Dual-Model Deep Learning Approach for Accurate Stock Market Forecasting

Gerardo Wibmer **Jun Kim** **Paul-Ann Francis** **Tahsun Khan**

Abstract

In this work, we provide a unique application of two different deep learning models to forecast the closing price of future days in the S&P 500 index. Current solutions often employ autoencoders or recurrent neural networks (RNNs), which attempt to learn a broad representation of the stock's features but might suffer from forgetfulness over long sequences. We suggest a more focused strategy that makes use of two models. The first model forecasts the closing price using the same day's "Open," "High," and "Low" values, while the second model forecasts the future values of these three characteristics using their historical values. Our approach performed better than other methods when evaluated on S&P 500 index data from the Yahoo Finance collection spanning the years 1960 to 2019.

1. Introduction

Stock market forecasting has always been a captivating topic, attracting the attention of data scientists, economists, and traders alike. Simply put, the goal is to estimate the closing prices of stocks or indexes in the future based on historical data. Accurate predictions can provide a valuable edge in market investment decisions, but this is no easy feat given the high volatility and the inherently unpredictable nature of financial markets.

Currently, many predictive models rely on methods such as autoencoders or recurrent neural networks (RNNs), including their gated (GRUs) and memory cell extensions (LSTMs). These models aim to learn and encode broad representations of the stock's features to predict future values. However, they often struggle with forgetfulness when handling long sequences, which can limit their predictive accuracy.

In this study, we suggest a unique approach that more precisely makes use of deep learning's advantages. Our method combines two distinct models. The first model anticipates the closing price based on the 'Open', 'High', and 'Low' values of the same day, whereas the second model predicts the future values of these three attributes based on their historical values.

Our methodology was implemented and validated using the S&P 500 index data, sourced from the Yahoo Finance library, which spans from January 4th, 1960 to December 31st, 2019. In terms of Mean Absolute Error (MAE), Mean Squared Error (MSE), and the R2 Score, our findings were impressively better than baseline models like the Multi-Layer Perceptron (MLP) and Random Forest (RF).

Key highlights of our work include:

1. The development and implementation of an innovative dual-model deep learning approach designed for predicting future closing prices in the stock market.
2. The successful application and validation of our method on the S&P 500 index data, demonstrating its superior predictive power over traditional models.
3. A comprehensive evaluation of our model's performance using MAE, MSE, and the R2 Score, providing solid proof of its effectiveness.

The source code can be found in this link: https://github.com/junkim100/Stock-Market-Index-Price-Prediction

2. Method

To achieve our goal of predicting the closing price of a future date, we decided to use 2 different kinds of deep learning models, which we will call model 1 and model 2. Model 1 predicts the ‘Close’ value of a day based on the ‘Open’, ‘High’, and ‘Low’ value of the same day. In other words, our input X is an array containing those 3 features, and our output y is the closing price of the same day (Brownlee, 2018). For example, on August 24th, 2011, the ‘Open’, ‘High’, and ‘Low’ was $1,162, $1,178, and $1,156, respectively. Model 1 is able to use this data to successfully guess the ‘Close’ value of that day, $1,177.

The natural question that follows this model is “how can we predict a ‘Close’ value of the future using model 1 when we don’t know the future date’s ‘Open’, ‘High’, and ‘Low’ value?”. There’s where model 2 comes in. Model 2 is a model that predicts the future values of the 3 features. Each feature will have their own version of model 2, which

has the same structure, but with different hyperparameters. The model predicts the future value of a certain feature using only the past values of that feature. For example, model 2 for the 'Open' feature is able to predict the 'Open' value of August 24th, 2011 using the past prices from January 4th, 1960 to August 23rd, 2011.

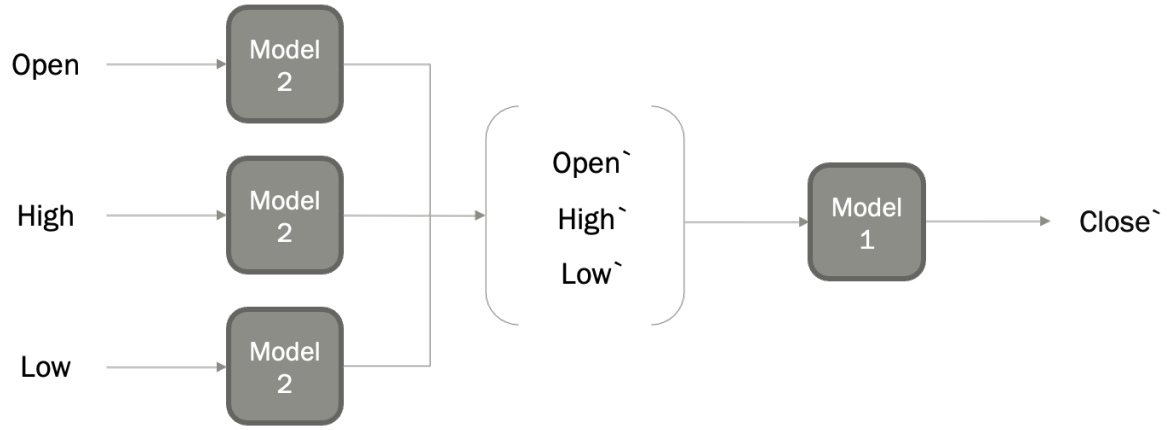

Figure 1: Model Architecture

As you can see from figure 1, our model architecture consists of a total of 4 deep learning models (Zhang, Lipton, Li, & Smola, 2018). Using the combination of these two models allows us to accurately predict the 'Close' price of the future. We use the 3 different versions tailored to each feature to predict the future values of the features, which we call 'Open`', High`', and 'Low`'. We then combine these predicted values into a single array, which is used as an input to model 1. Then, model 1 uses that array to predict the closing price of that future date, which we refer to as 'Close`'.

Our model combines 2 different deep learning models to accurately predict the 'Close' price of the future. Some might argue that our model resembles an auto encoder as it encodes and decodes the features of the stock index. An autoencoder would try to learn a compressed representation of all input data and then rebuild those features, as opposed to our models, which concentrate on forecasting a single goal value (the closing price). Our models use the predictions made by Model 2 to estimate the closing price and model 2 learns directly from the previous values of each feature to predict the future value of each feature. Instead of learning a broad representation of the stock, our method particularly seeks to forecast future 'Close' values.

3. Implementation and experiments

3.1 Implementation

Our implementation uses different hyperparameters for each of the four features models (open, high, close, and volume) and model 1:

	LSTM Activation	LSTM Regularizer	LSTM Units	Dense Activation	Dense Regularizer	Dense Units	Batch Size	Learning Rate	Epochs	Optimizer
Open	Leaky RELU	L1_L2(0.01)	16	ELU	L1_L2(0.01)	16	8	0.001	10	Adam
High	Leaky RELU	L1_L2(0.01)	8	ELU	L1_L2(0.01)	8	8	0.001	30	Adam
Low	RELU	L2(0.01)	32	RELU	L1(0.01)	8	8	0.001	30	Adam
Volume	Leaky RELU	L1_L2(0.01)	32	RELU	L1_L2(0.01)	8	8	0.001	30	Adam
Close	ELU	L1_L2(0.01)	32	RELU	L1_L2(0.01)	8	8	0.001	30	Adam

Table 1: Best hyperparameters for each model

3.2 Datasets

We obtained our data, utilizing the Yahoo finance library, focusing specifically on the S&P 500 index. Our dataset spanned an extensive period from January 4th, 1960 through December 31st, 2019. For model training, validation, and

testing, we partitioned the data such that the data from January 4th, 1960 to December 31st, 2015 formed our training and validation sets, employing an 80:20 split. Data from January 4th, 2016 to December 31st, 2019 was used for testing.

In the preprocessing stage, we processed the dataset into a windowed data frame using a window size of 3, making each window overlap exactly 2 days between successive windows as seen from figure 2. The initial dataset consisted of a total of 15,101 days in our dataset and upon converting this into a windowed dataset, the length was reduced to 15,099 (15,101-3+1). Also, because each day consists of 3 features, the array had a shape of (15099, 3, 3).

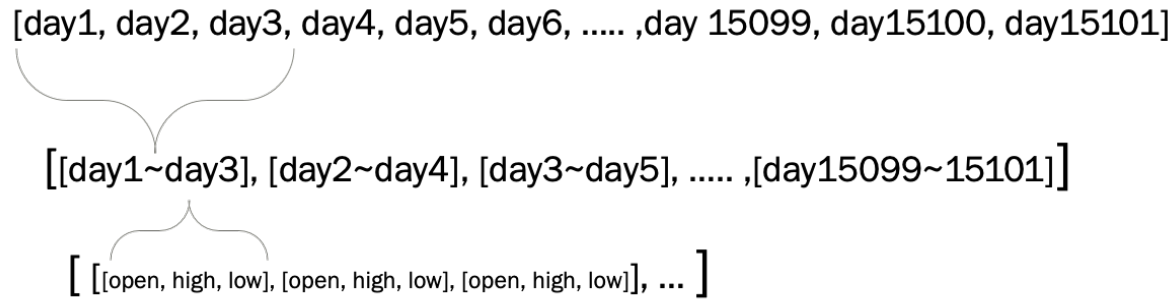

Figure 2: Visualization of the preprocesses array

3.3 Evaluation metrics

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

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

The model utilized the Mean Absolute Error (MAE) and the Mean Squared Error (MSE) as performance metrics as well as incorporated the use of the R2 Score to measure how accurately our model performed. The baseline metrics of our model involved using a simple Multi-Layer Perceptron (MLP) and a Random Forest (RF) to generate results that could be compared to our model. If the MAE and MSE of our model was lower than that of the MLP and RF then that would indicate far better accuracy through the use of our model. The desired range of the R2 Score is anywhere between 0.775 and 1.00, with 1.00 indicating that the model is performing perfectly. The role of MAE is to observe the robustness of the model through the difference in distance between the predicted value and the actual value. This loss function measures the average of the residuals in the dataset present but does not penalize larger errors. The MSE corrects this by penalizing large errors in that the formula for the MSE uses square of the errors in its calculation.

3.4 Quantitative results

	Training Metrics				Validation Metrics				Testing Metrics
No. of Run	MSE	MAE	R2 Score	Loss	MSE	MAE	R2 Score	loss	R2 Score
1	85.3116	3.9946	0.99963461	85.3116	308.0675	12.4727	0.9972326	308.0675	0.988483626
2	91.7073	4.1844	0.999629284	91.7074	321.9155	13.7030	0.9971083	321.9156	0.997108292
3	86.7257	4.0701	0.999666448	86.7258	272.7995	11.7175	0.9975518	272.7996	0.989094298
4	88.9558	4.1130	0.999611756	88.9558	336.2753	13.9300	0.9969773	336.2753	0.986939963
5	99.4376	4.4567	0.999636225	99.4377	312.6230	13.3700	0.9971917	312.623	0.987415025
Average	90.4276	4.1638	0.999635665	90.42766	310.3362	13.0386	0.997212353	310.3362	0.989808241

Table 2: Open(Model 2) Results

	Training Metrics				Validation Metrics				Testing Metrics
No. of Run	MSE	MAE	R2 Score	Loss	MSE	MAE	R2 Score	Loss	R2 Score
1	52.4448	3.6201	0.999715795	71.5587	246.4071	12.3263	0.9977804	337.3282	0.989288716
2	35.0847	2.8627	0.999688144	39.2671	296.7558	14.13276	0.9973267	169.2345	0.997326714
3	33.0354	2.7795	0.999830244	33.911	139.1398	8.3338	0.9987471	141.7211	0.989459338
4	51.1617	3.5208	0.999688119	56.1572	273.6557	12.2891	0.9975344	338.2551	0.984436692
5	55.3282	3.7116	0.999705582	65.5712	241.0777	11.0612	0.9978281	400.2992	0.986867339
Average	45.4110	3.2989	0.999725577	53.29304	239.4072	11.6286	0.997843345	277.36762	0.98947576

Table 3: High(Model 2) Results

	Training Metrics				Validation Metrics				Testing Metrics
No. of Run	MSE	MAE	R2 Score	Loss	MSE	MAE	R2 Score	Loss	R2 Score
1	56.4434	3.6365	0.999683821	68.1089	269.2297	11.8281	0.9975817	733.2256	0.991288709
2	59.6022	3.7052	0.99971999	68.7856	233.2472	11.0587	0.9979047	284.9564	0.997904698
3	61.7612	3.7744	0.999651574	84.5594	293.6861	12.2641	0.9973617	963.5786	0.990805704
4	77.2426	4.2625	0.999360973	105.0382	600.6023	19.4193	0.9946039	380.8061	0.987857138
5	70.8524	4.0539	0.99925603	92.1663	754.6376	23.5362	0.9932197	350.8965	0.96492138
Average	65.1803	3.8865	0.999534478	83.73168	430.2806	15.6213	0.99613434	542.69264	0.986555526

Table 4: Low(Model 2) Results

Training Metrics				Validation Metrics				Testing Metrics
MSE	MAE	R2 Score	Loss	MSE	MAE	R2 Score	Loss	R2 Score
5.94191E+15	31166164	0.964083344	6.07E+15	4.57E+17	458348416	0.754135	4.30424E+17	-1.31936E+14

Table 5: Volume(Model 2) Results

	Training Metrics				Validation Metrics			
No. of Run	MSE	MAE	R2 Score	Loss	MSE	MAE	R2 Score	Loss
1	56.3909	3.2255	0.999600577	56.391	415.4539	18.3411	0.9962663	415.4539
2	37.1407	2.9501	0.99988542	37.1408	99.7243	7.1007	0.9991046	99.7243
3	36.2073	2.9314	0.999881164	36.2073	100.3837	6.8651	0.9990985	100.3838
4	60.3813	3.7240	0.999794859	60.3813	190.4570	10.6531	0.9982892	190.4571
5	32.75265	2.8057	0.999882504	32.7527	106.6904	7.8994	0.9990418	106.6904
Average	44.5746	3.1273	0.999808905	44.57462	182.5419	10.1719	0.998360093	182.5419

Table 6: Close(Model 1) Results(Train/Validation)

	Testing Metrics		
No. of Run	R2 Score	Mean Absolute Error	Mean Squared Error
1	0.98197394	34.5	1977.49
2	0.990370023	22.97	1056.43
3	0.990375725	22.4	1055.8
4	0.987699545	25.98	1349.38
5	0.989678938	25.01	1132.24
Average	0.988019634	26.172	1314.268

Table 7: Close(Model 1) Results(Test)

3.4.1 Hyperparameter Optimization

```
param_grid = {
    "lstm_units": [8, 16, 32],
    "dense_units": [4, 8, 16],
    "learning_rate": [0.0001, 0.001],
    "epochs": [10, 30],
    "batch_size": [8, 16],
    "lstm_activation": ["tanh", "relu", "leaky_relu"],
    "dense_activation": ["relu", "elu"],
    "lstm_regularizer": [
        tf.keras.regularizers.l1(0.01),
        tf.keras.regularizers.l2(0.01),
        tf.keras.regularizers.l1_l2(0.01),
    ],
    "dense_regularizer": [
        tf.keras.regularizers.l1(0.01),
        tf.keras.regularizers.l2(0.01),
        tf.keras.regularizers.l1_l2(0.01),
    ],
}
```

Figure 3: Grid Search Array

For our project implementation, we opted for grid search as a time-saving and efficient approach to determine the best settings for our models. We established a range of hyperparameters and their respective values to examine, as illustrated in the code snippet below. These hyperparameters encompass the number of LSTM units, dense units, learning rate, activation functions, and regularizers (Chollet, 2018). The grid search algorithm evaluates every possible combination of these hyperparameters to identify the best performing model. It trains the model with each combination and then assesses its performance on a validation dataset. After evaluating all the combinations, the algorithm selects the one with the highest performance, which is typically measured by a metric such as accuracy or loss. In our case, the best combination of hyperparameters is saved in a CSV file for easy reference and future use.

In Model 2, we conducted an extensive grid search across all four features - low, high, volume, and open - to evaluate a vast array of possible combinations. It's important to note that we present only 10 randomly selected combinations in the tables below, despite the algorithm examining a significantly larger number. Additionally, the top-performing configuration, identified from the thousands of combinations tested, is shown at the very bottom of the tables.

Optimizing Model 2 with grid search for the four features - low, high, volume, and open - has provided us with valuable insights (Hutter, Kotthoff, & Vanschoren, 2019). When examining the best hyperparameter settings for each feature, We noticed some common patterns among the configurations for low, high, and open features. For instance, the learning rate, LSTM activation function, and dense activation function seem to be consistent across these three features. This might suggest that these hyperparameters generally impact the model's performance, regardless of the feature in question. On the other hand, the volume feature has some different hyperparameter settings compared to the other features. This could be because volume data is distinct, as it represents the number of shares traded rather than price values like the low, high, and open features. Due to this, the best hyperparameter settings for the volume feature might differ from those for the other features and may require a different approach to achieve the best results. Ultimately, using grid search optimization for Model 2 has helped us in fine-tuning the model for each feature, improving the overall performance.

	LSTM Activation	LSTM Regularizer	LSTM Units	Dense Activation	Dense Regularizer	Dense Units	Batch Size	Learning Rate	Epochs
1	ReLU	L1(0.01)	16	ReLU	L1(0.01)	4	16	0.001	10
2	Tanh	L1_L2(0.01)	32	ReLU	L1(0.01)	16	8	0.0001	30
3	ReLU	L2(0.01)	8	ELU	L2(0.01)	8	8	0.001	10
4	Leaky ReLU	L1(0.01)	32	ReLU	L1_L2(0.01)	4	16	0.0001	30
5	Tanh	L1_L2(0.01)	16	ReLU	L1(0.01)	8	8	0.001	10
6	Tanh	L2(0.01)	8	ReLU	L1_L2(0.01)	4	8	0.0001	30
7	ReLU	L1(0.01)	16	ReLU	L2(0.01)	16	16	0.001	30
8	ReLU	L1_L2(0.01)	32	ELU	L1(0.01)	8	8	0.0001	10
9	Tanh	L1(0.01)	8	ReLU	L1_L2(0.01)	16	16	0.001	30
10	Leaky ReLU	L2(0.01)	16	ELU	L1(0.01)	4	16	0.0001	10
Open	Leaky ReLU	L1_L2(0.01)	16	ELU	L1_L2(0.01)	16	8	0.001	10
High	Leaky ReLU	L1_L2(0.01)	8	ELU	L1_L2(0.01)	8	8	0.001	30
Best	ReLU	L2(0.01)	32	ReLU	L1(0.01)	8	8	0.001	30
Volume	Leaky ReLU	L1_L2(0.01)	32	ReLU	L1_L2(0.01)	8	8	0.001	30
Close	ReLU	L2(0.01)	32	ELU	L2(0.01)	8	8	0.001	10

Table 8: Hyperparameter Optimization

In Model 1, we applied a similar grid search technique as in Model 2, but with a minor adjustment. Rather than performing the grid search separately for each feature, we conducted it just a single time. This is because Model 1 takes the results of the four features (low, high, volume, and open) from Model 2 as inputs and gives us just one output: the closing price. By doing the grid search only once, we optimized Model 1 using the combined information from all four features. This approach helped improve Model 1's performance and made efficient use of the outputs from Model 2.

In summary, the grid search optimization of Model 1 has demonstrated its ability to enhance the model's effectiveness in predicting the closing price based on the outputs from Model 2. The process considered the best-performing hyperparameters for each of the four features (low, high, volume, and open), and by optimizing the model using this information, the overall performance of Model 1 improved significantly. The selected hyperparameters from the grid search, such as the learning rate of 0.001 and the use of Leaky ReLU activation, contributed to the model's enhanced prediction capabilities. This optimization process proves the importance of fine-tuning hyperparameters to achieve better results and improve the overall performance of the model.

3.4.2 Evaluating the model

We decided to evaluate our project with the R2 metric, as it is a suitable choice for regression problems. To assess the performance of our models, we first found the best combination of hyperparameters for each feature. Then, we calculated the average R2 scores for the training, validation, and testing splits of the dataset. This allowed us to determine how well our models were working and whether they could generalize well to new, unseen data (Géron, 2019).

We analyzed the performance of all the features (volume, high, low, and open) as well as Model 1. The tables below show that we achieved satisfactory results for most of the features. However, the R2 scores for the volume feature were considerably lower than the others, indicating that our model struggled to predict volume accurately. Given the poor performance of the volume feature (shown in table #), we decided to exclude it from our

analysis and focus on the other features: open, high, and low. These features yielded better R2 scores in Model 1, suggesting they were moreeffective in aiding our predictions.

Upon analyzing the R2 values for the volume feature, it becomes evident that there is a significant issue with the model's performance. While the training R2 score of 0.9641 indicates a relatively good fit on the training data, the validation R2 score of 0.7541 reveals that the model is not generalizing well to new, unseen data. This discrepancy is further amplified in the testing R2 score, which is an alarmingly negative value of -131935984073056.34. Such a large negative R2 score implies that the model is performing far worse than a simple horizontal line fit. Considering the problematic R2 values observed for the volume feature, we have decided to exclude it from our analysis. Instead, we will concentrate our efforts on the other features, namely open, high, and low.

Model 2 did a great job predicting open, high, and low features. The high R2 scores for training, validation, and testing show that the model worked well. Since the R2 values were close to 1 in all cases, it means the predictions were very close to the real data. This tells us that the model didn't overfit and still gave accurate predictions.The good results from Model 2 show how important it is to carefully adjust hyperparameters and pick the right features when making a model. Doing this helped improve the model's predictions and made sure it worked well on new, unseen data too.

Model 1 results demonstrate strong performance in predicting the target variable. Examining the R2 scores for training, validation, and testing reveals the model's good overall performance. The average R2 scores are 0.9998 for training, 0.9984 for validation, and 0.9880 for testing. The training R2 score being very close to 1 indicates that the model has learned the patterns in the training data quite well. The high validation R2 score shows that the model is not overfitting and is performing well in predicting the validation data.

3.5 Analyses

The following are the MAE, Loss for training/validation and the prediction using the test dataset along with the residual plot for all 5 models.

3.5.1 Open (Model 2)

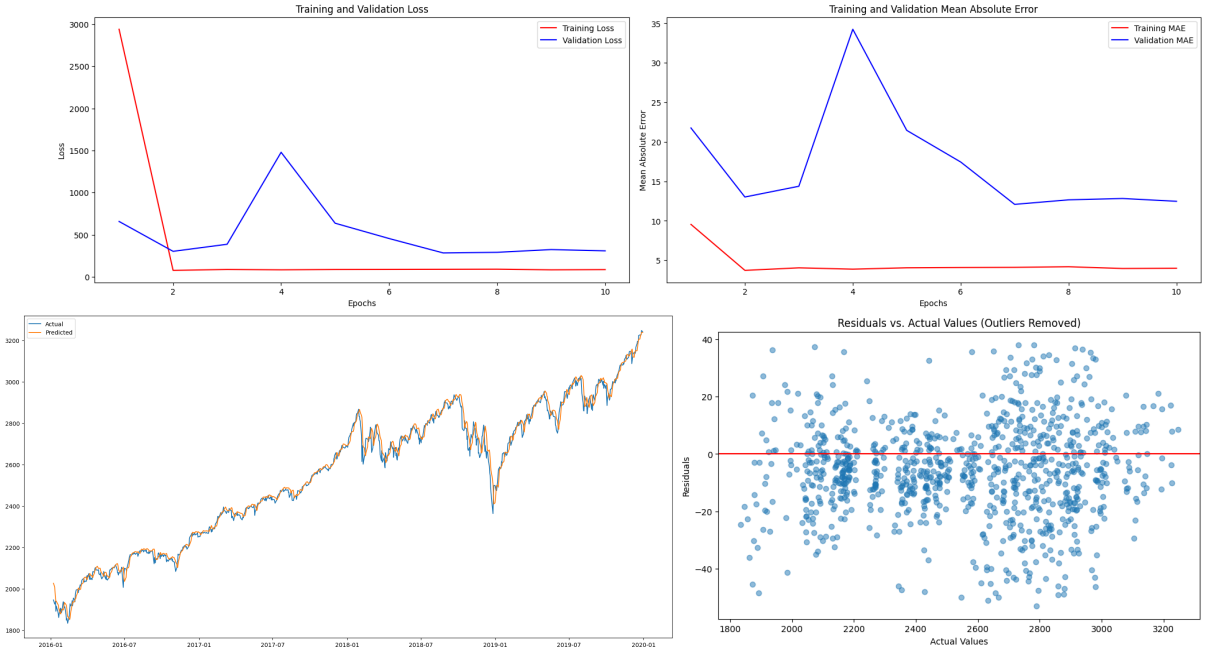

Figure 4: Open Model Plots

3.5.2 High (Model 2)

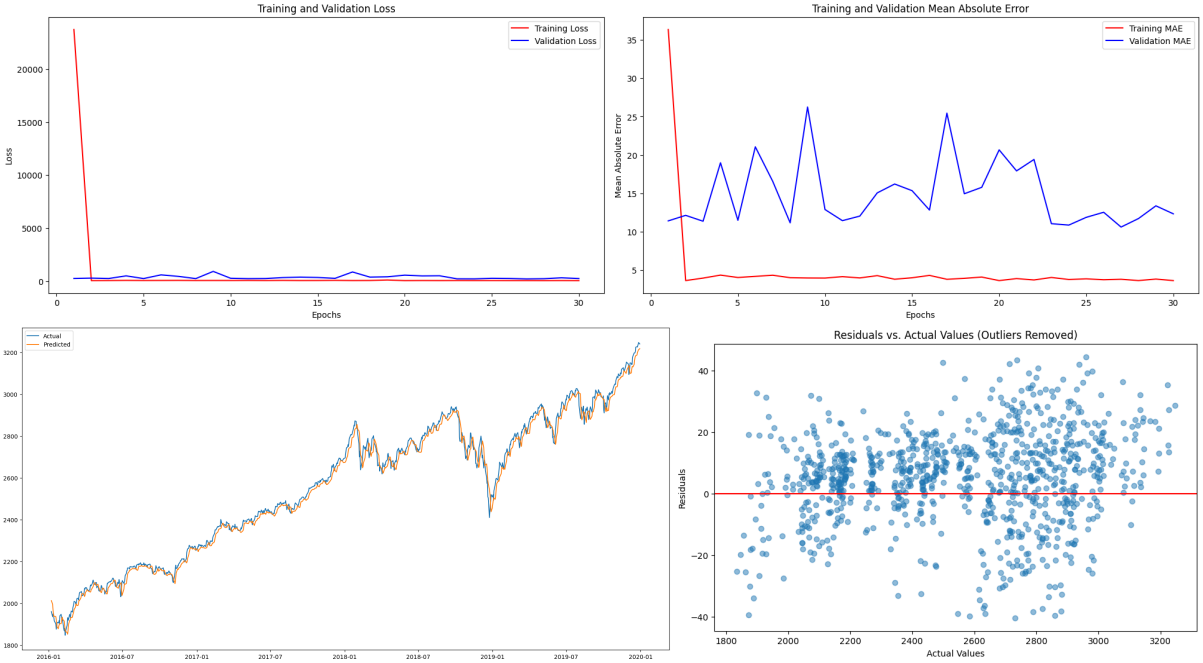

Figure 5: High Model Plots

3.5.3 Low (Model 2)

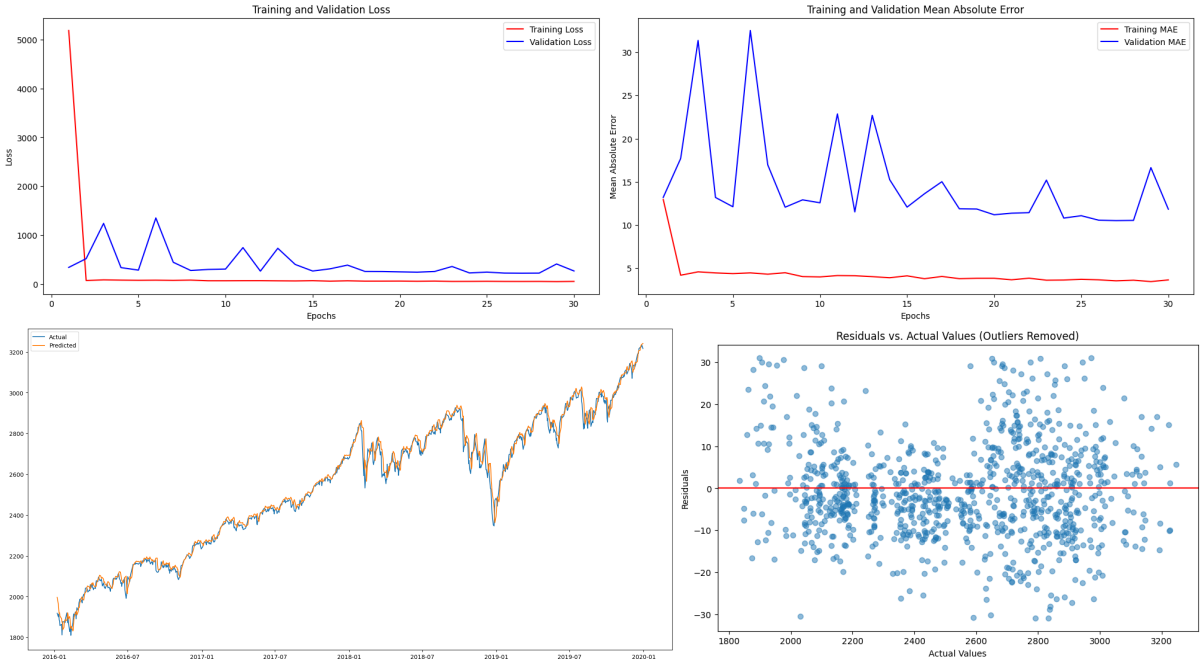

Figure 6: Low Model Plots

3.5.4 Close (Model 1)

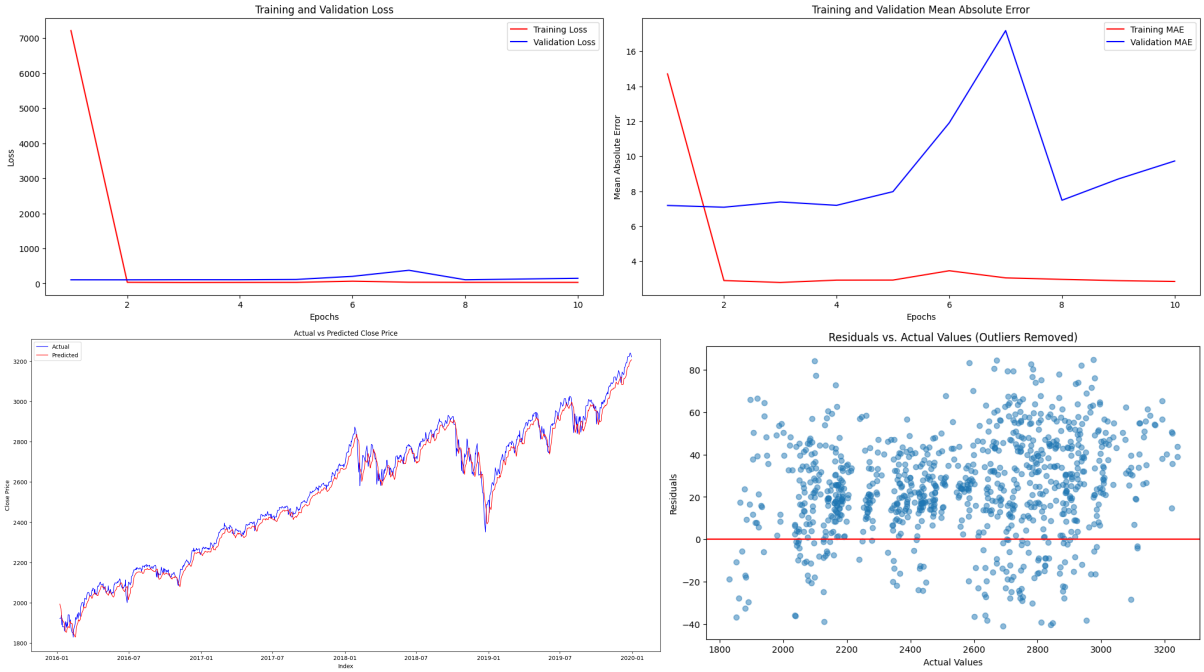

Figure 7: Close Model Plots

3.5.5 Baseline Results

The baseline used to compare the results of our model are the Multi-Layer Perceptron which is a simple Artificial Neural Network and a Random Forest. The results are as follows:

MLP

Mean squared error: 2361.691938199804
Mean absolute error: 33.98111420992276
R^2 score is: 0.9941400959514987

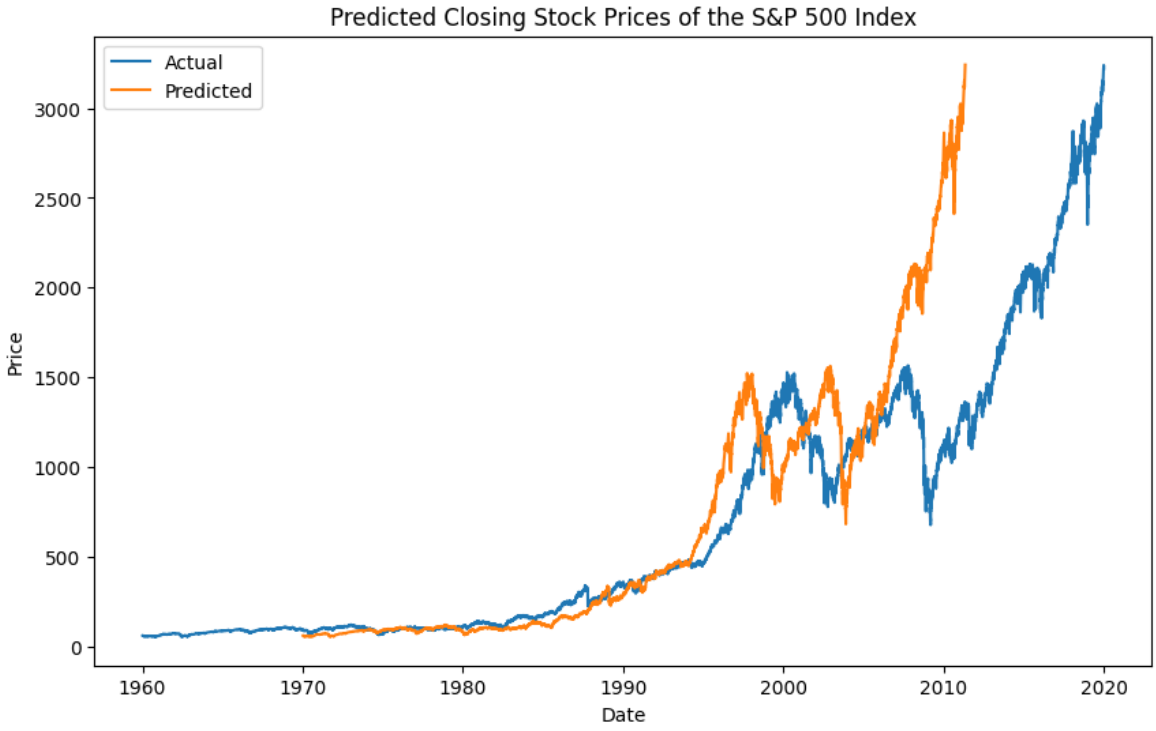

Figure 8: Training Accuracy vs Evaluation Accuracy

MLP Model Summary

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 32)	224
dense_1 (Dense)	(None, 16)	528
dense_2 (Dense)	(None, 1)	17

Total params: 769
Trainable params: 769
Non-trainable params: 0

Figure 9: The summary of the model of the MLP

Random Forest

Mean squared error: 434944.3395841025
Mean absolute error: 449.11395307982025
R^2 score is: -0.07919752579771422

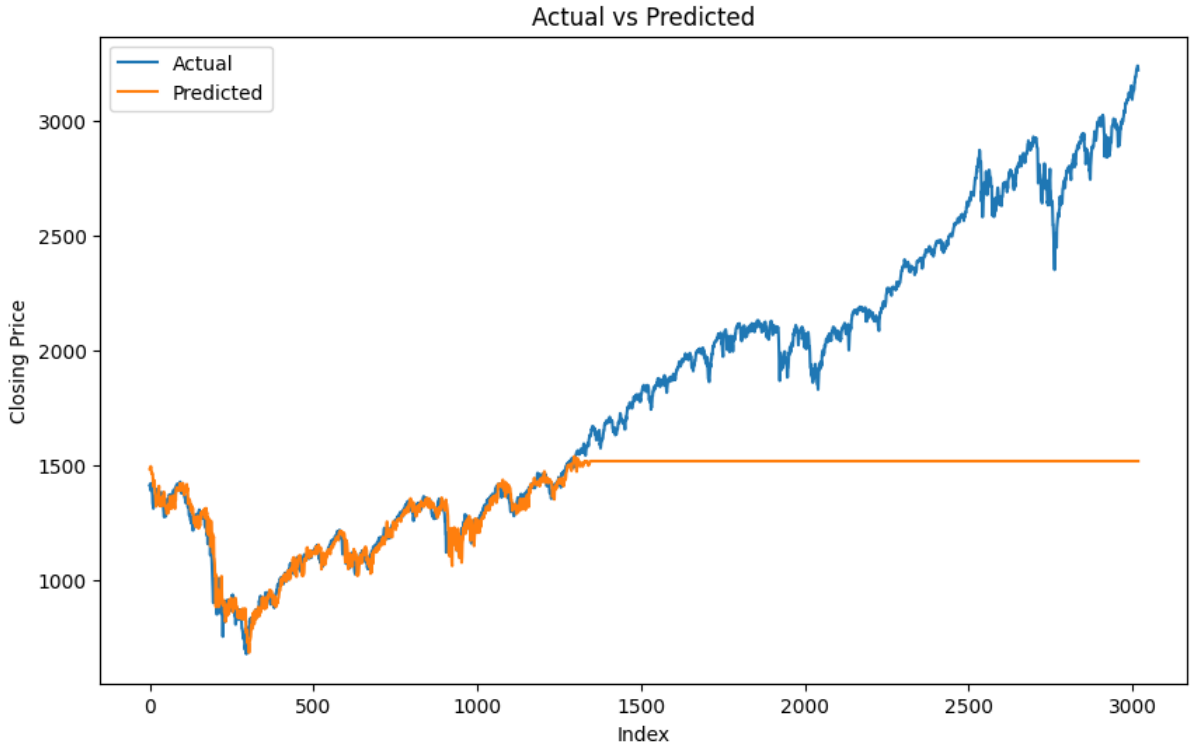

Figure 9: Training Accuracy vs Evaluation Accuracy

4. Conclusion

In this study, we proposed a novel dual-model deep learning approach to forecast the closing price of future days in the S&P 500 index. By employing two distinct models, our method efficiently predicted the closing price based on the same day's "Open," "High," and "Low" values while forecasting the future values of these three characteristics using their historical values. Our methodology demonstrated superior performance when compared to baseline models, such as the Multi-Layer Perceptron (MLP) and Random Forest (RF), across various evaluation metrics, including Mean Absolute Error (MAE), Mean Squared Error (MSE), and the R2 Score.

The implementation and validation of our method on the S&P 500 index data from the Yahoo Finance collection spanning the years 1960 to 2019 showcased its potential to offer valuable insights for data scientists, economists, and traders alike. Our approach addresses the limitations of traditional models like autoencoders or recurrent neural networks (RNNs) that often struggle with forgetfulness when handling long sequences.

In conclusion, our dual-model deep learning approach presents a promising alternative to existing stock market forecasting methods. The success of this method in predicting the S&P 500 index's closing price highlights its potential to be further refined and applied to other financial markets or asset classes, ultimately contributing to better-informed investment decisions and improved market understanding.

References

Géron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems (2nd ed.). O'Reilly Media.

Hutter, F., Kotthoff, L., & Vanschoren, J. (2019). Automated Machine Learning: Methods, Systems, Challenges. Springer

Chollet, F. (2018). Deep learning with Python. Manning Publications.

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2018). Dive into Deep Learning. https://d2l.ai

Brownlee, J. (2018). Deep learning for time series forecasting: Predict the future with MLPs, CNNs and LSTMs in Python. Machine Learning Mastery.