CS6301 MACHINE LEARNING
LAB PROJECT

TEAM MEMBERS:

PRIYADHARSHINI P - 2019103562
PRASHANTH S - 2019103556
PRAVEEN V M - 2019103559

SOCIAL MEDIA POST ANALYSIS - SENTIMENT CLASSIFICATION

ABSTRACT

Categorization is useful for analysis of communication as a framework for analyzing how people differentiate among things, people, experiences, or ideas. Our project aims at categorizing the social media posts by analyzing the sentiment of the posts. Sentiment classification is the process of identifying opinions in text and labeling them as positive, negative based on the emotions people express in them.

DEEP LEARNING ALGORITHMS

1. CONVOLUTIONAL NEURAL NETWORK (CNN)

METHODOLOGY

The captions are preprocessed. Stop words are eliminated. Neural networks can only learn to find patterns in numerical data and so, before we feed a description into a neural network as input, we have to convert each word into a numerical value. This process is often called word encoding or tokenization. The captions are tokenized by the Tokenizer module in Keras. All tokenized caption vectors are resized to equal size by padding. All caption vectors are resized to the length of the maximum length caption. So, short length captions are padded with zeros. These tokenized caption vectors and their corresponding labels are used to train the CNN model. The layers in the CNN are Convolutional layer, Pooling layer, Dense layer.

The activation functions used in the Dense layer are relu and sigmoid functions. The loss function used is BinaryCrossEntropy. The number of epochs is set as 5 and batch size is 1500. The CNN model is trained and tested.

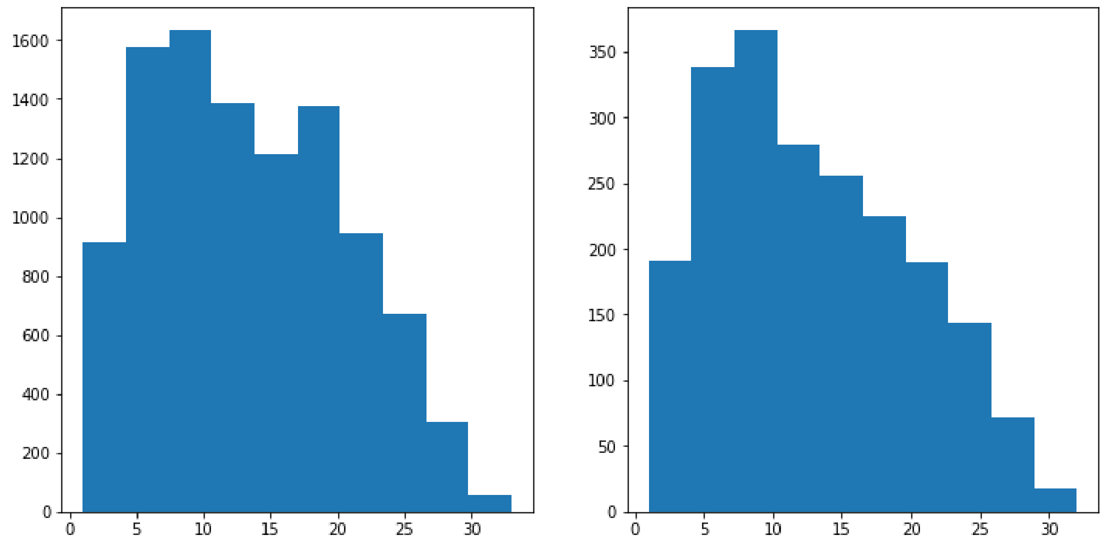

Fig 1 a. Length of training captions
Fig 1 b. Length of testing captions
The unequal length captions are resized to equal length by padding.

DATASET
Reference Link:
https://www.kaggle.com/datasets/kazanova/sentiment140

Attributes :
1. TARGET - The polarity of the post (0 - negative , 1 - positive)
2. ID - The ID of the post
3. DATE - The date of the post
4. FLAG - The query
5. USER - The user that posted
6. DESCRIPTION - The text of the post

This dataset contains 10,51,304 instances (Existing dataset from row 1 to 1046574, Appended dataset from row 1046575 to 1051304). We web scraped posts and collected 4700 instances and we have appended these instances in the dataset. Dataset is attached named as Dataset.csv .

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 300)          5679000

 conv1d (Conv1D)             (None, 100, 128)          153728

 max_pooling1d (MaxPooling1D  (None, 50, 128)          0
 )

 conv1d_1 (Conv1D)           (None, 50, 64)            32832

 max_pooling1d_1 (MaxPooling  (None, 25, 64)           0
 1D)

 conv1d_2 (Conv1D)           (None, 25, 32)            8224

 max_pooling1d_2 (MaxPooling  (None, 12, 32)           0
 1D)

 flatten (Flatten)           (None, 384)               0

 dense (Dense)               (None, 256)               98560

 dense_1 (Dense)             (None, 1)                 257

=================================================================
Total params: 5,972,601
Trainable params: 5,972,601
Non-trainable params: 0
```

Fig 2 - Layers of CNN and their output dimensions

RESULTS

Accuracy - 93.66 %

```
Epoch 1/5
WARNING:tensorflow:AutoGraph could not transform <function Model.make_train_function.<locals>.train_function at 0x000001FF07AD4
438> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=
10`) and attach the full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_train_function.<locals>.train_function at 0x000001FF07AD4438> and w
ill run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=
10`) and attach the full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
7/7 [==============================] - ETA: 0s - loss: 0.6877 - accuracy: 0.5503WARNING:tensorflow:AutoGraph could not transfor
m <function Model.make_test_function.<locals>.test_function at 0x000001FF0AB86558> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=
10`) and attach the full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_test_function.<locals>.test_function at 0x000001FF0AB86558> and wil
l run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=
10`) and attach the full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert

WARNING: AutoGraph could not transform <function Model.make_test_function.<locals>.test_function at 0x000001FF0AB86558> and wil
l run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=
10`) and attach the full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
7/7 [==============================] - 11s 1s/step - loss: 0.6877 - accuracy: 0.5503 - val_loss: 0.7801 - val_accuracy: 0.0000e
+00
Epoch 2/5
7/7 [==============================] - 9s 1s/step - loss: 0.6676 - accuracy: 0.5705 - val_loss: 0.7123 - val_accuracy: 0.5754
Epoch 3/5
7/7 [==============================] - 9s 1s/step - loss: 0.5841 - accuracy: 0.7319 - val_loss: 0.5848 - val_accuracy: 0.7272
Epoch 4/5
7/7 [==============================] - 9s 1s/step - loss: 0.3915 - accuracy: 0.8392 - val_loss: 0.6830 - val_accuracy: 0.7212
Epoch 5/5
7/7 [==============================] - 9s 1s/step - loss: 0.2192 - accuracy: 0.9134 - val_loss: 0.5606 - val_accuracy: 0.7976

<keras.callbacks.History at 0x1ff083b1c48>
```

Fig 3 - Training the CNN model

```
WARNING:tensorflow:AutoGraph could not transform <function Model.make_predict_function.<locals>.predict_function at 0x000001FF0
7AD4F78> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=
10`) and attach the full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_predict_function.<locals>.predict_function at 0x000001FF07AD4F78> a
nd will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=
10`) and attach the full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
250/250 [==============================] - 3s 11ms/step

250/250 [==============================] - 3s 12ms/step - loss: 0.1645 - accuracy: 0.9366
Accuracy: 93.66%
```

Fig 4 - Accuracy obtained

	precision	recall	f1-score	support
0	0.98	0.92	0.95	4999
1	0.87	0.97	0.92	3001
accuracy			0.94	8000
macro avg	0.93	0.94	0.93	8000
weighted avg	0.94	0.94	0.94	8000

	negative	positive
negative	4576	423
positive	84	2917

Fig 5 - Results - Precision, Recall, F1-score, Support, Confusion matrix

CONVOLUTIONAL NEURAL NETWORK (CNN) USING DISTILBERT

METHODOLOGY

DistilBERT is a small, fast, cheap and light Transformer model based on the BERT architecture. Knowledge distillation is performed during the pre-training phase to reduce the size of a BERT model by 40%. Distillation of Knowledge is an approach for generalization of knowledge within a neural network to train another neural network. It is the process of transferring knowledge from a large model to a smaller one.

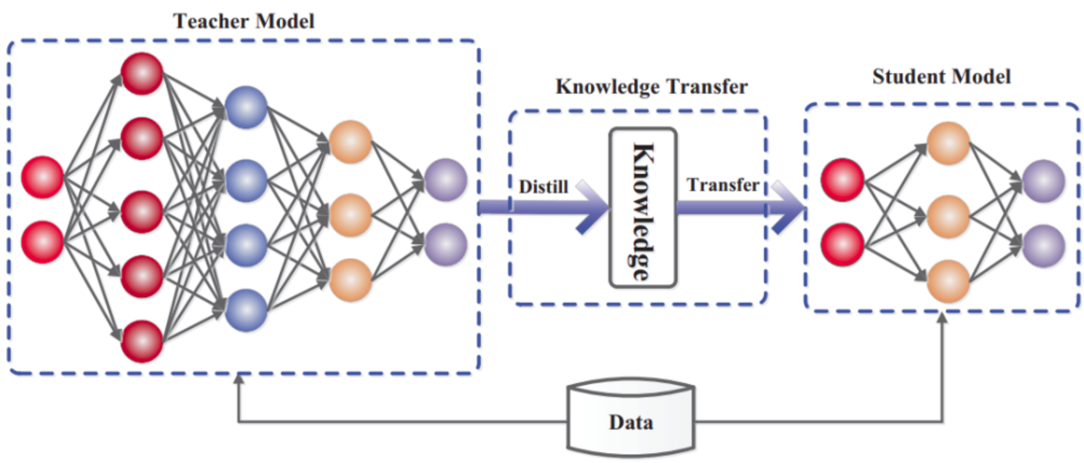

Fig 6 - Knowledge Distillation

DistilBERT has 40% less parameters than bert-base-uncased, runs 60% faster while preserving over 95% of BERT's performances as measured on the GLUE language understanding benchmark.

DistilBERT Implementation in Keras.

First, the trained distilBERT was used to generate sentence embedding (300 dimensions) for the dataset.

Then a basic CNN Architecture was used for the further classification task and the training.

Finally, the evaluation of the model.

All unique words are extracted from the description and this is used for building a token dictionary / vocabulary. Captions are converted to vectors which are lists of integer values.
Vocabulary - all unique words in the description
Token - an integer value assigned to each word in the vocabulary.

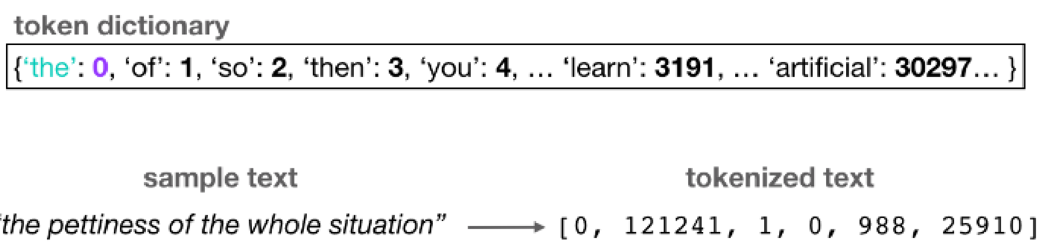

Fig 7 - Tokenization

Each description is padded with zeros so that the length is made equal to the length of the maximum length caption. Each word in the description is assigned an id by the pretrained DistilBert Tokenizer. Each caption is tokenized by the DistilBert Tokenizer and converted into a vector. The tokenizer has two attributes namely input id and attention mask. The input ids are the tokenized caption, simply the mapping between the tokens and their respective ids. The attention mask is to prevent the model from looking at the padding tokens. The DistilBert model takes in the input ids and attention masks encoded by the tokenizer as the input. In this way, the trained DistilBert model is used to embed the text.

Next, we construct a CNN model with Dense layer and a dropout layer. The inputs provided to the model are the input ids and attention masks encoded by the DistilBert tokenizer. The activation function defined is softmax and the loss function defined is SparseCategoricalCrossEntropy. The batch size is set to be 1500 and the number of epochs is 5. The model is trained and then tested. The accuracy obtained is 74.5%.

BLOCK DIAGRAM

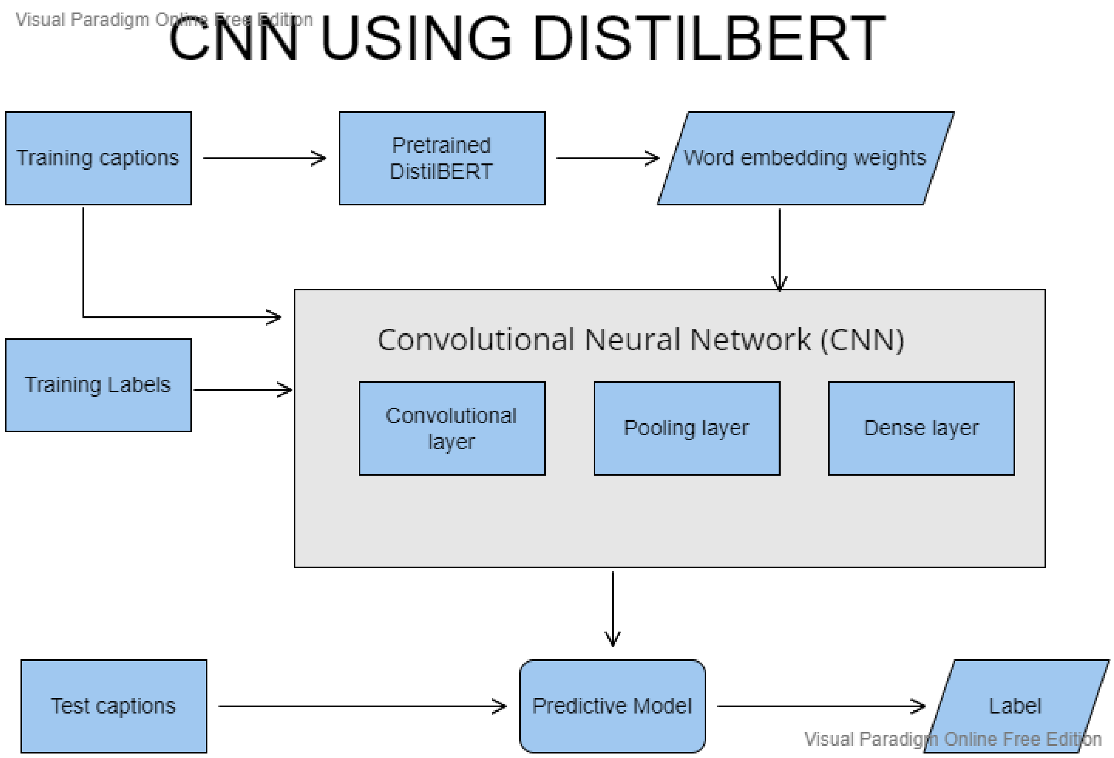

Fig 8 - Block Diagram

```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_3 (InputLayer)           [(None, 32)]         0           []

 input_4 (InputLayer)           [(None, 32)]         0           []

 tf_distil_bert_model (TFDistil  ((None, 32, 768),   66362880    ['input_3[0][0]',
 BertModel)                     )                                 'input_4[0][0]']

 tf.__operators__.getitem_1 (Sl  (None, 768)         0           ['tf_distil_bert_model[1][0]']
 icingOpLambda)

 dense_2 (Dense)                (None, 512)          393728      ['tf.__operators__.getitem_1[0][0
                                                                 ]']

 dropout_20 (Dropout)           (None, 512)          0           ['dense_2[0][0]']

 dense_3 (Dense)                (None, 2)            1026        ['dropout_20[0][0]']

==================================================================================================
Total params: 66,757,634
Trainable params: 66,757,634
Non-trainable params: 0
__________________________________________________________________________________________________
None
```

Fig 9 - Layers of embedded CNN and their output dimensions

RESULTS

Accuracy - 74.5%

```
Epoch 1/5
WARNING:tensorflow:AutoGraph could not transform <function Model.make_train_function.<locals>.train_function at 0x0000018B9FC57
798> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=
10`) and attach the full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_train_function.<locals>.train_function at 0x0000018B9FC57798> and w
ill run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=
10`) and attach the full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert

C:\Users\user\anaconda3\envs\env1\lib\site-packages\tensorflow\python\util\dispatch.py:1082: UserWarning: "`sparse_categorical_
crossentropy` received `from_logits=True`, but the `output` argument was produced by a sigmoid or softmax activation and thus d
oes not represent logits. Was this intended?"
  return dispatch_target(*args, **kwargs)

6/6 [==============================] - ETA: 0s - loss: 6.8636 - accuracy: 0.5476  WARNING:tensorflow:AutoGraph could not transf
orm <function Model.make_test_function.<locals>.test_function at 0x0000018BA2477A68> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=
10`) and attach the full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_test_function.<locals>.test_function at 0x0000018BA2477A68> and wil
l run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=
10`) and attach the full output.
Cause: 'arguments' object has no attribute 'posonlyargs'

To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_test_function.<locals>.test_function at 0x0000018BA2477A68> and wil
l run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=
10`) and attach the full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
6/6 [==============================] - 1481s 252s/step - loss: 6.8636 - accuracy: 0.5476 - val_loss: 6.7708 - val_accuracy: 0.6
429
Epoch 2/5
6/6 [==============================] - 1672s 274s/step - loss: 6.7518 - accuracy: 0.6383 - val_loss: 6.6842 - val_accuracy: 0.6
736
Epoch 3/5
6/6 [==============================] - 1742s 284s/step - loss: 6.6553 - accuracy: 0.6900 - val_loss: 6.5906 - val_accuracy: 0.7
257
Epoch 4/5
6/6 [==============================] - 1609s 264s/step - loss: 6.5493 - accuracy: 0.7445 - val_loss: 6.5259 - val_accuracy: 0.7
470
Epoch 5/5
6/6 [==============================] - 1644s 266s/step - loss: 6.4592 - accuracy: 0.7733 - val_loss: 6.4894 - val_accuracy: 0.7
445
```

Fig 10 - Training the CNN model with DistilBERT word embeddings

```
F1 score 0.7265002655337227
Classification Report
              precision    recall  f1-score   support

    negative       0.70      0.83      0.76       979
    positive       0.81      0.66      0.73      1037

    accuracy                           0.74      2016
   macro avg       0.75      0.75      0.74      2016
weighted avg       0.75      0.74      0.74      2016

63/63 [==============================] - 45s 719ms/step - loss: 6.4893 - accuracy: 0.7445
Accuracy: 74.45%
Training and saving built model.....
              precision    recall  f1-score   support

           0       0.70      0.83      0.76       979
           1       0.81      0.66      0.73      1037

    accuracy                           0.74      2016
   macro avg       0.75      0.75      0.74      2016
weighted avg       0.75      0.74      0.74      2016
```

	negative	positive
negative	817	162
positive	353	684

Fig 11 - Results - Precision, Recall, F1-Score, Support, Confusion Matrix

ACCURACY GRAPH

Accuracy graph is the accuracy obtained for a varying number of epochs.

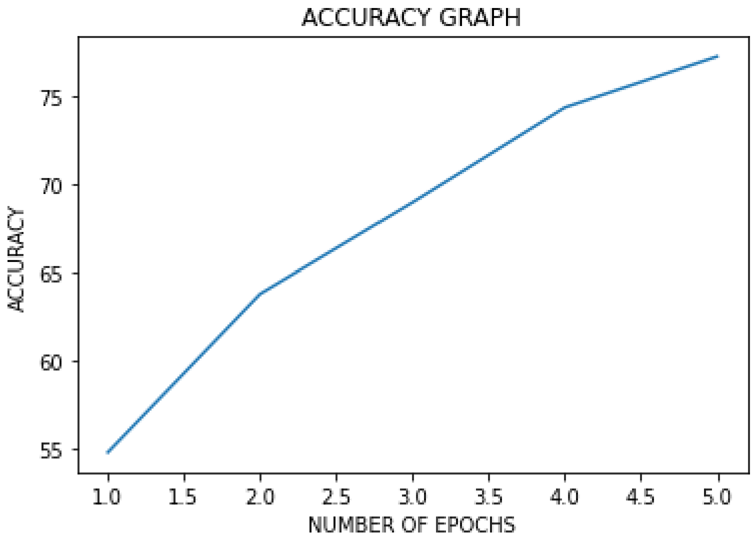

Fig 12 - Accuracy Graph

2. LONG SHORT-TERM MEMORY (LSTM)

METHODOLOGY

Numbers, symbols and special characters are removed as well as stop words are eliminated from the captions. Those are then tokenized by the Tokenizer module in Keras. All tokenized caption vectors are resized to equal size by padding. All caption vectors are resized to the length of the maximum length caption. So, short length captions are padded with zeros. These tokenized caption vectors and their corresponding labels are used to train the LSTM model. The layers in the LSTM are Embedding layer, bidirectional layer(LSTM), Dense layer.

```
[['Awww', 'bummer', 'shoulda', 'got', 'David', 'Carr', 'Third', 'Day'],
 ['upset',
  'update',
  'Facebook',
  'texting',
  'might',
  'cry',
  'result',
  'School',
  'today',
  'also',
  'Blah'],
 ['dived', 'many', 'times', 'ball', 'Managed', 'save', 'rest', 'go', 'bounds'],
 ['whole', 'body', 'feels', 'itchy', 'like', 'fire'],
 ['behaving', 'mad', 'see']]
```

Fig 13 - Caption before tokenizing

```
[[351, 1010, 2973, 11, 687, 7237, 1624, 3],
 [576, 215, 419, 345, 199, 229, 946, 74, 9, 184, 962],
 [3515, 221, 10, 806, 683, 495, 337, 1, 2630],
 [329, 627, 20, 2442, 6, 837],
 [3881, 444, 19]]
```

Fig 14 - Caption after tokenizing

The activation function used in the Dense layer is sigmoid. The loss function used is BinaryCrossEntropy. The number of epochs is set as 5 and batch size is 1024 with the validation split as 0.1. The LSTM model is then trained and tested.

BLOCK DIAGRAM

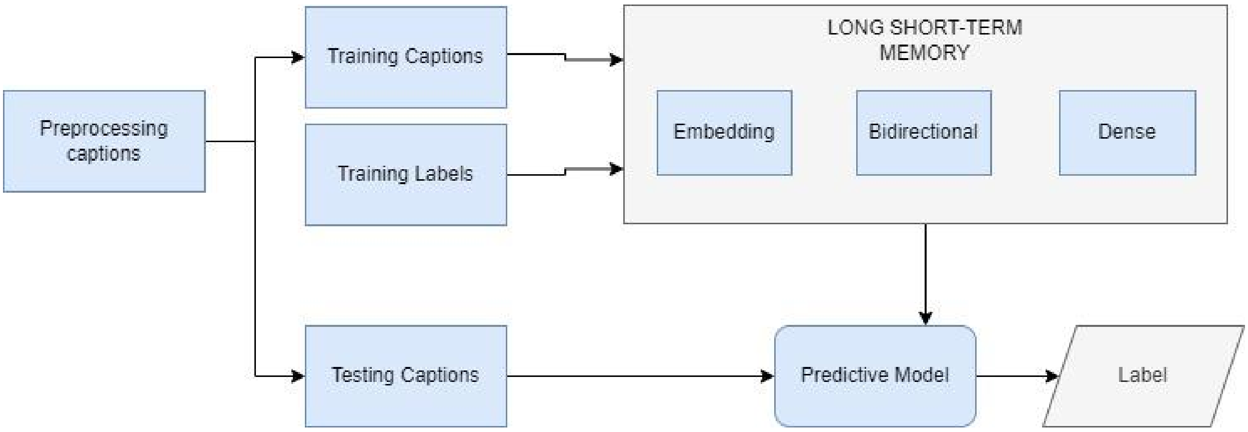

Fig 15 - Block Diagram for LSTM

The neural network involves the layers in the given figure:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 50, 15)            3314850

 bidirectional (Bidirectiona  (None, 128)              40960
 l)

 dense (Dense)               (None, 1)                 129

=================================================================
Total params: 3,355,939
Trainable params: 3,355,939
Non-trainable params: 0
_________________________________________________________________
```

Fig 16 - Layers and their output dimensions

RESULTS

The model runs for five epochs and produces the following results:

Accuracy - 78.40%

```
Epoch 1/5
1266/1266 [==============================] - 1002s 784ms/step - loss: 0.4909 - accuracy: 0.7618 - val_loss: 0.4674 - val_accura
cy: 0.7740
Epoch 2/5
1266/1266 [==============================] - 966s 763ms/step - loss: 0.4582 - accuracy: 0.7826 - val_loss: 0.4598 - val_accurac
y: 0.7795
Epoch 3/5
1266/1266 [==============================] - 993s 784ms/step - loss: 0.4489 - accuracy: 0.7881 - val_loss: 0.4579 - val_accurac
y: 0.7808
Epoch 4/5
1266/1266 [==============================] - 977s 772ms/step - loss: 0.4425 - accuracy: 0.7924 - val_loss: 0.4554 - val_accurac
y: 0.7825
Epoch 5/5
1266/1266 [==============================] - 982s 776ms/step - loss: 0.4367 - accuracy: 0.7964 - val_loss: 0.4558 - val_accurac
y: 0.7833
```

Fig 17 - Training the LSTM model

The below graph illustrates the comparison between accuracy and validation accuracy in each epoch.

```
Text(0.5, 0, 'epochs')
```

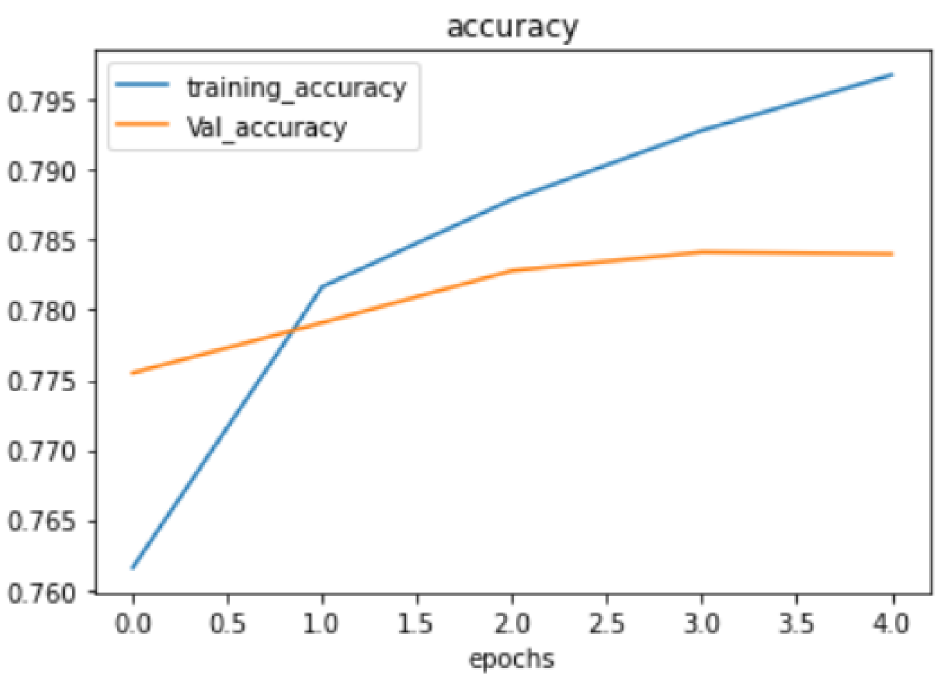

Fig 18 - Accuracy graph

The below graph illustrates the comparison between loss and validation loss in each epoch.

```
Text(0.5, 0, 'epochs')
```

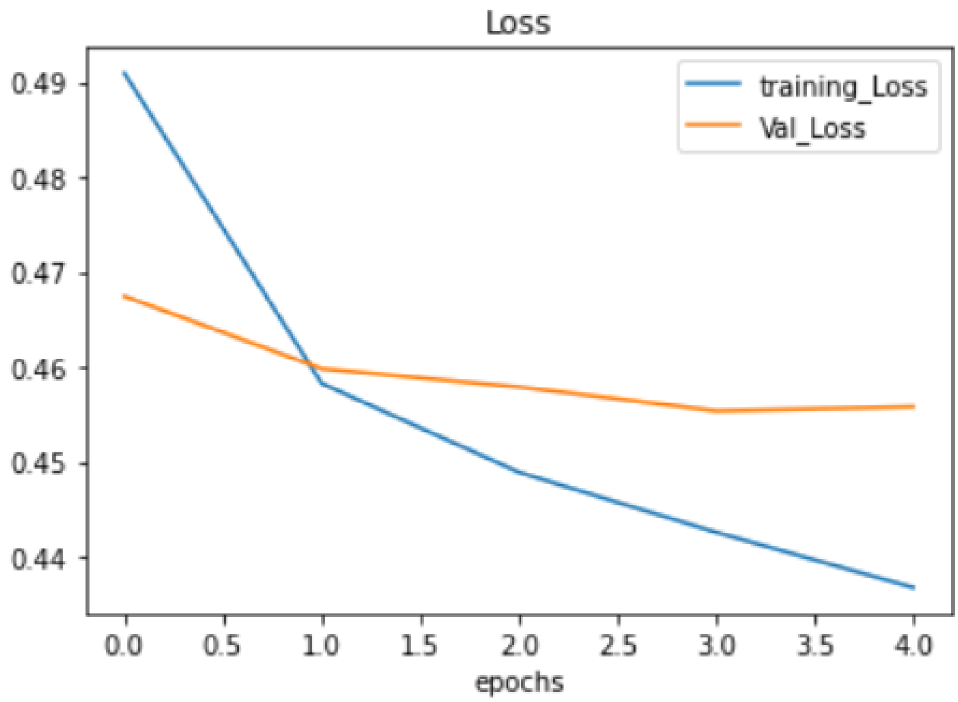

Fig 19 - Loss graph

LSTM USING GLOVE EMBEDDING

Methodology

Numbers, symbols and special characters are removed as well as stop words are eliminated from the captions. Those are then tokenized by the Tokenizer module in Keras. All tokenized caption vectors are resized to equal size by padding. All caption vectors are resized to the length of the maximum length caption. So, short length captions are padded with zeros. Then, the embedding vector is constructed from GloVe 100 and the corresponding embedding matrix is constructed. These tokenized caption vectors and their corresponding labels are used to train the LSTM model. The layers in the LSTM are Embedding layer, Dropout layer, two bidirectional layers(LSTM) and two dense layers.

Block Diagram

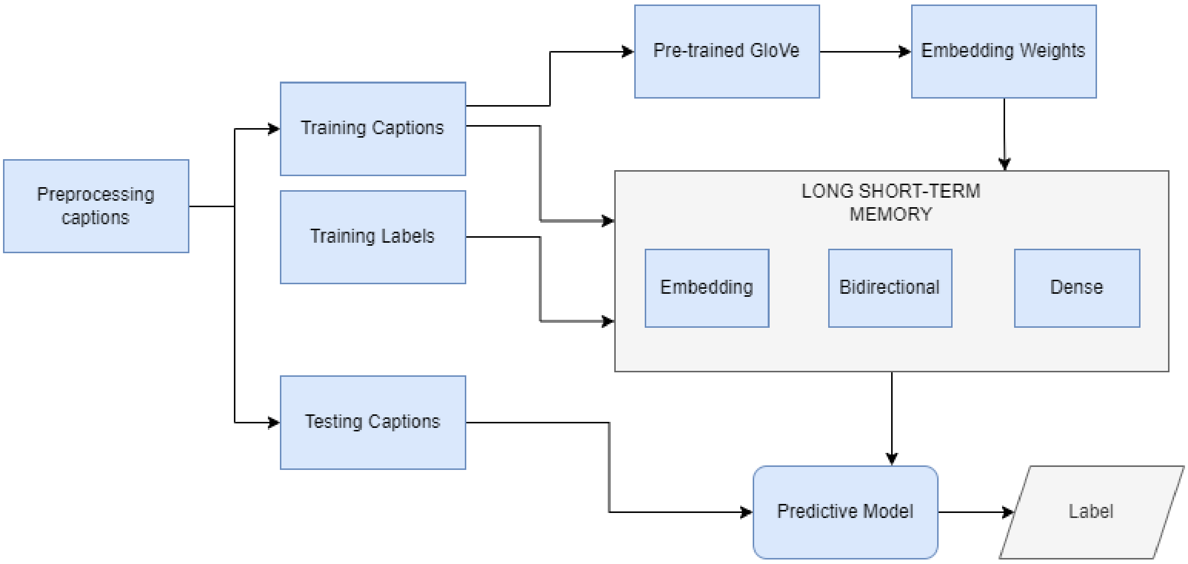

Fig 20 - Block diagram for LSTM with GloVe embeddings

The neural network involves the layers in the given figure:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 20, 100)           9999500

 dropout_1 (Dropout)         (None, 20, 100)           0

 bidirectional_2 (Bidirectio  (None, 20, 128)          84480
 nal)

 bidirectional_3 (Bidirectio  (None, 256)              263168
 nal)

 dense_2 (Dense)             (None, 64)                16448

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 10,363,661
Trainable params: 364,161
Non-trainable params: 9,999,500
```

Fig 21 - Layers and their output dimensions

Results

The model runs for 15 epochs and produces the following results

Accuracy - 48.6%

```
________________________________________________________________
Epoch 1/15
381/381 [==============================] - 134s 307ms/step - loss: 0.8010 - accuracy: 0.5009 - val_loss: 0.6933 - val_accuracy:
0.4860
Epoch 2/15
381/381 [==============================] - 121s 318ms/step - loss: 0.6931 - accuracy: 0.5025 - val_loss: 0.6934 - val_accuracy:
0.4860
Epoch 3/15
381/381 [==============================] - 119s 311ms/step - loss: 0.6931 - accuracy: 0.5004 - val_loss: 0.6934 - val_accuracy:
0.4860
Epoch 4/15
381/381 [==============================] - 118s 310ms/step - loss: 0.6931 - accuracy: 0.5025 - val_loss: 0.6933 - val_accuracy:
0.4860
Epoch 5/15
381/381 [==============================] - 119s 312ms/step - loss: 0.6931 - accuracy: 0.5025 - val_loss: 0.6933 - val_accuracy:
0.4860
Epoch 6/15
381/381 [==============================] - 121s 318ms/step - loss: 0.6931 - accuracy: 0.5025 - val_loss: 0.6933 - val_accuracy:
0.4860
Epoch 7/15
381/381 [==============================] - 109s 286ms/step - loss: 0.6931 - accuracy: 0.5019 - val_loss: 0.6934 - val_accuracy:
0.4860
Epoch 8/15
381/381 [==============================] - 80s 211ms/step - loss: 0.6931 - accuracy: 0.5025 - val_loss: 0.6932 - val_accuracy:
0.4860
Epoch 9/15
381/381 [==============================] - 86s 226ms/step - loss: 0.6931 - accuracy: 0.5006 - val_loss: 0.6935 - val_accuracy:
0.4860
Epoch 10/15
381/381 [==============================] - 72s 190ms/step - loss: 0.6932 - accuracy: 0.5025 - val_loss: 0.6934 - val_accuracy:
0.4860

Epoch 11/15
381/381 [==============================] - 78s 206ms/step - loss: 0.6931 - accuracy: 0.5017 - val_loss: 0.6934 - val_accuracy:
0.4860
Epoch 12/15
381/381 [==============================] - 88s 230ms/step - loss: 0.6931 - accuracy: 0.5025 - val_loss: 0.6934 - val_accuracy:
0.4860
Epoch 13/15
381/381 [==============================] - 85s 223ms/step - loss: 0.6932 - accuracy: 0.5018 - val_loss: 0.6934 - val_accuracy:
0.4860
Epoch 14/15
381/381 [==============================] - 110s 287ms/step - loss: 0.6931 - accuracy: 0.5025 - val_loss: 0.6932 - val_accuracy:
0.4860
Epoch 15/15
381/381 [==============================] - 91s 240ms/step - loss: 0.6931 - accuracy: 0.5017 - val_loss: 0.6933 - val_accuracy:
0.4860
Training Complete
```

Fig 22 - Training the LSTM model

The below graph illustrates the comparison between accuracy and validation accuracy in each epoch.

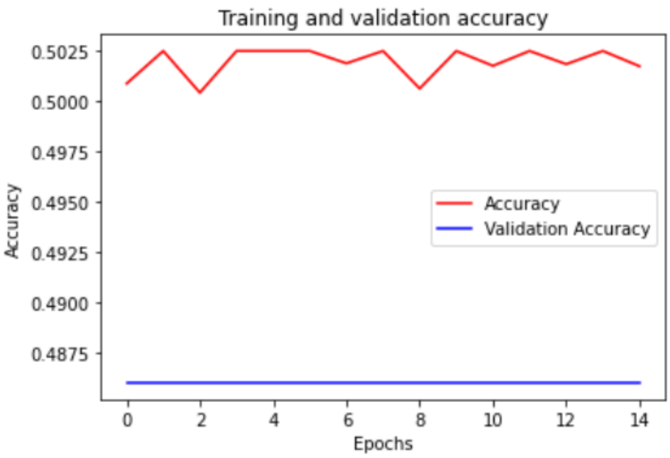

Fig 23 - Accuracy graph

The below graph illustrates the comparison between loss and validation loss in each epoch.

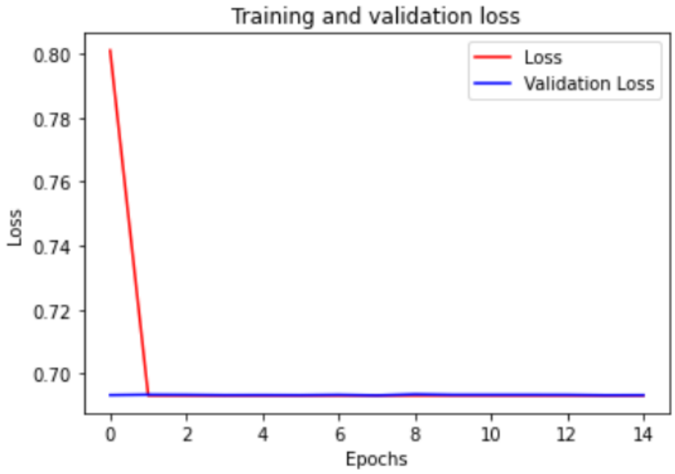

Fig 24 - Loss graph

3. LONG SHORT TERM MEMORY (LSTM) USING WORD2VEC EMBEDDING

METHODOLOGY

Long Short Term Memory (LSTM) has layers - Embedding layer , Dropout layer and Dense layer. The embedding layer is given inputs by the pre-trained Word2Vec model. The tweets are tokenized and provided as inputs to the pre-trained Word2Vec model. The LSTM is trained for 8 epochs and the batch size is set as 738. Dropout layer is used in case of overfitting. The model is trained and tested. The accuracy obtained is 80.20 %.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 300, 300)          62101500
_________________________________________________________________
dropout (Dropout)            (None, 300, 300)          0
_________________________________________________________________
lstm (LSTM)                  (None, 100)               160400
_________________________________________________________________
dense (Dense)                (None, 1)                 101
=================================================================
Total params: 62,262,001
Trainable params: 160,501
Non-trainable params: 62,101,500
```

Fig 25 - Layers of embedded LSTM and their output dimensions

The model is evaluated and prediction of tweets is done.The report of the classified tweet is generated.Finally the model is saved.

BLOCK DIAGRAM

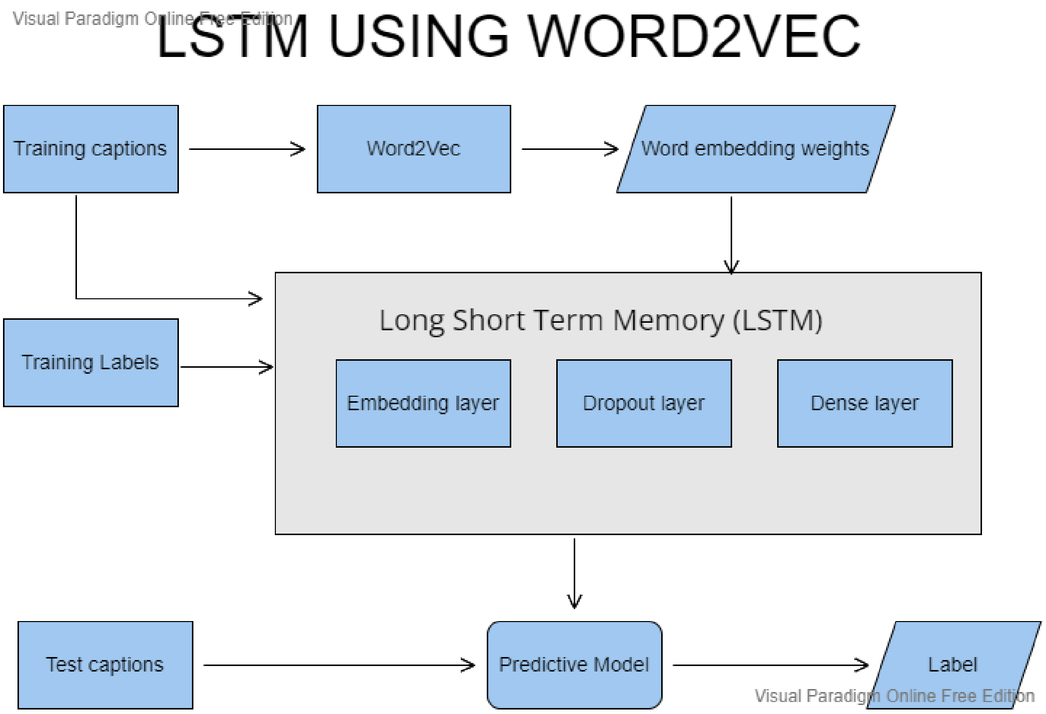

Fig 26 - Block diagram

RESULTS

The model runs for eight epochs and produces the following results:

Accuracy - 80.20%

```
Epoch 1/8
738/738 [==============================] - 857s 1s/step - loss: 0.4431 - accuracy: 0.7752 - val_loss: 0.3946 - val_accuracy: 0.7912
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
2022-06-12 09:23:36,678 : WARNING : Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 2/8
738/738 [==============================] - 840s 1s/step - loss: 0.4089 - accuracy: 0.7873 - val_loss: 0.3761 - val_accuracy: 0.7973
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
2022-06-12 09:37:36,947 : WARNING : Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 3/8
738/738 [==============================] - 821s 1s/step - loss: 0.3966 - accuracy: 0.7909 - val_loss: 0.3679 - val_accuracy: 0.7955
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
2022-06-12 09:51:17,675 : WARNING : Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 4/8
738/738 [==============================] - 850s 1s/step - loss: 0.3904 - accuracy: 0.7917 - val_loss: 0.3628 - val_accuracy: 0.7995
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
2022-06-12 10:05:27,589 : WARNING : Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 5/8
738/738 [==============================] - 851s 1s/step - loss: 0.3827 - accuracy: 0.7929 - val_loss: 0.3546 - val_accuracy: 0.8019
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
2022-06-12 10:19:38,501 : WARNING : Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 6/8
738/738 [==============================] - 845s 1s/step - loss: 0.3773 - accuracy: 0.7932 - val_loss: 0.3495 - val_accuracy: 0.7986
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
2022-06-12 10:33:43,943 : WARNING : Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 7/8
738/738 [==============================] - 855s 1s/step - loss: 0.3703 - accuracy: 0.7932 - val_loss: 0.3415 - val_accuracy: 0.8024
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
2022-06-12 10:47:58,568 : WARNING : Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
Epoch 8/8
738/738 [==============================] - 836s 1s/step - loss: 0.3617 - accuracy: 0.7929 - val_loss: 0.3310 - val_accuracy: 0.7988
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
2022-06-12 11:01:55,071 : WARNING : Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy,lr
CPU times: user 2h 41min 52s, sys: 22min 10s, total: 3h 4min 2s
Wall time: 1h 52min 35s
```

```
205/205 [==============================] - 22s 106ms/step - loss: 0.3280 - accuracy: 0.8021

ACCURACY: 0.8020933270454407
LOSS: 0.32804572582244873
```

Fig 27 - Training the model and results obtained.

	precision	recall	f1-score	support
NEGATIVE	0.80	0.78	0.79	159494
POSITIVE	0.79	0.80	0.79	160506
accuracy			0.79	320000
macro avg	0.79	0.79	0.79	320000
weighted avg	0.79	0.79	0.79	320000

Fig 28 - Results obtained

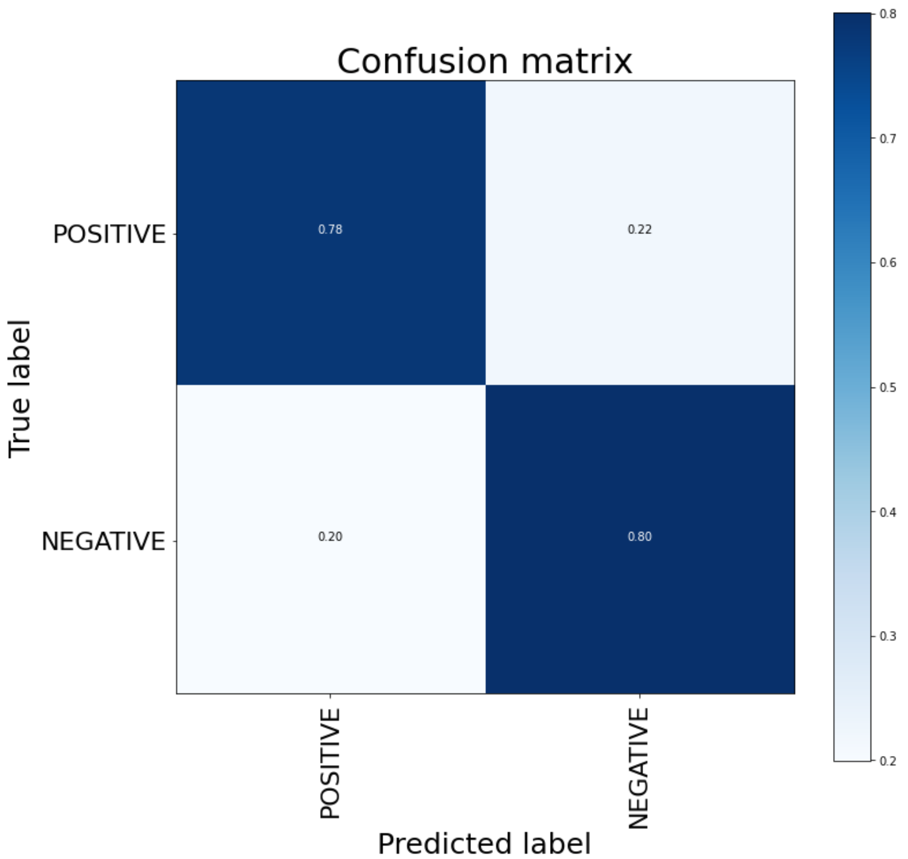

Fig 29 - Confusion Matrix

Training and validation Accuracy graph:

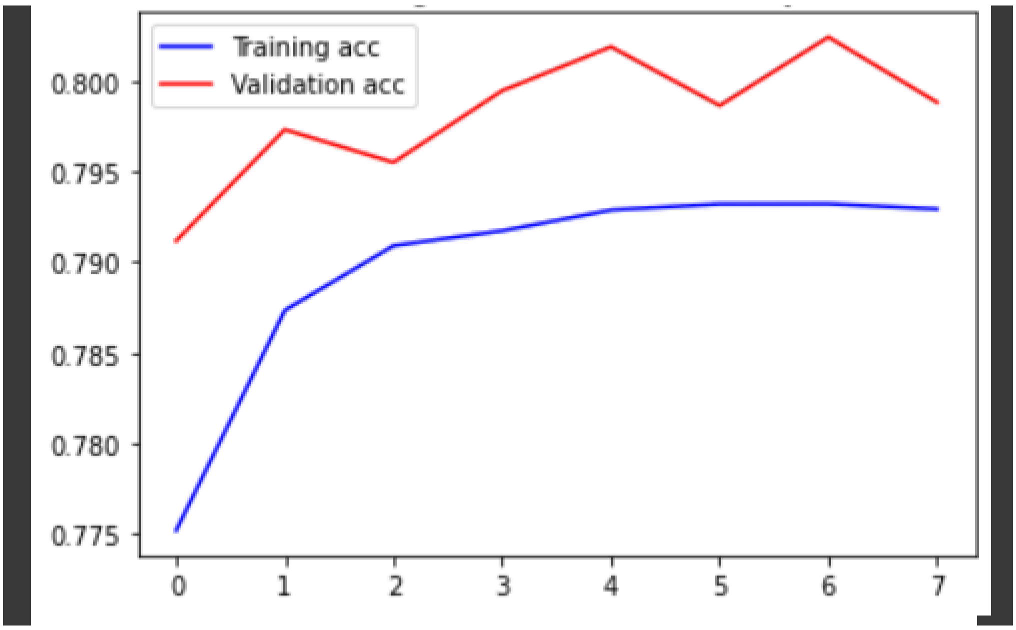

Fig 30 - Training and validation accuracy graph

Training and validation loss graph:

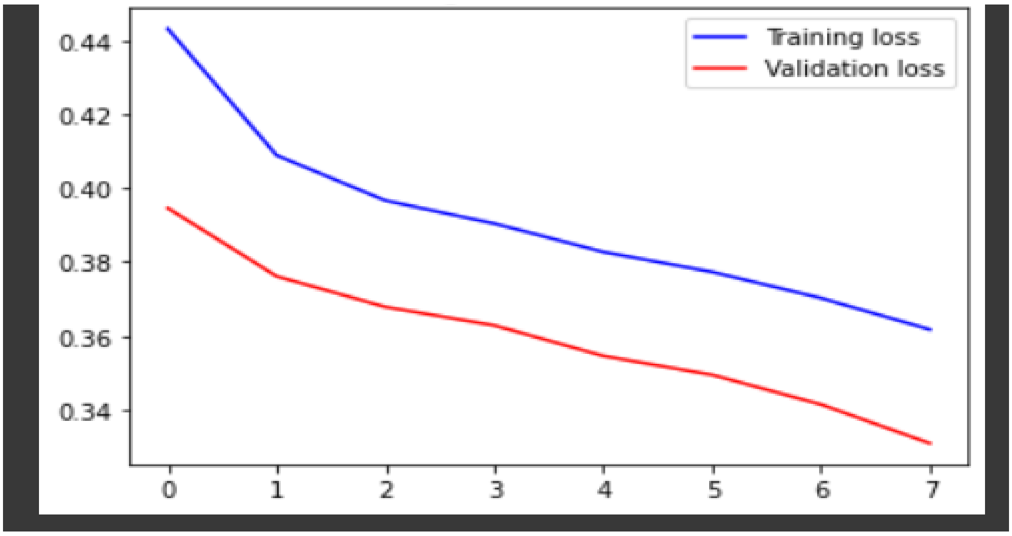

Fig 31- Training and validation loss graph

DISCUSSION

The above results show that the LSTM is better than CNN. LSTM is trained to recognize patterns across time, while a CNN learns to recognize patterns across space. This shows the fact that LSTM models can capture long-term dependencies between word sequences hence they are better for text classification. LSTM is good for text classification because it has a good hold over memorizing certain patterns. As with every other NN, LSTM can have multiple hidden layers and as it passes through every layer, the relevant information is kept and all the irrelevant information gets discarded in every single cell. Hence the above results is a proof for better text classification by LSTM.