Machine Learning (ECE-GY 6143) Course Project Report

Stock Price Prediction using RNNs

Aadarsh Lakshmi Narasiman - al9581

Karuna V - fk2496

Nitish KS - nk4277

GitHub - https://github.com/AadarshLN/Stock-Price-Prediction-Using-RNNs

10th December, 2025

Introduction

Financial markets are inherently dynamic, non-linear, and influenced by temporal dependencies. Stock prices do not evolve independently at each time step; instead, their current value is strongly influenced by historical trends, patterns, and momentum. Traditional statistical models often struggle to capture such temporal dependencies effectively, especially when the relationships between past and future values are non-linear.

RNNs are commonly used in time-series forecasting and are designed to model sequential data making them well suited for stock price prediction.

Background

A time series is a sequence of data points collected or recorded at successive time intervals. In this project, the closing price is used because it represents the final consensus price of a stock for a trading day and is widely used in financial analysis.

Recurrent Neural Networks are a class of neural networks designed to process sequential data. Unlike feed-forward neural networks, RNNs have internal memory that allows them to retain information from previous time steps.

For a given time step t RNN computes as follows:

$$h_t = f(W_h h_{t-1} + W_x x_t + b)$$

where:

x_t is the input at time step t

h_t is the hidden state

$h_{t-1}$is the previous hidden state

W_h, W_x are weight matrices

$f(.)$ is a non linear activation function

This recurrent connection enables the network to model temporal dependencies, making RNNs suitable for time-series forecasting tasks such as stock price prediction.

In this project, SimpleRNN layers are used. It does the following:

- Processes input sequences step-by-step
- Maintains a hidden state that captures information from previous time steps
- Is computationally simpler than advanced variants such as LSTM or GRU

To convert time-series data into a supervised learning problem, a sliding window technique is employed:

- A fixed number of past observations (look back) are used as input.
- The model predicts the next time step value.

We have used look_back = 60 which would mean the model uses the previous 60 days of closing prices to predict the price on day 61.

Neural networks perform better when input values are scaled to a uniform range. In this project we have done the following:

- Min-Max Scaling is used to normalize prices between 0 and 1.
- Scaling is inverted after prediction to evaluate performance in actual price units.

We have evaluated the model using RMSE and MAE

Approach

We had run the RNN across all the available company data by initially doing it company wise just to play around with how the model was performing company wise. This was done by using the ticker set for each company separately with a train-test split of 80-20 and training of 100 epochs with mini batch gradient descent with batch_size=64 and then training the model separately for each company. (Let us call this **Approach-1**) However, that would not be practical in real life so we went about combining the data across all companies (i.e. across all ticker values and then using the yfinance(YahooFinance) library to call the data) and combining them and then doing the train test split (80-20). This would ensure that it would predict across all companies in a combined fashion and then in the case of continual learning wherein the model updates itself with new data, the model's relevant checkpoints can be used and then it can be finetuned. Additionally it was observed that when combining it and learning it took 50 epochs to get good results wherein when training separately we needed 100 epochs for learning and Mini-Batch gradient descent was used with batch_size=64 . The data from yfinance is sourced from Jan 1st 2020 to December 1st 2025.

Model information:

- The RNN architecture has 2 stacked SimpleRNN layers with 50 neurons each
- There is a dense output layer with one neuron for price prediction
- The loss function used is Mean Square Error
- The optimizer used is Adam

Following this, plots were generated wherein the actual stock price vs predicted stock price was drawn across time.

Note on some Conventions used:

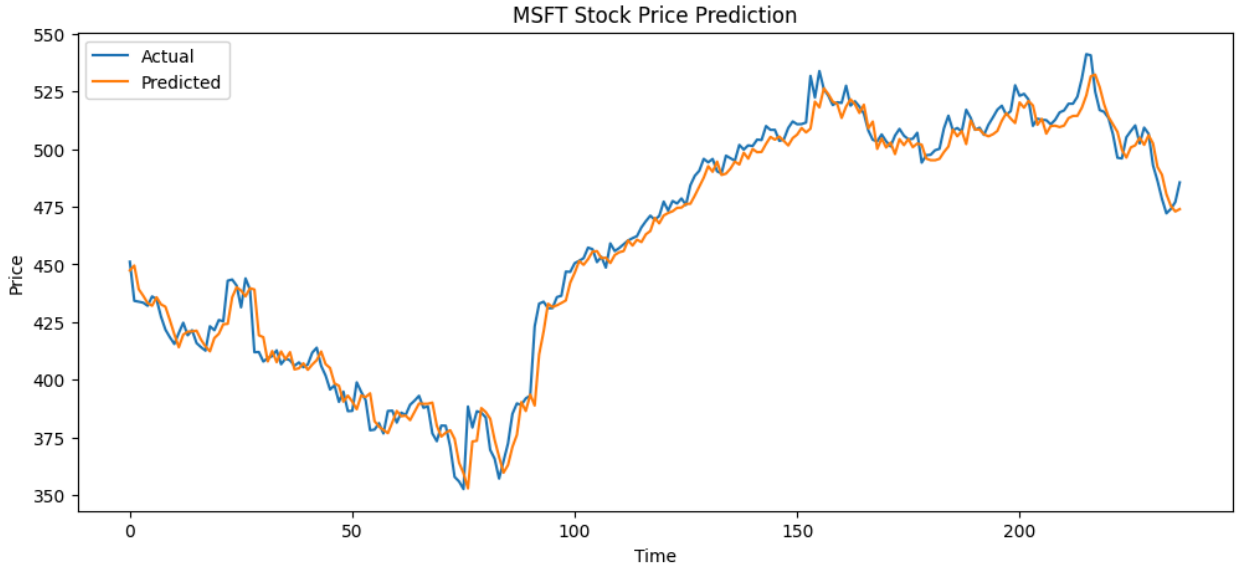
The following are conventions associated with the yfinance library to fetch the company data and in it is used across this report too

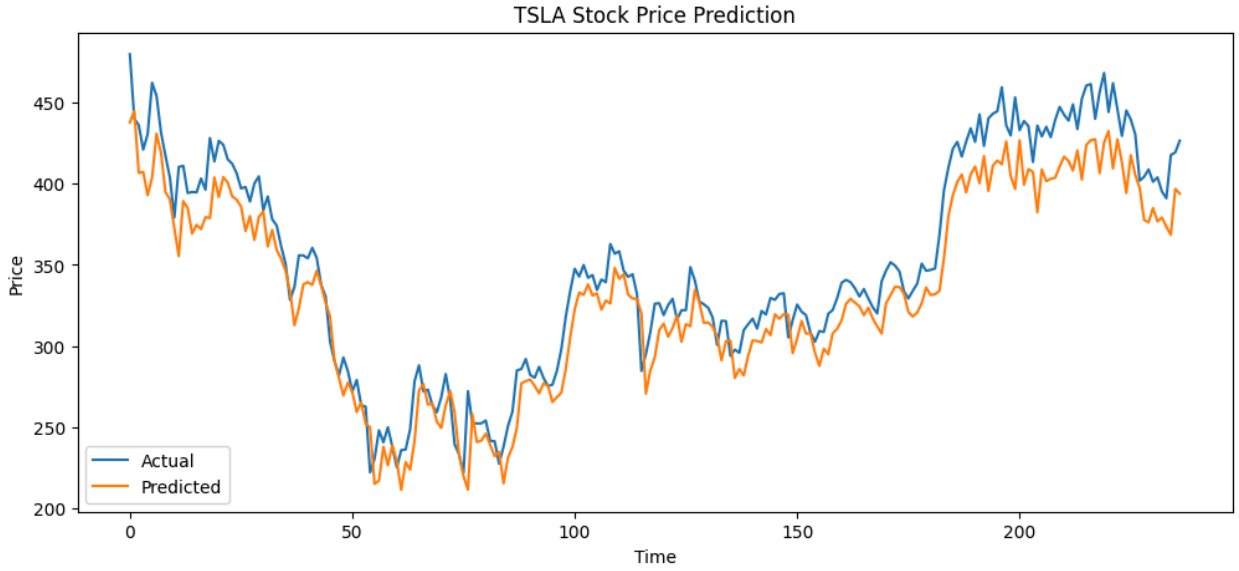
- AAPL - Apple
- MSFT - Microsoft
- GOOGL - Google
- TSLA - Tesla
- AMZN - Amazon

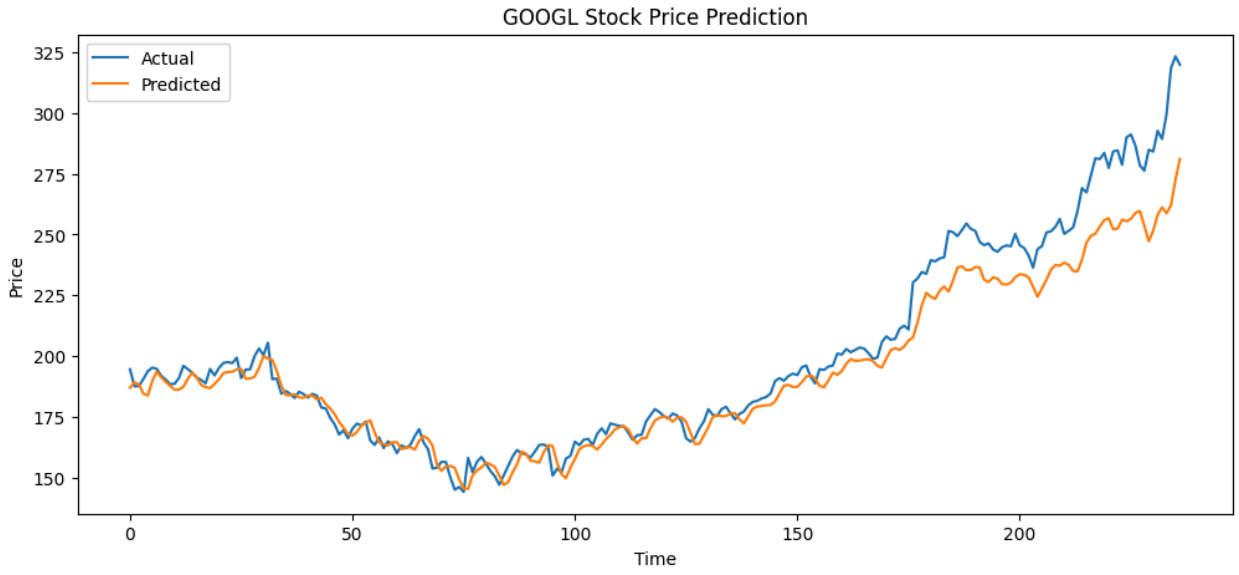
Results - Approach 1

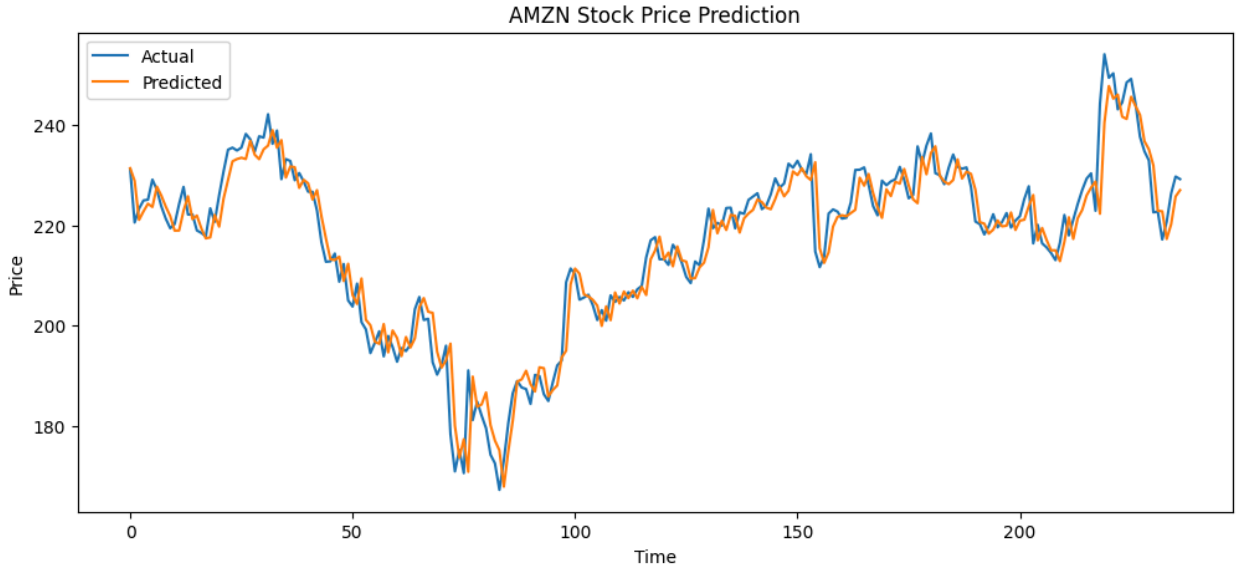
The table entails the Test RMS and Mean Absolute errors across the test data followed by graphs on expected vs actual price prediction across time as a Price vs Time graph

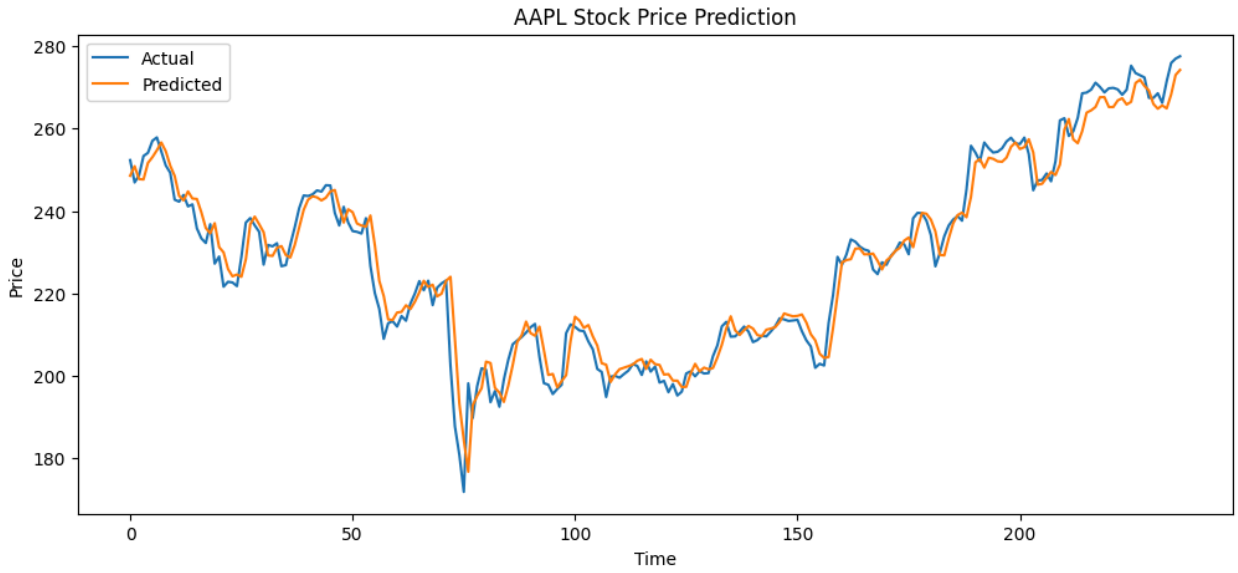
Stock	**Test RMSE**	**Test MAE**
MSFT	13.525	11.345
GOOGL	10.051	7.385
TSLA	14.942	11.404
AMZN	5.730	4.451
AAPL	4.859	3.504


MSFT Stock Price Prediction
Actual
Predicted
Price
Time
550
525
500
475
450
425
400
375
350
0
50
100
150
200


TSLA Stock Price Prediction
Actual
Predicted
Price
Time
450
400
350
300
250
200
0
50
100
150
200


GOOGL Stock Price Prediction
Actual
Predicted
Price
Time
325
300
275
250
225
200
175
150
0
50
100
150
200

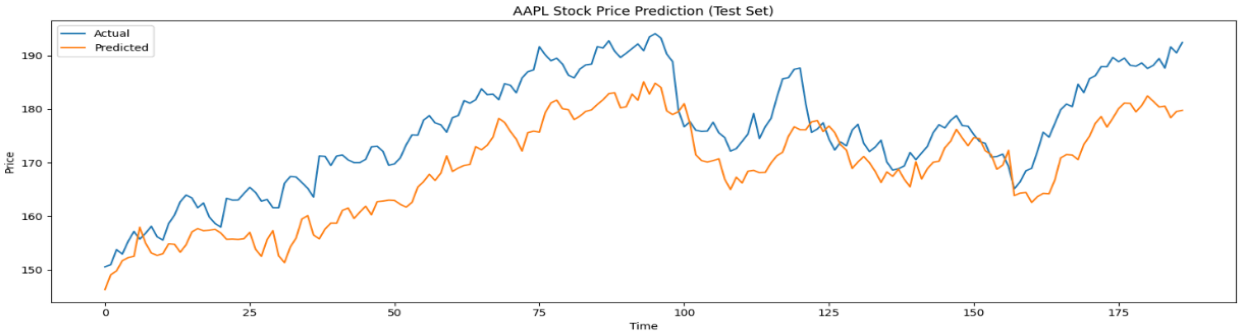

AMZN Stock Price Prediction
Actual
Predicted
Price
Time
240
220
200
180
0
50
100
150
200

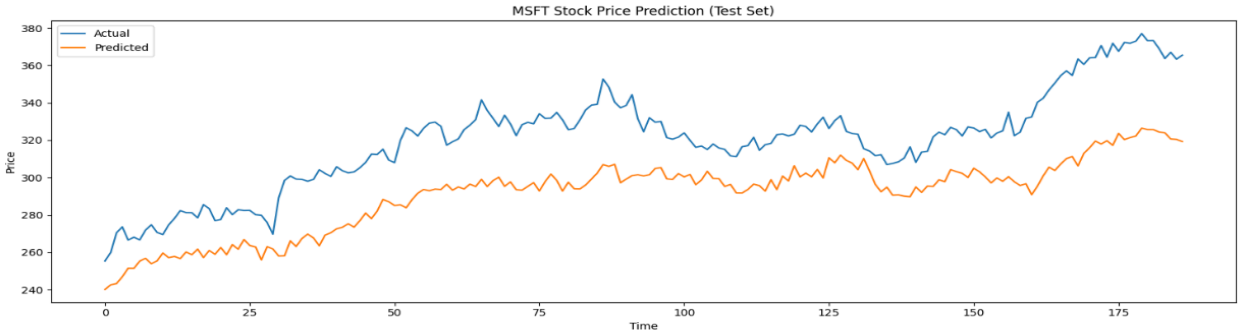

AAPL Stock Price Prediction
Actual
Predicted
Price
Time
280
260
240
220
200
180
0
50
100
150
200

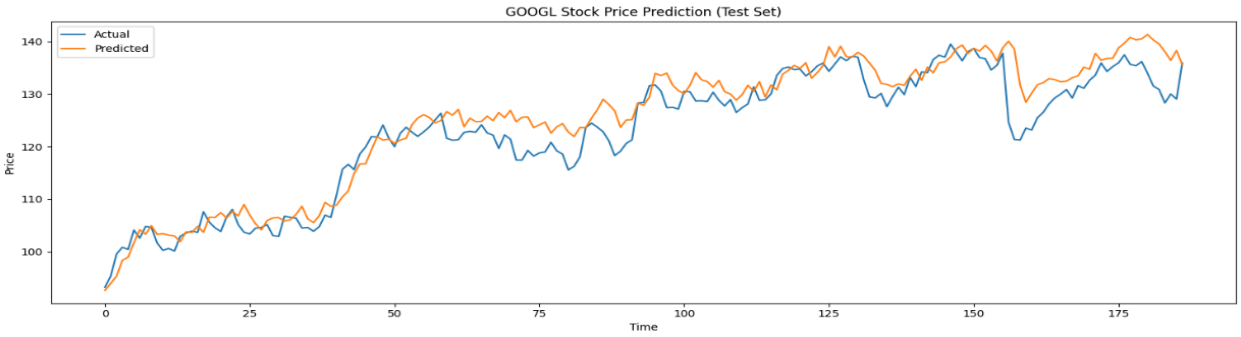
Results - Approach 2

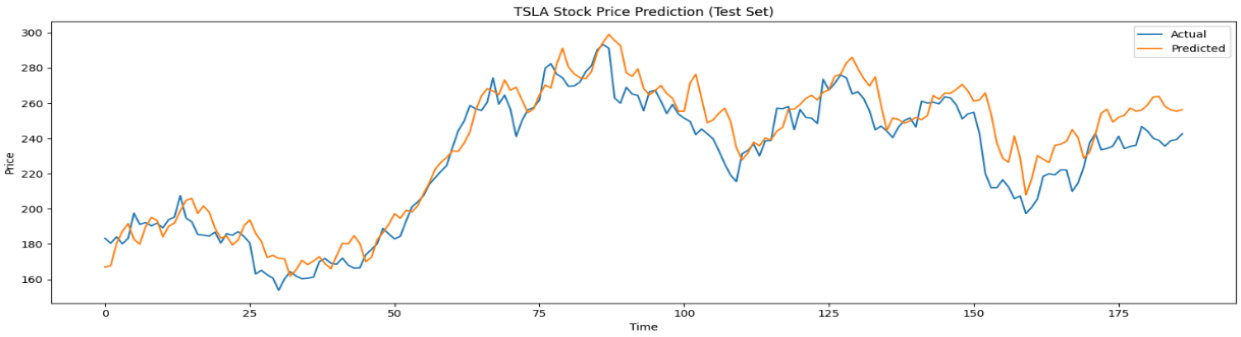
The table entails the Test RMS and Mean Absolute errors across the test data followed by graphs on expected vs actual price prediction across time as a Price vs Time graph

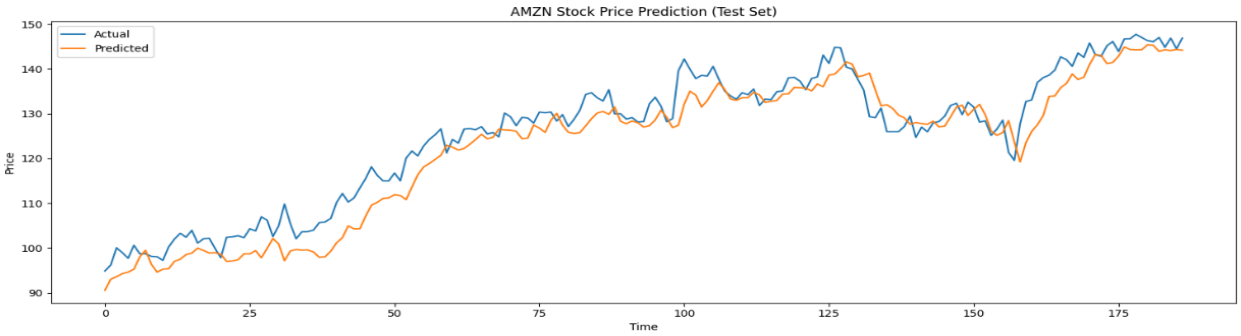
Stock	**Test RMSE**	**Test MAE**
MSFT	30.813	28.849
GOOGL	4.024	3.050
TSLA	13.856	10.735
AMZN	4.555	3.695
AAPL	8.139	7.250


AAPL Stock Price Prediction (Test Set)
Actual
Predicted
Price
Time


MSFT Stock Price Prediction (Test Set)
Actual
Predicted
Price
Time


GOOGL Stock Price Prediction (Test Set)
Actual
Predicted
Price
Time


TSLA Stock Price Prediction (Test Set)
Actual
Predicted
Price
Time


AMZN Stock Price Prediction (Test Set)
Actual
Predicted
Price
Time

UI Dashboard

In order to help with easily visualizing the stock prices' predicted values, we have built a dashboard using FastAPI and Dash. FastAPI is used as the backend service to handle model inference and data processing. It provides a lightweight, high-performance REST ful API through which stock prediction results can be requested dynamically.

Dash, a Python-based web application framework built on top of Flask and Plotly, is used to design the interactive frontend dashboard. Dash enables the visualization of time-series data through dynamic graphs, allowing users to:

1. Select individual stocks
2. View actual versus predicted closing prices

The image below is a screenshot of how the dashboard looks from the user perspective.

Conclusion

On comparing the results of both approaches, the error has increased for MSFT and AAPL stocks but it has decreased for the rest by decently significant amounts. GOOGL and AMZN demonstrate strong generalization performance, with low RMSE and MAE values, indicating that their price movements are relatively stable and easier to model using shared temporal representations. TSLA continues to show higher prediction error due to its high volatility and abrupt price changes, which are difficult to capture using Simple RNNs. The improved performance in GOOGL and AMZN suggests that correlated market patterns learned from multiple stocks enhances generalization.MSFT's performance degraded in the multi-stock model, indicating negative transfer, where shared representations adversely affect certain stocks.TSLA remains challenging in both approaches, reinforcing the need for richer features or advanced architectures. In order to improve performance the following could have been used:

1. LSTMS or GRUs to better handle long-term dependencies.
2. Incorporate additional features such as volume and technical indicators as only closing prices were used.
3. Introduce stock-specific output heads on top of shared layers i.e. have a neural network architecture wherein the first few layers i.e. the early ones are to learn the common patterns and the final layers being stock specific.