Project 2: Paper replication II: (Re-)Imag(in)ing Price Trends

Qichuan Chen, Hengyu Shen, Sihan Yang, Yichen Zhu

October 29, 2024

Introduction

The original paper aims to predict future stock price trends by converting price data into grayscale images and using Convolutional Neural Networks (CNN) for forecasting. Unlike traditional numerical time series, image data allows the model to automatically extract complex patterns without manually setting features (Data). In the experimental setup, the authors divided samples into training, validation, and test sets by time periods and optimized the model using a cross-entropy loss function (Experiment Setting). The CNN model's capability for automatic feature extraction and noise resistance makes it especially suitable for identifying hard-to-describe price patterns in financial markets (Model).

1 Replication Approach

In this project, we will replicate the method outlined in the original paper to establish a baseline model, which will serve as a foundation for further research extensions, including Sensitivity Analysis, CNN Model Visualization, Trading Backtesting of Model Signals and Multi - Classification Model.

2 Sensitivity and Ablation Analysis

To analyze the sensitivity of the CNN model to parameter changes, we adjusted various parameters—such as the number of convolutional layers and filter sizes—and observed the model's loss under each configuration. The table below presents the specific performance of each model in this sensitivity analysis.

3 Model Training

In this section, our goal is to train a Convolutional Neural Network (CNN) on stock price trends to im-

prove the accuracy and stability of the model's predictions. The model will learn price trends based on feature images and adjust its performance through a series of evaluation metrics.

Data Preprocessing and Label Generation To train the CNN model, we load preprocessed stock price images and labels from historical data ranging from 1993 to 2000. The model's objective is to predict the future trend of stock prices. Specifically, the label generation logic is as follows: Trend labels: Calculate the stock return rate based on a specified time window (e.g., 20 days). If the return rate is positive, the label is set to "upward" (1); otherwise, it is set to "downward" (0), forming binary classification labels. Training Set Division and Data Loader Training and validation set division: We divide the dataset into a 7:3 ratio, with 70% of the data as the training set and the remaining 30% as the validation set. If the random split option is enabled, the data will be split randomly; otherwise, it will be divided in chronological order. Data loader: Training data is loaded in batches using DataLoader, with a batch size of 128 for the training set and 256 for the validation set. This setting helps reduce memory usage and improve computational efficiency. Model Training and Early Stopping Mechanism Epoch setting: The maximum number of epochs for model training is set to 100. Each epoch includes training and validation. After each training iteration, the model's performance is evaluated through the validation set, and the model is saved based on minimizing the loss. Early stopping mechanism: After each epoch's validation, if the validation loss is lower than the historical minimum min_val_loss, update the best loss and epoch count $last\_min\_ind$. If the interval between the current epoch and the epoch with the best loss exceeds the early stopping threshold ($early\_stopping\_epoch$), the training will terminate early to prevent overfitting. Training Process and Model Saving At the end of each epoch, the model is saved to a folder based on the current epoch and loss value for subsequent loading and testing. This saving step ensures that models from each stage can be tracked and further adjusted. Summary Through the above training process, the model can gradually learn the trends of stock prices and monitor the model's performance on the validation set to adjust the learning strategy. The trained CNN model will be further used for evaluation to detect its performance and predictive ability on real data.

4 Model Testing

In this section, our goal is to evaluate the trained Convolutional Neural Network (CNN) model to detect its effectiveness in stock price prediction. Through this test, we not only measure the model's performance in terms of classification accuracy but also validate its investment strategy performance through financial indicators such as annualized return, volatility, and the Sharpe ratio.

4.1 Data Preprocessing and Label Generation

To evaluate the CNN model, we loaded preprocessed stock price images and labels from the period of 2000 to 2019. The model's objective is to predict the future trend of stock price increases or decreases. Specifically, the logic for label generation is as follows:

Rise and fall labels: Based on different prediction days (e.g., 5 days, 20 days, or 60 days), we determine whether the stock's rate of return over the specified time period is positive to generate binary classification labels. If the rate of return is positive, the label is set to "rise" (1); otherwise, it is set to "fall" (0).

Label refinement: In addition, labels can be further refined based on other logic, such as defining more categories (e.g., three-class classification) based on the relative changes in short-term and medium-term rates of return of the stock.

4.2 Loading the Tested Model and Data Loader

Loading the trained model: We load multiple pretrained model files that have been trained with different training hyperparameters (such as the number of layers, filter size, dropout rate, etc.). The purpose of loading multiple models is to compare them to select the best model.

Data loader: The test data is loaded in batches using DataLoader, with each batch loading 2048 samples. Batch loading reduces memory usage and accelerates computation. The test data loader passes all test data (2000-2019) to the model in batches.

4.3 Model Evaluation

In the model evaluation section, we calculate a series of financial indicators, including annualized return (Annual Profit), maximum drawdown (Max Drawdown), annualized volatility (Annual Volatility), Sharpe Ratio, and Calmar Ratio, by defining an Evaluation class. We use the *eval_loop* function to calculate the model's average loss and collect all prediction results. We use the cross-entropy loss function to measure the model's prediction accuracy and calculate the loss for each batch's predictions and accumulate it. During the calculation process, we store the true values and predicted values for each batch to calculate classification accuracy and other indicators later. Through this loop structure, the model can be evaluated on all test data, avoiding over-computation and maintaining memory efficiency.

4.4 Output Results

Cumulative Return Chart:

Figure 1 shows the cumulative return of the model from 2000 to 2019. The blue curve represents the cumulative return of the CNN model, while the orange curve represents the benchmark's cumulative return. From the chart, we can see that the CNN model per-

formed well in most years, especially during the 2008 financial crisis and the subsequent market recovery phase, where the model captured the market's trend changes and achieved steady growth.

Equal-weighted Return Chart:

Figure 2 shows the comparison of the benchmark model, CNN model, and benchmark excess return. The CNN model (orange curve) consistently outperformed the benchmark model (blue curve), especially during periods of significant market volatility, where the CNN model demonstrated better resilience and recovery speed. This indicates that the CNN has certain advantages in dealing with market uncertainty, and the price trend signals extracted during the model training process indeed have high predictive power.

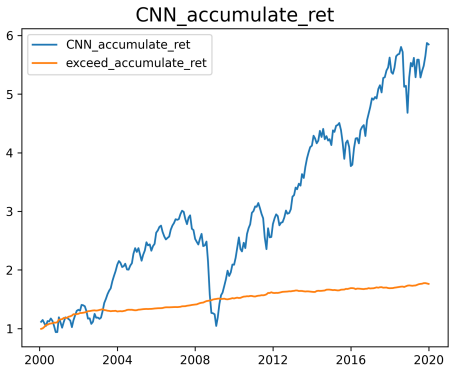


Figure 1: Cumulative Return Chart

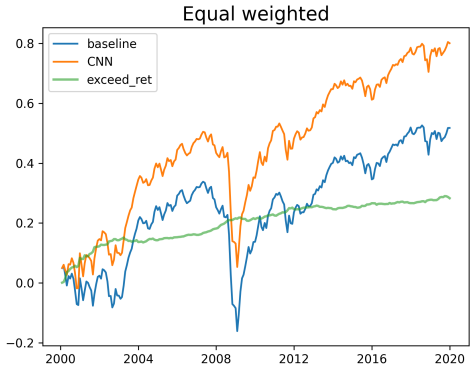


Figure 2: Equal-weighted Return Chart

4.5 Summary

The test results show that the CNN-based model excels in both long-only portfolio and risk-adjusted returns (such as the Sharpe ratio). Compared to traditional price trend signals, the model's prediction accuracy and cumulative return have significantly improved. These results validate the application potential of price trend signals learned through image recognition by the CNN model in financial markets.

5 Output Explanation

5.1 Sensitivity Analysis

To test the sensitivity of the model to architectural settings, our team has set up 14 different I20/R20 models based on the original text. The modifications made to the models are as follows:

This table presents a sensitivity analysis of the

model_names	Description
act_relu	Set the activation function to ReLU
baseline	Baseline Model
bn_no	Disable Batch Normalization
dropout_025	Set the dropout rate of the fully connected layer to 0.25
dropout_075	Set the dropout rate of the fully connected layer to 0.75
dropout_0	Disable dropout in the fully connected layer
ds_11_31	Set the dilation to (1,1) and the stride to (3,1)
filters_128	Set the number of output channels in the convolutional layer to 128
filters_32	Set the number of output channels in the convolutional layer to 32
filter_3_3	Set the size of the convolutional kernel to (3,3)
filter_7_3	Set the size of the convolutional kernel to (7,3)
layers_2	Set the number of convolutional layers to 2
layers_4	Set the number of convolutional layers to 4
maxpool_2_2	Set the pooling size of the max pooling layer to (2,2)

Table 1: Model Configuration Options

model structure and estimation methods, with each row showing different experimental configurations, each of which modifies certain model parameters. Below are the parameters modified in each experiment:

Filters (Number of convolutional kernels): Modify the number of output channels in the convolutional layer from 64 to 32 or 128.

Layers (Number of convolutional layers): Change the number of convolutional layers from 3 to 2 or 4.

Dropout (Dropout rate): Modify the dropout rate parameter of the Dropout layer, changing it from 0.50 to 0.00, 0.25, or 0.75.

BN (Batch Normalization): Choose whether to use batch normalization (Batch Normalization).

Xavier (Xavier weight initialization): Choose whether to use Xavier initialization for weights.

Activation (Activation function): Change the activation function from LeakyReLU to ReLU.

Max Pool Size (Maximum pooling size): Change the pooling size of the max pooling layer from 2x1 to 2x2.

Filter Size (Size of convolutional kernels): Change the size of the convolutional kernels from 5x3 to 3x3 or 7x3.

Dilation/Stride (Dilation rate/Stride): Modify the dilation rate and stride parameters of the convolutional layer, changing from (2,1)/(3,1) to other combinations, including (2,1)/(1,1), (1,1)/(3,1), and (1,1)/(1,1).

The following results were obtained by selecting the period from 2000 to 2019 as the backtesting period for the test set, with parameter adjustments made every 20 days. The returns of the market equal-weighted portfolio and the model equal-weighted portfolio were calculated separately, and the performance of the excess return curves was statistically analyzed. In conjunction with the model's loss on the test set and classification accuracy, the following outcomes were achieved:

From the perspective of backtesting performance, all models have yields that exceed the market return, showing a significant positive excess return. Among them, the baseline model has the most robust performance with the lowest drawdown. The *dropout_075* model has the most impressive return performance, with an excess return rate exceeding 811% during the backtesting period, and the maximum excess drawdown is less than 10%. The *layer_2* model has the highest Sharpe ratio, close to 1.25. In terms of model performance, the relu model has the highest accu-

racy, while the $filters\_32$ model has the lowest accuracy. The $dropout\_075$ model has the smallest cross-entropy loss, while the baseline model has the largest loss. One point worth noting is that the order of model precision and loss is not necessarily the same; a model with high precision may have a greater loss. This is due to the model predicting a low probability (p) for correct samples and a high probability (p) for incorrect samples. This phenomenon can also be explained at the level of returns. Model precision corresponds to the win rate at the level of returns, and the predicted probability (p) corresponds to the odds of the model. For example, the $dropout\_075$ model has an accuracy of less than 50%, but it has the lowest loss, indicating that its predicted probability (p) for correct samples (label=1) is closer to 1. At the level of returns, this corresponds to a low win rate but high odds, which is why the model's drawdown is relatively large and the returns are relatively high. For the baseline model, with a high win rate and low odds, its backtesting has a lower but stable net worth curve.

From the perspective of sensitivity, the model is not sensitive to adjustments in the activation function, as well as adjustments in dilation and stride, as the results in terms of accuracy, loss, and backtesting performance show little difference. Changes in other parameters will cause some degree of change in the results. From the perspective of model precision, adjustments to the model architecture will reduce the prediction accuracy of the model, but at the same time, it also reduces the model's loss, indicating that the original model's parameter settings fully consider the prediction accuracy of the model. The Dropout model effectively reduces the model's loss by reducing the number of neurons in the fully connected layer, which suggests that the original model has some overfitting on the training set.

Performance of Exceed Return

Model names	Period return	Annual return	Max drawback	Sharpe	Loss	Accuracy
Act relu	0.959236427	0.034180499	0.045361151	1.090609893	0.699113	0.5225438
baseline	0.927518734	0.033337348	0.039220873	1.087347898	0.699265	0.52253246
Bn no	2.379344141	0.062740277	0.061061347	1.163148269	0.692155	0.50032055
dropout_025	1.138326005	0.038710825	0.03468929	1.183072397	0.697474	0.51901406
dropout_075	8.117855729	0.116781862	0.095169228	1.107537629	0.690766	0.4894832
dropout_0	1.719270779	0.05126142	0.047817623	1.239071847	0.696602	0.503321
ds_11_31	0.955557327	0.034083365	0.066149493	0.928594782	0.694087	0.49555728
filters_128	2.318533547	0.061776346	0.048384919	0.992272115	0.691951	0.50323445
filters_32	4.09485317	0.084768888	0.083144068	0.938510063	0.691551	0.48874027
filter_3_3	3.584071741	0.079057192	0.049059781	1.178625868	0.692099	0.49722892
filter_7_3	3.131736165	0.07346962	0.039983285	1.204993386	0.691924	0.50311595
layers_2	1.590480906	0.048715533	0.044455384	1.249186813	0.69384	0.501885
layers_4	1.352287143	0.04367275	0.040685237	1.226407676	0.693792	0.49825415

5.2 Multi-classification and I20/R60

In addition, our team has expanded the classification labels and conducted training on the I20/R5 and I20/R60 models. Since the label frequency is monthly, it is not possible to backtest the weekly frequency model results. Therefore, only the I20/R60 model is selected for expansion in terms of backtesting. Furthermore, considering the momentum effect, our team has constructed a three-class prediction model. Based on the original R20, an additional category is added, with the following criteria for determination:

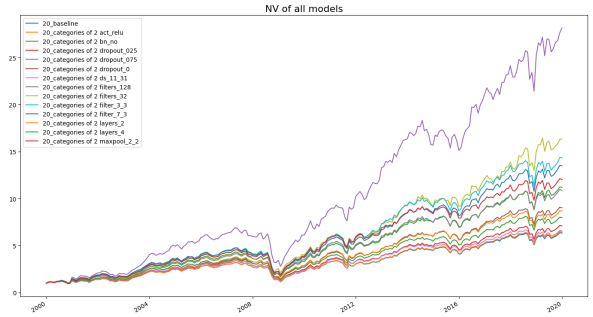
$$label = f(x) = \begin{cases} 0, & Ret_{20} > 0 \text{ and } Ret_{20} > Ret_5 \\ 1, & Ret_{20} > 0 \text{ and } Ret_{20} \leq Ret_5 \\ 2, & Ret_{20} \leq 0 \end{cases}$$

Based on the aforementioned backtesting framework, the backtesting results of the I20/R60 model and the three-classification model are shown in the figure below:

Performance of Exceed Return						
Model names	Period return	Annual return	Max drawback	Sharpe	Loss	Accuracy
Three categories	0.761446581	0.028695266	0.029491383	1.520899845	0.939591	0.46753427
Baseline_60d	0.404877396	0.017067328	0.175760254	0.462711058	0.693398	0.5015275

From the backtesting results, the three-category model has the most robust excess return among all models, with a maximum drawdown of less than 3% over 20 years, and a Sharpe ratio exceeding 1.52. The backtesting results of the I20/R60 model are unsatisfactory in all aspects. From a model perspective, the accuracy and loss of the multi-category model are both higher than those of the binary classification model, which is a disadvantage brought by the increase in the number of categories. However, this model has a more robust rate of return, indicating that the assets represented by category 0 do indeed have a more stable rate of return. For the I20/R60 model, the prediction effect is quite poor, which also suggests that it is difficult to predict the future 60-day return on assets based on the 20-day trend.

5.3 Comparison of net value of 16 types of models



It can be seen that *Dropout_075* is leading by a wide margin. The performance of Filter is also very good. All models outperform the equal-weighted market.

6 Captum

We used the IntegratedGradients method provided by Captum to interpret the model baseline. Figure3 is the original input image, Figure4 displays the absolute value mapping of Integrated Gradients. The absolute value merges positive and negative activations, allowing one to intuitively see the important areas of the image that the model focuses on. Figure5 displays the original IG activation values (including positive and negative values), which help distinguish the areas of positive and negative contributions. Positive activations indicate that the model considers these areas

conducive to predicting an increase; negative activations indicate that these areas suppress the prediction of an increase. Figure6 shows the negative IG activation values to highlight the areas that have an inhibitory effect on the prediction.

We use GuidedGradCam, a variant of Grad-CAM, which is used to visualize the feature responses of convolutional neural networks. Figure7, Figure8 and Figure9 show the absolute activation maps of the first, second, and third convolutional layers (layers) of Guided Grad-CAM, respectively. They show the image areas that the model focuses on at the shallowest layer, relatively more abstract and complex features compared to the first convolutional layer, and the highest level of abstract features, respectively.

Integrated Gradients explains the contribution of each pixel to the model's prediction. The absolute value map (test2) and the original value maps (test3, test4) allow us to see the overall important areas and the different activations (positive and negative contributions), respectively. The Guided Grad-CAM maps (test5, test6, test7) show the model's response at different levels, which helps to understand how the model gradually extracts useful features from the image. These visualizations can assist in analyzing and validating whether the model has focused on reasonable image areas, thereby enhancing the interpretability of the model.

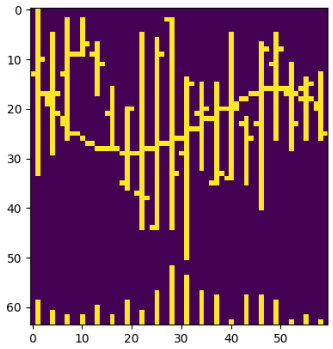


Figure 3: original

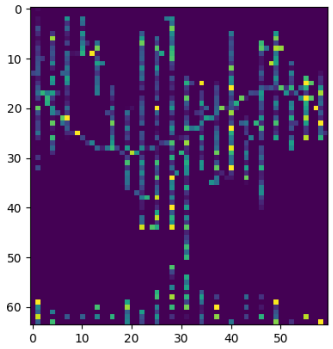


Figure 4: absolute value mapping

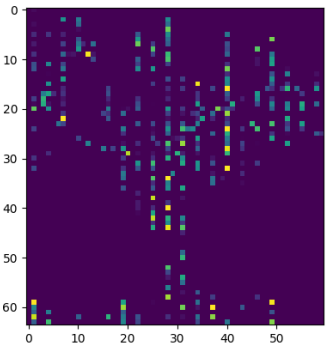


Figure 5: original IG activation values

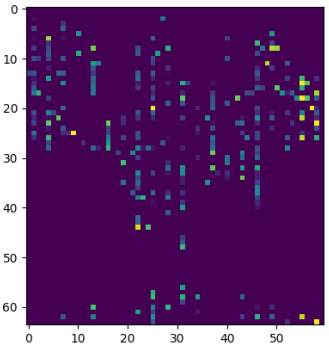


Figure 6: negative IG activation values

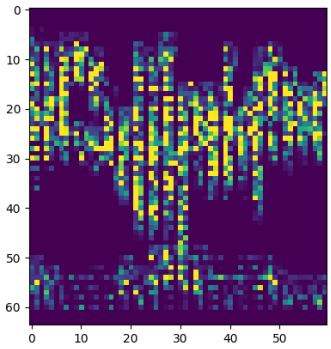


Figure 7: First

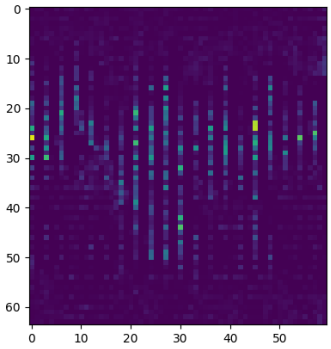


Figure 8: Second

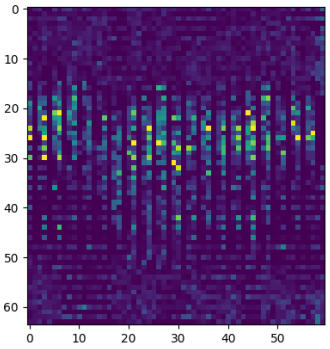


Figure 9: Third

7 References

Jiang, Jingwen and Kelly, Bryan T. and Xiu, Dacheng, (Re-)Imag(in)ing Price Trends (December 1, 2020). Chicago Booth Research Paper No. 21-01 http://dx.doi.org/10.2139/ssrn.3756587