# *Interday Stock Predictions via Algorithmic Trading: A Comparative Analysis of Machine Learning Approaches*

## I. Abstract

This project aims to explore how the predictive power of machine learning models can reap financial benefits for investors who trade based on future price prediction. Investors in the stock market can maximize their profit by buying or selling their investment if they can determine when to enter and exit a position. However, the volatile, stochastic nature of the market makes it difficult to forecast stock results. With increasing popularity, machine learning algorithms have been used to predict financial markets with some degree of success. To this end, we implemented multiple machine learning algorithms including: support vector machines (SVM), support vector regression (SVR), Long-Short Term memory (LSTM), Convolutional Neural Networks (CNN), and combinations of the above algorithms to determine the next-day trading actions and outcomes. The construction of the predictive models is based on historical information of the index extracted through Yahoo finance. Using the predicted results from our models to generate portfolio value over time, a combination of CNN+LSTM algorithm had the best performance.

## II. Introduction

Due to the volatile and non-stationary nature of the stock market, experienced traders cannot confidently predict the future movements of the market; Therefore, by exploiting advancements in the field of machine learning, there has been increased reliance on machine learning algorithms as a scientific and intelligent mechanism to predict future market movements. However, the prevailing question remains to be "what ML models are capable of predicting time series that are volatile, non-stationary, non-linear, and high noise?" According to a paper published by Qiu et al.[1], researchers have tested a variety of Machine Learning algorithms such as Support Vector Machine (SVM), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and different variations of the above algorithms, for example Long Short-term Memory (LSTM). LSTM is a variation of RNN that solves exponential decay or explosion by adding Constant Error Carousel to each cell such that the training time lags for LSTM can be larger than that of regular RNN [2]. Furthermore, LSTM's unique storage structures are more capable of predicting random price series from the stock market. Long Short-term Memory (LSTM) is commonly used in speech recognition and text processing [1]; thus, this project aims to investigate and implement vanilla LSTM (the simple one-layer LSTM) and other variations of LSTM so that we can compare their results with results from other type of Machine Learning algorithms (SVM, CNN). Model architectures will be explained in the subsequent sections. For this project, we mainly use models from keras and sklearn.

This project has two main tasks: regression and classification. For regression, we aim to predict the next day's closing price based on historical closing price information; we also explored adding additional parameters into the Long Short-term Memory model such as daily Exponential Moving Average (EMA) and daily Bollinger Bandwidth for the training window. For classification, we aim to predict the general price movement (increase or decrease) of the next day based on historical closing price information.

Long Short-term Memory and its variations are extremely powerful Machine Learning algorithms, so tuning the parameters was one of the main challenges this project faced; these parameters include number of days included in the training window, size of layers, batch size, epochs, learning rate, and kernel initialization method.  Empirical experiments were run to discover the optimal parameters, and the experiments plot will be discussed in the later section.

## III.1 Methods – Regression

*iii.1.a* <u>*Dataset and Features*</u>

The dataset was retrieved using Yahoo Finance's "*yfinance*" API, which included the Apple stock price from January 2000 to July 2020. The raw dataset included 5 different columns and 5219 rows; there is approximately one row per day in this time frame, although certain gaps exist between days in the dataset. The following are the 5 columns included in the dataset and their definitions:

- *Open*: price of the stock when the market opens on that day
- *Close:* price of the stock when the market closes on that day
- *High:* highest price of the stock during the trading period on that day
- *Low:* lowest price of the stock during the trading period on that day
- *Volume:* number of shares bought and sold on that day

We also included another column to our dataset which we called *% Increase.* This was meant to represent the percent increase of the stock price in a given day (or any given time frame). One way in which we were able to visualize trends in the data before creating a model was the use of a bar chart. The static version is shown below, where green indicates upwards trends, and red indicates downwards trends. The dynamic version can be found in the source code, where the user is able to zoom in on any given time frame.

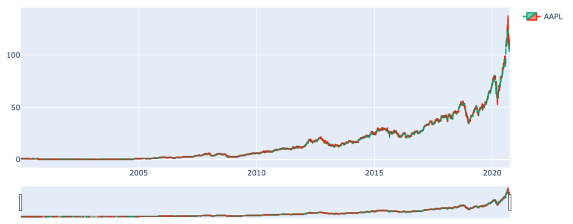


Figure iii.1. Plot of Raw Data.

In order to compare multivariate LSTM model with single variate LSTM model, we have also calculated the following financial indicators to incorporate into our multivariate model in addition to the closing price:

- *Bollinger Bandwidth***:**

$\alpha \times 2 \times StandardDeviation(Pt, t- \ 10\ )$ where αis scalar where we set as 1.5 and $Pt, t- \ 10$ is the price for the past 10 days

- *Exponential Moving Average***:**

$(Pt \times \frac{C}{1+TrainingWindow}) + (EMAt - 1 \times (1 - \frac{C}{1+Training\ Window}))$

where C is a smoothing constant

### *iii.1.b* <u>*Dataset Preprocessing*</u>

"Close" price data is extracted from the dataset. Training windows and forward windows are selected for each model. The data dimension is decided based on the sizes of the training window and forward window and transformed using numpy. For the two-layer Univariate LSTM model and two-layer multivariate LSTM model, the training window and the forward window are set to 10 days and 1 day respectively; for the CNN-LSTM hybrid model, the training window and forward window is set to 20 days and 1 day respectively. The dataset is split into train data and test data. The train data consists of 4000 data points, and the test data consists of 1219 data points.

Furthermore, normalization is performed through *sklearn.preprocessing.MinMaxScaler* on the X values of multivariate LSTM model. To prevent data snooping, only a single instance of MinMaxScaler was created but the train set and test set were transformed seperately.

### *iii.1.c* <u>*Model Universe*</u>

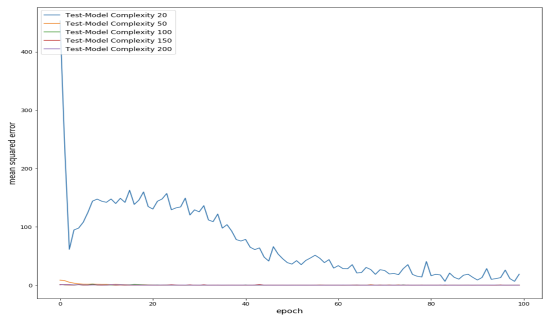
For regression task, we have implemented and trained the following models:

1. Univariate Two-Layer LSTM with adam optimizer and MSE as loss function
    a. First layer contains 100 input nodes, relu activation function, and input size of (batch_size, training_window, # of parameters) where batch_size is 64, training_window is 10, and # of parameters is 1
    b. Second layer contains 50 nodes and relu activation function
    c. The Output layer is contains of 1 node and linear activation function due to the nature of regression task
2. Multivariate Two-Layer LSTM with adam optimizer and MSE as loss function
    a. First layer contains 300 input nodes, relu activation function, and input size of (batch_size, training_window, # of parameters) where batch_size is 128, training_window is 10, and # of parameters is 3
    b. The second layer is batch normalization layer to avoid overfitting; the hyperparameters for the this layer are default keras setting
    c. Second layer contains 150 nodes and relu activation function
    d. The Output layer is contains of 1 node and linear activation function due to the nature of regression task
3. Convolutional Neural Network-LSTM Hybrid with adam optimizer and MSE as loss function
    a. The first layer is a Conv1D layer with 32 filters and kernel_size of 5 with a TimeDistributed wrapper
    b. The second layer is a MaxPooling1D layer with pool_size of 2 to reduce TimeDistributed wrapper
    c. The third layer is a flatten layer with a TimeDistributed wrapper
    d. The fourth layer is a LSTM layer with 100 nodes and relu activation function
    e. The output layer consists of 1 node with linear activation function

### *iii.1.d* <u>*Model Tuning*</u>

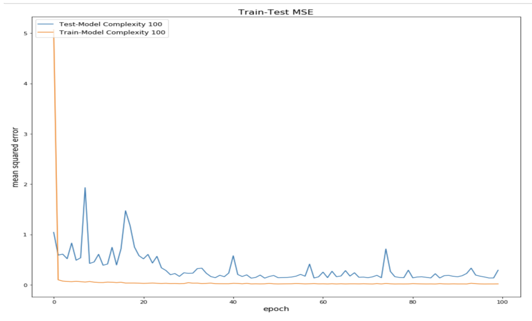
In order to tune model parameters, we conducted numerous empirical experiments. Below we describe a few examples:

To determine the model complexity for Univariate Two-Layer LSTM model, we ran an experiment to determine the optimal # of nodes for first two layers where i = # of nodes for the first layer and $\frac{i}{2}$= # of nodes for the second layers; and i was chosen from the following universe {10, 50, 100, 150, 200}. The results are shown below:



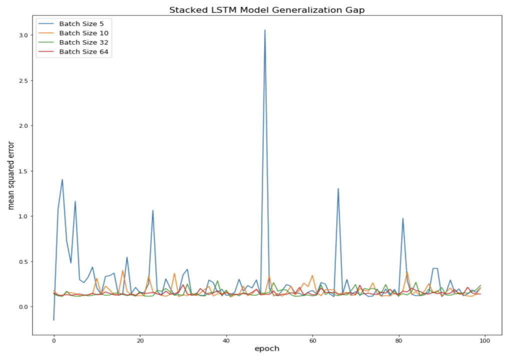
When i >= 100, test set MSE are barely distinguishable, so we have decided to use i = 100

Next, we want to decide # of epochs when training, so we plot the following train-test chart



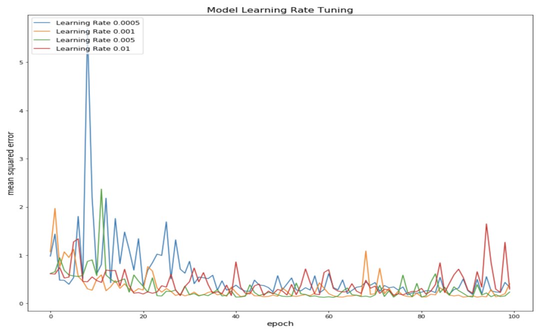
Once # of epochs reaches 50, the test MSE became relatively stable; however, there are still undesirable fluctuations, for example when # of epoch ≈60 and # of epoch ≈75, which we will investigate in the next experiment

After, we have discovered that training batch_size affects the extent of fluctuations on test set MSE as shown below:



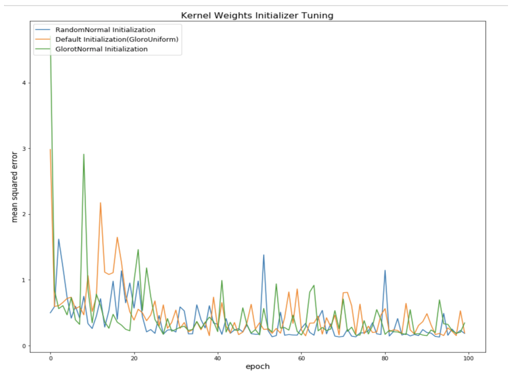
When batch_size=64, the fluctuation tends to be the mildest; furthermore, we have applied the following early stopping mechanism from Tensorflow to restore the best weights: *tf.keras.callbacks.EarlyStopping(monitor="val_loss", min_delta=0.001, patience=30, restore_best_weights=True)*

Furthermore, we tuned the model learning rate through changing the parameter in Adam optimizer instance: *tf.keras.optimizers.Adam( learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False, name="Adam")* where we set the learning_rate = i for i in {0.0005, 0.001, 0.005, 0.01}



According to the plot above, the green curve with learning_rate = 0.005 fluctuates the least once # of epoch >= 20

Additionally, we explored whether using another kernel weights initializer would increase the model performance. Therefore, we conducted an experiment with RandomUniform initializer while setting *stddev* = $\frac{C}{\sqrt{\# \text{ of inputs}}}$ and C = 2 because of the relu activation function (according to lecture), GlorotNormal initializer, and the model's default GlorotUniform initializer. The results are shown below:



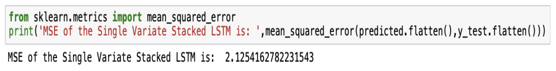
There is no clear winner, but RandomNormal Initializer seems to have less fluctuations.

Similarly, we repeated the above tuning steps for the other Multivariate LSTM and CNN-LSTM as well.

## *iii.1.e* *Model Selection*

Mean Squared Error (MSE) is the main metric we used to evaluate our models. Therefore, we would rely on it to select our best performing model. Below are the MSE of our three regression models on the test set after fine-tuning and training:

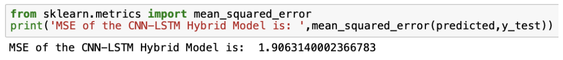
- Univariate LSTM:



```
from sklearn.metrics import mean_squared_error
print('MSE of the Single Variate Stacked LSTM is: ',mean_squared_error(predicted.flatten(),y_test.flatten()))
```

```
MSE of the Single Variate Stacked LSTM is:  2.1254162782231543
```

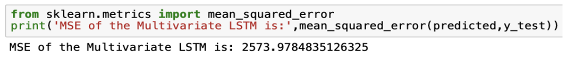
- CNN-LSTM Hybrid:



```
from sklearn.metrics import mean_squared_error
print('MSE of the CNN-LSTM Hybrid Model is: ',mean_squared_error(predicted,y_test))
```

```
MSE of the CNN-LSTM Hybrid Model is:  1.9063140002366783
```

- Multivariate LSTM:



```
from sklearn.metrics import mean_squared_error
print('MSE of the Multivariate LSTM is:',mean_squared_error(predicted,y_test))
```

```
MSE of the Multivariate LSTM is: 2573.9784835126325
```

Due to its extremely long running time and model complexity, the Multivariate LSTM model was not tuned and trained to its optimal when compared to the other two models. If tuned and trained properly on a specialized hardware,  we believe its performance would not be drastically different from that of the other two models.

As we can observe above, CNN-LSTM Hybrid model performs better than Univariate LSTM; however, as a sanity check, we just want to explore the model complexity before making the final decision. Below are the model summary:

- Univariate LSTM

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_62 (LSTM)               (None, 10, 100)           40800     
_________________________________________________________________
lstm_63 (LSTM)               (None, 50)                30200     
_________________________________________________________________
dense_30 (Dense)             (None, 1)                 51        
=================================================================
Total params: 71,051
Trainable params: 71,051
Non-trainable params: 0
_________________________________________________________________
```

- CNN+LSTM Hybrid

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
time_distributed_96 (TimeDis (None, None, 6, 32)       192       
_________________________________________________________________
time_distributed_97 (TimeDis (None, None, 3, 32)       0         
_________________________________________________________________
time_distributed_98 (TimeDis (None, None, 96)          0         
_________________________________________________________________
lstm_96 (LSTM)               (None, 100)               78800     
_________________________________________________________________
dense_63 (Dense)             (None, 1)                 101       
=================================================================
Total params: 79,093
Trainable params: 79,093
Non-trainable params: 0
_________________________________________________________________
```

Since there is no drastic difference between number of parameters between these two models, we have decided to prioritize predictive performance, thus choosing CNN+LSTM as our top performer.

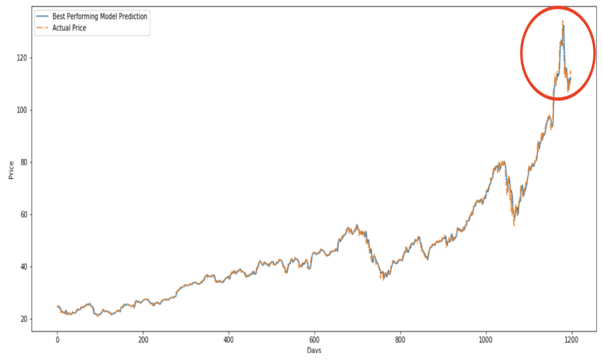
### *iii.1.e* *Model Evaluation*

In order to gain a more objective perspective on our best performing model, we have decided to compare it with two baseline models: 10-day Simple Moving Average and 10-day Exponential Moving Average (see Dataset and Features section for their definition and formulas). First, we compared the best performing model's MSE on test set with that of two baseline models:

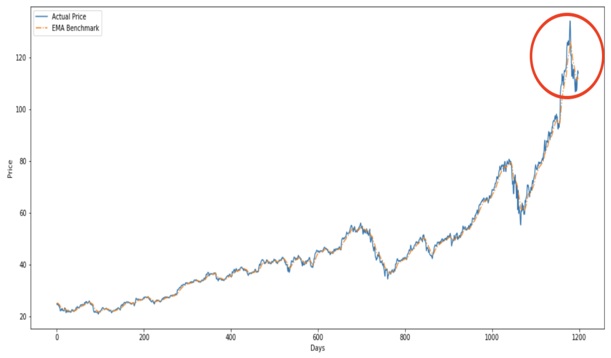
```
from sklearn.metrics import mean_squared_error
print('MSE of our baseline model using 10-day Exponential Moving Average is: '
      ,mean_squared_error(hist['Close'][4020:],hist['EMA_10'][4020:]))
print('MSE of our baseline model using 10-day Simple Moving Average is: '
      ,mean_squared_error(hist['Close'][4020:],hist['SMA_10'][4020:]))
```

```
MSE of our baseline model using 10-day Exponential Moving Average is:  2.6811371665962294
MSE of our baseline model using 10-day Simple Moving Average is:  4.867242029190897
```

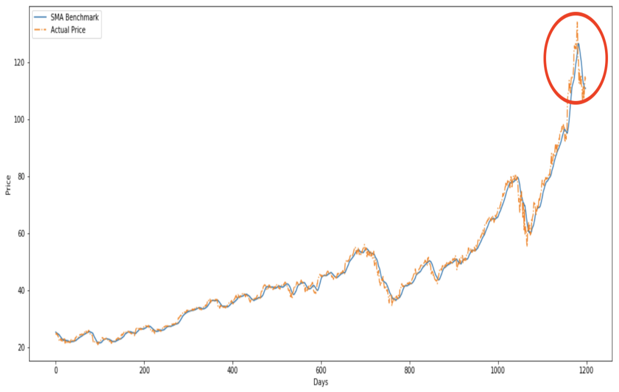
Both baseline models have higher MSE on test set than that of the best performing model, CNN-LSTM Hybrid, which has an MSE of approximately 1.90. Next, we want to compare the performance of these models visually:



Best Performing Model vs Actual Price



10-day SMA vs. Actual Price



10-day EMA vs. Actual Price

As we can observe from the plots above, all three models fit the actual price line fairly well; however, toward the end of the testing period (near Day 1200 as circled above), SMA model under-predicted the actual price and EMA model over-predicted the actual price while our best performing model, CNN-LSTM, predicted appropriately. Therefore, we can conclude that the CNN-LSTM Hybrid model performs fairly well.

## III.2 Methods – Classification

### *iii.2.a* *Feature Extraction*

Basic time series data requires pre-processing of the data before it is used. We apply the StandardScalar to features that are non-stationary such that $\mu = 0$and $\sigma = 1$, as per [5]. In particular, we apply this to the change to the data described by the difference between close prices (op) between day t and day t+1 divided by open price of t: $\frac{cp_t - cp_{t-1}}{cp_t}$

$$\text{Standardization: } z = \frac{x - \mu}{\sigma}$$

$$\text{with mean: } \mu = \frac{1}{N}\sum_{i=1}^{N}(x_i)$$

$$\text{and standard deviation: } \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

### iii.2.b *Models and Implementation*

This part of the project focuses on binary classification, we explore the data with several models:

1. Support Vector Machine (SVM): SVMs are a set of supervised learning methods used for classification, regression, and outlier detection. SVM finds the largest margin that separates the closest samples. We explored various kernels, including Linear, Polynomial (degree = 3), and Radial Basis Function kernel. We further adjust the model by modifying the value C in the optimization equation below:

$$\min w, \xi, b \ = \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}(\xi^{(i)})$$

$$such\ that\ y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi^{(i)} \qquad \forall i \in \{1, \ldots, n\}$$

2. Gradient Boosting: Gradient boosting utilizes an interactive function gradient algorithm that minimizes a loss function (in this case, exponential loss) by iteratively choosing a function that points toward the negative gradient.
3. Long-short Term Memory (LSTM): LSTM networks are a type of recurrent neural network (RNN) within deep learning. Models are trained on the Mean Squared Error (MSE) loss function. Below Table ii.ii.a shows the pipeline of the network:

| Layer | Units/Rate |
|---|---|
| LSTM | 20 |
| Dropout | 0.2 |
| Dense | 10, activation = relu |
| Dense | 2, activation = softmax |

Table iii.ii.a Pipeline for LSTM network

4. Convolution Neural Networks (CNN): CNN is a class of deep learning networks that rely heavily on the use of convolution layers. They are fully connected regularized Multilayer Perceptrons which reduce complexity as network depth increases. Models are trained on the Mean Squared Error (MSE) loss function. Below Table ii.ii.b shows the pipeline of the network

| Layer | Units/Rate |
|---|---|
| Convolution 1D | 20, activation = relu |
| Dropout | 0.2 |
| Batch Normalization | |
| Convolution 1D | 10, activation = relu |
| Batch Normalization | |
| Flatten | |
| Dense | 10, activation = relu |
| Dense | 2, activation = softmax |

Table iii.ii.b Pipeline for CNN network

### iii.2.c *Models Evaluation*

Models were evaluated on the basis of their portfolio performance. A simulation was created in which an account is provided with $100 at the beginning. Usage of MSE or accuracy is not necessarily indicative of a successful model; therefore we quantify the models based on their returns.

Returns: At day t-1, the closing prices are known as well. The classifier is run to predict whether the stock will increase or decrease at day t. If the model predicts 'increase,' then we proceed to buy stock, otherwise we sell short [see appendix for definition of buy and short]. Given the stock change to be 'r' of the close prices between t and t-1, if the model chooses to buy stock, the value of your account becomes $(t - 1\,close) * (1 + r)$. Conversely, if the model opts to short stock, the value of our account becomes $(t - 1\,close) * (1 - r)$. The results presented below show our profit in a zero-transaction cost environment.

| Model | Accuracy (training set) | Accuracy (test set) | Returns |
|---|---|---|---|
| Apple Stock | N/A | N/A | 215.36% |
| SVM (Linear) | 0.53 | 0.52 | 261.38% |
| SVM (Polynomial) | 0.53 | 0.52 | 257.36% |
| SVM (RBF) | 0.88 | 0.53 | 116.87% |
| Gradient Boost | 0.67 | 0.52 | 139.36% |
| Extreme Gradient Boost | 0.96 | 0.52 | 19.55% |
| LSTM (Classifier) | 0.53 | 0.53 | 671.83% |
| CNN | 0.59 | 0.54 | 61.62% |

Table iii.ii.c Statistical Performance and Cumulative Returns of all models

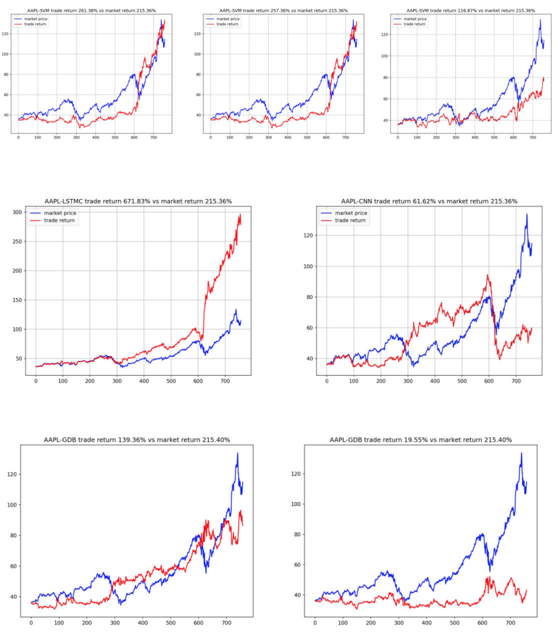


Figure iii.6 Plots showing trade return from the above models. Blue represents the amount of return if you buy stocks at the beginning of the test period and do nothing for the entire duration. Red represents model performance. Top row (SVM: linear, poly, rbf from left to right). Middle (LSTMC, CNN). Bottom (Gradient Boost, XGB)

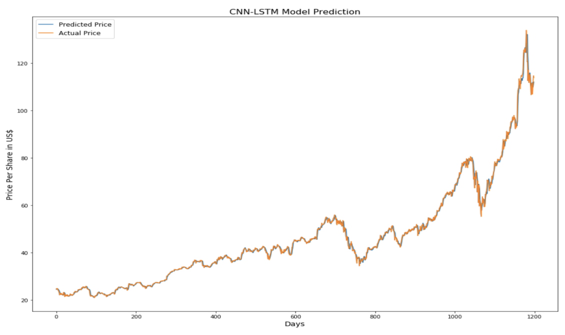
According to Table ii.ii.c, of the models we tested, the LSTM classifier had the best returns at 671.38% of the original portfolio value. Comparison of the accuracies illustrate that all the models show similar values, suggesting that forecasting the direction of stock prices is a difficult and stochastic process. It is important to note that accuracy; however, does not necessarily translate to higher economic profits. For instance, CNN had a higher accuracy for the test set although its returns were much lower than that of LSTMC which had a lower accuracy. Correct predictions could correspond to small changes (less profit) whereas some incorrect predictions could correspond to large changes (huge loss). At this point, we were not able to control that.

In order to ground the values we obtained from the models, we can compare performance of these models to that of SMP500 (yfinance '^GSPC')

## IV. Results and Discussion

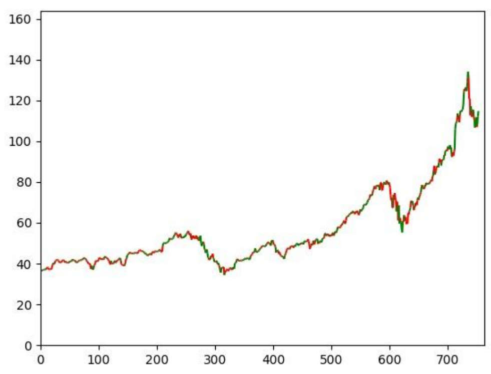
The CNN-LSTM hybrid model yields the lowest MSE of 1.90 compared to the Univariate LSTM model and the Multivariate LSTM model, which yield an MSE of 2.12 and 2573.97 respectively.

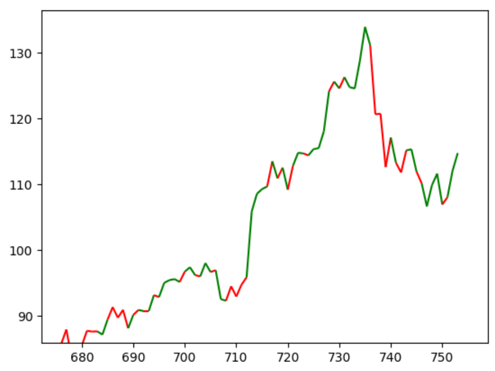
The model accuracy is where it was expected to be. The model correctly mirrors the long-term trends of stocks despite some level of misclassification of short-term fluctuation. This behavior is observed because of the nature of an LSTM model. By retaining longer-term information than that of a Recurrent Neural Network, for example, the model can make predictions that apply to longer periods of time in addition to the acute adjustments in price fluctuation. The result of this is a broader trend view of the stock price.



In applications of financial time series where there is a high level of deviation in the data, making accurate predictions cannot be guaranteed at all times. Especially in the case of stocks, there is a large level of human intervention and outside factors related to world events and public trust in a company that heavily influence the stock's value in addition to the features the model was trained on. This influenced our expectations of our models and our cautions regarding overfitting the data. With high levels of approximation, it would be expected that generalization performance would be adversely affected to a large extent. The CNN-LSTM model finds the balance between approximation and generalization performance from our empirical testing. In the future, we may seek to incorporate more features into the dataset relating to impactful stock news and earnings reports which could also have a large impact on price trends. Although our model accurately captures the trend of Apple stock, price trends in itself may be too singular to predict stock movements to a greater accuracy threshold.

As seen in the images below, the model cannot be perfect due to the certain level of unpredictability. That is, the variability in acute stock fluctuation. The goal of our model is to capitalize on the larger stock movements over a period of time. In our visualizations, this capitalization is represented by a green segment of the line. For every green section, it indicates our model would have either bought the stock and the stock price increased by closing time, or the model would have sold the stock and the stock price decreased by closing time. The red segments, on the other hand, indicate that our model would have either bought the stock and the stock price decreased, or would have sold the stock and the stock price increased. In essence, the green lines can be considered true positives and true negatives while the red lines can be considered false negatives and false positives. A good strength of our model is capturing the larger shifts in stock value over shorter periods of time. In the visualization on the right, which is the first visualization zoomed in at around days 680-750, there is a noticeable recognition of the bullish stock. At the same time, the LTSM model recognizes in some instances where the stock is at an all time high (ATH) and makes predictions on the stock decreasing in price from its patterns, which given the previous graph of this stock, would be a reasonable human assumption as well. Finally, we notice around the 740 day mark that the model correctly identifies both decreases and increases in prices for multiple fluctuations, indicating that the model is more than capable of capturing the stock's price changes in relatively shorter periods of time.





Our model complexity is appropriate for the Apple stock dataset due to appropriate train/test split and employment of early stopping and batch normalization (in Multivariate LSTM only) technique. In the future, we will explore the most suitable way to incorporate batch normalization into the other two models, Univariate LST and CNN+LSTM Hybrid. Additionally, while the model's predictions may appear overfit to the data since its graph conforms to the shape of the correct data, the predictions are generated on a day-to-day basis where the previous day's closing price is known as one of the features in the dataset. This allows the model to make predictions similar to the actual price fluctuation without making the identical closing values.

## V. Conclusions

Combining two deep learning methodologies - short-term memory networks (LSTM) added to convolutional neural networks (CNN) - we have the CNN-LSTM hybrid model. The model gave the best prediction of the stock market price movement with an MSE score of 2.07. The results may not be generalized to other stocks due to the fact that only Apple stock has been tested. However, the predictive capacity of this training method provides evidence for the possibility of general application. The same methods could be applied to many (if not all) stocks in order to provide some predictive measure and aid companies and investors in generating better investment returns. Of course, many unpredictable events can cause stock fluctuations that trump these models and methods, but reliable companies can expect to have generally predictable share-price growth.

## VI. Source Code

All source code can be found here: Github

## VI. Appendix

A *bullish* stock refers to a stock that is expected to rise in value in a given period of time.

A *bearish* stock refers to a stock that is expected to lower in value in a given period of time.

*Buying* a stock refers to the purchasing of a stock with the prediction that the stock price will rise in a given time period. In purchasing the stock, the investor aims to make money in the appreciation of it's value.

*Shorting* a stock refers to the selling of stock with the prediction that the stock price will go down. The motivation behind this is to purchase the stock at a later point in time following a drop in price.

## VII. References

[1] Qiu, Jiayu & Wang, Bin & Zhou, Changjun. (2020). Forecasting stock prices with long-short term memory neural network based on attention mechanism. PLOS ONE. 15. e0227222. 10.1371/journal.pone.0227222.

[2] Learning to Forget: Continual Prediction with LSTM. Felix A. Gers , Jürgen Schmidhuber , and Fred Cummins. Neural Computation 2000 12:10, 2451-2471

[3] Caterini, Anthony L., and Chang, Dong Eui, author. Deep Neural Networks in a Mathematical Framework. Springer, 2018.

[4] Jabin, Suraiya. "Stock Market Prediction Using Feed-Forward Artificial Neural Network." International Journal of Computer Applications, vol. 99, no. 9, 2014, pp. 4–8., doi:10.5120/17399-7959. Focuses on back-propagation for the Neural Net.

[5] Petnehazi, Gabor. Recurrent Neural Networks for Time Series Forecasting, Doctoral School of Mathematical and Computational Sciences University of Debrecen, 1 Jan. 2019, arxiv.org/pdf/1901.00069.pdf.