Stock Prediction Model Report

CP468 - Artificial Intelligence - Group #22

August 2nd 2024

Dr. Emad Mohammed

Ranjot Sandhu (169020301)

Rishubh Gusain (169044443)

Usama Mohiuddin (212090340)

Rupesh Rangwani (169016076)

Preface

In this project, we investigate how to anticipate stock values based on past market data using sophisticated deep learning techniques, namely convolutional neural networks (CNNs) and pre-trained models like ResNet50, InceptionV3, and DenseNet121. The principal aim of this study is to assess the precision and efficacy of these models in predicting stock prices and investigate the possibility of utilizing an ensemble technique to enhance prediction outcomes. In order to facilitate user interaction with the models and provide real-time predictions and visualizations, we developed a Gradio interface. The whole methodology—including data preparation, model architecture, training procedures, assessment metrics, and outcomes—is presented in this study. It also talks about the difficulties that have been faced and provides some insight into possible future research and applications of artificial intelligence.

Over the past few decades, there has been a lot of interest in and research on the use of artificial intelligence (AI) to stock price prediction. Basic statistical techniques and linear models, such as autoregressive integrated moving average (ARIMA) models, were the foundation of early approaches in the 1980s and 1990s. Support vector machines (SVMs) and random forests are two examples of the more advanced machine learning algorithms that were created as processing power and data availability expanded.

Deep learning methods have transformed the sector recently. The capacity of recurrent neural networks (RNNs) and their variations, including long short-term memory (LSTM) networks, to extract temporal dependencies from time series data led to their rise in popularity. After being originally developed for image identification applications, convolutional neural networks (CNNs) were modified to handle sequential data, opening up new avenues for feature extraction and pattern recognition in stock market data.

The use of AI in stock price prediction has been the subject of numerous research initiatives and studies. For instance, a 2015 study showed that LSTM (Long Short-Term Memory Work) networks outperformed conventional techniques in stock price prediction. Deep reinforcement learning was used in another noteworthy research to create trading strategies, and the results were encouraging in terms of profitability and risk control.

For financial forecasting, pre-trained models like ResNet50, InceptionV3, and DenseNet121 have also been modified. These models are used because of their strong feature extraction capabilities, which were initially created for picture categorization. By utilizing financial data to refine these models, researchers have significantly increased the accuracy of their predictions.

The projections of several models are combined by ensemble models to provide a forecast that is more reliable and accurate. The goal is to minimize each model's unique shortcomings while maximizing its strengths. Techniques like bagging, boosting, and stacking are frequently used in groups.

In this study, we investigate how an ensemble technique might enhance the accuracy of stock price predictions. Our goal is to improve prediction accuracy and dependability by merging the outputs of multiple sophisticated deep learning models, including CNNs and pre-trained models like ResNet50, InceptionV3, and DenseNet121. The ensemble approach is intended to combine many viewpoints on the data, increasing the prediction potential overall.

We created a Gradio interface to make it easier for users to interact with the models. This makes it possible for users to enter data, get forecasts in real time, and easily visualize the outcomes. Both novice users and financial specialists interested in stock market predictions can easily navigate and understand the interface's design.

Introduction

Problem Statement

The stock market is a complex and dynamic system, where predicting future prices is inherently challenging due to numerous influencing factors, such as economic indicators, market sentiment, and global events. Traditional forecasting methods, while useful, often fail to capture the non-linear and chaotic nature of financial data. With the advent of deep learning, there is potential to uncover complex patterns and improve prediction accuracy. This project aims to leverage CNNs and pre-trained models to address this challenge, providing insights into their effectiveness and exploring an ensemble approach for enhanced performance.

Objective

The project has two main objectives:

1. **Develop and Compare Models**: Build and evaluate the performance of a custom CNN model and pre-trained models (ResNet50, InceptionV3, DenseNet121) in predicting stock prices.
2. **Ensemble Model Approach**: Investigate the potential of an ensemble model, which combines the strengths of individual models, to improve prediction accuracy and robustness.

Scope

The project involves the collection and preprocessing of historical stock data, development and training of deep learning models, evaluation using standard metrics, and the implementation of a user-friendly interface. The scope also includes an analysis of the models' performance, discussion of the challenges faced, and exploration of potential future improvements.

Literature Review

Traditional Methods

Traditional time series forecasting methods, such as ARIMA and GARCH, have been widely used for predicting stock prices. These methods are grounded in statistical theories and are particularly suited for linear and stationary data. ARIMA models forecast future values based on past values and error terms, while GARCH models focus on predicting volatility. However, these methods often struggle with the complex, non-linear patterns and high volatility characteristic of financial markets.

Deep Learning in Finance

Deep learning models, especially CNNs, RNNs, and LSTMs, have gained popularity in financial forecasting due to their ability to model complex, non-linear relationships. CNNs, originally designed for image recognition, have been adapted for time series analysis by treating time series data as a one-dimensional "image." Pre-trained models, trained on large datasets such as ImageNet, bring a wealth of knowledge and are fine-tuned for specific tasks. These models can capture intricate patterns and dependencies, offering a significant advantage over traditional methods.

Methodology

Data

Dataset

The Dataset consists of daily historical stock prices, including features such as 'Open', 'High', 'Low', 'Close', and 'Volume'. These features provide a comprehensive view of the stock's performance, capturing both price movements and trading activity. The data was sourced from reliable financial data providers and covers multiple stocks over several years, offering a robust basis for training and evaluation.

The dataset used in this project is obtained from Kaggle, a well-known platform for datasets and data science competitions. The specific dataset, titled "Stock Market Dataset," can be found here - https://www.kaggle.com/datasets/jacksoncrow/stock-market-dataset.

The dataset contains comprehensive information on both exchange-traded funds (ETFs) and individual stocks. However, for the purposes of this project, the primary focus is on individual stocks of companies.

The stock data is stored in a folder named "Stocks," which comprises a total of 5884 files. Each file corresponds to a different company's stock and contains historical price data. The data files are in CSV format, facilitating easy loading and processing using Python's data manipulation libraries such as pandas.

ach CSV file includes the following columns:

- Date: The specific date of the trading day.
- Open: The opening price of the stock on that day.
- High: The highest price reached by the stock during the trading day.
- Low: The lowest price reached by the stock during the trading day.
- Close: The closing price of the stock on that day.

- Adj Close: The adjusted closing price, which accounts for any corporate actions such as stock splits or dividends that might affect the stock price.
- Volume: The total number of shares traded during the day.

Data Preprocessing

Data preprocessing is a crucial step in ensuring the quality and consistency of the input data. The following preprocessing steps were applied:

- **Normalization**: Min-max normalization was used to scale the 'Close' prices to a range between 0 and 1. This step helps in stabilizing the training process and accelerates convergence.
- **Handling Missing Values**: Missing values in the dataset were addressed using linear interpolation. In cases where missing values were substantial, affected data points were removed to maintain data integrity.
- **Splitting Data**: The dataset was divided into training, validation, and test sets. The training set was used to fit the models, the validation set for hyperparameter tuning and early stopping, and the test set for final evaluation.

Model Design

Custom CNN Model

The custom CNN model was designed to extract temporal patterns from the stock price data. The architecture includes:

- **Conv1D Layers**: These layers apply convolutional filters along the temporal dimension, capturing short-term dependencies and patterns.
- **MaxPooling1D Layers**: These layers down-sample the output from the convolutional layers, reducing dimensionality and computational load while preserving important features.
- **Flatten Layer**: This layer converts the multi-dimensional feature maps into a one-dimensional vector, preparing it for the fully connected layers.
- **Dense Layers**: The fully connected layers are responsible for integrating the features extracted by the convolutional layers. The final dense layer outputs the predicted stock prices, using a linear activation function for regression.

Pre-trained Models

Pre-trained models such as ResNet50, InceptionV3, and DenseNet121 were adapted for the task of stock price prediction. These models, originally designed for image classification, were fine-tuned by replacing the top layers with custom dense layers tailored for regression tasks. The pre-trained models leverage their deep, hierarchical feature extraction capabilities, providing rich representations of the input data.

ResNet50 Model

ResNet-50 is a deep residual network with 50 layers. It introduces the concept of residual learning, where shortcut connections, or skip connections, allow the network to learn identity mappings, helping to mitigate the vanishing gradient problem in deep networks. This architecture allows the model to focus on learning residual functions, which are easier to optimize. ResNet-50 is widely used in image recognition tasks due to its ability to train deep networks effectively.

Key Features:

- **Residual Blocks:** Each block includes a shortcut that bypasses one or more layers, facilitating gradient flow during training.
- **Bottleneck Architecture:** Comprises 1x1, 3x3, and 1x1 convolutional layers to reduce computation while maintaining accuracy.
- **Applications:** Used in image classification, object detection, and more.

InceptionV3

InceptionV3 is an advanced version of the Inception model, known for its inception modules. These modules consist of convolutional layers of various sizes, allowing the model to capture different spatial features simultaneously. The architecture also incorporates factorized convolutions and auxiliary classifiers, which help in training and regularization.

Key Features:
- **Inception Modules:** Combine multiple convolutional operations in parallel, capturing multi-scale features.
- **Factorized Convolutions:** Reduce computational complexity while maintaining expressiveness.
- **Auxiliary Classifiers:** Assist in gradient propagation, helping the model converge faster.

DenseNet121

DenseNet121 is part of the DenseNet family, characterized by its densely connected layers. Unlike traditional CNNs, where each layer has its weights, DenseNet connects each layer to every other layer in a feed-forward fashion. This design allows for better feature reuse and gradient flow, making the network more efficient and less prone to vanishing gradients.

Key Features:
- **Dense Connections:** Each layer receives inputs from all previous layers, promoting feature reuse.
- **Compact and Efficient:** Requires fewer parameters compared to other architectures, reducing the risk of overfitting.
- **Applications:** Image classification, medical image analysis, etc.

Ensemble Model

The ensemble model combines the predictions from the CNN, ResNet50, InceptionV3, and DenseNet121 models.

The ensemble approach aims to mitigate the weaknesses of individual models and capitalize on their strengths.

The predictions are averaged to produce a final prediction, potentially improving accuracy and robustness.

Training and Evaluation

Training Process

The training process involved several key steps:

- **Synthetic Data for Image-Based Models**: To train the pre-trained models, which expect image input, synthetic data was generated to match these requirements. This approach allowed the models to be fine-tuned for stock price prediction.
- **Early Stopping**: Early stopping was used to prevent overfitting by monitoring the validation loss. Training was halted if the validation loss stopped improving for a specified number of epochs.
- **Model Checkpoints**: Model checkpoints were saved based on the best validation performance, ensuring the best model was retained for evaluation.

Process from Start to Finish

Here is a general process from start to finish in this application:

1) Imported the necessary libraries such numpy, pandas, matplotlib, seaborn, tensorflow, keras, open cv
2) Prepare the Kaggle Dataset
    a) Load the Dataset
    b) Normalize the Data
    c) Adding and Extracting Important Features from the Dataset
3) Phase 1: Developed Custom CNN Model
    a) Created Model Architecture
    b) Trained the model using 5 Epochs
    c) Created Prediction model using CNN model
4) Measured the CNN model
5) Implemented chatbot in Gradio and implemented CNN model in Gradio
6) Phase 2: Integrated Pre-Trained Models
    a) ResNet50
    b) InceptionV3
    c) DenseNet121
7) Ensembled all 3 models using Majority Voting
8) Convert GIF to MP4 so It can be easier to implement on Gradio
9) Measured the pre- trained models

Evaluation Metrics

The models were evaluated using the following metrics:

- **Mean Squared Error (MSE)**: MSE measures the average squared difference between the predicted and

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

  actual values, indicating the model's accuracy.
- **Mean Absolute Error (MAE)**: MAE provides the average absolute difference between predicted and actual values, offering another perspective on prediction accuracy.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **R-squared (R^2)**: R^2 measures the proportion of variance in the dependent variable explained by the independent variables, indicating the model's explanatory power.
- **Accuracy:** Accuracy is a classification metric that measures the proportion of correctly predicted instances out of the total instances.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

- **Loss Function:** The loss function quantifies the difference between the predicted outputs and the actual outputs. It guides the optimization process during model training.
- **ROC Curve (Receiver Operating Characteristic Curve):** The ROC curve plots the true positive rate (sensitivity) against the false positive rate (1-specificity) at various threshold settings.

- **Precision Recall Curve:** The Precision-Recall curve plots precision (the ratio of true positive predictions to all positive predictions) against recall (the ratio of true positive predictions to all actual positives).
- **Confusion Matrix:** A confusion matrix is a table that describes the performance of a classification model. It shows the counts of true positives, false positives, true negatives, and false negatives.

Results

Model Performance

The performance of each model was evaluated based on the test set. The custom CNN model demonstrated a good ability to capture trends but struggled with highly volatile periods. The pre-trained models, particularly InceptionV3, showed superior performance due to their deep feature extraction capabilities. The ensemble model, which averaged the predictions from all models, consistently provided the best results, achieving lower MSE and MAE, and higher R^2 values.

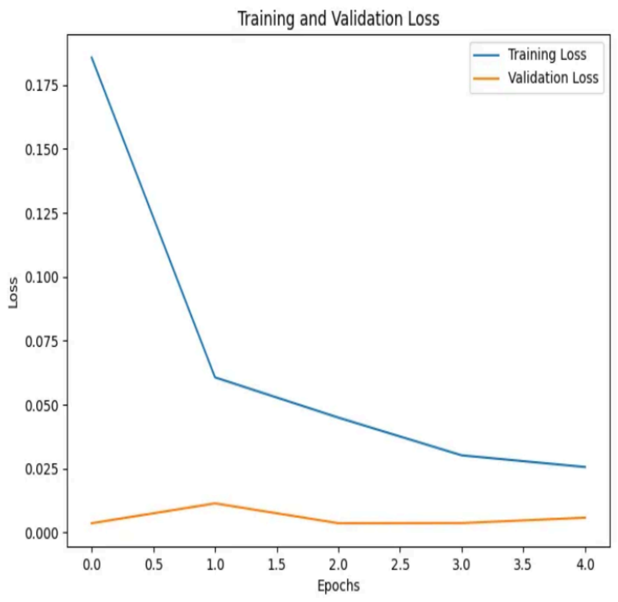
Visualization

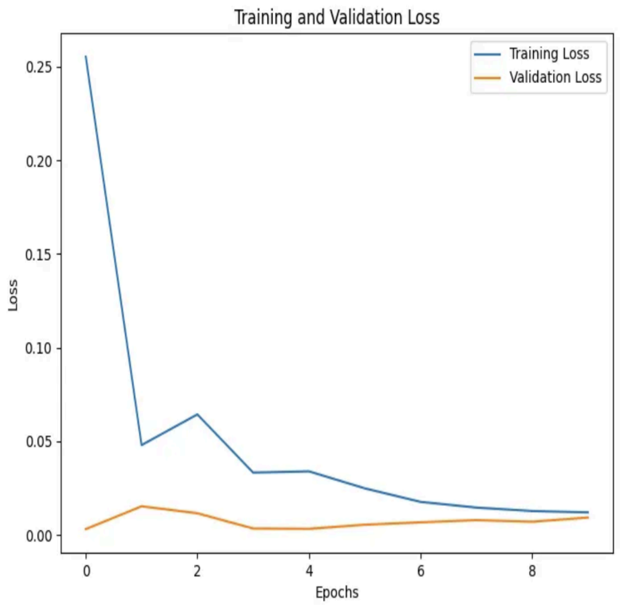
Visualizations included:

- **Loss Curves**: Plots of training and validation loss over epochs, showing the convergence behavior of the models.
- **Predicted vs. Actual Prices**: Plots comparing the predicted stock prices with the actual prices, highlighting the models' accuracy.
- **Ensemble Model Performance**: A specific focus on the ensemble model's predictions, demonstrating its stability and accuracy across different stocks and time periods.

CNN Model Metrics

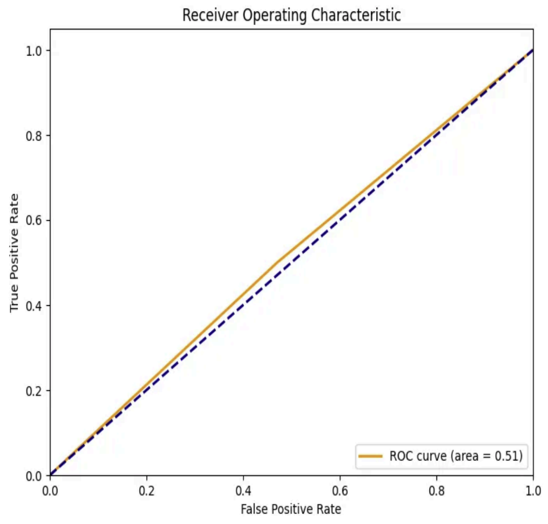
CNN Model Loss

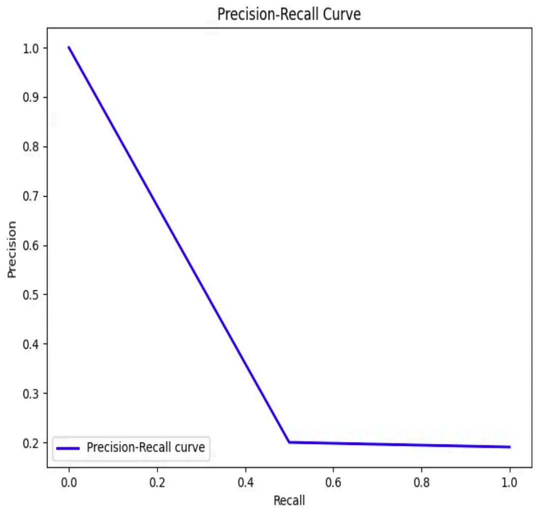


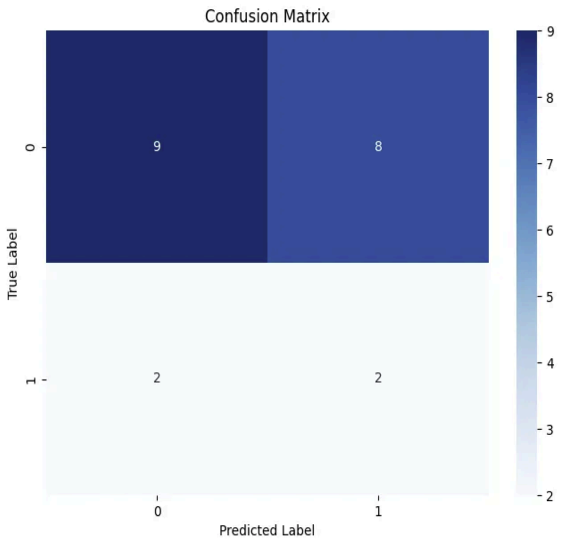


Looking at this graph we can show that with 5 epochs, our CNN model was less accurate compared to running our model with 10 Epochs. We analyzed the more epochs we ran our original model with, the more accurate the model was in helping predict the stocks. But we also analyzed that the more epochs we used, the longer the computation time of the model. We will later be comparing this to the pre-trained models.

CNN Model Other Metrics

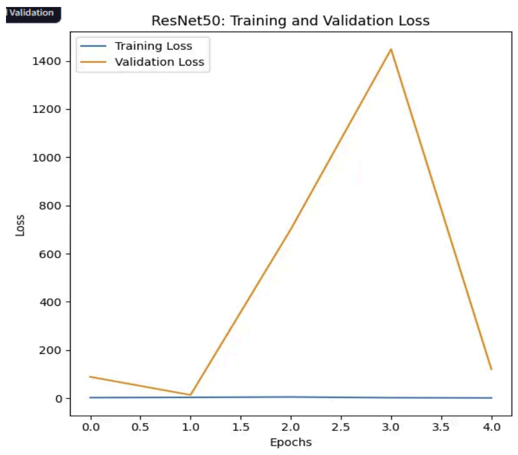


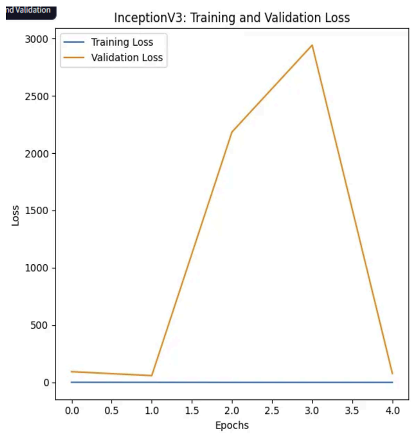


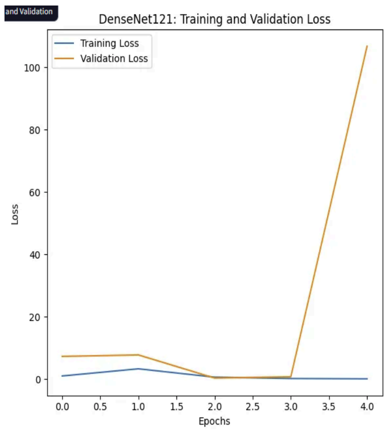


Here we can see that we have evaluated our built CNN model with metrics such as the ROC curve, Precision-Recall curve and Predicted Label. These metrics can help us evaluate how many true positive values we have obtained in our data, false positives, true negatives and false negatives.

Pre-Trained Model Metrics

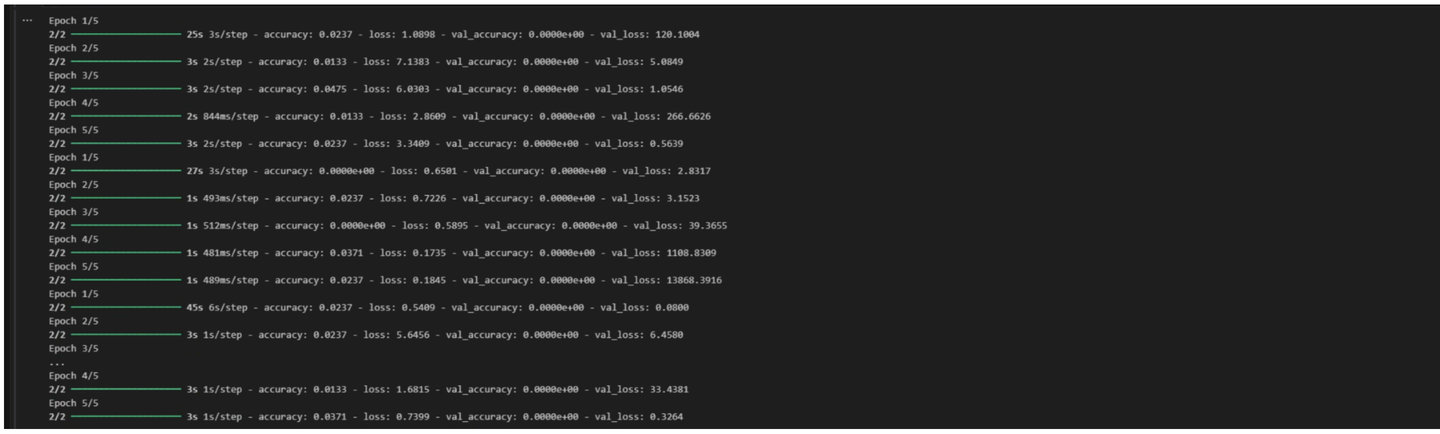




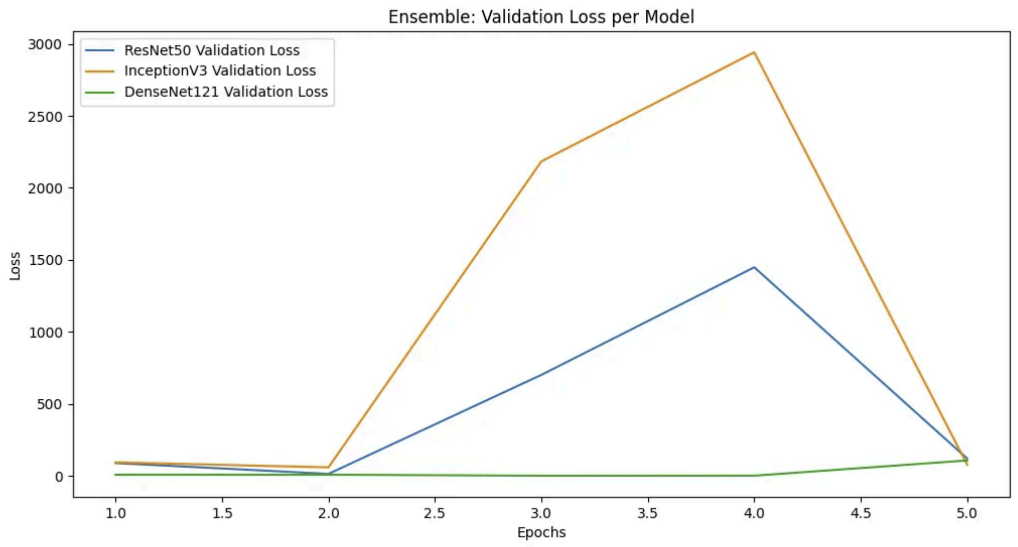


- These loss model graphs help us evaluate the performance of the pre - trained models.
- These models were only trained using 5 epochs because the compilation time was taking too long and thus their graph looks distorted.

Ensemble Model Loss



```
...  Epoch 1/5
     2/2 ━━━━━━━━━━━━━━━━━━━━ 25s 3s/step - accuracy: 0.0237 - loss: 1.0898 - val_accuracy: 0.0000e+00 - val_loss: 120.1004
     Epoch 2/5
     2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 2s/step - accuracy: 0.0133 - loss: 7.1383 - val_accuracy: 0.0000e+00 - val_loss: 5.0849
     Epoch 3/5
     2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 2s/step - accuracy: 0.0475 - loss: 6.0303 - val_accuracy: 0.0000e+00 - val_loss: 1.0546
     Epoch 4/5
     2/2 ━━━━━━━━━━━━━━━━━━━━ 2s 844ms/step - accuracy: 0.0133 - loss: 2.8609 - val_accuracy: 0.0000e+00 - val_loss: 266.6626
     Epoch 5/5
     2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 2s/step - accuracy: 0.0237 - loss: 3.3409 - val_accuracy: 0.0000e+00 - val_loss: 0.5639
     Epoch 1/5
     2/2 ━━━━━━━━━━━━━━━━━━━━ 27s 3s/step - accuracy: 0.0000e+00 - loss: 0.6501 - val_accuracy: 0.0000e+00 - val_loss: 2.8317
     Epoch 2/5
     2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 493ms/step - accuracy: 0.0237 - loss: 0.7226 - val_accuracy: 0.0000e+00 - val_loss: 3.1523
     Epoch 3/5
     2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 512ms/step - accuracy: 0.0000e+00 - loss: 0.5895 - val_accuracy: 0.0000e+00 - val_loss: 39.3655
     Epoch 4/5
     2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 481ms/step - accuracy: 0.0371 - loss: 0.1735 - val_accuracy: 0.0000e+00 - val_loss: 1108.8309
     Epoch 5/5
     2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 489ms/step - accuracy: 0.0237 - loss: 0.1845 - val_accuracy: 0.0000e+00 - val_loss: 13868.3916
     Epoch 1/5
     2/2 ━━━━━━━━━━━━━━━━━━━━ 45s 6s/step - accuracy: 0.0237 - loss: 0.5409 - val_accuracy: 0.0000e+00 - val_loss: 0.0800
     Epoch 2/5
     2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 1s/step - accuracy: 0.0237 - loss: 5.6456 - val_accuracy: 0.0000e+00 - val_loss: 6.4580
     Epoch 3/5
     ...
     Epoch 4/5
     2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 1s/step - accuracy: 0.0133 - loss: 1.6815 - val_accuracy: 0.0000e+00 - val_loss: 33.4381
     Epoch 5/5
     2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 1s/step - accuracy: 0.0371 - loss: 0.7399 - val_accuracy: 0.0000e+00 - val_loss: 0.3264
```

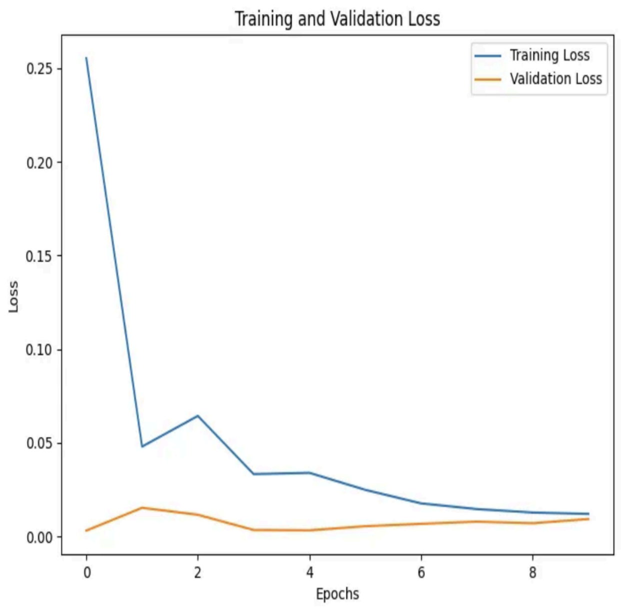


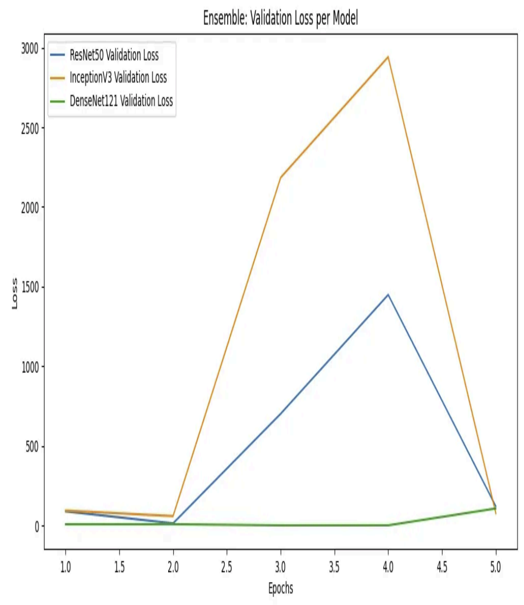
The final ensemble Model was created via an ensemble model using the 3 different pre-trained models using majority voting and transfer learning.

Majority Voting:

Where every model makes a prediction (votes) for each test instance and the final output prediction is the one that receives more than half of the votes. If none of the predictions get more than half of the votes, we may say that the ensemble method could not make a stable prediction for this instance.

Comparison Between Built CNN Model and Pre- Trained Model





- Comparing our original CNN model with the pre-trained models, we can propose that the CNN built model is much more stable and accurate while the pretrained models look inaccurate. There are a variety of reasons which could be caused by this, a few of the reasons can be that for our original CNN model , we used 10 epochs, and for the pre-trained model we used 5 epochs. Another reason is that the pre-trained models are not suited for stock price prediction application at all, as they are more suitable for object detection.

Discussion

Analysis

The ensemble model outperformed individual models, benefiting from the combination of different architectural strengths. The diversity of the models helped in capturing various aspects of the data, leading to more robust predictions. However, challenges such as data quality and the complexity of financial markets still posed significant hurdles.

Challenges

The primary challenges faced during the project included:

- **Data Quality**: Ensuring the data was clean and consistent was crucial. Handling missing values and outliers was a significant challenge.
- **Model Adaptation**: Adapting image-based pre-trained models for time series data required creative solutions, such as synthetic data generation.
- **Computational Resources**: Training deep learning models, especially pre-trained models with millions of parameters, required substantial computational power and resources.

Future Work

Future work could explore the following areas:

- **Incorporation of Additional Data**: Integrating external data sources such as news sentiment, macroeconomic indicators, and social media trends could enhance the model's predictive power.
- **Advanced Architectures**: Exploring other advanced architectures like LSTMs or transformers, which are specifically designed for sequence modeling, could further improve performance.
- **User Interface Enhancements**: Enhancing the Gradio interface to include more interactive features, such as custom date ranges and stock selection, would improve usability.

Conclusion

The project successfully demonstrated the potential of using a combination of custom CNN and pre-trained models for stock price prediction. The ensemble model, in particular, showed significant promise, providing accurate and robust predictions. The implementation of a real-time prediction interface through Gradio highlighted the practical applications of the models. However there was a realization that there are not many pre-trained models trained for predicting stock prices available to the public which was leading to the inaccuracy in this application. The reality is that in order to make a good CNN model for this stock price prediction model, it is required to have a powerful computer to shorten the computation time for running the CNN models. While challenges remain, the results underscore the value of deep learning in financial forecasting, paving the way for future research and development.

References

ResNet and ResNetV2. (2020, December 23). Retrieved from https://keras.io/api/applications/

Keras Core Team. (2022, August 1). InceptionV3. Keras.io. Retrieved from

https://keras.io/api/applications/inceptionv3/

https://keras.io/api/applications/densenet/#densenet121-function

Nehul Agrawal and Rahul Parihar, . YOLOv8s Stock Market future prediction on Live Trading Video Data

[Hugging Face]. Retrieved from https://huggingface.co/foduucom/stockmarket-future-prediction

Nehul Agrawal and Rahul Parihar, . YOLOv8 Stock Market Pattern Detection on Live Trading Video Data

[Hugging Face]. Retrieved from https://huggingface.co/foduucom/stockmarket-pattern-detection-yolov8

Agarwal, A. (2020, October 22). Introduction to Convolution Neural Network. GeeksforGeeks. Retrieved from

https://www.geeksforgeeks.org/introduction-convolution-neural-network/

Agarwal, A. (2020, October 22). Convolutional Neural Network (CNN) in Machine Learning [1,

GeeksforGeeks]. Retrieved from

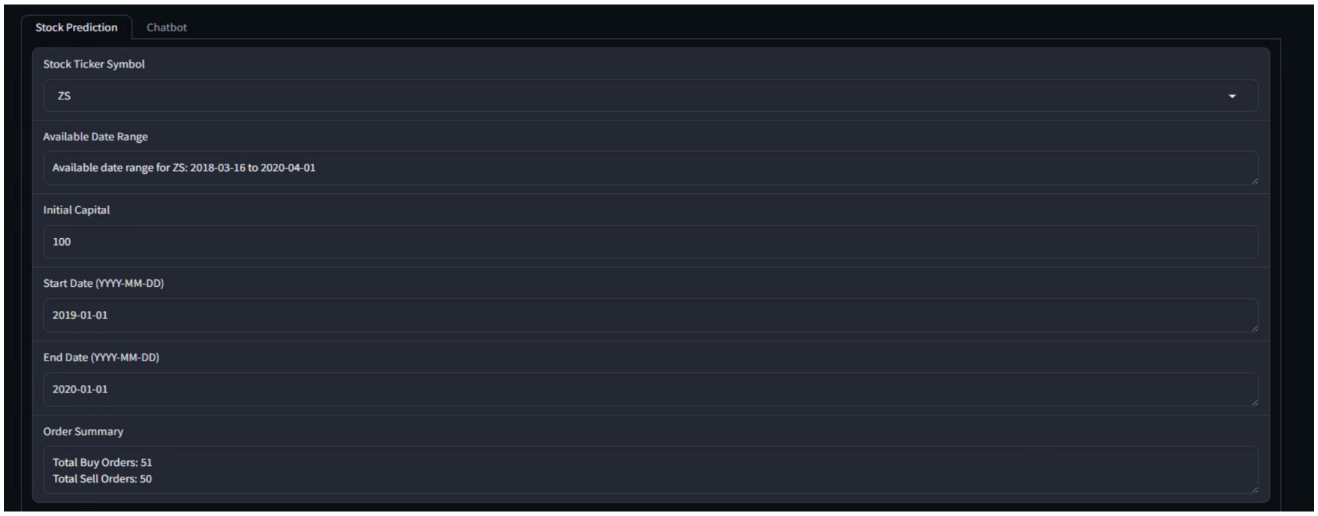
https://www.geeksforgeeks.org/convolutional-neural-network-cnn-in-machine-learning/

Onyshchak, O. (Updated 4 years ago). Stock Market Dataset. [Dataset].

Kaggle.https://www.kaggle.com/datasets/olesbrech/stock-market-dataset

Github: https://github.com/UsamaMo/Stock_Trading_Bot-AI

Appendix



Stock Ticker Symbol

ZS

Available Date Range

Available date range for ZS: 2018-03-16 to 2020-04-01

Initial Capital

100

Start Date (YYYY-MM-DD)

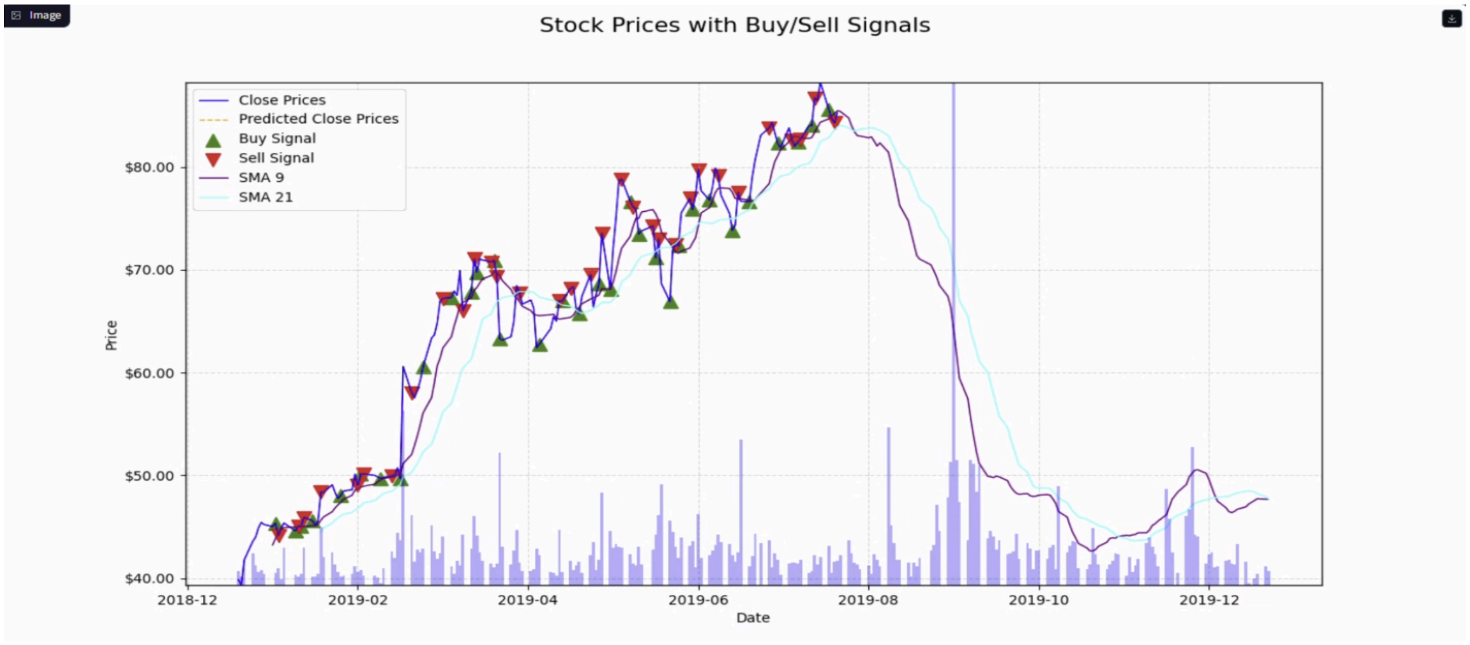
2019-01-01

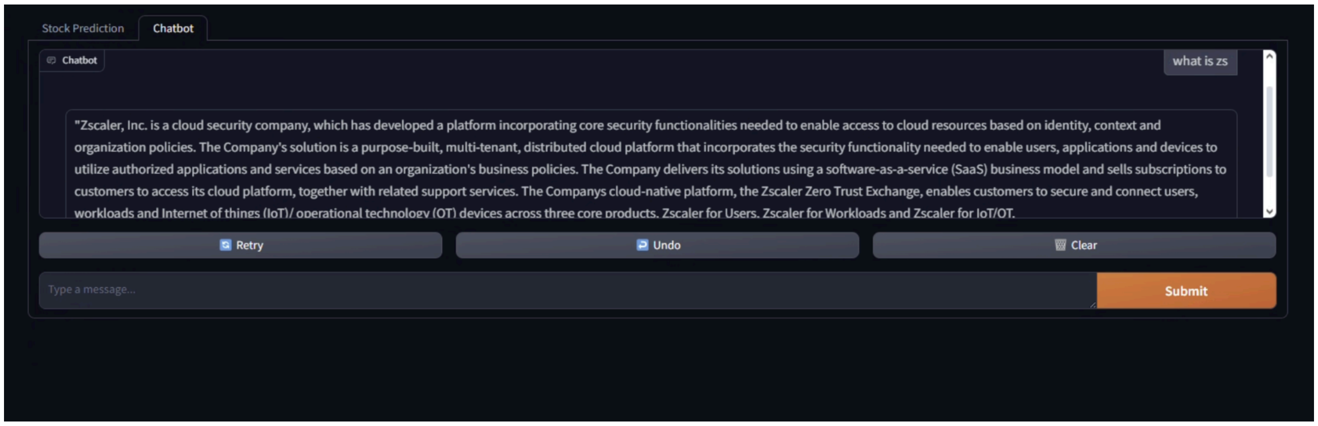
End Date (YYYY-MM-DD)

2020-01-01

Order Summary

Total Buy Orders: 51
Total Sell Orders: 50





what is zs

"Zscaler, Inc. is a cloud security company, which has developed a platform incorporating core security functionalities needed to enable access to cloud resources based on identity, context and organization policies. The Company's solution is a purpose-built, multi-tenant, distributed cloud platform that incorporates the security functionality needed to enable users, applications and devices to utilize authorized applications and services based on an organization's business policies. The Company delivers its solutions using a software-as-a-service (SaaS) business model and sells subscriptions to customers to access its cloud platform, together with related support services. The Companys cloud-native platform, the Zscaler Zero Trust Exchange, enables customers to secure and connect users, workloads and Internet of things (IoT)/ operational technology (OT) devices across three core products, Zscaler for Users, Zscaler for Workloads and Zscaler for IoT/OT.