Computer Science Extended Essay

“The use of Neural Networks in predicting share prices”

Research Question:

To what extent is Artificial Neural Network efficacious in predicting future share prices of firms listed on the stock exchange?

Word count: 3969

Table of Contents

1 INTRODUCTION 4

2 BACKGROUND OF NEURAL NETWORK 6

2.1 Artificial Intelligence 6

2.2 What is Machine Learning? 6

2.2.1 Types of machine learning 6

2.2.2 Supervised Learning 7

2.3 Definition of Deep Learning 7

2.4 Definition of Artificial Neural Network 8

2.5 Types of Neural Networks 9

2.6 Activation Function - ReLU 10

2.6 Loss Function – Mean Squared Error 10

2.7 Optimizer Algorithm - ADAM 11

2.8 Training the Network using Backward Propagation 12

3 METHODOLOGY 13

3.1 Experimental Procedure 13

3.2 Code Structure 14

3.3 Dataset used 17

3.4 Libraries used 18

3.4.1 Matplotlib 18

3.4.2 NumPy 18

3.4.3 Pandas 18

3.4.4 OS 18

3.4.5 Warnings 18

3.4.6 Sklearn 19

3.4.7 Tensorflow 19

3.4.8 Keras 19

3.5 Model Architecture 20

3.6 Parameters used 22

3.6.1 Units = 50 22

3.6.2 Activation = 'relu' *22*
3.6.3 Lookback = 20 ..... *22*
3.6.4 batch_size = 20 ..... *22*
3.6.5 Epochs = 100 ..... *23*
3.6.6 Verbose = 1 ..... *23*
3.6.7 Patience = 3 ..... *23*
3.6.8 save_best_only = True *24*
3.6.9 Mode = 'min' *24*
3.6.10 Loss = 'mean_squared_error' *24*

4 RESULTS **25**

4.1 Historical Stock Price Graph 25
4.2 Training vs Testing graph 25
4.3 Predicted stock prices 26
4.4 Loss and Improvement 27
4.5 Different Metrics 30

5 ANALYSIS **31**

6 DISCUSSION AND EVALUATION **33**

7 CONCLUSION **34**

8 WORKS CITED **35**

8.1 Dataset Used 35
8.2 Articles regarding Neural Networks 35
8.3 Research Papers, Books and other sources 37
8.4 Articles about Stock Markets 37

9 APPENDICES **38**

9.1 Python Code in Kaggle 38
9.2 Output 43
9.2.1 Output when the Patience = 3 43
9.2.2 Output when the Patience = 4 53

1 Introduction

People's interest in the stock market has increased exponentially over the last few years. Billions of dollars' worth of stock are traded every day with the basic goal of earning profit. In a free-market economy, stock markets play a vital role by enabling the democratization of investment and capital exchange.[1] They promote investment and also help companies to raise capital for businesses to grow, expand, create jobs and grow the economy as a whole.

There are several ways to predict share prices, but it is very difficult to accurately do it because of the volatility and non-linear nature of the stock exchange. This makes it very difficult for traditional linear models to accurately predict share prices.

Machine learning algorithm known as Neural Networks simulate workings of a human brain and are utilized to model intricate patterns in data. They are capable of recognizing patterns from the input data. Therefore, they have been shown to be particularly effective in predicting share prices because of their ability to handle non-linear and large amounts of data.

In this essay, I will aim to investigate and answer the research question, "**To what extent is Artificial Neural Network efficacious in predicting future share prices of firms listed on the stock exchange?"**

This research question is worthy of the EE because it explores the modern and developing field of machine learning and neural network and the pre-existent field of the stock market.

[1] ----. "Stock Market | Investopedia." *Investopedia*, 12 Mar. 2022, www.investopedia.com/terms/s/stockmarket.asp.

This essay, therefore, attempts to answer the question by evaluating experiments and studying theories related to it in order to build an artificial neural network to predict the stock market. The model will be trained using datasets of historical data of different companies. The data will then be compared and analyzed to understand the efficiency and accuracy of its use in predicting stock prices.

2 Background of Neural Network

2.1 Artificial Intelligence

John McCarthy defined Artificial Intelligence as the science of creating and engineering intelligent machines and computer programs. AI is considered to be similar to the task of using computers to comprehend and use human intelligence, but it doesn't limit itself to biologically observable methods.

Therefore, in simpler terms, AI is the technology that performs tasks that would normally require human intelligence to perform but are now able to do on its own.

2.2 What is Machine Learning?

Machine learning's primary objective is to develop models that can learn and improve themselves, recognize complex patterns, and find solutions to new problems by utilizing past data. [2] Machine learning is the Artificial Intelligence that teaches the computer to function the same way as a human think. It then uses its self-learning to improve from its past experiences to provide the best possible outcome.

2.2.1 Types of machine learning

The three types of machine learning are:

- Reinforced
- Supervised
- Unsupervised

For predicting stock prices the model will need to be provided with historical data, hence it will be a type of supervised machine learning.

---

[2] Çelik, Özer. "A Research on Machine Learning Methods and Its Applications." *Journal of Educational Technology and Online Learning*, 9 Sept. 2018, https://doi.org/10.31681/jetol.457046.

2.2.2 Supervised Learning

Supervised learning involves training the model using data that has been explicitly labelled, meaning that large dataset of labeled examples is provided and then the model is evaluated for its performance on a separate dataset of unseen examples. The objective of supervised learning is to develop a model that can effectively handle new, unseen data.

It can be divided into two main categories: classification and regression.

Classification

It predicts a discrete outcome and classifies the data into different categories based on the values of one or more independent variables. The dependent variable in classification is a categorical variable.

Regression

It predicts a continuous outcome and is used to model the association between independent variables and dependent variable (outcome).

Since, regression helps in predicting a continuous variable, to predict the stock prices in the essay, it will be more beneficial to use.

2.3 Definition of Deep Learning

It is a subset of machine learning aimed to work using algorithms that are concerned with modelling high-level abstractions in data. The ANN algorithms are inspired by and modelled based on the functioning of the human brain. The benefits of deep learning algorithms are that they can analyse unstructured data, and they can also be used to perform several tasks at once, with greater accuracy and efficiency compared to humans.

When referring to a deep learning algorithm, the word "deep" denotes a neural network that comprises more than three layers, including the input and output layers.

2.4 Definition of Artificial Neural Network

Artificial neural network is a branch of machine learning that utilizes a sequence of algorithms designed to imitate the operations of the human brain, with the objective of recognizing underlying correlations and relationships in given set of data. It consists of interconnected processing nodes, or neurons, arranged in layers that learn to detect patterns in input data.

ANNs have the ability to learn and model complex and nonlinear relationships. After being trained on a small set of inputs and their relationships, the model can extrapolate to new, unrelated data, allowing the model to generalize and make predictions based on new data. It does not have any limitations on the input variables and has been shown to effectively handle data with high volatility and non-constant variance, as they can discover hidden patterns in the data without predetermined relationships. This makes it well-suited in forecasting financial time series where there is high volatility in the data. Therefore, ANNs can be used in image processing, character recognition and forecasting. Thus, ANNs can be trained to predict the stock market prices using the provided data sets, which will be discussed and covered later on in the essay.

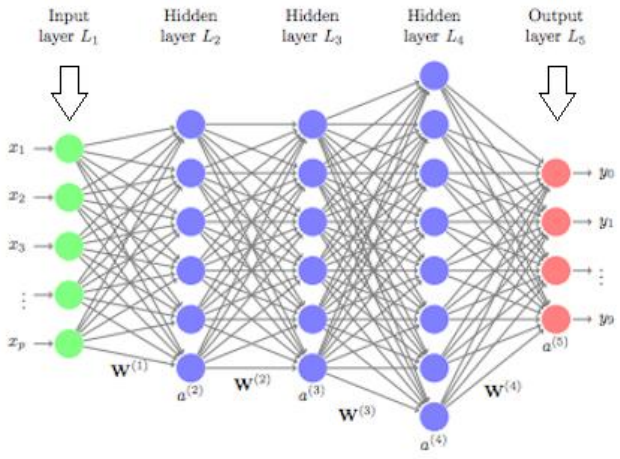


Figure 1: Artificial Neural Network [3]

[3] *Orbograph.com*, 2023, orbograph.com/wp-content/uploads/2019/01/DeepLearn.png.

2.5 Types of Neural Networks

The neural networks are usually classified based on their layers, structure, depth activation filters, neurons, etc:

a) Long Short-Term Memory (LSTM) Network
b) Perceptron
c) Convolutional Neural Network
d) Feed Forward Neural Networks
e) Multilayer Perceptron
f) Recurrent Neural Network
g) Modular Neural Network

To perform the prediction of the stock prices in this essay, we will use LSTM Networks. LSTMs employ a set of gates which manage the flow of data through the network in a sequence. There are three gates that make up a standard LSTM: the forget gate, the input gate, and the output gate. These gates act as filters and each one is a separate neural network. The LSTM cell also has a hidden state, which is passed from one LSTM cell to the next and used to store information about the previous input.

The model uses a sequence of input data and the hidden state to update the cell state and output the prediction. The input data is passed through the input gate and the forget gate to update the cell state. It is then passed through the output gate to produce the output.

2.6 Activation Function - ReLU

For every artificial neuron, the activation function determines if the incoming signals have surpassed the required threshold and, if so, whether to produce signals for the next level. [4] Many types of activation functions exist, such as Sigmoid, ReLU, and Tanh.

For this essay we will be using Rectified Linear Unit (ReLU). It is a piece-wise linear function that outputs the positive values as it is, but outputs all the negative values as zero. This enables the model to learn non-linear input-output relationships.

x is the input to the neutron where the function is $f(x) = max(0, x)$.

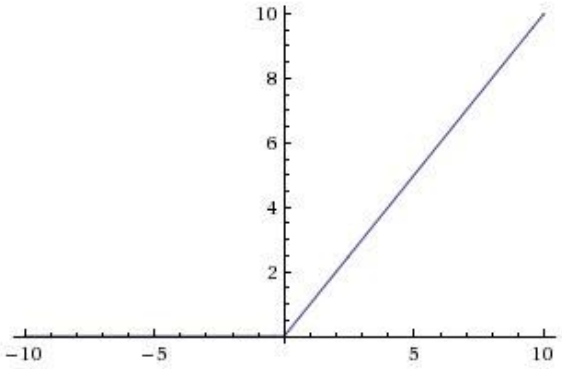


Figure 2: Graphical Representation of a ReLU function [5]

2.6 Loss Function – Mean Squared Error

Loss is a metric that quantifies the total errors in a model, it reflects how well the model is doing its job. When the errors are high, the loss will also be high. The objective is to minimize the loss, since a smaller loss indicates that the model is more successful at performing the task.

---

[4] Di, Wei, et al. *Deep Learning Essentials : Your Hands-on Guide to the Fundamentals of Deep Learning and Neural Network Modeling*. Birmingham, UK, Packt Publishing, 2018.

[5] "Rectified Linear Units (ReLU) in Deep Learning." *Kaggle.com*, www.kaggle.com/code/dansbecker/rectified-linear-units-relu-in-deep-learning/notebook.

There are a variety of loss functions available, selecting the loss function depends on the intended use of the neural network. For this essay, we will be using mean squared error (MSE) and root mean squared error (RMSE).

Mean squared error (MSE) is a widely-used basic loss function due to its simplicity, ease of implementation, and overall effectiveness. MSE is calculated by first finding the difference between the actual results and their corresponding predicted values, then squaring them, and taking the average of all these differences over the entire dataset.

The equation of MSE will be:

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

2.7 Optimizer Algorithm - ADAM

Optimizer Algorithms are used to minimize a loss function or to maximize the output efficiency. They are mathematical functions that depend on the model's adjustable parameters and assist in determining the changes needed in weights and learning rate to reduce the loss in the neural network.

There are several different types of optimizers, but the best optimizer that can be used to have efficiency and less time is ADAM, hence the code will include ADAM as the optimizer algorithm. It is used to adjust the weights of the model to minimize the prediction error. ADAM algorithm builds on the classical stochastic gradient descent algorithm as it uses the concept of “adaptive learning rates” which means that it updates the learning rates of the model during training, based on the historical gradient information.

2.8 Training the Network using Backward Propagation

After the input data and network architecture are set up, training starts by initializing the weights with random values. Backward Propagation refers to the process of adjusting the weights during each epoch to minimize the difference between the predicted and actual outputs in the network. By iteratively adjusting its weights, a neural network can gradually improve its performance over time, using the feedback it receives from its predicted outputs.

The gradient of error is calculated with respect to the weights of the network, and is then adjusted in the opposite direction of the gradient to reduce error. It is repeated until the error is minimized, or the algorithm reaches a predefined stopping criterion. The algorithm starts by forwarding the input data through the network to produce an output. The error is therefore understood to be the difference between the actual output and the predicted output. The error is then propagated backwards through the network, beginning from the output layer and working backwards through hidden layers. [6]

[6] "Backward Propagation | Backward Propagation Working in Neural Network." *Analytics Vidhya*, 1 June 2021, www.analyticsvidhya.com/blog/2021/06/how-does-backward-propagation-work-in-neural-networks/.

3 Methodology

3.1 Experimental Procedure

1. Import necessary libraries and modules
2. Load the historical stock price data
3. Plot and visualize the historical stock price
4. Parameters for the training process defined
5. Split the data into training and testing sets
6. Scale the training and testing sets
7. Create and train the LSTM model using the training data
8. Evaluate the LSTM model using the test data and calculate the mean squared error as a performance metric

Running a model of this depth on a normal CPU-based machine is highly challenging and takes a lot of time. Therefore, I decided to use Kaggle to train and run the model because it offers hardware accelerators that speed up the training process.

3.2 Code Structure

The overall structure of the code can be divided into several parts:

Data Loading and Preprocessing:

The code begins by importing the necessary libraries and loading the data from the specified CSV file. It then checks the properties of the dataframe using the check_df function, which prints the shape, types, head, tail, NA values and quantiles. The code then converts the 'Date' column into a datetime type and removes it from the dataframe. It then only selects the 'Close' column and assigns it to company_data.

```
data=pd.read_csv("../input/amazondata2/AMZN.csv")

data.head()

def check_df(dataframe,head=5):
    print("##################### Shape ##################### ")
    print(dataframe.shape)
    print("##################### Types ##################### ")
    print(dataframe.dtypes)
    print("##################### Head ##################### ")
    print(dataframe.head(head))
    print("##################### Tail ##################### ")
    print(dataframe.dtypes)
    print("##################### NA ##################### ")
    print(dataframe.isnull().sum())
    print("##################### Quantiles ##################### ")
    print(dataframe.quantile([0,0.5,0.50,0.95,0.99,1]).T)


check_df(data)

data["Date"]=pd.to_datetime(data["Date"])

data.head()

company_data=data[["Date","Close"]]

company_data.head()
```

Code 1: Data Preprocessing

Data Visualization:

Matplotlib library is used to create a line graph of the "Close" column of the dataframe "company_data" to visualize the historical stock prices. The y-axis represents the stock price and the x-axis represents the time. The graph is then displayed using the "show()" function.

```
print("Min. Date:",company_data["Date"].min())
print("Max. Date:",company_data["Date"].max())

company_data.index=company_data["Date"]


company_data.drop("Date",axis=1,inplace=True)

result_data=company_data.copy()

plt.figure(figsize=(12,6))
plt.plot(company_data["Close"],color="blue");
plt.ylabel("Stock Price")
plt.title("Amazon Stock Price")
plt.xlabel("Time")
plt.show()
```

Code 2: Graph of Historical Stock Prices

Data Preparation

The code then converts company_data to a numpy array and sets the data type to float32.

```
company_data=company_data.values

company_data[0:5]

company_data=company_data.astype("float32")
```

Code 3: Converting to float32

In addition, the model applies data preparation (a subset of data preprocessing) to a dataframe called "company data" by first dividing the data into train and test sets with a 20% defined test size. The data is then scaled for both the train set and the test set using the MinMaxScaler, with a range of 0 to 1.

The function "create features" is then defined which accepts the data and a "lookback" value that is given as 20. A set of input features and intended outcomes are produced. It accomplishes this by repeatedly going through the data, starting from the lookback value, and generating a set of input features made up of the number of lookback data points that came before, as well as a target output for each set of input features.

```
def split_data(dataframe,test_size):
    pos=int(round(len(dataframe)*(1-test_size)))
    train=dataframe[:pos]
    test=dataframe[pos:]
    return train,test,pos

train,test,pos=split_data(company_data,0.20)

print(train.shape,test.shape)

scaler_train=MinMaxScaler(feature_range=(0,1))

train=scaler_train.fit_transform(train)

scaler_test=MinMaxScaler(feature_range=(0,1))

test=scaler_test.fit_transform(test)

train[0:5]

test[0:5]

def create_features(data,lookback):
    X,Y=[],[]
    for i in range(lookback,len(data)):
        X.append(data[i-lookback:i,0])
        Y.append(data[i,0])
    return np.array(X),np.array(Y)

lookback=20
```

Code 4: Splitting and Scaling data

3.3 Dataset used

The dataset used in this code is historical stock price data for Amazon. The dataset is used to train and evaluate the LSTM model.

The dataset contains several columns:

- Date: represents the date of the relevant Transaction Day
- Open: refers to the initial share price at start of the relevant Trading Day
- High: refers to the highest price of the relevant Trading Day
- Low: refers to the lowest price of the relevant Trading Day
- Close: refers to the closing price of the stock on the relevant Trading Day. It is used as the target variable to predict the stock price.
- Adj Close: represents the adjusted closing price of the stock of the relevant Trading Day
- Volume: refers to the trading volume information of the relevant Trading Day

3.4 Libraries used

3.4.1 Matplotlib

It is a library that is used to create visual representations of data in the form of graphs such as including line plots, scatter plots, bar plots, histograms and more.

3.4.2 NumPy

It is a Python library that allows for the manipulation and mathematical operations on multidimensional matrices and arrays, including trigonometric, statistical, and algebraic routines.

3.4.3 Pandas

Pandas is an advanced and versatile data analysis package that streamlines the preprocessing stage of projects. It also offers a wide range of functionalities and techniques for data analysis.

3.4.4 OS

Module allows the users to access the operating system dependent functionality such as reading or writing to files. It provides a variety of useful functions for interacting with the file system, executing shell commands, and working with environment variables.

3.4.5 Warnings

It is a library that allows to issue warnings. It is used to filter warnings and ignore them.

3.4.6 Sklearn

It is a library used for machine learning and data preprocessing. It splits the data into training and testing sets, scales the data, and calculates the mean squared error.

3.4.7 Tensorflow

It is a library for building, training and deploying machine learning models. It is used to build the LSTM model and train the model.

3.4.8 Keras

It is an open-source Python library for deep neural networks, that is optimized for speed and is focused on extensibility, modularity, and ease of use.

```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,LSTM,Dropout
from tensorflow.keras.callbacks import ModelCheckpoint,EarlyStopping

import warnings
warnings.filterwarnings("ignore")

import os
os.environ["TF_CPP_MIN_LOG_LEVEL"]="3"
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
```

Code 6: All the Libraries and Modules Imported and Used

3.5 Model Architecture

The model is made using a sequential model API. The LSTM layer, Dropout layer, and Dense layer are added to the model. The sequential data is processed by the LSTM layer, regularization is applied by the Dropout layer to avoid overfitting, and the final predictions are generated by the Dense layer, which is the output layer of the model.

The ReLU activation function helps the model to converge faster and performs better than other activation functions. Additionally, it produces sparse representations, which can improve the interpretability of the model.

ADAM is used in this code because it is an efficient optimization algorithm that is well-suited for training large neural networks. It has also been shown to work well in practice for a wide range of problems, including training deep neural networks.

The Dense layer uses the linear activation function, which is the identity function that maps any input value to the same output value. In the dense layer the number of units is set to 1, which means that the output of the model will be a single value.

The Dropout layer in this code does not use an activation function. It regularizes data and prevents overfitting by setting a fraction of input units to 0 during training. The dropout rate (the fraction of input units) affects the ability of the model to generalize to new data.
It is set to 0.2, which means that during training, 20% of the input units will be set to 0. This technique prevents the network from relying too heavily on any one feature and forces the model to learn multiple independent representations of the data, which helps prevent overfitting.

```
model=Sequential()
model.add(LSTM(units=50,activation="relu",
               input_shape=(X_train.shape[1],lookback)))
model.add(Dropout(0.2))
model.add(Dense(1))

model.summary()

model.compile(loss="mean_squared_error",optimizer="adam")
```

Code 7: Establishing Model Architecture

3.6 Parameters used

3.6.1 Units = 50

It represents the number of neurons or units within a LSTM layer. The model's ability to learn and represent complex relationships in the data is determined by the number of neurons in the layer. Depending on the problem's complexity and the amount of available training data, the ideal number of neurons in a layer varies.

3.6.2 Activation = 'relu'

It specifies the activation function that has to be used for each output.

3.6.3 Lookback = 20

It is used to define the lookback window. Which means that it defines the number of previous time steps to include as input features in each sample. In this case, the value of Lookback is set to 20, meaning that each sample of the input data will consist of 20 consecutive time steps of the input data, which are used as features to make a prediction.

3.6.4 batch_size = 20

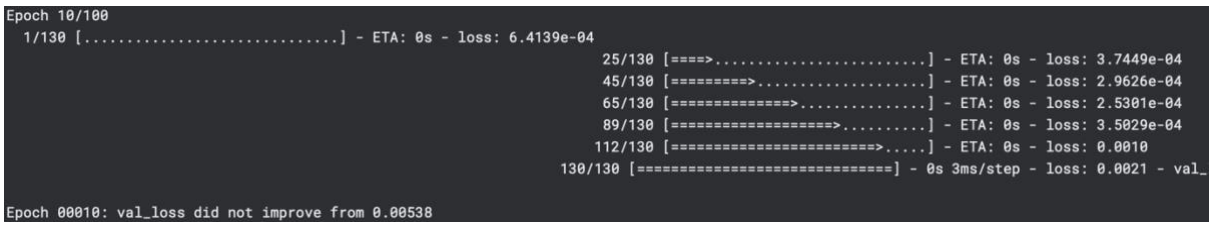
In training, it indicates the number of samples that undergo a single forward/backward pass before the model's parameters are updated. The batch_size is set to 20, meaning that 20 samples will be processed by the model in each forward and backward pass during the training process.

3.6.5 Epochs = 100

An epoch is a complete iteration over the entire training data. It indicates the number of complete cycles that the model will make over the entire training dataset during the training process. In this case, epoch is set to 100, meaning that it will run the entire training data 100 times. This enables the model to understand the underlying patterns in the data and produce accurate forecasts. Thus, the amount of available training data and the difficulty of the task at hand should both be considered while settling on the optimal number of epochs. Having a high epoch value may result in overfitting, whereas too few may lead to a badly-trained, less effective model.

3.6.6 Verbose = 1

It is an optional argument in the model training process which specifies the level of logging that should be performed while training the model. The value of "Verbose" is an integer ranging from 0 to 2. In the case of "Verbose = 1", it means that the training process will output progress bar-style logging for each epoch.



Output 1: Verbose = 1 example with a sample test run

3.6.7 Patience = 3

It is used in the model training process to determine the number of epochs to wait before early stopping. It sets the number of epochs without improvement after which training will be stopped. For example, Patience = 3, it means that the training process will stop if the model does not show improvement for 3 consecutive epochs. This helps to avoid overfitting.

3.6.8 save_best_only = True

This parameter with a value of "True" is used in the model's call to the ModelCheckpoint() function. It determines whether to save the model when it has the best performance as determined by the monitored metric. If set to "True", only the best model weights are saved, overwriting the previous best model. If set to "False", all intermediate model weights will be saved, resulting in a potentially large number of saved models.

3.6.9 Mode = 'min'

It is used to specify the mode in which the EarlyStopping callback operation should work. It determines whether the training should be stopped when the monitored metric improves or worsens. The value of 'min' specifies that the training should be stopped if the monitored metric becomes lower than the best value it has achieved so far.

3.6.10 Loss = 'mean_squared_error'

This specifies criterion used to measure and evaluate the model's performance during the training process. In this case, the loss function used is the mean_squared_error.

```
model=Sequential()
model.add(LSTM(units=50,activation="relu",
               input_shape=(X_train.shape[1],lookback)))
model.add(Dropout(0.2))
model.add(Dense(1))

model.summary()

model.compile(loss="mean_squared_error",optimizer="adam")

callbacks=[EarlyStopping(monitor="val_loss",patience=3,verbose=1,mode="min"),
           ModelCheckpoint(filepath="mymodel.h5",monitor="val_loss",mode="min",
                           save_best_only=True,save_weights_only=False,verbose=1)]

history = model.fit(x=X_train,
                    y=y_train,
                    epochs=100,
                    batch_size=20,
                    validation_data=(X_test,y_test),
                    callbacks=callbacks,
                    shuffle=False)
```

Code 8: Parameters used

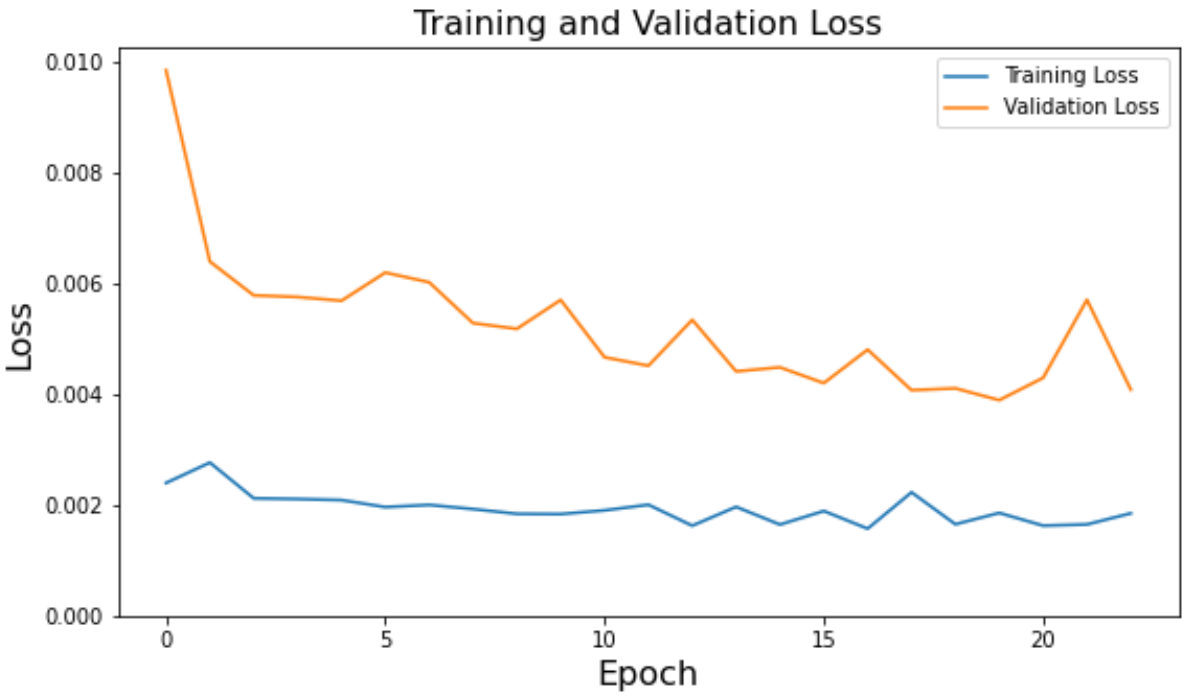
4 Results

Here the Matplotlib was used to plot the graphs for historical stock price, training vs validation loss and predicted stock prices.

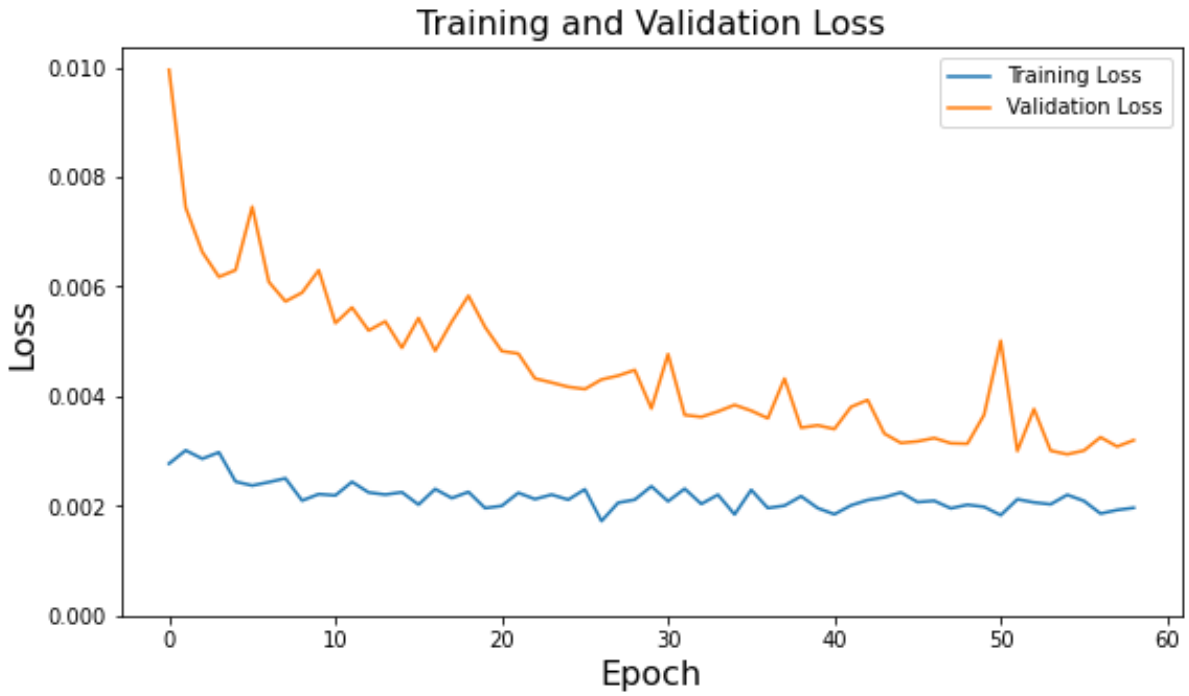
4.1 Historical Stock Price Graph

Graph 1: Stock price from 23th Jan 2010 to 23th Jan 2023

4.2 Training vs Testing graph

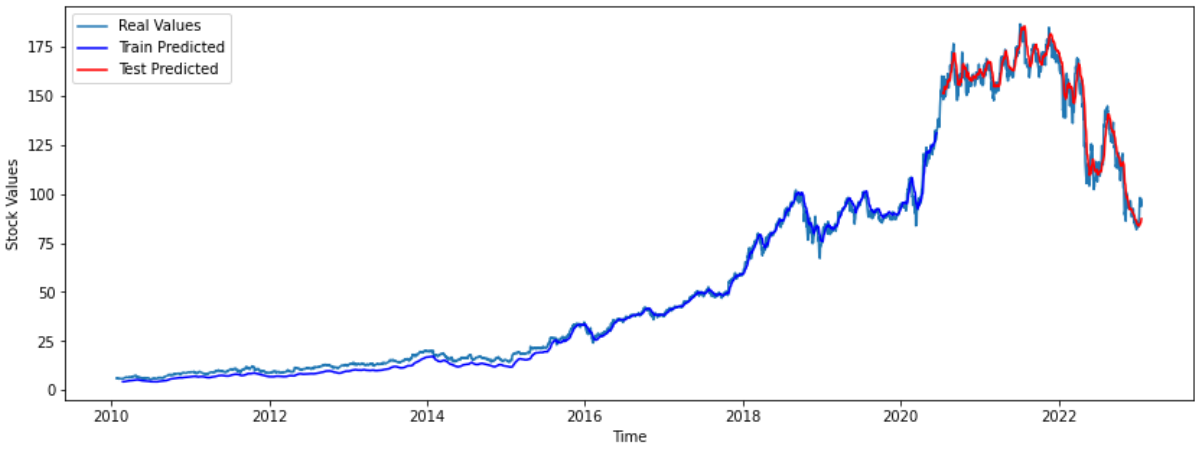


Graph 2: When the Patience = 3

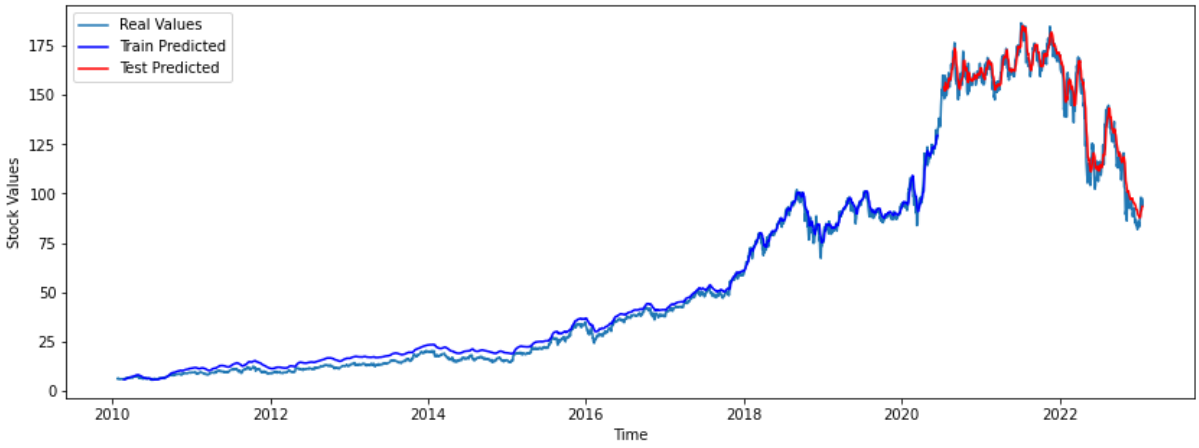


Graph 3: When the Patience = 4

4.3 Predicted stock prices



Graph 4: When the Patience = 3



Graph 5: When the Patience = 4

4.4 Loss and Improvement

epoch	Val_loss	Improvement?
1	0.00986	yes
2	0.00640	yes
3	0.00579	yes
4	0.00576	yes
5	0.00569	yes
6	0.00620	no
7	0.00600	no
8	0.00529	yes
9	0.00518	yes
10	0.00570	no
11	0.00467	yes
12	0.00452	yes
13	0.00530	no
14	0.00442	yes
15	0.00450	no
16	0.00420	yes
17	0.00480	no
18	0.00408	yes
19	0.00410	no
20	0.00390	yes
21	0.00430	no
22	0.00570	no
23	0.0041	no
Early Stopping		

Table 1: When the Patience = 3

epoch	Val_loss	Improvement?
1	0.00995	Yes
2	0.00744	Yes
3	0.00663	Yes
4	0.00618	Yes
5	0.00630	No
6	0.00750	No
7	0.00609	Yes
8	0.00573	Yes
9	0.00590	No
10	0.00630	No
11	0.00553	Yes
12	0.00560	No
13	0.00520	Yes
14	0.00540	No
15	0.00488	Yes
16	0.00540	No
17	0.00483	Yes
18	0.00540	No
19	0.00580	No
20	0.00530	No
21	0.00482	Yes
22	0.00478	Yes
23	0.00432	Yes
24	0.00425	Yes
25	0.00417	Yes
26	0.00413	Yes
27	0.00430	No
28	0.00440	No

29	0.00450	No
30	0.00378	Yes
31	0.00480	No
32	0.00366	Yes
33	0.00362	Yes
34	0.00370	No
35	0.00380	No
36	0.00370	No
37	0.00360	Yes
38	0.00430	No
39	0.00343	Yes
40	0.00350	No
41	0.00340	Yes
42	0.00380	No
43	0.00390	No
44	0.00332	Yes
45	0.00315	Yes
46	0.00320	No
47	0.00320	No
48	0.00314	Yes
49	0.00314	Yes
50	0.00370	No
51	0.00500	No
52	0.00300	Yes
53	0.00380	No
54	0.00300	No
55	0.00294	Yes
56	0.00300	No
57	0.00330	No

58	0.00310	No
59	0.00320	No
Early Stopping		

Table 2: When the Patience = 4

4.5 Different Metrics

Test Loss	0.4%
Train RMSE	2.7260358333587646
Test RMSE	6.695877552032471

Table 3: Loss and Error when the Patience = 3

Test Loss	0.3%
Train RMSE	2.995021343231201
Test RMSE	5.92483377456665

Table 4: Loss and Error when the Patience = 4

5 Analysis

The test loss is low for all three values of patience. In the first table (patience = 3), the model has a test loss of 0.4% and a train RMSE of around 2.73. The test RMSE is approximately 6.70, which is higher than the train RMSE, indicating that the model is overfitting to the training data.

In the second table (patience = 4), the model has a lower test loss of 0.3% and higher train RMSE of 3.00. The test RMSE is 5.92, which is still higher than the train RMSE, but it is lower compared to the test RMSE in the first table. This suggests that increasing the patience value from 3 to 4 improved the model's generalization performance, as it resulted in a lower error on the test set.

Overall, it can be understood that the second model (patience = 4) performed better than the first model (patience = 3) based on the lower test loss and lower test RMSE.

The model stops after epoch 23 when Patience = 3, because of the EarlyStopping callback function, which is set up to monitor the validation loss and stop the training if the validation loss stops improving for a certain number of epochs.

The validation loss (val_loss) is compared with the previous validation loss and in this case, it did not improve for 3 consecutive epochs, the training process is stopped. This helps save computational resources and prevent overfitting.

The test RMSE is greater than the train RMSE. This suggests that the model is overfitting to the training data, meaning that while the model is able to fit the training data well, it is not able to perform and generalize well to new, unseen data (i.e., the test data).

The results may not be optimal due to limited parameter tuning, as demonstrated by the varying results from just altering one parameter (patience). This suggests that there could be other methods for improving the results.

From the graphs, it is observed that the validation loss is converging with training loss as the number of epochs increases in the chart. When validation loss begins to diverge from training loss, this is again a sign of overfitting, and under most training methods, training is stopped at this point.

6 Discussion and Evaluation

The limitation in my model was the datasets used and the limited amount of information it provided. I used data datasets with historical data of the past 13 years, which I could have taken of even more years. The model did not take into consideration the external factors such as the current real-world news. This is a huge drawback in the code because breaking news such war, natural disaster, major political event and other changing influencing factors are not included, hence the resultant possible changes in the stock prices are not factored in the model, causing few inaccuracies and reduced precision.

The parameters could have been checked by changing its values to determine the best values for the model to achieve minimum loss.

The low processing power of the computer limited the number of epochs that could be run. Hence, the size of the model and the amount of data drastically reduced. This technical limitation however can be overcome by increasing the processing power, meaning the training for ANNs will be improved and enhanced.

7 Conclusion

The aim of the investigation was to predict the prices of stocks with the help of machine learning and neural networks. This was done by providing historical datasets for the stocks. After training and testing the data, the results showed that the investigation can be considered to be successful because of the low value of test loss and high similarity of predicted graph and actual graph of the stock market.

In conclusion, ANNs have shown promise in predicting future share prices. From the experiment it has been observed that they are effective in identifying patterns and trends in large data sets and have the ability to learn and adapt to changing market conditions. However, the accuracy of ANNs in predicting stock prices is dependent on the quality and quantity of the data used in the model, the chosen parameters, and the volatility of the stock market.

Therefore the use of Neural Networks therefore, is a great addition to the programming world and society as a whole, as it will help in various different fields as well.

8 Works Cited

8.1 Dataset Used

"Amazon.com, Inc. (AMZN) Stock Historical Prices & Data." *Yahoo! Finance*, Yahoo!, 21 Feb. 2023, finance.yahoo.com/quote/AMZN/history?p=AMZN.

8.2 Articles regarding Neural Networks

Brown, Sara. "Machine Learning, Explained." *MIT Sloan*, 21 Apr. 2021, mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained.

Gupta, Sakshi. "Regression vs. Classification in Machine Learning: What's the Difference?" *Springboard Blog*, 6 Oct. 2021, www.springboard.com/blog/data-science/regression-vs-classification/.

"ML | Classification vs Regression - GeeksforGeeks." *GeeksforGeeks*, 8 Jan. 2019, www.geeksforgeeks.org/ml-classification-vs-regression/.

IBM Cloud Education. "What Is Artificial Intelligence (AI)?" *IBM*, 3 June 2020, www.ibm.com/cloud/learn/what-is-artificial-intelligence.

Chen, James. "Neural Network Definition." *Investopedia*, 2019, www.investopedia.com/terms/n/neuralnetwork.asp.

Jahnavi Mahanta. "Introduction to Neural Networks, Advantages and Applications." *Medium*, Towards Data Science, 10 July 2017, towardsdatascience.com/introduction-to-neural-networks-advantages-and-applications-96851bd1a207.

"ReLU (Rectified Linear Unit) Activation Function." *OpenGenus IQ: Computing Expertise & Legacy*, 30 Dec. 2021, iq.opengenus.org/relu-activation/.

"Rectified Linear Units (ReLU) in Deep Learning." *Kaggle.com*, www.kaggle.com/code/dansbecker/rectified-linear-units-relu-in-deep-learning/notebook.

Seif, George. "Understanding the 3 Most Common Loss Functions for Machine Learning Regression." *Medium*, 3 Oct. 2021, towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3.

Doshi, Sanket. "Various Optimization Algorithms for Training Neural Network." *Medium*, 3 Aug. 2020, towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6.

Musstafa. "Optimizers in Deep Learning." *MLearning.ai*, 12 Feb. 2022, medium.com/mlearning-ai/optimizers-in-deep-learning-7bf81fed78a0.

"Backward Propagation | Backward Propagation Working in Neural Network." *Analytics Vidhya*, 1 June 2021, www.analyticsvidhya.com/blog/2021/06/how-does-backward-propagation-work-in-neural-networks/.

"Run Data Science & Machine Learning Code Online." *Kaggle*, www.kaggle.com/code.

"Batch Size in a Neural Network Explained." *Deeplizard.com*, deeplizard.com/learn/video/U4WB9p6ODjM. Accessed 13 July 2021.

Brownlee, Jason. "Use Early Stopping to Halt the Training of Neural Networks at the Right Time." *Machine Learning Mastery*, 9 Dec. 2018, machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/.

8.3 Research Papers, Books and other sources

Unimi.it.borghese.di.unimi.it/Teaching/AdvancedIntelligentSystems/Old/IntelligentSystems_2008_2009/Old/IntelligentSystems_2005_2006/Documents/Symbolic/04_McCarthy_whatisai.pdf.

Çelik, Özer. "A Research on Machine Learning Methods and Its Applications." *Journal of Educational Technology and Online Learning*, 9 Sept. 2018, https://doi.org/10.31681/jetol.457046.

Orbograph.com, 2023, orbograph.com/wp-content/uploads/2019/01/DeepLearn.png.

Di, Wei, et al. *Deep Learning Essentials : Your Hands-on Guide to the Fundamentals of Deep Learning and Neural Network Modeling*. Birmingham, UK, Packt Publishing, 2018.

8.4 Articles about Stock Markets

---. "Stock Market | Investopedia." *Investopedia*, 12 Mar. 2022, www.investopedia.com/terms/s/stockmarket.asp.

"The Importance of Stock Markets." *Finance.yahoo.com*, finance.yahoo.com/news/importance-stock-markets-075948914.html.

9 Appendices

9.1 Python Code in Kaggle

```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,LSTM,Dropout
from tensorflow.keras.callbacks import ModelCheckpoint,EarlyStopping

import warnings
warnings.filterwarnings("ignore")

import os
os.environ["TF_CPP_MIN_LOG_LEVEL"]="3"
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

data=pd.read_csv("../input/amazondata2/AMZN.csv")

data.head()

def check_df(dataframe,head=5):
    print("##################### Shape #################### ")
    print(dataframe.shape)
    print("##################### Types #################### ")
    print(dataframe.dtypes)
    print("##################### Head #################### ")
    print(dataframe.head(head))
    print("##################### Tail #################### ")
    print(dataframe.dtypes)
    print("##################### NA #################### ")
    print(dataframe.isnull().sum())
    print("##################### Quantiles #################### ")
    print(dataframe.quantile([0,0.5,0.50,0.95,0.99,1]).T)


check_df(data)

data["Date"]=pd.to_datetime(data["Date"])

data.head()
```

```
company_data=data[["Date","Close"]]

company_data.head()

print("Min. Date:",company_data["Date"].min())
print("Max. Date:",company_data["Date"].max())

company_data.index=company_data["Date"]


company_data.drop("Date",axis=1,inplace=True)

result_data=company_data.copy()

plt.figure(figsize=(12,6))
plt.plot(company_data["Close"],color="blue");
plt.ylabel("Stock Price")
plt.title("Amazon Stock Price")
plt.xlabel("Time")
plt.show()


company_data=company_data.values

company_data[0:5]

company_data=company_data.astype("float32")


def  split_data(dataframe,test_size):
     pos=int(round(len(dataframe)*(1-test_size)))
     train=dataframe[:pos]
     test=dataframe[pos:]
     return  train,test,pos

train,test,pos=split_data(company_data,0.20)

print(train.shape,test.shape)

scaler_train=MinMaxScaler(feature_range=(0,1))

train=scaler_train.fit_transform(train)

scaler_test=MinMaxScaler(feature_range=(0,1))

test=scaler_test.fit_transform(test)

train[0:5]
```

```
test[0:5]

def create_features(data,lookback):
    X,Y=[],[]
    for i in range(lookback,len(data)):
        X.append(data[i-lookback:i,0])
        Y.append(data[i,0])
    return np.array(X),np.array(Y)

lookback=20


X_train,y_train=create_features(train,lookback)

X_test,y_test=create_features(test,lookback)

print(X_train.shape,y_train.shape,X_test.shape,y_test.shape)

X_train[0:5]


y_test[0:5]


X_train=np.reshape(X_train,(X_train.shape[0],1,X_train.shape[1]))

X_test=np.reshape(X_test,(X_test.shape[0],1,X_test.shape[1]))

y_train=y_train.reshape(-1,1)

y_test=y_test.reshape(-1,1)

print(X_train.shape,y_train.shape,X_test.shape,y_test.shape)


model=Sequential()
model.add(LSTM(units=50,activation="relu",
               input_shape=(X_train.shape[1],lookback)))
model.add(Dropout(0.2))
model.add(Dense(1))

model.summary()

model.compile(loss="mean_squared_error",optimizer="adam")
```

```
callbacks=[EarlyStopping(monitor="val_loss",patience=3,verbose=1,mode="min"),
           ModelCheckpoint(filepath="mymodel.h5",monitor="val_loss",mode="min",

save_best_only=True,save_weights_only=False,verbose=1)]


history = model.fit(x=X_train,
                    y=y_train,
                    epochs=100,
                    batch_size=20,
                    validation_data=(X_test,y_test),
                    callbacks=callbacks,
                    shuffle=False)


plt.figure(figsize=(20,5))
plt.subplot(1,2,2)
plt.plot(history.history["loss"],label="Training Loss")
plt.plot(history.history["val_loss"],label="Validation Loss")

plt.legend(loc="upper right")
plt.xlabel("Epoch",fontsize=16)
plt.ylabel("Loss",fontsize=16)
plt.ylim([0,max(plt.ylim())])
plt.title("Training and Validation Loss",fontsize=16)
plt.show()


loss=model.evaluate(X_test,y_test,batch_size=20)

print("\nTest loss:%.1f%%"%(100.0*loss))

train_predict=model.predict(X_train)
test_predict=model.predict(X_test)

train_predict=scaler_train.inverse_transform(train_predict)
test_predict=scaler_test.inverse_transform(test_predict)

y_train=scaler_train.inverse_transform(y_train)
y_test=scaler_test.inverse_transform(y_test)


train_rmse=np.sqrt(mean_squared_error(y_train,train_predict))

test_rmse=np.sqrt(mean_squared_error(y_test,test_predict,))

print(f"Train RMSE:{train_rmse}")
print(f"Test RMSE:{test_rmse}")
```

```
train_prediction_data=result_data[lookback:pos]

train_prediction_data["Predicted"]=train_predict

train_prediction_data.head()

test_prediction_data=result_data[pos+lookback:]

test_prediction_data["Predicted"]=test_predict

test_prediction_data.head()

plt.figure(figsize=(14,5))
plt.plot(result_data,label="Real Values")
plt.plot(train_prediction_data["Predicted"],color="blue",label="Train Predicted")
plt.plot(test_prediction_data["Predicted"],color="red",label="Test Predicted")
plt.xlabel("Time")
plt.ylabel("Stock Values")
plt.legend()
plt.show()
```

9.2 Output

9.2.1 Output when the Patience = 3

```
#################### Shape ####################
(3271, 7)
#################### Types ####################
Date          object
Open         float64
High         float64
Low          float64
Close        float64
Adj Close    float64
Volume         int64
dtype: object
#################### Head ####################
         Date    Open    High    Low   Close  Adj Close     Volume
0  2010-01-25  6.1050  6.1140  5.906  6.0155     6.0155  240478000
1  2010-01-26  6.0280  6.1490  5.953  5.9740     5.9740  191180000
2  2010-01-27  6.0515  6.1665  5.940  6.1375     6.1375  295306000
3  2010-01-28  6.2215  6.3600  6.140  6.3015     6.3015  545862000
4  2010-01-29  6.4885  6.5925  6.207  6.2705     6.2705  589426000
#################### Tail ####################
Date          object
Open         float64
High         float64
Low          float64
Close        float64
Adj Close    float64
Volume         int64
dtype: object
#################### NA ####################
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
#################### Quantiles ####################
                   0.00          0.50          0.50          0.95  \
Open       5.296500e+00  3.725000e+01  3.725000e+01  1.658750e+02
High       5.564500e+00  3.742450e+01  3.742450e+01  1.675575e+02
Low        5.290000e+00  3.678050e+01  3.678050e+01  1.642425e+02
Close      5.430500e+00  3.711900e+01  3.711900e+01  1.656255e+02
Adj Close  5.430500e+00  3.711900e+01  3.711900e+01  1.656255e+02
Volume     1.762600e+07  7.439200e+07  7.439200e+07  1.764650e+08
```

```
                  0.99          1.00
Open       1.762695e+02  1.872000e+02
High       1.777882e+02  1.886540e+02
Low        1.743521e+02  1.848395e+02
Close      1.760120e+02  1.865705e+02
Adj Close  1.760120e+02  1.865705e+02
Volume     2.802676e+08  8.484220e+08
Min. Date: 2010-01-25 00:00:00
Max. Date: 2023-01-20 00:00:00
```


```
(2617, 1) (654, 1)
(2597, 20) (2597,) (634, 20) (634,)
(2597, 1, 20) (2597, 1) (634, 1, 20) (634, 1)
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 50)                14200
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
Total params: 14,251
Trainable params: 14,251
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100

  1/130 [..............................] - ETA: 2:47 - loss: 7.4494e-05

 33/130 [======>.......................] - ETA: 0s - loss: 7.7215e-05
```

```
 70/130 [===============>.............] - ETA: 0s - loss: 1.8689e-04

104/130 [========================>......] - ETA: 0s - loss: 8.3013e-04

130/130 [==============================] - 2s 3ms/step - loss: 0.0024 -
val_loss: 0.0099

Epoch 00001: val_loss improved from inf to 0.00986, saving model to mym
odel.h5
Epoch 2/100

  1/130 [..............................] - ETA: 0s - loss: 0.0066

 32/130 [======>.......................] - ETA: 0s - loss: 0.0026

 66/130 [==============>...............] - ETA: 0s - loss: 0.0014

 97/130 [=====================>........] - ETA: 0s - loss: 0.0012

130/130 [==============================] - 0s 2ms/step - loss: 0.0028 -
val_loss: 0.0064
Epoch 00002: val_loss improved from 0.00986 to 0.00640, saving model to
mymodel.h5
Epoch 3/100

  1/130 [..............................] - ETA: 0s - loss: 0.0013

 33/130 [======>.......................] - ETA: 0s - loss: 6.4166e-04

 63/130 [==============>...............] - ETA: 0s - loss: 3.9663e-04

 93/130 [====================>.........] - ETA: 0s - loss: 5.1206e-04

127/130 [============================>.] - ETA: 0s - loss: 0.0019

130/130 [==============================] - 0s 2ms/step - loss: 0.0021 -
val_loss: 0.0058

Epoch 00003: val_loss improved from 0.00640 to 0.00579, saving model to
mymodel.h5
Epoch 4/100

  1/130 [..............................] - ETA: 0s - loss: 4.5045e-04

 29/130 [=====>........................] - ETA: 0s - loss: 1.5374e-04

 64/130 [==============>...............] - ETA: 0s - loss: 1.1555e-04

 98/130 [=====================>........] - ETA: 0s - loss: 3.5209e-04
```

```
130/130 [==============================] - 0s 2ms/step - loss: 0.0021 -
val_loss: 0.0058

Epoch 00004: val_loss improved from 0.00579 to 0.00576, saving model to
mymodel.h5
Epoch 5/100

  1/130 [..............................] - ETA: 0s - loss: 9.4550e-04

 28/130 [=====>........................] - ETA: 0s - loss: 4.6221e-04

 61/130 [==========>...................] - ETA: 0s - loss: 2.6287e-04

 88/130 [===================>..........] - ETA: 0s - loss: 3.1588e-04

120/130 [==========================>...] - ETA: 0s - loss: 0.0015

130/130 [==============================] - 0s 2ms/step - loss: 0.0021 -
val_loss: 0.0057
Epoch 00005: val_loss improved from 0.00576 to 0.00569, saving model to
mymodel.h5
Epoch 6/100

  1/130 [..............................] - ETA: 0s - loss: 8.2083e-04

 38/130 [=======>......................] - ETA: 0s - loss: 2.3291e-04

 64/130 [==========>...................] - ETA: 0s - loss: 1.6940e-04

 96/130 [=====================>........] - ETA: 0s - loss: 4.0562e-04

128/130 [============================>.] - ETA: 0s - loss: 0.0018

130/130 [==============================] - 0s 2ms/step - loss: 0.0020 -
val_loss: 0.0062

Epoch 00006: val_loss did not improve from 0.00569
Epoch 7/100

  1/130 [..............................] - ETA: 0s - loss: 0.0034

 31/130 [======>.......................] - ETA: 0s - loss: 0.0012

 63/130 [==========>...................] - ETA: 0s - loss: 6.4023e-04

 98/130 [=====================>........] - ETA: 0s - loss: 6.8257e-04

130/130 [==============================] - 0s 2ms/step - loss: 0.0020 -
val_loss: 0.0060
```

```
Epoch 00007: val_loss did not improve from 0.00569
Epoch 8/100

  1/130 [..............................] - ETA: 0s - loss: 0.0016

 35/130 [=======>......................] - ETA: 0s - loss: 7.4911e-04

 68/130 [==============>...............] - ETA: 0s - loss: 4.8284e-04

 99/130 [=====================>........] - ETA: 0s - loss: 5.8934e-04

130/130 [==============================] - 0s 2ms/step - loss: 0.0019 -
val_loss: 0.0053
Epoch 00008: val_loss improved from 0.00569 to 0.00529, saving model to
mymodel.h5
Epoch 9/100

  1/130 [..............................] - ETA: 0s - loss: 4.5693e-04

 32/130 [======>.......................] - ETA: 0s - loss: 2.9284e-04

 62/130 [=============>................] - ETA: 0s - loss: 2.1564e-04

 93/130 [====================>.........] - ETA: 0s - loss: 3.1501e-04

125/130 [===========================>..] - ETA: 0s - loss: 0.0015

130/130 [==============================] - 0s 2ms/step - loss: 0.0018 -
val_loss: 0.0052

Epoch 00009: val_loss improved from 0.00529 to 0.00518, saving model to
mymodel.h5
Epoch 10/100

  1/130 [..............................] - ETA: 0s - loss: 0.0016

 31/130 [======>.......................] - ETA: 0s - loss: 5.3811e-04

 62/130 [=============>................] - ETA: 0s - loss: 3.4410e-04

 97/130 [=====================>........] - ETA: 0s - loss: 4.4179e-04

128/130 [============================>.] - ETA: 0s - loss: 0.0018

130/130 [==============================] - 0s 2ms/step - loss: 0.0018 -
val_loss: 0.0057

Epoch 00010: val_loss did not improve from 0.00518
Epoch 11/100
```

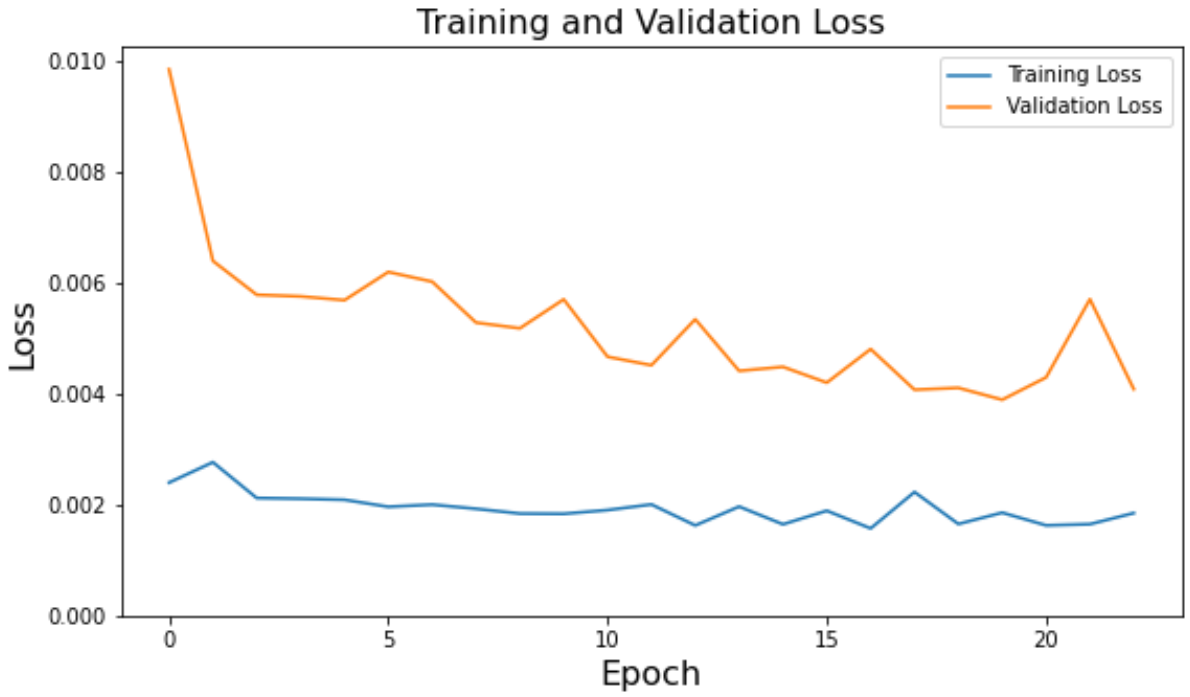
```
  1/130 [..............................] - ETA: 0s - loss: 0.0011

 32/130 [======>.......................] - ETA: 0s - loss: 6.8167e-04

 65/130 [==============>...............] - ETA: 0s - loss: 4.8983e-04

 98/130 [=====================>........] - ETA: 0s - loss: 5.7932e-04

129/130 [============================>.] - ETA: 0s - loss: 0.0018

130/130 [==============================] - 0s 2ms/step - loss: 0.0019 -
val_loss: 0.0047

Epoch 00011: val_loss improved from 0.00518 to 0.00467, saving model to
mymodel.h5
Epoch 12/100

  1/130 [..............................] - ETA: 0s - loss: 6.3374e-04

 37/130 [=======>......................] - ETA: 0s - loss: 3.0577e-04

 67/130 [==============>...............] - ETA: 0s - loss: 2.7886e-04

 97/130 [=====================>........] - ETA: 0s - loss: 4.2822e-04

127/130 [============================>.] - ETA: 0s - loss: 0.0017

130/130 [==============================] - 0s 2ms/step - loss: 0.0020 -
val_loss: 0.0045

Epoch 00012: val_loss improved from 0.00467 to 0.00452, saving model to
mymodel.h5
Epoch 13/100

  1/130 [..............................] - ETA: 0s - loss: 5.4120e-04

 37/130 [=======>......................] - ETA: 0s - loss: 3.4451e-04

 72/130 [===============>..............] - ETA: 0s - loss: 2.3971e-04

105/130 [=======================>......] - ETA: 0s - loss: 5.5055e-04

130/130 [==============================] - 0s 2ms/step - loss: 0.0016 -
val_loss: 0.0053

Epoch 00013: val_loss did not improve from 0.00452
Epoch 14/100

  1/130 [..............................] - ETA: 0s - loss: 0.0027
```

```
 34/130 [======>.....................] - ETA: 0s - loss: 7.5712e-04

 72/130 [===============>..............] - ETA: 0s - loss: 4.7981e-04

104/130 [========================>......] - ETA: 0s - loss: 7.4685e-04

130/130 [==============================] - 0s 2ms/step - loss: 0.0020 -
val_loss: 0.0044

Epoch 00014: val_loss improved from 0.00452 to 0.00442, saving model to
mymodel.h5
Epoch 15/100

  1/130 [..............................] - ETA: 0s - loss: 5.0004e-04

 35/130 [=======>......................] - ETA: 0s - loss: 3.7779e-04

 64/130 [=============>................] - ETA: 0s - loss: 2.6719e-04

 96/130 [=====================>........] - ETA: 0s - loss: 3.5713e-04

130/130 [==============================] - ETA: 0s - loss: 0.0016

130/130 [==============================] - 0s 2ms/step - loss: 0.0016 -
val_loss: 0.0045
Epoch 00015: val_loss did not improve from 0.00442
Epoch 16/100

  1/130 [..............................] - ETA: 0s - loss: 0.0012

 33/130 [======>.....................] - ETA: 0s - loss: 5.7299e-04

 66/130 [==============>...............] - ETA: 0s - loss: 3.6308e-04

 99/130 [=====================>........] - ETA: 0s - loss: 5.0844e-04

130/130 [==============================] - 0s 2ms/step - loss: 0.0019 -
val_loss: 0.0042

Epoch 00016: val_loss improved from 0.00442 to 0.00420, saving model to
mymodel.h5
Epoch 17/100

  1/130 [..............................] - ETA: 0s - loss: 0.

 32/130 [======>.....................] - ETA: 0s - loss: 4.5179e-04

 64/130 [=============>................] - ETA: 0s - loss: 2.8356e-04

 96/130 [=====================>........] - ETA: 0s - loss: 3.6850e-04
```

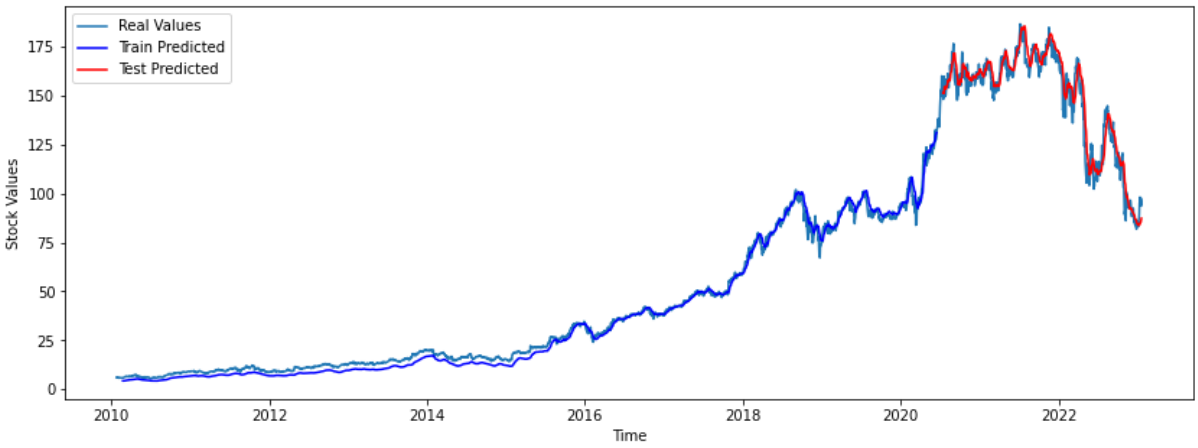
```
126/130 [============================>.] - ETA: 0s - loss: 0.0013

130/130 [==============================] - 0s 2ms/step - loss: 0.0016 -
val_loss: 0.0048

Epoch 00017: val_loss did not improve from 0.00420
Epoch 18/100

  1/130 [..............................] - ETA: 0s - loss: 0.0032

 31/130 [======>.......................] - ETA: 0s - loss: 0.0012

 65/130 [==============>...............] - ETA: 0s - loss: 6.9607e-04

 97/130 [=====================>........] - ETA: 0s - loss: 7.0463e-04

128/130 [============================>.] - ETA: 0s - loss: 0.0021

130/130 [==============================] - 0s 2ms/step - loss: 0.0022 -
val_loss: 0.0041
Epoch 00018: val_loss improved from 0.00420 to 0.00408, saving model to
mymodel.h5
Epoch 19/100

  1/130 [..............................] - ETA: 0s - loss: 6.0939e-04

 31/130 [======>.......................] - ETA: 0s - loss: 4.4646e-04

 64/130 [=============>................] - ETA: 0s - loss: 2.7239e-04

 94/130 [====================>.........] - ETA: 0s - loss: 3.2406e-04

126/130 [============================>.] - ETA: 0s - loss: 0.0014

130/130 [==============================] - 0s 2ms/step - loss: 0.0017 -
val_loss: 0.0041

Epoch 00019: val_loss did not improve from 0.00408
Epoch 20/100

  1/130 [..............................] - ETA: 0s - loss: 0.0021

 32/130 [======>.......................] - ETA: 0s - loss: 7.1958e-04

 64/130 [=============>................] - ETA: 0s - loss: 4.2508e-04

 95/130 [====================>.........] - ETA: 0s - loss: 4.4605e-04

127/130 [============================>.] - ETA: 0s - loss: 0.0017
```

```
130/130 [==============================] - 0s 2ms/step - loss: 0.0019 -
val_loss: 0.0039

Epoch 00020: val_loss improved from 0.00408 to 0.00390, saving model to
mymodel.h5
Epoch 21/100

  1/130 [..............................] - ETA: 0s - loss: 0.0011

 37/130 [=======>......................] - ETA: 0s - loss: 5.5076e-04

 70/130 [===============>..............] - ETA: 0s - loss: 3.4306e-04

103/130 [======================>.......] - ETA: 0s - loss: 5.7997e-04

130/130 [==============================] - 0s 2ms/step - loss: 0.0016 -
val_loss: 0.0043

Epoch 00021: val_loss did not improve from 0.00390
Epoch 22/100

  1/130 [..............................] - ETA: 0s - loss: 0.0028

 31/130 [======>.......................] - ETA: 0s - loss: 9.4610e-04

 60/130 [============>.................] - ETA: 0s - loss: 5.7324e-04

 92/130 [====================>.........] - ETA: 0s - loss: 5.0415e-04

123/130 [===========================>..] - ETA: 0s - loss: 0.0013

130/130 [==============================] - 0s 2ms/step - loss: 0.0016 -
val_loss: 0.0057
Epoch 00022: val_loss did not improve from 0.00390
Epoch 23/100

  1/130 [..............................] - ETA: 0s - loss: 8.6887e-04

 34/130 [======>.......................] - ETA: 0s - loss: 6.4272e-04

 67/130 [==============>...............] - ETA: 0s - loss: 4.7029e-04

 98/130 [=====================>........] - ETA: 0s - loss: 5.7187e-04

128/130 [============================>.] - ETA: 0s - loss: 0.0017

130/130 [==============================] - 0s 2ms/step - loss: 0.0018 -
val_loss: 0.0041
```

```
Epoch 00023: val_loss did not improve from 0.00390
Epoch 00023: early stopping
```




```
 1/32 [..............................] - ETA: 0s - loss: 0.0012

32/32 [==============================] - 0s 903us/step - loss: 0.0041

Test loss:0.4%
Train RMSE:2.7260358333587646
Test RMSE:6.695877552032471
```



9.2.2 Output when the Patience = 4

```
#################### Shape ####################
(3271, 7)
#################### Types ####################
Date          object
Open         float64
High         float64
Low          float64
Close        float64
Adj Close    float64
Volume         int64
dtype: object
#################### Head ####################
         Date    Open    High    Low   Close  Adj Close     Volume
0  2010-01-25  6.1050  6.1140  5.906  6.0155     6.0155  240478000
1  2010-01-26  6.0280  6.1490  5.953  5.9740     5.9740  191180000
2  2010-01-27  6.0515  6.1665  5.940  6.1375     6.1375  295306000
3  2010-01-28  6.2215  6.3600  6.140  6.3015     6.3015  545862000
4  2010-01-29  6.4885  6.5925  6.207  6.2705     6.2705  589426000
#################### Tail ####################
Date          object
Open         float64
High         float64
Low          float64
Close        float64
Adj Close    float64
Volume         int64
dtype: object
#################### NA ####################
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
#################### Quantiles ####################
                   0.00          0.50          0.50          0.95  \
Open       5.296500e+00  3.725000e+01  3.725000e+01  1.658750e+02
High       5.564500e+00  3.742450e+01  3.742450e+01  1.675575e+02
Low        5.290000e+00  3.678050e+01  3.678050e+01  1.642425e+02
Close      5.430500e+00  3.711900e+01  3.711900e+01  1.656255e+02
Adj Close  5.430500e+00  3.711900e+01  3.711900e+01  1.656255e+02
Volume     1.762600e+07  7.439200e+07  7.439200e+07  1.764650e+08

                   0.99          1.00
Open       1.762695e+02  1.872000e+02
High       1.777882e+02  1.886540e+02
```

```
Low         1.743521e+02  1.848395e+02
Close       1.760120e+02  1.865705e+02
Adj Close   1.760120e+02  1.865705e+02
Volume      2.802676e+08  8.484220e+08
Min. Date: 2010-01-25 00:00:00
Max. Date: 2023-01-20 00:00:00
```


```
(2617, 1) (654, 1)
(2597, 20) (2597,) (634, 20) (634,)
(2597, 1, 20) (2597, 1) (634, 1, 20) (634, 1)
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 50)                14200
_________________________________________________________________
dropout_4 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 51
=================================================================
Total params: 14,251
Trainable params: 14,251
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
130/130 [==============================] - 2s 5ms/step - loss: 0.0028 -
val_loss: 0.0100

Epoch 00001: val_loss improved from inf to 0.00995, saving model to mym
odel.h5
Epoch 2/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0030 -
val_loss: 0.0074
```

```
Epoch 00002: val_loss improved from 0.00995 to 0.00744, saving model to
mymodel.h5
Epoch 3/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0029 -
val_loss: 0.0066

Epoch 00003: val_loss improved from 0.00744 to 0.00663, saving model to
mymodel.h5
Epoch 4/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0030 -
val_loss: 0.0062

Epoch 00004: val_loss improved from 0.00663 to 0.00618, saving model to
mymodel.h5
Epoch 5/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0024 -
val_loss: 0.0063

Epoch 00005: val_loss did not improve from 0.00618
Epoch 6/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0024 -
val_loss: 0.0075

Epoch 00006: val_loss did not improve from 0.00618
Epoch 7/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0024 -
val_loss: 0.0061

Epoch 00007: val_loss improved from 0.00618 to 0.00609, saving model to
mymodel.h5
Epoch 8/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0025 -
val_loss: 0.0057

Epoch 00008: val_loss improved from 0.00609 to 0.00573, saving model to
mymodel.h5
Epoch 9/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0021 -
val_loss: 0.0059

Epoch 00009: val_loss did not improve from 0.00573
Epoch 10/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0022 -
val_loss: 0.0063

Epoch 00010: val_loss did not improve from 0.00573
Epoch 11/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0022 -
val_loss: 0.0053
```

```
Epoch 00011: val_loss improved from 0.00573 to 0.00533, saving model to
mymodel.h5
Epoch 12/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0024 -
val_loss: 0.0056

Epoch 00012: val_loss did not improve from 0.00533
Epoch 13/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0022 -
val_loss: 0.0052

Epoch 00013: val_loss improved from 0.00533 to 0.00520, saving model to
mymodel.h5
Epoch 14/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0022 -
val_loss: 0.0054

Epoch 00014: val_loss did not improve from 0.00520
Epoch 15/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0023 -
val_loss: 0.0049

Epoch 00015: val_loss improved from 0.00520 to 0.00488, saving model to
mymodel.h5
Epoch 16/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0020 -
val_loss: 0.0054

Epoch 00016: val_loss did not improve from 0.00488
Epoch 17/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0023 -
val_loss: 0.0048

Epoch 00017: val_loss improved from 0.00488 to 0.00483, saving model to
mymodel.h5
Epoch 18/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0021 -
val_loss: 0.0054

Epoch 00018: val_loss did not improve from 0.00483
Epoch 19/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0023 -
val_loss: 0.0058

Epoch 00019: val_loss did not improve from 0.00483
Epoch 20/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0020 -
val_loss: 0.0053
```

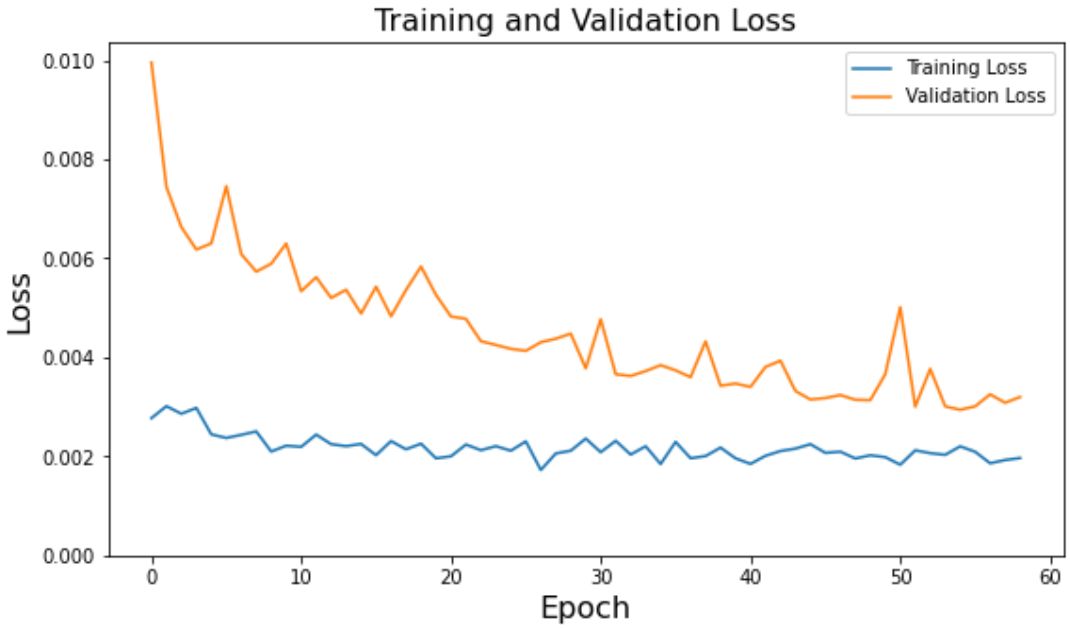
```
Epoch 00020: val_loss did not improve from 0.00483
Epoch 21/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0020 -
val_loss: 0.0048

Epoch 00021: val_loss improved from 0.00483 to 0.00482, saving model to
mymodel.h5
Epoch 22/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0022 -
val_loss: 0.0048

Epoch 00022: val_loss improved from 0.00482 to 0.00478, saving model to
mymodel.h5
Epoch 23/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0021 -
val_loss: 0.0043

Epoch 00023: val_loss improved from 0.00478 to 0.00432, saving model to
mymodel.h5
Epoch 24/100
130/130 [==============================] - 0s 2ms/step - loss: 0.0022 -
val_loss: 0.0043

Epoch 00024: val_loss improved from 0.00432 to 0.00425, saving model to
mymodel.h5
Epoch 25/100
130/130 [==============================] - 0s 2ms/step - loss: 0.0021 -
val_loss: 0.0042

Epoch 00025: val_loss improved from 0.00425 to 0.00417, saving model to
mymodel.h5
Epoch 26/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0023 -
val_loss: 0.0041

Epoch 00026: val_loss improved from 0.00417 to 0.00413, saving model to
mymodel.h5
Epoch 27/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0017 -
val_loss: 0.0043

Epoch 00027: val_loss did not improve from 0.00413
Epoch 28/100
130/130 [==============================] - 0s 2ms/step - loss: 0.0021 -
val_loss: 0.0044

Epoch 00028: val_loss did not improve from 0.00413
Epoch 29/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0021 -
val_loss: 0.0045
```

```
Epoch 00029: val_loss did not improve from 0.00413
Epoch 30/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0024 -
val_loss: 0.0038

Epoch 00030: val_loss improved from 0.00413 to 0.00378, saving model to
mymodel.h5
Epoch 31/100
130/130 [==============================] - 0s 2ms/step - loss: 0.0021 -
val_loss: 0.0048

Epoch 00031: val_loss did not improve from 0.00378
Epoch 32/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0023 -
val_loss: 0.0037

Epoch 00032: val_loss improved from 0.00378 to 0.00366, saving model to
mymodel.h5
Epoch 33/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0020 -
val_loss: 0.0036

Epoch 00033: val_loss improved from 0.00366 to 0.00362, saving model to
mymodel.h5
Epoch 34/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0022 -
val_loss: 0.0037

Epoch 00034: val_loss did not improve from 0.00362
Epoch 35/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0018 -
val_loss: 0.0038

Epoch 00035: val_loss did not improve from 0.00362
Epoch 36/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0023 -
val_loss: 0.0037

Epoch 00036: val_loss did not improve from 0.00362
Epoch 37/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0020 -
val_loss: 0.0036

Epoch 00037: val_loss improved from 0.00362 to 0.00360, saving model to
mymodel.h5
Epoch 38/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0020 -
val_loss: 0.0043
```

```
Epoch 00038: val_loss did not improve from 0.00360
Epoch 39/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0022 -
val_loss: 0.0034

Epoch 00039: val_loss improved from 0.00360 to 0.00343, saving model to
mymodel.h5
Epoch 40/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0020 -
val_loss: 0.0035

Epoch 00040: val_loss did not improve from 0.00343
Epoch 41/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0018 -
val_loss: 0.0034

Epoch 00041: val_loss improved from 0.00343 to 0.00340, saving model to
mymodel.h5
Epoch 42/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0020 -
val_loss: 0.0038

Epoch 00042: val_loss did not improve from 0.00340
Epoch 43/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0021 -
val_loss: 0.0039

Epoch 00043: val_loss did not improve from 0.00340
Epoch 44/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0022 -
val_loss: 0.0033

Epoch 00044: val_loss improved from 0.00340 to 0.00332, saving model to
mymodel.h5
Epoch 45/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0022 -
val_loss: 0.0031

Epoch 00045: val_loss improved from 0.00332 to 0.00315, saving model to
mymodel.h5
Epoch 46/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0021 -
val_loss: 0.0032

Epoch 00046: val_loss did not improve from 0.00315
Epoch 47/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0021 -
val_loss: 0.0032

Epoch 00047: val_loss did not improve from 0.00315
```

```
Epoch 48/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0020 -
val_loss: 0.0031

Epoch 00048: val_loss improved from 0.00315 to 0.00314, saving model to
mymodel.h5
Epoch 49/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0020 -
val_loss: 0.0031

Epoch 00049: val_loss improved from 0.00314 to 0.00314, saving model to
mymodel.h5
Epoch 50/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0020 -
val_loss: 0.0037

Epoch 00050: val_loss did not improve from 0.00314
Epoch 51/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0018 -
val_loss: 0.0050

Epoch 00051: val_loss did not improve from 0.00314
Epoch 52/100
130/130 [==============================] - 0s 2ms/step - loss: 0.0021 -
val_loss: 0.0030

Epoch 00052: val_loss improved from 0.00314 to 0.00300, saving model to
mymodel.h5
Epoch 53/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0021 -
val_loss: 0.0038

Epoch 00053: val_loss did not improve from 0.00300
Epoch 54/100
130/130 [==============================] - 0s 2ms/step - loss: 0.0020 -
val_loss: 0.0030

Epoch 00054: val_loss did not improve from 0.00300
Epoch 55/100
130/130 [==============================] - 0s 2ms/step - loss: 0.0022 -
val_loss: 0.0029

Epoch 00055: val_loss improved from 0.00300 to 0.00294, saving model to
mymodel.h5
Epoch 56/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0021 -
val_loss: 0.0030

Epoch 00056: val_loss did not improve from 0.00294
Epoch 57/100
```

```
130/130 [==============================] - 0s 3ms/step - loss: 0.0019 -
val_loss: 0.0033

Epoch 00057: val_loss did not improve from 0.00294
Epoch 58/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0019 -
val_loss: 0.0031

Epoch 00058: val_loss did not improve from 0.00294
Epoch 59/100
130/130 [==============================] - 0s 3ms/step - loss: 0.0020 -
val_loss: 0.0032

Epoch 00059: val_loss did not improve from 0.00294
Epoch 00059: early stopping
```




```
32/32 [==============================] - 0s 1ms/step - loss: 0.0032

Test loss:0.3%
Train RMSE:2.995021343231201
Test RMSE:5.92483377456665
```