STM JOURNALS

Current Trends in
Signal Processing

https://journals.stmjournals.com/ctsp

ISSN: 2277-6176 (Online)
ISSN: 2321-4252 (Print)
Volume 15, Issue 3, 2025
September—December
DOI (Journal): 10.37591/CTSP

Research CTSP

Deep Learning Architectures for Predictive Modeling in Financial Time Series

Nagajayant Nagamani*

Abstract

This study investigates the application of deep learning architectures, especially convolutional neural networks (CNNs), to tackle the challenging task of financial time series forecasting. Financial markets are inherently complex and volatile, driven by nonlinear, non-stationary patterns that traditional statistical methods often fail to fully model. By leveraging historical financial data—including stock prices, trading volumes, and technical indicators—the study trains CNNs to uncover both short-term local fluctuations and broader long-range temporal dependencies within market behavior. The convolutional layers act as powerful feature extractors, automatically learning latent structures that conventional approaches might overlook and enabling recognition of both local patterns (e.g., recent price swings) and global trends (e.g., seasonal cycles). Evaluation is conducted via rigorous back testing on diverse datasets, where performance metrics such as mean squared error, classification accuracy, precision, recall, and F1 score are used to assess predictive fidelity across different market conditions. These experiments demonstrate promising forecasting capabilities that offer potential value for financial decision-making and risk management. However, several key challenges are identified. Overfitting is a significant risk when training deep networks on historical data, and regularization methods such as dropout, early stopping, and weight decay are essential to improve generalization. Moreover, the non-stationary nature of financial time series, manifesting as shifting means and variances, complicates modeling and demands normalization techniques and potentially more sophisticated architectures like RevIN, meta learning enhancements, or hybrid CNN Transformer models. Recommendations for future research include refining CNN architecture design, integrating GNUs or hybrid CNN transformer or CNN BiLSTM models to better capture long-term dependencies, leveraging multimodal data including sentiment and macro indicators, and enhancing optimization strategies for increased robustness and accuracy. Overall, this work adds to the growing corpus on applying deep learning in finance, shedding light on both practical applications and key avenues for future improvements.

Keywords: Reinforcement learning, intrinsic reward, exploration strategy, random latent exploration (RLE), latent vector conditioning, latent distribution, adaptive exploration variants

INTRODUCTION

Currently, predicting stock market dynamics remains a significant concern because of the substantial financial gains it offers to stakeholders. Identifying instances when a stock price attains a local minimum presents a potential opportunity for acquisition, whereas recognizing local maxima signals optimal selling points for maximum returns [1].

Beyond conventional forecasting techniques such as fundamental and technical evaluations, computational intelligence methods have emerged prominently, particularly those leveraging artificial neural architectures. These models are proficient in uncovering latent patterns within data and

***Author for Correspondence**
Nagajayant Nagamani
E-mail: nagajayant@live.com

Manager, Software Engagement, Virtusa Limited, Chennai, Tamil Nadu, India

Received Date: July 05, 2025
Accepted Date: July 25, 2025
Published Date: August 08, 2025

Citation: Nagajayant Nagamani. Deep Learning Architectures for Predictive Modeling in Financial Time Series. Current Trends in Signal Processing. 2025; 15(3): 45–55p.

extrapolating outcomes based on the learned representations. Among them, convolutional neural networks (CNNs) have gained traction, initially propelled by the requirements of visual recognition tasks [2]. This research presents a computational framework employing CNNs not for traditional image-related tasks but to discern phases in the financial market—whether bullish, bearish, or neutral—and anticipate subsequent directional shifts. The practical utility of this approach lies in pinpointing strategic entry and exit points: initiating asset acquisition during upward movements and liquidating positions during downturns synchronized with anticipated market inflection moments to optimize profitability.

Problem Formulation

The core objective is to develop a CNN-based decision support mechanism for managing stock market assets. Rather than forecasting specific stock values for future sessions, this model assesses whether the prevailing market phase is likely to persist. It postulates a binary classification of trending versus static behavior. Investors, upon determining a trend, are advised to maintain their stance, whereas a transition to a neutral phase suggests the closure of open positions [3]. A software solution was implemented in Python to realize this predictive framework by employing a CNN to identify potential turning points in the market behavior.

Methodological Foundation

The predictive model is anchored on annotated financial market archives segmented by domain specialists into labeled intervals, each representing a particular market condition. An interactive graphical tool facilitates this annotation process, as illustrated in Figure 1 [4].

Each segment was categorized into one of three states: directional, lateral, or undetermined. For computational purposes, undefined states are equated with the lateral category and not treated as distinct classifications.

Dataset Synopsis

Table 1 presents an outline of the dataset used to construct and evaluate the model.

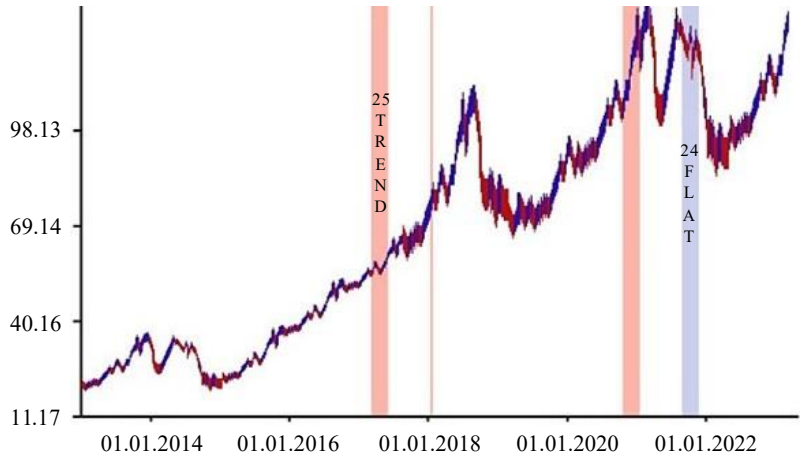

Figure 1. Graphical interface used for data segmentation.

Table 1. Summary of dataset characteristics.

Attribute	Value
File count	1389
Expert count	2
Total records	3,304,488
Financial instruments	700 (S&P Index)
Date range	28.01.2011–31.12.2022

On average, each file includes approximately 2600 daily entries. The detailed field descriptions are provided in Table 2.

SYSTEM REALIZATION

Conceptual Framework

The architectural framework comprises several interrelated components implemented in Python [5]. These include modules for training and validating classifiers and regressors to determine potential inflection points and classify the trend phases.

The primary elements are:

- Classifier to identify market state change indicators (ChP-c);
- Regressor to estimate recent trend onset positions (ChP-r);
- Categorization model for market dynamics (TF);
- Simulation engine that integrates ChP-c, ChP-r, and TF for emulating trading.
- Utility package (trend_functions) for preprocessing and support operations.

The model was trained to distinguish between trending and neutral periods (classification) and to pinpoint specific intervals (detection).

Table 2. Main field descriptions.

Field	Explanation
Date	Day of quote entry
Open	Initial trading value
High	Maximum value observed during the day
Low	Minimum trading value
Close	Final transaction price
Volume	Total traded quantity
Stock name	Identifier of the financial entity
ID select	Market condition window index; may be null
Type	Market phase (Trend, Flat; N/A treated as Flat)
Username	Expert identifier for annotation

```
In [25]:
## load the required libraries (Anaconda built-in libraries)
import pandas as pd
import numpy as np
import os
#import time
#import datetime as dt
from __future__ import print_function
from __future__ import division
#from sklearn import linear_model
#from sklearn import ensemble, metrics
#from sklearn import ensemble, cross_validation, learning_curve, metrics
#from sklearn.externals import joblib
#from sklearn.model_selection import TimeSeriesSplit, GridSearchCV,RandomizedSearchCV
In [42]:
import matplotlib
matplotlib.use('agg')## this backend is needed to convert images directly to an RGB matrix
from matplotlib import pyplot as plt
from matplotlib.finance import candlestick2_ohlc#module for drawing candlestick charts
In [26]:
```

Figure 2. Code excerpt for market classification model.

Input matrices, each corresponding to a single segmented interval, are supplied to CNN. The initial phase involves teaching the model to classify market behavior into upward, downward, or non-trending patterns [6–9]. The subsequent phase trains it to locate the boundaries between these trends. A segment of the implementation logic is shown in Figure 2.

The structure of each module generally comprises:

- Feature and label construction;
- Data ingestion;
- Model configuration;
- Model training and validation;
- Evaluation and result visualization.

The simulation logic is as follows: the ChP-c module evaluates n-day windows (e.g., n = 25 for approximately five weeks) at intervals (e.g., step size of five days). If a change point is detected, the ChP-r estimates the starting position of the trend. If no transition is noted, the previous or initial point is retained [10–12].

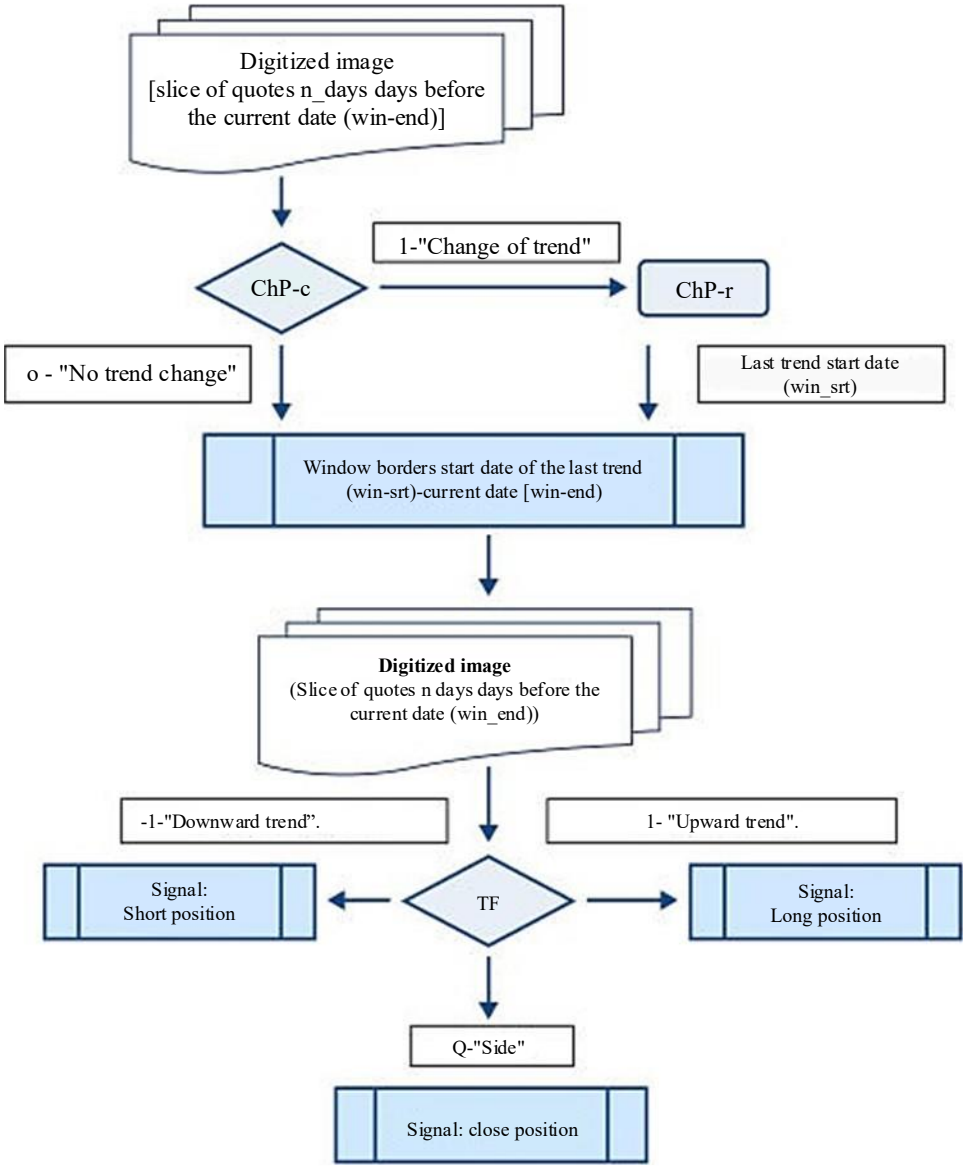

Figure 3. Interaction model of system components in real-time simulation.

TF then classified the trend between the identified start and end dates. This classification was valid until the next iteration. Final market labels were assigned using TF, and the interim values were temporarily stored for reference. The algorithm exhibits conditional behavior based on whether labeled or unlabeled data are supplied and other preset parameters. The real-time operational structure of these submodels is depicted in Figure 3.

Image Preparation

An alternative strategy for formulating a generalized computational model that maintains applicability across diverse financial instruments and is not constrained by specific set price fluctuations or transaction volumes over temporal spans involves treating market quotations as visual entities rather than numerical sequences. One representative visualization approach utilizes Japanese candlestick charts, wherein each graphical instance symbolizes a distinct categorical label [13].

This visual reinterpretation of the input data motivates the adoption of convolutional neural networks (CNNs), which are inherently designed for tasks involving visual recognition.

CNNs are employed not only for class determination, but also for spatial object localization within images. In such contexts, identification includes both classification and regression components, recognizing object types and determining their spatial extent [14].

For the application under discussion, object localization translates to trend identification, delineating both the onset and termination of financial movements over defined intervals and effectively forming snapshots. Therefore, the input to all three subnetworks (ChP-c, ChP-r, TF) comprises numerical matrices derived from the digitization of price charts corresponding to specified time frames.

Among various charting techniques, candlestick graphs (as illustrated in Figure 4) are widely utilized because of their capacity to encapsulate multiple pricing variables—opening, closing, peak, and trough values. These visuals are typically color-coded, with green denoting price increases (closing value exceeding the opening) and red denoting declines. The selection of hues is secondary to their contrast and distinguishability [15].

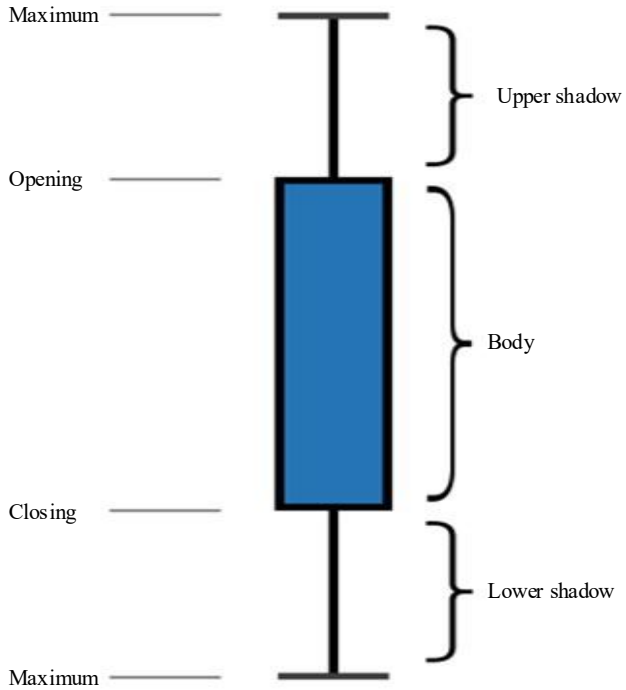

Figure 4. Visual depiction of a candlestick chart.

A candlestick image is essentially a structured array of pixels, each defined by intensity levels within the range 0–255. In grayscale representation, value 0 indicates white, while 255 denotes black, with intermediary values corresponding to varying shades. For color images, the RGB model is employed, assigning each pixel a three-component vector reflecting the red, green, and blue intensities. For instance, vectors (255, 0, 0), (0, 255, 0), and (0, 0, 255) represent pure red, green, and blue, respectively. All other hues emerge from the linear combinations of these intensities [16].

Each channel's intensity can be treated as an independent feature. Consequently, the total number of input features depends on the image resolution (measured in dots per inch, dpi) and the channel count [17]. For instance, a 3×4 image sampled at 10dpi with three channels yielded a feature map with dimensions of $48 \times 64 \times 3 = 9216$. Increasing the resolution to 20 dpi quadrupled the feature count to

$$(2 \times 48) \times (2 \times 64) \times 3 = 36864 \quad (1)$$

Conversely, converting the image to a monochromatic scheme (single channel) reduces the feature count by two-thirds.

$$48 \times 64 = 3072 \quad (2)$$

Reducing either the resolution or the number of channels enhances computational efficiency by shrinking the feature space but may result in diminished informational content.

DESIGN AND IMPLEMENTATION OF A PREDICTIVE FRAMEWORK BASED ON CONVOLUTIONAL NEURAL NETWORKS

The development of an effective predictive modeling system utilizing convolutional neural networks (CNNs) is a multistage process that requires careful consideration of various design choices and hyperparameter configurations. CNNs are widely recognized for their exceptional capability to extract spatial hierarchies and complex features from image data, making them highly suitable for tasks involving pattern recognition and classification. The current implementation is realized using the Computational Network Toolkit (CNTK) library within a Python environment, although the core methodology can be adapted to other prominent deep learning frameworks, such as TensorFlow or Caffe. Such adaptation would involve addressing differences in syntax, data-handling conventions, and system compatibility [18].

Configuration of Input Features and Output Labels

The initial step in constructing a CNN-based predictive model involves meticulous setup of the input features and their corresponding labels. This setup involves pre-processing the raw data, normalizing it, and converting it into formats compatible with the requirements of the model. The key considerations during this phase include the following.

Image Data Specifications

The choice of image resolution and the number of channels significantly impact the richness of extracted features and computational resource requirements. In the present study, the datasets consist of colored images captured at a resolution of 10 dots per inch (dpi), which were used to train two principal models labeled ChP-c and ChP-r. For the TF model, which was constrained by a smaller sample size, additional experiments were conducted using images at higher resolutions of 20 dpi and 60 dpi. While these elevated resolutions offered finer granularity, they did not translate into notable improvements in model performance but considerably increased both data size and computational load. This underscores the need for balancing resolution with practical processing limits [19].

```
labels 0 1 0|features 255 0 0 255 ….255
labels 0 0 1|features 0 0 0 255 … 0
 ….
```

Figure 5. Illustration of data recording in CTF format.

Handling Data Redundancies and Inconsistencies

Before feeding data into the model, it is crucial to identify and address potential redundancies, such as duplicate entries, as well as inconsistencies arising from conflicting or erroneous labels. Effective cleansing protocols ensure the integrity of the dataset, which, in turn, supports more reliable model training and evaluation.

Temporal Data Segmentation Parameters

For models that focus on detecting trend change points (specifically ChP-c and ChP-r), it is necessary to define the temporal window size, denoted as n_{days}, and step size or skip parameter. These parameters govern the granularity and overlap of data slices extracted from the time series. In this study, two distinct window lengths were applied: 25 d to approximate monthly intervals and 75 d for quarterly segments. The skip parameter controls the stride with which these windows traverse the dataset, thereby regulating the number of generated samples and influencing both dataset size and processing time. Dataset sizes can expand rapidly, reaching several gigabytes, and the preprocessing time can extend to multiple hours [20].

Following preprocessing, the resultant features and labels are compiled and saved in the CNTK Text Format (CTF), which is tailored for seamless integration with CNTK-based training pipelines. An illustrative example of data representation in CTF format is provided in Figure 5, where each line corresponds to a discrete data instance.

To put the data scale into perspective, a dataset comprising roughly 100,000 entries, each with 9,216 features (corresponding to one colored image at 10 dpi), occupies approximately 3–4 gigabytes of storage. This footprint can be substantially reduced by applying a binary color scheme that restricts the pixel values to either 0 or 255. Normalization by dividing these values by 255 converts them into a binary 0/1 format, enabling efficient storage and processing without significant loss of information.

Architectural Design of the CNN

Once the data preparation is finalized, the subsequent step is to architect the CNN itself. The design of the CNN involves assembling layers that extract and progressively refine feature representations from the input data. The primary layer types incorporated into the network were as follows.

- *Convolutional Layers:* These layers apply multiple learnable filters (kernels) that convolve across the input image to produce feature maps that capture local spatial features such as edges, textures, and patterns. The hyperparameters at this stage include the number of filters, kernel size, stride, padding type, and activation functions. The choice and tuning of these parameters strongly influence the model's capacity to capture relevant features [11].
- *Pooling Layers:* Pooling operations reduce the spatial dimensions of the feature maps generated by convolutional layers, thereby lowering computational complexity and aiding generalization by providing translational invariance. Common types include max-pooling and average-pooling. The sizes of the pooling window and stride are important hyperparameters that affect the degree of dimensionality reduction.
- *Fully Connected Layers:* Positioned near the network's output, these layers interpret the extracted features and map them to the final prediction space, whether they are class probabilities for classification or continuous values for regression tasks. The number of neurons and their connectivity patterns are crucial design choices for this segment of the network [10].

The network accepts a multidimensional input tensor representing the preprocessed features and produces output vectors corresponding to the desired prediction targets.

Training Strategy and Hyperparameter Specification

Training a CNN involves iterative optimization, where the network parameters are adjusted to minimize a specified loss function, effectively learning to approximate the underlying data distribution.

Within the CNTK framework, training requires explicit declaration of several critical hyperparameters that govern learning dynamics.

- *Loss Function:* This function quantifies the discrepancy between the predicted and actual outputs. Common choices depend on the task; for example, cross-entropy loss for classification problems or mean squared error for regression.
- *Optimization Algorithm:* Algorithms such as stochastic gradient descent (SGD), Adam, and RMSProp are employed to update the model weights by descending the loss landscape. The choice of optimizer affects the convergence speed and final model accuracy [2].
- *Minibatch Size:* The number of samples processed in a single forward/backward pass affects both training stability and computational efficiency. Smaller minibatches offer noisier gradient estimates but can improve generalization, whereas larger minibatches exploit hardware parallelism.
- *Number of Iterations (Epochs):* This parameter sets the number of times the entire training dataset is traversed, which directly influences the extent of learning and the risk of overfitting.
- *Learning Rate (α):* The step size used during weight updates determines how aggressively the model moves towards the minima in the loss function. Proper tuning is essential for balancing convergence speed and stability.

The selection and tuning of these hyperparameters require careful experimentation, as they have profound effects on the convergence behavior and predictive performance of the model.

Evaluation and Validation Protocols

The efficacy of the trained CNN was evaluated using validation procedures tailored to the nature of the predictive task.

- *For Classification Tasks:* Performance is assessed using metrics such as classification accuracy, which measures the proportion of correctly predicted instances. In addition, confusion matrices provide detailed insights into true-positives, false-positives, true-negatives, and false-negatives. For binary classification problems, as seen in the ChP-c model, metrics including the Area Under the Receiver Operating Characteristic Curve (AUC), precision, recall, and F-score are invaluable for understanding the model discrimination capabilities and the balance between sensitivity and specificity [4].
- *For Regression Tasks:* Evaluation relies on continuous-valued metrics such as the coefficient of determination (R^2), which indicates the proportion of variance in the dependent variable explained by the model. The mean absolute error (MAE) is another common metric that quantifies the average magnitude of prediction errors, providing an interpretable measure of accuracy.

Rigorous validation ensures that the model generalizes well to unseen data and provides robust predictions, which are critical for real-world deployment.

In summary, the construction of CNN-based predictive models requires systematic attention to data pre-processing, architectural design, training hyperparameters, and validation methodologies. Each of these components plays a vital role in shaping the performance and applicability of the model to complex image-based predictive tasks.

EVALUATION OF TRADING SIMULATION OUTCOMES

Table 3 presents the outcomes of the trading simulations for:

$n_{\text{days}} = 25$ (ChP-c_4, ChP-r_15, TF_19 sub-models), and

$n_{\text{days}} = 75$ (ChP-c_6, ChP-r_13, TF_19 sub-models).

In both scenarios, experiments were conducted on the entire test dataset (data from October 17, 2014, to May 13, 2017) encompassing all 700 stocks. A skip step of 5 was employed to reduce the number of

iterations. Despite this, each simulation required approximately two days to complete because of the extensive number of stocks and time span involved. The total number of data points for which the predictions were generated was 477,094 for $n_{days} = 25$ and 442,115 for $n_{days} = 75$.

The simulation results, when compared with those of the "average" Expert Advisor, suggest the necessity for model refinement. Despite executing trades approximately six times more frequently (increasing transaction costs), final profits are several times lower [6].

To elucidate the sources of errors, contingency matrices resulting from trading simulations were analyzed (Figures 6 and 7).

The analysis indicates that, for $n_{days} = 25$ (Figure 6), 63% of upward trend points and 86% of downward trend points were incorrectly classified as sideways, hindering profit generation from these trends. However, the model rarely confuses uptrends with downtrends, which directly resulted in losses.

Table 3. Trading simulation summary.

Model	*n*days	Profit (%)	Days in	YrFreq	YrAvg (%)
ChP-c 4, r 15, TF 19	25	811.89	101,359	5,088	0.43
Avg Expert	25	14,680.46	145,293	832	7.69
ChP-c 6, r 13, TF 19	75	1,858.58	130,529	4,747	1.05
Avg Expert	75	11,158.39	129,325	795	6.30

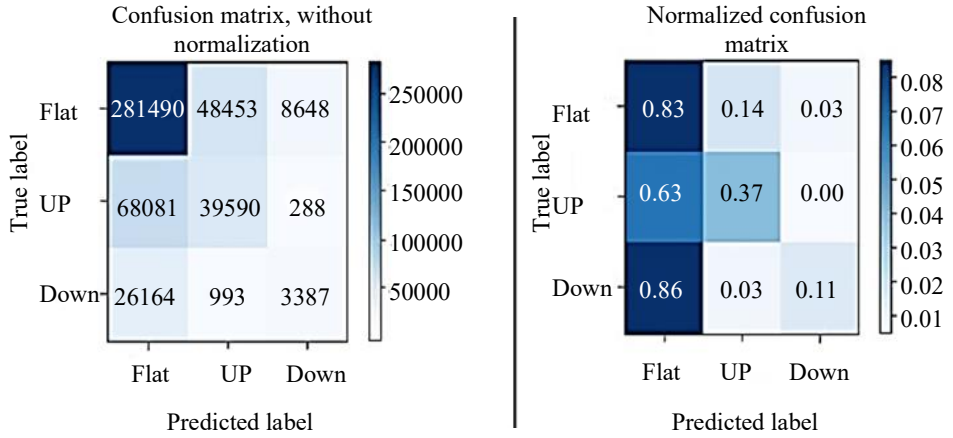

Figure 6. Confusion matrix and its normalized version for $n_{days} = 25$.

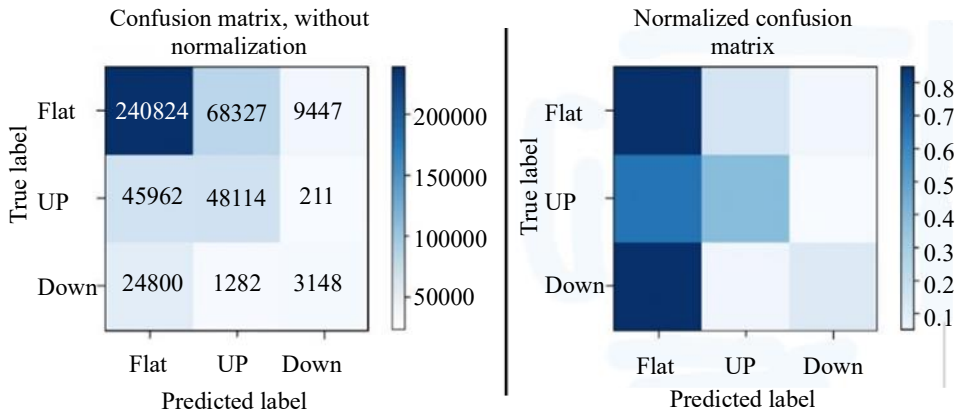

Figure 7. Confusion matrix and its normalized version for $n_{days} = 75$.

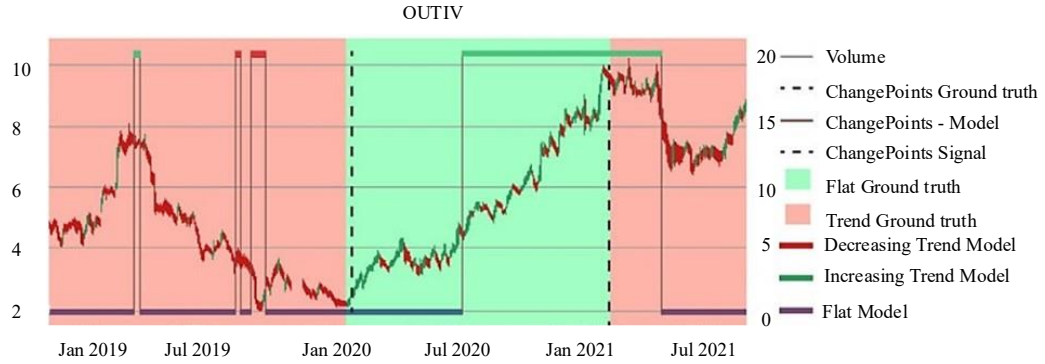

Figure 8. Example of trading simulation for $n_{days} = 25$.

Figure 9. Example of trading simulation for $n_{days} = 75$.

For $n_{days} = 75$ (Figure 7), the quality of the final labeling improved slightly. While 85% of the downtrend points are still misclassified as sideways, the proportion of unrecognized uptrends decreases to 49%, allowing for marginally better profit from this trend compared to the $n_{days} = 25$ case. Considering that most trends identified by experts are medium- to long-term, utilizing a broader data slice ($n_{days} = 75$) is preferable to accurately pinpoint market change points [17].

The primary reason for the suboptimal prediction accuracy lies in the limitations of the ChP-c and ChP-r sub-models, as previously described. These limitations may stem from inconsistencies in the original data labeling. In addition, the class imbalance concerning downtrends contributed to the relatively lower recognition accuracy of the TF model. Nevertheless, the mathematical model developed for forecasting future market condition changes using a CNN is valid, albeit improvable. Enhancing the prediction accuracy of the ChP-c, ChP-r, and TF models is the main direction for future research. Figures 8 and 9 showcase the most successful trading simulation results.

CONCLUSION

This study demonstrates an effective pipeline for forecasting stock market trends using convolutional neural networks in conjunction with preprocessed chart data. Em- phasis is placed on generalizability: the trained model must remain agnostic to specific assets, markets, or timeframes. Once trained, it is expected to operate on any security given standard market metrics, such as daily open, close, high, low, and trading volume. It is essential to recognize that training such models using historical datasets involves forecasting without prior knowledge of future asset behavior or volume dynamics. The results indicate promising prospects for deploying neural-network-based models in financial forecasting. However, this potential comes at the cost of significant computational demands and time-consuming training. In conclusion, this study represents a foundational effort toward building a comprehensive and efficient predictive system for equity markets. Future directions include refining the CNN structure, addressing class imbalances, and enhancing data-labeling techniques to improve prediction fidelity and trading outcomes.

REFERENCES

1. Cheng Y, Xu Z, Chen Y, Wang Y, Lin Z, Liu J. A deep learning framework integrating CNN and BiLSTM for financial systemic risk analysis and prediction. arXiv [Preprint]. 2025. arXiv:2502.06847. doi:10.48550/arXiv.2502.06847.
2. Duong NT, Hoang KT, Duong KQ, Dinh DQ, Le HD, Nguyen TH, Duong BX, Tran BQ, Bui AN. Investigating market strength prediction with CNNs on candlestick chart images. In: Proceedings of the 2024 6th Asia Conference on Machine Learning and Computing. New York (NY): Association for Computing Machinery; 2025. p. 120-7. doi: 10.1145/3690771.3690776.
3. Wen H. Prediction of stock price by neural network based on CNN, LSTM, ANN. Adv Econ Manag Political Sci. 2024;87:229–37. doi: 10.54254/2754-1169/87/20240899.
4. Thaker A, Chan LH, Sonner D. Forecasting agriculture commodity futures prices with convolutional neural networks with application to wheat futures. J Risk Financ Manag. 2024;17:143. doi: 10.3390/jrfm17040143.
5. Xu Z. Research on the application of convolutional neural network in stock market forecasting. Adv Eng Technol Res. 2024;10:430–7. doi: 10.56028/aetr.10.1.430.2024.
6. Varshney S, Srivastava P. A comparative study of future stock price prediction through artificial neural network and ARIMA modelling. NMIMS Manag Rev. 2023;31:229–39. doi: 10.1177/09711023241230367.
7. Zhou Z, Wu R. Stock price prediction model based on convolutional neural networks. J Ind Eng Appl Sci. 2024;2:1–10. doi:10.5281/zenodo.12780145.
8. Chahuán-Jiménez K. Neural network-based predictive models for stock market index forecasting. J Risk Financ Manag. 2024;17:242. doi: 10.3390/jrfm17060242.
9. Wang J. Stock price prediction using convolutional neural networks on various time frames. Highlights Sci Eng Technol. 2024;94:1–8. doi: 10.54097/j5ztze89.
10. Xie L, Chen Z, Yu S. Deep convolutional transformer network for stock movement prediction. Electronics. 2024;13:4225. doi: 10.3390/electronics13214225.
11. Jia D, Gao Q, Deng H. Stock market prediction based on time-frequency analysis and convolutional neural network. J Phys Conf Ser. 2022;2224:012017. doi: 10.1088/1742-6596/2224/1/012017.
12. Xiang S, Cheng D, Shang C, Zhang Y, Liang Y. Temporal and heterogeneous graph neural network for financial time series prediction. In: Proceedings of the 31st ACM International Conference on Information & Knowledge Management; 2022 Oct 17-21; Atlanta, GA, USA. New York (NY): Association for Computing Machinery; 2022. p. 3584–93. doi: 10.1145/3511808.3557089.
13. Aadhitya A, Rajapriya R, Vineetha RS, Bagde MA. Predicting stock market time-series data using CNN-LSTM neural network model. arXiv [Preprint]. 2023. arXiv:2305.14378. doi: 10.48550/arXiv.2305.14378.
14. Chen S, Guo L, Ge L. Increasing the Hong Kong stock market predictability: A temporal convolutional network approach. Comput Econ. 2024 Nov;64(5):2853–78.
15. Arauco Ballesteros MA, Martínez Miranda EA. Stock market forecasting using a neural network through fundamental indicators, technical indicators and market sentiment analysis. Comput Econ. 2025 Aug;66(2):1715–45.
16. Ran X, Shan Z, Fan Y, Gao L. A model based LSTM and graph convolutional network for stock trend prediction. PeerJ Comput Sci. 2024;10:e2326. doi: 10.7717/peerj-cs.2326.
17. Gorenc Novak M, Velušček D. Prediction of stock price movement based on daily high prices. Quant Financ. 2015;16(5):793-826. doi: 10.1080/14697688.2015.1070960.
18. Di Persio L, Honchar O. Recurrent neural networks approach to the financial forecast of Google assets. Int J Math Comput Simul. 2017;11:7-13.
19. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521:436–44. doi: 10.1038/nature 14539. PubMed: 26017442.
20. Lea C, Flynn MD, Vidal R, Reiter A, Hager GD. Temporal convolutional networks for action segmentation and detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017. p. 156–65.

