Configuration Manual

MSc Research Project
MSc. Data Analytics

Neil Sahay
Student ID: x19238061

School of Computing
National College of Ireland

Supervisor: Martin Alain

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Neil Sahay		
Student ID:	x19238061		
Programme:	MSc. Data Analytics	**Year:**	2021/22
Module:	Research Project		
Lecturer:	Martin Alain		
Submission Due Date:	31/01/2022		
Project Title:	Fake News Detection Using Deep Learning and Computational Linguistics		
Word Count:	1782	**Page Count**:	15

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Neil Sahay
Date:	30/01/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Neil Sahay
x19238061

1 Introduction

This configuration manual contains detailed information about the different hardware and software configurations that were required to set up the project build-up, and also the crucial setup and libraries that needed to be imported and installed while setting up the environment. As the code is executed in Google Collaboratory, thus this manual also contains detailed information about the setup process. Moreover, it also contains the code snippets and executed models architectures of the three experiments performed.

2 Hardware and Software Configurations

Due to the sheer influx of data and usage of extensive memory, Google colaboratory is utilized as it has an inbuilt Graphical Processing Unit (GPU), and memory of 13 GB, which makes the computation faster as compared to the normal processor. Table 1 and 2 shows the tabular structure in which all the information regarding software and hardware configurations is encapsulated.

Table 1: Hardware Configurations Adopted

Host Machine	HPE EliteBook with i5 Processor
RAM	8 GB
GPU	Google Colaboratory integrated GPU with 80 GB free storage and 13 GB RAM

Table 2: Software Employed

Programming language	Python
Cloud environment	Google Collaboratory
Browser	Google chrome

3 Colaboratory Setup

This section contains detailed steps for configuring the Google Collaboratory so as to import the data efficiently and process the data while also performing computational algorithms to build the model in scalable manner.

1. Initial step is to configure the google collboratory file by going into the file section and clicking on New Notebook option as shown in Figure 1.

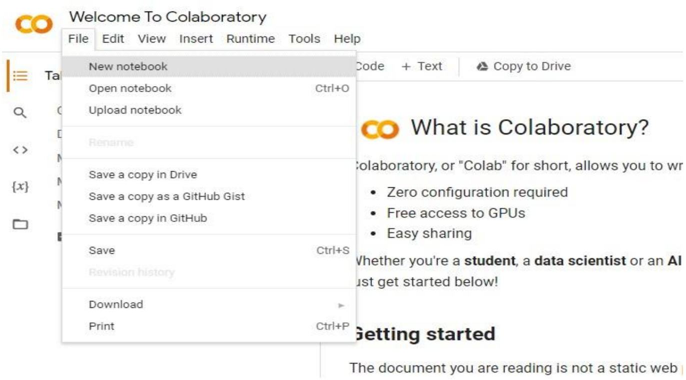


Figure 1: Creating New Notebook in Google Colab

2. The second step is to change the Runtime to the integrated GPU, which Google Colab provides as shown in Figure 2.

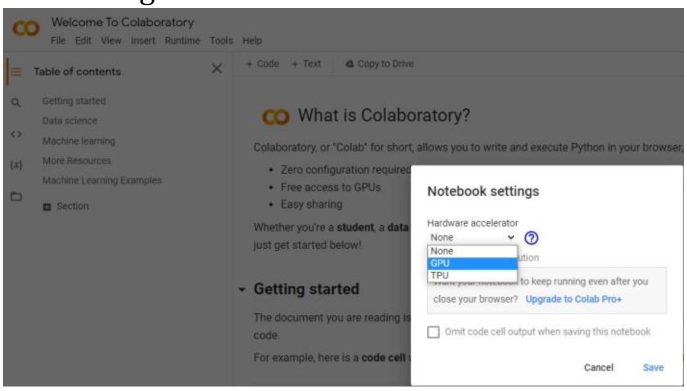


Figure 2: Changing the runtime to GPU

3. The third step involves uploading the dataset in Colab environment to perform operations.

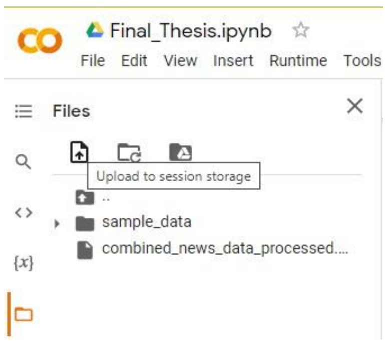


Figure 3: Uploading the input data in colab

4. The next step involves installing the required libraries. In this project, libraries such as spacy which is an advanced NLP toolkit is used, also the library en_core _web_md consists of components such as Lemmatizer and Tokenizer, which is also downloaded using pip command as shown in figure 4.

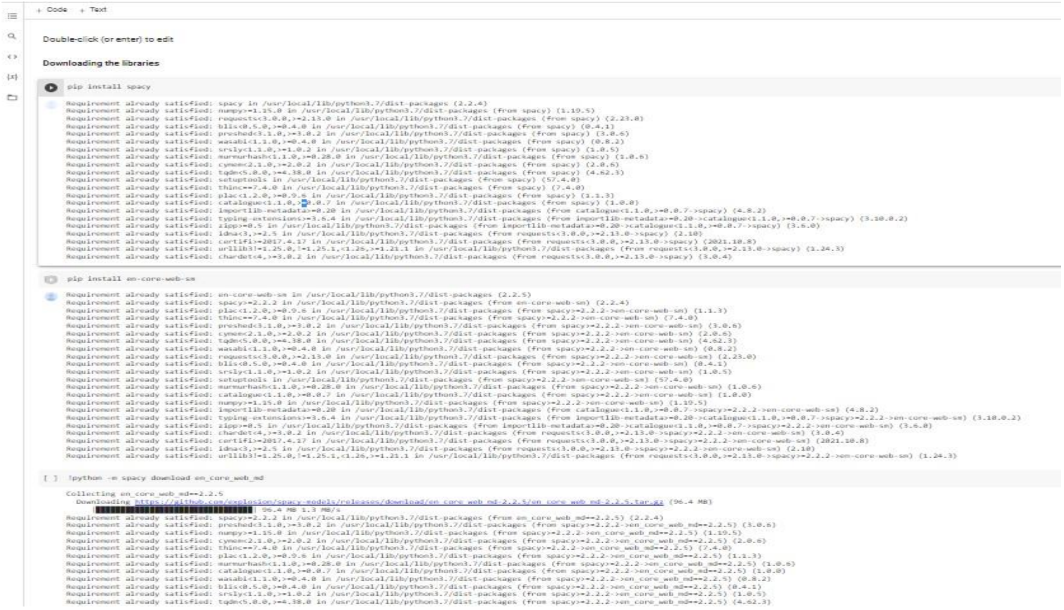


Figure 4: Installing the libraries

4 Data Pre-processing

This section contains steps for data pre-processing, and also include code snippets in the way, how the data is imported and cleaned.

1. The first step involves importing the data within Google Colab., by clicking Files section as shown in Fig 5.

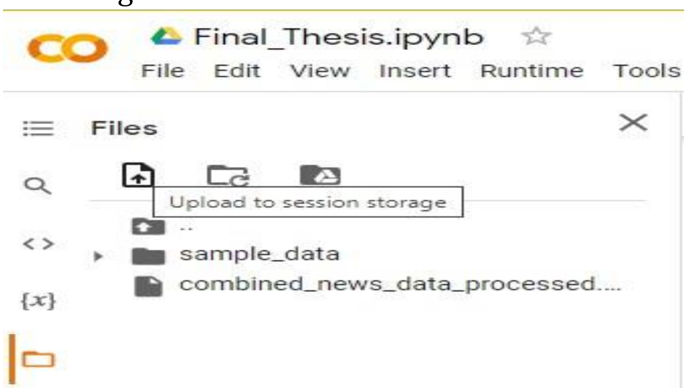


Figure 5: Uploading the dataset into the Colab environment

2. Next the data is imported and read into the system using pandas library, with command pd.read _csv("File Name") as shown in figure 6.

Reading The Data

```
data = pd.read_csv('/content/combined_news_data_processed.csv')
data.dropna(inplace=True)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 74011 entries, 0 to 74011
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   title   74011 non-null  object
 1   text    74011 non-null  object
 2   label   74011 non-null  int64
dtypes: int64(1), object(2)
memory usage: 2.3+ MB
```

Figure 6: Reading the data

3. The data is then needs to be pre-processed i.e. special characters and stop words are removed, moreover lemmatization is also performed, which is as shown in Fig 7.

Pre-processing text

```
import re # regex library
import en_core_web_md
from spacy.lang.en.stop_words import STOP_WORDS

nlp = en_core_web_md.load()

def preprocessor(text):
    text = re.sub('<[^>]*>', '', text) # Effectively removes HTML markup tags
    emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)', text)
    text = re.sub('[\W]+', ' ', text.lower()) + ' '.join(emoticons).replace('-', '')
    doc = nlp(text)
    text = ' '.join([token.lemma_ for token in doc if token.text not in STOP_WORDS])
    return text
```

Figure 7: Pre-processing the data

4. The data is then splitted into training and test data sets using the scikit learn library and under the component mode selection, train test split is imported as shown in Figure 8.

```
X = data['text']
y = data['label']
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

data_processed = pd.DataFrame({'title': data['title'], 'text': X, 'label': y})
data_processed.to_csv('/content/combined_news_data_processed.csv', index=False)
```

+ Code + Text

Figure 8: Splitting the data

5 Defining the Model

This section gives detailed information regarding the defining and training of the implemented Recurrent Convolutional Neural Network (RCNN) plus Long short term memory (LSTM) model. The specific libraries are imported and tensorflow is invoked to implement the desired model.

1. The initial step involves, importing all the necessary libraries from keras module, also including tensorflow.

Defining and Training the Model

```
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.metrics import accuracy_score
import tensorflow as tf
from keras.preprocessing.text import hashing_trick
from keras.preprocessing.text import text_to_word_sequence
from keras.models import Sequential, load_model
from keras.layers import Embedding, LSTM, Dense, Conv1D, MaxPooling1D, Dropout
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.callbacks import EarlyStopping, ModelCheckpoint
```

Figure 9: Importing the libraries

2. After importing all the libraries, I created a class with name LSTM Text Classifier.

3. Once the class is declared, initialization of variables is done, wherein all the default parameter values are also set as shown in Figure 10.

```
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.metrics import accuracy_score
import tensorflow as tf
from keras.preprocessing.text import hashing_trick
from keras.preprocessing.text import text_to_word_sequence
from keras.models import Sequential, load_model
from keras.layers import Embedding, LSTM, Dense, Conv1D, MaxPooling1D, Dropout
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.callbacks import EarlyStopping, ModelCheckpoint

class LSTM_Text_Classifier(BaseEstimator, ClassifierMixin):

    def __init__(self, embedding_vector_length, max_seq_length, lstm_layers, batch_size=32, num_epochs=3, use_hash=False,
                 dropout=None, conv_params=None):

        self.embedding_vector_length = embedding_vector_length
        self.max_seq_length = max_seq_length
        self.lstm_layer_sizes = lstm_layers
        self.num_epochs = num_epochs
        self.batch_size = batch_size
        self.use_hashing_trick = use_hash
        if not self.use_hashing_trick:
            self.tokenizer = Tokenizer()
        self.dropout = dropout
        self.conv_params = conv_params
```

Figure 10: Initializing the variables in base class

4. Two methods are declared wherein the first method i.e. _get word index which is used to retrieve the word token, while text _to int sequence is used to get the word sequence for all the new words within the text as shown in figure 11.

```
def _get_word_index(self, word):

    try:
        return self.tokenizer.word_index[word]
    except:
        return None

def _text_to_int_sequence(self, text):
    seq = [self._get_word_index(word) for word in text_to_word_sequence(text)]
    return [index for index in seq if index]
```

Figure 11: Function to retrieve word index and sequence

5. In the next step, another method ”fit” is created , in which parameters are passed such as Training data i.e. x train and y train along with validation data. This method is also used to compile the model and with the help of this method, the entire structure will be executed.

```
def fit(self,X, y, validation_data):
    all_X = pd.concat([X, validation_data[0]])
    if self.use_hashing_trick:
        all_words = set()
        for text in all_X:
            new_words = set(text_to_word_sequence(text))
            all_words = all_words.union(new_words)
        self.max_vocab = len(all_words)*1.3

        for i in range(len(X)):
            X[i] = hashing_trick(X[i], max_vocab, hash_function='md5')
        X_pad = sequence.pad_sequences(X, maxlen=self.max_seq_length)

        X_valid = validation_data[0]

        for i in range(len(X_valid)):
            X_valid[i] = hashing_trick(X_valid[i], max_vocab, hash_function='md5')
        X_valid_pad = sequence.pad_sequences(X_valid, maxlen=self.max_seq_length)

        y_valid = validation_data[1]

    else:
        print('Fitting Tokenizer...')
        self.tokenizer.fit_on_texts(all_X)
        self.max_vocab = len(self.tokenizer.word_index) + 20
        X = X.apply(self._text_to_int_sequence)
        X_pad = sequence.pad_sequences(X, maxlen=self.max_seq_length)

        X_valid = validation_data[0].apply(self._text_to_int_sequence)
        X_valid_pad = sequence.pad_sequences(X_valid, maxlen=self.max_seq_length)

        y_valid = validation_data[1]

    self.model = Sequential()
```

Figure 12: Fit method declaration

6. The model layers are then added, starting initially by adding the sequential layer, along with Convolutional 1D layer, wherein kernel size along with activation function is declared. Also, LSTM layer with sigmoid function, CNN layer with ReLU activation, adam optimizer and callbacks are all defined in the fit method as shown in figure 13.

```
        for i in range(self.conv_params['n_layers']):
            self.model.add(Conv1D(filters=2*(i+1)*self.conv_params['filters'],
                                  kernel_size=self.conv_params['kernel_size'],
                                  padding='same', activation='relu'))
            if use_pooling:
                self.model.add(MaxPooling1D(pool_size=self.conv_params['pool_size']))


    if len(self.lstm_layer_sizes) > 1:
        for lstm_layer_size in self.lstm_layer_sizes[:-1]:
            self.model.add(LSTM(lstm_layer_size, return_sequences=True))
            self.model.add(Dropout(self.dropout))
        self.model.add(LSTM(self.lstm_layer_sizes[-1]))
    else:
        self.model.add(LSTM(self.lstm_layer_sizes[0]))
    if self.dropout is not None:
        self.model.add(Dropout(self.dropout))
    self.model.add(Dense(1, activation='sigmoid'))
    self.model.compile(loss='binary_crossentropy', optimizer='adam', metrics=["accuracy","Recall",tf.keras.metrics.Precision()])
    early_stopping = EarlyStopping(monitor='val_accuracy',
                          min_delta=0,
                          patience=1,
                          verbose=2, mode='max')

    checkpoint = ModelCheckpoint(filepath='best_model',
                                 monitor='val_accuracy',
                                 mode='max',
                                 save_best_only=True)

    callbacks_list = [early_stopping, checkpoint]
    print(self.model.summary())
```

Figure 13: Adding the layers

7. In the next step, predict method to predict Fake and Real news on test data is created along with predict classes method as shown in Figure 14., which will aid to judge the different class, which is assigned for fake and real news that is 0 and 1.

```
def predict(self, X):

    if type(X) == pd.core.frame.DataFrame or type(X) == pd.core.series.Series:
        X = X.apply(self._text_to_int_sequence)
        X = sequence.pad_sequences(X, maxlen = self.max_seq_length)
        return self.model.predict(X)
    elif type(X) == str:
        X = self._text_to_int_sequence(X)
        X = sequence.pad_sequences(X, maxlen = self.max_seq_length)
        return self.model.predict(X)
    else:
        X = map(X, self._text_to_int_sequence)
        X = sequence.pad_sequences(X, maxlen = self.max_seq_length)
        return self.model.predict(X)

def predict_classes(self, X):

    if type(X) == pd.core.frame.DataFrame or type(X) == pd.core.series.Series:
        X = X.apply(self._text_to_int_sequence)
        X = sequence.pad_sequences(X, maxlen = self.max_seq_length)
        return self.model.predict_classes(X)
    elif type(X) == str:
        X = self._text_to_int_sequence(X)
        X = sequence.pad_sequences(X, maxlen = self.max_seq_length)
        return self.model.predict_classes(np.array(X))
    else:
        X = map(X, self._text_to_int_sequence)
        X = sequence.pad_sequences(X, maxlen = self.max_seq_length)
        return self.model.predict_classes(np.array(X))
```

Figure 14: Method declaration to predict classes and documents

8. Once all the necessary classes and methods are declared, then finally two more methods are created for loading the best model, along with retrieving the accuracy score of the implemented model as shown in figure 15.

```python
def load_model(self, file_path):

    self.model = load_model(file_path)

def score(self, X, y):

    pred = self.predict(X)
    return accuracy_score(y, pred)
```

Figure 15: Data Distribution of Length of the articles

6 Experimentation and Evaluation of the Models

Once the model buildup is completed, thus in the next phase 3 different models are built by adjusting the hyper-parameters in each of the experiment. This section contains execution of the 3 experiments performed and also details about the architecture obtained for each experiment performed.

6.1 Experiment 1.

In the first experiment, the python notebook is articulated with variation in hyper parameters along with code structure as discussed in the steps below:

1. The model is trained by putting in the initial values of the hyper-parameters such as number of LSTM layers set to 100, while the number of CNN layers has been set to 3, along with the dropout rate set to 0.1 in the function ”LSTM _Text Classifier” ,as depicted in table 3. Moreover figure 16, shows the model architecture.

Table 3: Executed Results for Experiment 1.

LSTM neurons	CNN layers	Dropout	Number of epochs
100	3	0.1	3,5,10

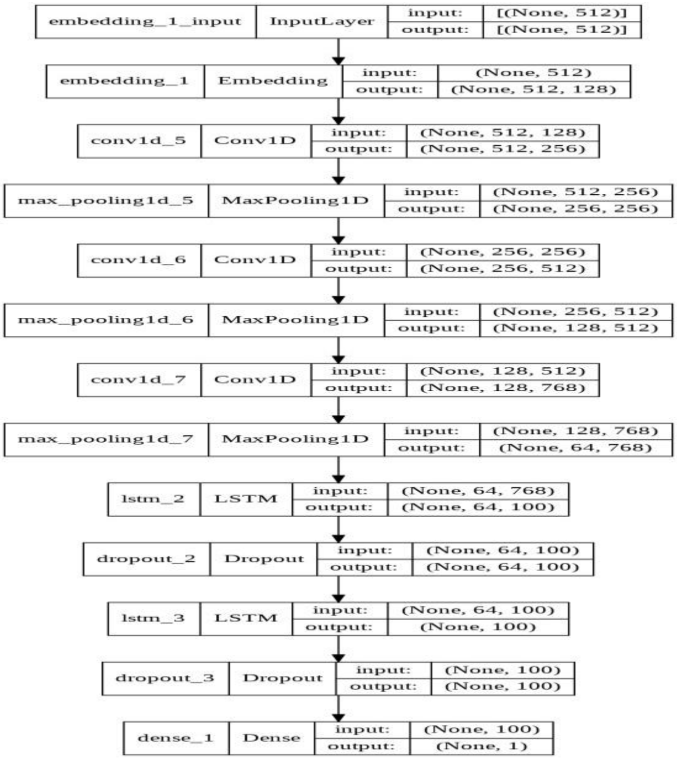


Figure 16: Experiment 1. Model architecture

2. In the second stage of experiment , now as the model parameters are set in place, thus in this step the model is compiled and executed using the function ”fit”, under which the model is executed as shown in figure 17.

```
In [19]: lstm_classifier = LSTM_Text_Classifier(embedding_vector_length=128, max_seq_length=512, dropout=0.1,

                                                lstm_layers=[100, 100], batch_size=256, num_epochs=10, use_hash=False,

                                                conv_params={'filters': 128,'kernel_size': 5, 'pool_size': 2,'n_layers': 3})

         lstm_classifier.fit(X_train, y_train, validation_data=(X_valid, y_valid))

Epoch 1/10
37/37 [==============================] - 64s 1s/step - loss: 0.8141 - accuracy: 0.5173 - val_loss: 0.6821 - val_accuracy: 0.559
6
Epoch 2/10
37/37 [==============================] - 31s 845ms/step - loss: 0.6964 - accuracy: 0.5286 - val_loss: 0.6922 - val_accuracy: 0.
5414
Epoch 3/10
37/37 [==============================] - 31s 835ms/step - loss: 0.6920 - accuracy: 0.5364 - val_loss: 0.6913 - val_accuracy: 0.
5414
Epoch 4/10
37/37 [==============================] - 31s 835ms/step - loss: 0.6913 - accuracy: 0.5364 - val_loss: 0.6907 - val_accuracy: 0.
5414
```

Figure 17: Experiment 1. Compiling and Fitting the model

3. Now as the model has been executed, thus the trained model is then validated over the test set dataset and the results are obtained as a classification report which is shown in figure 18.

```
In [21]: from sklearn.metrics import accuracy_score

         y_pred_test = lstm_classifier.predict(X_test)

         cf_matrix=confusion_matrix(y_test, y_pred_test.round())

         print(cf_matrix)

         print(classification_report(y_test, y_pred_test.round(), digits=4))
```

```
              precision    recall  f1-score   support

           0       1.00      0.50      0.67       500
           1       0.00      0.00      0.00         0

    accuracy                           0.50       500
   macro avg       0.50      0.25      0.34       500
weighted avg       1.00      0.50      0.67       500
```

Figure 18: Testing the results on test data

4. To visualize the results, the confusion matrix is plotted using seaborn library.

6.2 Experiment 2.

After performing the first experiment, the hyper-parameters are tweaked by adjusting the number of layers in CNN, LSTM, etc. Also the rationale for implementing the second experiment is to avoid over fitting within the model while also increasing the accuracy of the model.

1. The hyper-parameters such as the number of LSTM Layers and CNN layers, along with dropout rate set to 0.25 as shown in table 2. Moreover the figure 20 shows the model structure retrieved in second stage.

Table 4: Executed Results for Experiment 2.

LSTM neurons	**CNN layers**	**Dropout**	**Number of epochs**
128	5	0.25	3,5,10

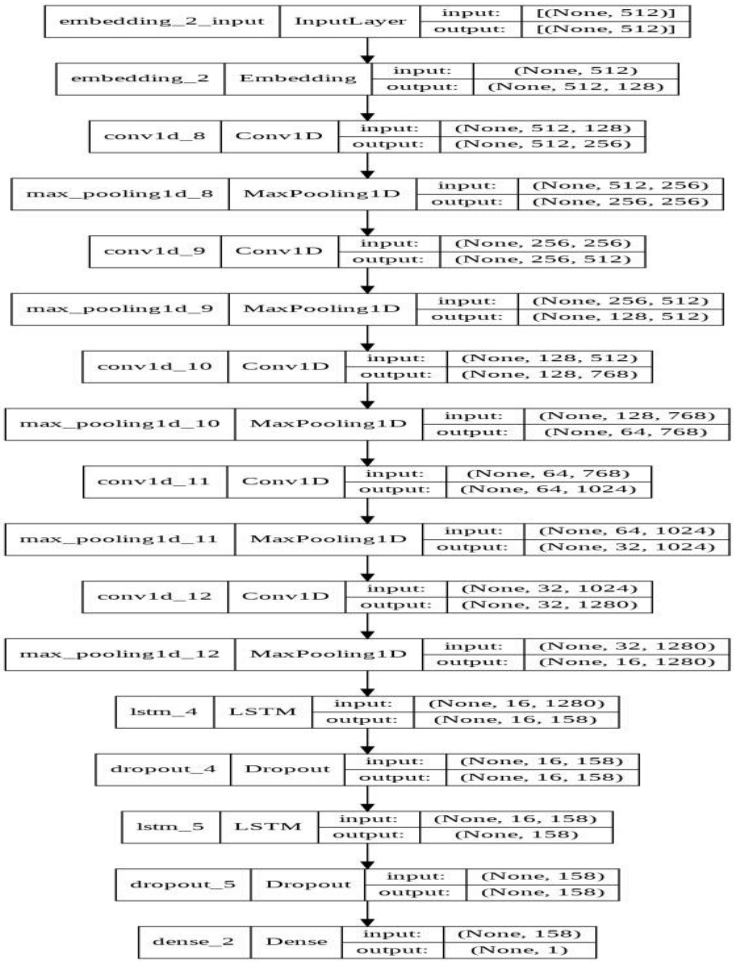


Figure 20: Experiment 2 model architecture

2. The model is then compiled again by implementing the fit function as shown in figure 21, the model architecture is detailed with 5 CNN layers and 128 neurons in LSTM layers.

▾ Model Training

```
lstm_classifier = LSTM_Text_Classifier(embedding_vector_length=128, max_seq_length=512, dropout=0.25,
                                       lstm_layers=[128, 128], batch_size=256, num_epochs=10, use_hash=False,
                                       conv_params={'filters': 128,
                                                    'kernel_size': 5,
                                                    'pool_size': 2,
                                                    'n_layers': 5})
```

```
lstm_classifier.fit(X_train, y_train, validation_data=(X_valid, y_valid))
```

```
Fitting model...
Epoch 1/10
162/162 [==============================] - 213s 1s/step - loss: 0.2703 - accuracy: 0.8741 - recall: 0.8420 - precision: 0.8992 - val_loss: 0.1151 - val_accuracy: 0.9572 - val_recall: 0.9365 - val_precis
Epoch 2/10
162/162 [==============================] - 181s 1s/step - loss: 0.0683 - accuracy: 0.9772 - recall: 0.9740 - precision: 0.9802 - val_loss: 0.1094 - val_accuracy: 0.9587 - val_recall: 0.9559 - val_precis
Epoch 3/10
162/162 [==============================] - 181s 1s/step - loss: 0.0249 - accuracy: 0.9927 - recall: 0.9916 - precision: 0.9937 - val_loss: 0.2030 - val_accuracy: 0.9314 - val_recall: 0.9704 - val_precis
Epoch 4/10
162/162 [==============================] - 181s 1s/step - loss: 0.0137 - accuracy: 0.9962 - recall: 0.9960 - precision: 0.9963 - val_loss: 0.1895 - val_accuracy: 0.9589 - val_recall: 0.9540 - val_precis
Epoch 5/10
162/162 [==============================] - 181s 1s/step - loss: 0.0110 - accuracy: 0.9966 - recall: 0.9967 - precision: 0.9965 - val_loss: 0.1696 - val_accuracy: 0.9578 - val_recall: 0.9603 - val_precis
Epoch 6/10
162/162 [==============================] - 182s 1s/step - loss: 0.0089 - accuracy: 0.9974 - recall: 0.9972 - precision: 0.9976 - val_loss: 0.1689 - val_accuracy: 0.9598 - val_recall: 0.9531 - val_precis
Epoch 7/10
162/162 [==============================] - 182s 1s/step - loss: 0.0066 - accuracy: 0.9980 - recall: 0.9980 - precision: 0.9981 - val_loss: 0.2292 - val_accuracy: 0.9554 - val_recall: 0.9349 - val_precis
Epoch 8/10
162/162 [==============================] - 182s 1s/step - loss: 0.0056 - accuracy: 0.9986 - recall: 0.9985 - precision: 0.9988 - val_loss: 0.2458 - val_accuracy: 0.9460 - val_recall: 0.9067 - val_precis
Epoch 9/10
162/162 [==============================] - 181s 1s/step - loss: 0.0095 - accuracy: 0.9968 - recall: 0.9967 - precision: 0.9969 - val_loss: 0.1894 - val_accuracy: 0.9557 - val_recall: 0.9641 - val_precis
Epoch 10/10
162/162 [==============================] - 181s 1s/step - loss: 0.0079 - accuracy: 0.9973 - recall: 0.9970 - precision: 0.9976 - val_loss: 0.1651 - val_accuracy: 0.9589 - val_recall: 0.9595 - val_precis
```

Figure 21: Experiment 2. Compiling and Fitting the model

3. The executed model is evaluated on test set and the results are then arranged in a confusion matrix as shown in figure 22.

```python
group_names = ['True Neg','False Pos','False Neg','True Pos']

group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]

group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]

labels = np.asarray(labels).reshape(2,2)

ax = sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')

ax.set_title('Seaborn Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

## Display the visualization of the Confusion Matrix.
plt.show()
```

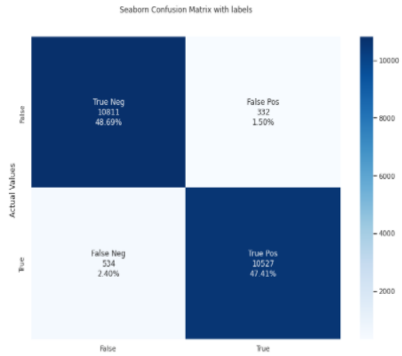


Figure 22: Experiment 2 confusion matrix

6.3 Experiment 3.

After the execution of 2nd experiment, a third and final experiment is performed to achieve maximum accuracy and precision from the model.

1. The hyper-parameters are adjusted again with dropout rate set to standard value of 0.5, moreover the number of neurons are increased in LSTM layers to228 as shown in table 3, also the architecture is plotted as shown in fig 23, with the final model architecture.

Table 5: Executed Results for Experiment 3.

LSTM neurons	CNN layers	Dropout	Number of epochs
228	5	0.5	3,5,10

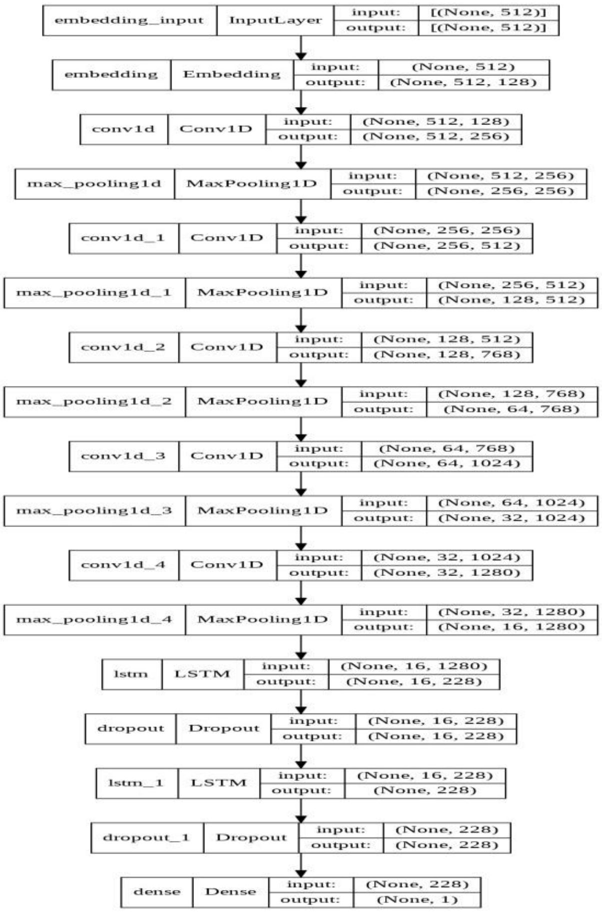
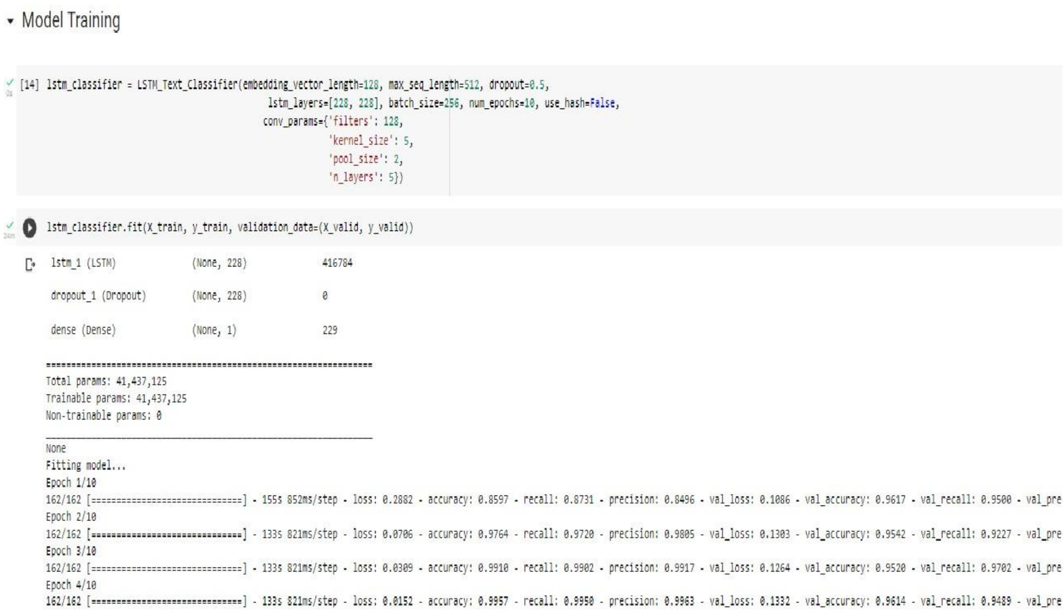


Figure 23: Experiment 3. Model architecture

2. The model is compiled for the last time as shown in figure 24.



▾ Model Training

```
[14] lstm_classifier = LSTM_Text_Classifier(embedding_vector_length=128, max_seq_length=512, dropout=0.5,
                                        lstm_layers=[228, 228], batch_size=256, num_epochs=10, use_hash=False,
                                       conv_params={'filters': 128,
                                                    'kernel_size': 5,
                                                    'pool_size': 2,
                                                    'n_layers': 5})
```

```
lstm_classifier.fit(X_train, y_train, validation_data=(X_valid, y_valid))
```

```
lstm_1 (LSTM)            (None, 228)           416784

dropout_1 (Dropout)      (None, 228)           0

dense (Dense)            (None, 1)             229

=================================================================
Total params: 41,437,125
Trainable params: 41,437,125
Non-trainable params: 0
_________________________________________________________________
None
Fitting model...
Epoch 1/10
162/162 [==============================] - 155s 852ms/step - loss: 0.2882 - accuracy: 0.8597 - recall: 0.8731 - precision: 0.8496 - val_loss: 0.1086 - val_accuracy: 0.9617 - val_recall: 0.9500 - val_pre
Epoch 2/10
162/162 [==============================] - 133s 821ms/step - loss: 0.0706 - accuracy: 0.9764 - recall: 0.9720 - precision: 0.9805 - val_loss: 0.1303 - val_accuracy: 0.9542 - val_recall: 0.9227 - val_pre
Epoch 3/10
162/162 [==============================] - 133s 821ms/step - loss: 0.0309 - accuracy: 0.9910 - recall: 0.9902 - precision: 0.9917 - val_loss: 0.1264 - val_accuracy: 0.9520 - val_recall: 0.9702 - val_pre
Epoch 4/10
162/162 [==============================] - 133s 821ms/step - loss: 0.0152 - accuracy: 0.9957 - recall: 0.9950 - precision: 0.9963 - val_loss: 0.1332 - val_accuracy: 0.9614 - val_recall: 0.9489 - val_pre
```

Figure 24: Experiment 3. Compiling and fitting the model

3. The results obtained after executing the final model is depicted in figure 25. with confusion matrix plotted and accuracy obtained as 95%.

```python
group_names = ['True Neg','False Pos','False Neg','True Pos']

group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]

group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]

labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]

labels = np.asarray(labels).reshape(2,2)

ax = sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')

ax.set_title('Seaborn Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

## Display the visualization of the Confusion Matrix.
plt.show()
```

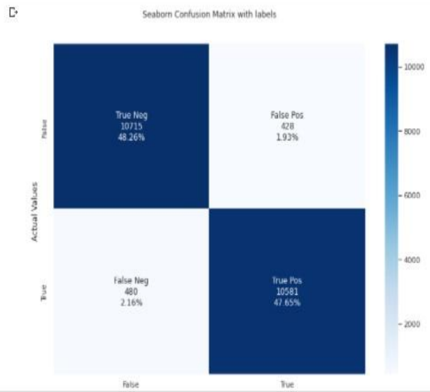


Figure 25: Plotting experiment 3. Confusion matrix