# CS449 Deep Learning: Final Project Report

## Goals and Discussion

For our project, we set out to build multiple models using various deep learning architectures to predict stock prices based on historical patterns of the stock. This is inherently a difficult task, as stocks tend to follow random motion purely as a function of time. However, after reading papers on the subject and understanding the strengths of certain deep learning architectures, we believed that we could build a model to learn these extremely complex patterns within the movement of a stock's price.

We fully built five models and completed many intermediate goals along the way. Our goals were split into three sections: essential goals, desired goals, and stretch goals. In all, we completed both of our essential goals, all three of our desired goals, and made really solid progress on two of our three stretch goals.

### Essential Goals

The first essential goal that we completed was building an LSTM to correctly predict if a stock's price was going to increase or decrease over a certain predetermined period of time. This was a straightforward task, although it did require us to problem solve in order to get a meaningful result. We originally approached this as a binary classification problem, either up or down. This proved to be too general, as the overall upwards trend of the market as a whole caused our model to simply always choose to predict an increase, as this allowed it to most easily minimize the loss. In order to address this properly, we decided to alter our approach to address the problem as a regression task. In order for this to produce meaningful results, we decided to alter the output to predict percentage changes, instead of raw values. This allowed the model to be completely price (and therefore time) invariant. Additionally, to get meaningful predictions from the LSTM, we normalized the input data so that it could train easily, as well as being provided more distinguishable data, since most data points in an input set would be within less than a dollar of each other.

The second essential goal that we completed was being able to use this LSTM over a wide variety of securities. This required writing a data reading file that could pull data from many different sources and return it in the same format that the model needed for training. Our file reads data from both a Kaggle dataset and Yahoo Finance. While the patterns we are trying to predict are already extremely abstract, observation of the movement of many securities makes it clear that different classes of securities tend to move differently. ETFs have a tendency to move in a controlled manner, whereas cryptocurrency appears to have an extremely sporadic pattern. In order to quantify our goal of making "meaningful" predictions, we ran it on several different windows across individual stocks, cryptocurrency, and ETFs and monitored both the model's performance, as well as the actual predictions to examine whether the values lied in a reasonable range for a prediction. Figure 1 illustrates the model's performance when trained for three years, and allowed to trade for three consecutive years on different securities, where red represents a period of short selling, and green represents a period of holding. While the performance varies, our model is able to predict several dips in the market on each, leading to yields that surpass that which would be obtained from buying the stock and holding it for the trading time.

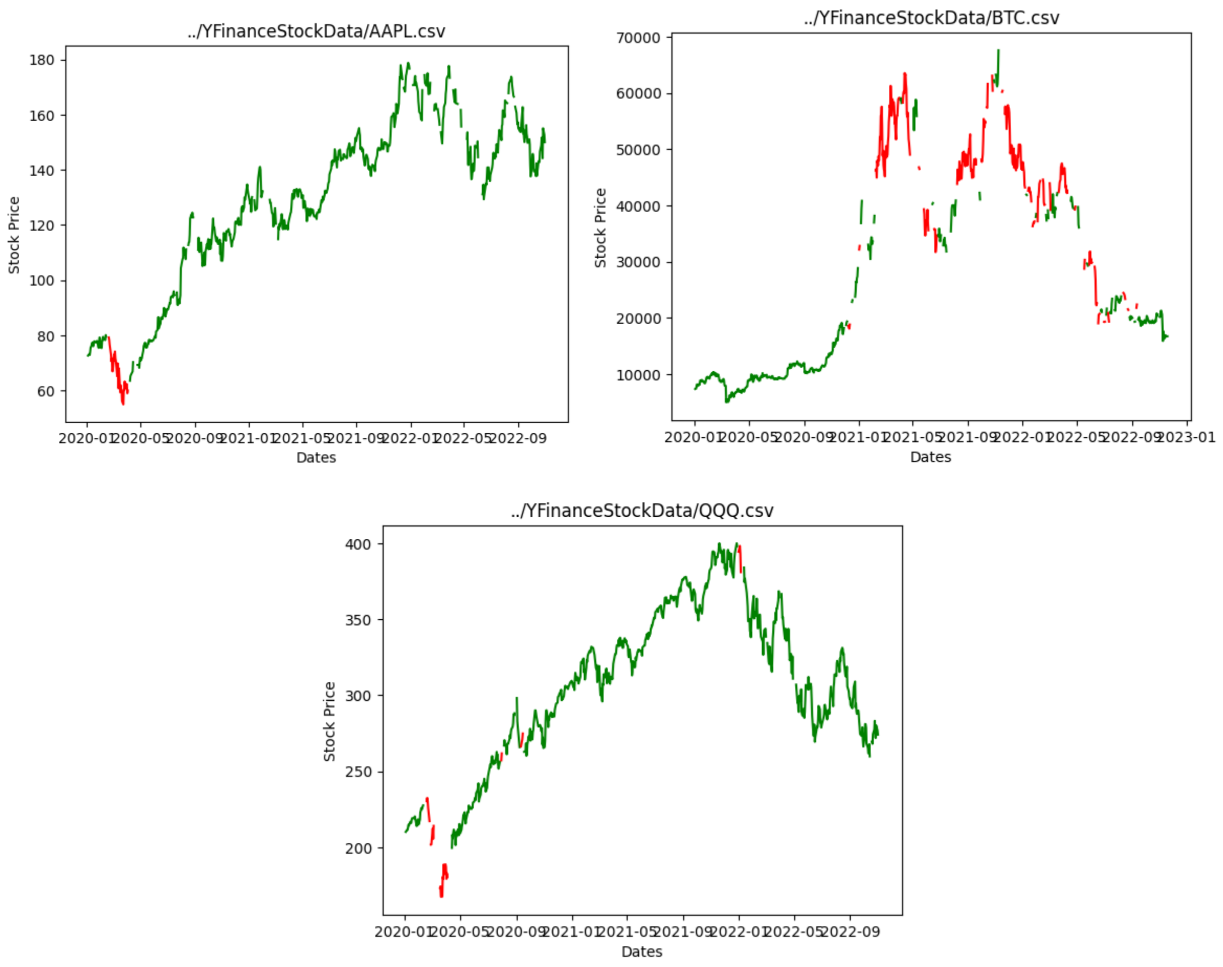

Figure 1: Positions across various securities

**Desired Goals**

The first desired goal that we accomplished was making our model effective over any arbitrary time interval. All of the models we have built in this project are able to make decisions for any number of days out. This is handled in the data reader file, which allows us to determine the sequence length used in training/evaluation, as well as the amount of days in the future our model will be predicting. These served as valuable hyperparameters that determined performance heavily, as a wider window improved accuracy to a point, until too much information was present for patterns to be detected.

The second goal in our desired category was the implementation of two additional models for the sake of both comparison as well as implementation into our final ensemble model.

The first of these models was the multilayer CNN. This network used one dimensional convolutional neural networks along with max pooling layers to make predictions. The motivation for this model was to detect specific patterns in stock data, rather than examine trends across time like an LSTM. This model was configurable in its kernel size and depth, but as was true with the LSTM, a shallow depth and limited kernel size proved to be most effective, likely due to large sizes causing nuances in information to be overshadowed by the sheer size of the incoming sequence.

The second of these models, labeled our "hybrid" model, was a model of an LSTM and CNN sequentially, motivated by the findings and architecture of Kanwal et. al, in that this configuration allows the model to detect both trends across time and inside individual sequences. These models also possess configurability in their hyperparameters independently from the other two homogeneous models, although we found similar parameters to fit the models best.

To finalize our desired goals, we implemented a fourth model, an ensemble of all three models previously discussed. The ensemble model (Figure 2)  combined all three individual models, along with technical data often used by professional traders, to make a single prediction. The individual models execute in parallel, and the results from each model are combined by the MLP for a prediction that would ideally be able to take advantage of each architecture's advantages.

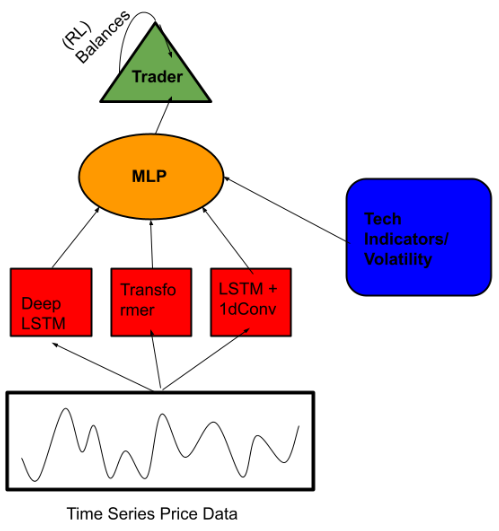

Figure 2: Illustration of Ensemble Architecture Concept

## Model Evaluation

In order to confirm our success in completing these goals, we used several graphs and statistics to ensure that our models were not only functional, but that the hyperparameters and training strategies were selected efficiently so that we could achieve optimal results (Figure 3). The first step for ensuring model success was individual hyperparameter tuning. One of the difficulties in choosing these parameters was that we needed them to generalize well to a variety of stocks, since the model is only designed to learn a single stock.

One of the biggest improvements came when we selected our optimizer to use Adam instead of Gradient Descent, as this allows the model to train well more generally. This, combined with the normalized input data, also allowed us to achieve solid results without overtraining in a triple digit number of training steps. Smaller training steps meant quicker evaluation and tuning, as well as ease of use in the case this model was deployed on the market in real time. We also tuned our model by trying out different architectures such as the number of layers. The final parameters were chosen after a Grid Search like technique.

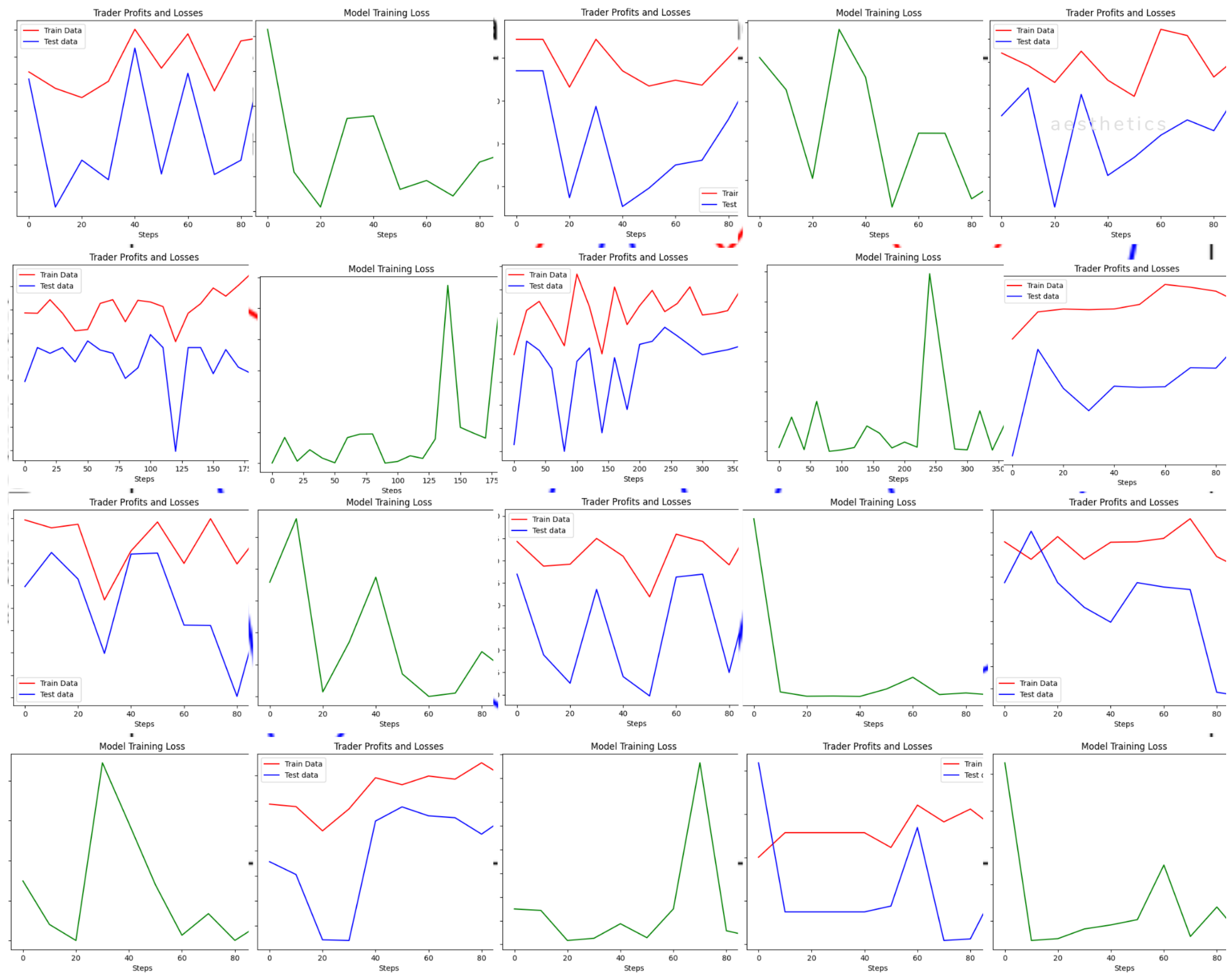

Figure 3: Hyperparameter Tuning and Analysis Results

In order to evaluate the models in a way that was meaningful for financial analysis, we then constructed the “Trader” class. This class takes a prediction (In percentage form) of a model, and takes an action. These actions include buying a stock, holding a position, or selling a stock. It also supports a configurable option to allow short selling, which we enable in our analysis. The trader keeps track of its balances as well as transactions, so that we can further analyze the results of the predictions. When the testing is done, the trader closes its position. For the purpose of normalizing results among stocks, results are collected as final return as a percentage compared to the initial trade price (considered to be the initial balance) and the trader is always allowed to hold exactly one position, with exactly one share.

Once this class was implemented, we were able to analyze how our model performed on the actual market. The first analysis took a grab bag of popular stocks across various sectors, and trained one of each of our four models on this stock. Additionally a fifth stat for the “Buy and Hold” trader was added, to analyze whether our models outperformed the common strategy of simply holding a position for a long period of time regardless of market (Figures 4 & 5) We trained this model on 3 years of data, from 2014 to 2016, and tested it on 2017-2019. This window was selected to avoid the chaos of the 2020 market which had potential to confuse the model.

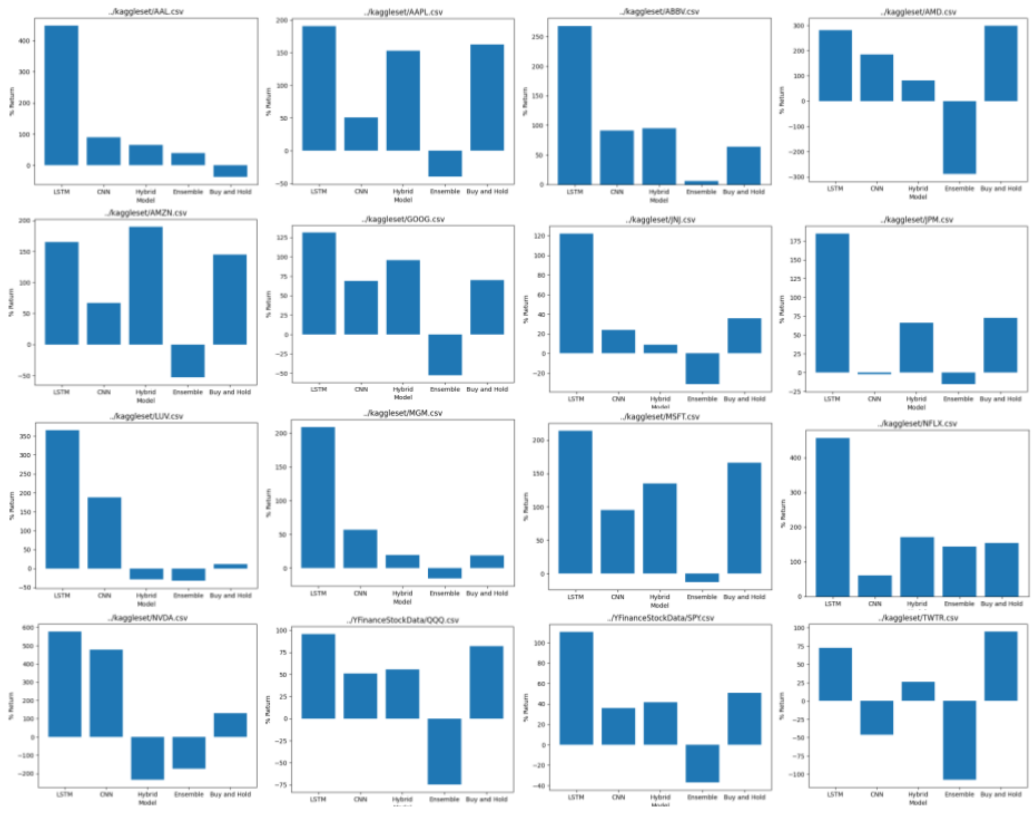

Figure 4: Model performance by stock

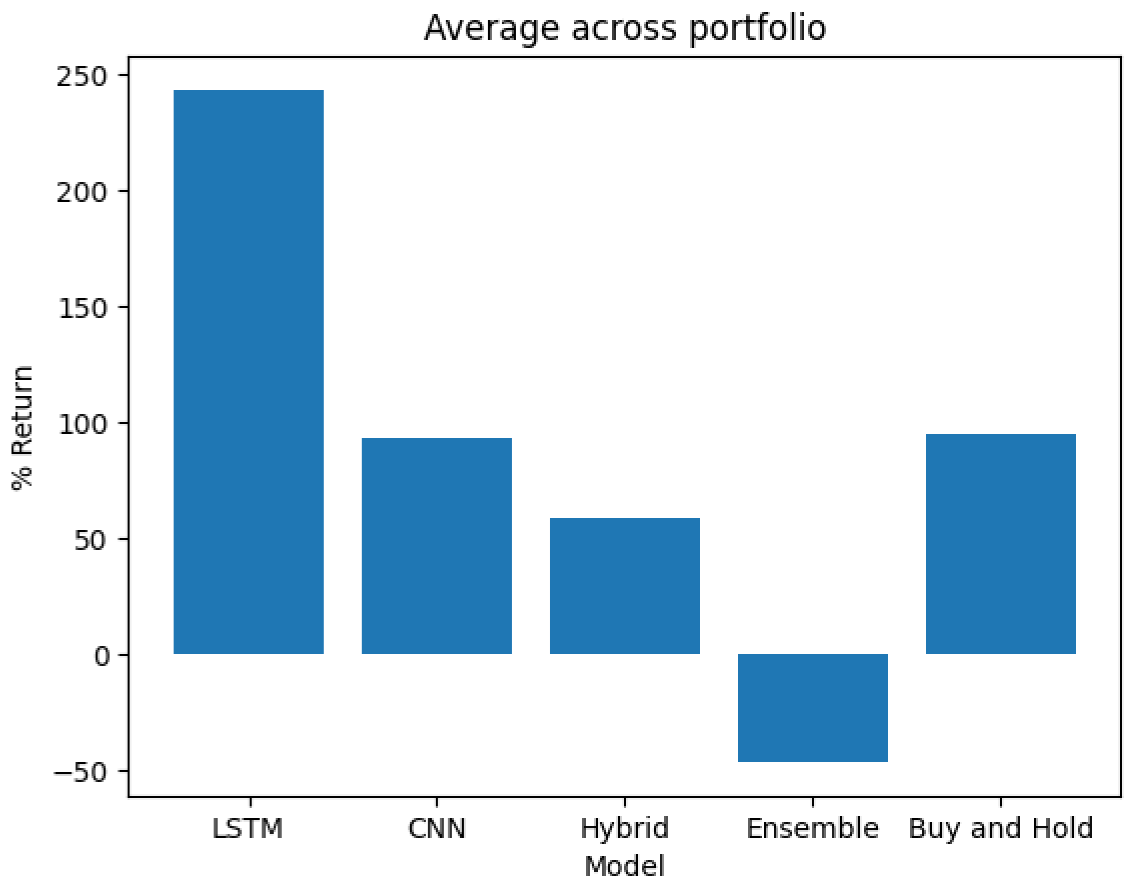

Figure 5: Combined model performance

From this we were able to include that in general, the LSTM performed best, with the CNN and hybrids beating buy and hold on certain securities, and the ensemble being extremely unreliable, likely due to the complexity of the model compared to the training data set size. This was, however, only on one six year window, and for a broader consensus we expanded our testing to six of these windows. We ran the same tests as above, beginning with a training window of 2012-2014, and a test set of 2015-2017. We logged results, wiped the models, and moved the window into the future by one year, and repeated six times for six different total experiments across different time periods, including the heavily volatile and unpredictable results of the pandemic market, shown in Figure 6.

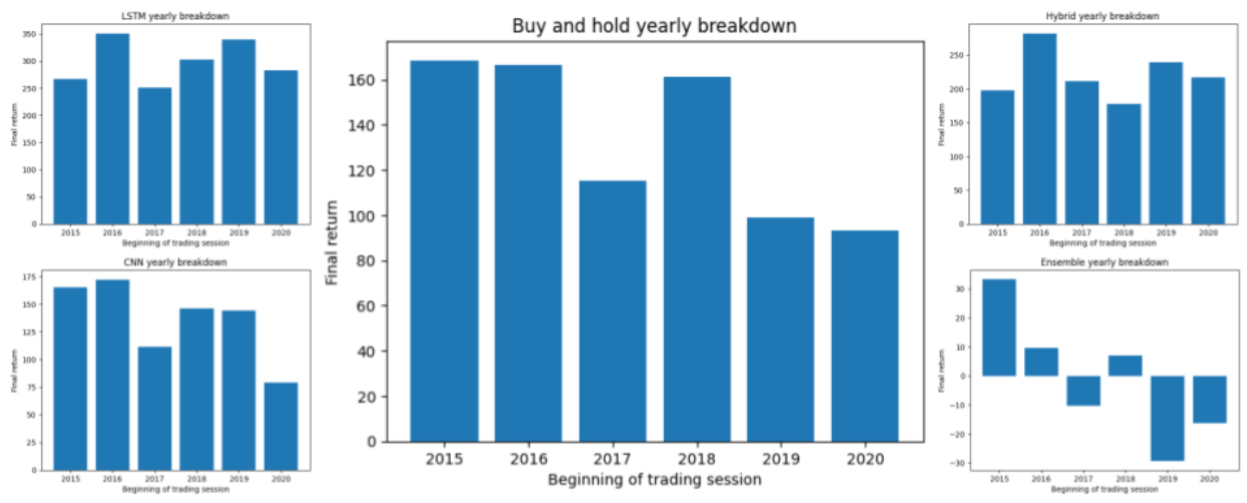

Figure 6: 12 Year evaluation results

Our conclusions were thus reinforced by this test, and our results were shown to be relatively consistent across years, and more dependent on model architecture and specific security. Because of these results, we opted to include technical indicator support for our LSTM, with the intention of supplying additional potentially useful information for the model to further improve its accuracy.

The technical indicator and LSTM model incorporates both of the inputs and combines them in an MLP. From the graphs, the LSTM trains decently, but it performs much worse than the other models on the actual trading done. While the technical indicators provide extra information, the are merely based off of the close price, and it seems as if these extra indicators which are based off of them are not reliable metrics.

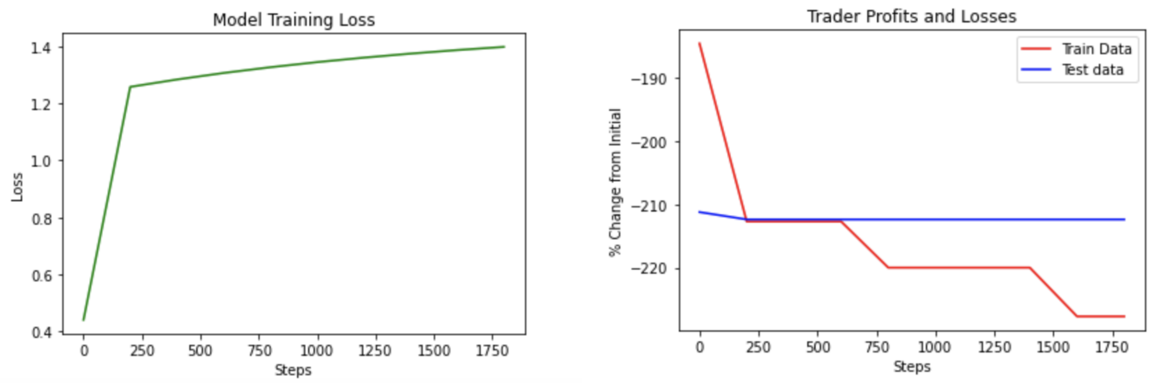

Figure 7: LSTM-MLP Ensemble Model Performance

**Stretch Goals**

The first stretch goal we set out to complete was to build a reinforcement learning model that could dynamically react to its own predictions and learn from itself. The goal was for the model to make a prediction, and then the difference between the prediction and the actual stock price at that time interval to be the model's reward/punishment. We did not finish this goal, but we did make significant progress. We built a new class that we call the Trader class, and this would be our agent that is making the buy or sell decisions. Given a prediction and the current price of the stock, the trader instance can decide to buy, sell, short, or close a short all with its actOnPrediction method. The most difficult part of this is formulating the decision making in a way that is useful for a reinforcement learning model. We already have the clear objective of making money and therefore the traders quite simply allocate funds in accordance with their current prediction. The reinforcement learning aspect would come in the form of learning how to allocate funds to mitigate risk and maximize profit given a handful of securities to choose from. This was further made difficult by the indeterminate nature of the reward, and the limited data available for an individual stock. This prevents our training from being able to explore large pools of choices and thus learn an optimal decision making policy to maximize profits. We were unable to get a functioning reinforcement model working, and this would likely be one of the more productive avenues for future work on the project.

The second stretch goal that we set out to complete was to incorporate more information into our models such as economic and political announcements along with earnings releases and corporate press releases. This goal requires a sentiment analysis model to provide this data, and while we weren’t able to incorporate this, our models that support technical indicators are capable of supporting the results of such models in their training as well. If we implemented such a sentiment analysis model, including its outputs into the ensemble or LSTM would be trivial.

The third stretch goal that we hoped to accomplish was hooking our best models up to a trading simulation platform to truly analyze our results. The goal would be for the model to make live predictions on live data, then for the trader class to take the best action based on the prediction. Additionally, we would give the trader some amount of money to use as initial investment, and it would have to learn the best way to allocate. However, this would be difficult for our architecture as we do not yet have any concept of risk. The way in which our current trader performs evaluation would allow for it to make decisions, and given an interface between the model and the platform, it would be able to make trades based on the model. Similarly to our other stretch goals, a single missing piece exists preventing the implementation, but upon the development of this interface between the platform, expanding it to make real predictions would be straightforward.

**Intermediate accomplishments**

One of our biggest accomplishments that was not necessarily a fully outlined goal was our data processing module. This is self contained in the dataReader.py file. This module allowed us to add a lot of robustness to our testing and our analysis by permitting a lot of configurations. It allowed for many different types of data (binary, percent, and real values) along with different size input windows and prediction offsets (week, month, etc.). This was key in allowing our models to be trained and evaluated over different ranges of dates and for different types of predictions.

Another intermediate accomplishment we achieved that aided in our progress is our robust evaluation methods. We put together a notebook that allows us to not only see the performance of our models under various positions, but also the performance on a broad scale with a script that is able to be run overnight to obtain over a decade worth of data. This allows us to understand how our model is operating by providing us with graphs showing the model's positions at each time step, comparing models over long time frames, and analyzing our model's output against common trading strategies.

## Code and Documentation

src/dataReader.py

- This file pulls data from a Kaggle dataset that we use for training and testing. It is very configurable, allowing for data to be normalized or unnormalized, binary (increase or decrease), percent (change over previous day), or real values (stock price). This file also allows for pulling data with any arbitrary combination of training window and prediction offset.

src/trader.py

- This file keeps track of all of the predictions as they are relevant to a potential trader. It tracks balances, transactions, and statistics about performance
- This allows us to expand our analysis of performance beyond the abstract loss values to percentage values of actual returns, had the trades taken place.

src/model.py

- This file is the backbone of our architectures, and describes the forward method, initialization, and structure of the three initial models: LSTM, CNN, and hybrid.

src/trainer.py

- This file includes the trainer classes that train the models included in model.py

src/evaluation.ipynb

- This notebook allows running all tests discussed in the model evaluation section of the report, including testing individual models, individual years, and the twelve year window.

src/ensemble.py

- This file defines the ensemble model for use throughout the other files. It contains an initialization function, a weight initialization, and a forward pass function.

src/ensembleTrainer.py

- This file provides the trainer class that trains our ensemble model. It is in its own file since the ensemble is so complicated, adjusting hyperparameters and evaluating performance was easier with it separate from the rest of the models.

src/ensembleNotebook.ipynb

- The ensemble notebook allows us to test the ensemble alone. Being a more complex model it takes much longer to train, therefore isolating it in testing initially was necessary to save both time and energy.

src/lstmMLP.py

- The final model we constructed was the LSTM MLP model that included technical analysis indicators. Since this model is a continuation of the model stored in model.py, we opted to create a separate file for this model.

src/lstmMLPTrainer.py

- This file trains the model given in the lstmMLP.py, which is isolated to allow individualized training adjustments

## **<u>Reflections</u>**

### What was interesting?

We expected our hybrid model and the ensemble method to outperform the Lstm model, however not only is our Lstm model better but its performance is way better than we expected. One reason for such a consistent trend could be that our hybrid model and ensemble model are too complex for the given data size, we used 3 years of training data- which took us 7 hours to evaluate on each evaluation. Using a greater window was infeasible for the project, but we hope that with a larger training set, our complex models will be more effective.

### What was difficult?

One of the most difficult parts of this project was constructing models that generalize well. Our focus in model tuning was more emphasis on consistency over individual performance, as we would prefer a more modest and consistent positive return over an unpredictable one. This was further made difficult by the limited data size, along with the inefficiency of batch training.

We explored different time windows and found that too large of a data size caused that model to fail in recognizing patterns, but smaller data sizes underperformed on our larger models. This meant that we had to adjust every aspect of our models to allow them to perform on the decided three year window for optimal performance.

Additionally, after comparing batched input against unbatched input, the unbatched models performed far better, and therefore we opted to work without batches. This slowed training, and therefore evaluation.

### What's left to do?

There are several ways in which this model could be improved going forward. Firstly, the implementation of both the RL based trader class along with a trading platform interface would allow for full analysis of the model by allowing it to function on a live market. For the sake of evaluation and improvement, this could be done on a paper trading platform to prevent financial risk, but allow for more useful decisions to be made regarding future improvements and hyperparameter adjustments.

Additionally, the larger models (Hybrid and Ensemble) could be adjusted further and given more train data and time to find a more profitable window. These models likely require more in depth analysis to their training to improve efficiency, simply because they are so large and complex.

Either of these approaches would likely result in a more robust model able to produce even more accurate predictions, which ideally would improve profit margins.

## References

Kanwal, A., Lau, M. F., Ng, S. P. H., Sim, K. Y., &amp; Chandrasekaran, S. (2022). BICUDNNLSTM-1dCNN — a hybrid deep learning-based predictive model for stock price prediction. Expert Systems with Applications, 202, 117123. https://doi.org/10.1016/j.eswa.2022.117123