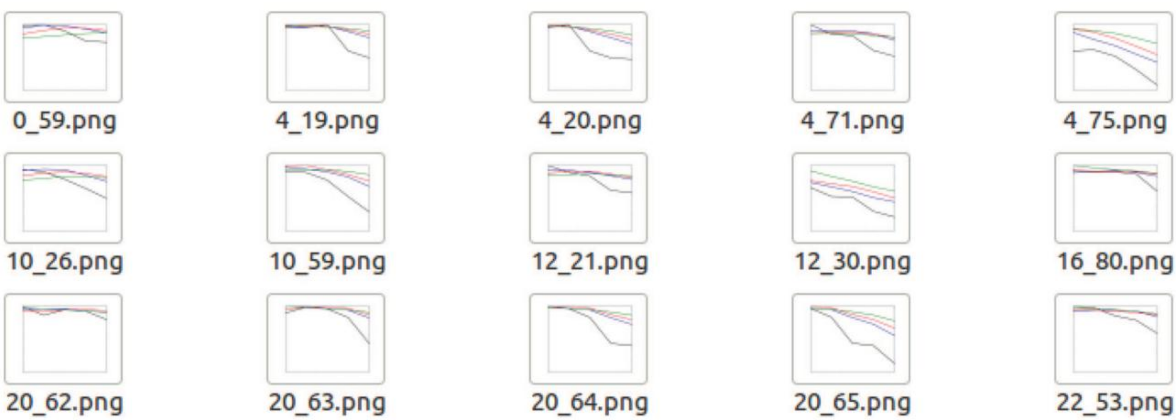

Figure 31: Experiment 2 with Scaling with Label ÿ1.

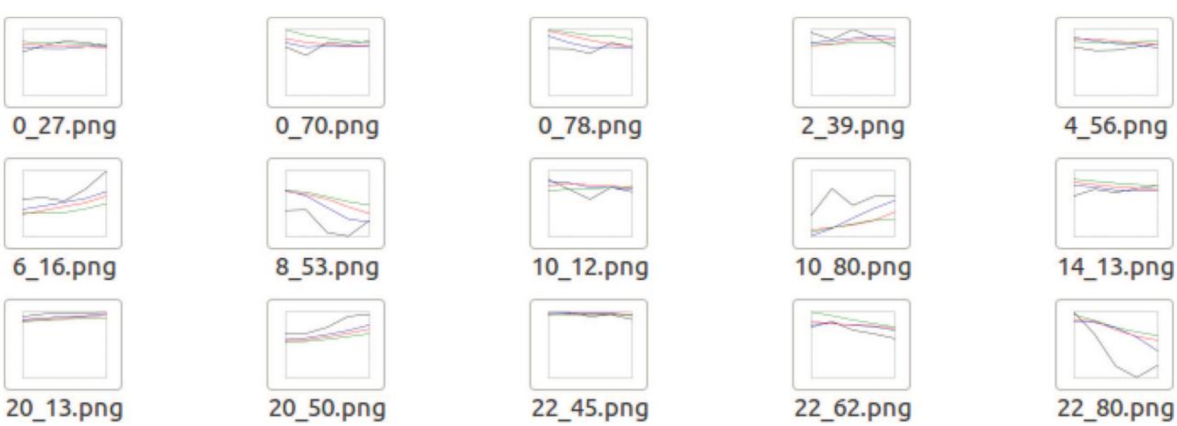

Figure 32: Experiment 2 with Scaling with Label 0.

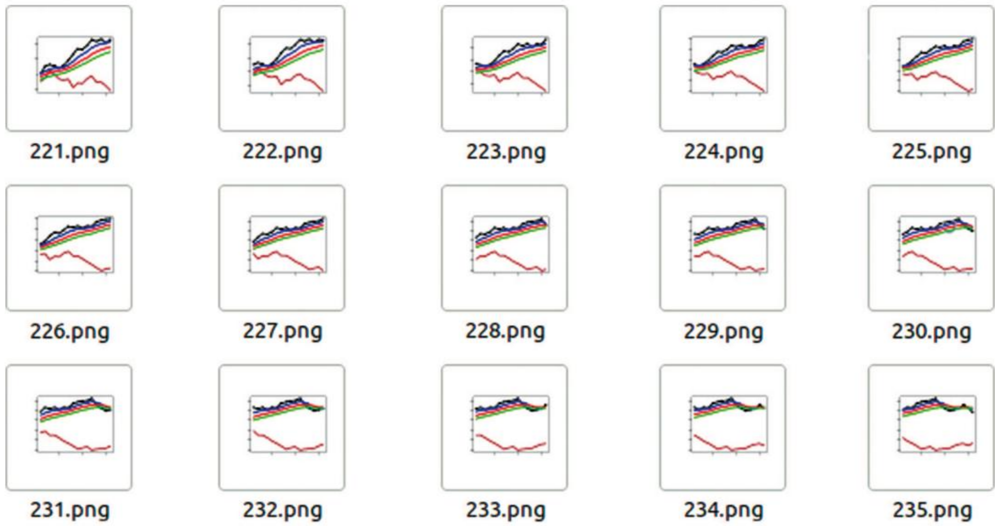

Figure 33: Experiment 3 using Open, Close, MA5, MA10, and MA20 with Label 1.

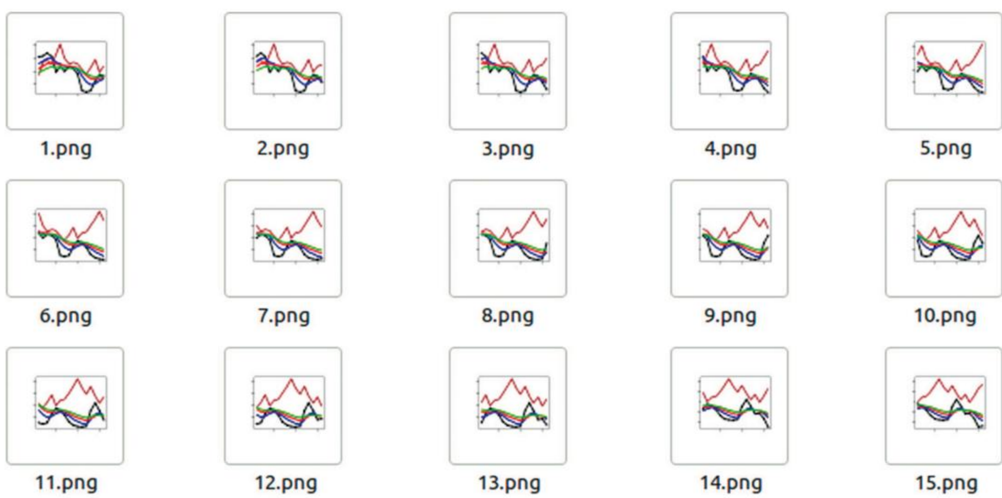

Figure 34: Experiment 3 using Open, Close, MA5, MA10, and MA20 with Label ÿ1.

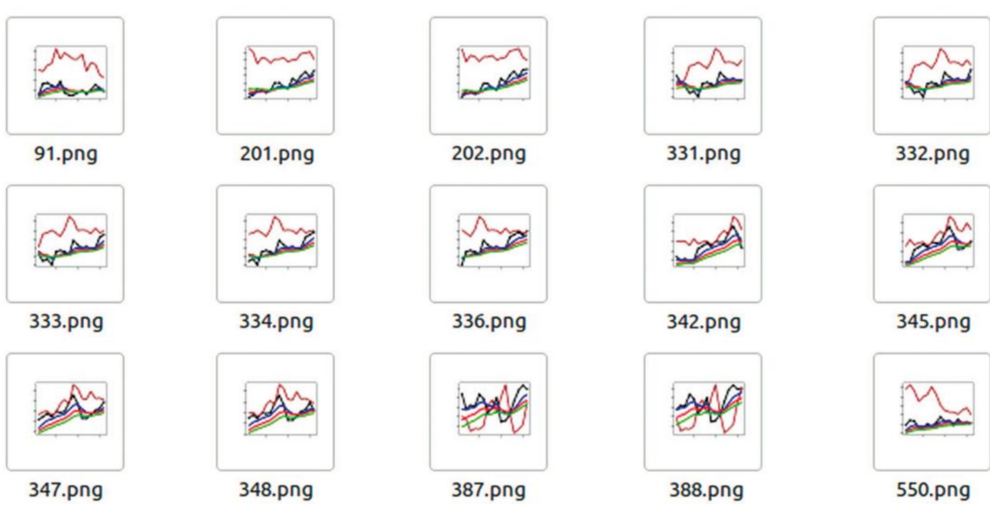

Figure 35: Experiment 3 using Open, Close, MA5, MA10, and MA20 with Label 0.

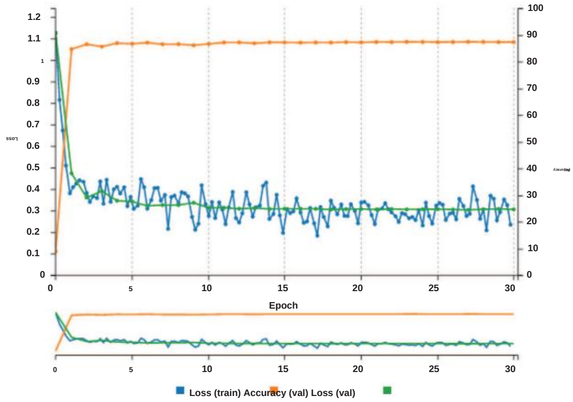

Figure 36: Training Process of Experiment 3.

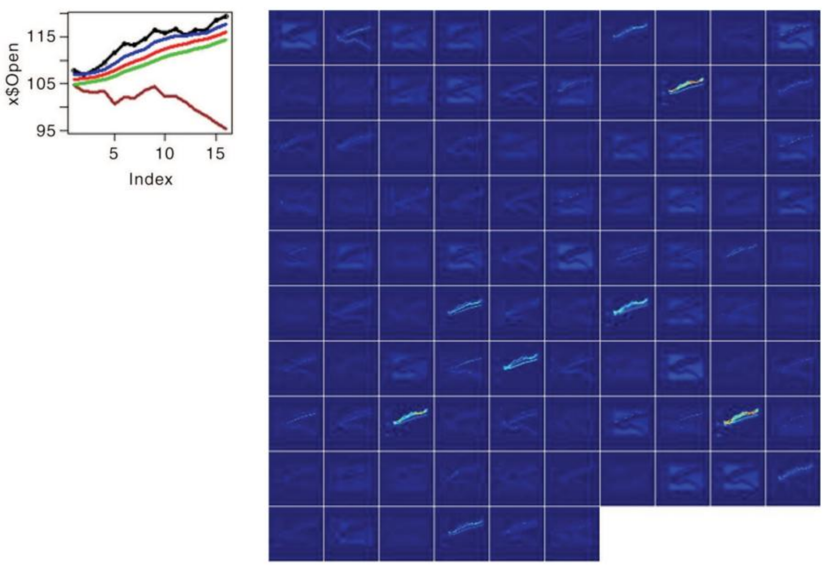

Figure 37: Visualization of the First Convolution Layer with the Demo Image.

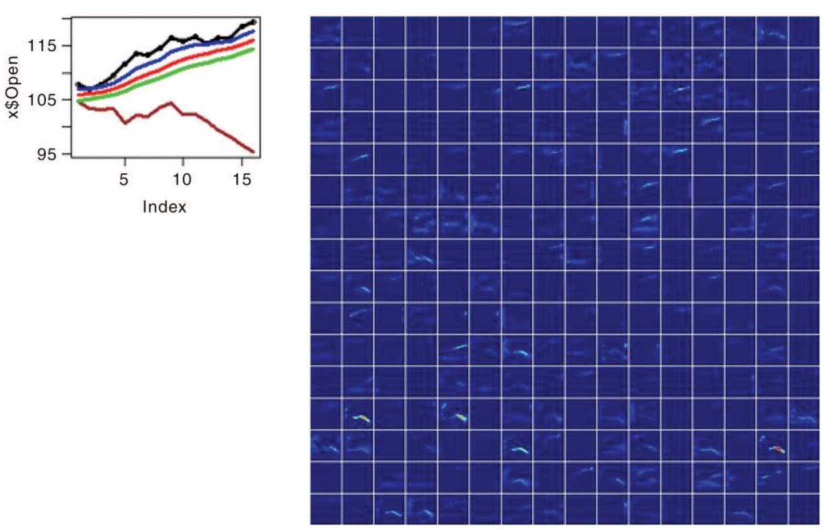

Figure 38: Visualization of the Second Convolution Layer with the Demo Image.

on day 20. If the opening price on day 20 is less than the closing price on day 15 by at least 1%, then we will sell on day 15 and buy on day 20. The three kinds of labeled images are shown in Figures 33–35.

In this case, the images are also distinguished by our strategy. We expected that the accuracy of the clas sification will be good; the results proved this. In Figure 36, the model obtained an accuracy rate of 87% in the 30th epoch, and the accuracy rate for each class was also better than that of experiment 2.

We also examine the visualization after the convolution layer. The outputs after the first two convolution layers with the demo image are shown in Figures 37 and 38; we can clearly see that the kernels in the first two layers can capture the shape of the lines. In this image, which is the buy action, the convolution model can clearly capture the pattern of the increasing trend.

5 Conclusions

In workflow 1, neither the simple nor the complex CNN architecture produced the expected performance. The main cause of this is the lack of data for each convolution model. To expand more training data to fit CNN mod els parameters, we may attempt to use older historical data. However, the older historical data would only introduce additional noise and further mislead the convolution model. Therefore, we narrowed the scope of our research to fit in the simulated world, which is generated by applying the GBM calibrated from the real-world data.

In workflow 2, the main difference between the first two experiments (experiments 1 and 2) and the last two experiments (experiments 3 and 4) is the strategies employed. In the first two experiments, the trend of the different labels was not obvious, whereas, in the last two experiments, the trend was clearly seen by the human eye. Therefore, the convolution model showcases better performance for the last two strategies, especially for the buy and sell actions.

We conclude that if the strategy is clear enough to make the images obviously distinguishable, then the CNN model can predict the prices of a financial asset; the default AlexNet model is also considered good enough for prediction.

There are additional factors that we intend to research in future, for example, combining the convolu tion model with the other architectures like the LSTM. The architecture of the time-series model may help the convolution model to capture more information from the pixel images.

Acknowledgments: We would like to thank Ministry of Science and Technology (MOST) of Taiwan for providing funding (project id: 107-2218-E-002-065-) to support our researchers.

Bibliography

[1] M. Binkowski, G. Marti and P. Donnat, Autoregressive convolutional neural networks for asynchronous time series, (2017). arXiv preprint arXiv:1703.04122.

[2] A. Borovykh, S. Bohte and C. W. Oosterlee, Conditional time series forecasting with convolutional neural networks, (2017). arXiv preprint arXiv:1703.04691.

[3] S. Browne, Optimal investment policies for a firm with a random risk process: exponential utility and minimizing the probability of ruin, *Math. Oper. Res.* **20** (1995), 937–958.

[4] L. Di Persio and O. Honchar, Artificial neural networks approach to the forecast of stock market price movements, *Int. J. Econ. Manag. Syst.* **1** (2016), 158–162.

[5] C. S. Eun and S. Shim, International transmission of stock market movements, *J. Finan. Quant. Anal.* **24** (1989), 241–256.

[6] K. Fukushima and S. Miyake, Neocognitron: a self-organizing neural network model for a mechanism of visual pattern recognition, in: S. Amari and M. A. Arbib, Eds., *Competition and Cooperation in Neural Nets,* pp. 267–285, Springer, Berlin, Heidelberg, 1982.

[7] R. Mittelman, Time-series modeling with undecimated fully convolutional neural networks, (2015). arXiv preprint arXiv:1508.00317.

[8] S. E. Shreve, *Stochastic Calculus for Finance II: Continuous-Time Models,* Springer, New York, 2004.

[9] H. Wang, B. Raj and E. P. Xing, On the origin of deep learning, (2017). arXiv preprint arXiv:1702.07800.