Configuration Manual

MSc Research Project
MSc Data Analytics

Kshitija Kiran Manore
Student ID: x20191308

School of Computing
National College of Ireland

Supervisor: Dr Catherine Mulwa

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Kshitija Kiran Manore
Student ID:	x20191308
Programme:	MSc Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr Catherine Mulwa
Submission Due Date:	15/08/2022
Project Title:	Configuration Manual
Word Count:	1337
Page Count:	23

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:	Kshitija Kiran Manore
Date:	14th August 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

Kshitija Kiran Manore
x20191308

1 Introduction

How to execute the developed scripts for the current study subject is described in the configuration document. This will guarantee error-free operation of the code. This provides the same specified minimum need as well as details about the hardware configuration of the machine on which the programs are run. Following these steps will make it easier to reproduce the project's results. This may then be examined, making it simple to do more study.

2 System Specification

2.1 Hardware configuration

The system's hardware specifications, which are listed below, are as follows:

Processor: Ryzen 7 – 8265U CPU @ 1.60GHz
RAM: 8 GB
Storage: 1TB SSD
Operating System: 64-bit operating system, Windows 11

Python (version 3.6.9) was used to performthe task's execution since it has a wealth of readily importable library modules. Its deployment made advantage of both the local workstation and the Google online services. The on-site PC was a 64-bit Windows 11 laptop including an 8GB RAM and Ryzen 7 CPU. Because Step 2 required more processing power as well as a graphics processing unit, the evaluation was conducted on a local workstation (GPU).

2.2 Software configuration

The Google Compute Engine serves as the foundation for all computing activities on the Google Cloud Platform, which is essentially an Infrastructure as a Service (IaaS). Google provides it. The configuration was set up to make advantage of the 2496 CUDA cores, 12GB of RAM, and 1xTesla K80 available GPU for the length of the execution. The GPU service was restricted to a total of twelve hours per day because the cloud hosting was simply a free service. The model training procedure therefore took around a week.

3 Downloads and Installation

- **Python**

Python is utilized in this research study because of the abundance of libraries, machine learning models, plus deep learning tools it offers. Additionally, it has a number of modules that facilitate pre-processing and image alteration, making it easier to use and put into practice. As a result, it is essential that the machine running the script has the most recent version of Python downloaded. To achieve this, go to the Python website's download link at [1] and download the installer for the chosen version based upon that machine's operating system. Fig. 1 displays a snapshot of the website where the most recent version may be downloaded.

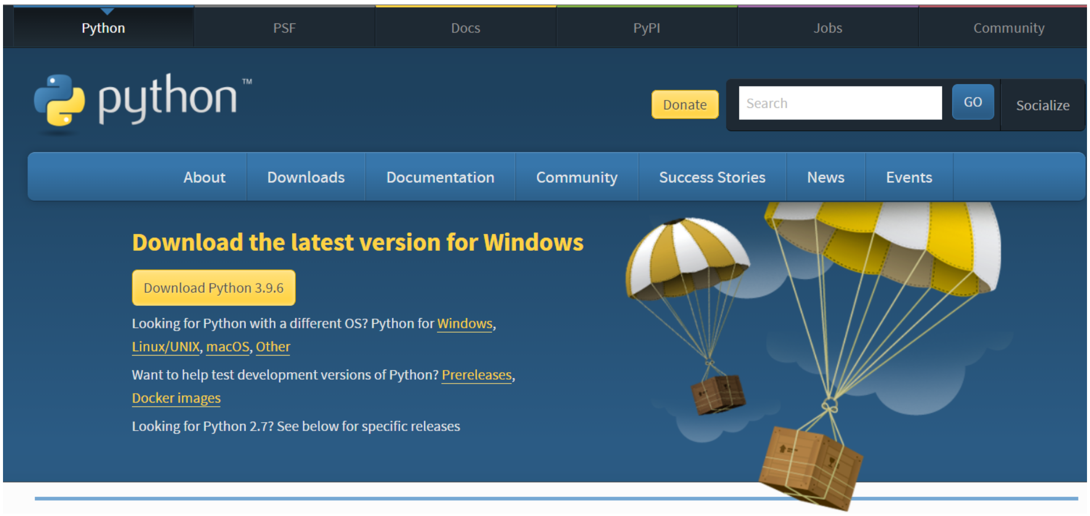


Figure 1. Download page of python

By using 'python -version' command on the Command line, you may check whether the installation was successful. You can find out what version of Python is installed there.

- **Data Source**

The data for this study was collected from Amazon's online reviews[2] of gourmet foods. It includes a variety of information, including the user's identity, user ID, product information, ratings, the text of the user reviews, and a reference summary of those user reviews. All of the aforementioned data was retrieved and put together into csv file for later use. The total dataset contains around 570,000 reviews, and depending on the computing capacity or resource we have available, we may choose sequence data of 50,000 or 100,000 entries for our application.

- **Project Development**

Additional Python modules will be required as necessary because the project uses transfer learning-based machine as well as deep learning methodologies. You may install

[1] https://www.python.org/downloads/
[2] https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews

them by using pip install at the Windows command - line interface, as seen in the example below.

- TensorFlow
- Keras
- Pandas
- RE
- Numpy
- OS
- BeautifulSoup
- Tokenizer
- pad_Sequences
- Stopwords
- Warnings
- Wget
- NLTK
- Pickle
- Stringcode
- Unicodedata
- Randint
- Seaborn
- Matplotlib
- Wordcloud
- SKlearn
- contractions
- Rouge-Score

```
!pip install tensorflow-gpu==1.15
!pip install keras==2.2.4
!pip install numpy==1.19.5
#import keras==2.2.4
import numpy as np
import pandas as pd
import re
import os
from bs4 import BeautifulSoup
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from nltk.corpus import stopwords
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense, Concatenate, TimeDistributed, Bidirectional
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping
import warnings
pd.set_option("display.max_colwidth", 200)
warnings.filterwarnings("ignore")
!pip install wget
import wget
import nltk
```

Figure 2. Necessary Libraries-1

```
import os
import re
import pickle
import string
import unicodedata
from random import randint

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from nltk.corpus import stopwords
from wordcloud import STOPWORDS, WordCloud

from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras import Input, Model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.layers import LSTM, Bidirectional, Dense, Embedding, TimeDistributed
from tensorflow.keras.models import load_model
```

```
!pip install -q contractions==0.0.48
!pip install rouge-score
```

Figure 3. Necessary Libraries-2

4 Code

- **Preprocessing of Data**
- **Defining the Contraction Dictionary**

```
from contractions import contractions_dict

for key, value in list(contractions_dict.items())[:10]:
    print(f'{key} == {value}')
```

```
I'm == I am
I'm'a == I am about to
I'm'o == I am going to
I've == I have
I'll == I will
I'll've == I will have
I'd == I would
I'd've == I would have
Whatcha == What are you
amn't == am not
```

Figure 4

Figure 4 shows declaration of contraction dictionary.

- **Expanding the Contractions**

```
def expand_contractions(text, contraction_map=contractions_dict):
    # Using regex for getting all contracted words
    contractions_keys = '|'.join(contraction_map.keys())
    contractions_pattern = re.compile(f'({contractions_keys})', flags=re.DOTALL)

    def expand_match(contraction):
        # Getting entire matched sub-string
        match = contraction.group(0)
        expanded_contraction = contraction_map.get(match)
        if not expand_contractions:
            print(match)
            return match
        return expanded_contraction

    expanded_text = contractions_pattern.sub(expand_match, text)
    expanded_text = re.sub("'", "", expanded_text)
    return expanded_text


expand_contractions("y'all can't expand contractions i'd think")

'you all can not expand contractions id think'
```

Figure 5

Figure 5 shows the expansion of contractions.

- **Removing Punctuation Marks**

```
# Remove puncuation from word
def rm_punc_from_word(word):
    clean_alphabet_list = [
        alphabet for alphabet in word if alphabet not in string.punctuation
    ]
    return ''.join(clean_alphabet_list)

print(rm_punc_from_word('#cool!'))


# Remove puncuation from text
def rm_punc_from_text(text):
    clean_word_list = [rm_punc_from_word(word) for word in text]
    return ''.join(clean_word_list)

print(rm_punc_from_text("Frankly, my dear, I don't give a damn"))

cool
Frankly my dear I dont give a damn
```

Figure 6

Figure 6 shows the removal of punctuation marks.

- **Removing Numbers**

```
# Remove numbers from text
def rm_number_from_text(text):
    text = re.sub('[0-9]+', '', text)
    return ' '.join(text.split())  # to rm `extra` white space

print(rm_number_from_text('You are 100times more sexier than me'))
print(rm_number_from_text('If you taught yes then you are 10 times more delusional than me'))

You are times more sexier than me
If you taught yes then you are times more delusional than me
```

Figure 7

Figure 7 shows the removal of the numbers.

- **Removing Stopwords**

```
# Remove stopwords from text
def rm_stopwords_from_text(text):
    _stopwords = stopwords.words('english')
    text = text.split()
    word_list = [word for word in text if word not in _stopwords]
    return ' '.join(word_list)

rm_stopwords_from_text("Love means never having to say you're sorry")

'Love means never say sorry'
```

Figure 8

Figure 8 shows the removal of the stop words.

- **Saving the data after Preprocessing**

```
# saving the cleaned data
df.to_csv('/content/drive/MyDrive/customer_review_summarizer/Data/cleaned_data.csv')
```

Figure 9

Figure 9 shows saving of the data after preprocessing.

- **Creating a Word Cloud**

Figure 10

Figure 10 shows the aspects in a WordCloud.

- **Rare word Analysis**

```
# rare word analysis
def get_rare_word_percent(tokenizer, threshold):
    # threshold: if the word's occurrence is less than this then it's rare word

    count = 0
    total_count = 0
    frequency = 0
    total_frequency = 0

    for key, value in tokenizer.word_counts.items():
        total_count += 1
        total_frequency += value
        if value < threshold:
            count += 1
            frequency += value

    return {
        'percent': round((count / total_count) * 100, 2),
        'total_coverage': round(frequency / total_frequency * 100, 2),
        'count': count,
        'total_count': total_count
    }
```

Figure 11

Figure 11 shows the rare word analysis. The word which has less occurrence is a rare word.

- **Splitting the dataset**

```python
# Splitting the training and validation sets
x_train, x_val, y_train, y_val = train_test_split(
    np.array(df['text']),
    np.array(df['summary']),
    test_size=0.1,
    random_state=1,
    shuffle=True
)
```

Figure 12

Figure 12 shows splitting the data in training and testing sets.

- **Tokenizing**

```python
x_tokenizer = Tokenizer()
x_tokenizer.fit_on_texts(list(x_train))

x_tokens_data = get_rare_word_percent(x_tokenizer, 4)
print(x_tokens_data)
```

```
{'percent': 71.52, 'total_coverage': 2.37, 'count': 48321, 'total_count': 67561}
```

```python
x_tokenizer = Tokenizer(num_words=x_tokens_data['total_count'] - x_tokens_data['count'])
```

```python
# else use this
x_tokenizer = Tokenizer()
x_tokenizer.fit_on_texts(list(x_train))
```

```python
# save tokenizer
with open('/content/drive/MyDrive/customer_review_summarizer/x_tokenizer', 'wb') as f:
    pickle.dump(x_tokenizer, f, protocol=pickle.HIGHEST_PROTOCOL)
```

Figure 13

Figure 13 shows the code for tokenizing the words.

- **LSTM Model**

```python
def build_seq2seq_model_with_just_lstm(
    embedding_dim, latent_dim, max_text_len,
    x_vocab_size, y_vocab_size,
    x_embedding_matrix, y_embedding_matrix
):
  # instantiating the model in the strategy scope creates the model on the TPU
   # =====================
        # =====================
      encoder_input = Input(shape=(max_text_len, ))

        # encoder embedding layer
      encoder_embedding = Embedding(
            x_vocab_size,
            embedding_dim,
            embeddings_initializer=tf.keras.initializers.Constant(x_embedding_matrix),
            trainable=False
        )(encoder_input)

        # encoder lstm 1
      encoder_lstm1 = LSTM(
            latent_dim,
            return_sequences=True,
            return_state=True,
            dropout=0.4,
            recurrent_dropout=0.4
        )
      encoder_output1, state_h1, state_c1 = encoder_lstm1(encoder_embedding)

        # encoder lstm 2
      encoder_lstm2 = LSTM(
            latent_dim,
```

Figure 14

Figure 14 shows building of the LSTM Model.

- **Summary of LSTM Model**

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 100)]        0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 100, 300)     20268600    input_1[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________________
lstm (LSTM)                     [(None, 100, 240), ( 519360      embedding[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, None, 300)    4467900     input_2[0][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, 100, 240), ( 461760      lstm[0][0]
__________________________________________________________________________________________________
lstm_2 (LSTM)                   [(None, None, 240),  519360      embedding_1[0][0]
                                                                 lstm_1[0][1]
                                                                 lstm_1[0][2]
__________________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 14893)  3589213     lstm_2[0][0]
==================================================================================================
Total params: 29,826,193
Trainable params: 9,557,593
Non-trainable params: 20,268,600
__________________________________________________________________________________________________
```

Figure 15

Figure 15 shows the summary for LSTM model.

- **Epochs of LSTM**

```
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/tensorflow_core/python/ops/math_grad.py:1424: where (from tensorflow.python.op
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Train on 84048 samples, validate on 9339 samples
Epoch 1/8
84048/84048 [==============================] - 119s 1ms/sample - loss: 1.9503 - acc: 0.7505 - val_loss: 1.5707 - val_acc: 0.7883
Epoch 2/8
84048/84048 [==============================] - 117s 1ms/sample - loss: 1.6272 - acc: 0.7818 - val_loss: 1.5058 - val_acc: 0.7915
Epoch 3/8
84048/84048 [==============================] - 117s 1ms/sample - loss: 1.5604 - acc: 0.7845 - val_loss: 1.4457 - val_acc: 0.7934
Epoch 4/8
84048/84048 [==============================] - 127s 2ms/sample - loss: 1.5087 - acc: 0.7864 - val_loss: 1.4099 - val_acc: 0.7949
Epoch 5/8
84048/84048 [==============================] - 115s 1ms/sample - loss: 1.4731 - acc: 0.7882 - val_loss: 1.3918 - val_acc: 0.7961
Epoch 6/8
84048/84048 [==============================] - 116s 1ms/sample - loss: 1.4432 - acc: 0.7899 - val_loss: 1.3634 - val_acc: 0.7983
Epoch 7/8
84048/84048 [==============================] - 117s 1ms/sample - loss: 1.4158 - acc: 0.7916 - val_loss: 1.3439 - val_acc: 0.7995
Epoch 8/8
84048/84048 [==============================] - 118s 1ms/sample - loss: 1.3925 - acc: 0.7930 - val_loss: 1.3322 - val_acc: 0.8003
```

Figure 16

Figure 16 shows the 8 epochs for LSTM model.

- **Accuracy of LSTM model for Text Summarization**

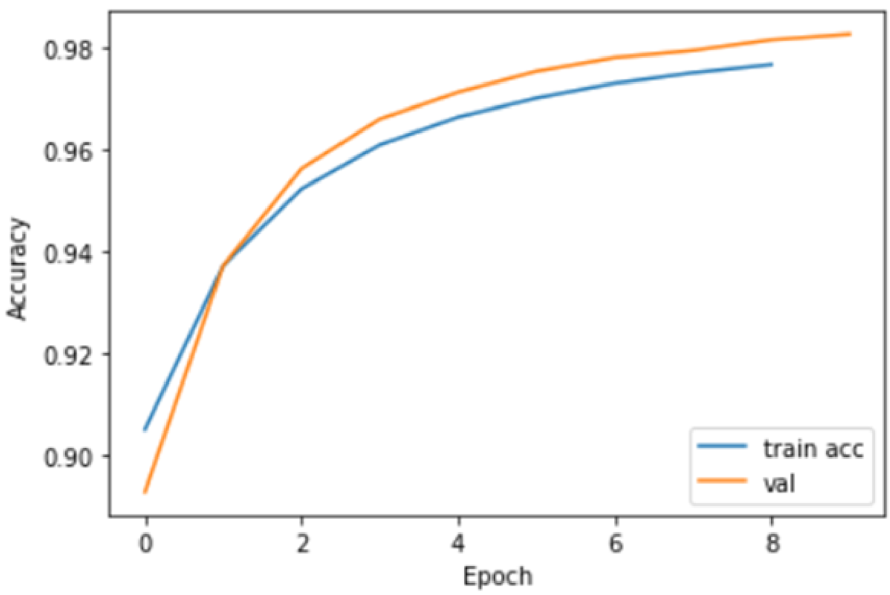


Figure 17

Figure 17 shows a graph for accuracy of training and testing data.

- **Loss of LSTM model for Text Summarization**

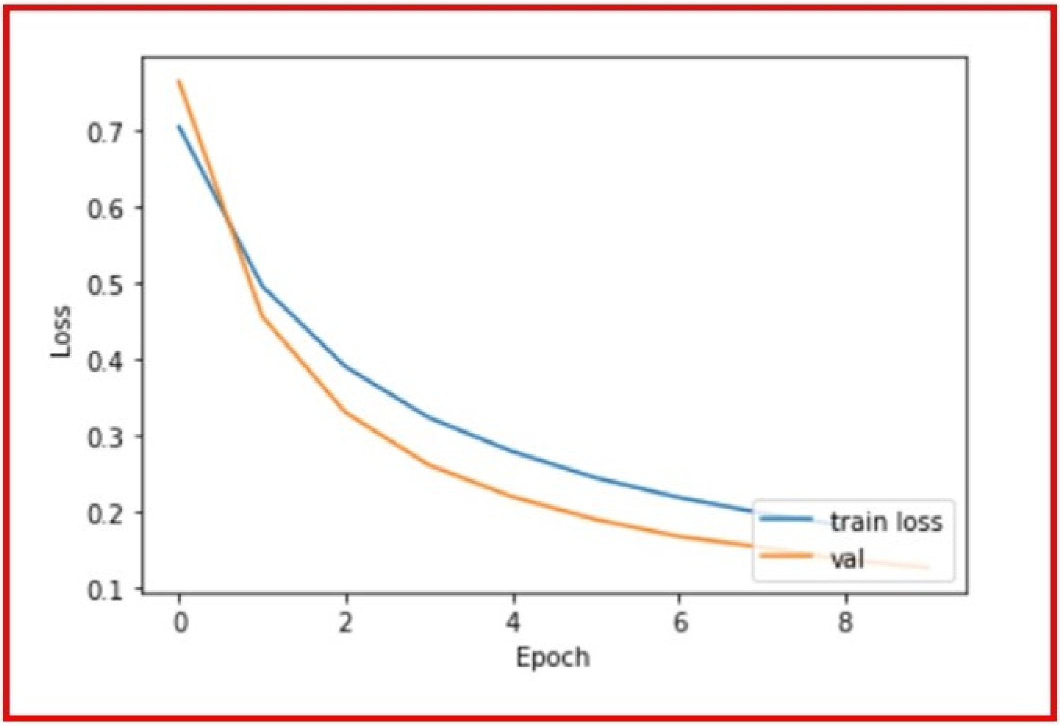


Figure 18

Figure 18 shows a graph for loss of training and testing data

- **Inference LSTM model**

Inference LSTM

```
[ ] # Next, let's build the dictionary to convert the index to word for target and source vocabulary:
    reverse_target_word_index = y_tokenizer.index_word
    reverse_source_word_index = x_tokenizer.index_word
    target_word_index = y_tokenizer.word_index

[ ] def build_seq2seq_model_with_just_lstm_inference(
        max_text_len, latent_dim, encoder_input, encoder_output,
        encoder_final_states, decoder_input, decoder_output,
        decoder_embedding_layer, decoder_dense, last_decoder_lstm
    ):
        # Encode the input sequence to get the feature vector
        encoder_model = Model(
            inputs=encoder_input, outputs=[encoder_output] + encoder_final_states
        )

        # Decoder setup
        # Below tensors will hold the states of the previous time step
        decoder_state_input_h = Input(shape=(latent_dim, ))
        decoder_state_input_c = Input(shape=(latent_dim, ))
        decoder_hidden_state_input = Input(shape=(max_text_len, latent_dim))

        # Get the embeddings of the decoder sequence
        decoder_embedding = decoder_embedding_layer(decoder_input)

        # To predict the next word in the sequence, set the initial
        # states to the states from the previous time step
        decoder_output, *decoder_states = last_decoder_lstm(
            decoder_embedding,
```

Figure 19

Figure 19 shows the inference LSTM model.

- **Creating a Predicting Model**

```python
def predict_text(text, decode_sequence, encoder_model, decoder_model):
    original_text = text
    text = clean_text([text])  # generator
    text_list = original_text.split()

    if len(text_list) <= max_text_len:
        text = expand_contractions(text)
        text = clean_text(text)
        text = f'_START_ {text} _END_'
        text = f'{start_token} {text} {end_token}'

        seq = x_tokenizer.texts_to_sequences([' '.join(text_list)])
        padded = pad_sequences(seq, maxlen=max_text_len, padding='post')
        pred_summary = decode_sequence(
            padded.reshape(1, max_text_len), encoder_model, decoder_model
        )
        return pred_summary
    else:
        pred_summary = ''

        # breaking long texts to individual max_text_len texts and predicting on them
        while len(text_list) % max_text_len == 0:
            text_list.append('')

        lst_i = max_text_len
        for i in range(0, len(text_list), max_text_len):
            _text_list = original_text.split()[i:i + lst_i]
            _text = ' '.join(_text_list)
            _text = ' '.join(
                _text.split()
            )  # to remove spaces that were added to make len(text_list) % max_text_len == 0
```

Figure 20

Figure 20 shows creation of a model to predict the summary.

- **Summarization with LSTM**

```
# 1 News:  great snack bought try ordering morethey taste somewhat like cross jack parmesan cheeseyou get dimesized fat discs crispy baked cheese saltybut love ththem
Original summary:  start crispy salty cheezyyum end
Predicted summary:   start great snack end

# 2 News:  one favorite flavors green tea happy find amazon local wal stopped carrying itbr br got nice peach skin essence flavor easy taste peach flavor green tea get nice strong
Original summary:  start great tasting well priced green tea end
Predicted summary:   start great tea end

# 3 News:  using cat food years vet recommended dental cleaning expensive put ththem anesthesia way cat healthy teeth sethem good condition loves taste size eat small food anymore
Original summary:  start live without cat food end
Predicted summary:   start cat loves end

# 4 News:  received selection kcups im familiar green mountain thought id try something newand good made ounce cup well flavored nice deep bitter would definitely buy one
Original summary:  start nice strong cupnot bitter end
Predicted summary:   start good coffee end

# 5 News:  compared grocery store mix sissy
Original summary:  start great deal end
Predicted summary:   start great product end

# 6 News:  want know pleased quality service company provides sincerely appreciate responsiveness way conduct business recommended company others satisfaction service product look
Original summary:  start grrrrate end
Predicted summary:   start great product end

# 7 News:  honestly say shots sometimes crutches feel like im pass im majoring mechanical engineering things phenomenal need get late night studying done need energy focus exams t
Original summary:  start college students best friend end
Predicted summary:   start great product end

# 8 News:  amazon clearly stacks sitting around warehouse months bars chalky tasted old besides though know much would like bars anyway mean love anything peanut butter bars cut t
Original summary:  start poor taste fresh end
Predicted summary:   start great snack end

# 9 News:  tasty filling easy would recommend kraft velveeta ultimate cheesburger skillets dinner body wants hand fast tasty meal
```

Figure 21

Figure 21 shows the summarization of the food reviews by the LSTM model.

- **ROUGE Score**

```python
from rouge_score import rouge_scorer
scorer = rouge_scorer.RougeScorer(['rouge1'])
results = {'precision': [], 'recall': [], 'fmeasure': []}
for (h, r) in zip(text, predicted):
    # computing the ROUGE
    score = scorer.score(h, r)
    # separating the measurements
    precision, recall, fmeasure = score['rouge1']
    # add them to the proper list in the dictionary
    results['precision'].append(precision)
    results['recall'].append(recall)
    results['fmeasure'].append(fmeasure)
```

```python
result = pd.DataFrame(results)
result
```

Figure 22

Figure 22 shows the code for ROUGE score for LSTM model.

- **ROUGE Matrix for LSTM**

	precision	recall	fmeasure
0	0.000000	0.00000	0.000000
1	0.000000	0.00000	0.000000
2	0.000000	0.00000	0.000000
3	0.000000	0.00000	0.000000
4	0.000000	0.00000	0.000000
5	0.000000	0.00000	0.000000
6	0.000000	0.00000	0.000000
7	0.000000	0.00000	0.000000
8	0.000000	0.00000	0.000000
9	0.000000	0.00000	0.000000
10	0.000000	0.00000	0.000000
11	0.000000	0.00000	0.000000
12	0.000000	0.00000	0.000000
13	0.083333	0.01087	0.019231
14	0.000000	0.00000	0.000000

Figure 23

Figure 23 shows the precision, recall and f1measure for LSTM model.

- **BDLSTM Model**

Bidirectional LSTMs

```
[ ] def build_seq2seq_model_with_bidirectional_lstm(
        embedding_dim, latent_dim, max_text_len,
        x_vocab_size, y_vocab_size,
        x_embedding_matrix, y_embedding_matrix
    ):
        # instantiating the model in the strategy scope creates the model on the TPU

          # =====================
          #  Encoder
          # =====================
        encoder_input = Input(shape=(max_text_len, ))

          # encoder embedding layer
        encoder_embedding = Embedding(
              x_vocab_size,
              embedding_dim,
              embeddings_initializer=tf.keras.initializers.Constant(x_embedding_matrix),
              trainable=False,
              name='encoder_embedding'
          )(encoder_input)

          # encoder lstm1
        encoder_bi_lstm1 = Bidirectional(
              LSTM(
                  latent_dim,
                  return_sequences=True,
                  return_state=True,
                  dropout=0.4,
                  recurrent_dropout=0.4,
```

Figure 24

Figure 24 shows building of the BDLSTM Model.

- **Summary of BDLSTM Model**

```
Model: "seq2seq_model_with_bidirectional_lstm"
____________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
====================================================================================================
input_6 (InputLayer)            [(None, 100)]        0
____________________________________________________________________________________________________
encoder_embedding (Embedding)   (None, 100, 300)     20268600    input_6[0][0]
____________________________________________________________________________________________________
encoder_bidirectional_lstm_1 (B [(None, 100, 480), ( 1038720     encoder_embedding[0][0]
____________________________________________________________________________________________________
input_7 (InputLayer)            [(None, None)]       0
____________________________________________________________________________________________________
encoder_bidirectional_lstm_2 (B [(None, 100, 480), ( 1384320     encoder_bidirectional_lstm_1[0][0
____________________________________________________________________________________________________
decoder_embedding (Embedding)   (None, None, 300)    4467900     input_7[0][0]
____________________________________________________________________________________________________
encoder_bidirectional_lstm_3 (B [(None, 100, 480), ( 1384320     encoder_bidirectional_lstm_2[0][0
____________________________________________________________________________________________________
decoder_bidirectional_lstm_1 (B [(None, None, 480),  1038720     decoder_embedding[0][0]
                                                                 encoder_bidirectional_lstm_3[0][1
                                                                 encoder_bidirectional_lstm_3[0][2
                                                                 encoder_bidirectional_lstm_3[0][3
                                                                 encoder_bidirectional_lstm_3[0][4
____________________________________________________________________________________________________
time_distributed_1 (TimeDistrib (None, None, 14893)  7163533     decoder_bidirectional_lstm_1[0][0
====================================================================================================
Total params: 36,746,113
Trainable params: 12,009,613
Non-trainable params: 24,736,500
```

Figure 25

Figure 25 shows the summary for LSTM model.

- **Epochs of BDLSTM**

```
Train on 84048 samples, validate on 9339 samples
Epoch 1/10
84048/84048 [==============================] - 311s 4ms/sample - loss: 1.4095 - acc: 0.8137 - val_loss: 0.7651 - val_acc: 0.8926
Epoch 2/10
84048/84048 [==============================] - 303s 4ms/sample - loss: 0.7058 - acc: 0.9049 - val_loss: 0.4565 - val_acc: 0.9372
Epoch 3/10
84048/84048 [==============================] - 299s 4ms/sample - loss: 0.4963 - acc: 0.9371 - val_loss: 0.3300 - val_acc: 0.9562
Epoch 4/10
84048/84048 [==============================] - 298s 4ms/sample - loss: 0.3902 - acc: 0.9522 - val_loss: 0.2611 - val_acc: 0.9660
Epoch 5/10
84048/84048 [==============================] - 303s 4ms/sample - loss: 0.3233 - acc: 0.9609 - val_loss: 0.2194 - val_acc: 0.9713
Epoch 6/10
84048/84048 [==============================] - 315s 4ms/sample - loss: 0.2790 - acc: 0.9664 - val_loss: 0.1896 - val_acc: 0.9754
Epoch 7/10
84048/84048 [==============================] - 306s 4ms/sample - loss: 0.2442 - acc: 0.9701 - val_loss: 0.1672 - val_acc: 0.9781
Epoch 8/10
84048/84048 [==============================] - 304s 4ms/sample - loss: 0.2185 - acc: 0.9730 - val_loss: 0.1530 - val_acc: 0.9795
Epoch 9/10
84048/84048 [==============================] - 305s 4ms/sample - loss: 0.1978 - acc: 0.9751 - val_loss: 0.1367 - val_acc: 0.9816
Epoch 10/10
84048/84048 [==============================] - 303s 4ms/sample - loss: 0.1810 - acc: 0.9767 - val_loss: 0.1260 - val_acc: 0.9827
```

Figure 26

Figure 26 shows the 10 epochs for BDLSTM model.

- **Accuracy of BDLSTM model for Text Summarization**

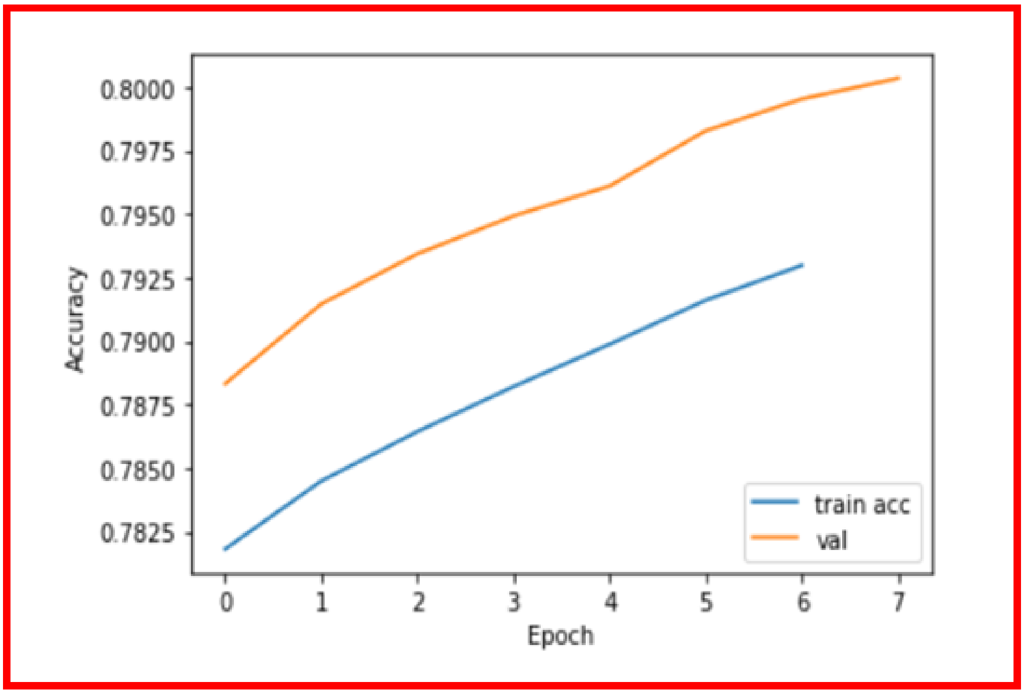


Figure 27

Figure 27 shows a graph for accuracy of training and testing data.

- **Loss of BDLSTM model for Text Summarization**

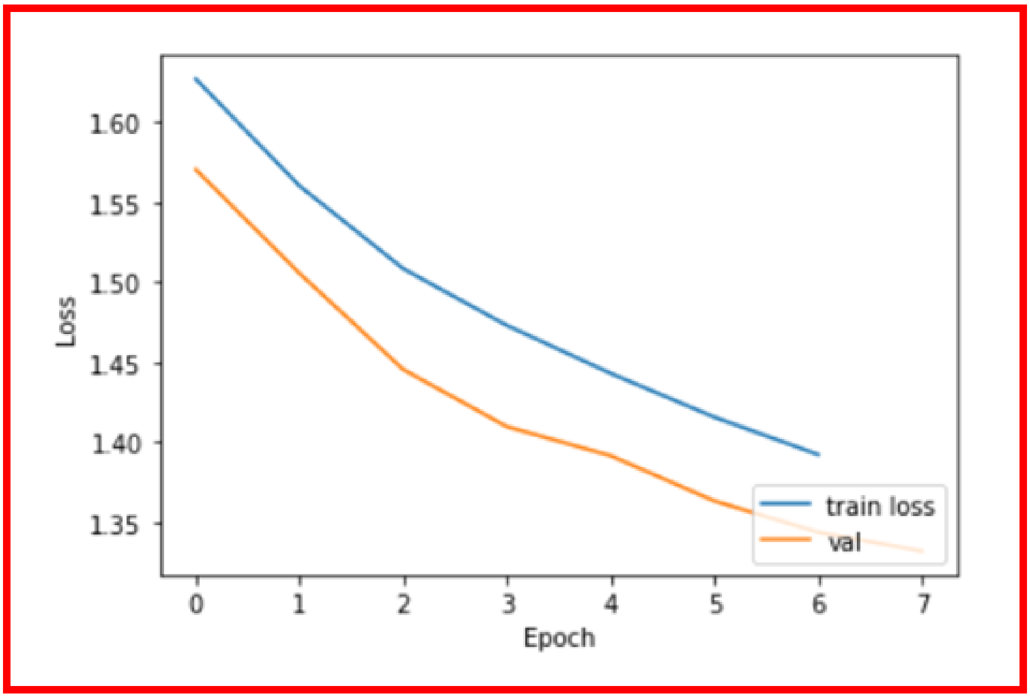


Figure 28

Figure 28 shows a graph for loss of training and testing data

- **Inference BDLSTM model**

```
# Inference
encoder_model, decoder_model = build_seq2seq_model_with_bidirectional_lstm_inference(
    max_text_len, latent_dim, encoder_input, encoder_output,
    encoder_final_states, decoder_input, decoder_output,
    decoder_embedding_layer, decoder_dense, last_decoder_lstm
)
```

```
encoder_model.summary()
```

```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
encoder_embedding (Embedding (None, 100, 300)          20268600
_________________________________________________________________
encoder_bidirectional_lstm_1 [(None, 100, 480), (None, 1038720
_________________________________________________________________
encoder_bidirectional_lstm_2 [(None, 100, 480), (None, 1384320
_________________________________________________________________
encoder_bidirectional_lstm_3 [(None, 100, 480), (None, 1384320
=================================================================
Total params: 24,075,960
Trainable params: 3,807,360
Non-trainable params: 20,268,600
_________________________________________________________________
```

Figure 29

Figure 29 shows the inference BDLSTM model.

- **Summarization with BDLSTM**

```
# 1 News:  great snack bought try ordering morethey taste somewhat like cross jack parmesan cheeseyou get dimesized fat discs crispy baked cheese saltybut love ththem
Original summary:  start crispy salty cheezyyum end
Predicted summary:   end start start start start start start start start start start start

# 2 News:  one favorite flavors green tea happy find amazon local wal stopped carrying itbr br got nice peach skin essence flavor easy taste peach flavor green tea get nice stron
Original summary:  start great tasting well priced green tea end
Predicted summary:   end start start start start start start start start start start start

# 3 News:  using cat food years vet recommended dental cleaning expensive put ththem anesthesia way cat healthy teeth sethem good condition loves taste size eat small food anymor
Original summary:  start live without cat food end
Predicted summary:   end start start start start start start start start start start start

# 4 News:  received selection kcups im familiar green mountain thought id try something newand good made ounce cup well flavored nice deep bitter would definitely buy one
Original summary:  start nice strong cupnot bitter end
Predicted summary:   end start start start start start start start start start start start

# 5 News:  compared grocery store mix sissy
Original summary:  start great deal end
Predicted summary:   end start start start start start start start start start start start

# 6 News:  want know pleased quality service company provides sincerely appreciate responsiveness way conduct business recommended company others satisfaction service product loo
Original summary:  start grrrrate end
Predicted summary:   end start start start start start start start start start start start

# 7 News:  honestly say shots sometimes crutches feel like im pass im majoring mechanical engineering things phenomenal need get late night studying done need energy focus exams
Original summary:  start college students best friend end
Predicted summary:   end start start start start start start start start start start start

# 8 News:  amazon clearly stacks sitting around warehouse months bars chalky tasted old besides though know much would like bars anyway mean love anything peanut butter bars cut
Original summary:  start poor taste fresh end
Predicted summary:   end start start start start start start start start start start start

# 9 News:  tasty filling easy would recommend kraft velveeta ultimate cheesburger skillets dinner body wants hand fast tasty meal
```

Figure 30

Figure 30 shows the summarization of the food reviews by the BDLSTM model.

- **ROUGE Score**

```python
original = []
predicted = []
text = []

for i in range(0, 15):
    text.append(seq2text(x_val_padded[i]))
    original.append(seq2summary(y_val_padded[i]))
    predicted.append(decode_sequence_seq2seq_model_with_bidirectional_lstm(x_val_padded[i].reshape(1, max_text_len), encoder_model,decoder_model))

from rouge_score import rouge_scorer
scorer = rouge_scorer.RougeScorer(['rouge1'])
results = {'precision': [], 'recall': [], 'fmeasure': []}
for (h, r) in zip(text, predicted):
    # computing the ROUGE
    score = scorer.score(h, r)
    # separating the measurements
    precision, recall, fmeasure = score['rouge1']
    # add them to the proper list in the dictionary
    results['precision'].append(precision)
    results['recall'].append(recall)
    results['fmeasure'].append(fmeasure)

result = pd.DataFrame(results)
result
```

Figure 31

Figure 31 shows the code for ROUGE score for BDLSTM model.

- **ROUGE Matrix for LSTM**

	precision	recall	fmeasure
0	0.40	0.058824	0.102564
1	0.50	0.125000	0.200000
2	0.00	0.000000	0.000000
3	0.00	0.000000	0.000000
4	0.25	0.030303	0.054054
5	0.25	0.015873	0.029851
6	0.00	0.000000	0.000000
7	0.00	0.000000	0.000000
8	0.00	0.000000	0.000000
9	0.20	0.022727	0.040816
10	0.25	0.028571	0.051282

Figure 32

Figure 32 shows the precision, recall and f1measure for LSTM model.

- **LSTM with Attention Mechanism Model**

ATTENTION LAYERS

```
[ ] #!pip install tensorflow-gpu==1.15
```

```
[ ] import tensorflow as tf
    import os
    from tensorflow.python.keras.layers import Layer
    from tensorflow.python.keras import backend as K


    class AttentionLayer(Layer):

        def __init__(self, **kwargs):
            super(AttentionLayer, self).__init__(**kwargs)

        def build(self, input_shape):
            assert isinstance(input_shape, list)
            # Create a trainable weight variable for this layer.

            self.W_a = self.add_weight(name='W_a',
                                       shape=tf.TensorShape((input_shape[0][2], input_shape[0][2])),
                                       initializer='uniform',
                                       trainable=True)
            self.U_a = self.add_weight(name='U_a',
                                       shape=tf.TensorShape((input_shape[1][2], input_shape[0][2])),
                                       initializer='uniform',
                                       trainable=True)
            self.V_a = self.add_weight(name='V_a',
                                       shape=tf.TensorShape((input_shape[0][2], 1)),
```

Figure 33

Figure 33 shows building of the LSTM with Attention Mechanism Model.

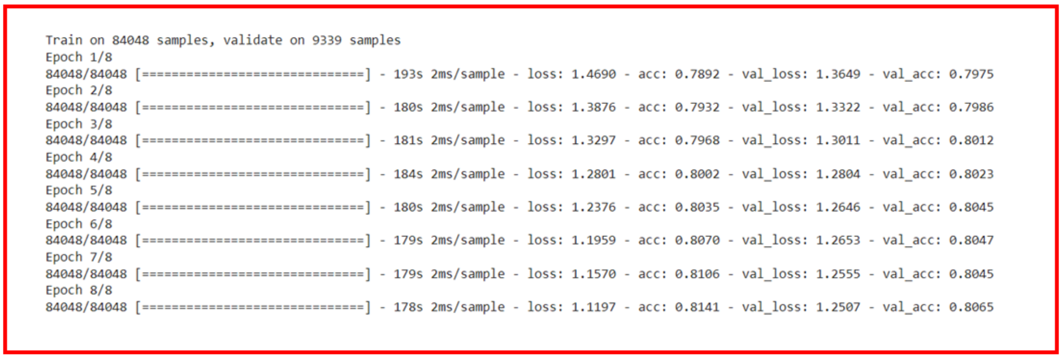
- **Summary of LSTM with Attention Mechanism Model**

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 100)]        0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 100, 300)     20268600    input_1[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                     [(None, 100, 240), ( 519360      embedding[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, 100, 240), ( 461760      lstm[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, None, 300)    4467900     input_2[0][0]
__________________________________________________________________________________________________
lstm_2 (LSTM)                   [(None, 100, 240), ( 461760      lstm_1[0][0]
__________________________________________________________________________________________________
lstm_3 (LSTM)                   [(None, None, 240),  519360      embedding_1[0][0]
                                                                 lstm_2[0][1]
                                                                 lstm_2[0][2]
__________________________________________________________________________________________________
attention_layer (AttentionLayer ((None, None, 240),  115440      lstm_2[0][0]
                                                                 lstm_3[0][0]
__________________________________________________________________________________________________
concat_layer (Concatenate)      (None, None, 480)    0           lstm_3[0][0]
                                                                 attention_layer[0][0]
__________________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 14893)  7163533     concat_layer[0][0]
==================================================================================================
Total params: 33,977,713
Trainable params: 33,977,713
```

Figure 34

Figure 34 shows the summary for LSTM with Attention Mechanism model.

- **Epochs of LSTM**



```
Train on 84048 samples, validate on 9339 samples
Epoch 1/8
84048/84048 [==============================] - 193s 2ms/sample - loss: 1.4690 - acc: 0.7892 - val_loss: 1.3649 - val_acc: 0.7975
Epoch 2/8
84048/84048 [==============================] - 180s 2ms/sample - loss: 1.3876 - acc: 0.7932 - val_loss: 1.3322 - val_acc: 0.7986
Epoch 3/8
84048/84048 [==============================] - 181s 2ms/sample - loss: 1.3297 - acc: 0.7968 - val_loss: 1.3011 - val_acc: 0.8012
Epoch 4/8
84048/84048 [==============================] - 184s 2ms/sample - loss: 1.2801 - acc: 0.8002 - val_loss: 1.2804 - val_acc: 0.8023
Epoch 5/8
84048/84048 [==============================] - 180s 2ms/sample - loss: 1.2376 - acc: 0.8035 - val_loss: 1.2646 - val_acc: 0.8045
Epoch 6/8
84048/84048 [==============================] - 179s 2ms/sample - loss: 1.1959 - acc: 0.8070 - val_loss: 1.2653 - val_acc: 0.8047
Epoch 7/8
84048/84048 [==============================] - 179s 2ms/sample - loss: 1.1570 - acc: 0.8106 - val_loss: 1.2555 - val_acc: 0.8045
Epoch 8/8
84048/84048 [==============================] - 178s 2ms/sample - loss: 1.1197 - acc: 0.8141 - val_loss: 1.2507 - val_acc: 0.8065
```

Figure 35

Figure 35 shows the 8 epochs for LSTM with Attention Mechanism model.

- **Accuracy of LSTM with Attention Mechanism model for Text Summarization**

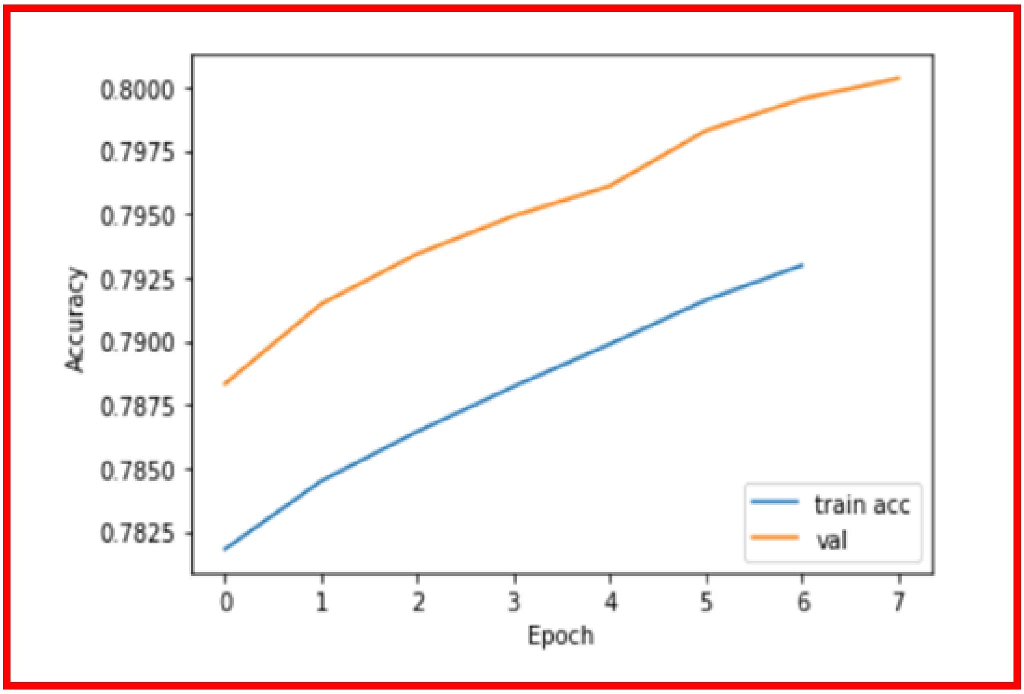


Figure 36

Figure 36 shows a graph for accuracy of training and testing data.

- **Loss of LSTM with Attention Mechanism model for Text Summarization**

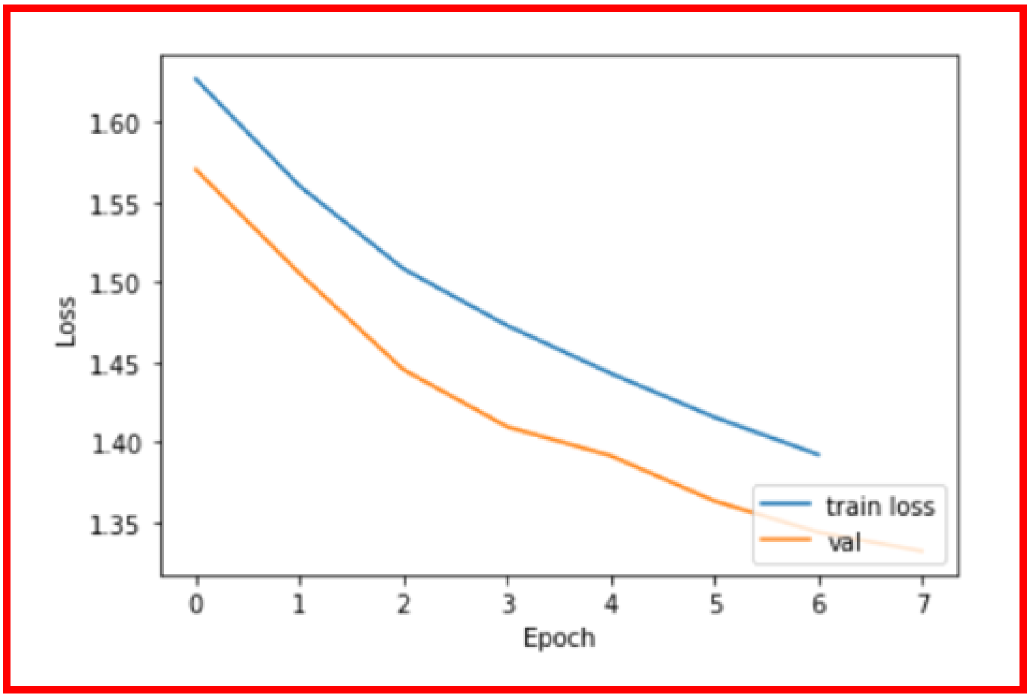


Figure 37

Figure 37 shows a graph for loss of training and testing data

- **Encoder Decoder LSTM with Attention Mechanism model**

```python
encoder_model = Model(inputs=encoder_inputs,outputs=[encoder_outputs, state_h, state_c])


decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_hidden_state_input = Input(shape=(max_text_len,latent_dim))

dec_emb2= dec_emb_layer(decoder_inputs)

decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=[decoder_state_input_h, decoder_state_input_c])


attn_out_inf, attn_states_inf = attn_layer([decoder_hidden_state_input, decoder_outputs2])
decoder_inf_concat = Concatenate(axis=-1, name='concat')([decoder_outputs2, attn_out_inf])


decoder_outputs2 = decoder_dense(decoder_inf_concat)

decoder_model = Model(
[decoder_inputs] + [decoder_hidden_state_input,decoder_state_input_h, decoder_state_input_c],
[decoder_outputs2] + [state_h2, state_c2])
```

```python
def decode_sequence(input_seq):

    e_out, e_h, e_c = encoder_model.predict(input_seq)
    print('input_seq: {}, e_out: {} '.format(input_seq,e_out))

    target_seq = np.zeros((1,1))
```

Figure 38

Figure 38 shows the Encoder Decoder LSTM with Attention Mechanism model.

- **Summarization with LSTM with Attention Mechanism model**

```
Review: fans special bars general house like several varieties including lower calorie ones great taste sweet super sweet nt size expected
Original summary: start loved house end
input_seq: [[2061  434  153 1072  302    1  214  694  825  664  589  229    5    6
    44  383   44  670  130  427  114    6  226  179 4254   37   42    2
   675  249  919   81  152   13    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    0    0    0    0    0    0    0    0
     0    0]], e_out: [[[ 3.2786969e-02  2.8049177e-02  8.7306574e-02 ... -8.4276207e-02
   -7.4950397e-02  1.0502533e-03]
  [ 1.0906560e-01  1.0310741e-01  2.0168208e-01 ... -1.7765377e-01
   -1.4954491e-01  4.6468940e-02]
  [ 4.3882394e-01  4.0380952e-01  2.0437217e-01 ... -3.4313601e-01
   -9.9306196e-02 -7.5081497e-02]
  ...
  [ 7.8391671e-01 -6.4601987e-18 -2.1850672e-11 ...  5.2688652e-01
    3.4803629e-01 -5.4071647e-01]
  [ 7.8371793e-01 -2.5516748e-18 -1.2761886e-11 ...  5.2736801e-01
    3.4755141e-01 -5.4147786e-01]
  [ 7.8349012e-01 -1.0081952e-18 -7.4506634e-12 ...  5.2825880e-01
    3.4706676e-01 -5.4227281e-01]]]
sampled_token: kind
sampled_token: bar
sampled_token: end
Predicted summary:  kind bar
```

Figure 39

Figure 39 shows the summarization of the food reviews by the LSTM with Attention Mechanism model.

- **ROUGE Score**

```python
from rouge_score import rouge_scorer
scorer = rouge_scorer.RougeScorer(['rouge1'])
results = {'precision': [], 'recall': [], 'fmeasure': []}
for (h, r) in zip(text, predicted):
    # computing the ROUGE
    score = scorer.score(h, r)
    # separating the measurements
    precision, recall, fmeasure = score['rouge1']
    # add them to the proper list in the dictionary
    results['precision'].append(precision)
    results['recall'].append(recall)
    results['fmeasure'].append(fmeasure)

result = pd.DataFrame(results)
result
```

Figure 40

Figure 40 shows the code for ROUGE score for LSTM with Attention Mechanism model.

- **ROUGE Matrix for LSTM**

	precision	recall	fmeasure
0	0.40	0.058824	0.102564
1	0.50	0.125000	0.200000
2	0.00	0.000000	0.000000
3	0.00	0.000000	0.000000
4	0.25	0.030303	0.054054
5	0.25	0.015873	0.029851
6	0.00	0.000000	0.000000
7	0.00	0.000000	0.000000
8	0.00	0.000000	0.000000
9	0.20	0.022727	0.040816
10	0.25	0.028571	0.051282

Figure 23

Figure 23 shows the precision, recall and f1measure for LSTM with Attention Mechanism model.