A CNN-LSTM Based Model for Stock Price Forecasting

By: Danny Hong

ECE-411: Quantitative Finance

Project Goal

The purpose of this project is to attempt to recreate the model proposed in the article *A CNN-LSTM-Based Model to Forecast Stock Prices*.

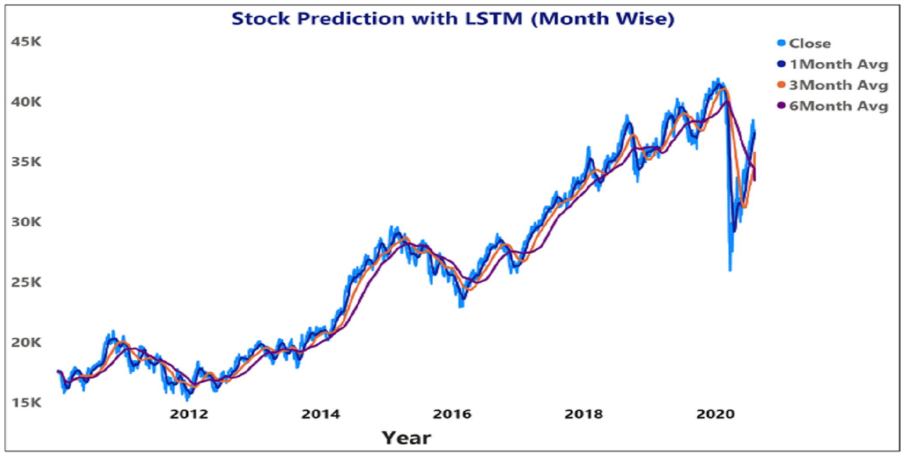
The paper introduces a new (at the time) deep learning method that can be used to predict future stock prices.

Machine/Deep Learning in Stock Price Forecasting

ML/DL techniques are being used more in predicting future stock prices because it can:

1. Handle vast amounts of data
2. Understand complex patterns and make more accurate predictions.
3. Adapt and learn from evolving trends from new data

```
Epoch 1/100
29/29 [==============================] - 3s 25ms/step - loss: 0.0232 - val_loss: 0.0108
Epoch 2/100
29/29 [==============================] - 0s 6ms/step - loss: 0.0152 - val_loss: 0.0092
Epoch 3/100
29/29 [==============================] - 0s 6ms/step - loss: 0.0134 - val_loss: 0.0082
Epoch 4/100
29/29 [==============================] - 0s 6ms/step - loss: 0.0126 - val_loss: 0.0080
Epoch 5/100
29/29 [==============================] - 0s 6ms/step - loss: 0.0124 - val_loss: 0.0081
Epoch 6/100
29/29 [==============================] - 0s 6ms/step - loss: 0.0123 - val_loss: 0.0079
Epoch 7/100
29/29 [==============================] - 0s 6ms/step - loss: 0.0123 - val_loss: 0.0078
Epoch 8/100
29/29 [==============================] - 0s 6ms/step - loss: 0.0123 - val_loss: 0.0078
Epoch 9/100
29/29 [==============================] - 0s 5ms/step - loss: 0.0123 - val_loss: 0.0077
Epoch 10/100
29/29 [==============================] - 0s 6ms/step - loss: 0.0123 - val_loss: 0.0078
Epoch 11/100
29/29 [==============================] - 0s 6ms/step - loss: 0.0124 - val_loss: 0.0077
Epoch 12/100
29/29 [==============================] - 0s 6ms/step - loss: 0.0123 - val_loss: 0.0076
```



Model

The CNN-LSTM model is an example of a hybrid ML model.

From the name, the model consists of two parts: the CNN (Convolutional Neural Network) and the LSTM (Long Short Term Memory).

The CNN part is used for extracting the time features (“spatial” characteristics of the graph) of data, while the LSTM part is used for data forecasting.

CNN (Convolutional Neural Network)

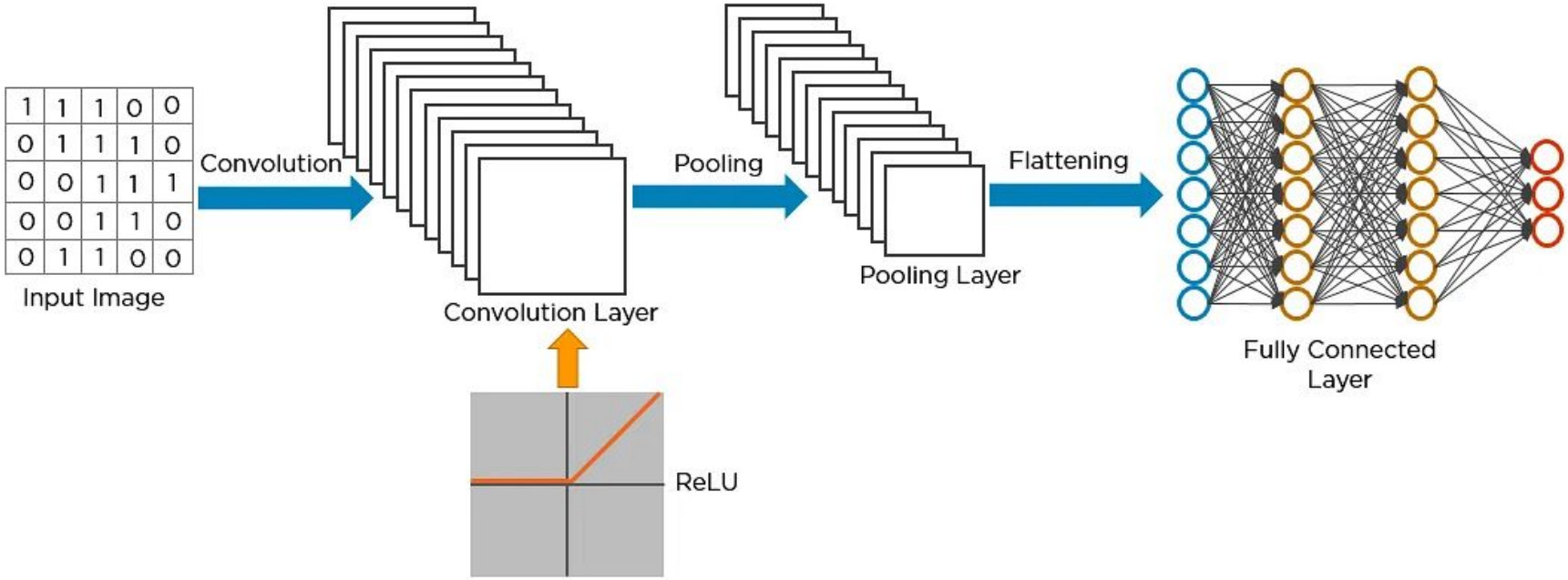
CNN is a type of deep learning model that is commonly used for image and video processing tasks.

It can be thought of as a network of interconnected nodes (neurons) that are designed to automatically recognize patterns in visual data.

The key idea behind a CNN is its ability to learn hierarchical representations of images by using layers of specialized filters, called convolutional layers, that scan an image to identify different features.

Each filter detects specific patterns, such as edges, corners, or textures, at various levels of abstraction.

CNN Steps


1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 0
0 1 1 0 0
Input Image
Convolution
Convolution Layer
ReLU
Pooling
Pooling Layer
Flattening
Fully Connected Layer

CNN for Stock Time Series

The convolutional layers in a CNN can be used to automatically extract meaningful temporal patterns and relationships from stock time series data.

The filters in the convolutional layers scan the input matrix, identifying relevant patterns or features that are important for predicting future values.

These filters can capture short-term patterns, such as local trends or specific patterns that repeat over time.

LSTM (Long Short Term Memory)

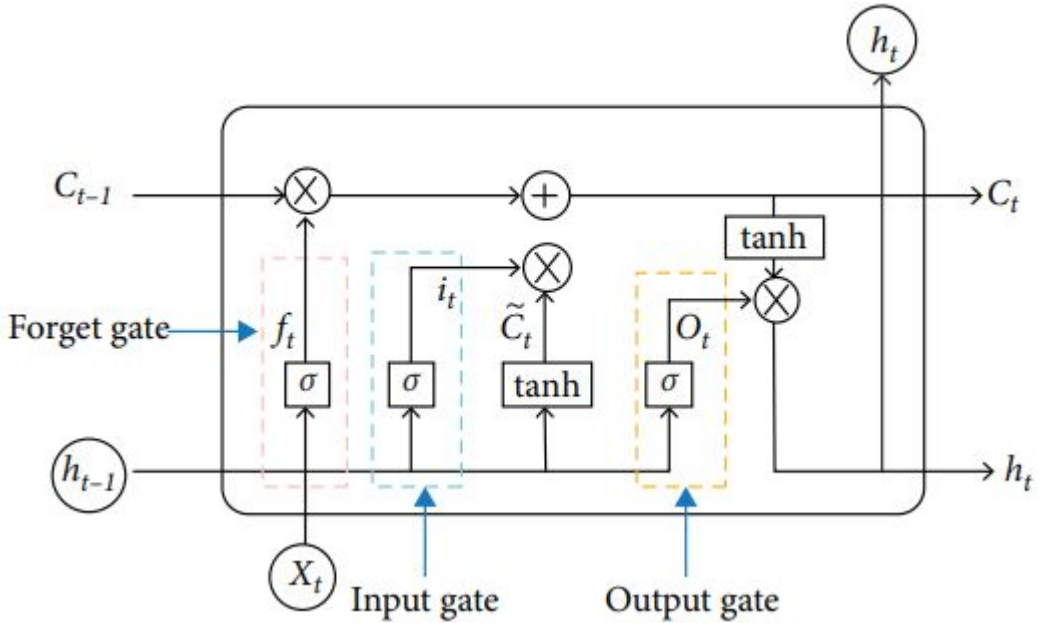
LSTM (Long Short-Term Memory) is a type of recurrent neural network (RNN) that is designed to capture long-term dependencies and patterns in sequential data.

In simple terms, it is a specialized type of neural network that can remember and forget information over long periods of time.

By incorporating memory cells and gating mechanisms, LSTMs are capable of selectively remembering and forgetting information over long sequences.

This makes LSTMs well-suited for tasks involving sequential data, such as natural language processing, speech recognition, and stock time series forecasting.

LSTM Steps


h_t
C_{t-1}
C_t
tanh
i_t
Forget gate
f_t
$\tilde{C}_t$
O_t
σ
σ
tanh
σ
h_{t-1}
h_t
X_t
Input gate
Output gate

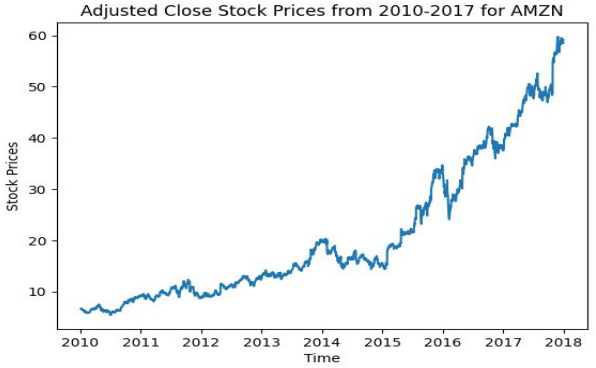
Model Implementation

Many of model specifications of this implementation remained identical to what was defined by the authors in the paper. However, there are a few other differences between this procedure and the procedure done by the authors.

1. The dataset used is different.
2. There is no data normalization for this implementation.
3. Data preprocessing step is different, which is due to the fact that the authors in their paper used a number of features to train the model (Multi-Dimensional) whereas this implementation will only use 1 feature (1-Dimensional).

Dataset

Using Yahoo Finance, stock data for Apple, Amazon, JP Morgan & Chase, and Goldman Sachs for the years 2010-2017 are obtained, and the Adjusted Closing Prices for those stocks during that time period are shown below.



Data Preparation/Preprocessing

The code prepares the data by creating input-output pairs from the historical stock prices, where the input consists of a window (timestep value) of previous prices and the output is the next price to be predicted.

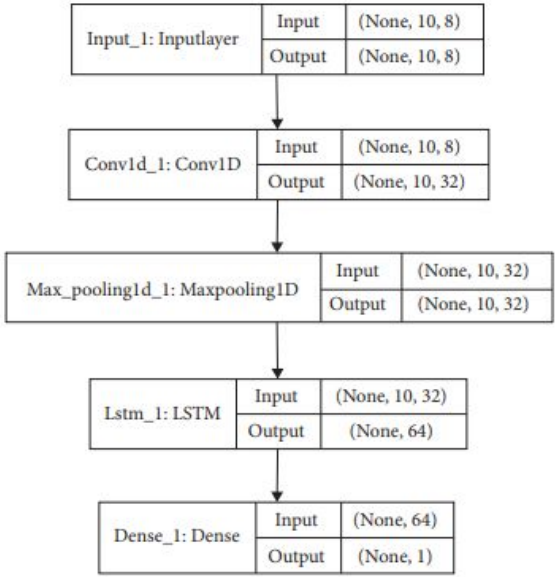
The code analyzes the relative change in prices by calculating the percentage change of each stock price within a window relative to the first price in that window.

The data is then split into training and testing sets (90% train/10% test) and reshaped to be inputted for model training and evaluation.

Model Setup

Parameters	Value
Convolution layer filters	32
Convolution layer kernel_size	1
Convolution layer activation function	tanh
Convolution layer padding	Same
Pooling layer pool_size	1
Pooling layer padding	Same
Pooling layer activation function	Relu
Number of hidden units in LSTM layer	64
LSTM layer activation function	tanh
Time_step	10
Batch_size	64
Learning rate	0.001
Optimizer	Adam
Loss function	mean_absolute_error
Epochs	100

Model Parameter Settings



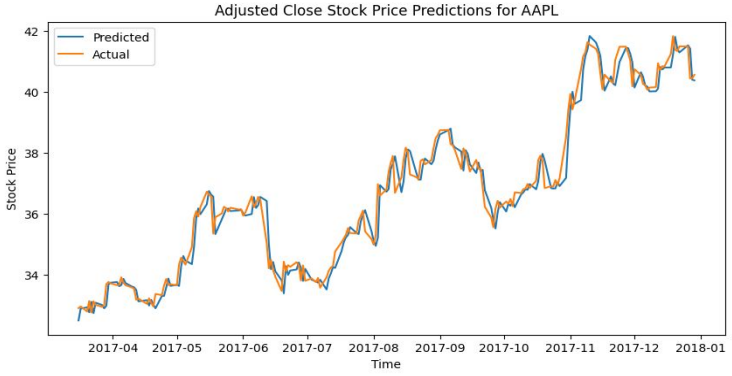
Model Structure

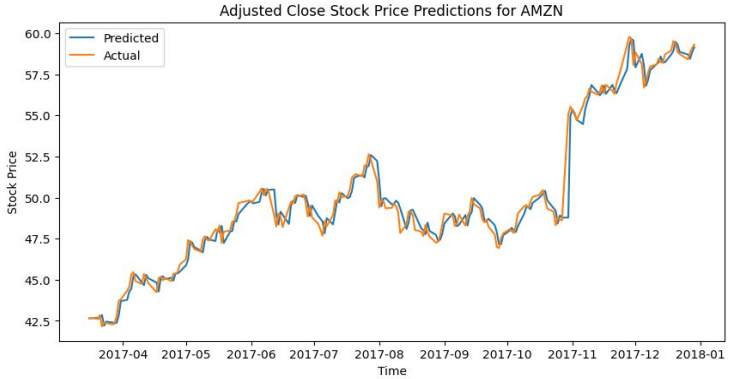
Model Code

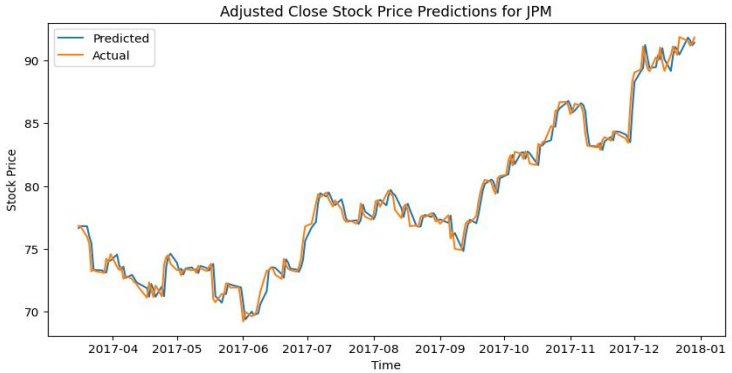
```
model = Sequential()

model.add(TimeDistributed(Conv1D(filters = 32, kernel_size = 1, activation = 'tanh', padding = 'same', input_shape = (None, timesteps, 1))))
model.add(TimeDistributed(MaxPooling1D(pool_size = 1, padding = 'same')))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(64, activation = 'tanh', return_sequences = True))
model.add(Dense(1, activation = 'tanh'))

adam = Adam(learning_rate = 0.001)
model.compile(optimizer = adam, loss = 'mean_absolute_error')

model.fit(train_X, train_label, validation_data = (test_X, test_label), epochs = 100, batch_size = 64, shuffle = False, verbose = 1)
```

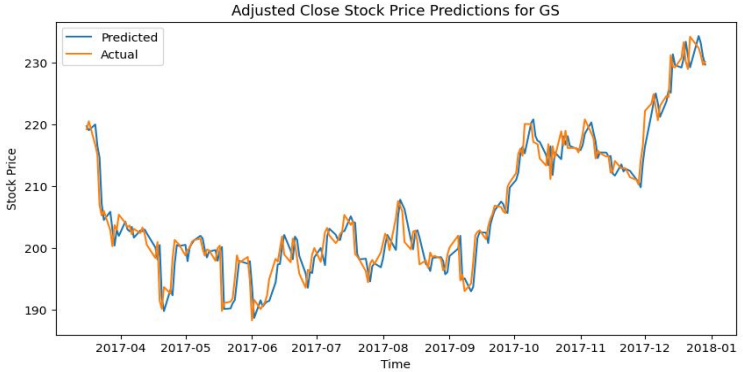
Prediction Results

Since the last 10% of the data were used for testing, the following plots shows the predicted v.s. the actual results from April 2017 up to the end of 2017.









Test Losses For Each Stock

CNN-LSTM Model Test Loss Scores From Training:

	Stocks	Test Loss Values
0	AAPL	0.008363
1	AMZN	0.009011
2	JPM	0.007532
3	GS	0.009409

Conclusion/Possible Improvements

The model wasn’t very complex to create, and for what it’s worth, generated mostly accurate predictions for the given input data.

Given more time, having a bigger dataset to feed into the model would be helpful.

It will also be interesting to see how the model will perform for other years.

However, one concern is that there is possible data leakage during the train/test data preparation

Therefore, if leakage does exist, then a better way on preparing the training/testing data for time series should be explored.