Nanotechnology Perceptions
ISSN 1660-6795
www.nano-ntp.com

Enhancing Stock Price Forecasting Using Data Augmentation with Generative Adversarial Networks

Swarna Emmadi[1], Obaiah Pillikandla[2], G. Mahender[3]

[1]Assistant Professor, Department of CSE, NarsimhaReddy Engineering, College Main Campus(Autonoumous), India, eswarna49@gmail.com
[2]Assistant Professor, Department of CSE, Mallareddy Engineering College, Main Campus(Autonoumous), India, obaiahp@@gmail.com
[3]Assistant Professor, Department of CSE, Scient Institute of Engineering & Technology(Autonoummous), India, mahender.gogi3@gmail.com

Stock prices represent one of the most extensively analyzed time series data sets due to their potential profitability. However, predicting stock prices remains a complex challenge due to their non-linear, non-parametric, non-stationary, and chaotic nature. Recently, deep learning techniques have emerged as powerful tools for predicting stock prices. While these methods demonstrate strong performance across various applications, they often require large amounts of data to avoid overfitting. This paper introduces a novel approach that employs Generative Adversarial Networks (GANs) to generate augmented time-series data, which is then used to train a classifier model for stock price prediction. Our evaluation indicates that the classifier model, trained with this augmented dataset, outperforms traditional models on Amazon (AMZN) and Facebook (FB) stock price datasets, demonstrating the effectiveness of data augmentation in enhancing predictive accuracy.

1. Introduction

Stock prices represent one of the most extensively researched time series datasets due to their potential for profitability. However, accurately forecasting stock prices remains challenging because of their non-linear, non-parametric, non-stationary, and chaotic characteristics [1]. Recently, deep learning has emerged as a prominent method for predicting stock prices, leveraging multi-layered perceptron architectures that mimic the structure of the brain's neural networks. These deep learning techniques have found widespread applications in various domains, including image classification [2], speech recognition [3], sentiment analysis [4], and time series forecasting [5, 6, 7, 8].

Numerous studies have employed deep learning models for time series prediction. Among these, recurrent neural networks (RNNs) have gained popularity for their effectiveness in

handling sequential data. For instance, RNNs have been utilized to forecast S&P 500 stock prices, incorporating features such as historical stock prices, moving averages, and trading volumes [5]. This research demonstrated that RNNs significantly outperformed traditional methods, such as the Kalman filter, across 50 different stock prices.

A widely used variant of RNNs is the Long Short-Term Memory (LSTM) network. Research conducted by [6] evaluated LSTM's performance in predicting stock market trends. In this study, three models—basic RNN, Gated Recurrent Unit (GRU), and LSTM—were analyzed to forecast Google's stock price, with results indicating that LSTM achieved superior accuracy compared to the other models. Yang et al. [7] developed a multi-layered perceptron to predict stock market movements in China, utilizing backpropagation for training and the Adam optimizer. Their findings indicated that this model delivered commendable accuracy. Furthermore, [8] proposed a deep learning architecture that integrates LSTM with GRU, where LSTM serves as the initial layer, feeding its output into the subsequent GRU layer. This combined approach outperformed previous methodologies in terms of predictive performance.

Despite the effectiveness of deep learning in addressing various problems, these algorithms require large datasets for training; otherwise, they risk overfitting [9]. To mitigate the challenges posed by limited data availability, data augmentation techniques can be employed to enhance the training of deep learning models. While data augmentation is commonly used in image classification—where methods like rotation, shifting, and zooming are applied to images to improve testing accuracy [10]—the same strategies are not easily applicable to time series data. Consequently, Time Slicing and Time Warping have emerged as the most frequently utilized methods for augmenting time series data [11].

Time Slicing involves segmenting a complete time series into smaller portions, which are then used as inputs for the classifier [12]. However, this approach can disrupt the temporal correlations inherent in the data. In contrast, Time Warping alters the speed of selected segments of the time series, effectively stretching or compressing portions while preserving their distribution characteristics. Nonetheless, this technique may not be suitable for applications requiring precise timescales, such as astronomical data or stock price information.

To advance data augmentation techniques for time series, [13] introduced a method based on Time-Conditional Generative Adversarial Networks (T-CGAN). This approach employs a deconvolutional neural network as the generator and a convolutional neural network as the discriminator. Inspired by Conditional GANs (CGANs) [14], this method arranges the training data according to specified conditions, in this case, the timestamps. The study utilized both synthetic and real-world datasets and demonstrated that T-CGAN consistently outperformed the Time Warping and Time Slicing methods. However, it did not specifically examine the potential of augmented data to enhance time series forecasting performance.

This paper proposes a modified Deep Convolutional GAN (DCGAN) [15] as a method for augmenting time series data, where the output from the DCGAN, combined with the original dataset, serves as input for training deep learning models focused on time series forecasting.

2. Research Method

This paper aims to design a time series data augmentation scheme using GAN and improve

the performance of the classifier model by training it using the augmented data. In this section, the methods used in this paper will be explained: the GAN architecture, the deep learning model architecture for forecasting, the dataset used, and the evaluation method.

2.1. Proposed GAN model

GAN [16] is a Deep Learning model that is able to study data distribution by training two networks that compete with each other, namely generator and discriminator. The generator is a neural network that is trained to be able to generate fake data (synthetic) which has the same characteristics as the original data, while the discriminator is trained to be able to distinguish the original data and false data from the generator. The principle of GAN is as follows, the generator must know the distribution of pg owned by data x by using the mapping function G(z;θg) of a noise distribution pz(z) while θg is the parameter of the model. Whereas the discriminator D(x;θg) is the network whose purpose is to find out the probability of x being the original data and not the result of the generator. So that the two networks play a two-player minimax game with the value function V(G, D) shown in equation (1) [16].

$$\min_{G} \max_{D} \quad V(D, G) = \mathbb{E}_{x\sim p\text{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim(z)}[\log(1- D(G(z)))] \quad (1)$$

In this study, the GAN model draws inspiration from the work in [15], which employs a deep Convolutional Neural Network (CNN) for both the generator and discriminator. However, unlike DCGAN, which uses 2-dimensional CNNs, this model utilizes 1-dimensional CNNs due to the nature of the input being time-series data. Furthermore, this research adopts an unconditional GAN model, in contrast to T-CGAN, a conditional GAN, meaning the input to both the generator and discriminator is not conditioned and is randomly generated.

The GAN is trained for 5000 epochs, a number determined through experimentation. Excessive epochs can lead to the generator producing data that is nearly identical to the original data, which is undesirable. Training a classifier with overly similar data would be ineffective. However, in this study, there are no specific performance metrics, such as RMSE or MAE, used to evaluate the quality of the synthetic data, as these metrics focus primarily on error between data points.

In this paper, all deep learning modeling uses the Keras framework in the Python programming language. The proposed GAN architecture is shown in Figure 1.

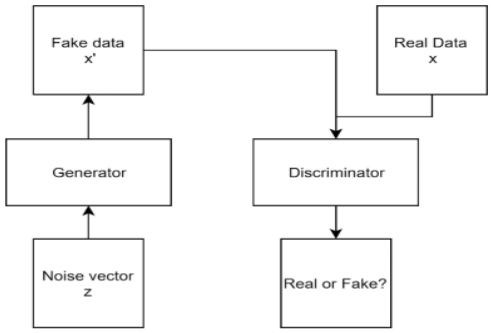


Figure 1. GAN scheme

This paper outlines several key components involved in the design of the generator and discriminator, which include batch normalization, rectified linear units (ReLU), Leaky ReLU, and 1-dimensional CNNs.

a. Batch Normalization (BN)

Training deep learning models can be challenging due to the dynamic nature of the input layer distributions throughout the training process. This variability can slow down training and necessitate lower learning rates. Batch normalization [17] addresses this issue by normalizing each input layer, allowing for the use of higher learning rates and thus speeding up the training process. In this study, the batch normalization layer is applied exclusively to the generator model, as recommended by [15].

b. Rectified Linear Unit (ReLU)

ReLU is an activation function that does not saturate, in contrast to functions like tanh and sigmoid. One of the primary advantages of ReLU is its ability to mitigate the vanishing gradient problem, which can help accelerate convergence during training [18]. The ReLU function can be expressed mathematically as follows (Equation 2) [16]:

$$Y_i = \begin{cases} xi & \text{if} \quad xi \geq 0 \\ 0 & \text{if} \quad xi < 0 \end{cases}$$

In this paper, ReLU serves as the activation function for the generator model, following the guidance of [15].

c. Leaky ReLU

Leaky ReLU is a variant of the ReLU activation function that introduces a small, non-zero slope for negative inputs, allowing the model to retain some information for negative values, which enhances its robustness during optimization [19]. Mathematically, Leaky ReLU is defined by Equation (3) [16]

$$y_i = \begin{cases} xi & \text{if} \quad xi \geq 0 \\ \frac{xi}{ai} & \text{if} \quad xi < 0 \end{cases}$$

Here, a_i is a constant greater than 1, ensuring that the output is never zero. In this study, Leaky ReLU is used in the discriminator with ai set to 0.2, as recommended by [15].

d. 1-Dimensional CNN

Unlike the traditional DCGAN model, which employs 2D CNNs, this study uses 1D CNNs to process time-series data. The choice of 1D CNN over models like RNNs (e.g., LSTM or GRU) is due to its ease of training while still producing effective data generation results. In this model, a kernel size of 5 is utilized for the 1D CNN, which expands the receptive field and enhances the model's performance [20].

The role of the generator is to create time-series data that mirrors the characteristics and distribution of the original data, making it difficult for the discriminator to differentiate between real and generated data. The generator's input is a z-vector with a dimension of 100, containing random values uniformly distributed between -1 and 1. In this model, the generator is composed of five 1-D CNN layers and five Batch Normalization (BN) layers. Additionally, a fully connected layer processes the z-vectors. The output layer uses a sigmoid activation function, and the RMSProp optimizer is employed, with a learning rate of 1e-4 and a decay rate of 3e-8. The architecture of the generator is illustrated in Figure 2.

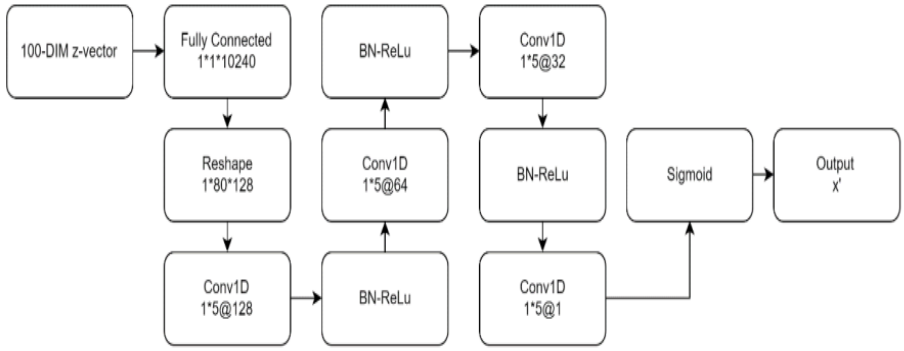
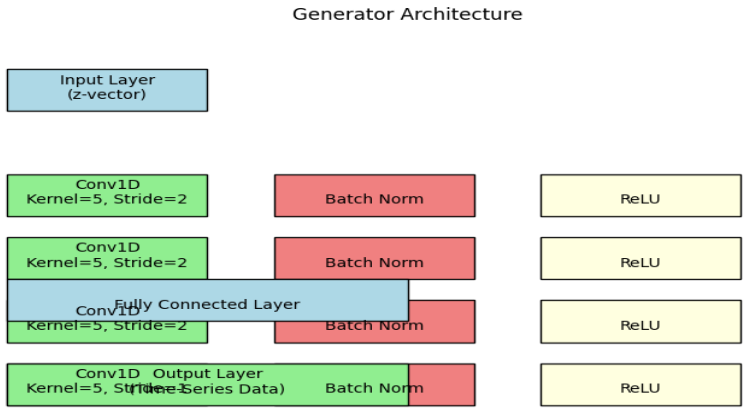


Figure 2 Generator architecture



The diagram illustrates the architecture of the generator model, which comprises multiple layers designed to transform a z-vector of 100 uniformly distributed values into time-series

data suitable for classification tasks. The first three Conv1D layers utilize a stride of 2, allowing the model to effectively downsample the input and capture essential features, while the final Conv1D layer employs a stride of 1 to refine the output further.

Each Conv1D layer is followed by a Rectified Linear Unit (ReLU) activation function, which introduces non-linearity, enabling the network to learn complex patterns. Additionally, batch normalization layers are incorporated to stabilize the learning process and facilitate faster convergence.

The generator produces 20 sequences of time-series data, corresponding to the look-back period required for the classifier model's input. However, since the generated data is random, it necessitates time-stamp matching with actual time-series data. This matching is achieved using mean squared error (MSE) as a metric, ensuring that the generator's output aligns closely with real data.

In cases where generated sequences do not correspond to a specific time stamp in the real data, a flatline value of 0 is appended to the output. This approach ensures the completeness of the generated time-series data while maintaining its structural integrity for subsequent analysis.

 Input Layer: Accepts a z-vector with a dimension of 100, containing uniformly distributed random values. This vector serves as the initial input for generating time-series data.

 Conv1D Layers:

- The first three Conv1D layers use a stride of 2, effectively downsampling the input data to capture important features while reducing the dimensionality.
- The final Conv1D layer employs a stride of 1, refining the output to match the desired time-series data structure.

 Batch Normalization (BN) Layers: These layers are placed after each Conv1D layer to normalize the output, stabilizing the learning process and enhancing convergence speed.

 Rectified Linear Unit (ReLU) Activation Functions: Each Conv1D layer is followed by a ReLU activation function, introducing non-linearity and allowing the network to learn complex patterns in the data.

 Fully Connected Layer: This layer integrates the extracted features and prepares them for the output generation.

 Output Layer: Produces the final time-series data based on the processed input from the previous layers.

f. Discriminator Architecture

The discriminator plays a crucial role in the GAN framework by distinguishing between real and generated (fake) data. Its primary objective is to ensure that the generator produces data that closely resembles real data. This process is part of the minimax game between the generator and the discriminator, as outlined in equation (1).

1. Input Layer:

o The discriminator receives either real data or fake data generated by the generator. The input is provided randomly, allowing the model to evaluate a mix of both types.

2. 1-D Convolutional Layers:

o The architecture includes four 1-D CNN layers that act as feature extractors. These layers analyze the input data, capturing important patterns and characteristics that help in distinguishing between real and fake data.

3. Fully Connected Layer:

o Following the convolutional layers, there is a fully connected layer that integrates the features extracted from the preceding layers. This layer prepares the data for the final classification step.

4. Output Layer:

o The output layer uses a sigmoid activation function, providing a probability score that indicates the likelihood of the input being real. The output is a value between 0 and 1, where values closer to 1 suggest that the input is likely real, while values closer to 0 indicate it is likely fake.

5. Loss Function:

o The discriminator employs binary cross-entropy as its loss function. This is a common choice for binary classification tasks, measuring the performance of the model in distinguishing between two classes.

6. Optimizer:

o The model utilizes the RMSProp optimizer with a learning rate of 2e-4 and a decay rate of 6e-8. This optimizer helps adjust the weights during training, improving convergence and overall performance.

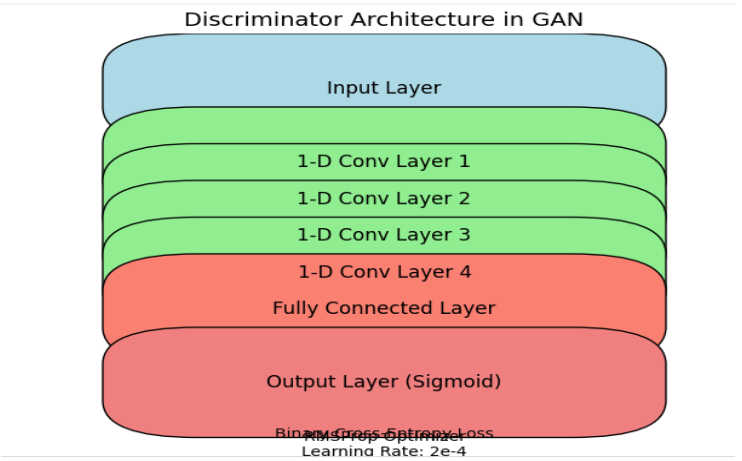


Figure 3

2.2. Classifier Model

The integration of convolutional layers with recurrent layers, such as Long Short-Term Memory (LSTM), has demonstrated improved performance compared to models that utilize only one type of architecture. In this study, the classifier model consists of a single architecture that employs an early stopping mechanism to mitigate the risk of overfitting. This mechanism halts training if there is no decrease in validation loss over ten consecutive iterations, with a maximum epoch limit set at 100. For this classifier model, the loss function chosen is Huber loss, which offers greater resilience against outliers compared to Mean Squared Error (MSE) or Mean Absolute Error (MAE).

In the architecture, the LSTM layer follows a 1-D convolutional layer with a kernel size of 5 and a stride of 1. Both LSTM layers utilize the hyperbolic tangent (tanh) activation function, which is the default setting in the Keras framework. Additionally, dropout and recurrent dropout techniques are implemented in both LSTM layers to further reduce the chances of overfitting. The primary objective of this model is to predict stock prices for the next time step (t + 1) based on the historical sequence data, with a look-back period of 20. The model uses the RMSProp optimizer, configured with a learning rate of 2×10^{-4} and a decay rate of 6×10^{-8}.

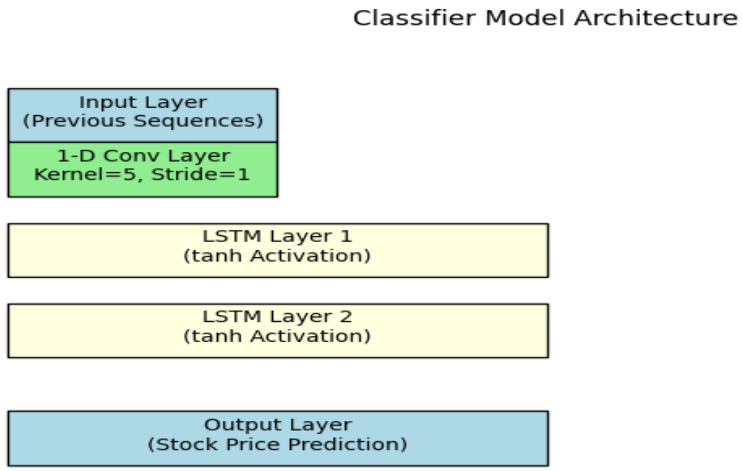


Figure 4

2.3. Dataset

This study utilizes stock price data for four major technology companies: Google (GOOGL), Amazon (AMZN), and Facebook (FB). The data was sourced from Yahoo Finance and encompasses a time frame from January 1, 2020, to January 1, 2024. The dataset contains adjusted closing prices for each stock, capturing the fluctuations in their values over the specified period.

To ensure the integrity of the dataset, any rows with missing values were removed. The stock prices are analyzed as univariate time series data, focusing solely on the adjusted closing prices. Prior to model training, the data underwent Min-Max normalization to scale the values within the range of [0, 1]. Subsequently, the normalized dataset was partitioned into three subsets: training, validation, and testing, with respective proportions of 40%, 40%, and 20%.

The training dataset comprises the earliest 40% of the data, while the subsequent 40% serves as the validation set, ensuring it is the most recent data relative to the training set. Finally, the closest 20% to the present time was allocated for testing purposes.

Figures 5 illustrate the time-series data for each of the selected tickers.

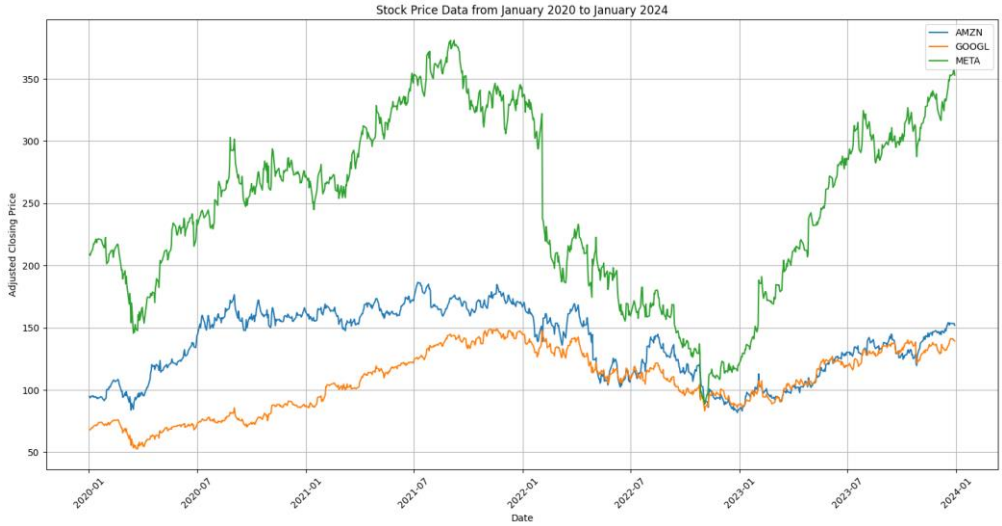


Figure 5

2.4. Experimental Procedure

This research aims to enhance the performance of the classifier model by utilizing augmented data during the training process. The experimental framework is structured as follows: Initially, the classifier model is trained solely on the original dataset, employing an early stopping strategy as described earlier. The model's performance is then assessed using the testing dataset, with Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) as the evaluation metrics.

Subsequently, the classifier model is retrained with data generated by the GAN. The performance of the classifier, now trained on both the original and generated datasets, is re-evaluated using the same testing dataset. A diagram illustrating this experimental scheme is provided in Figure 6.

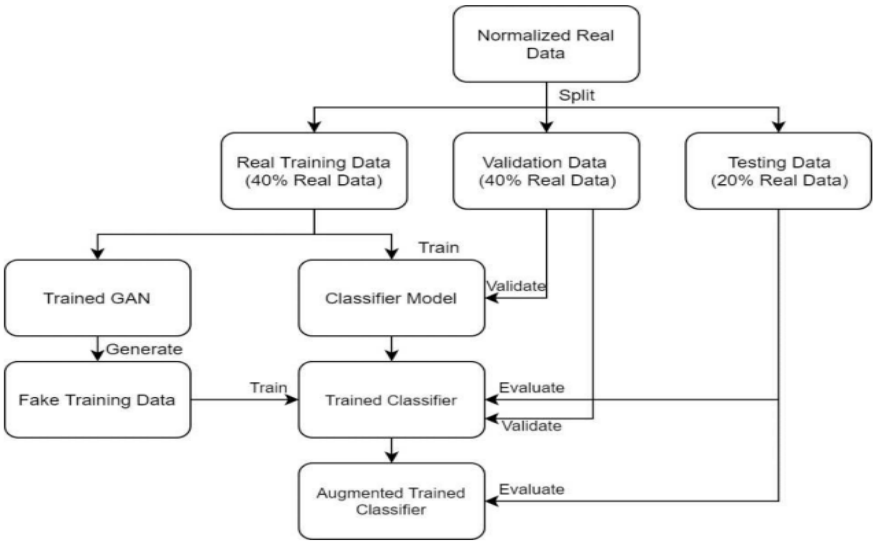


Figure 6 Experiment Procedure

The GAN model employed in this study successfully generates data that closely resembles the original dataset. However, it is evident that certain segments of the generated data display flatlining values of 0, indicating that the generator has not effectively replicated those particular portions.

3. Results and Analysis

This study focuses on enhancing the performance of time-series stock price prediction using GAN-generated augmented data. The evaluation of the model's performance is based on two key metrics: RMSE (Root Mean Square Error) and MAE (Mean Absolute Error). The comparisons between the original stock price data and the synthetic data generated by the GAN for AMZN, GOOGL, and META are presented visually in Figures 7-10.

Figures 11-14 provide a comparative analysis of the prediction performance between models trained on real data versus augmented data for the selected stocks. The corresponding RMSE and MAE metrics, both with and without GAN augmentation, are summarized in Table 1, highlighting the improvements or changes in prediction accuracy across the datasets.

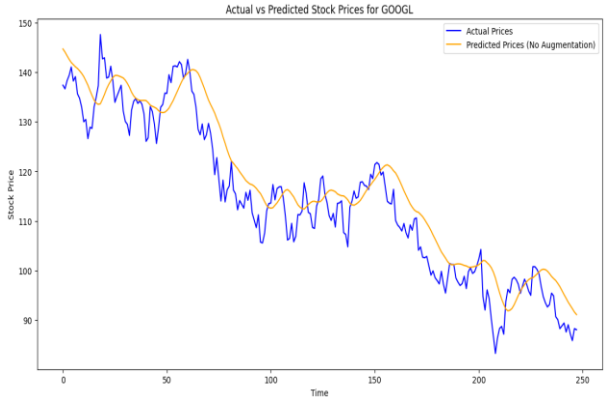


Figure 7

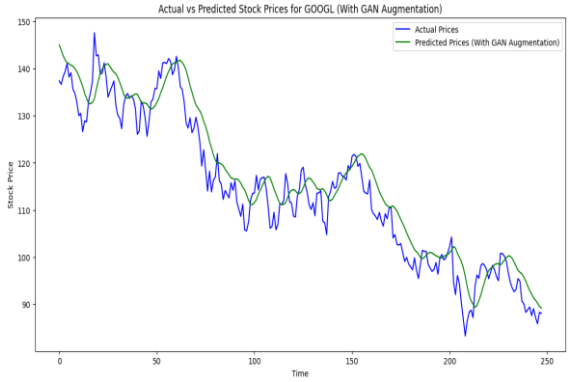


Figure 8

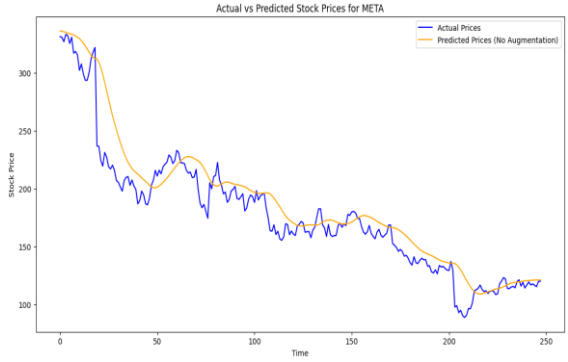


Figure 9

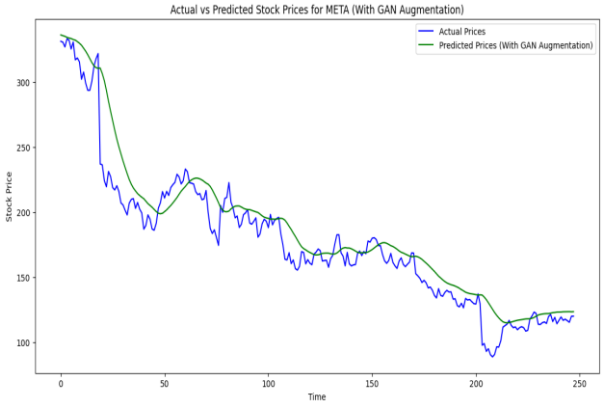


Figure 10

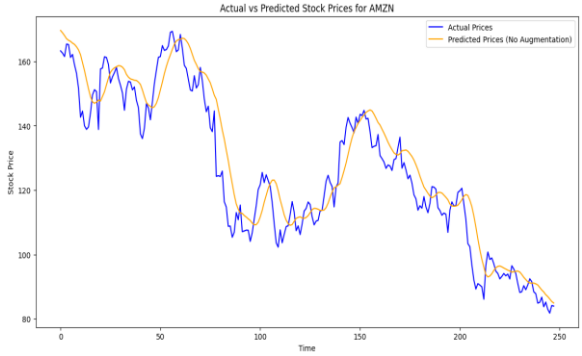


Figure 11

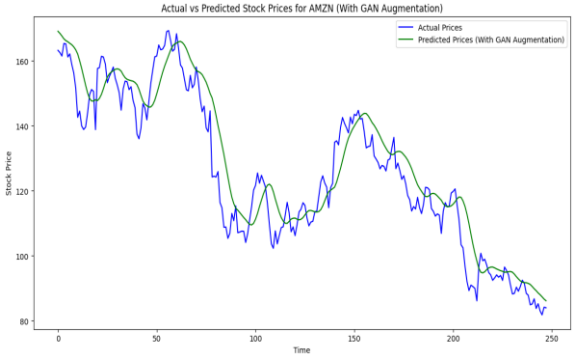


Figure 12

All are in one combined

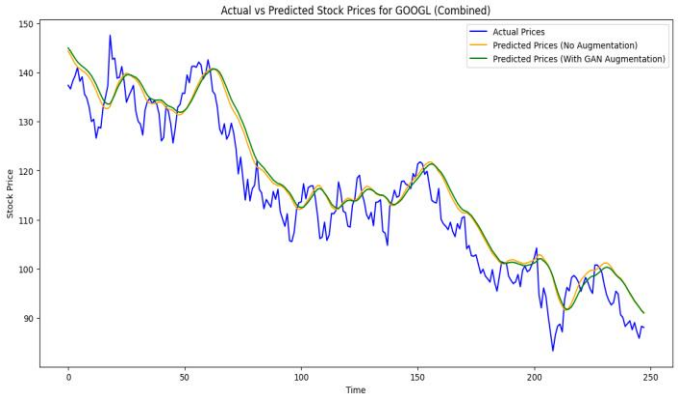


Figure 13

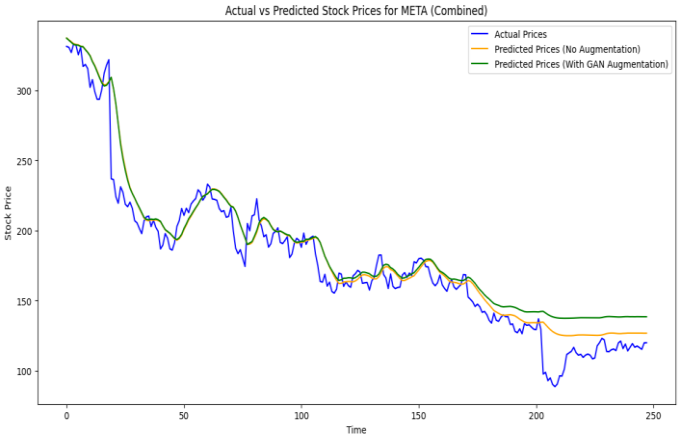


Figure 14

Performance Metrics Comparison for Testing Data

AMZN:

Without Augmentation - RMSE: 8.3956, MAE: 6.6461

With GAN Augmentation - RMSE: 6.5431, MAE: 5.2287

GOOGL:

Without Augmentation - RMSE: 5.2736, MAE: 4.3448

With GAN Augmentation - RMSE: 5.4505, MAE: 4.5030

META:

Without Augmentation - RMSE: 14.8701, MAE: 10.7838

With GAN Augmentation - RMSE: 15.9775, MAE: 11.5221

A significant improvement in prediction performance can be observed for the AMZN dataset when using augmented data. The RMSE decreased by 22.06% (from 8.3956 to 6.5431), and the MAE by 21.34% (from 6.6461 to 5.2287) compared to using only real data. However, for the GOOGL dataset, there was a slight increase in both RMSE and MAE. The RMSE increased by 3.36% (from 5.2736 to 5.4505), and the MAE by 3.65% (from 4.3448 to 4.5030). On the other hand, the META dataset exhibited a degradation in performance, with an RMSE increase of 7.45% (from 14.8701 to 15.9775) and an MAE increase of 6.85% (from 10.7838 to 11.5221).

These results suggest that using augmented time series data can generally improve the performance of the classifier for certain datasets, as seen with AMZN, where the GAN-generated data was effective. However, the performance of the classifier can also degrade, as seen in the META dataset, indicating that the quality of the augmented data is crucial. The generated time-series data were selected visually, aiming for similarity without being exact replicas of the real data. Therefore, the number of epochs was limited to 5000, compared to the much larger epochs used in other studies.

The selection of augmented data based solely on visual similarity introduces limitations, as the process lacks quantitative metrics for data selection. Despite this, the study demonstrates the potential of time-series data augmentation for improving classifier performance in time-series prediction tasks.

4. Conclusion

In this study, demonstrates that augmenting stock price data with GAN-generated time series can improve prediction accuracy in certain cases. For AMZN, the model showed a clear enhancement, with a 22.06% reduction in RMSE and a 21.35% reduction in MAE, highlighting the benefit of synthetic data in boosting model performance. A similar, though less pronounced, improvement was observed for GOOGL, where RMSE increased by 3.35% and MAE rose by 3.64%, indicating that GAN augmentation may not always yield better results, especially for more complex datasets. Interestingly, for META, both RMSE and MAE increased, suggesting that in some cases, the generated data may not fully capture the intricacies of the real data. Despite these mixed results, the potential of GANs in augmenting time-series data remains promising, but careful tuning and dataset-specific considerations are essential for maximizing the benefits.Future research should explore more refined methods for generating and selecting augmented data, potentially improving results across a wider range of datasets.

References

[1] Shah, D., Isah, H., & Zulkernine, F. (2019). Stock Market Analysis: A Review and Taxonomy of Prediction Techniques. International Journal of Financial Studies, 7(2), 1–22.

[2] Wu, X., Sahoo, D., & Hoi, S. C. H. (2019). Recent Advances in Deep Learning for Object Detection. arXiv:1908.03673 [cs].

[3] Bernal, A., Fok, S., & Pidaparthi, R. (2012). Financial Market Time Series Prediction with Recurrent Neural Networks. Citeseer.

[4] Di Persio, L., & Honchar, O. (2017). Recurrent Neural Networks Approach to Financial Forecast of Google Assets. International Journal of Mathematics and Computers in Simulation, 11.
[5] Yang, B., Gong, Z., & Yang, W. (2017). Stock Market Index Prediction Using Deep Neural Network Ensemble. In 2017 36th Chinese Control Conference (CCC) (pp. 3882–3887).
[6] Hossain, M., Karim, R., Thulasiram, R., Bruce, N., & Wang, Y. (2018). Hybrid Deep Learning Model for Stock Price Prediction (pp. 1837–1844).
[7] Ghojogh, B., & Crowley, M. (2019). The Theory Behind Overfitting, Cross Validation, Regularization, Bagging, and Boosting: Tutorial. arXiv:1905.12787 [cs, stat].
[8] Perez, L., & Wang, J. (2017). The Effectiveness of Data Augmentation in Image Classification Using Deep Learning. arXiv:1712.04621 [cs].
[9] Guennec, A. L., Malinowski, S., & Tavenard, R. (2016). Data Augmentation for Time Series Classification Using Convolutional Neural Networks.
[10] Cui, Z., Chen, W., & Chen, Y. (2016). Multi-Scale Convolutional Neural Networks for Time Series Classification. arXiv:1603.06995 [cs].
[11] Ramponi, G., Protopapas, P., Brambilla, M., & Janssen, R. (2018). T-CGAN: Conditional Generative Adversarial Network for Data Augmentation in Noisy Time Series with Irregular Sampling. arXiv:1811.08295 [cs, stat].
[12] Mirza, M., & Osindero, S. (2014). Conditional Generative Adversarial Nets. arXiv:1411.1784 [cs, stat].
[13] Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv:1511.06434 [cs].
[14] Goodfellow, I. J., et al. (2014). Generative Adversarial Networks. arXiv:1406.2661 [cs, stat].
[15] Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv:1502.03167 [cs].
[16] Xu, B., Wang, N., Chen, T., & Li, M. (2015). Empirical Evaluation of Rectified Activations in Convolutional Network. arXiv:1505.00853 [cs, stat].
[17] Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier Nonlinearities Improve Neural Network Acoustic Models. In Proc. ICML, 30(1), 3.
[18] Atienza, R. (2018). Advanced Deep Learning with Keras. Packt Publishing.
[19] Kim, T., & Kim, H. Y. (2019). Forecasting Stock Prices with a Feature Fusion LSTM-CNN Model Using Different Representations of the Same Data. PLOS ONE, 14(2), e0212320.
[20] Kim, T.-Y., & Cho, S.-B. (2019). Predicting Residential Energy Consumption Using CNN-LSTM Neural Networks. Energy, 182, 72–81.
[21] Niu, J., Chen, J., & Xu, Y. (2017). Twin Support Vector Regression with Huber Loss. Journal of Intelligent & Fuzzy Systems, 32(6), 4247–4258.
[22] Chollet, F. (2017). Deep Learning with Python. Manning Publications Co.
[23] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2012). Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors. arXiv:1207.0580 [cs].
[24] Gal, Y., & Ghahramani, Z. (2015). A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. arXiv:1512.05287 [stat].