17/06/2021

SWE 599

Deep Learning Application for Emotion Recognition From Text

Rohat Vural

2019719180

ABSTRACT

In today's world, the popularity of artificial intelligence has increased considerably. One of the most popular topics in the artificial intelligence world is the deep learning architecture inspired by neural networks. The use of artificial intelligence and deep learning models has increased and continues to increase in daily life. One of the areas where deep learning models are used in is natural language processing and deep learning models have proven their success in this area. In this project, the Bidirectional LSTM deep learning model was trained to classify texts according to the emotions of joy, sadness, fear, anger, love, and surprise. In this article, you can find the stages from the preprocessing of the data to the training of the model and the test results in detail.

Keywords: Deep Learning Application, Bidirectional LSTM, Emotion Classification, Text Processing

Table of Content

1 Introduction

Artificial intelligence is a very broad field that was first mentioned in the 1950s. Artificial intelligence consists of many different sub-classes. Deep learning which is a form of machine learning, is one of these sub-classes. Deep learning is an artificial intelligence function developed inspired by the way the human brain processes and learns information[1].In the twentieth century with the advancement of technology, deep learning applications have become an important part of our life and it has brought effective solutions to the problems faced by human beings in various fields since its entry into our lives. Deep learning facilitates the life of human beings in many areas such as object recognition, classification, decision making and image processing, and so on. In addition, natural language processing(NLP) is one of the popular areas of deep learning. Areas such as translation systems, questions and answering (Q&A) systems, information retrieval (IR), text summarization systems, sentiment analysis (SA), are the areas where the use of NLP stands out[2]. We can easily define sentiment analysis as analysis that aims to detect positive, neutral, or negative emotions[3]. Emotion detection and recognition is a field of research that is related to sentiment analysis[3]. Emotion detection and recognition aim to get more granular data than sentiment analysis, such as anger, fear, love, sadness, happiness, disgust, etc. In this study, a brief information of the use area of emotion recognition and detecting from text applications will be given and a Bidirectional LSTM model will be developed to classify texts in terms of emotions.

2 Use Areas

Emotions are vital to people's survival. The importance of emotions cannot be denied for people to express themselves and to be understood correctly by the environment. Emotions also provide observers with important information about the person himself and his current situation. Below you can find some applications where emotion detection from text is being used and can be used:

- Marketing firms or content providers can better understand the expectations of their customers and develop their products or contents accordingly by analyzing the emotions in users' comments about the products they offer.
- Governments can measure the happiness levels of their citizens from social media posts.
- What makes people happy and what makes them angry can be analyzed and individually customized ads can be created.
- The emotional state of suicidal people can be followed through social media posts and suicide attempts can be detected beforehand.
- By detecting insulting, humiliating, and intense anger posts, individuals with a tendency to commit violence and crime can be identified and crimes can be prevented before they are committed.

3 Design

3.1 Sequence Diagram

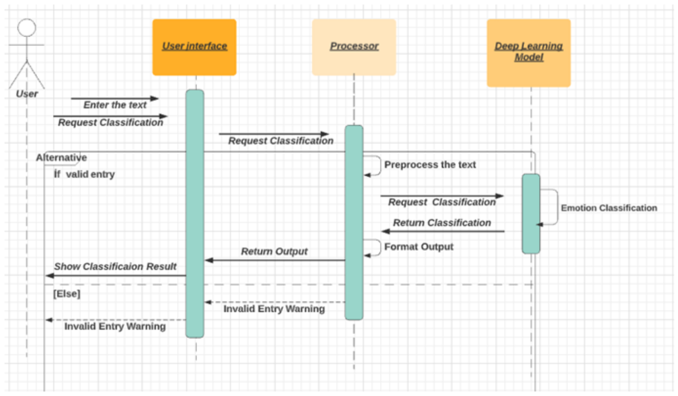
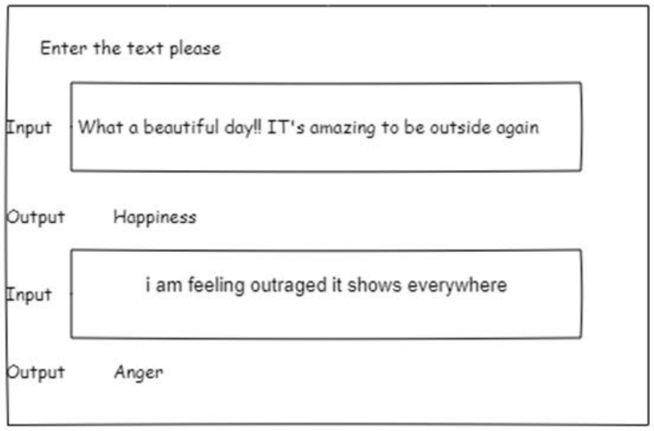


Figure 1 - Sequence Diagram of Application

3.2 User Interface Mockup


Enter the text please

Input What a beautiful day!! IT's amazing to be outside again

Output Happiness

Input i am feeling outraged it shows everywhere

Output Anger

Figure 2 – User Interface Mockup

4 Data Exploration

In order to find or create the data set to be used, the data sets available on the internet and available for use were examined. Below you can find some of the data sets examined.

4.1 Emotions In Texts

There are two columns in this data set. The first column is text and the second column is emotion. There is a total of 21459 rows of data. In the following table, you can find the categories of emotions and the number of the data for each emotion in this data set.

Emotion	Number of data
happy	7029
sadness	6265
anger	2993
fear	2652
love	1641
surprise	879

Table 1 – Label counts in emotions in texts data set

4.2 Emotion Detection from text

The data set consists of the tweets that are annotated with the emotions. There are 3 three columns in this data set which are tweet_id, sentiment, and content.

There is a total of 40000 rows of data. In the following table, you can find the categories of emotions and the number of the data for each emotion in this data set.

Sentiment	Number of data
neutral	8638
worry	8459
happiness	5209
sadness	5165
love	3842
surprise	2187
fun	1776
relief	1526
hate	1323
empty	827
enthusiasm	759
boredom	179
anger	110

Table 2 – Label counts in emotion detection from texts data set

4.3 Emotion Dataset For NLP

This data set consists of 3 different txt files as training, validation, and testing. While there are 1999 texts in the test and verification sets, there are 15999 texts in the training set.

In the following table, you can find the categories of emotions and the number of the data for each emotion in this data set.

Emotion	Training data	Validati on data	Test data
joy	5362	704	695
sadness	4665	549	580
anger	2159	275	275
fear	1937	212	224
love	1304	178	159
surprise	572	81	66

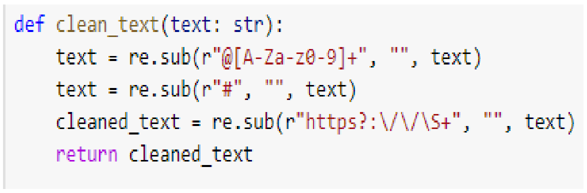
Table 3 – Label counts in emotion dataset for NLP data set

After examining the data sets, it was observed that the label classes and label accuracy of the data in the “Emotion Dataset For NLP” data set were slightly better than the other data sets, and therefore it was decided to be used.

5 Preparing The Data For Use

5.1 Cleaning The Text

In order not to feed the model with irrelevant information, first the cleaning process was applied to the data. The function created to implement this cleaning process deletes the mentions and URLs in the text. It also deletes the hashtag symbol from hashtags and keeps the rest of the word. Because the words in hashtags can contain emotions (e.g. #happiness) and we do not want to lose these words. In addition, stop words are not deleted from the text as they may be important for emotion classification from the text.



```
def clean_text(text: str):
    text = re.sub(r"@[A-Za-z0-9]+", "", text)
    text = re.sub(r"#", "", text)
    cleaned_text = re.sub(r"https?:\/\/\S+", "", text)
    return cleaned_text
```

Figure 3 – clean_text function.

Example input:
Today is a great day, https://picture.com #joy #hap piness

Example output:Today is a great day, joy happiness

5.2 Data Validation

In order to increase the trust in the data and the labels, we might use sentiment analysis. For example, we expect the sentiment analysis result of the data labeled as "joy" and "love" to be positive, and the result of the sentiment analysis of the data labeled as "sadness", "anger" and "fear" to be negative. Removing the data with a negative sentiment analysis result when the tag is a positive emotion, or the data with a positive sentiment analysis result when the tag is a negative emotion, from the data set will leave a more reliable data set. The python nltk library was used for sentiment analysis.

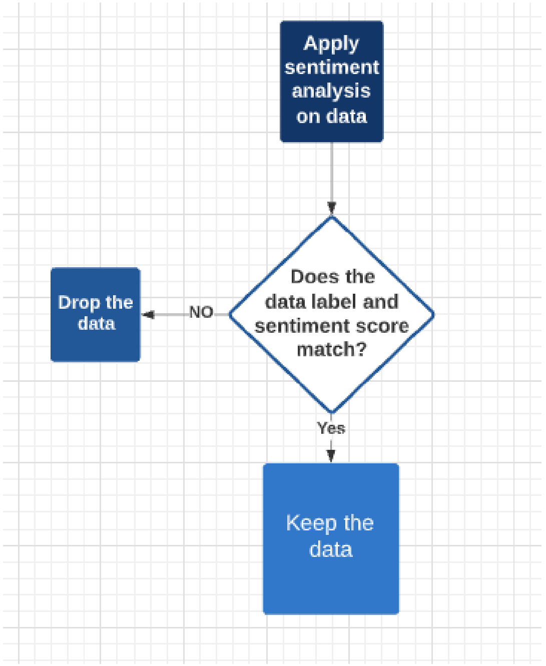

Apply sentiment analysis on data
Does the data label and sentiment score match?
NO
Drop the data
Yes
Keep the data

Figure 4 – flow diagram of data validation

This data validation process has been applied on the training_data, validation_data, and the test_data respectively. In the following figures you can find the label distribution of datasets after the validation process.

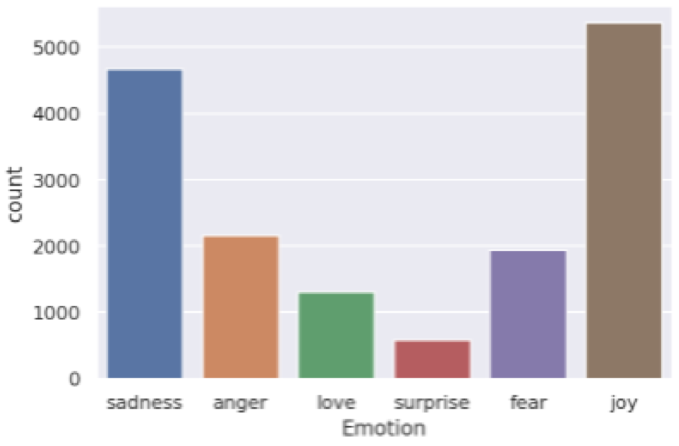


Figure 5 – Label distribution in training_data

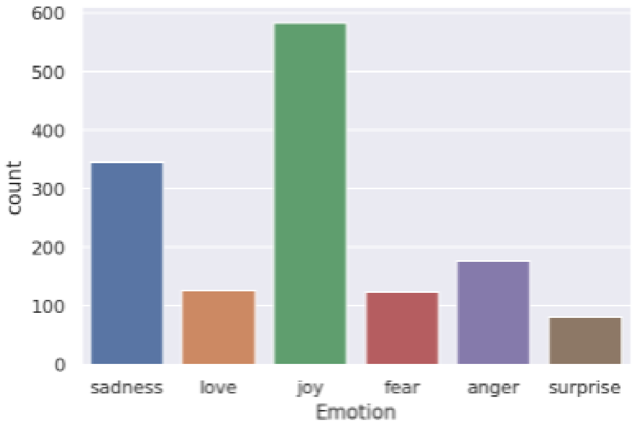


Figure 6 – Label distribution in validation_data

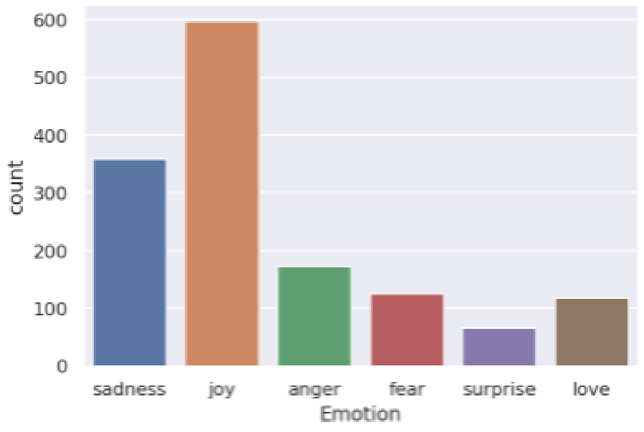


Figure 7 – Label distribution in test_data

5.3 Converting Texts into the Vectors

Deep learning models can not be fed with raw text. The text must be turned into vectors in order to use it as input. To handle this process Keras library has been used.

5.3.1 Tokenization

As the first step of converting the text into a vector, sentences should split into words.

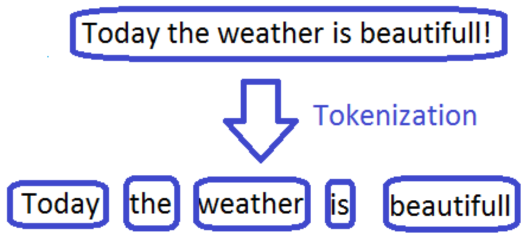
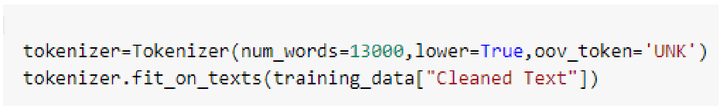


Figure 8 – Tokenization

For this process, the Tokenizer function of keras has been used.



```
tokenizer=Tokenizer(num_words=13000,lower=True,oov_token='UNK')
tokenizer.fit_on_texts(training_data["Cleaned Text"])
```

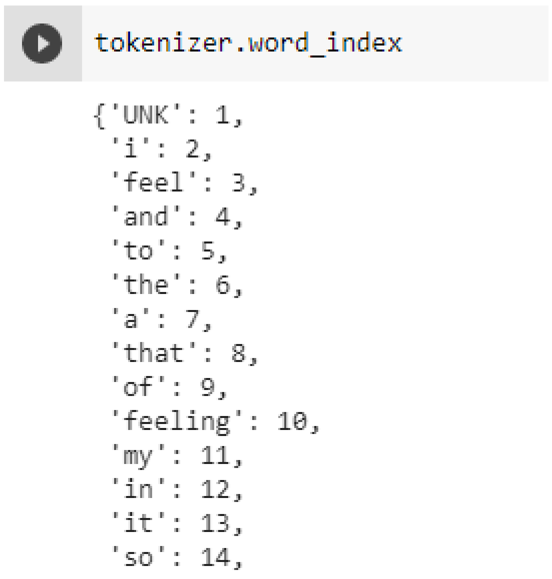
Figure 9 – Use of Tokenizer function

num_words: This number defines the maximum number of words that are going to be kept in the word index. We define it as 13000 because in our data set we have 12709 different words and setting this option as 13000 will be enough to keep all the words.

lower: converts the text into the lower case.

oov_token: out of vocabulary words will be replaced with the given pattern to keep in word_index

Tokenizer function creates a dictionary and keeps each word in the data as keys of dictionaries and as a value, it gives an index number.



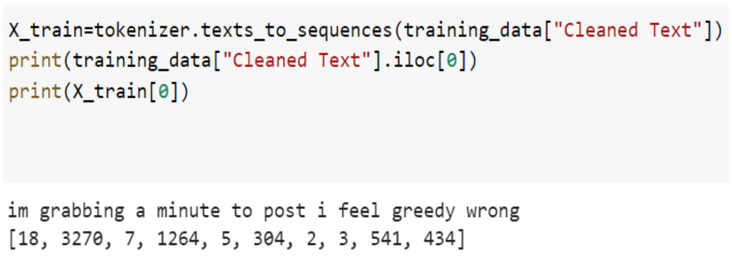
```
tokenizer.word_index
```

```
{'UNK': 1,
 'i': 2,
 'feel': 3,
 'and': 4,
 'to': 5,
 'the': 6,
 'a': 7,
 'that': 8,
 'of': 9,
 'feeling': 10,
 'my': 11,
 'in': 12,
 'it': 13,
 'so': 14,
```

Figure 10 – output of the tokenizer function as word_index

5.3.2 Texts to Sequences

After the tokenization process, we got our word_index which keeps every word in the data set with an indexing number. Now we can turn our sentences into the vectors with this indexing numbers. For converting sentences into vectors text_to_sequances function of the Keras tokenizer has been used.



```
X_train=tokenizer.texts_to_sequences(training_data["Cleaned Text"])
print(training_data["Cleaned Text"].iloc[0])
print(X_train[0])
```

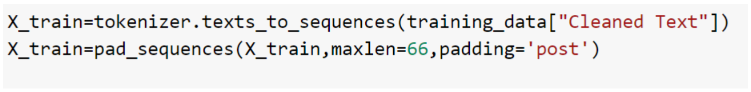
```
im grabbing a minute to post i feel greedy wrong
[18, 3270, 7, 1264, 5, 304, 2, 3, 541, 434]
```

Figure 11 – output of text to sequences function

From the figure, we can see an example output of the text_to_sequences function. The sentence “im grabbing a minute to post I feel greedy wrong” is converted into a vector, based on the indexing number of each word.

5.3.3 Padding

In the data set, the sentences have a different count of words which means after converting the sentences into vectors we will have different vector/input shapes. We should avoid having various vector shapes. In order to handle this problem pad_sequences function of the Keras tokenizer has been used.

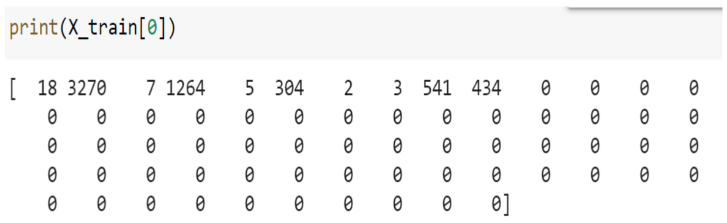


```
X_train=tokenizer.texts_to_sequences(training_data["Cleaned Text"])
X_train=pad_sequences(X_train,maxlen=66,padding='post')
```

Figure 12 – use of pad sequence function

maxlen: we defined maxlen as 66. This is the count of the words in the longest sentence of our data set. Any sentences that has less than 66 words in it will be padded with zeroes after being converted to a vector.

padding: this option can be set as “pre” or “post”. This option determine whether to add zeroes after the sequence or before the sequence. We set it as “post”.



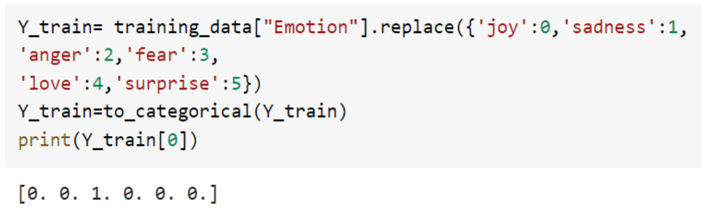
```
print(X_train[0])
```

```
[  18 3270    7 1264    5  304    2    3  541  434    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0]
```

Figure 13 – output of pad_sequence function

5.3.4 One-Hat Encode the Labels

We have categorical data as labels. Our labels consist of “joy”, “sadness”, “anger”, “fear”, “love” and “surprise”. Although there are some algorithms that can work with categorical data, most of the machine learning algorithms require numeric input and output variables[13]. That is why we are going to convert our labels into One-Hat representation. As a first step, we are going to make intiger encoding, which means each unique label will be assigned as an integer. After integer encoding, one-hat encoding will be applied on the integer representation of labels. For this process, the to_categorical function of keras.utils will be used.



```
Y_train= training_data["Emotion"].replace({'joy':0,'sadness':1,
'anger':2,'fear':3,
'love':4,'surprise':5})
Y_train=to_categorical(Y_train)
print(Y_train[0])
```

```
[0. 0. 1. 0. 0. 0.]
```

Figure 14 – One-Hat Encoding the labels

6 Deep Learning Model

Our data consists of sentences. In sentences, the meaning of any word can change according to the meaning of the words that exist before or after that word. For example, the book word in “I did not read that book” and “I want to book that room” has different meanings. It is a good choice to use Bidirectional LSTMs for sequential data where both past and future values are also important.

To define Bidirectional LSTM simply, we can say that it is putting 2 independent LSTM together. By using Bidirectional LSTM our inputs will be processed in two ways. The first layer is running the data from backward to forward and preserve past informations, the second layer is running the data from forward to backward and preserve future information. So at any point in time, both past and future information are available.

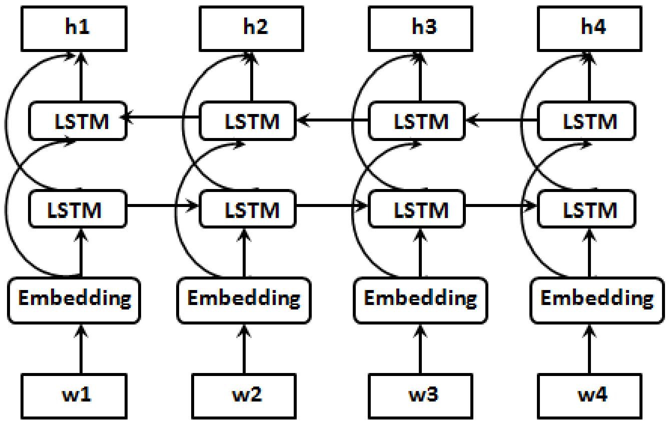
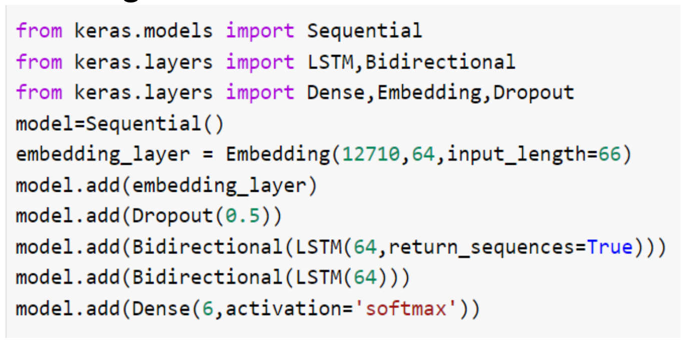


Figure 15 – a deep learning model with embedding and Bidirectional LSTM layers

6.1 Creating The Model



```
from keras.models import Sequential
from keras.layers import LSTM,Bidirectional
from keras.layers import Dense,Embedding,Dropout
model=Sequential()
embedding_layer = Embedding(12710,64,input_length=66)
model.add(embedding_layer)
model.add(Dropout(0.5))
model.add(Bidirectional(LSTM(64,return_sequences=True)))
model.add(Bidirectional(LSTM(64)))
model.add(Dense(6,activation='softmax'))
```

Figure 16 – Creating the model

6.1.1 Adding Embedding Layer

When we prepared our data, we convert each sentence in the data set into the word vectors which consist of positive integers. But we can’t directly feed the model with those integers, before that we should convert each word that is represented as integer into the vector. Using

One-Hat representation just like we did on labels would be useless here because the vocabulary size of our dataset is 12709 and it would require a very large storage space to use one-hat representation.

Embedding layers converts positive integer indexes in to the fixed size dense vectors. By using embedding layer we can convert each word index to a fixed size vector.

- **input_dim**: This option determines the size of the vocabulary. We have 12709 different words in our data set.
- **output_dim**: This option determines the size of a vector for each word. It can be another integer value. Should be tested for different problems.
- **input_length**: This option determines the length of the input sequence. While preparing our data we convert each sentence in the data set into sequences with a length of 66.

6.1.2 Adding Dropout Layer

When neural networks are trained with relatively small datasets, they are more likely to experience overfitting. One of the regularization techniques used to avoid this problem is dropout. Dropout is a method that makes regularization by randomly setting the incoming inputs to equate to 0 at a specified “rate” and enlarges the inputs that are not equated to 0 by “1-rate”, without changing the total input value. The dropout layer is applied on the input only in training mode, when we evaluate the model with the test data or when we make real-time predictions the dropout will not be applied.

rate: It is observed that usually values between 0.5 to 0.8 are being used as dropouts in the models. We got good results by setting the rate to 0.5. Different results can be achieved with different values.

6.1.3 Adding Bidirectional LSTM Layers

In the beginning of deep learning model, what is the Bidirectional LSTMs and why it is selected to be used has been explained.

As a structure, stacked 2 Bidirectional LSTM layer has been added to the model.

memory unit: There is no specific rule for determining the memory units that should be used. We achieved good results by using 64 memory units in each LSTM layer.

return_sequance: This option is defaultly set as False**.** In order to feed the 2nd Bidirectional LSTM layer with the output of the 1st Bidirectional LSTM layer, return_sequence has been set as True in for 1st Bidirectional LSTM layer. By setting return_sequance as True, we can access to hidden state outputs for each time step.

activation_function: For Bidirectional LSTM Tanh(hyperbolic tangent) or ReLU could be used as activation function. While making literature search it is observed that researchers usually prefer Tanh for NLP problems. The default activation function for the LSTM layer in Keras is Tanh. That is why setting this option is not required.

6.1.4 Adding Dense Layer

This is our output layer. Since we have a classification problem with the 6 categories, the unit number is set to 6. Unit number defines the dimesion of the output space. In multi-class classification problesm it is recommended to use softmax as activation function, thats why we chose softmax as activation function of the output layer.

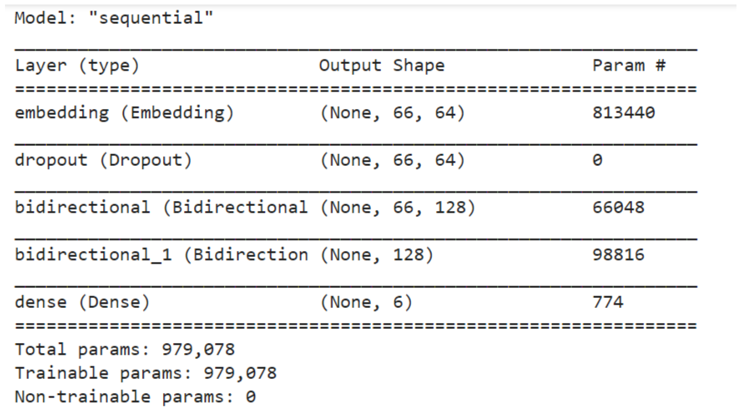

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 66, 64)            813440
_________________________________________________________________
dropout (Dropout)            (None, 66, 64)            0
_________________________________________________________________
bidirectional (Bidirectional (None, 66, 128)           66048
_________________________________________________________________
bidirectional_1 (Bidirection (None, 128)               98816
_________________________________________________________________
dense (Dense)                (None, 6)                 774
=================================================================
Total params: 979,078
Trainable params: 979,078
Non-trainable params: 0
```

Figure 17 – summary of the model

After adding all these layers we created a sequantial model which consist of one embedding layer, one dropout layer, two bidirectional LSTM layes and one output layer. The model has total trainable 979078 parameters.

6.2 Compiling The Model

Loss function: When we prepared our data we convert our labels into the one-hat representation. The categorical-crossentropy function should be used as loss function where there are 2 or more label classes and these label classes are represented in one-hat format. Categorical_crossentropy function computes the loss between prediction and label.

Optimizer: As optimizer i selected Adam. Adam is robost and the most popular optimizer. It is working well even with little tunning of hyper parameters.

6.3 Training the Model

After creating the model and defining the hyper parameters, now it's time to train the model. Deciding where to stop training is a chalange in training neural networks. If we train less our model can have underfitting problem. If we train more than necessary our model can have overfitting problem. In order to have good fit model we should stop training where validation accuracy starts to degrade while training accuracy continues to rising. In order to understand where is that point training started with 20 epochs.

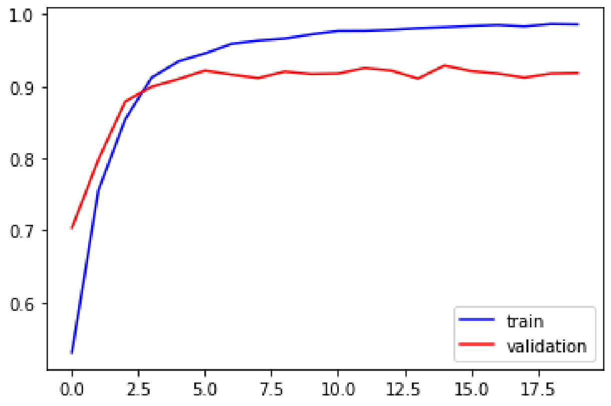


Figure 18 – training and validation accuracy for 20 epoch

As we can see from the figure after the 14th Epoch validation accuracy stop improvement but training accuracy continues to rise. In order to prevent overfitting to training data, the model will be trained from begining for 14 Epoch.

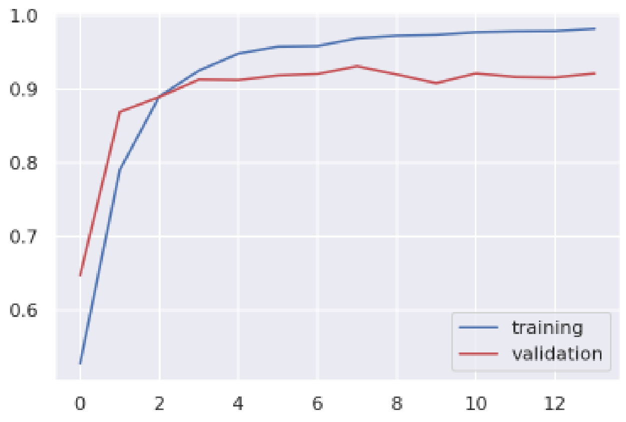
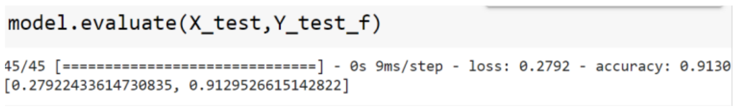


Figure 19 – training and validation accuracy for 14 epoch

6.4 Evaluating the model

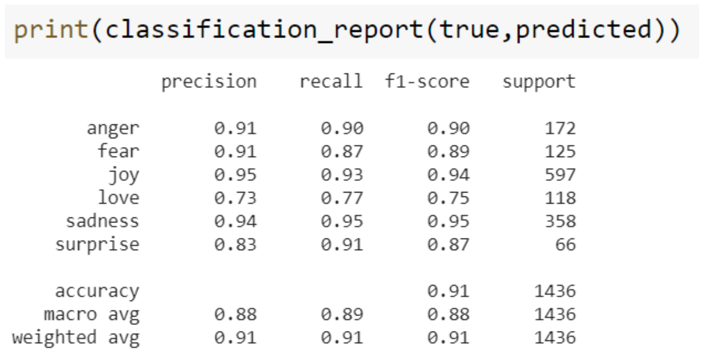
After the training of the model was completed, it was run on the test data to evaluate its real performance.



```
model.evaluate(X_test,Y_test_f)
```

```
45/45 [==============================] - 0s 9ms/step - loss: 0.2792 - accuracy: 0.9130
[0.27922433614730835, 0.9129526615142822]
```

Figure 20 – Model Evaulation



```
print(classification_report(true,predicted))
```

```
              precision    recall  f1-score   support

       anger       0.91      0.90      0.90       172
        fear       0.91      0.87      0.89       125
         joy       0.95      0.93      0.94       597
        love       0.73      0.77      0.75       118
     sadness       0.94      0.95      0.95       358
    surprise       0.83      0.91      0.87        66

    accuracy                           0.91      1436
   macro avg       0.88      0.89      0.88      1436
weighted avg       0.91      0.91      0.91      1436
```

Figure 21 – Classification Accuracy Report

Precision: It is a metric that shows how many of the predictions of our model was actually matching with the true label.

$$\text{Precision} = \frac{\mathit{True\ Positive}}{\mathit{True\ Positive} + \mathit{False\ Positive}}$$

Figure 22 – Computation of precision

Recall: it is a metric that shows how many true labels have been captured by our model.

$$\text{Recall} = \frac{\mathit{True\ Positive}}{\mathit{True\ Positive} + \mathit{False\ Negative}}$$

Figure 23 – Computation of recall

As we can see from the classification report, our model gets a good result with an accuracy of around 90% in classifying emotions such as joy, sadness, fear and anger on the test data. In contrast, our model classifies feelings of love and surprise with relatively low accuracy. In the data set we selected, the least number of training data was the data belonging to the love and surprise classes. Given this information, it is not surprising that the accuracy of these two classes is lower than the other classes.

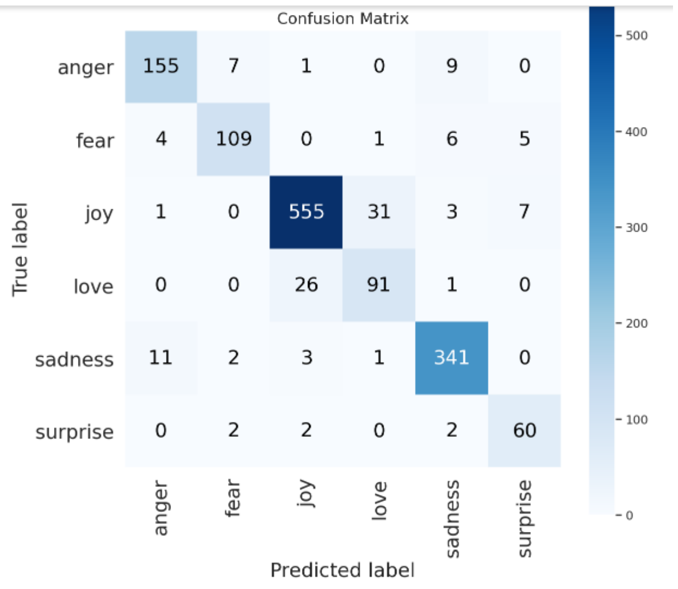
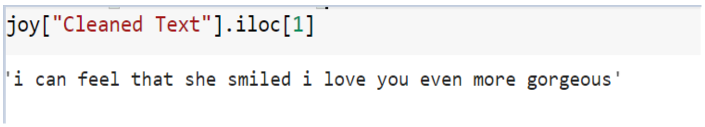


Figure 24 – Confusion matrix

When the confusion matrix is examined, we see that our model classifies the data belonging to joy, sadness, anger and fear classes well. On the other hand, 28.5% of the data belonging to the love class were classified as joy. When the reason for this is examined, it has been observed that the training data of the joy class

contains some data that should be actually belong to the love class. It is a known fact that incorrectly labeled data causes the model to learn incorrectly.



```
joy["Cleaned Text"].iloc[1]
```

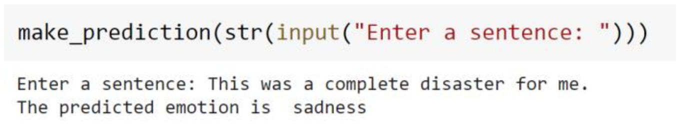
```
'i can feel that she smiled i love you even more gorgeous'
```

Figure 25 – Example of Miss Labeled Data

When the confusion matrix is continued to be examined, it is observed that the data belonging to the surprise class can actually be classified well, but some data that should actually be classified as joy or fear are also classified as surprise by our model. It has been observed that the reason for this is the same as in the love class.

6.5 Testing With Real Data

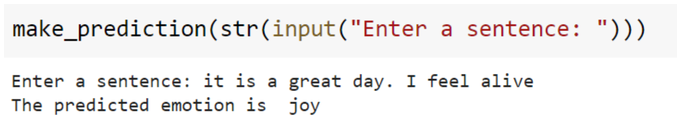
After receiving the evaluation results of the model, I wanted to test the model with the data entered by the user. In to following figures you can find some examples.



```
make_prediction(str(input("Enter a sentence: ")))

Enter a sentence: This was a complete disaster for me.
The predicted emotion is  sadness
```

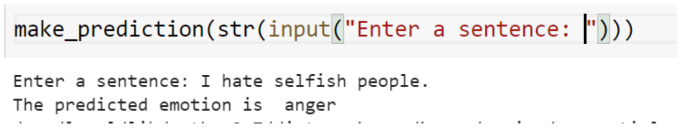
Figure 26 – test with user input 1



```
make_prediction(str(input("Enter a sentence: ")))

Enter a sentence: it is a great day. I feel alive
The predicted emotion is  joy
```

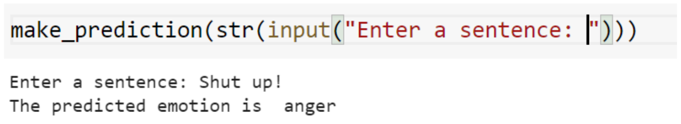
Figure 27 – test with user input 2



```
make_prediction(str(input("Enter a sentence: ")))

Enter a sentence: I hate selfish people.
The predicted emotion is  anger
```

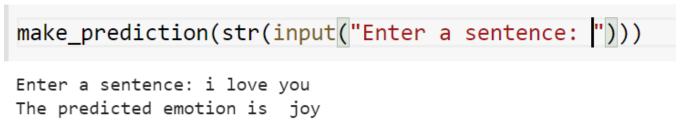
Figure 28 – test with user input 3



```
make_prediction(str(input("Enter a sentence: ")))

Enter a sentence: Shut up!
The predicted emotion is  anger
```

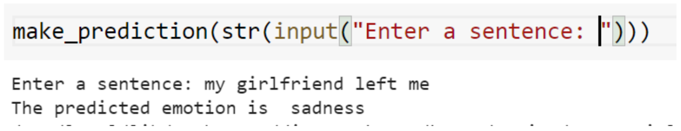
Figure 29 – test with user input 4



```
make_prediction(str(input("Enter a sentence: ")))

Enter a sentence: i love you
The predicted emotion is  joy
```

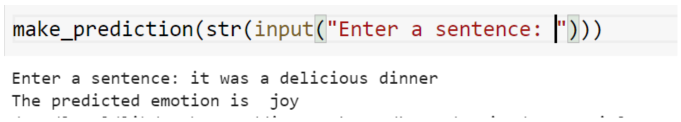
Figure 30 – test with user input 5



```
make_prediction(str(input("Enter a sentence: ")))

Enter a sentence: my girlfriend left me
The predicted emotion is  sadness
```

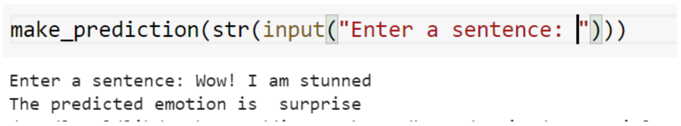
Figure 31 – test with user input 6



```
make_prediction(str(input("Enter a sentence: ")))

Enter a sentence: it was a delicious dinner
The predicted emotion is  joy
```

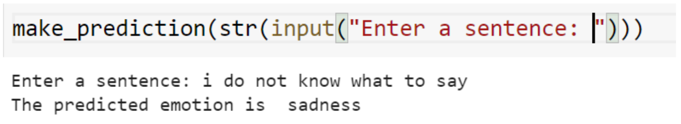
Figure 32 – test with user input 7



```
make_prediction(str(input("Enter a sentence: ")))

Enter a sentence: Wow! I am stunned
The predicted emotion is  surprise
```

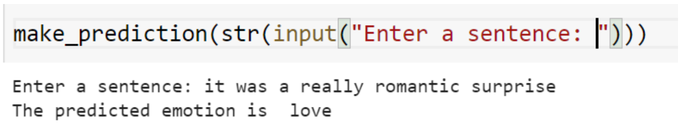
Figure 33 – test with user input 8



```
make_prediction(str(input("Enter a sentence: ")))

Enter a sentence: i do not know what to say
The predicted emotion is  sadness
```

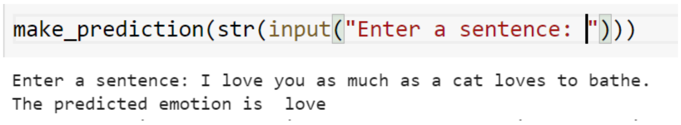
Figure 34 – test with user input 9



```
make_prediction(str(input("Enter a sentence: ")))

Enter a sentence: it was a really romantic surprise
The predicted emotion is  love
```

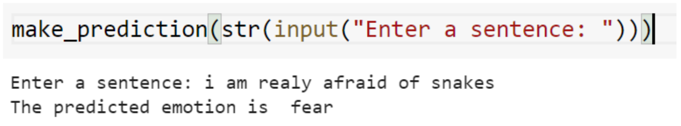
Figure 35 – test with user input 10



```
make_prediction(str(input("Enter a sentence: ")))

Enter a sentence: I love you as much as a cat loves to bathe.
The predicted emotion is  love
```

Figure 36 – test with user input 11



```
make_prediction(str(input("Enter a sentence: ")))

Enter a sentence: i am realy afraid of snakes
The predicted emotion is  fear
```

Figure 37 – test with user input 12

During the test of the model with the sentences entered by the user, it was observed that the model classified the emotion classes with high accuracy well, whereas it was not very successful in classifying emotions such as love and surprise, which had relatively low accuracy. In addition, it was also observed that the model was not successful in classifying sentences containing irony in terms of emotion.

7 Conclusion

In this project, by using Keras library I created a deep learning model with Bidirectional LSTM architecture that will classify text into the 6 emotion. In the project, not only the model was trained, but also the data pre-preparation processes were applied in detail. As a result of the studies and tests, it was seen that although the model could not classify the emotions of love and surprise very well, it classified the emotions of joy, anger, sadness and fear well. When the reasons for the emotions that the model classified with relatively low accuracy were examined, it was seen that the distribution of the classes in the data set was not proportional and the data belonging to these classes were very few. In addition, it was seen that incorrectly labeled data were also included in the data set. Nevertheless, the model achieved a satisfactory level of overall accuracy and met the expectation on user-input data.

8 Future Works

As future work, the model can be retrained and higher accuracy rates can be achieved by finding a larger data set and more proportionally distributed classes.

Another future work can be better elimination of incorrectly labeled data from data set. I expect that this will make a significant contribution to the model to produce more accurate results.

Finally, models such as BERT, which have gained popularity in the field of natural language processing in recent years and give good results, can be used.

References

[1] Deep Learning

Marshall Hargrave

https://www.investopedia.com/terms/d/deep-learning.asp#what-is-deep-learning

[2] Text-based emotion detection: Advances, challenges, and opportunities. Francisca Adoma Acheampong, Chen Wenyu, Henry Nunoo-Mensah

https://onlinelibrary.wiley.com/doi/10.1002/eng2.12189

[3] Emotion Detection and Recognition from Text Using Deep Learning

Chew-Yean

https://devblogs.microsoft.com/cse/2015/11/29/emotion-detection-and-recognition-from-text-using-deep-learning/

[4] From Sentiment Analysis to Emotion Recognition: A NLP story

Rodrigo Masaru Ohashi

https://medium.com/neuronio/from-sentiment-analysis-to-emotion-recognition-a-nlp-story-bcc9d6ff61ae

[5] EMOTION DETECTION FROM TEXT

Shiv Naresh Shivhare and Prof. Saritha Khethawat

[6] Classify emotions in text with BERT NLP model

Praveen Govindaraj

https://www.kaggle.com/praveengovi/classify-emotions-in-text-with-bert

[7] Emotion Classification using Bidirectional LSTM

Deepak Vedantam

https://www.kaggle.com/deepakvedantam/emotion-classification-using-bidirectional-lstm

[8] Tweet Emotions Analysis using-LSTM-Glove-RoBERTA

Ouassim Adnane

https://www.kaggle.com/ishivinal/tweet-emotions-analysis-using-lstm-glove-roberta

[9] NLP Beginner - Text Classification using LSTM

Arun Pandian R

https://www.kaggle.com/arunrk7/nlp-beginner-text-classification-using-lstm

[10] How to Prepare Text Data for Deep Learning with Keras

Jason Brownlee

https://machinelearningmastery.com/prepare-text-data-deep-learning-keras/

[11] A Gentle Introduction to Dropout for Regularizing Deep Neural Networks

Jason Brownlee

https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

[12] Bi-LSTM

Raghav Aggarval

https://medium.com/@raghavaggarwal0089/bi-lstm-bc3d68da8bd0

[13] Why One-Hot Encode Data in Machine Learning?

Jason Brownlee

https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

[14] How Can I Change A Neural Network From “Regression” to “Classification” ?

Jason Brownlee

https://machinelearningmastery.com/faq/single-faq/how-can-i-change-a-neural-network-from-regression-to-classification/

[15] How to Use Word Embedding Layers for Deep Learning with Keras

Jason Brownlee

https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/