National College of Ireland

Configuration Manual

MSc Research Project
MSc In Cloud Computing

Aishwarya Raut
Student ID: 21175748

School of Computing
National College of Ireland

Supervisor: Prof. Shivani Jaswal

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Aishwarya Raut
Student ID:	21175748
Programme:	MSc In Cloud Computing
Year:	2023
Module:	MSc Research Project
Supervisor:	Prof. Shivani Jaswal
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	931
Page Count:	9

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:	Aishwarya Raut
Date:	14th December 2023

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

Aishwarya Raut
21175748

1 Introduction

This document provides insights into the hardware and software components essential for building and deploying the incident classification model developed. Users can follow the outlined steps to execute the code and reproduce the project results.Cristina (2023)

2 Hardware and Software configuration

The hardware configuration details for the host device employed in executing this research project are presented in Figure 1 . Refer to Figure 1 for a comprehensive overview of the technical specifications.

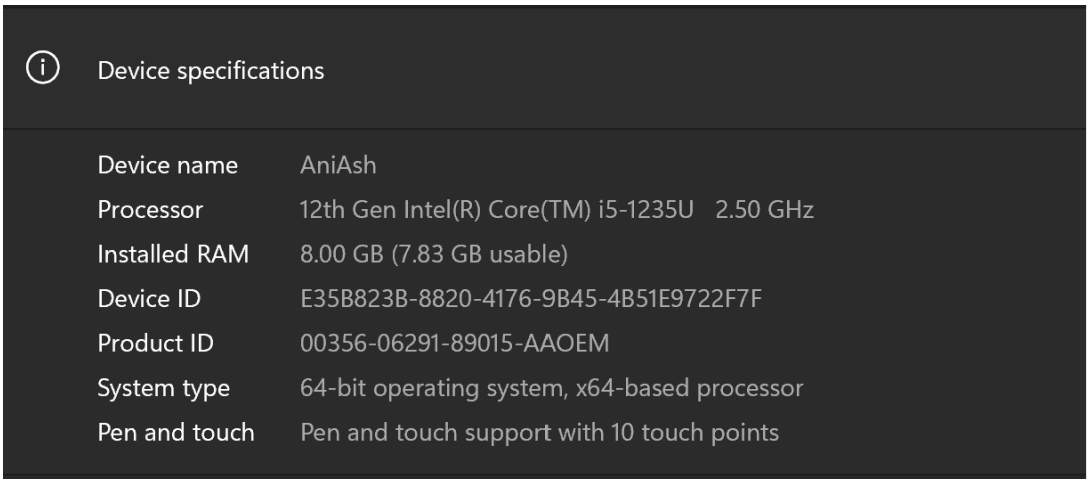


Figure 1: Hardware and Software configuration

2.1 To get the code running

Run the comparative analysis of models on Google Colaboratory:

- For a convenient and swift setup, you can effortlessly execute the entire configuration process by navigating to the Google Colab file provided in artifacts named "Final_code.ipynb" and simply clicking 'Run All' in the 'Runtime' tab.
- This will sequentially run all the configurations outlined in the manual, streamlining the setup process for optimal efficiency.

Run the ticket classification system User Interface:

- To access and run the UI project, follow these step-by-step instructions:

1.Deployment of application over Elastic Beanstalk

- CodePipeline has been created, thus upon detecting changes in the connected repository, it will trigger automatic deployment of the Flask application to the Elastic Beanstalk environment.
- The link to the deployed application to AWS Elastic Beanstalk is `http://myelasticbeanstalkappsupportticketnew5.eba-3hi7gf52.ap-southeast-2.elasticbeanstalk.com/`

OR

1. Download Code Artifacts:

1. Download the project's code artifacts from the submitted artifacts.

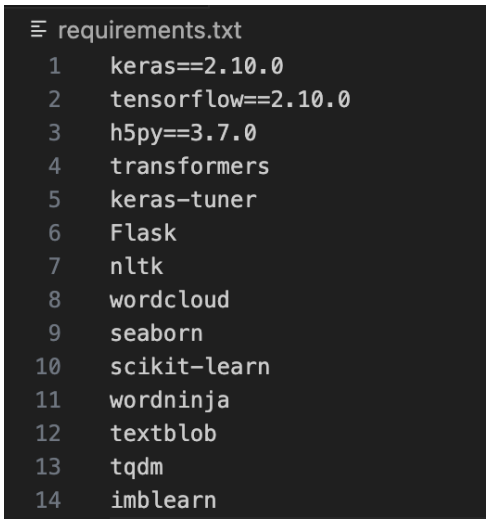
2. Download and Install Visual Studio Code (VSCode):

1. Download VSCode from the official website
2. Install VSCode by following the on-screen instructions.

3. Open Project in VSCode:

1. Open VSCode and click on "Open Folder."
2. Navigate to the Flask folder within the downloaded artifacts and select it.

4. Download Requirements: To install the required dependencies listed in a `requirements.txt` file for a Flask app in VSCode, follow these steps:



1. Open the terminal in VSCode.
2. Navigate to the directory where your Flask app is located using the `cd` command. For example:

```
cd /path/to/your/flask/app
```

3. Run the following command to install the dependencies:

```
pip install -r requirements.txt
```

   This command reads the dependencies from the `requirements.txt` file and installs them in your Python environment.

Now, your Flask app should have all the necessary dependencies installed. You can proceed with running your Flask application. If there are additional steps or commands required for running your specific Flask app, you may find them in the app's documentation or source code.

5. Run the Flask Application:

1. Run the 'app.py' file in VSCode.

6. Access the Application:

1. Open the link displayed in the terminal after running the application.
2. You will be redirected to the application site.

3 Dataset

- The project makes use of a dataset obtained from Kaggle, accessible at Kaggle Dataset Link. This dataset contains complaints from customers within a financial company, stored in a .json format.
- The primary aim of the dataset is to address the need for an automated system to classify customer support tickets based on products and services, aiming to enhance the efficiency of resolving issues.
- With a total of 78,313 customer complaints, the dataset encompasses 22 features, each representing different facets of issues reported by customers.

4 Data Exploration

Google Drive has been connected to the current Colab Notebook session to read the dataset using the code given in Figure 2.

4.1 Installing Libraries

To ensure the proper functioning of the project, it is essential to install the required libraries. Run the following commands to install the necessary dependencies:

After successfully installing the required libraries, the environment is now equipped with the essential dependencies for executing the project. To proceed with the next steps, a few libraries need to be imported.

Figure 2: Data loading

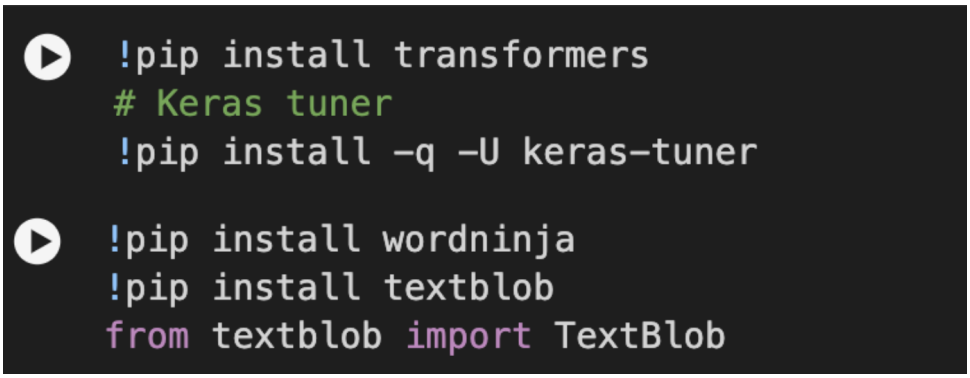
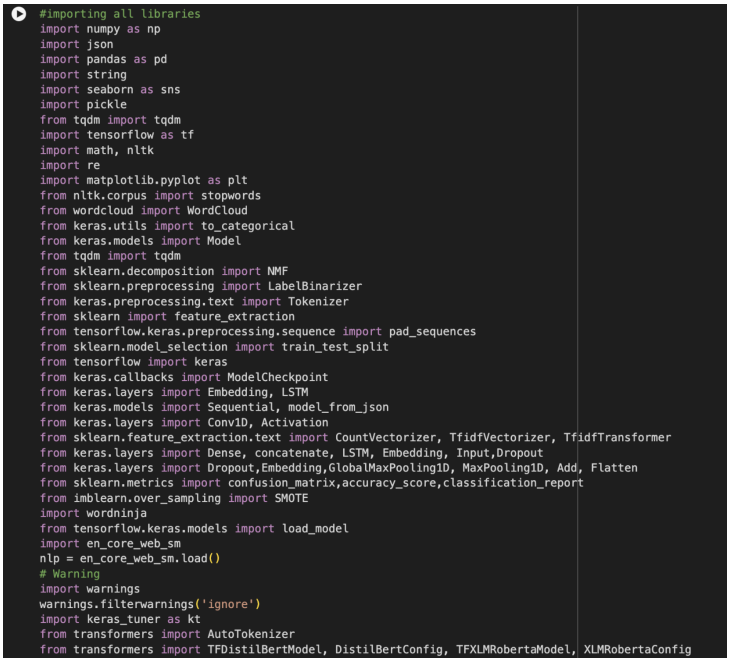


Figure 3: Installing libraries

4.2 Importing All the Libraries

The following crucial libraries are imported, setting the groundwork for data manipulation, natural language processing, machine learning, and various other tasks.

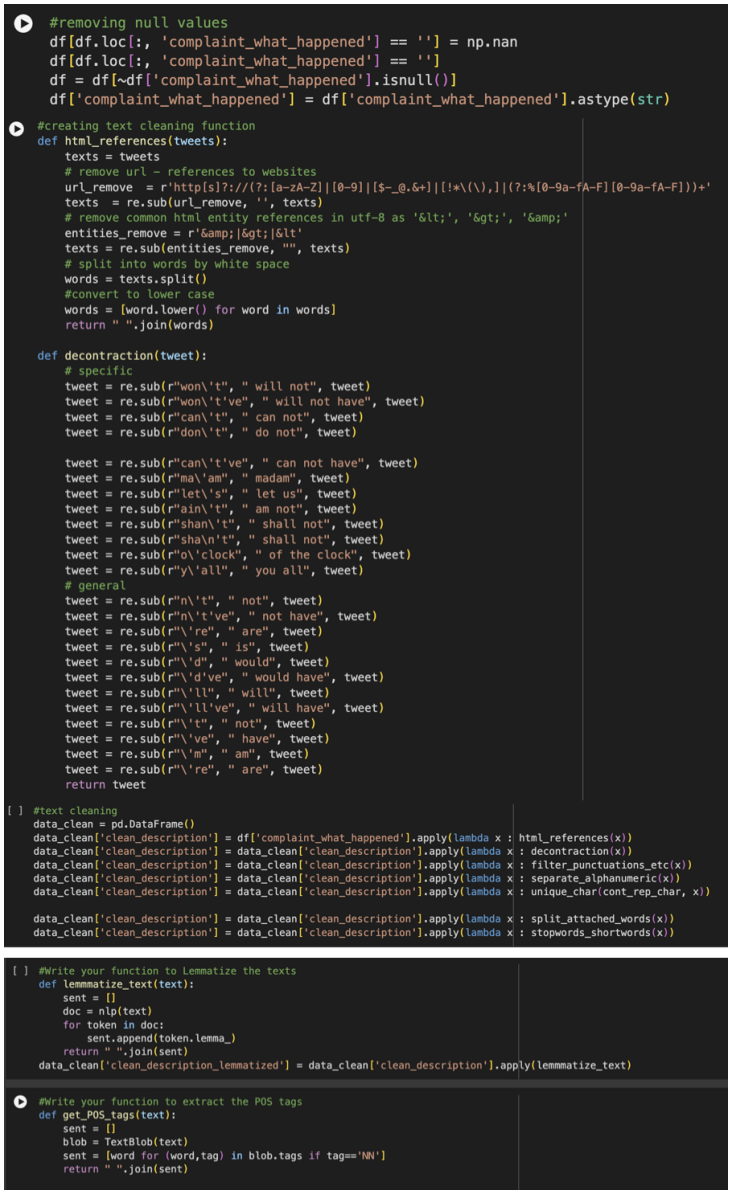


```python
#importing all libraries
import numpy as np
import json
import pandas as pd
import string
import seaborn as sns
import pickle
from tqdm import tqdm
import tensorflow as tf
import math, nltk
import re
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from wordcloud import WordCloud
from keras.utils import to_categorical
from keras.models import Model
from tqdm import tqdm
from sklearn.decomposition import NMF
from sklearn.preprocessing import LabelBinarizer
from keras.preprocessing.text import Tokenizer
from sklearn import feature_extraction
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from tensorflow import keras
from keras.callbacks import ModelCheckpoint
from keras.layers import Embedding, LSTM
from keras.models import Sequential, model_from_json
from keras.layers import Conv1D, Activation
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer, TfidfTransformer
from keras.layers import Dense, concatenate, LSTM, Embedding, Input,Dropout
from keras.layers import Dropout,Embedding,GlobalMaxPooling1D, MaxPooling1D, Add, Flatten
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
from imblearn.over_sampling import SMOTE
import wordninja
from tensorflow.keras.models import load_model
import en_core_web_sm
nlp = en_core_web_sm.load()
# Warning
import warnings
warnings.filterwarnings('ignore')
import keras_tuner as kt
from transformers import AutoTokenizer
from transformers import TFDistilBertModel, DistilBertConfig, TFXLMRobertaModel, XLMRobertaConfig
```

Figure 4: Importing all libraries

4.3 Data Preprocessing

Below are the essential steps for data cleaning and preprocessing, crucial for preparing the dataset for subsequent analysis and model development, such as CNN + LSTM with Glove Embedding.



```
#removing null values
df[df.loc[:, 'complaint_what_happened'] == ''] = np.nan
df[df.loc[:, 'complaint_what_happened'] == '']
df = df[~df['complaint_what_happened'].isnull()]
df['complaint_what_happened'] = df['complaint_what_happened'].astype(str)
```

```
#creating text cleaning function
def html_references(tweets):
    texts = tweets
    # remove url - references to websites
    url_remove  = r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+'
    texts  = re.sub(url_remove, '', texts)
    # remove common html entity references in utf-8 as '&lt;', '&gt;', '&amp;'
    entities_remove = r'&amp;|&gt;|&lt'
    texts = re.sub(entities_remove, "", texts)
    # split into words by white space
    words = texts.split()
    #convert to lower case
    words = [word.lower() for word in words]
    return " ".join(words)

def decontraction(tweet):
    # specific
    tweet = re.sub(r"won\'t", " will not", tweet)
    tweet = re.sub(r"won\'t've", " will not have", tweet)
    tweet = re.sub(r"can\'t", " can not", tweet)
    tweet = re.sub(r"don\'t", " do not", tweet)

    tweet = re.sub(r"can\'t've", " can not have", tweet)
    tweet = re.sub(r"ma\'am", " madam", tweet)
    tweet = re.sub(r"let\'s", " let us", tweet)
    tweet = re.sub(r"ain\'t", " am not", tweet)
    tweet = re.sub(r"shan\'t", " shall not", tweet)
    tweet = re.sub(r"sha\n't", " shall not", tweet)
    tweet = re.sub(r"o\'clock", " of the clock", tweet)
    tweet = re.sub(r"y\'all", " you all", tweet)
    # general
    tweet = re.sub(r"n\'t", " not", tweet)
    tweet = re.sub(r"n\'t've", " not have", tweet)
    tweet = re.sub(r"\'re", " are", tweet)
    tweet = re.sub(r"\'s", " is", tweet)
    tweet = re.sub(r"\'d", " would", tweet)
    tweet = re.sub(r"\'d've", " would have", tweet)
    tweet = re.sub(r"\'ll", " will", tweet)
    tweet = re.sub(r"\'ll've", " will have", tweet)
    tweet = re.sub(r"\'t", " not", tweet)
    tweet = re.sub(r"\'ve", " have", tweet)
    tweet = re.sub(r"\'m", " am", tweet)
    tweet = re.sub(r"\'re", " are", tweet)
    return tweet
```

```
[ ] #text cleaning
    data_clean = pd.DataFrame()
    data_clean['clean_description'] = df['complaint_what_happened'].apply(lambda x : html_references(x))
    data_clean['clean_description'] = data_clean['clean_description'].apply(lambda x : decontraction(x))
    data_clean['clean_description'] = data_clean['clean_description'].apply(lambda x : filter_punctuations_etc(x))
    data_clean['clean_description'] = data_clean['clean_description'].apply(lambda x : separate_alphanumeric(x))
    data_clean['clean_description'] = data_clean['clean_description'].apply(lambda x : unique_char(cont_rep_char, x))

    data_clean['clean_description'] = data_clean['clean_description'].apply(lambda x : split_attached_words(x))
    data_clean['clean_description'] = data_clean['clean_description'].apply(lambda x : stopwords_shortwords(x))
```

```
[ ] #Write your function to Lemmatize the texts
    def lemmmatize_text(text):
        sent = []
        doc = nlp(text)
        for token in doc:
            sent.append(token.lemma_)
        return " ".join(sent)
    data_clean['clean_description_lemmatized'] = data_clean['clean_description'].apply(lemmmatize_text)
```

```
#Write your function to extract the POS tags
def get_POS_tags(text):
    sent = []
    blob = TextBlob(text)
    sent = [word for (word,tag) in blob.tags if tag=='NN']
    return " ".join(sent)
```

5 Model Training with CNN + LSTM with GloVe Embeddings

Below is the implementation for training the support ticket classification model, incorporating GloVe embeddings for enhanced text representation. This section encompasses loading the embeddings, tokenizing the dataset, and building the model for effective classification.

```python
def filter_punctuations_etc(tweets):
    words = tweets.split()
    # prepare regex for char filtering
    re_punc = re.compile( '[%s]' % re.escape(string.punctuation))
    # remove punctuation from each word
    words = [re_punc.sub('', w) for w in words]
    # filter out non-printable characters
    re_print = re.compile( '[^%s]' % re.escape(string.printable))
    words = [re_print.sub(' ', w) for w in words]
    return " ".join(words)

def separate_alphanumeric(tweets):
    words = tweets
    # separate alphanumeric
    words = re.findall(r"[^\W\d_]+|\d+", words)
    return " ".join(words)

def cont_rep_char(text):
    tchr = text.group(0)

    if len(tchr) > 1:
        return tchr[0:2] # take max of 2 consecutive letters

def unique_char(rep, tweets):
    substitute = re.sub(r'(\w)\1+', rep, tweets)
    return substitute


def split_attached_words(tweet):
    words = wordninja.split(tweet)
    return" ".join(words)

def stopwords_shortwords(tweet):
    # filter out stop words
    words = tweet.split()
    stop_words = set(stopwords.words( 'english' ))
    words = [w for w in words if not w in stop_words]
    # filter out short tokens
    for word in words:
        if word.isalpha():
            words = [word for word in words if len(word) > 1 ]
        else:
            words = [word for word in words]
    return" ".join(words)
```

Figure 5: Data Preprocessing

Training the support ticket classification model using the provided configuration. The model undergoes 5 epochs with a specified batch size, while monitoring performance on the validation set for each epoch.

```python
history = model.fit(X_train,y_train,epochs=5,batch_size=BATCH_SIZE,verbose=1,validation_data=(X_val,y_val), shuffle=True)
```

To find the training validation and accuracy as well as validation and loss run below code snipped:

```python
#Train and validation accuracy
plt.plot(history.history['accuracy'], 'b', label='Training accurarcy')
plt.plot(history.history['val_accuracy'], 'r', label='Validation accurarcy')
plt.title('Training and Validation accurarcy')
plt.legend()

plt.figure()
#Train and validation loss
plt.plot(history.history['loss'], 'b', label='Training loss')
plt.plot(history.history['val_loss'], 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```

To fetch confusion matrix for CNN-LSTM follow below code snippet:

6 Model Training with FineTune XLMRoberta Transformer

The data loading and pre-processing steps for FineTune XLMRoberta models remain the same as CNN+LSTM with glove model. The only difference is data preprocessing and model building. These steps are given below.

```
#confusion Matrix
plt.figure(figsize=(6,4))
matrix =confusion_matrix(y_test_new, predictions)
class_names=[0,1,2,3,4]
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="crest" ,fmt='g')
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
```

Below is the model function that defines the architecture for FineTune Distillbert Transformer. The function incorporates a transformer layer, setting its weights to be non-trainable, and includes dense layers for feature extraction. The output layer utilizes the sigmoid activation function for multi-label classification.Sharma (2023)

```
embedding_dict={}
with open('/content/drive/My Drive/support_ticket_classification/glove.6B.300d.txt','r') as f:
    for line in f:
        values=line.split()
        word=values[0]
        vectors=np.asarray(values[1:],'float32')
        embedding_dict[word]=vectors
f.close()

tok= Tokenizer()
tok.fit_on_texts(data_clean['clean_description_clean'])
titles=tok.texts_to_sequences(data_clean['clean_description_clean'])
titles = pad_sequences(titles,maxlen=max_lenght,padding='post')

global vocab_size
vocab_size= len(tok.word_index)+1

word_index=tok.word_index
print('Number of unique words:',len(word_index))

num_words=len(word_index)+1
embedding_matrix=np.zeros((num_words,300))

for word,i in tqdm(word_index.items()):
    if i > num_words:
        continue

    emb_vec=embedding_dict.get(word)
    if emb_vec is not None:
        embedding_matrix[i]=emb_vec

labels= targetlabel
max_len=max_lenght
```

7 Model Training with FineTune Distilbert Transformer

The data loading and pre-processing steps for FineTune Distilbert Transformer models remain the same as CNN+LSTM with glove model. The only difference is data preprocessing and model building. These steps are given below.Olafenwa (2022)

```
tokenizer = AutoTokenizer.from_pretrained("xlm-roberta-base")
```

```
X_train, X_test, y_train, y_test = train_test_split(data_clean['clean_description_clean'], targetlabel, test_size=0.2, random_state=42, shuffle= True)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42, shuffle= True)
```

```
# Model function
def create_model(transformer):

    # Make Transformer layers untrainable
    for layer in transformer.layers:
        layer.trainable = False
    # Input layers
    input_ids_layer = keras.Input(shape =(max_lenght,),
                          dtype=tf.int32,
                          name='input_ids')
    input_attention_layer = keras.Input(shape=(max_lenght,),
                                dtype=tf.int32,
                                name='attention_mask')

    #  outputs a tuple where the first element at index 0
    # represents the hidden-state at the output of the model's last layer.
    # It is a tf.Tensor of shape (batch_size, sequence_length, hidden_size=768).
    last_hidden_state = transformer([input_ids_layer, input_attention_layer])[0]

    # We only care about 's output for the [CLS] token,
    # which is located at index 0 of every encoded sequence.
    # Splicing out the [CLS] tokens gives us 2D data.
    cls_token = last_hidden_state[:, 0, :]
    # Hidden layers
    output = keras.layers.Dense(256,
                                kernel_initializer=keras.initializers.GlorotUniform(seed=1),
                                kernel_constraint=None,
                                bias_initializer='zeros',
                                activation='relu')(cls_token)
    output = keras.layers.Dropout(0.2)(output)
    output = keras.layers.Dense(128, activation = 'relu')(output)
    # Output layer
    output = keras.layers.Dense(cls, activation='sigmoid')(output)
    # Define the model
    model = keras.Model([input_ids_layer, input_attention_layer],
                        output)
    model.summary()
    keras.utils.plot_model(model)

    return model
```

References

Cristina, S. (2023). Training the transformer model, *machinelearningmastery* .
URL: *https://machinelearningmastery.com/training-the-transformer-model/*

Olafenwa, A. (2022). The concept of transformers and training a transformers model, *towardsdatascience* .
URL: *https://towardsdatascience.com/the-concept-of-transformers-and-training-a-transformers-model-45a09ae7fb50*

Sharma, D. (2023). Training an adapter for roberta model for sequence classification task, *analyticsvidhya* .