Stock price prediction using CNN-LSTM model

Project Group 04

Deepak Singh - MA23M006
Hareesh P Nair - AE20B027
Mousina Barman - MA23M011
Snehal - MA23M020
Sonu Kumari - MA23M021

Dept. of Mathematics
Indian Institute of Technology Madras

June 2, 2024

Investing in Stocks

Wealth Management

: a holistic service that focus on optimizing the wealth of affluent clients.

Stocks

: a share in ownership of a company.

Stock Market

: Stock markets are venues where buyers and sellers meet to exchange equity shares of public corporations.

Investing in Stocks

Low-cost and low minimums to get started

Typically follows indexing strategies, best-suited for most long-term investors

Automation eliminates human error and can continuously monitor portfolios

Expanding set of choices, such as ESG-focused portfolios

Why predict stocks ?

Minimize the risks of losing money

Better investing and trading decisions

Diversify portfolio

Performance Evaluation

Statistical Method

Time Series Analysis

a statistical technique used to analyze time-ordered data to identify patterns, trends, and forecast future values.

Regression

Making predictions using linear equations on historical data on one or more indicators.

Volatility Models

a technique to analyze volatality or variance of data.Models like ARCH and GARCH along with an estimation of conditional variance dynamics provide insights into the volatility of stocks

Machine Learning Methods

Random Forest

Aggregate predictions from multiple decision trees, making them capable of capturing complex patterns in stock data.

Support Vector Machines

Classify stocks by finding the best separating hyperplane, leveraging kernel functions to handle nonlinear relationships.

Deep Neural Networks

Use CNN and LSTM to learn hierarchical representations of stock data, capturing temporal dependencies or spatial patterns for improved prediction performance.

Problem statement

A hybrid CNN – LSTM model for stock price prediction

Comparison of different methods

Factors	Statistical Method	Classical ML methods	Hybrid CNN - LSTM Model
Probability based prediction	Yes	Yes	Yes
Directional trend prediction	Yes	Yes	Yes
Independent of assumptions	No	Yes	Yes
Capture non-linear relationships	No	No	Yes
Predict long sequence accurately	No	No	Yes
Capture effects of external factors	No	No	Yes
Avoid Overfitting	No	Yes	Yes

Why a hybrid CNN-LSTM model ?

Capture long-term dependencies

Handling non-linear relationships

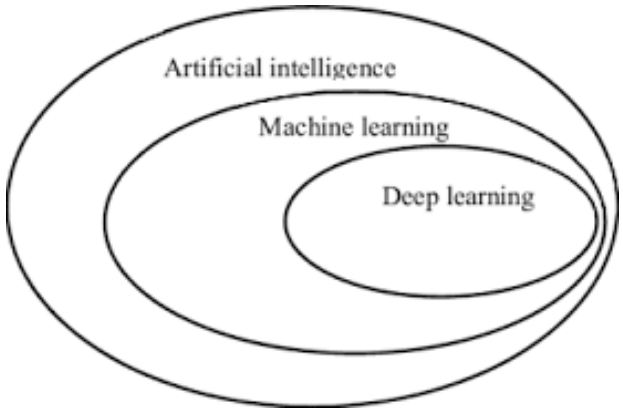
Feature learning and representations

Temporal dynamics and memory

Adaptability and generalization

Deep Learning Models

Deep Learning is a a technique within machine learning which is used to identify complex representations by training deep neural networks.

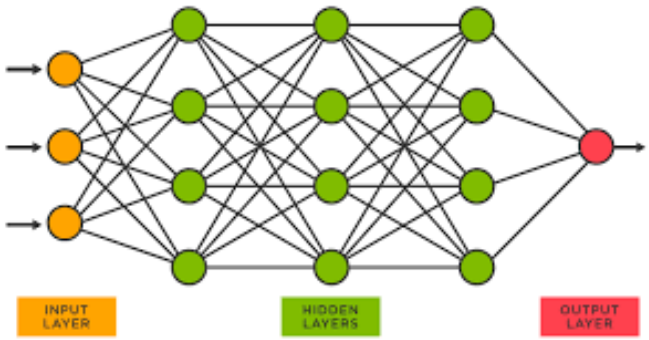


It was initially introduced as deep feedforward multilayered perceptrons by Alexey Ivakhnenko and Lapa in 1967.
The term **deep learning** was proposed in 1986 by Rina Dechter.

How do Neural Networks work ?

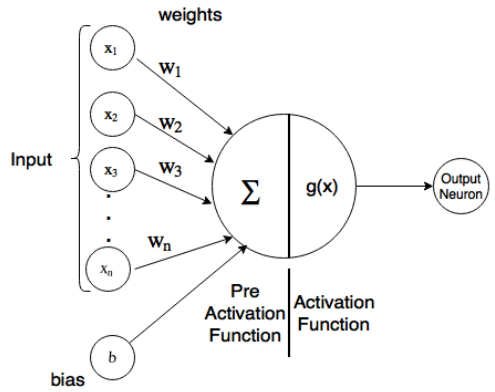
Neural networks are powerful algorithms that work similarly to neurons in the human brain. They rely on input data for improvement in accuracy.
A neural network consists of three parts

- Input layer
- Hidden layers
- Output layer



How do Neural Networks work? (contd.)

Each neuron has two parts: Pre-activation and Activation



An Artificial Neuron: Basic unit of Neural Networks

Pre-activation at i^{th} level is given by:

$$a_i(x) = b_i + w_i h_{i-1}(x)$$

The activation level is given by:

$$h_i(x) = g(a_i(x))$$

where $a_i, h_i, b_i, x \in \mathbb{R}^N, w_i \in \mathbb{R}^{N \times N}$ and g is a vector $g : \mathbb{R}^N \to \mathbb{R}^N$, N is the number of inputs to a neuron i^{th} level.

How do Neural Networks work? (contd.)

g is called the activation function, and the popular choices are:

Linear function : $g(x) = x$

ReLU : $g(x) = \max\{0, x\}$

Sigmoid function : $g(x) = \frac{1}{1+e^{-x}}$

The neural network model can be summarized as :

Data : $\{x_i, y_i\}_{i=1}^{N}$

Model : $\hat{y}_i = f(x_i) = O(W^k g(W^{k-1} g(.....(W^2(g(W^1 x + b_1) + b_2) + ...) + b_{k-1}) + b_k))$ for k+1 layered neural network.

The objective is to find the weight matrices W^i and bias terms b_i, which can be done by selecting an appropriate loss function and then implementing a forward and backward propagation.

Convolutional Neural Network

Convolutional Neural Network (CNN) is a type of powerful neural network that is used in a variety of industrial applications like text, image, audio recognition, and classification.

It was developed by Yann LeCun, a post-doctoral computer science researcher during the 1980s.

It was further developed and practically implemented in the 2000s when K. S. Oh and K. Jung showed that Neural Network calculation can be accelerated using GPUs.

It contains three types of process:

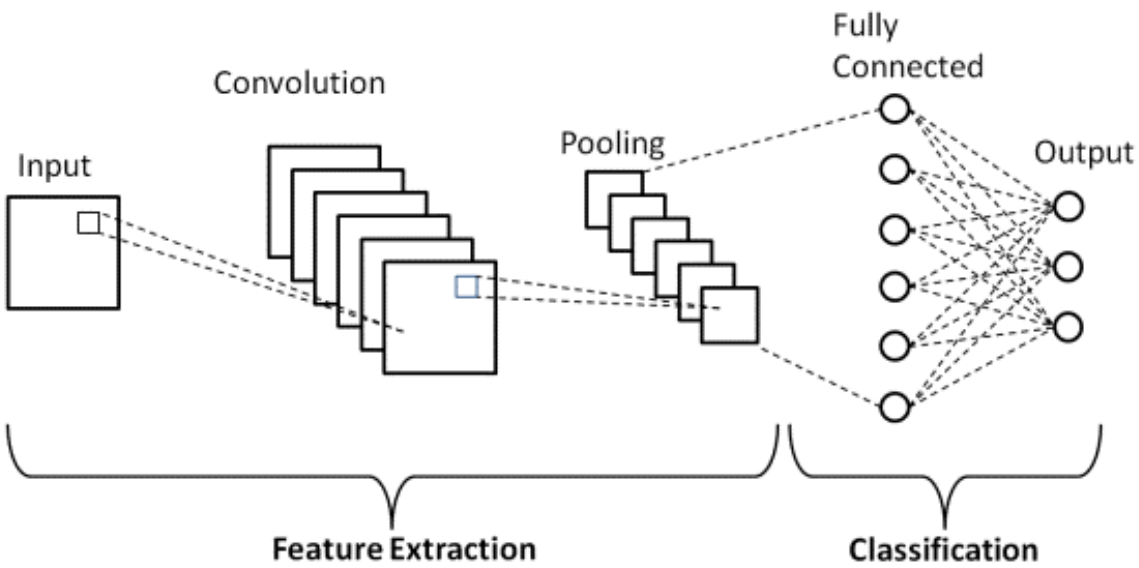
- Convolution
- Pooling
- Fully connected layer

CNN - Architecture

The convolution layer of the architecture is used to extract different features of the input data.

The Pooling layer reduces the dimension of the feature map

A Fully connected layer flattens the final feature map and then feeds it into a feed-forward neural network.

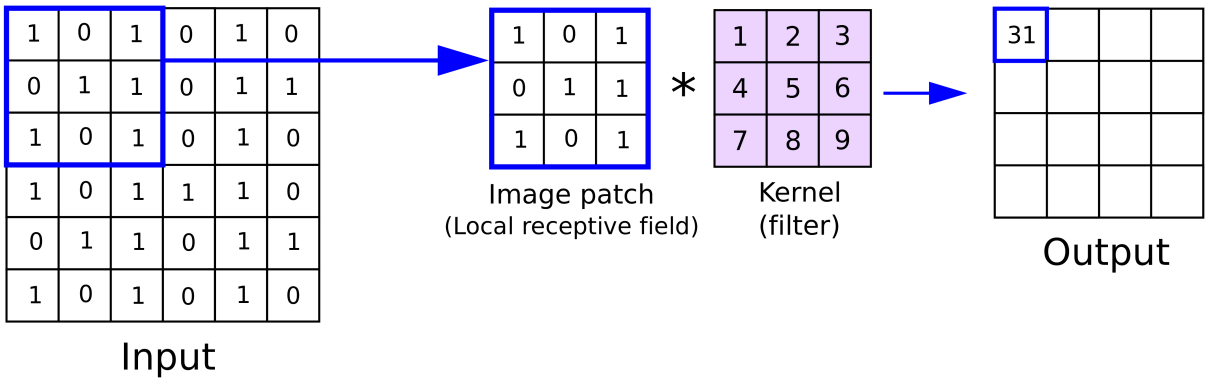


CNN - Convolution

Convolution is performed on an input matrix to generate a feature map.

It works with the use of a filter or kernel.

A single convolution can have multiple kernels to extract different sets of features depending upon the dimension of the input.



Output dimension of convolution $= \left(\frac{n-k+2p}{s}\right) + 1$.

Where n is the dimension of the input matrix, k is the dimension of the kernel, p is the padding, and s is the stride length.

CNN - Pooling

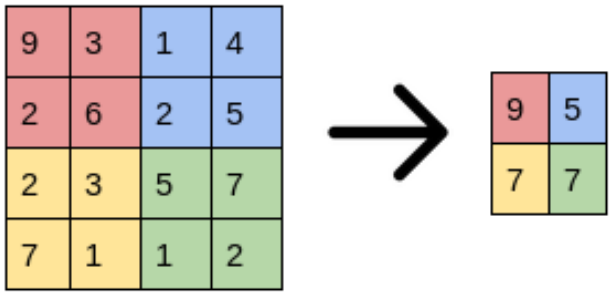
Pooling receives the input from the convolution layer and compresses it.

It helps to make the data from the convolution layer more abstract.

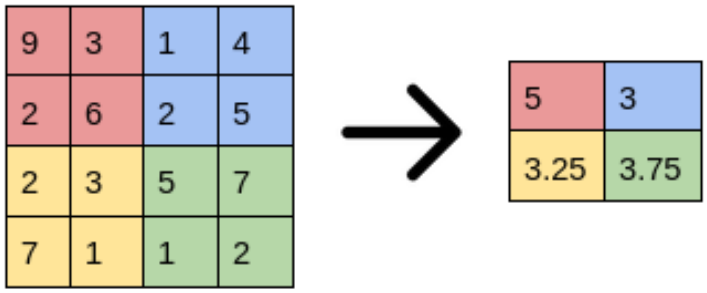
Uses a 2x2 matrix (patch) to compress the feature map.

Two types of pooling:

Maximum pooling : Calculates the maximum of the feature map part superimposed with the patch.
Average pooling : Calculate the average of the feature map part superimposed with the patch.



Max pooling : Filter = (2x2), Stride = (2,2)



Max pooling : Filter = (2x2), Stride = (2,2)

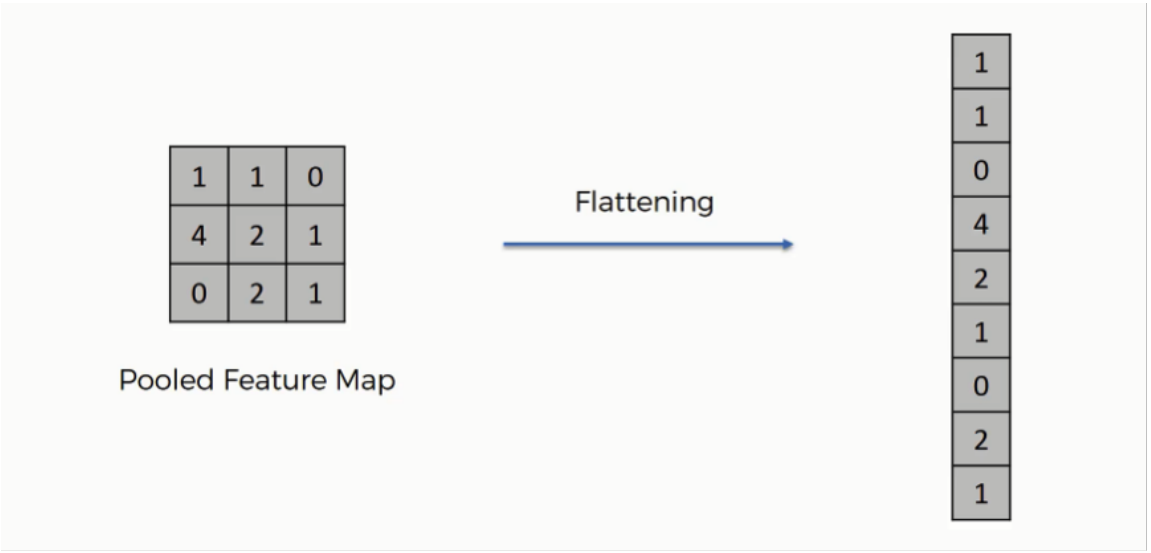
Output dimension of convolution $= \lfloor n/q \rfloor$.
Where n is the dimension of the input matrix, and q is the dimension of the patch taken.

CNN - Fully Connected Layer

Finally, the pooled layer of the CNN is converted to a 1-D array

This process is also called flattening.

The flattened layer is then fed to a feed-forward neural network and then the weights and bias terms are trained accordingly.

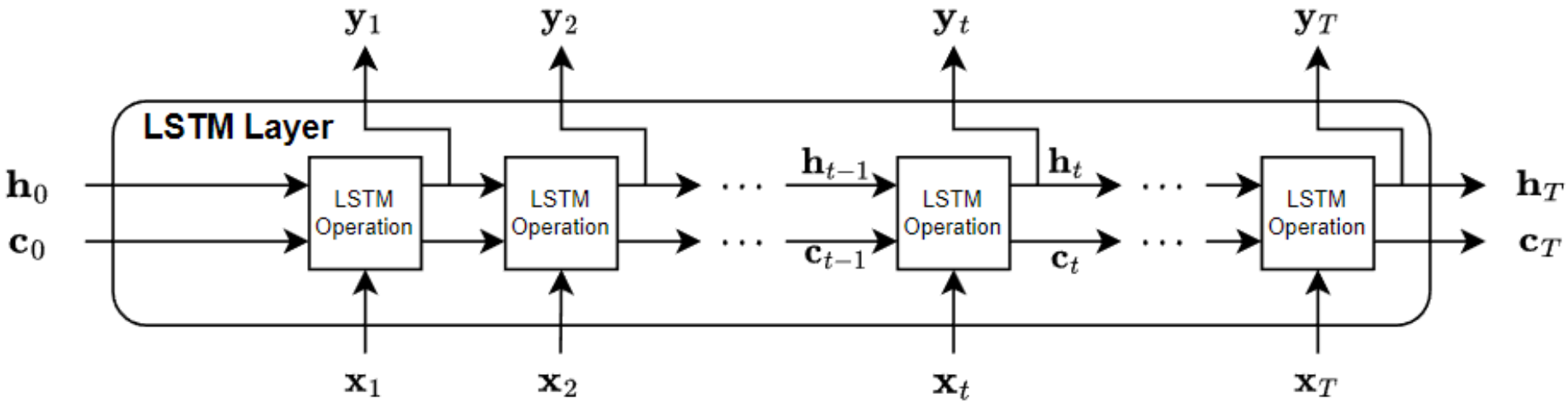


Long Short Term Memory Network

Long Short Term Memory (LSTM) network is a type of Recurrent Neural Network (RNN) that can detain long-term dependencies in **sequential data**.

They are intended to catch short-term dependencies in input data.

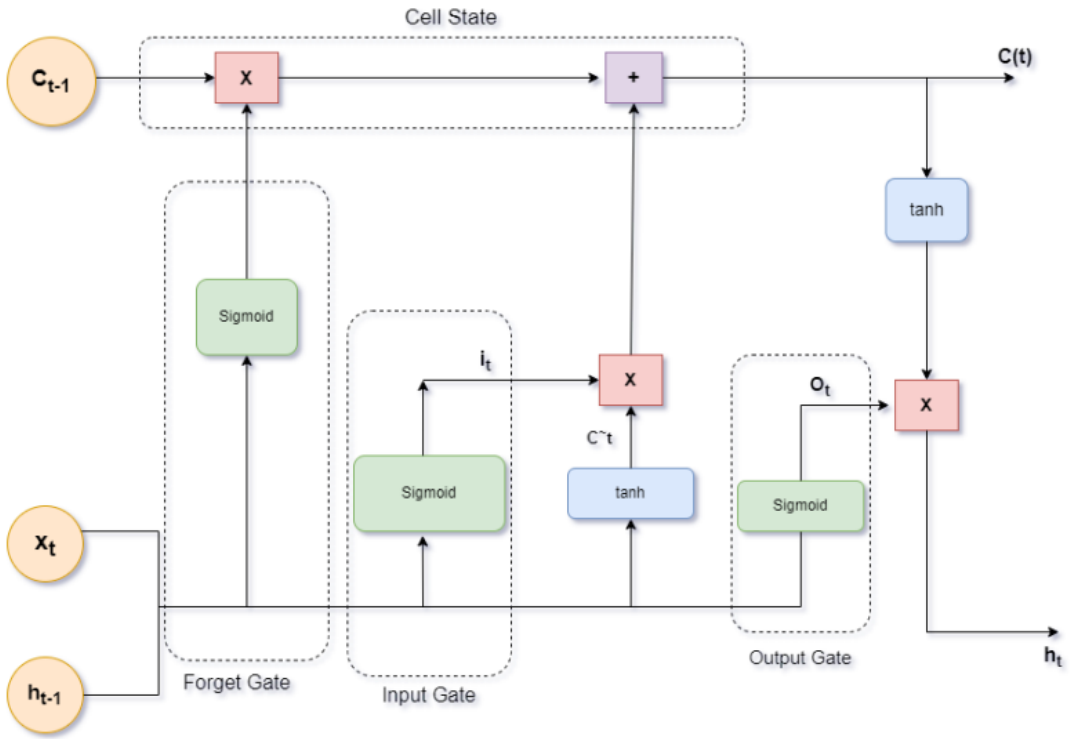
The algorithm for LSTM was proposed by two computer scientists, Sepp Hochreiter and Jurgen Schmidhuber, in 1997.



LSTM - Architecture

LSTM network is made up of several LSTM cells. Each LSTM cell contains three parts

- Forget gate
- Input gate
- Output gate



LSTM - Working

The input to the LSTM cells are $x_t, h_{t-1},$ and c_{t-1}, where t is the current time value and $t-1$ is the previous one.

Forget gate:

$$f_t = \sigma_g(W_f x_t + U_f c_{t-1} + b_f) \in (0,1)$$

Input gate:

$$i_t = \sigma_g(W_i x_t + U_i c_{t-1} + b_i) \in (0,1)$$

Output gate:

$$o_t = \sigma_g(W_o x_t + U_o c_{t-1} + b_o) \in (0,1)$$

This value is multiplied with $\tanh c_t$ and then updated as h_t

$$h_t = o_t \odot \sigma_h(c_t)$$

The c_t value is obtained as

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

where $\tilde{c}_t = \sigma_h(W_c x_c + U_c h_{t-1} + b_c)$

$x_t \in \mathbb{R}^d, f_t \in (0,1)^h, i_t \in (0,1)^h, o_t \in (0,1)^h$

$h_t \in (-1,1)^h, \tilde{c}_t \in (-1,1)^h, c_t \in \mathbb{R}^h$

$W \in \mathbb{R}^{h \times d}, U \in \mathbb{R}^{h \times h}, b \in \mathbb{R}^h$

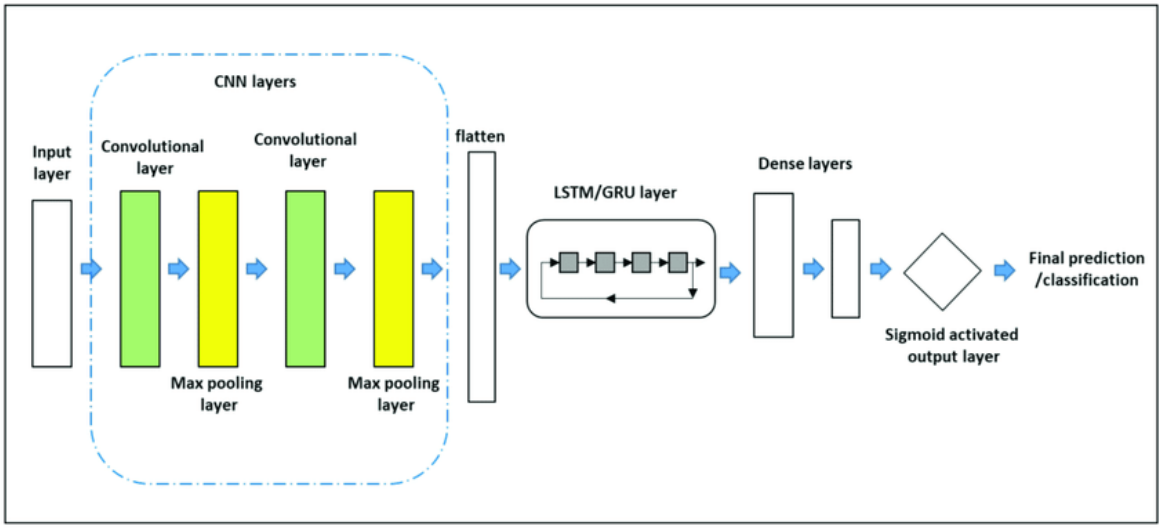
CNN - LSTM

CNN - LSTM is a hybrid algorithm for the prediction of sequential data.

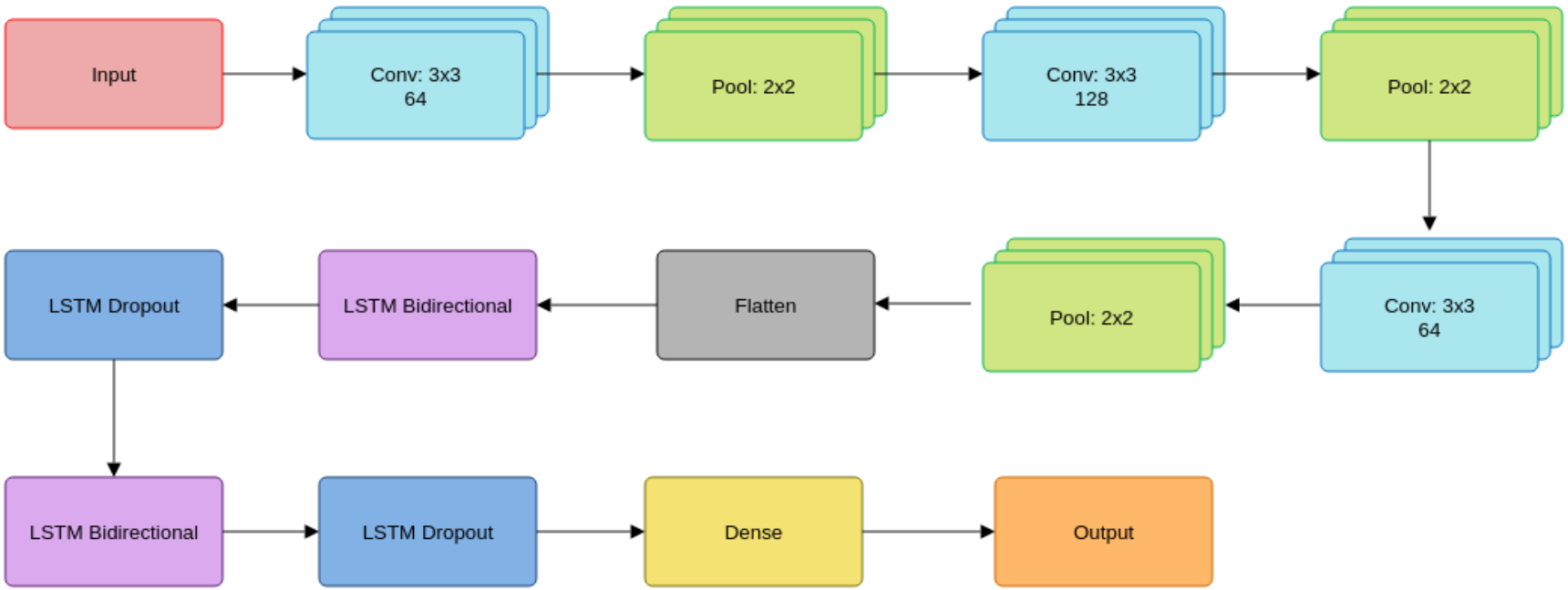
The input data is first fed through multiple convolution and pooling layers and then transformed into 1D data similar to CNN.

The 1D data is fed to an LSTM network which then uses the sequential data to define the short-term and temporal spatial dependencies in the data.

Finally, it is passed towards a dense layer, and then the backward propagation is commenced to find the weights and bias.



Methodology



Dataset

Quote-Equity-TECHM-EQ-11-03-2023-to-11-03-2024 (1)

Date	series	OPEN	HIGH	LOW	PREV. CLOSE	ltp	close	vwap	52W H	52W L	VOLUME	VALUE	No of trades
07-Mar-2024	EQ	1,275.00	1,293.10	1,265.55	1,271.00	1,283.20	1,288.15	1,281.70	1,416.30	981.05	20,73,691	2,65,78,50,269.40	94,123
06-Mar-2024	EQ	1,267.50	1,274.35	1,250.10	1,272.50	1,271.00	1,271.00	1,261.48	1,416.30	981.05	20,64,350	2,60,41,34,310.50	94,379
05-Mar-2024	EQ	1,274.90	1,279.90	1,259.10	1,280.05	1,270.60	1,272.50	1,271.83	1,416.30	981.05	14,20,379	1,80,64,78,256.30	91,152
04-Mar-2024	EQ	1,275.60	1,285.50	1,270.10	1,272.50	1,280.80	1,280.05	1,277.72	1,416.30	981.05	18,43,831	2,35,59,07,739.75	79,537
02-Mar-2024	EQ	1,278.00	1,282.00	1,271.00	1,271.80	1,271.90	1,272.50	1,274.88	1,416.30	981.05	2,17,683	27,75,19,800.50	6,538
01-Mar-2024	EQ	1,287.00	1,287.00	1,268.00	1,273.85	1,273.00	1,271.80	1,275.29	1,416.30	981.05	15,60,034	1,98,94,90,077.70	1,00,599
29-Feb-2024	EQ	1,280.00	1,285.20	1,264.25	1,285.85	1,278.90	1,273.85	1,275.31	1,416.30	981.05	22,51,498	2,87,13,47,959.85	1,19,463
28-Feb-2024	EQ	1,302.00	1,306.05	1,271.00	1,296.05	1,275.00	1,285.85	1,296.66	1,416.30	981.05	12,89,779	1,67,24,05,215.20	90,157
27-Feb-2024	EQ	1,295.00	1,305.05	1,283.70	1,298.25	1,293.25	1,296.05	1,295.42	1,416.30	981.05	14,88,661	1,92,84,42,169.55	81,153
26-Feb-2024	EQ	1,322.25	1,324.90	1,295.00	1,322.25	1,295.80	1,298.25	1,301.92	1,416.30	981.05	11,96,507	1,55,77,57,862.35	77,328
23-Feb-2024	EQ	1,338.90	1,342.55	1,316.30	1,328.10	1,319.90	1,322.25	1,329.57	1,416.30	981.05	16,60,506	2,20,77,53,978.00	74,224
22-Feb-2024	EQ	1,307.00	1,330.50	1,299.35	1,296.60	1,325.00	1,328.10	1,315.33	1,416.30	981.05	22,01,424	2,89,56,05,544.50	1,20,069
21-Feb-2024	EQ	1,320.00	1,325.00	1,292.20	1,320.55	1,298.55	1,296.60	1,309.26	1,416.30	981.05	17,69,508	2,31,67,39,662.50	67,783
20-Feb-2024	EQ	1,310.00	1,323.65	1,302.00	1,310.00	1,323.00	1,320.55	1,312.39	1,416.30	981.05	21,13,200	2,77,33,41,468.00	89,941
19-Feb-2024	EQ	1,302.95	1,318.00	1,287.45	1,302.55	1,308.30	1,310.00	1,305.79	1,416.30	981.05	14,96,111	1,95,36,06,928.00	71,193
16-Feb-2024	EQ	1,305.10	1,309.20	1,298.00	1,300.35	1,300.40	1,302.55	1,302.43	1,416.30	981.05	19,94,659	2,59,79,12,989.90	81,092
15-Feb-2024	EQ	1,304.70	1,308.00	1,291.55	1,292.90	1,299.35	1,300.35	1,300.62	1,416.30	981.05	17,03,763	2,21,59,54,346.60	87,600
14-Feb-2024	EQ	1,310.00	1,319.95	1,282.65	1,328.45	1,291.10	1,292.90	1,294.69	1,416.30	981.05	34,26,103	4,43,57,34,287.30	1,32,735
13-Feb-2024	EQ	1,321.95	1,334.80	1,300.00	1,318.80	1,334.80	1,328.45	1,316.49	1,416.30	981.05	11,27,361	1,48,41,62,645.60	1,03,251
12-Feb-2024	EQ	1,318.40	1,331.00	1,311.10	1,311.05	1,317.10	1,318.80	1,321.42	1,416.30	981.05	11,49,683	1,51,92,11,486.75	67,000
09-Feb-2024	EQ	1,317.00	1,321.45	1,305.95	1,309.25	1,310.00	1,311.05	1,311.77	1,416.30	981.05	10,85,349	1,42,37,27,984.10	78,853

Features used

Date: The specific day when the trades occurred.

Open: It is price at which a particular stock starts trading when the market opens for the day.

High:It is highest price at which a security traded during a specific period, typically within a single trading day.

Low: The lowest price at which a security traded during a specific period, usually within a single trading day.

Close: The final price at which a security trades at the end of a trading session.

Normalization Function

The normalization function is defined as:

$$Y_t = \frac{Z_t - \text{mean}}{\text{std. Deviation}}$$

where Z_t represents the feature vector for time t, Y_t is the normalized feature vector, and *mean*, and *std. Deviation* are the mean, standard deviation values of the feature vector, respectively.

Code

Stock Market Prediction using CNN-LSTM model

[1]
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os


import math
import seaborn as sns
import datetime as dt
from datetime import datetime
sns.set_style("whitegrid")
from pandas.plotting import autocorrelation_plot
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use("ggplot")
```

Then the datasets are loaded

[2]
```
data = pd.read_csv('tcs.csv')
data.head()
```

	Date	series	OPEN	HIGH	LOW	PREV. CLOSE	ltp	close	vwap	52W H	52W L	VOLUME	VALUE	No of trades
0	19-Apr-2024	EQ	3,838.00	3,852.55	3,800.90	3,862.00	3,822.15	3,826.20	3,827.74	4,254.75	3,070.25	29,57,749	11,32,15,04,897.25	2,52,913
1	18-Apr-2024	EQ	3,876.80	3,936.00	3,850.00	3,872.80	3,871.00	3,862.00	3,891.42	4,254.75	3,070.25	34,76,284	13,52,76,91,591.60	1,91,254
2	16-Apr-2024	EQ	3,902.00	3,928.70	3,862.85	3,941.20	3,875.95	3,872.80	3,888.73	4,254.75	3,070.25	30,51,420	11,86,61,47,836.30	1,47,341
3	15-Apr-2024	EQ	4,001.40	4,064.20	3,919.05	4,001.40	3,940.80	3,941.20	3,990.50	4,254.75	3,070.25	42,00,329	16,76,14,16,762.10	2,45,404
4	12-Apr-2024	EQ	3,971.00	4,013.35	3,945.50	3,984.65	4,003.80	4,001.40	3,990.61	4,254.75	3,070.25	43,54,821	17,37,83,76,127.55	2,00,517

Figure: library and data

Code

```
[4] data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 248 entries, 0 to 247
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Date           248 non-null    object
 1   series         248 non-null    object
 2   OPEN           248 non-null    object
 3   HIGH           248 non-null    object
 4   LOW            248 non-null    object
 5   PREV. CLOSE    248 non-null    object
 6   ltp            248 non-null    object
 7   close          248 non-null    object
 8   vwap           248 non-null    object
 9   52W H          248 non-null    object
 10  52W L          248 non-null    object
 11  VOLUME         248 non-null    object
 12  VALUE          248 non-null    object
 13  No of trades   248 non-null    object
dtypes: object(14)
memory usage: 27.2+ KB
```

```
[5] data.shape
```

```
(248, 14)
```

```
[6] # Rename columns
data.rename(columns={"Date ":"date","OPEN ":"open","HIGH ":"high","LOW ":"low","close ":"close", "PREV. close ":"prev.close","52W H ":"52wh", "52W L ":"52wl" ,
data.head()
```

	date	series	open	high	low	PREV. CLOSE	ltp	close	vwap	52wh	52wl	volume	value	no.oftrade
0	19-Apr-2024	EQ	3,838.00	3,852.55	3,800.90	3,862.00	3,822.15	3,826.20	3,827.74	4,254.75	3,070.25	29,57,749	11,32,15,04,897.25	2,52,913
1	18-Apr-2024	EQ	3,876.80	3,936.00	3,850.00	3,872.80	3,871.00	3,862.00	3,891.42	4,254.75	3,070.25	34,76,284	13,52,76,91,591.60	1,91,254

Figure: Data info

Code

```
[12] # Checking Data type of each column
     print("Date column data type: ", type(data['date'][0]))
     print("open column data type: ", type(data['open'][0]))
     print("close column data type: ", type(data['close'][0]))
     print("High column data type: ", type(data['high'][0]))
     print("Low column data type: ", type(data['low'][0]))


     data['open'] = data['open'].str.replace(',', '')
     data['open'] = data['open'].astype(float)

     data['open'] = data['open'].astype(float)


     data['close'] = data['close'].str.replace(',', '')
     data['close'] = data['close'].astype(float)

     data['close'] = data['close'].astype(float)


     data['high'] = data['high'].str.replace(',', '')
     data['high'] = data['high'].astype(float)

     data['high'] = data['high'].astype(float)


     data['low'] = data['low'].str.replace(',', '')
     data['low'] = data['low'].astype(float)
     data['low'] = data['low'].astype(float)
```

```
Date column data type:  <class 'str'>
open column data type:  <class 'str'>
close column data type:  <class 'str'>
High column data type:  <class 'str'>
Low column data type:  <class 'str'>
```

Figure: convert to float

Code

```
[15] data.plot(legend=True,subplots=True, figsize = (12, 6))
     plt.show()

     data.shape
     data.size
     data.describe(include='all').T
     data.dtypes
     data.nunique()
     ma_day = [10,50,100]

     for ma in ma_day:
         column_name = "MA for %s days" %(str(ma))
         data[column_name]=pd.DataFrame.rolling(data['close'],ma).mean()

     data['Daily Return'] = data['close'].pct_change()
     # plot the daily return percentage
     data['Daily Return'].plot(figsize=(12,5),legend=True,linestyle=':',marker='o')
     plt.show()

     sns.displot(data['Daily Return'].dropna(),bins=100,color='green')
     plt.show()

     date=pd.DataFrame(data['date'])
     closing_df1 = pd.DataFrame(data['close'])
     close1  = closing_df1.rename(columns={"close": "data_close"})
     close2=pd.concat([date,close1],axis=1)
     close2.head()

     #data.reset_index(drop=True, inplace=True)
     #data.fillna(data.mean(), inplace=True)
     data.head()

     data.nunique()

     data.sort_index(axis=1,ascending=True)

     cols_plot = ['open', 'high', 'low','close','volume','MA for 10 days','MA for 50 days','MA for 100 days','Daily Return']
     axes = data[cols_plot].plot(marker='.', alpha=0.7, linestyle='None', figsize=(11, 9), subplots=True)
     for ax in axes:
         ax.set_ylabel('Daily trade')

     plt.plot(data['close'], label="close price")
     plt.xlabel("Timestamp")
     plt.ylabel("Closing price")
     df = data
     print(df)
```

Figure: figure plotting

Code

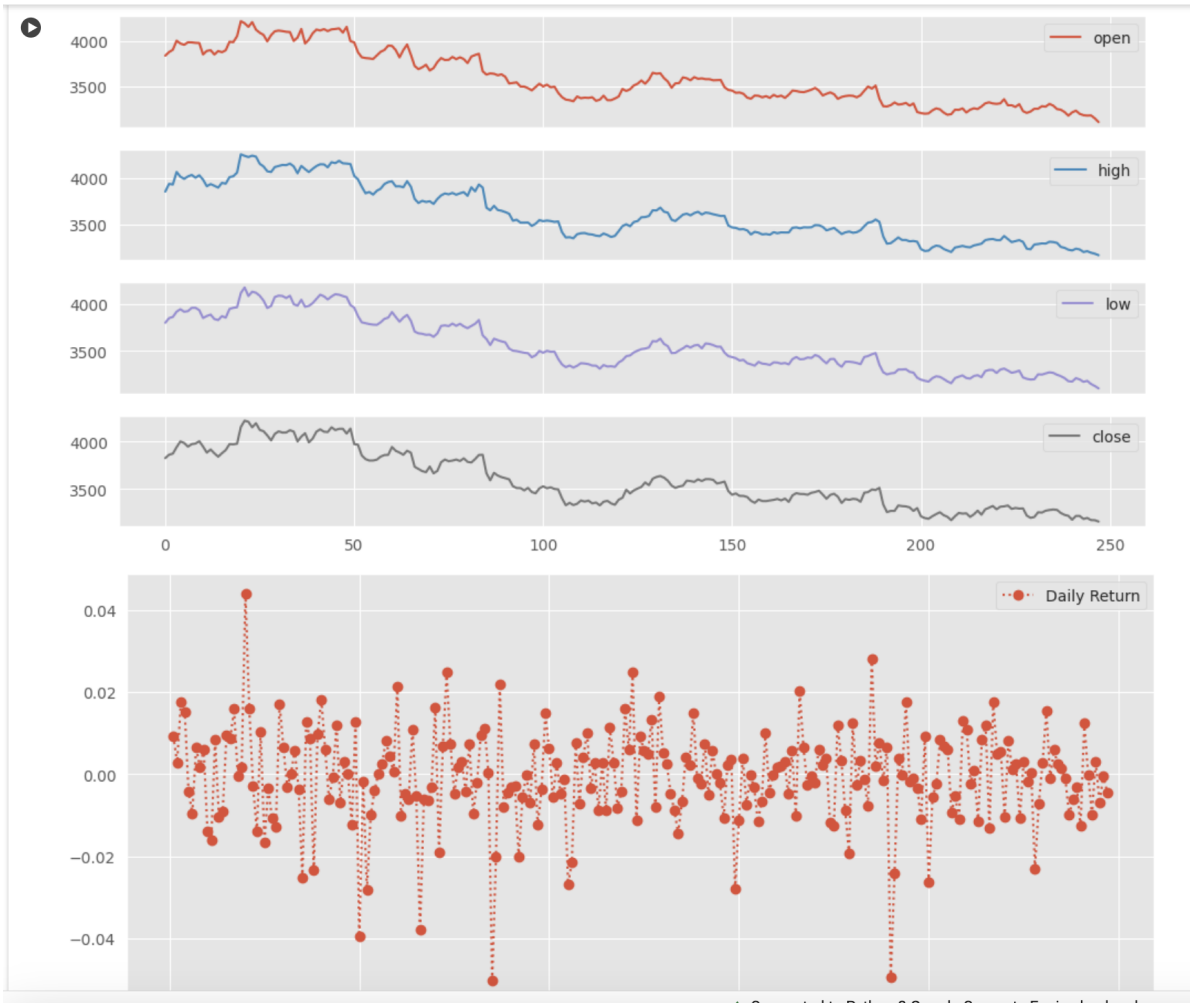


Figure: graph

Code

```
[ ] from sklearn.model_selection import train_test_split
    import numpy as np
    import matplotlib.pyplot as plt
    import tensorflow as tf
    from tensorflow.keras.layers import Conv1D, LSTM, Dense, Dropout, Bidirectional, TimeDistributed, MaxPooling1D, Flatten
    from tensorflow.keras.regularizers import L1, L2
    from tensorflow.keras.metrics import Accuracy, RootMeanSquaredError
    from sklearn.metrics import explained_variance_score, r2_score, max_error

    # Function to create and train the model
    def create_and_train_model(window_size):
        X = []
        Y = []
        for i in range(1, len(df) - window_size - 1, 1):
            first = df.iloc[i, 2]
            temp = []
            temp2 = []
            for j in range(window_size):
                temp.append((df.iloc[i + j, 2] - first) / first)
            temp2.append((df.iloc[i + window_size, 2] - first) / first)
            X.append(np.array(temp).reshape(window_size, 1))
            Y.append(np.array(temp2).reshape(1, 1))

        x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, shuffle=True)

        train_X = np.array(x_train)
        test_X = np.array(x_test)
        train_Y = np.array(y_train)
        test_Y = np.array(y_test)

        train_X = train_X.reshape(train_X.shape[0], 1, window_size, 1)
        test_X = test_X.reshape(test_X.shape[0], 1, window_size, 1)

        model = tf.keras.Sequential()

        # CNN layers
        model.add(TimeDistributed(Conv1D(64, kernel_size=3, activation='relu', padding='same', input_shape=(None, window_size, 1))))
        model.add(TimeDistributed(MaxPooling1D(2)))
        model.add(TimeDistributed(Conv1D(128, kernel_size=3, activation='relu')))
        model.add(TimeDistributed(MaxPooling1D(2)))
        model.add(TimeDistributed(Conv1D(64, kernel_size=3, activation='relu')))
        model.add(TimeDistributed(MaxPooling1D(2)))
        model.add(TimeDistributed(Flatten()))

        # LSTM layers
        model.add(Bidirectional(LSTM(100, return_sequences=True)))
```

Figure: CNN + LSTM

Code

```
[ ]
        train_X = train_X.reshape(train_X.shape[0], 1, window_size, 1)
        test_X = test_X.reshape(test_X.shape[0], 1, window_size, 1)

        model = tf.keras.Sequential()

        # CNN layers
        model.add(TimeDistributed(Conv1D(64, kernel_size=3, activation='relu', padding='same', input_shape=(None, window_size, 1))))
        model.add(TimeDistributed(MaxPooling1D(2)))
        model.add(TimeDistributed(Conv1D(128, kernel_size=3, activation='relu')))
        model.add(TimeDistributed(MaxPooling1D(2)))
        model.add(TimeDistributed(Conv1D(64, kernel_size=3, activation='relu')))
        model.add(TimeDistributed(MaxPooling1D(2)))
        model.add(TimeDistributed(Flatten()))

        # LSTM layers
        model.add(Bidirectional(LSTM(100, return_sequences=True)))
        model.add(Dropout(0.5))
        model.add(Bidirectional(LSTM(100, return_sequences=False)))
        model.add(Dropout(0.5))

        # Final layers
        model.add(Dense(1, activation='linear'))
        model.compile(optimizer='adam', loss='mse', metrics=['mse', 'mae'])

        history = model.fit(train_X, train_Y, validation_data=(test_X, test_Y), epochs=40, batch_size=40, verbose=0, shuffle=True)

        model.evaluate(test_X, test_Y)

        # Predict probabilities for test set
        yhat_probs = model.predict(test_X, verbose=0)
        # Reduce to 1d array
        yhat_probs = yhat_probs[:, 0]

        var = explained_variance_score(test_Y.reshape(-1, 1), yhat_probs)
        r2 = r2_score(test_Y.reshape(-1, 1), yhat_probs)
        var2 = max_error(test_Y.reshape(-1, 1), yhat_probs)

        return var, r2, var2 , model

    # List of window lengths to iterate over
    window_lengths = [20,30,40,50,60]

    # Dictionary to store metrics for each window length
    metrics = {}

    # Iterate over window lengths and train models
    # Iterate over window lengths and train models
```

Figure: CNN + LSTM

Code

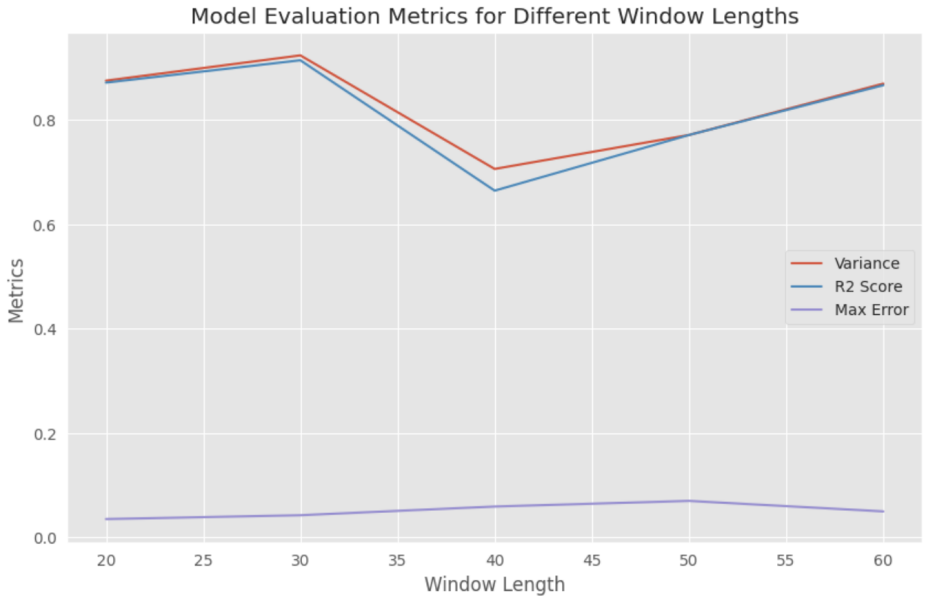
```
[ ] # Iterate over window lengths and train models
    # Iterate over window lengths and train models
    for window_length in window_lengths:
        print(f"Window Length: {window_length}")
        var, r2, var2, model = create_and_train_model(window_length)
        print(f"Variance: {var}")
        print(f"R2 Score: {r2}")
        print(f"Max Error: {var2}")
        metrics[window_length] = {'Variance': var, 'R2 Score': r2, 'Max Error': var2}


    # Plotting the metrics
    plt.figure(figsize=(10, 6))
    plt.plot(window_lengths, [metrics[length]['Variance'] for length in window_lengths], label='Variance')
    plt.plot(window_lengths, [metrics[length]['R2 Score'] for length in window_lengths], label='R2 Score')
    plt.plot(window_lengths, [metrics[length]['Max Error'] for length in window_lengths], label='Max Error')
    plt.xlabel("Window Length")
    plt.ylabel("Metrics")
    plt.title("Model Evaluation Metrics for Different Window Lengths")
    plt.legend()
    plt.show()
```

```
Window Length: 20
2/2 [==============================] - 0s 12ms/step - loss: 2.4271e-04 - mse: 2.4271e-04 - mae: 0.0125
Variance: 0.874931299214075
R2 Score: 0.8711870818747247
Max Error: 0.034862545940423914
Window Length: 30
2/2 [==============================] - 0s 13ms/step - loss: 2.4964e-04 - mse: 2.4964e-04 - mae: 0.0128
Variance: 0.9233987070069901
R2 Score: 0.9140037528415105
Max Error: 0.04222556139885351
Window Length: 40
2/2 [==============================] - 0s 10ms/step - loss: 4.7866e-04 - mse: 4.7866e-04 - mae: 0.0165
Variance: 0.7057876777074898
R2 Score: 0.6641711065777941
Max Error: 0.05887276920349743
Window Length: 50
2/2 [==============================] - 2s 13ms/step - loss: 5.6340e-04 - mse: 5.6340e-04 - mae: 0.0175
Variance: 0.7708464510132211
R2 Score: 0.7708454797394308
Max Error: 0.06967705894303036
Window Length: 60
2/2 [==============================] - 0s 12ms/step - loss: 3.2227e-04 - mse: 3.2227e-04 - mae: 0.0135
WARNING:tensorflow:5 out of the last 9 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7d9eed453640> t
Variance: 0.869251411360256
R2 Score: 0.8662314103063171
Max Error: 0.04957315800110916
```

Figure: result

Code




```
[ ] # After the model has been constructed, we'll summarise it
    from tensorflow.keras.utils import plot_model
    print(model.summary())
    plot_model(model, to_file='model.png', show_shapes=True, show_layer_names=True)
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 time_distributed_28 (TimeD  (None, 1, 60, 64)         256
 istributed)

 time_distributed_29 (TimeD  (None, 1, 30, 64)         0
 istributed)
```

Figure: result by graph

Code

SVM

```
[ ]  #using svm to predict stock
     import numpy as np
     import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn import svm,preprocessing
     from sklearn.metrics import classification_report
```

```
[ ]  def get_x_and_y(price,window_length=7,predict_day_length=1):
         '''get train and test set
         every time get window from price and
         '''
         m = len(price.iloc[0])
         n = len(price) - window_length
         m = window_length * m

         x = np.ones((n,m))
         y = np.ones((n,1))

         for i in range(len(price)-window_length):
             #ans = [float(price.iloc[j,4]) for j in range(i,i+window_length)]
             ans = [list(price.iloc[j].tolist() for j in range(i,i+window_length))]
             #ans = [list(price.iloc[j] for j in range(i,i+window_length))]
             ans = np.array(ans).flatten()
             x[i] = ans
             y[i] = 1 if price.close[i+window_length+predict_day_length-1] - price.close[i+window_length-1] >0 else 0
         return [x,y]
```

Figure: SVM

Code

```
[ ]  !pip install sklearn
     import sklearn.preprocessing

     window_lengths = [20,30,40,50,60]
     accurarys = {}
     reports ={}

     def train_and_test(price, window_length, accurarys, reports):
         x,y = get_x_and_y(data,window_length,predict_day_length=1)
         y = y.flatten()
         scaler = sklearn.preprocessing.StandardScaler()
         scaler.fit_transform(x)
         x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=233)
         for kernel_arg in ['rbf','poly','linear']:
             clf = svm.SVC(kernel=kernel_arg,max_iter=5000)
             clf.fit(x_train,y_train)
             y_predict = clf.predict(x_test)

             accurary = clf.score(x_test,y_test)
             report = classification_report(y_test,y_predict,target_names = ['drop','up'])
             if window_length in accurarys:
                 accurarys[window_length].append(accurary)
                 reports[window_length].append(report)
             else:
                 accurarys[window_length] = [accurary]
                 reports[window_length] = [report]
             print('The Accurary of %s : %f'%(kernel_arg,clf.score(x_test,y_test)))
             print(report)

     Collecting sklearn
```

Figure: svm

Code

```
[ ]  for l in window_lengths:
         print('window_length:',l)
         train_and_test(data,l,accurarys,reports)
```

```
window_length: 20
The Accurary of rbf : 0.350877
              precision    recall  f1-score   support

        drop       0.00      0.00      0.00        37
          up       0.35      1.00      0.52        20

    accuracy                           0.35        57
   macro avg       0.18      0.50      0.26        57
weighted avg       0.12      0.35      0.18        57

The Accurary of poly : 0.368421
              precision    recall  f1-score   support

        drop       0.57      0.11      0.18        37
          up       0.34      0.85      0.49        20

    accuracy                           0.37        57
   macro avg       0.46      0.48      0.33        57
weighted avg       0.49      0.37      0.29        57

The Accurary of linear : 0.456140
              precision    recall  f1-score   support

        drop       0.67      0.32      0.44        37
          up       0.36      0.70      0.47        20

    accuracy                           0.46        57
   macro avg       0.51      0.51      0.46        57
weighted avg       0.56      0.46      0.45        57

window_length: 30
```

Figure: svm output

Code

Random Forest

```
[ ]  from sklearn.ensemble import RandomForestClassifier
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import classification_report


     accuracies = {}
     reports = {}

     def train_and_test_rf(price, window_length, accuracies, reports):
         x, y = get_x_and_y(price, window_length, predict_day_length=1)
         y = y.flatten()
         scaler = sklearn.preprocessing.StandardScaler()
         x_scaled = scaler.fit_transform(x)
         x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.25, random_state=233)

         clf = RandomForestClassifier(n_estimators=100, random_state=0)
         clf.fit(x_train, y_train)
         y_predict = clf.predict(x_test)

         accuracy = clf.score(x_test, y_test)
         report = classification_report(y_test, y_predict, target_names=['drop', 'up'])

         if window_length in accuracies:
             accuracies[window_length].append(accuracy)
             reports[window_length].append(report)
         else:
             accuracies[window_length] = [accuracy]
             reports[window_length] = [report]

         print('The Accuracy of Random Forest: %f' % (accuracy))
         print(report)
```

Figure: Random Forest

Code

```
[ ]            reports[window_length] = [report]

        print('The Accuracy of Random Forest: %f' % (accuracy))
        print(report)

    for l in window_lengths:
        print('window_length:', l)
        train_and_test_rf(data, l, accuracies, reports)
```

```
window_length: 20
The Accuracy of Random Forest: 0.368421
              precision    recall  f1-score   support

        drop       0.52      0.46      0.49        37
          up       0.17      0.20      0.18        20

    accuracy                           0.37        57
   macro avg       0.34      0.33      0.33        57
weighted avg       0.39      0.37      0.38        57

window_length: 30
The Accuracy of Random Forest: 0.490909
              precision    recall  f1-score   support

        drop       0.51      0.62      0.56        29
          up       0.45      0.35      0.39        26

    accuracy                           0.49        55
   macro avg       0.48      0.48      0.48        55
weighted avg       0.48      0.49      0.48        55

window_length: 40
The Accuracy of Random Forest: 0.423077
              precision    recall  f1-score   support

        drop       0.43      0.46      0.44        26
```

Figure: Random forest accuracy

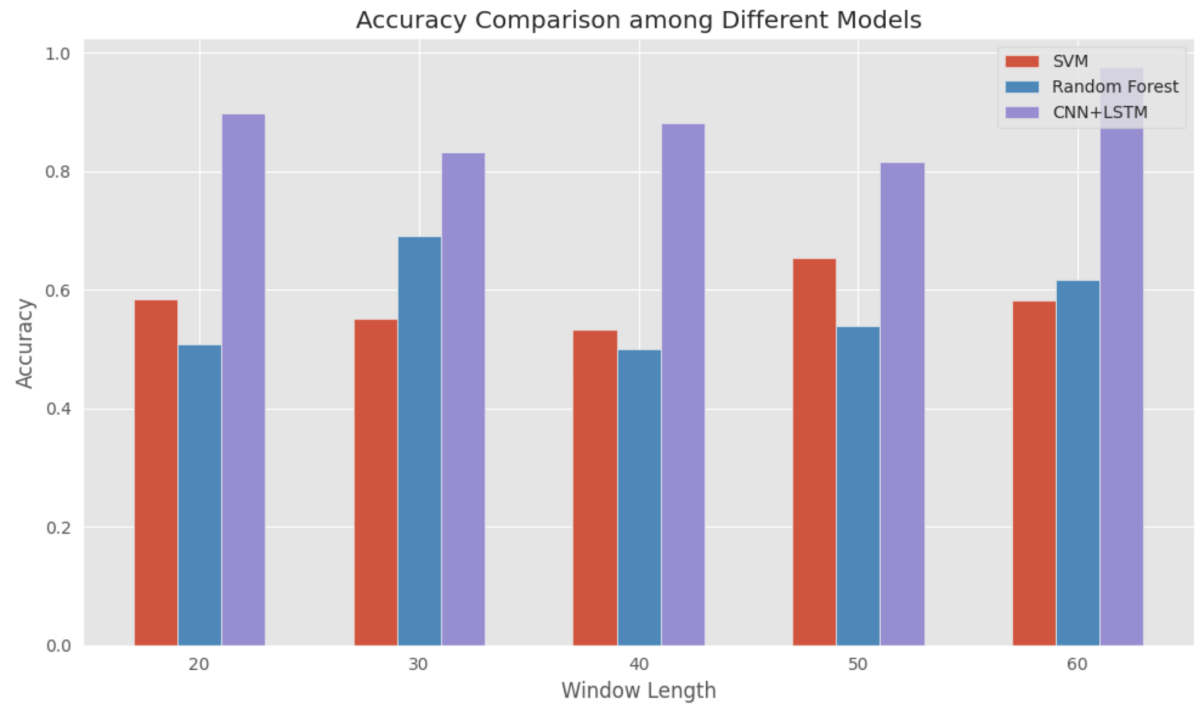
Experimental Results:

The project presented an algorithm for stock market forecast using (CNN+LSTM). Validation of it was done through trading simulations on 6 financial stocks.
In comparison with historical price data and different window sizes for prediction, three different models; SVM, Random Forest, and our own CNN+LSTM model were tested in relation to each other.

Below are the tables showing accuracy for different algorithm for different window lengths and bar-plots

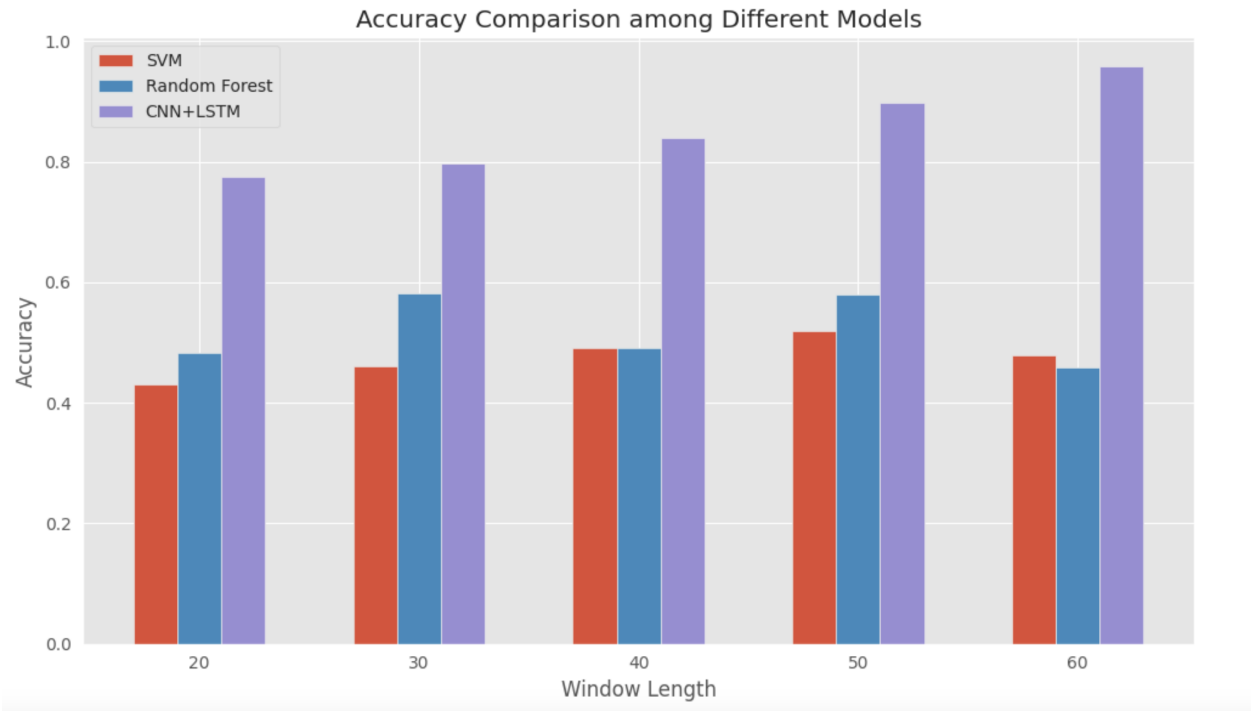
SBI Stocks Accuracy

	Window Length	CNN+LSTM Accuracy	SVM Accuracy	Random Forest Accuracy
0	20	0.869503	0.584795	0.508772
1	30	0.838080	0.551515	0.690909
2	40	0.839800	0.532051	0.500000
3	50	0.849352	0.653333	0.540000
4	60	0.958249	0.581560	0.617021



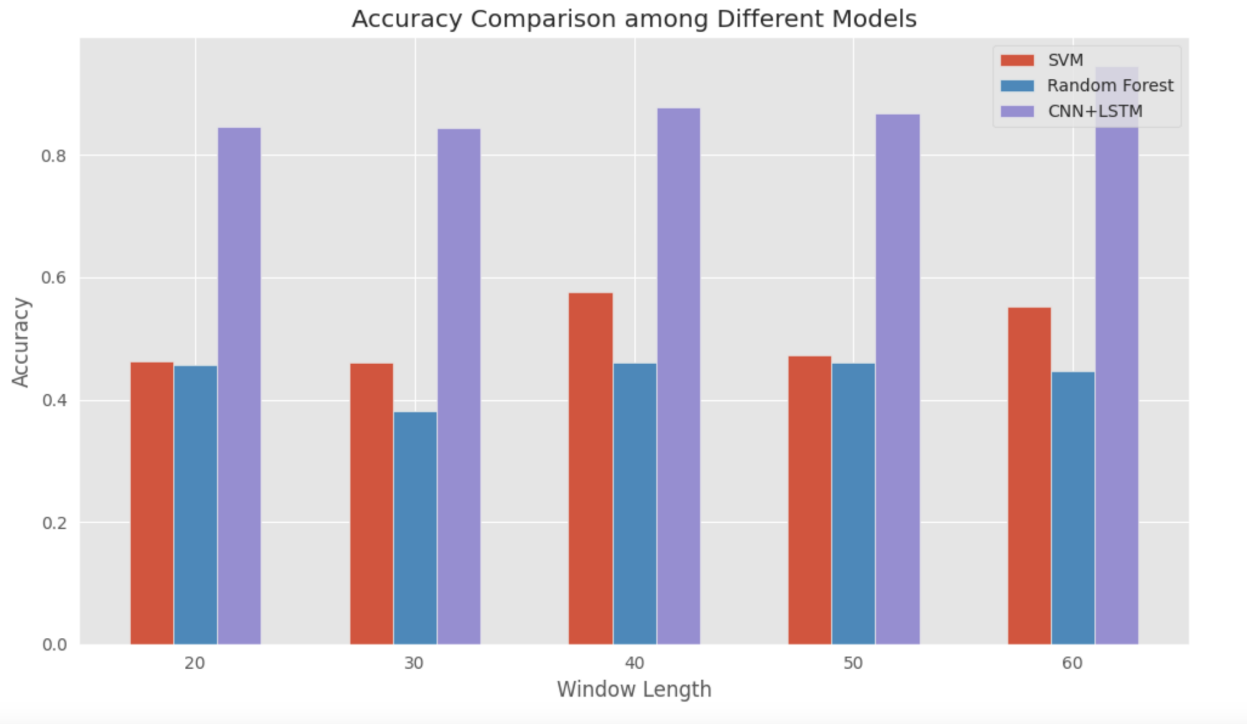
IDEA Stocks Accuracy

	Window Length	CNN+LSTM Accuracy	SVM Accuracy	Random Forest Accuracy
0	20	0.886800	0.431034	0.482759
1	30	0.798072	0.460606	0.581818
2	40	0.817496	0.490566	0.490566
3	50	0.921629	0.520000	0.580000
4	60	0.947969	0.479167	0.458333



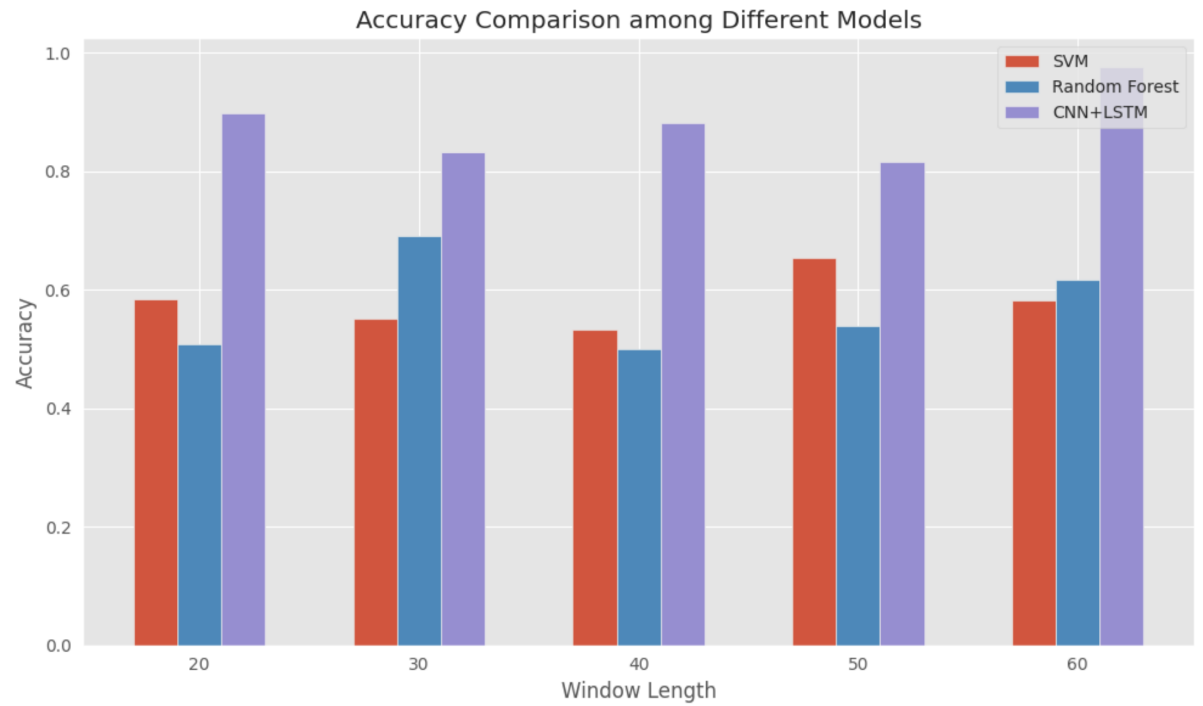
Tata Motor Stocks Accuracy

	Window Length	CNN+LSTM Accuracy	SVM Accuracy	Random Forest Accuracy
0	20	0.843877	0.461988	0.456140
1	30	0.796264	0.460606	0.381818
2	40	0.816698	0.576923	0.461538
3	50	0.924799	0.473333	0.460000
4	60	0.925551	0.553191	0.446809



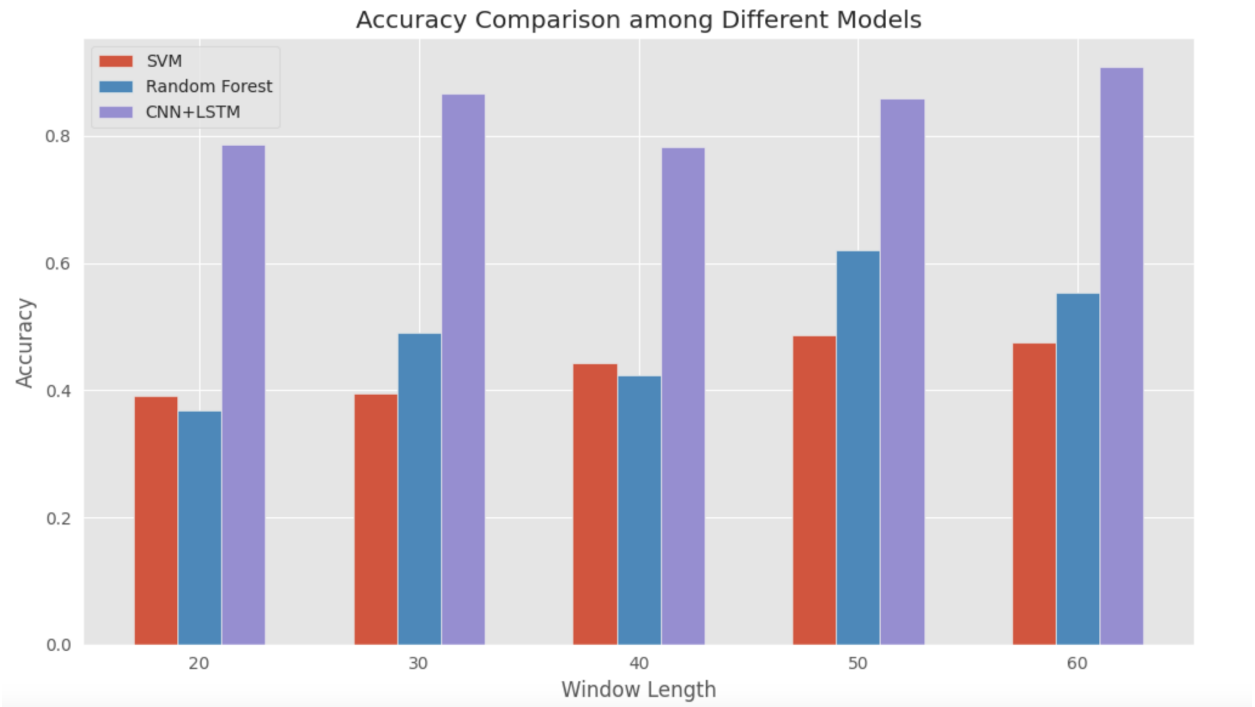
Adani Ports Stocks Accuracy

	Window Length	CNN+LSTM Accuracy	SVM Accuracy	Random Forest Accuracy
0	20	0.789267	0.568966	0.482759
1	30	0.915473	0.551515	0.563636
2	40	0.893336	0.528302	0.509434
3	50	0.833451	0.513333	0.500000
4	60	0.957952	0.562500	0.625000



TCS Stocks Accuracy

	Window Length	CNN+LSTM Accuracy	SVM Accuracy	Random Forest Accuracy
0	20	0.856659	0.391813	0.368421
1	30	0.859218	0.393939	0.490909
2	40	0.760773	0.442308	0.423077
3	50	0.820144	0.486667	0.620000
4	60	0.905907	0.475177	0.553191



Tech Mahindra Stocks Accuracy

	Window Length	CNN+LSTM Accuracy	SVM Accuracy	Random Forest Accuracy
0	20	0.899774	0.488506	0.482759
1	30	0.870160	0.478788	0.490909
2	40	0.819692	0.433962	0.452830
3	50	0.863173	0.466667	0.400000
4	60	0.886486	0.388889	0.312500

Experimental Results

Based on above graphs of different stocks , we discovered that for certain window lengths, the accuracy from SVM surpasses that of Random Forest, with both achieving 50% or higher accuracy. However, for other window lengths, Random Forest outperforms SVM across all stocks.
Moreover,we observed that the accuracy achieved by CNN-LSTM consistently exceeds 90%.
Thus, overall, CNN-LSTM emerges as the superior model, consistently achieving accuracy levels above 90%.
Prior to this, SVM showed better performance, followed by Random Forest, which was the least accurate.

Conclusion

Experimental results suggest its superiority over benchmark methodologies like SVM and Random Forest in forecasting stock trends.

This model holds promise in revolutionizing stock market prediction strategies.

Thank you