CAP 6615 - NEURAL NETWORKS
Programming Assignment 4
Recurrent Neural Network + Convolutional Neural Network for Time Series Prediction

Monica Bhargavi Kodali
Sravanti Ratnakaram
Pagolu Carol Navya
Karthik Gannamaneni
Pramod Kumar Varma Manthena

Spring Semester 2021
22 April 2021

ABSTRACT

This assignment uses the Recurrent Neural Network (RNN) model. A Recurrent Neural Network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. It is a form of generalized feed forward network that has internal memory which can be used to process the input data.

Along with the RNN , a **convolution neural network (CNN)** is also used. A convolutional neural network consists of an **input layer, hidden layers and an output layer**. In any feed-forward neural network, middle layers are called hidden because their inputs and outputs are masked by the activation function and final convolution. In a convolutional neural network, the hidden layers include layers that perform convolutions. Typically this includes a layer that does multiplication or other dot product, and its activation function is commonly **ReLU**.

This is followed by other layers such as pooling layers, fully connected layers, and normalization layers. In this assignment, a recurrent neural network (RNN) is designed and trained on **S&P 500 data**. The model is tested by predicting the next one,two,three or four values using a sliding sampling window of 180 days.

The models are also re-tested on *noise-corrupted* data to observe the performance. The goal of this project is to optimize the RNN to give the best possible results.

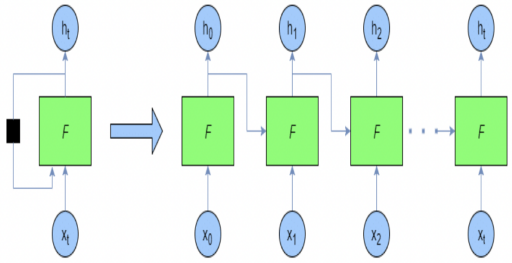


Figure 1: Sample Recurrent Neural Network

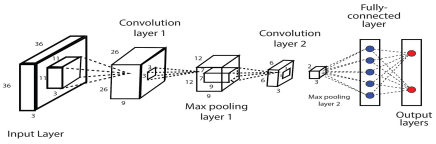


Figure 2: Sample Convolutional Neural Network

TABLE OF CONTENTS

1. DATASET GENERATION

The S&P 500 dataset is collected from the *Yahoo Finance website* between the time frame of January 4, 1960 to December 31, 2020. The table from the website is scraped and stored into a csv file. Shiller P/E ratio is one of the standard metrics used to evaluate whether a market is overvalued, undervalued, or fairly-valued. It can be downloaded from *the Shiller-PE website*.

The Shiller P/E ratio data was collected monthly from 1960 to 2020. We use a linear interpolation method to convert this monthly data to daily data. The formula for linear interpolation is as follows:

$$y = y1 + ((x - x1) / (x2 - x1)) * (y2 - y1)$$ where,

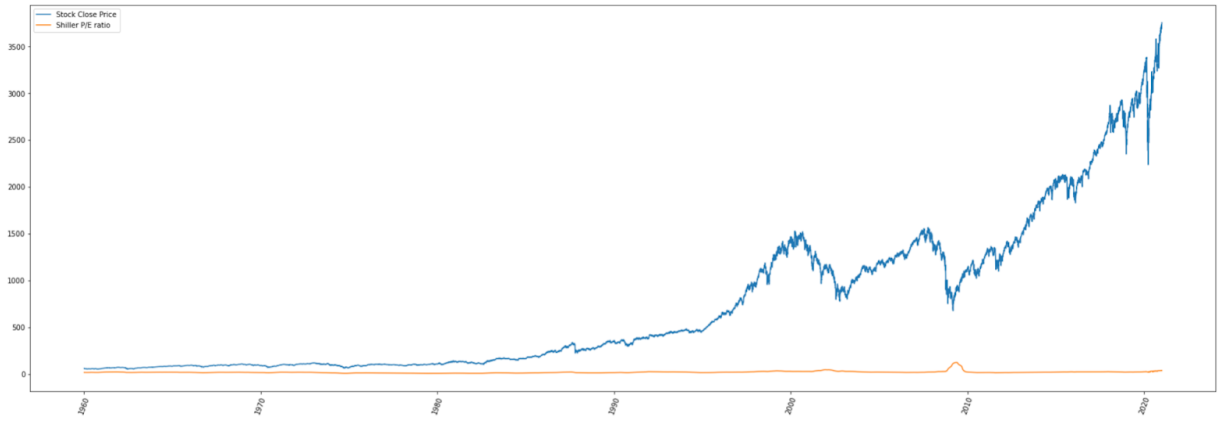
X = known value
y= unknown value
(x1,y1) = coordinates below the known value
(x2,y2) = coordinates above the known value

S&P 500 data and Shiller P/E ratio data were then merged and stored in *finalDS.csv.* The data was then divided into training and test sets.

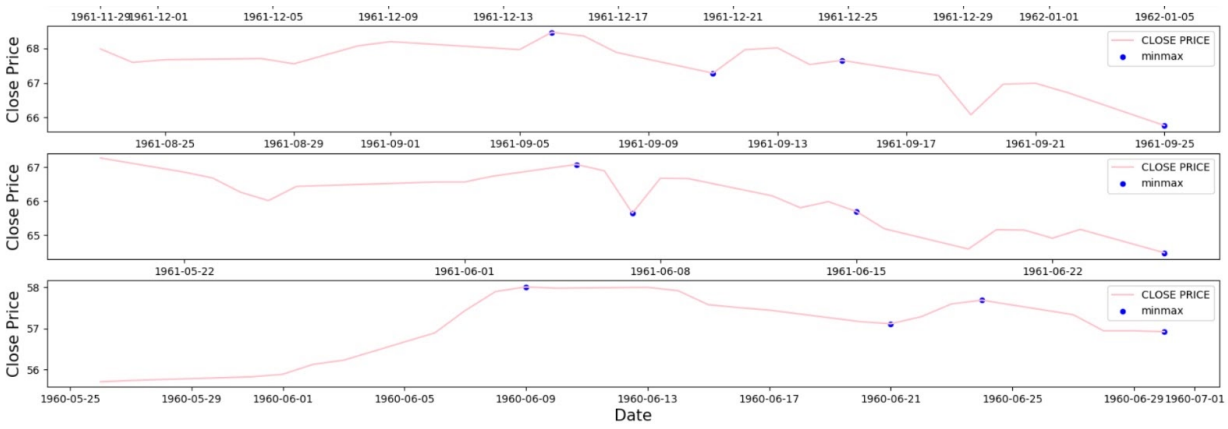
	Unnamed: 0	Adj Close	Close	Date	High	Low	Open	Volume	S&P 500 PE Ratio
0	0	3756.07	3756.07	2020-12-31	3760.20	3726.88	3733.27	3172510000	37.850000
1	1	3732.04	3732.04	2020-12-30	3744.63	3730.21	3736.19	3145200000	37.842333
2	2	3727.04	3727.04	2020-12-29	3756.12	3723.31	3750.01	3387030000	37.834667
3	3	3735.36	3735.36	2020-12-28	3740.51	3723.03	3723.03	3527460000	37.827000
4	4	3703.06	3703.06	2020-12-24	3703.82	3689.32	3694.03	1885090000	37.796333

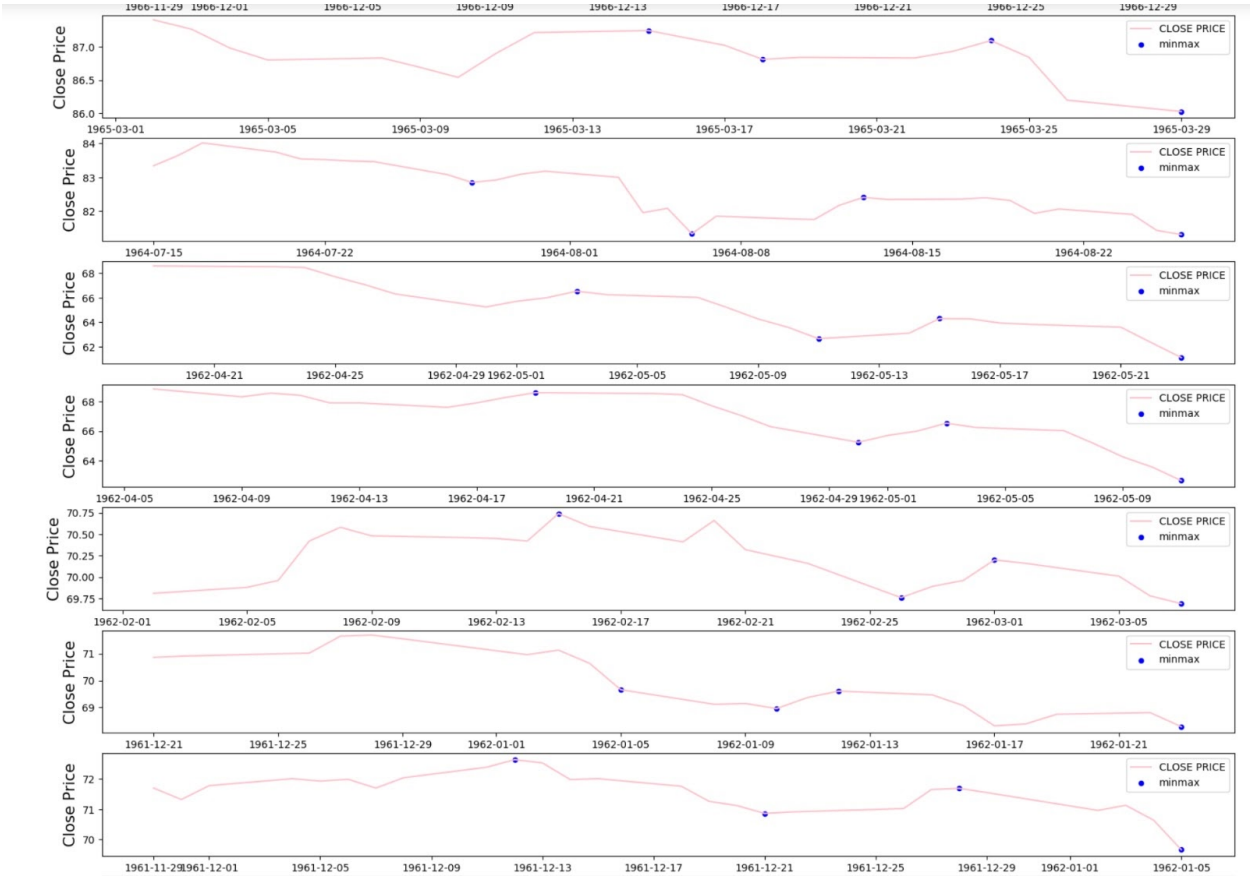
Figure 3: snapshot of dataset

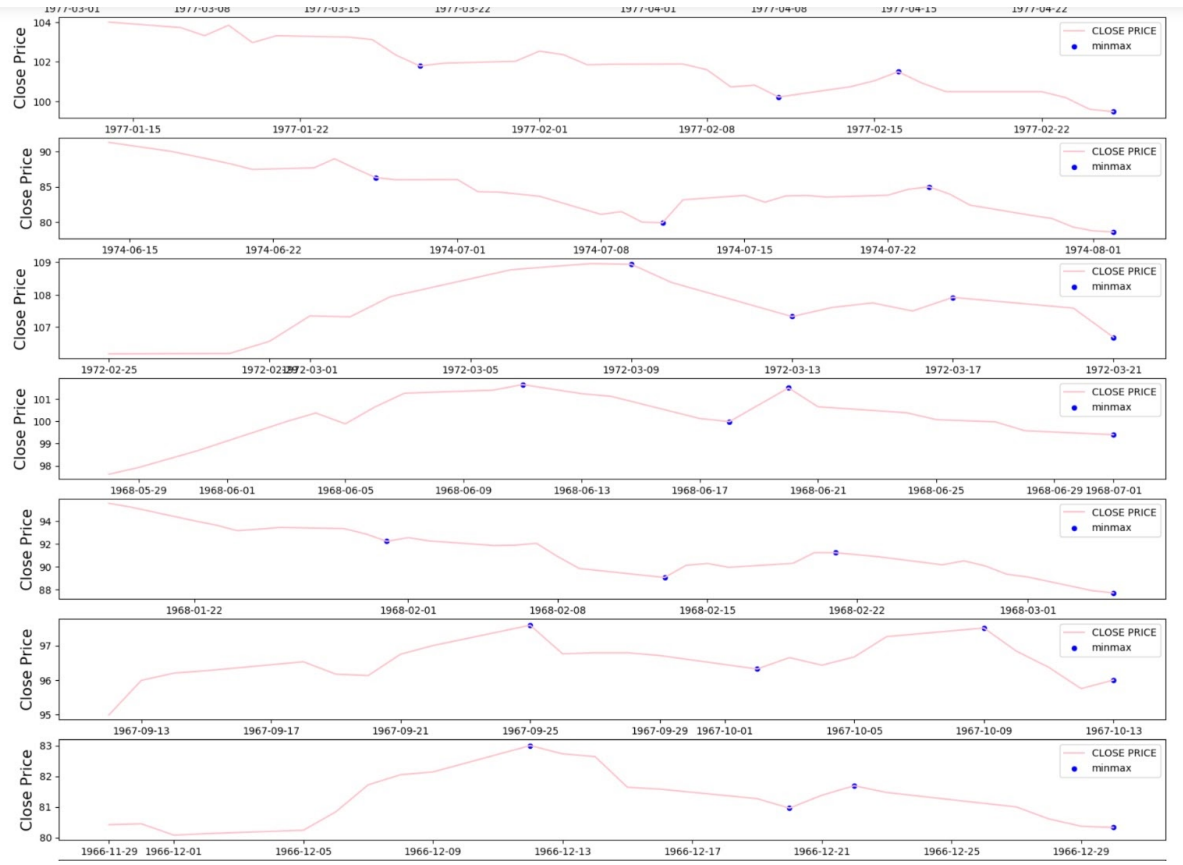
The generated dataset is a time series data of length N, defined as $CP_0, CP_1, \ldots, CP_{N-1}$ in which CP_i is the close price on day 'i', $0 \leq i < N$. The entire dataset is divided into fixed window size of W = 180. Therefore, whenever the window is moved to the right by size W , there is no overlapping between data in all the sliding windows. Then the data values are normalized within range of [0,1]. Then the data values are split into test and train datasets.

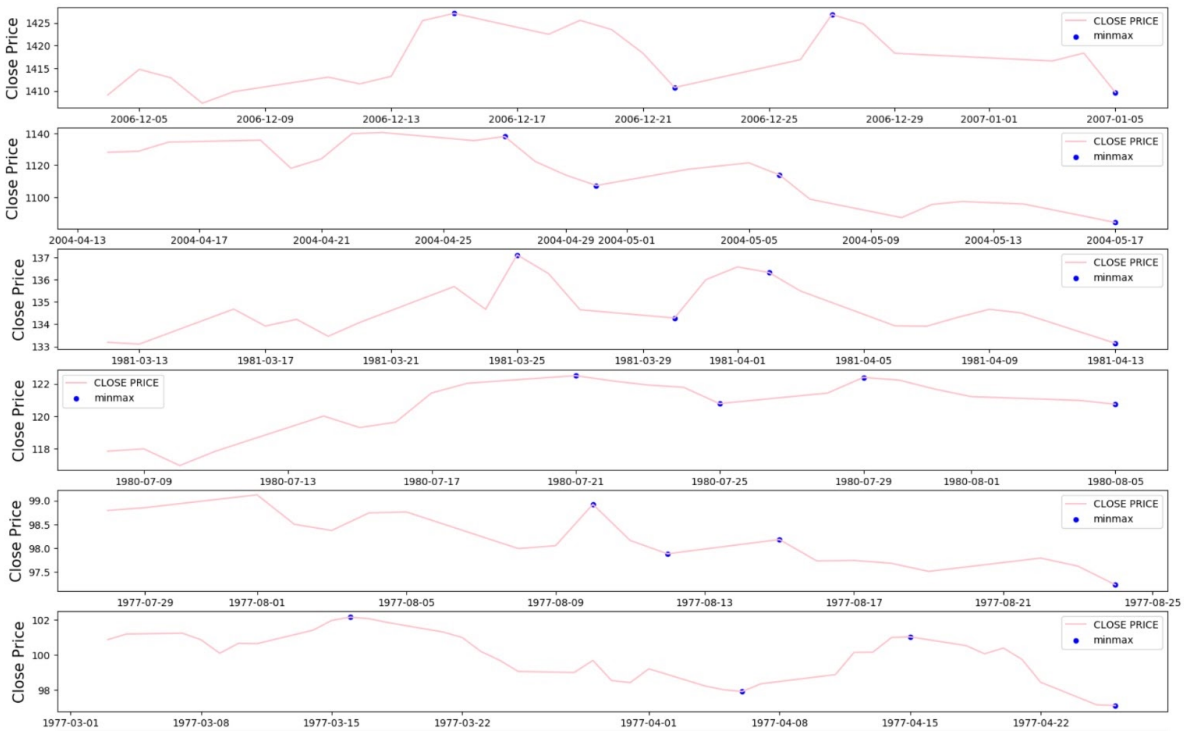


INPUT FOR CNN




Close Price
CLOSE PRICE
minmax
1965-03-01
1965-03-05
1965-03-09
1965-03-13
1965-03-17
1965-03-21
1965-03-25
1965-03-29
1964-07-15
1964-07-22
1964-08-01
1964-08-08
1964-08-15
1964-08-22
1962-04-21
1962-04-25
1962-04-29
1962-05-01
1962-05-05
1962-05-09
1962-05-13
1962-05-17
1962-05-21
1962-04-05
1962-04-09
1962-04-13
1962-04-17
1962-04-21
1962-04-25
1962-04-29
1962-05-01
1962-05-05
1962-05-09
1962-02-01
1962-02-05
1962-02-09
1962-02-13
1962-02-17
1962-02-21
1962-02-25
1962-03-01
1962-03-05
1961-12-21
1961-12-25
1961-12-29
1962-01-01
1962-01-05
1962-01-09
1962-01-13
1962-01-17
1962-01-21
1961-11-29
1961-12-01
1961-12-05
1961-12-09
1961-12-13
1961-12-17
1961-12-21
1961-12-25
1961-12-29
1962-01-01
1962-01-05


Close Price
CLOSE PRICE
minmax
1977-01-15
1977-01-22
1977-02-01
1977-02-08
1977-02-15
1977-02-22
1974-06-15
1974-06-22
1974-07-01
1974-07-08
1974-07-15
1974-07-22
1974-08-01
1972-02-25
1972-03-01
1972-03-05
1972-03-09
1972-03-13
1972-03-17
1972-03-21
1968-05-29
1968-06-01
1968-06-05
1968-06-09
1968-06-13
1968-06-17
1968-06-21
1968-06-25
1968-06-29
1968-07-01
1968-01-22
1968-02-01
1968-02-08
1968-02-15
1968-02-22
1968-03-01
1967-09-13
1967-09-17
1967-09-21
1967-09-25
1967-09-29
1967-10-01
1967-10-05
1967-10-09
1967-10-13
1966-11-29
1966-12-01
1966-12-05
1966-12-09
1966-12-13
1966-12-17
1966-12-21
1966-12-25
1966-12-29


Close Price
CLOSE PRICE
minmax
2006-12-05
2006-12-09
2006-12-13
2006-12-17
2006-12-21
2006-12-25
2006-12-29
2007-01-01
2007-01-05
2004-04-13
2004-04-17
2004-04-21
2004-04-25
2004-04-29
2004-05-01
2004-05-05
2004-05-09
2004-05-13
2004-05-17
1981-03-13
1981-03-17
1981-03-21
1981-03-25
1981-03-29
1981-04-01
1981-04-05
1981-04-09
1981-04-13
1980-07-09
1980-07-13
1980-07-17
1980-07-21
1980-07-25
1980-07-29
1980-08-01
1980-08-05
1977-07-29
1977-08-01
1977-08-05
1977-08-09
1977-08-13
1977-08-17
1977-08-21
1977-08-25
1977-03-01
1977-03-08
1977-03-15
1977-03-22
1977-04-01
1977-04-08
1977-04-15
1977-04-22

2. NETWORK PARAMETERS

FOR RNN

1. **Input :** generated dataset
2. **Predicted Output feature:** predicted ‘Close’ price
3. **Activation function:** sigmoid, tanh
4. **Epochs**:50
5. **Optimizer:** Adam
6. **Evaluation metrics:** prediction error, accuracy
7. **Number of timesteps:** 180
8. **Loss metrics:** Mean squared error

FOR CNN

1. **Input :** OHLC chart (Open-High-Low-Close)
2. **Predicted Output feature:** A-B-C-D curve
3. **Activation function:** Relu, Leaky Relu
4. **Optimizer:** Adam
5. **Evaluation metrics:** prediction error, accuracy
6. **Loss metrics:** Mean squared error
7. **Epochs:** 50

Initially, the model is trained on the CNN with input as the OHLC graphs and following that, the output from the CNN is used as input to the RNN which then makes predictions on the price values. The model is trained on the entire data set from 1960 to 2014 and tested on data from 2015 to 2020. The RNN built is a regression model that predicts the future Stock Close Price. The resulting output of the predicted price is then compared to the original price values to evaluate the model.

The optimized RNN+CNN model has higher accuracy compared to the unoptimized RNN+CNN. Adam optimizer was employed along with ReLU as the activation function. Once these modifications were added, the optimized CNN+RNN performed much better.

3. RNN Model

The architecture employed to deploy this model is the **LSTM model**. ***LSTM*** stands for Long Short Term Memory. Long Short-Term Memory (LSTM) networks are a ***modified version*** of recurrent neural networks, which makes it easier to remember past data in memory. The ***vanishing gradient*** problem of RNN is resolved here. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. It trains the model by using back-propagation. In an LSTM network,**three gates** are present:

i. Input gate — discover which value from input should be used to modify the memory. **Sigmoid** function decides which values to let through **0,1.** and **tanh** function gives weightage to the values which are passed deciding their level of importance ranging from**-1** to **1**

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

ii. Forget gate — discover what details to be discarded from the block. It is decided by the **sigmoid function.** it looks at the previous state(**ht-1**) and the content input(**Xt**) and outputs a number between **0(***omit this*)and **1(***keep this***)**for each number in the cell state **Ct−1**.

$$f_t = \sigma\left(W_f \cdot \left[h_{t-1}, x_t\right] \; + \; b_f\right)$$

iii. Output gate — the input and the memory of the block is used to decide the output. **Sigmoid** function decides which values to let through **0,1.** and **tanh** function gives weightage to the values which are passed deciding their level of importance ranging from**-1** to **1** and multiplied with output of **Sigmoid.**

$$o_t = \sigma\left(W_o \; [\, h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

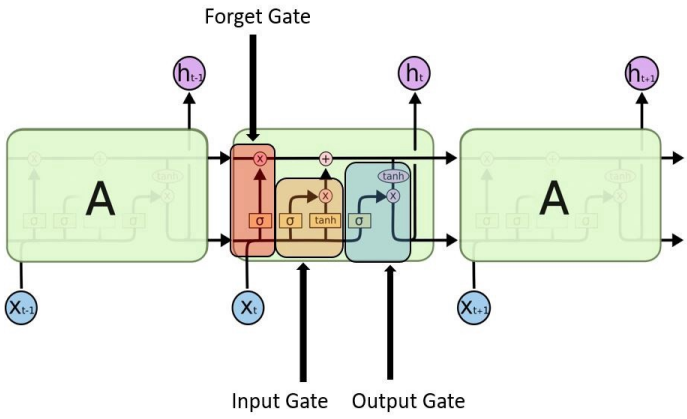
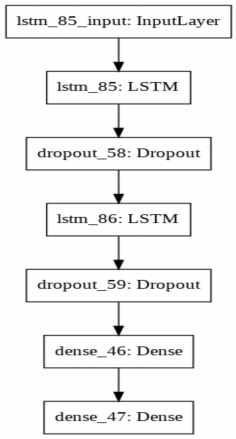


Figure 4: Sample LSTM architecture

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_12 (LSTM)               (None, 180, 30)           4080
_________________________________________________________________
lstm_13 (LSTM)               (None, 180, 30)           7320
_________________________________________________________________
lstm_14 (LSTM)               (None, 30)                7320
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 31
=================================================================
Total params: 18,751
Trainable params: 18,751
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: LSTM architecture and Details



3.1 LOSS METRIC

The model uses ***mean squared error*** as a loss metric. Mean squared error is used to tell how close a regression line is to a set of points. It also gives more weight to larger differences.

3.2 ACTIVATION FUNCTION

Sigmoid Function:

Since every neuron must predict a value within the range of [0,1], we chose sigmoid activation function. The sigmoid function can be represented as:

$$f(x) = \frac{1}{1 + e^{-x}}$$

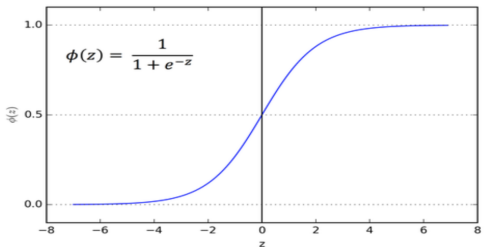


Figure 6: Sigmoid Activation function

Tanh Function:

Tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped).

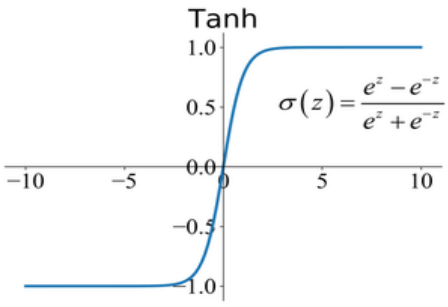


Figure 7: Tanh Activation function

3.3 OPTIMIZER

The model uses ***Adam*** optimizer.

- **Adam optimizer:**

  Adam can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using the moving average of the gradient.

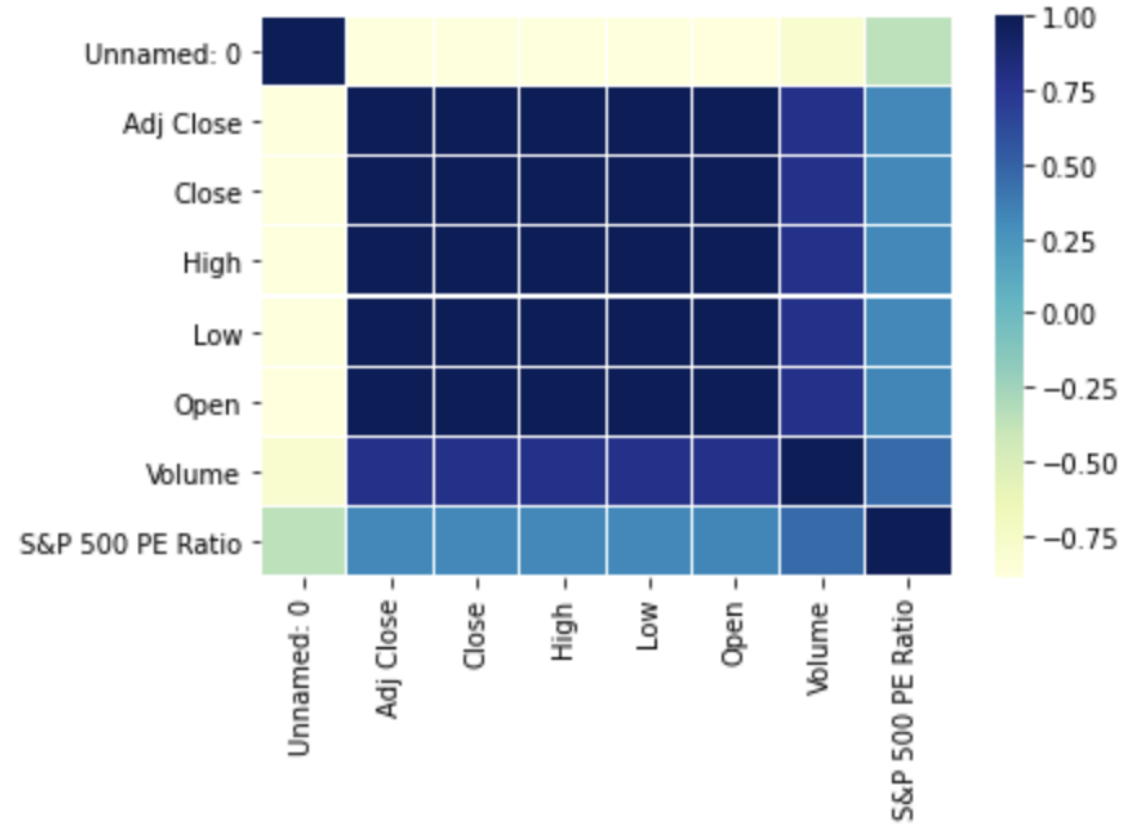
$$For\ each\ Parameter\ w^j$$
$$(j\ subscript\ dropped\ for\ clarity)$$
$$\nu_t = \beta_1 * \nu_{t-1} - (1 - \beta_1) * g_t$$
$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2$$
$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$
$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

$\eta : Initial\ Learning\ rate$
$g_t : Gradient\ at\ time\ t\ along\ \omega^j$
$\nu_t : Exponential\ Average\ of\ gradients\ along\ \omega_j$
$s_t : Exponential\ Average\ of\ squares\ of\ gradients\ along\ \omega_j$
$\beta_1, \beta_2 : Hyperparameters$

3.4 SELECTING INPUT FEATURES

The parameters were chosen based on the correlation between the columns in the data. It was observed that the correlation between Close and S&P 500 PE Ratio was high and they were highly preferred parameters.

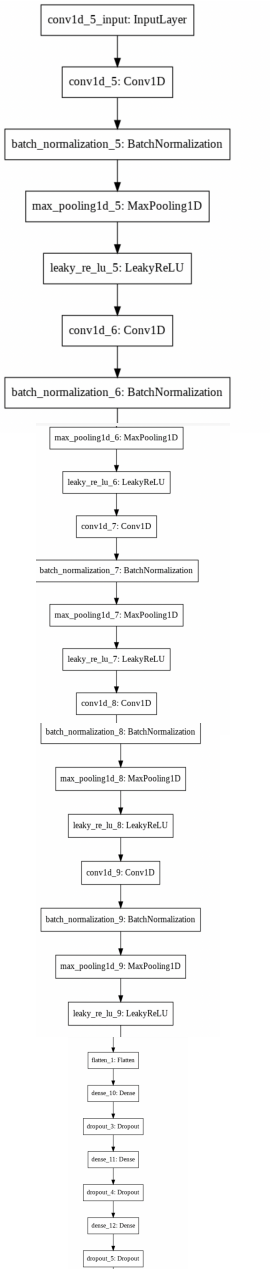
```
[8 rows x 8 columns]
```

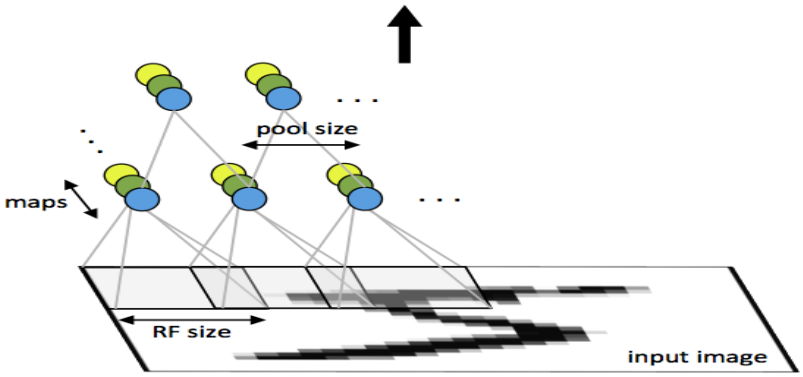


4. CNN Model

Convolutional neural networks are designed to recognize complex patterns and features in images. It works by dividing an image up into multiple overlapping perceptive fields and running a myriad of trainable filters through them, capturing basic features and patterns. This process is repeated several times, and as the filtered image is ran through more filters, deeper and more meaningful features are extracted and quantified. For example, to recognize an image of a car we might have several filters that are sensitive to wheels, or windows, or exhaust pipes, or licence plates and all of the results of these filters are gathered and quantified into a final classifier.

CNN works by stacking several filters on top of each other to form complex feature-sensitive filters; if stock data was treated as images, CNN can be applied to extract useful and deep information.





Layer (type)	Output Shape	Param #
conv1d_17 (Conv1D)	(None, 180, 64)	1024
batch_normalization_17 (Batc	(None, 180, 64)	256
max_pooling1d_17 (MaxPooling	(None, 90, 64)	0
leaky_re_lu_11 (LeakyReLU)	(None, 90, 64)	0
conv1d_18 (Conv1D)	(None, 90, 32)	6176
batch_normalization_18 (Batc	(None, 90, 32)	128
max_pooling1d_18 (MaxPooling	(None, 45, 32)	0
leaky_re_lu_12 (LeakyReLU)	(None, 45, 32)	0
conv1d_19 (Conv1D)	(None, 45, 16)	1552
batch_normalization_19 (Batc	(None, 45, 16)	64
max_pooling1d_19 (MaxPooling	(None, 22, 16)	0
leaky_re_lu_13 (LeakyReLU)	(None, 22, 16)	0
conv1d_20 (Conv1D)	(None, 22, 8)	392
batch_normalization_20 (Batc	(None, 22, 8)	32
max_pooling1d_20 (MaxPooling	(None, 11, 8)	0
leaky_re_lu_14 (LeakyReLU)	(None, 11, 8)	0
conv1d_21 (Conv1D)	(None, 11, 4)	100
batch_normalization_21 (Batc	(None, 11, 4)	16
max_pooling1d_21 (MaxPooling	(None, 5, 4)	0
leaky_re_lu_15 (LeakyReLU)	(None, 5, 4)	0
flatten_4 (Flatten)	(None, 20)	0
dense_13 (Dense)	(None, 8)	168
dropout_10 (Dropout)	(None, 8)	0
dense_14 (Dense)	(None, 4)	36
dropout_11 (Dropout)	(None, 4)	0
dense_15 (Dense)	(None, 16)	80
dropout_12 (Dropout)	(None, 16)	0
dense_16 (Dense)	(None, 2)	34

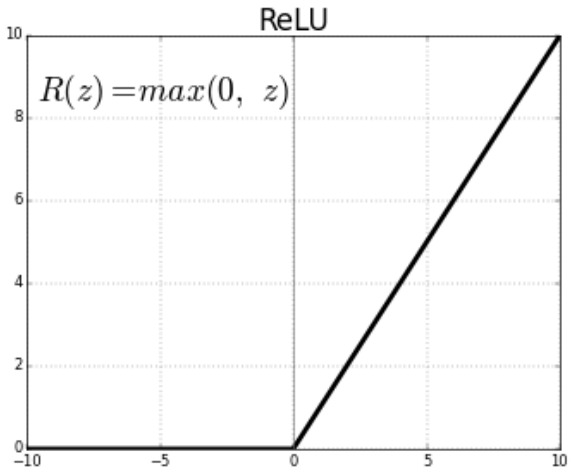
Figure 7:CNN Model architecture and Details

4.1 ACTIVATION FUNCTION

ReLU Function:

A ***Rectified Linear Unit*** (A unit employing the rectifier is also called a rectified linear unit ReLU) has output 0 if the input is less than 0, and *raw* output otherwise. That is, if the input is greater than 0, the output is equal to the input.

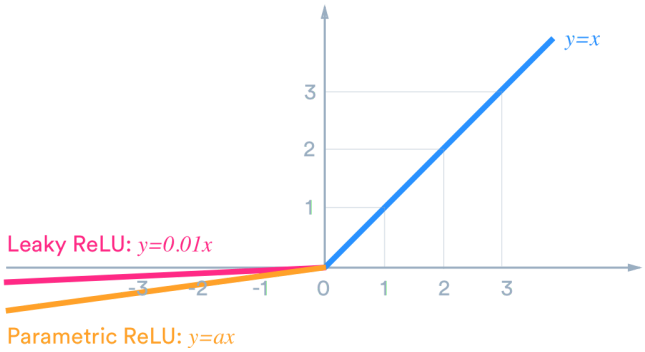
$$f(x) = max(x, 0)$$



Leaky ReLU Function:

Leaky ReLU has a small slope for negative values, instead of altogether zero. For example, leaky ReLU may have $y = 0.01x$ when $x < 0$

$$f(x) = \max(x, \alpha x) \qquad \text{with } \alpha \in (0, 1)$$



4.2 OPTIMIZER

The model uses ***Adam*** optimizer.

- **Adam optimizer:**

  Adam can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using the moving average of the gradient.

$$For\ each\ Parameter\ w^j$$
$$(j\ subscript\ dropped\ for\ clarity)$$
$$\nu_t = \beta_1 * \nu_{t-1} - (1 - \beta_1) * g_t$$
$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2$$
$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$
$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

$\eta : Initial\ Learning\ rate$
$g_t : Gradient\ at\ time\ t\ along\ \omega^j$
$\nu_t : Exponential\ Average\ of\ gradients\ along\ \omega_j$
$s_t : Exponential\ Average\ of\ squares\ of\ gradients\ along\ \omega_j$
$\beta_1, \beta_2 : Hyperparameters$

5. PYTHON CODE FOR RNN +CNN

Dataset Generation

The following code snippet depicts the conversion of the data obtained from the yahoo finance website into a dataframe using Pandas library.

```
 import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
data = pd.read_csv('./finalDS.csv',index_col=0)
data['Date'] = pd.to_datetime(data['Date'])
data.head()
```

	Adj Close	Close	Date	High	Low	Open	Volume	S&P 500 PE Ratio
0	3756.07	3756.07	2020-12-31	3760.20	3726.88	3733.27	3172510000	37.850000
1	3732.04	3732.04	2020-12-30	3744.63	3730.21	3736.19	3145200000	37.842333
2	3727.04	3727.04	2020-12-29	3756.12	3723.31	3750.01	3387030000	37.834667
3	3735.36	3735.36	2020-12-28	3740.51	3723.03	3723.03	3527460000	37.827000
4	3703.06	3703.06	2020-12-24	3703.82	3689.32	3694.03	1885090000	37.796333

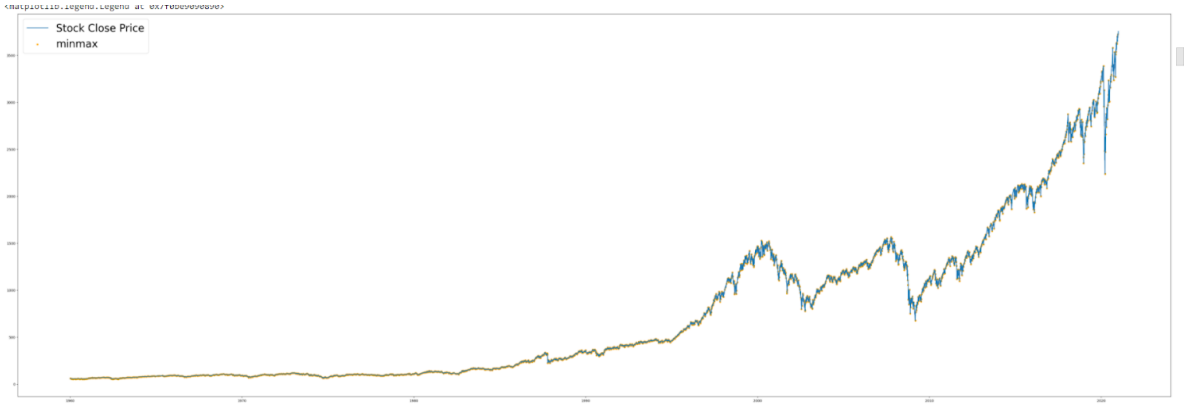
Correlation of stock data and Schiller P/E Ratio

CORRELATING STOCK DATA AND SHRILLER P/E RATIO TO RECOGNIZE BUY/SELL PATTERNS

```
def get_max_min(prices, smoothing, window_range):
    #print(prices.head())
    smooth_prices = prices['Close'].rolling(window=smoothing).mean()
    local_max = argrelextrema(smooth_prices.values, np.greater)[0]
    local_min = argrelextrema(smooth_prices.values, np.less)[0]
    price_local_max_dt = []
    for i in local_max:
        if (i>window_range) and (i<len(prices)-window_range):
            price_local_max_dt.append(prices.iloc[i-window_range:i+window_range]['Close'].idxmax())
    price_local_min_dt = []
    for i in local_min:
        if (i>window_range) and (i<len(prices)-window_range):
            price_local_min_dt.append(prices.iloc[i-window_range:i+window_range]['Close'].idxmin())
    maxima = pd.DataFrame(prices.loc[price_local_max_dt])
    minima = pd.DataFrame(prices.loc[price_local_min_dt])
    max_min = pd.concat([maxima, minima]).sort_index()
    max_min = max_min.reset_index()
    p = prices.reset_index()
    return max_min
```

```
[ ] smoothing = 3
    window = 10
    minmax = get_max_min(data[['Date','Open','High','Low','Close']], smoothing, window)
```

```
[ ] plt.figure(figsize=(60,20))
    plt.plot(data['Date'],data['Close'],linewidth=2,label='Stock Close Price')
    plt.scatter(minmax['Date'],minmax['Close'],color='orange',marker='o',label='minmax',s=20)
    plt.rc('xtick', labelsize=30)
    plt.rc('ytick', labelsize=30)
    plt.rc('legend', fontsize=30)
    plt.legend()
```



Model Initialization and training

In the code snippet below the model has been initialized with a Convolution 1D layer followed by max pooling , leaky ReLU , convolution 1D layers. The activation functions used are ReLu and Leaky ReLu. Batch Normalization has been employed as well.Adam optimizer has also been used.DropOut has also been implemented.

```
#MODEL ARCHITECTURE
with tpu_strategy.scope():
  model = Sequential()
  model.add(Conv1D(64, kernel_size=3,padding ='SAME',activation='relu', input_shape=input_shape))
  model.add(BatchNormalization())
  model.add(MaxPooling1D(pool_size=(2)))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Conv1D(32,kernel_size=3,padding = 'SAME', activation='relu'))
  model.add(BatchNormalization())
  model.add(MaxPooling1D(pool_size=(2)))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Conv1D(16,kernel_size=3,padding = 'SAME', activation='relu'))
  model.add(BatchNormalization())
  model.add(MaxPooling1D(pool_size=(2)))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Conv1D(8,kernel_size=3,padding = 'SAME', activation='relu'))
  model.add(BatchNormalization())
  model.add(MaxPooling1D(pool_size=(2)))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Conv1D(4,kernel_size=3,padding = 'SAME', activation='relu'))
  model.add(BatchNormalization())
  model.add(MaxPooling1D(pool_size=(2)))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Flatten())
  model.add(Dense(8, activation='relu'))
  model.add(Dropout(0.2))
  model.add(Dense(4, activation='relu'))
  model.add(Dropout(0.2))
  model.add(Dense(16, activation='relu'))
  model.add(Dropout(0.2))
  model.add(Dense(2, activation='softmax'))
  model.summary()

  model.compile(loss = 'categorical_crossentropy',
                optimizer='adam',
                metrics=['accuracy'])
```

```
model.fit(X_train,y_train,epochs=50,verbose=1)
```

```
Epoch 1/50
380/380 [==============================] - 14s 20ms/step - loss: 0.6560 - accuracy: 0.6667
Epoch 2/50
380/380 [==============================] - 6s 16ms/step - loss: 0.6026 - accuracy: 0.7135
Epoch 3/50
380/380 [==============================] - 6s 17ms/step - loss: 0.6023 - accuracy: 0.7086
Epoch 4/50
380/380 [==============================] - 6s 17ms/step - loss: 0.5890 - accuracy: 0.7185
Epoch 5/50
380/380 [==============================] - 6s 16ms/step - loss: 0.5928 - accuracy: 0.7124
Epoch 6/50
380/380 [==============================] - 6s 16ms/step - loss: 0.5865 - accuracy: 0.7143
Epoch 7/50
380/380 [==============================] - 6s 16ms/step - loss: 0.5834 - accuracy: 0.7129
Epoch 8/50
380/380 [==============================] - 6s 16ms/step - loss: 0.5797 - accuracy: 0.7160
Epoch 9/50
380/380 [==============================] - 6s 16ms/step - loss: 0.5765 - accuracy: 0.7130
Epoch 10/50
380/380 [==============================] - 6s 17ms/step - loss: 0.5666 - accuracy: 0.7196
Epoch 11/50
380/380 [==============================] - 6s 16ms/step - loss: 0.5655 - accuracy: 0.7206
Epoch 12/50
380/380 [==============================] - 6s 17ms/step - loss: 0.5650 - accuracy: 0.7132
Epoch 13/50
380/380 [==============================] - 6s 17ms/step - loss: 0.5603 - accuracy: 0.7217
Epoch 14/50
380/380 [==============================] - 6s 17ms/step - loss: 0.5465 - accuracy: 0.7235
Epoch 15/50
380/380 [==============================] - 6s 17ms/step - loss: 0.5539 - accuracy: 0.7175
Epoch 16/50
380/380 [==============================] - 6s 16ms/step - loss: 0.5479 - accuracy: 0.7151
Epoch 17/50
380/380 [==============================] - 7s 17ms/step - loss: 0.5440 - accuracy: 0.7121
```

Generating predictions

The predictions have been made on the test values. Following which, the score and accuracy are calculated. The precision and recall are also calculated.

```
#model predictions
y_pred = model.predict(X_test)
```

```
#MODEL EVALUATION

score = model.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

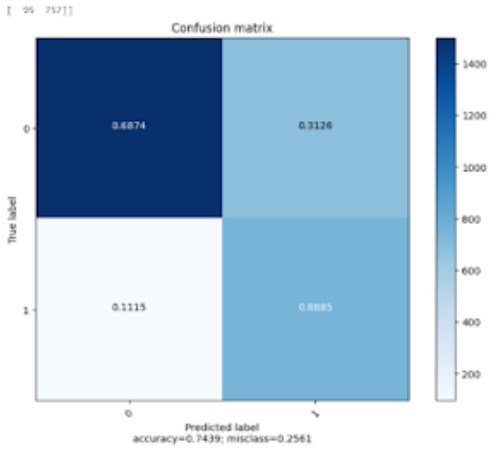
```
Test score: 0.4784053862094879
Test accuracy: 0.7923533320426941
```

```
y_pred = [np.argmax(y) for y in y_pred]
y_test = [np.argmax(y) for y in y_test]

precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')

print("Precision: ", precision)
print("Recall: ", recall)
```

```
Precision:  0.8030407014532911
Recall:  0.7923533289386948
```

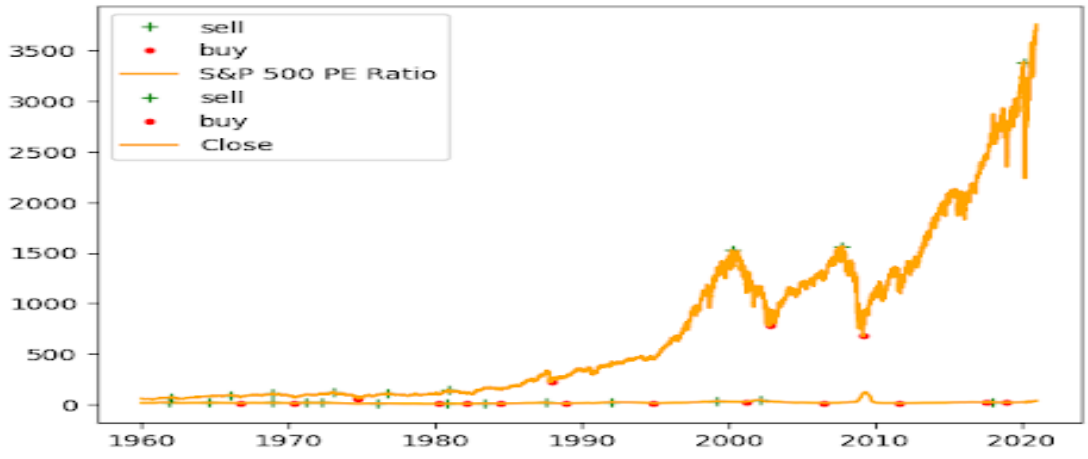
Calculation of confusion matrix

```
def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):

    import itertools

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")


    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(accuracy, misclass))
    plt.show()


labels = [0,1]
cm = confusion_matrix(y_test, y_pred,labels)
print(cm)
plot_confusion_matrix(cm,
                          [0,1],
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True)
```



Studying trends in stock close price with respect to Schiller P/E Ratio

```
def find_trends(data,colname):
  peaks2,_ = find_peaks(data[colname],width = 180)
  lows,_ = find_peaks(-data[colname],width = 180)
  #local maxima points
  peaks ={'date':[],'peak':[]}
  for p in peaks2:
    peaks['date'].append(data['Date'].loc[p])
    peaks['peak'].append(data[colname].loc[p])
  peak = pd.DataFrame(data=peaks)
  #local minima points
  lowp ={'date':[],'low':[]}
  for l in lows:
    lowp['date'].append(data['Date'].loc[l])
    lowp['low'].append(data[colname].loc[l])
  low = pd.DataFrame(data=lowp)
  #plotting graph
  plt.style.use('default')
  plt.plot(peak['date'],peak['peak'],'+',label='sell',color='green')
  plt.plot(low['date'],low['low'],'.',label = 'buy',color='red')
  plt.plot(data['Date'],data[colname],color='orange',label = colname)
  plt.legend()
```



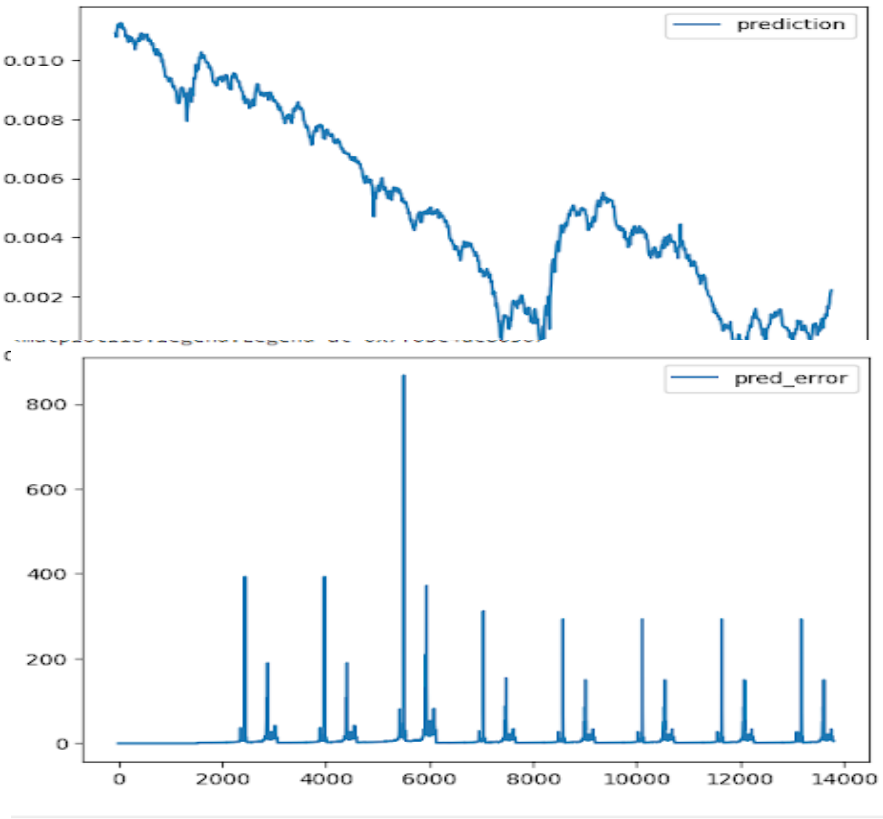
Calculation of performance metrics

The performance metrics used are prediction error and accuracy. The prediction error is calculated as :

error= prediction (predicted price - original price)/original price.

```
plt.plot(y_test,label='prediction')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f0bc4ca1dd0>
```



Introducing noise to input data

The input data are perturbed by adding additive gaussian noise. The prediction error has been tabulated in an excel file. As the table has a large dimension, only a portion of the **table** has been displayed in **models Output table for noisy data**

```
mu, sigma = 0, 0.001
noise = np.random.normal(mu, sigma, [15355,4])
print(noise)
signal = clean_signal + noise
print(signal)
np.savetxt("std0.001.csv", signal, delimiter=",")

mu, sigma = 0, 0.002
noise = np.random.normal(mu, sigma, [15355,4])
print(noise)
signal = clean_signal + noise
print(signal)
np.savetxt("std0.002.csv", signal, delimiter=",")

mu, sigma = 0, 0.003
noise = np.random.normal(mu, sigma, [15355,4])
print(noise)
signal = clean_signal + noise
print(signal)
np.savetxt("std0.003.csv", signal, delimiter=",")

mu, sigma = 0, 0.005
noise = np.random.normal(mu, sigma, [15355,4])
print(noise)
signal = clean_signal + noise
print(signal)
np.savetxt("std0.005.csv", signal, delimiter=",")

mu, sigma = 0, 0.01
noise = np.random.normal(mu, sigma, [15355,4])
print(noise)
signal = clean_signal + noise
print(signal)
np.savetxt("std0.01.csv", signal, delimiter=",")

mu, sigma = 0, 0.02
noise = np.random.normal(mu, sigma, [15355,4])
print(noise)
signal = clean_signal + noise
print(signal)
np.savetxt("std0.02.csv", signal, delimiter=",")
```

```
mu, sigma = 0, 0.03
noise = np.random.normal(mu, sigma, [15355,4])
print(noise)
signal = clean_signal + noise
print(signal)
np.savetxt("std0.03.csv", signal, delimiter=",")

mu, sigma = 0, 0.05
noise = np.random.normal(mu, sigma, [15355,4])
print(noise)
signal = clean_signal + noise
print(signal)
np.savetxt("std0.05.csv", signal, delimiter=",")

mu, sigma = 0, 0.1
noise = np.random.normal(mu, sigma, [15355,4])
print(noise)
signal = clean_signal + noise
print(signal)
np.savetxt("std0.1.csv", signal, delimiter=",")
```

6.OBSERVATION OF THE TRADING ACTIVITY FOR $1 MILLION DOLLARS

The data below indicates the trading activities with an initial principal amount of 1 million dollars. This is a segment of the data. The entire information is displayed in the csv file titled ***Investment_activity.csv.*** Investment activity trained along with a sell signal and RNN tells us when is the best time to buy and sell stocks.

Table of trading activity assuming $1 million startup funds

```
Activity = buy_stock(data.Close, labels)
Activity.to_csv('Investment_activity.csv', index=False)
Activity.head(50)
```

	DAY	TOTAL BALANCE	STATUS
0	0	996243.93	Bought 1 units for price $3756.07
1	1	992511.89	Bought 1 units for price $3732.04
2	2	988784.85	Bought 1 units for price $3727.04
3	3	985049.49	Bought 1 units for price $3735.36
4	4	981346.43	Bought 1 units for price $3703.06
5	5	977656.42	Bought 1 units for price $3690.01
6	6	973969.16	Bought 1 units for price $3687.26
7	7	970274.24	Bought 1 units for price $3694.92
8	8	966564.83	Bought 1 units for price $3709.41
9	9	962842.35	Bought 1 units for price $3722.48
10	10	959141.18	Bought 1 units for price $3701.17
11	11	955446.56	Bought 1 units for price $3694.62
12	12	951799.07	Bought 1 units for price $3647.49
13	13	948135.61	Bought 1 units for price $3663.46
14	14	944467.51	Bought 1 units for price $3668.1
15	15	940794.69	Bought 1 units for price $3672.82
16	16	937092.44	Bought 1 units for price $3702.25
17	17	933400.48	Bought 1 units for price $3691.96
18	18	929701.36	Bought 1 units for price $3699.12
19	19	926034.64	Bought 1 units for price $3666.72
20	20	922365.63	Bought 1 units for price $3669.01
21	21	918703.18	Bought 1 units for price $3662.45
22	22	915081.55	Bought 1 units for price $3621.63
23	23	911443.20	Bought 1 units for price $3638.35
24	24	907813.55	Bought 1 units for price $3629.65
25	25	904178.14	Bought 1 units for price $3635.41
26	26	900600.55	Bought 1 units for price $3577.59

7.OPTIMIZED MODEL OUTPUT FOR NOISY DATA

This is a portion of the table depicting prediction error for the standard deviation 0.01. There is an individual table for each standard deviation to maintain legibility and better understanding of the data.

Similarly, CSV tables have been created for the other standard deviations as well.

OPEN	HIGH	LOW	CLOSE
9.69E-01	9.73E-01	1.21E+00	8.94E-01
9.55E-01	1.05E+00	9.90E-01	1.05E+00
9.16E-01	1.01E+00	7.76E-01	9.80E-01
9.73E-01	8.82E-01	1.01E+00	9.20E-01
9.62E-01	9.04E-01	9.25E-01	1.02E+00
1.12E+00	8.36E-01	8.80E-01	1.06E+00
9.49E-01	1.03E+00	1.03E+00	9.55E-01
8.60E-01	1.10E+00	1.17E+00	1.24E+00
9.40E-01	1.07E+00	9.70E-01	9.61E-01
1.06E+00	1.03E+00	9.38E-01	1.05E+00
1.01E+00	8.37E-01	9.24E-01	1.12E+00
7.46E-01	9.30E-01	1.01E+00	9.34E-01
9.22E-01	8.39E-01	1.01E+00	9.39E-01
9.74E-01	9.89E-01	9.49E-01	8.44E-01
9.65E-01	7.41E-01	1.06E+00	1.03E+00
1.01E+00	1.09E+00	9.58E-01	9.29E-01
7.67E-01	9.89E-01	9.78E-01	1.00E+00
9.30E-01	8.32E-01	1.14E+00	8.82E-01
8.57E-01	9.38E-01	1.08E+00	8.11E-01
1.11E+00	9.96E-01	9.20E-01	8.67E-01
9.40E-01	1.04E+00	1.02E+00	8.91E-01
1.11E+00	8.29E-01	9.81E-01	8.58E-01
1.06E+00	1.02E+00	1.06E+00	1.00E+00

8. PERFORMANCE EVALUATION FOR THE MODEL

An attempt was made using the ReLu activation function but the performance of the model was very low. So, later using LeakyReLu activation function, the performance of the model has increased drastically. The LeakyReLu activation function helps in preventing vanishing gradients.Furthermore, mean activation close to zero makes the training process faster.

9. DISCUSSION

Market sentiment refers to the overall attitude of investors toward a particular security or **financial market**. It is the feeling or tone of a market, or its crowd psychology, as revealed through the activity and price movement of the securities traded in that market. In broad terms, rising prices indicate **bullish** market sentiment, while falling prices indicate **bearish** market sentiment. The Market sentiment has been employed in the model to predict the buying and selling trends.

CNN gives selling probability and this is given as input to RNN to enhance its performance.

Time series forecasting is about estimating the future value of a time series on the basis of past data. Many time series problems can be solved by looking at a single step in the future. Multi-step time series prediction models the distribution of future values of a signal over a prediction horizon. This approach predicts multiple output values at the same time which is forecasting approach to predict the further course of gradually rising sine wave. In addition, many of these variables are interdependent, which makes statistical modeling even are more complex.

Multivariate RNN model is implemented using the output of CNN, adjacent close price, Shiller P/E ratio because Shiller P/E ratio, adjacent close price are highly correlated. The selling probability given as input to RNN gives more productive information and enhances the predictions of RNN.

Optimized CNN+RNN enhances the performance of unoptimized CNN+RNN by using LeakyReLu activation function and adam optimizer.