CHAPTER 5
IMPLEMENTATION AND RESULTS

5.1. Implementation

In this chapter, the implementation and testing of projects development about textual sentiment analysis using Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM) neural network model. Below is the code of the SNN, CNN and LSTM used to obtain results from the project developed.

```
1. import pandas as pd
2. import numpy as np
3. import re
4. import nltk
5. from nltk.corpus import stopwords
6. from numpy import array
7.
8. from keras.preprocessing.text import one_hot, Tokenizer
9. from keras_preprocessing.sequence import pad_sequences
10. from keras.models import Sequential
11. from keras.layers.core import Activation, Dropout, Dense
12. from keras.layers import Flatten, GlobalMaxPooling1D, Embedding, Conv1D,
    LSTM, Bidirectional
13. from sklearn.model_selection import train_test_split
```

import necessary modules and dependencies:

1. Import pandas: *pandas* are used to read dataset
2. Import numpy: *numpy* is used for processing arrays and also for basic calculations. *Numpy* has the ability to create n-dimensional array objects.
3. Import re: a regular expression will help to check if a particular string matches a given regular expression.
4. Import nltk: is a toolkit built for working with NLP, in this research *nltk* will help in preprocessing the dataset.
5. From nltk.corpus import stopwords: is a *nltk* that includes a list of stop words like "a", "an", "the", "of", etc. These words are words that we do not want to use in this research.
6. From numpy import array: this library provides an array object that is up to 50x faster than a traditional python list. Also, it provides a lot of supporting functions that make working with n-dimensional arrays very easy.

7. From keras.preprocessing.text import one_hot, Tokenizer: is used for vectorizing the text corpus.
8. From keras_preprocessing.sequence import pad_sequences: is a keras function that transforms a list of sequences into a *numpy* array of shape.
9. From keras.models import sequential: is one of keras libraries that contains a sequential API for arranging the keras layers in a sequential order.
10. From keras.layers.core import activation, dropout, dense: is an essential keras library for processing the data in hidden layers.
11. From keras.layers import layers: use keras as a library to import layers.
12. From sklearn.model_selection import train_test_split: use to split train data and test data.

	text	sentiment
0	@thesuperadi1 @Clay_Mann_ @TomKingTK @BruceSim...	positive
1	How come no one told me @TheBatman was this go...	positive
2	just rewatched @TheBatman and yeaâ€¦yâ€™all tw...	positive
3	@TheBatman @warnerbros Best Batman ever! Good ...	positive
4	@Jamiebower fantastic job in #StrangerThings y...	positive

Figure 5.1 shows the dataset with the corresponding value of each tweet

```
14. def preprocess_text(sen):
15.     '''Cleans text data up, leaving only 2 or more char long non-
    stepwords composed of A-Z & a-z only
16.     in lowercase'''
17.
18.     sentence = sen.lower()
19.
20.     # Remove html tags
21.     sentence = remove_tags(sentence)
22.
23.     # Remove punctuations and numbers
24.     sentence = re.sub('[^a-zA-Z]', ' ', sentence)
25.
26.     # Single character removal
27.     sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)  # When we remove
    the apostrophe from the word "John's", the apostrophe is replaced by an
    empty space. Hence, we are left with the single character "s" that we are
    removing here.
28.
29.     # Remove multiple spaces
```

```
30.     sentence = re.sub(r'\s+', ' ', sentence)  # Next, we remove all the
   single characters and replace it by a space which creates multiple spaces
   in our text. Finally, we remove the multiple spaces from our text as well.
31.
32.     # Remove Stopwords
33.     pattern = re.compile(r'\b(' + r'|'.join(stopwords.words('english'))
   + r')\b\s*')
34.     sentence = pattern.sub('', sentence)
35.
36.     return sentence
```

Lines 14-36 is a function that performs a series of operations on the input review. First in line 18 is conversion to lowercase, then line 21 is a function to remove any html tags, line 24 is for removing any punctuations and numbers, and line 27 are explained in the code and then line 30 is for removing any multiple spaces and lastly line 33 is the process of removing stopwords.

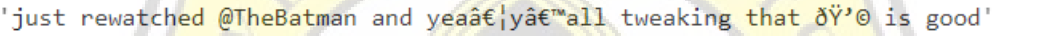

Figure 5.2 is an example of raw unprocessed data

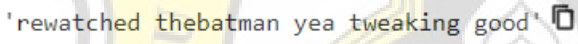

Figure 5.3 is a result of preprocessing the data

```
37. y = tweet_comments['sentiment']
38.
39. y = np.array(list(map(lambda x: 1 if x=="positive" else 0, y)))
```

Lines 37-39 is a process to convert the labeled sentiment of positive and negative into 1 if positive and 0 if negative.

```
40. X_train,   X_test,   y_train,   y_test   =   train_test_split(X,   y,
   test_size=0.20, random_state=20)
```

Line 40 is a process of splitting the dataset into a 70:30 training test.

```
41. word_tokenizer = Tokenizer()
42. word_tokenizer.fit_on_texts(X_train)
43.
44. X_train = word_tokenizer.texts_to_sequences(X_train)
45. X_test = word_tokenizer.texts_to_sequences(X_test)
```

Lines 41-45 is for tokenizing the data with the help of tokenizer class from keras.preprocessing module to create a word to index dictionary. In a word to index dictionary,

each word in the corpus is used as a key, while a corresponding unique index is used as the value of the key.

```
46. from numpy import asarray
47. from numpy import zeros
48.
49. embeddings_dictionary = dict()
50. glove_file = open('a2_glove.6B.100d.txt', encoding="utf8")
51.
52. for line in glove_file:
53.   records = line.split()
54.   word = records[0]
55.   vector_dimensions = asarray(records[1:], dtype='float32')
56.   embeddings_dictionary [word] = vector_dimensions
57. glove_file.close()
```

In lines 46-57 is a process to load the *glove* word embeddings and create a dictionary that will contain words as keys and their corresponding embedding list as values.

```
58. embedding_matrix = zeros ((vocab_length, 100))
59. for word, index in word_tokenizer.word_index.items():
60.   embedding_vector = embeddings_dictionary.get(word)
61.   if embedding_vector is not None:
62.     embedding_matrix[index] = embedding_vector
```

Lines 58-62 is a step where an embedding matrix will be created where each row number will correspond to the index of the word in the corpus. The matrix will have 100 columns where each column will contain the *glove* word embeddings.

```
63. cnn_model = Sequential()
64.
65. embedding_layer = Embedding(vocab_length, 100,
    weights=[embedding_matrix], input_length=maxlen , trainable = False)
66. cnn_model.add(embedding_layer)
67.
68. cnn_model.add(Conv1D(128, 5, activation='relu'))
69. cnn_model.add(GlobalMaxPooling1D())
70. cnn_model.add(Dense(1, activation='sigmoid'))
```

Lines 63-70 is the architecture for the CNN models, CNN mostly used for image but also can be used for text data by using Conv1D to extract features from the data. In this architecture, the model architecture is CNN with 1 convolutional layer and 1 pooling layer. The embedding layer will have an input length of 100 and the output will also be 100 and since in this research *glove* word embedding is used, trainable will be set into false state.

```
71. cnn_model_history = cnn_model.fit(X_train, y_train, batch_size=128,
    epochs=20, verbose=1, validation_split=0.2)
```

```
score = cnn_model.evaluate(X_test, y_test, verbose=1)
print("Test score:", score[0])
print("Test Accuracy:", score[1])
```

Lines 71-74 is the process of training the data into the CNN model with an epoch of 20 and then the prediction of the test will be conducted in line 72 and the accuracy of the model will be shown in line 74.

```
lstm_model = Sequential()
embedding_layer = Embedding(vocab_length, 100, weights=[embedding_matrix], input_length=maxlen , trainable=False)

lstm_model.add(embedding_layer)
lstm_model.add(Bidirectional(LSTM(128)))
lstm_model.add(Dense(1, activation='sigmoid'))
```

Lines 75-80 is the model architecture of the LSTM model with the same embedding layer and the LSTM model will use Bidirectional to further improve the accuracy.

```
lstm_model_history = lstm_model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=1, validation_split=0.2)
score = lstm_model.evaluate(X_test, y_test, verbose=1)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
```

Line 81 is the process of training the data into the LSTM model with the same epochs of 20 and then in line 82 prediction on test data is conducted and line 84 will show the accuracy of the model performance.

5.2. Results

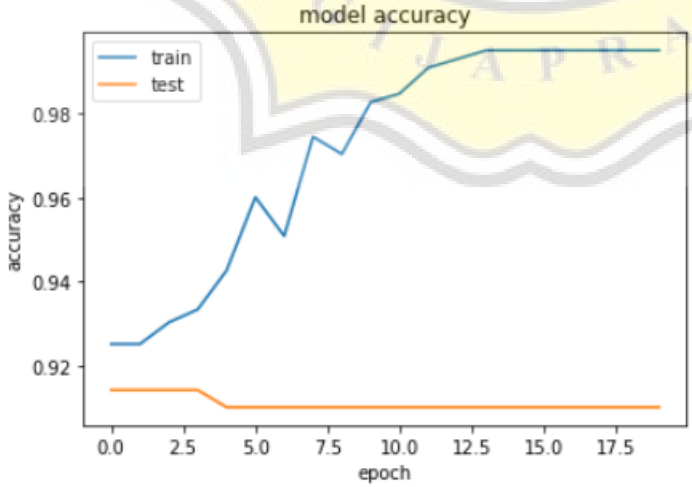

Figure 5.1 model accuracy of CNN

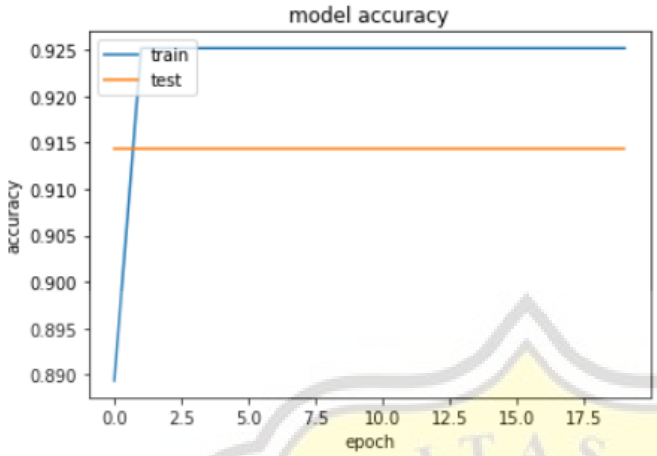

Figure 5.2 model accuracy of BidiLSTM

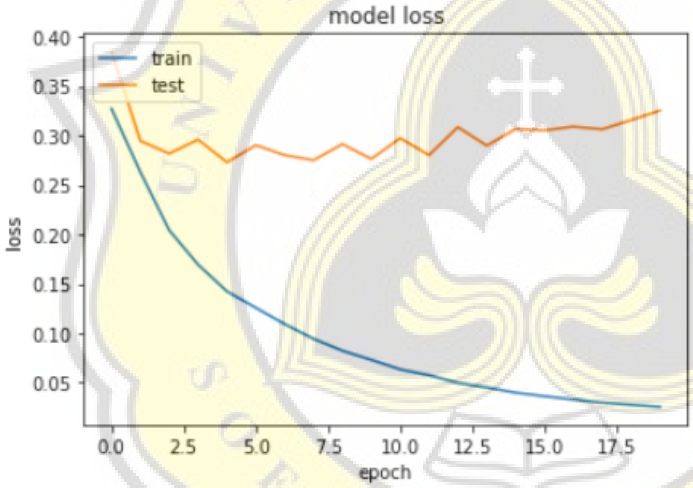

Figure 5.3 model loss of CNN

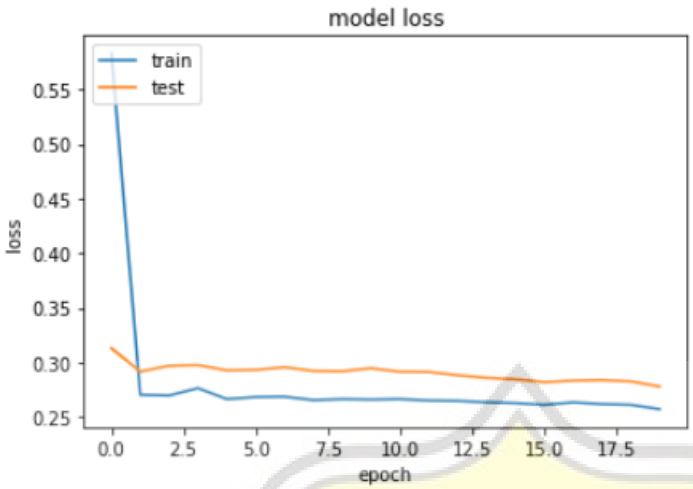

Figure 5.4 model loss of BidiLSTM

From the image above the image shows the training and validation accuracy of a CNN model and a LSTM model on a sentiment analysis task. The x-axes represents the epoch number (a full pass through the training data), and the y-axis represents the accuracy. The blue curve represents the training and accuracy, and the orange curve represents the validation accuracy.

From the images, we can see that both the CNN and LSTM models have similar performance in terms of accuracy and loss, although the CNN model seems to have slightly better performance. The models seem to have reached a plateau in terms of accuracy and loss after about 5-10 epochs, which means that further training is unlikely to lead to significant improvement in performance.

Table 5.1. Experimental result of both algorithms.

Algorithm	Model	Epochs	Batch Size	Accuracy
Convolutional Neural Network	Sequential	20	128	89%
Long Short-Term Memory Neural Network	Sequential	20	128	90%

The experiments that have been done, I obtained the best accuracy from LSTM of 90%. Judging from both accuracy, LSTM is indeed better than CNN for textual sentiment analysis using neural networks.

Table 5.2. Model summary of Convolutional Neural network

Layer	Output Shape	Param
Embedding_7	(None, 40, 100)	415600
Convi1d_1 (Conv1D)	(None, 36, 128)	64128
Global_max_pooling1d_1 (GlobalMaxPooling1D)	(None, 128)	0
Dense_6 (Dense)	(None, 1)	129
Total Params:	479,857	
Trainable params:	64,257	
Non-trainable params:	415,600	
Test set accuracy:	89%	

Table 5.3.
Model summary of Bidirectional Long Short Term Memory Neural Network

Layer	Output Shape	Param
Embedding_9	(None, 40, 100)	415600
Bidirectional_1 (Bidirectional)	(None, 256)	234496
Dense_8 (Dense)	(None, 1)	257
Total Params:	650,353	
Trainable params:	234,753	
Non-trainable params:	415,600	
Test set accuracy:	90%	