Al-Imam Muhammad Bin Saud Islamic University

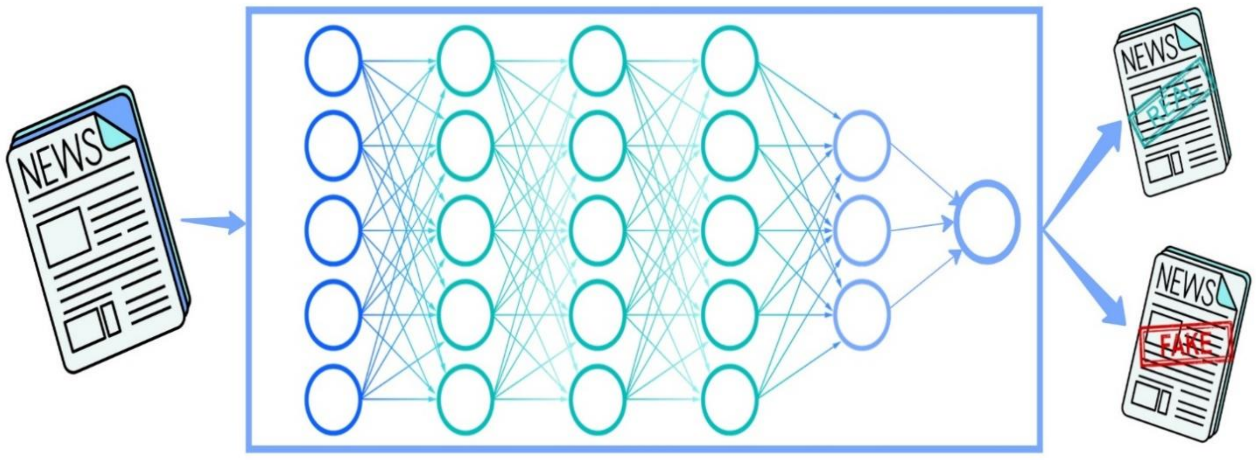
College of Computer and Information Sciences

Department of Computer Science

CS 364 Deep learning

Course project

Feb 11th, 2023



Student Name	*Student ID*	*Section*
Khloud Alnufaie	440020617	371
Raghad Albosais	440020209	371
Weaam Alghaith	440023306	371

Table of Contents

Figures:

Tables:

1.Introduction

1.1 Project Overview

In this project, we aim to develop a DL model on specific NLP dataset using what we have been learned during the lectures. The table below illustrate the main description for our project:

Input	Text.
Output	Binary classes (Fake and Real).
Field	AI-Natural Language Processing.
Task	Classification.
App	AI in Cyber Security.
Evaluation metric	Accuracy, Precision, Recall.
Learning method	Deep learning.
Learning type	Feature extractor: pre-trained on semi-supervised learning tasks. Classifier: Supervised learning.

Table 1 DL-project description

1.2 Problem Statement

In recent years, the growth of online social media has greatly facilitated communication among people. Online social media allows users to share information, connect with others, and stay updated on current news. The news can be presented in different formats in social media, such as articles. In the articles, the information is provided in a more formal way, which makes it more suitable for world, politics, and government news. While these articles are being shared on social media every day, it is not always credible, and some of it has been designed to mislead. Fake news is often used to describe such content. Fake news "is fabricated information that mimics news media content in form but…lack(s) the news media's editorial norms and processes for ensuring the accuracy and credibility of information". It overlaps with misinformation (false or misleading information) and disinformation (false information purposely spread to mislead people). One example of sophisticated fake news is "deepfakes". Deepfakes are fictional videos or images or text with either well-known or made-up people doing, or saying, things that are not real. This is created using artificial intelligence and machine learning.

1.3 Motivation and project goal

The presence of large amounts of fake news online can have a serious impact on individuals and society. fake news can have devastating effects on democracy if people can no longer distinguish between what information is real and what is fake, and what sources are legitimate or not. This makes it easier to feed people with propaganda, discredit and undermine the truth and create chaos. With lies intricately woven into truths, opinions and facts taken out of context, it obscures the truth. This again creates confusion and uncertainty, making it harder to trust any information one is being presented with. Therefore, fake news detection on social media has recently become emerging research that is attracting tremendous attention either for industry or academia. In order to detect this fake news and improve consumer trust in the constant flow of online news, we are motivated to deal with this problem. Where the information provided in the fake articles have been generated by AI machines in case of deepfake news in a way to be realistic as possible. It can be detected manually by analyzing the content, but it needs an expert domain knowledge about article's topic to correctly understand and differentiate between fake and real articles. However, this process consumes a lot of time and effort. Instead, it can be detected using AI systems that learn both real and fake content and can be relied on for this task. Thus, in this project, we aim to build an accurate DL-model to detect and classify articles as fake and news.

1.4 Problem formulation

Task (T)	Classify an article to a real or a fake
Experience (E)	A collection of real and fake articles
Performance (P)	Classification evaluation metrics, which is accuracy, precision and recall. All of them are calculated based on the confusion matrix part (TP, TN, FP, FN) **Accuracy**: it is the total number of TP and TN divided by total number of TP and TN and FP and FN. **Precision**: it is the number of TP divided by the total number of TP and TN. **Recall**: it is the number of TP divided by the total number of TP and FN.

Table 2 Problem formulation

2. Dataset overview

ISOT Fake News Dataset includes both fake and real news articles. This dataset is collected and provided in this IOST research lab [1]. Articles were retrieved by crawling Reuters.com (News website) for truthful articles. Several sources were used to collect the fake news articles. It is important to note that the fake news articles were gathered from unreliable websites that were flagged by Politifact (a fact-checking organization in the United States) and Wikipedia. Various types of articles are included in the dataset. Nevertheless, most articles cover political and world news topics. The full description of the dataset is shown in Figure 1. Note that, both of the real and fake set was cleaned and preprocessed, except that there are punctuations in fake set. We conducted two experiments in our implementation, one without removing the punctuation and another with removing them. Regarding the splitting of the dataset, we use 60, 20, 20 ration to split the dataset into train, test and validation, respectively. At the preprocessing step, we just split the dataset into training and testing, while the validation set has been split during training. We use this ratio of splitting since it is most commonly used in previous projects that work in this dataset for the same task, by using this ratio, our model produces high performance.

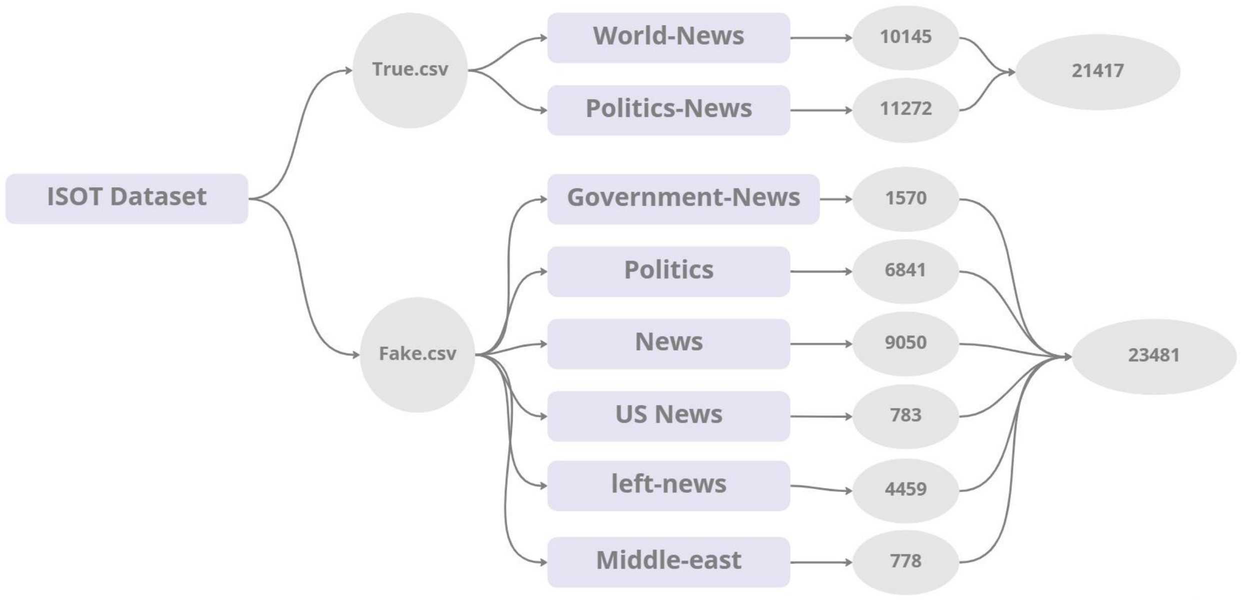


Figure 1. Dataset overview

3.Model architecture

3.1 Bidirectional Encoder Representations from Transformers (Bert)

Bert stands for Bidirectional Encoder Representations from Transformers. The meaning of each word in BERT abbreviation is explained in Figure 3. Bert is a deep learning model used in several tasks in NLP, such as Question Answering, Text summarization, and Classification. Also, Bert is used in the Google search engine and Response selection when you write an email. There are two types of Bert base and large. Bert base has 12 Encoder Layers (L) = 12, Attention Heads (A) = 12, Hidden Size (H) = 768, while Bert large has Encoder Layers (L) = 24, Attention Heads (A) = 16, Hidden Size (H) = 1024 [2]. The architecture is a stack of encoders, as shown in Figure 4.

The encoder part is inherited from the transformer. A transformer is an artificial neural network architecture used in deep learning applications to solve the problem of transforming input sequences into output sequences [3]. Transformers contain two components: encoder and decoder, as shown in Figure 2.

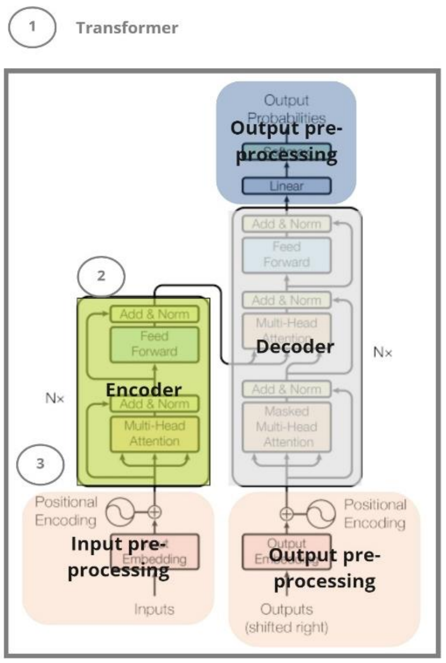


Figure 2. Transformer architecture

The transformers are better than RNN and LSTM for many reasons:

1. Transformers use non-sequential processing: Sentences are processed as a whole, rather than word by word.
2. Long-term memory transformers have much higher bandwidth and do not struggle with longer dependencies like RNN and LSTM.
3. The transformer is deeply bidirectional.

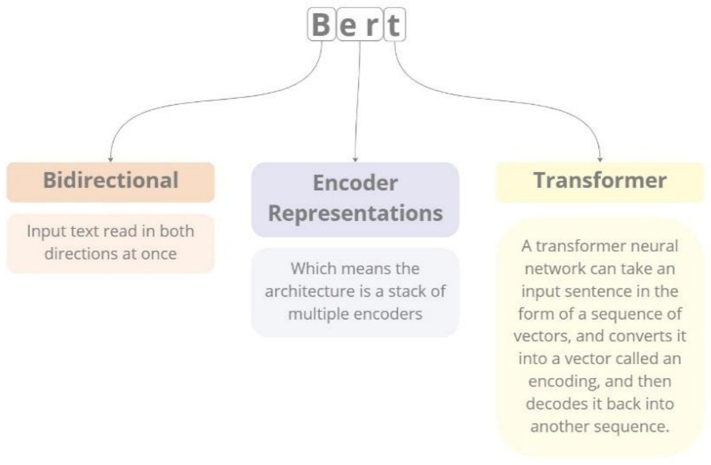

Bert
Bidirectional
Input text read in both directions at once
Encoder Representations
Which means the architecture is a stack of multiple encoders
Transformer
A transformer neural network can take an input sentence in the form of a sequence of vectors, and converts it into a vector called an encoding, and then decodes it back into another sequence.

Figure 3. The meaning of each word in BERT abbreviation

The transformer is discussed in the Bert model since it plays a role in the Bert model [4].

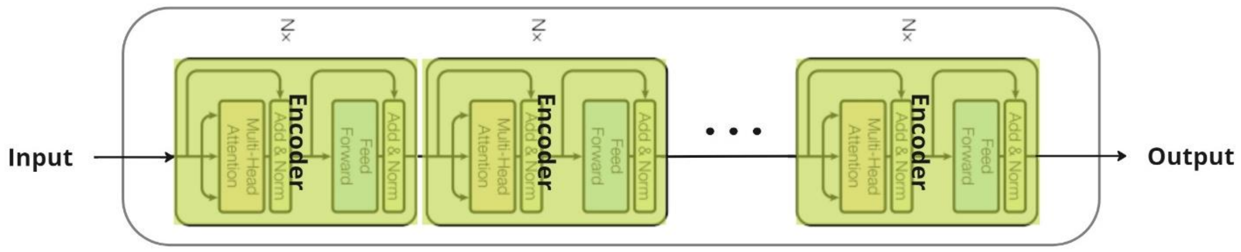


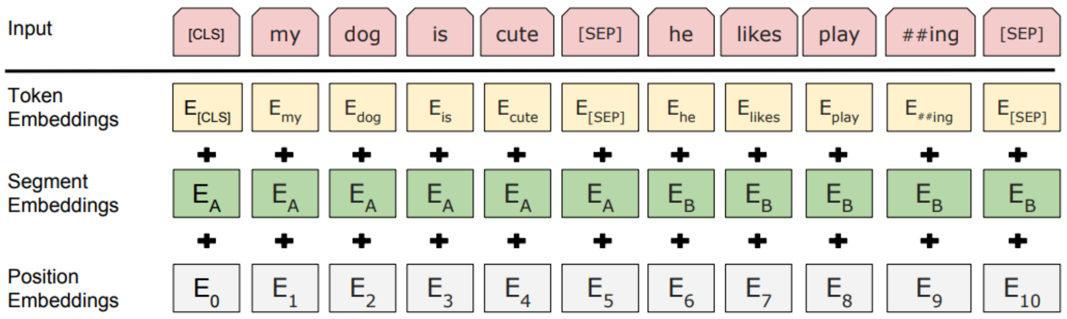
Figure 4. BERT architecture

3.1.1 The input of the model

Input to the model consists of three components as shown in Figure 5:

- **Embedding by position** takes into account the index number of the input token.
- **Segment embedding** indicates the number of a sentence within sequence of sentences.
- A **token embedding** holds a set of tokens for the words provided by the tokenizer.

Embeddings are added together and fed into BERT model [2].



Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	##ing	[SEP]
Token Embeddings	$E_{[CLS]}$	E_{my}	E_{dog}	E_{is}	E_{cute}	$E_{[SEP]}$	E_{he}	E_{likes}	E_{play}	$E_{\#\#ing}$	$E_{[SEP]}$
	+	+	+	+	+	+	+	+	+	+	+
Segment Embeddings	E_A	E_A	E_A	E_A	E_A	E_A	E_B	E_B	E_B	E_B	E_B
	+	+	+	+	+	+	+	+	+	+	+
Position Embeddings	E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}

Figure 5. Input representation for BERT. Token embeddings, segmentation embeddings, and position embeddings comprise input embeddings.

3.1.1.1 Tokenizer

The tokenizer has a vocabulary of 30000 words. The first token (at index position 0) given by the tokenizer at the input is a Special token provided by [CLS] known as the Classification token. In order to perform classification tasks such as sentiment analysis, we use the final hidden state of the last encoder layer that corresponds to the [CLS] token. Separator tokens provided by [SEP] are used to indicate two different sentences. The tokenizer uses the token [MASK] to mask a word. These unique tokens are explicitly processed when we call the tokenizer. Other than these unique tokens, there are regular tokens, and these regular tokens will outputted by the tokenizer, which are [2]:

- **input_ids**: These are the actual tokens for the input string. Each token is mappings to its respective IDs.
- **attention_mask**: A set of binary tokens which indicate which tokens are inputs and which are paddings.
- **Token_type_ids:** These are binary tokens that indicate whether the token belongs to the first sentence or the second sentence in a sequence of two sentences.

3.1.2 Encoder

An encoder maps an input sequence of symbol representations (x1, ..., xn) to the output sequence of representations z = (z1, ..., zn*)* [5]. Each encoder has two sub-layers. As shown in Figure 6 there are three components in the encoder.

1. A **multi-head self-attention mechanism** on the input vectors. The model can capture more relationships between words with multi-head attention than single-head attention.
2. **A fully connected feed-forward network** is a position-wise that applied to each position separately and identically. Two linear transformations are applied with a ReLU activation in between. Linear transformations are the same across different layers but use different parameters.

3. **"Adding and Normalizing"** - the "add" refers to the residual connections that add the input of each layer to the output, and the "Norm" refers to layer normalization.

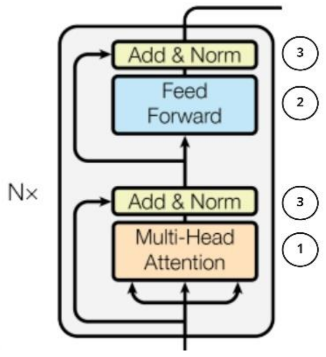


Figure 6. BERT encoder

3.1.3 The output of the model

As shown in Figure 7 for each position, a vector is an output with a size of hidden_size (768 in BERT Base). In the classification task, we consider only the output of the first position (passed the special [CLS] token) because the classifier needs input that can represent the entire sentence, not the meaning of each word [6].

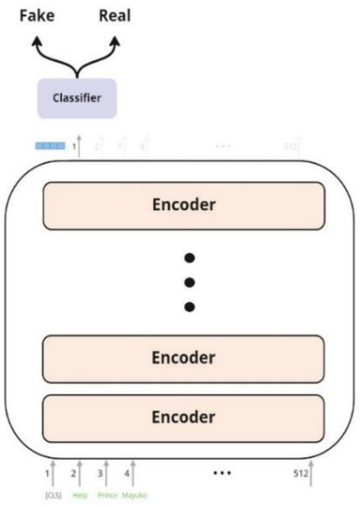


Figure 7. BERT output

3.2 Proposed Architecture

Our model architecture is shown in Figure 8. We use the title and content of the articles as input to our model, instead of using only the title or only the content. Because we want to preserve the important terms that help to make the final decision. For preprocessing step, we perform a cleaning processing (punctuation marks removal) for fake sets only in order to remove unhelpful parts of the data, or noise and keep only the words. In addition, we perform linguistics processing (tokenization) for both sets in order to make it in suitable format for the feature extractor (BERT). Tokenization is the process to split the sentence into words with their unique integer values.

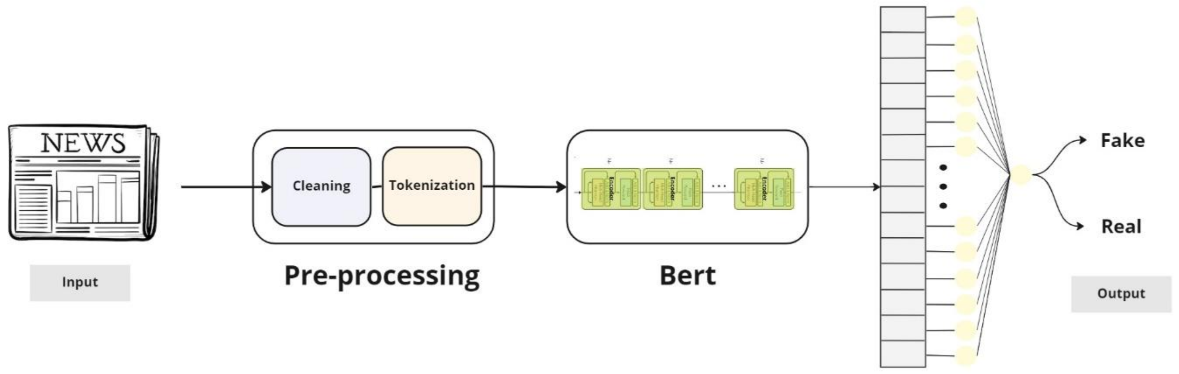


Figure 8 Model architecture

Regarding the feature extractor, we use BERT as pre-trained model to transfer the learning to our task. We will focus on the encoder part of the BERT since we need the machine to understand the context and features to help us classify fake news. We prefer to use pre-trained model instead of training the entire model from scratch to get more accurate results in quick time, since it reduces the efforts to engineer and build task specific architecture. In addition, our dataset is considered enough to feed it in such model. Meanwhile, the BERT has been trained in text data, it has similar features to our dataset. The BERT model has been shown the best performance on this dataset among other pre-trained models in this repository [7], also it achieves state-of-the-art performance on a large suite of sentence-level and token-level tasks. As we mentioned in section 3.1 about the two-size version of BERT, we decide to choose the Base BERT version that has 12 layers of encoder, because based on the previous projects that use BERT with our dataset, it performs better on Base version. In addition, the Base BERT version has two types: 1) cased 2) uncased. In BERT uncased, the text has been lowercased before tokenization step while in BERT cased, the text is same as the input text (no changes). We chose to use uncased since it is better than BERT cased in most applications except in applications where case information of text is important, and our application does not care about case sensitive letters.

The BERT model was trained on two self-supervised learning tasks with two objectives in pretraining model as given below:

- **Masked Language Model (MLM)**: In this pre-training approach, some input tokens are masked with the [MASK] token. Then the model is trained to predict the masked Tokens by gathering the context from the surrounding Tokens.
- **Next Sentence Prediction (NSP)**: This is a binary classification task in which we use the output token corresponding to the [CLS] token for modeling. The objective is to predict whether the second sentence is the next sentence.

In order to use BERT on text dataset to other NLP tasks, such as in our case “Fake news classification”, we must fine tune the BERT. The *Fine-tuning* in general is a process that takes a model that has already been trained for one given task and then tunes or tweaks the model to make it perform a second similar task. The fine-tunning process of BERT model as mentioned in its paper [8], it is simple process by plugging in the task specific inputs and outputs into BERT and finetune all the parameters end-to-end. Fine-tuning the parameters means you can either freeze all parameters of BERT and train only the output layers, or you can initialize the parameter with pre-trained values and train again the BERT with output layers but with only few amounts of time.

Therefore, the following is list illustrate what fine-tunning process we do to the BERT model on our task:

1) **Input plugging:** prepare our dataset in suitable format to BERT model**.** The input to the BERT model is the token embedding, produced by the tokenizer, which are the *input_ids, attention_mask and token_type_ids.* We just use the tokenizer without the segment and position embedding because depend on our search, we see that the segment and position embeddings are learned in the BERT during training.

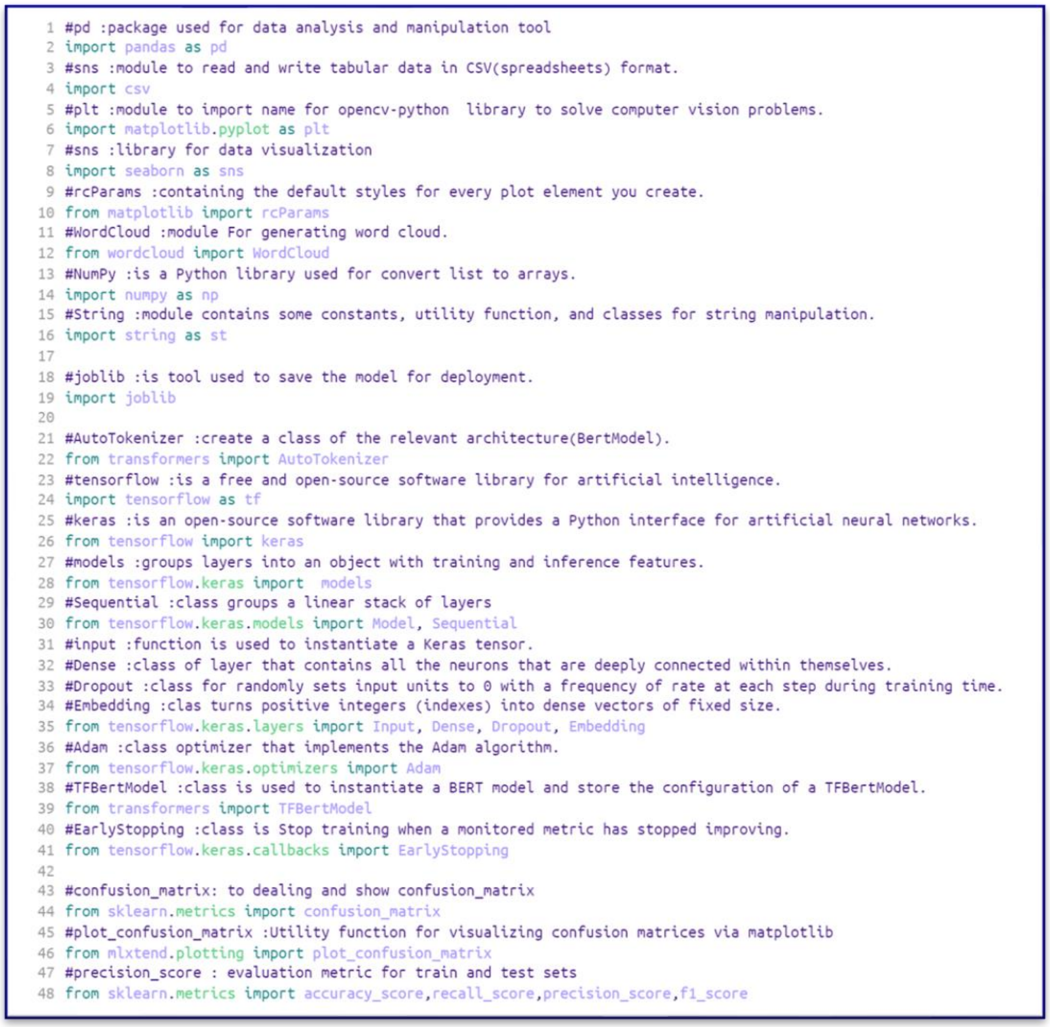
   We use built-in Tokenization function that take our input text (title + content of the articles) and output a dictionary, whose keys are *input_id* , and attention mask. The Tokenization function by default set the *token_type_id* to False, since it is used typically in a next sentence prediction task, where two sentences are given. Our task is classification of articles, we don’t need to care about do classification on pairs of sentences or question answering. This is why we just pass the *input_id* and *attention_mask* to the BERT model.

2) **BERT fine-tune parameters:** the BERT model is first initialized with the pre-trained parameters, and then all of the parameters are trained using our dataset for classification task tasks. We chose this approach to fine-tune the BERT parameters because the BERT paper [8] illustrates these procedures for using pre-trained BERT on any down-stream tasks. It shows that many of the previous projects that work in our dataset use this approach.
3) **Output plugging:** we add dense layers to take the output feature vectors from BERT model and classify the input article. In more details, we choose two dense layers and two dropout layers. The BERT paper [8] mentioned that the pre-trained BERT model can be finetuned with just one additional output layer to create state-of-the-art models for a wide range of tasks. This is why we just have enough to add two dense layers, the first one is with 64 units, and the second is with 1 unit as output (fake or real). Where the dropout layers are added to overcome the overfitting. One of the dropout layers is added after BERT with, and the other is added after first dense layer.

4. Step of Implementation

4.1 Import resources

- Import resources we need in implementation.



```
#pd :package used for data analysis and manipulation tool
import pandas as pd
#sns :module to read and write tabular data in CSV(spreadsheets) format.
import csv
#plt :module to import name for opencv-python  library to solve computer vision problems.
import matplotlib.pyplot as plt
#sns :library for data visualization
import seaborn as sns
#rcParams :containing the default styles for every plot element you create.
from matplotlib import rcParams
#WordCloud :module For generating word cloud.
from wordcloud import WordCloud
#NumPy :is a Python library used for convert list to arrays.
import numpy as np
#String :module contains some constants, utility function, and classes for string manipulation.
import string as st

#joblib :is tool used to save the model for deployment.
import joblib

#AutoTokenizer :create a class of the relevant architecture(BertModel).
from transformers import AutoTokenizer
#tensorflow :is a free and open-source software library for artificial intelligence.
import tensorflow as tf
#keras :is an open-source software library that provides a Python interface for artificial neural networks.
from tensorflow import keras
#models :groups layers into an object with training and inference features.
from tensorflow.keras import  models
#Sequential :class groups a linear stack of layers
from tensorflow.keras.models import Model, Sequential
#input :function is used to instantiate a Keras tensor.
#Dense :class of layer that contains all the neurons that are deeply connected within themselves.
#Dropout :class for randomly sets input units to 0 with a frequency of rate at each step during training time.
#Embedding :clas turns positive integers (indexes) into dense vectors of fixed size.
from tensorflow.keras.layers import Input, Dense, Dropout, Embedding
#Adam :class optimizer that implements the Adam algorithm.
from tensorflow.keras.optimizers import Adam
#TFBertModel :class is used to instantiate a BERT model and store the configuration of a TFBertModel.
from transformers import TFBertModel
#EarlyStopping :class is Stop training when a monitored metric has stopped improving.
from tensorflow.keras.callbacks import EarlyStopping

#confusion_matrix: to dealing and show confusion_matrix
from sklearn.metrics import confusion_matrix
#plot_confusion_matrix :Utility function for visualizing confusion matrices via matplotlib
from mlxtend.plotting import plot_confusion_matrix
#precision_score : evaluation metric for train and test sets
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score
```

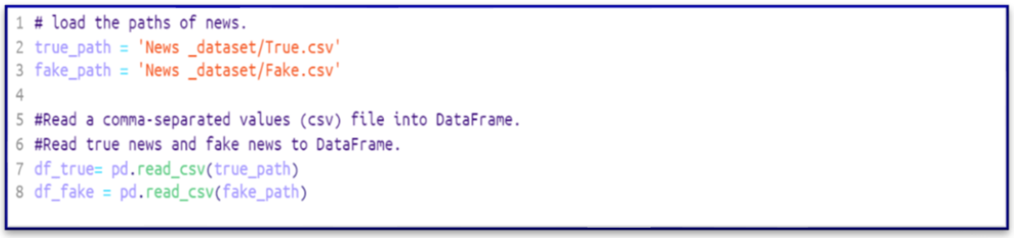
Figure 9 Import all required resources

4.2 Preparing the data

in this step we install the dataset then we explore and visualize the data then we applied preprocessing to data to cleaning news for best classification so that we can analyze it in a better way. Then we merge fake and true news, then we split the data into two subsets. In the last we apply tokenizer to preparing the inputs for a model.

4.2.1 Loading the data

- Load data and read csv files

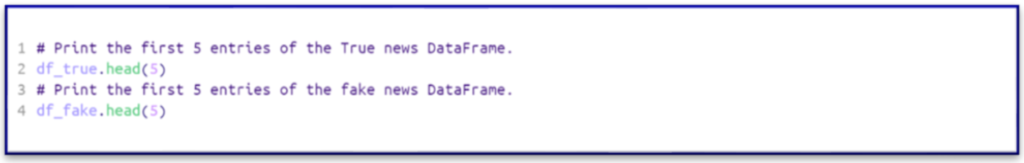


```python
# load the paths of news.
true_path = 'News _dataset/True.csv'
fake_path = 'News _dataset/Fake.csv'

#Read a comma-separated values (csv) file into DataFrame.
#Read true news and fake news to DataFrame.
df_true= pd.read_csv(true_path)
df_fake = pd.read_csv(fake_path)
```

Figure 10 load data

4.2.2 Exploring and visualizing the data

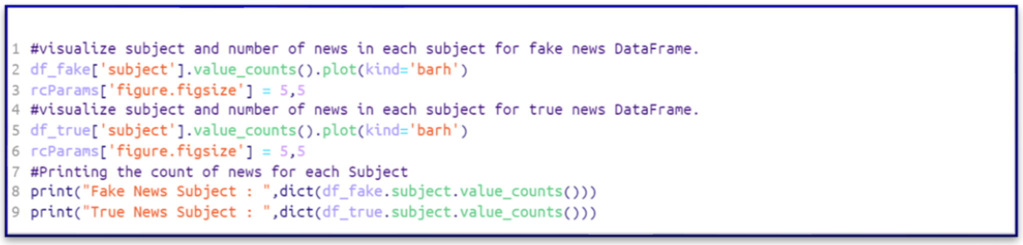
- Print head of true data frame and fake data frame



```python
# Print the first 5 entries of the True news DataFrame.
df_true.head(5)
# Print the first 5 entries of the fake news DataFrame.
df_fake.head(5)
```

Figure 11 Display data frame

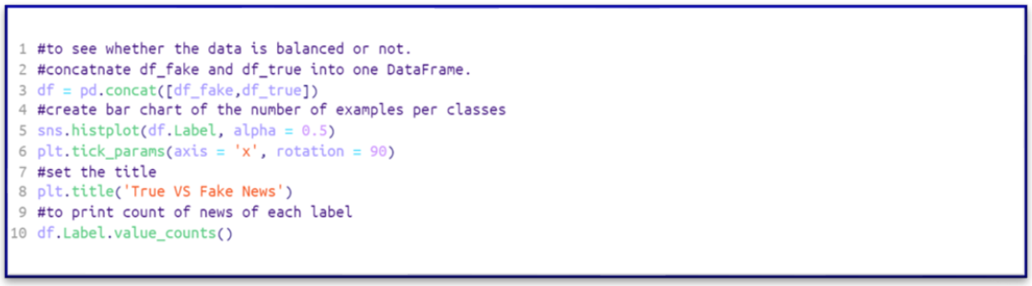
- Visualize subjects of news



```python
#visualize subject and number of news in each subject for fake news DataFrame.
df_fake['subject'].value_counts().plot(kind='barh')
rcParams['figure.figsize'] = 5,5
#visualize subject and number of news in each subject for true news DataFrame.
df_true['subject'].value_counts().plot(kind='barh')
rcParams['figure.figsize'] = 5,5
#Printing the count of news for each Subject
print("Fake News Subject : ",dict(df_fake.subject.value_counts()))
print("True News Subject : ",dict(df_true.subject.value_counts()))
```

Figure 12 visualize dataset subject

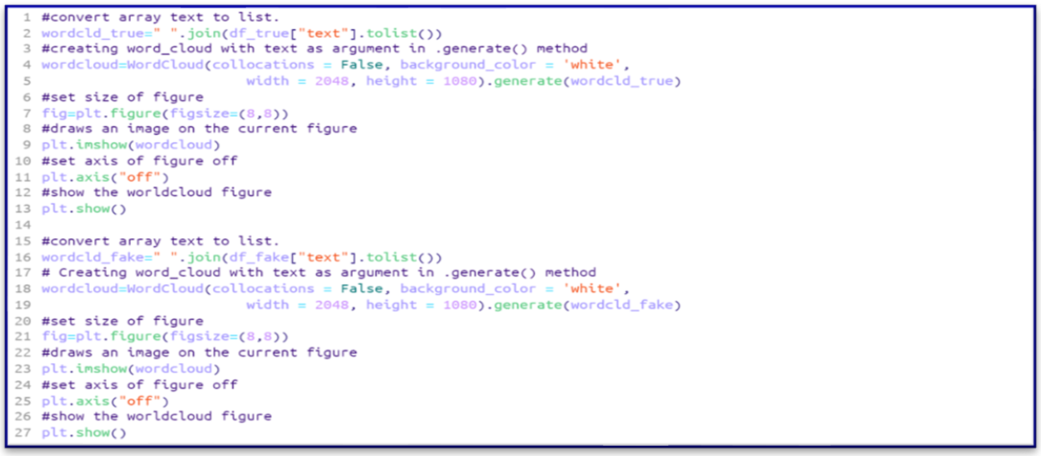
- Show bar chart for data



```python
#to see whether the data is balanced or not.
#concatnate df_fake and df_true into one DataFrame.
df = pd.concat([df_fake,df_true])
#create bar chart of the number of examples per classes
sns.histplot(df.Label, alpha = 0.5)
plt.tick_params(axis = 'x', rotation = 90)
#set the title
plt.title('True VS Fake News')
#to print count of news of each label
df.Label.value_counts()
```

Figure 13 Display dataset balancing of its classes

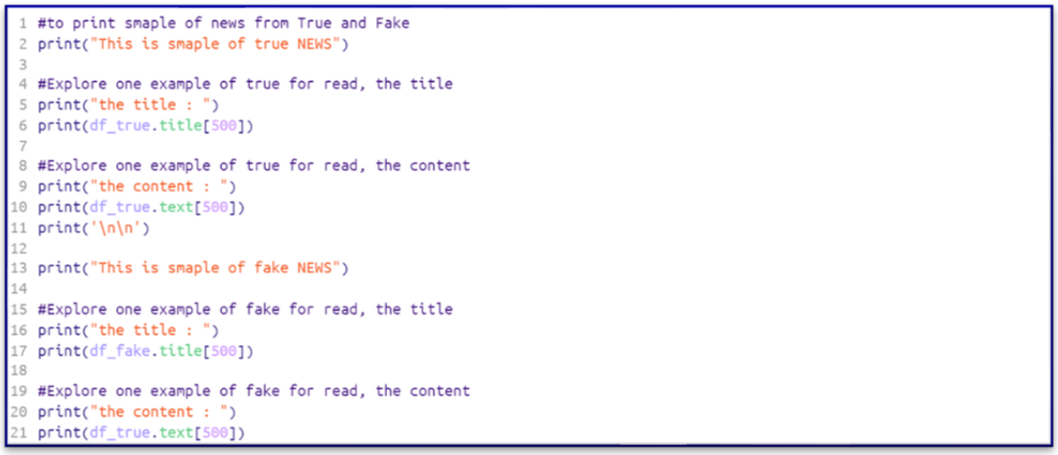
- Show word cloud for fake news and true news



```python
#convert array text to list.
wordcld_true=" ".join(df_true["text"].tolist())
#creating word_cloud with text as argument in .generate() method
wordcloud=WordCloud(collocations = False, background_color = 'white',
                              width = 2048, height = 1080).generate(wordcld_true)
#set size of figure
fig=plt.figure(figsize=(8,8))
#draws an image on the current figure
plt.imshow(wordcloud)
#set axis of figure off
plt.axis("off")
#show the worldcloud figure
plt.show()

#convert array text to list.
wordcld_fake=" ".join(df_fake["text"].tolist())
# Creating word_cloud with text as argument in .generate() method
wordcloud=WordCloud(collocations = False, background_color = 'white',
                              width = 2048, height = 1080).generate(wordcld_fake)
#set size of figure
fig=plt.figure(figsize=(8,8))
#draws an image on the current figure
plt.imshow(wordcloud)
#set axis of figure off
plt.axis("off")
#show the worldcloud figure
plt.show()
```

Figure 14 Display word cloud

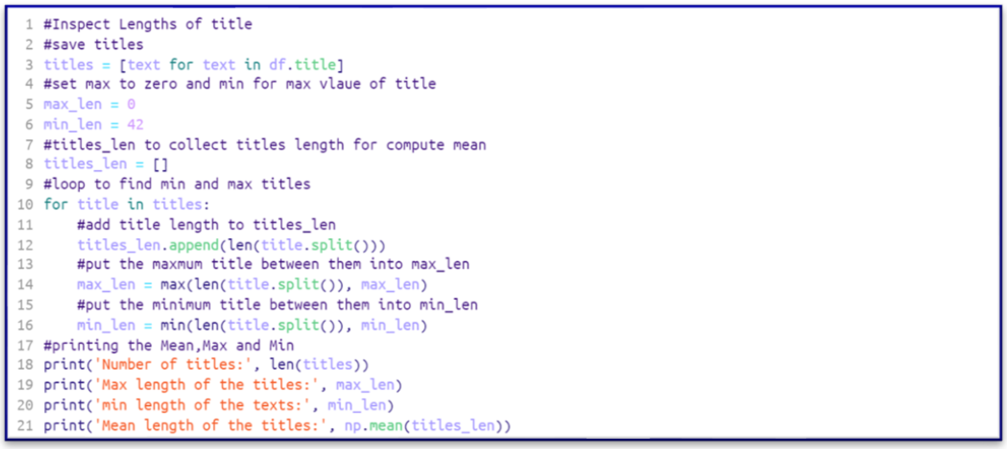
- Print sample from fake news and true news



```python
#to print smaple of news from True and Fake
print("This is smaple of true NEWS")

#Explore one example of true for read, the title
print("the title : ")
print(df_true.title[500])

#Explore one example of true for read, the content
print("the content : ")
print(df_true.text[500])
print('\n\n')

print("This is smaple of fake NEWS")

#Explore one example of fake for read, the title
print("the title : ")
print(df_fake.title[500])

#Explore one example of fake for read, the content
print("the content : ")
print(df_true.text[500])
```

Figure 15 Print samples

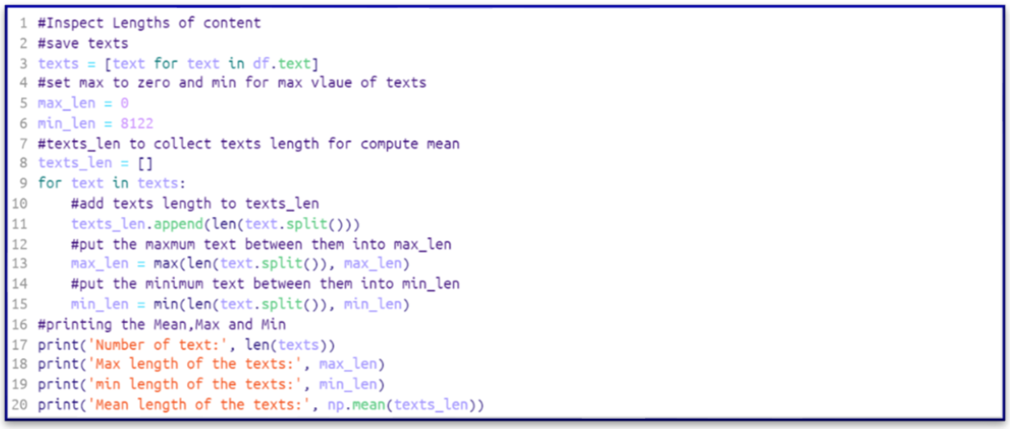
- Check title length and print minimum and maximum and mean



```python
#Inspect Lengths of title
#save titles
titles = [text for text in df.title]
#set max to zero and min for max vlaue of title
max_len = 0
min_len = 42
#titles_len to collect titles length for compute mean
titles_len = []
#loop to find min and max titles
for title in titles:
    #add title length to titles_len
    titles_len.append(len(title.split()))
    #put the maxmum title between them into max_len
    max_len = max(len(title.split()), max_len)
    #put the minimum title between them into min_len
    min_len = min(len(title.split()), min_len)
#printing the Mean,Max and Min
print('Number of titles:', len(titles))
print('Max length of the titles:', max_len)
print('min length of the texts:', min_len)
print('Mean length of the titles:', np.mean(titles_len))
```

Figure 16 Display length of titles

- Check content length and print minimum and maximum and mean

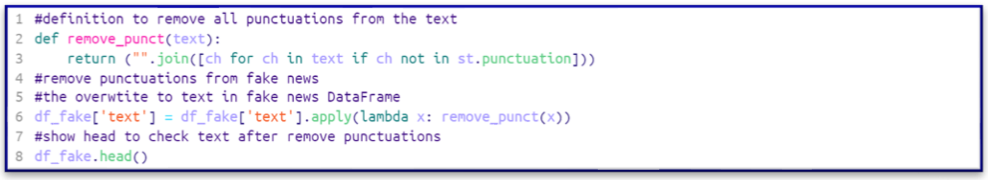


```python
#Inspect Lengths of content
#save texts
texts = [text for text in df.text]
#set max to zero and min for max vlaue of texts
max_len = 0
min_len = 8122
#texts_len to collect texts length for compute mean
texts_len = []
for text in texts:
    #add texts length to texts_len
    texts_len.append(len(text.split()))
    #put the maxmum text between them into max_len
    max_len = max(len(text.split()), max_len)
    #put the minimum text between them into min_len
    min_len = min(len(text.split()), min_len)
#printing the Mean,Max and Min
print('Number of text:', len(texts))
print('Max length of the texts:', max_len)
print('min length of the texts:', min_len)
print('Mean length of the texts:', np.mean(texts_len))
```

Figure 17 Display length of contents

4.2.3 Preprocess the data

- We are preprocessing fake set by remove all punctuations from the text.

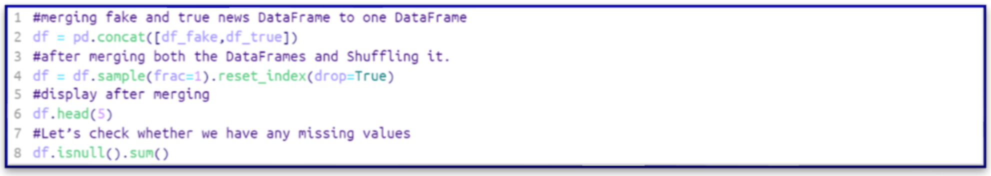


```python
#definition to remove all punctuations from the text
def remove_punct(text):
    return ("".join([ch for ch in text if ch not in st.punctuation]))
#remove punctuations from fake news
#the overwtite to text in fake news DataFrame
df_fake['text'] = df_fake['text'].apply(lambda x: remove_punct(x))
#show head to check text after remove punctuations
df_fake.head()
```

Figure 18 Preprocess data

4.2.4 Merge data

- Merge fake data and true data after preprocessing.

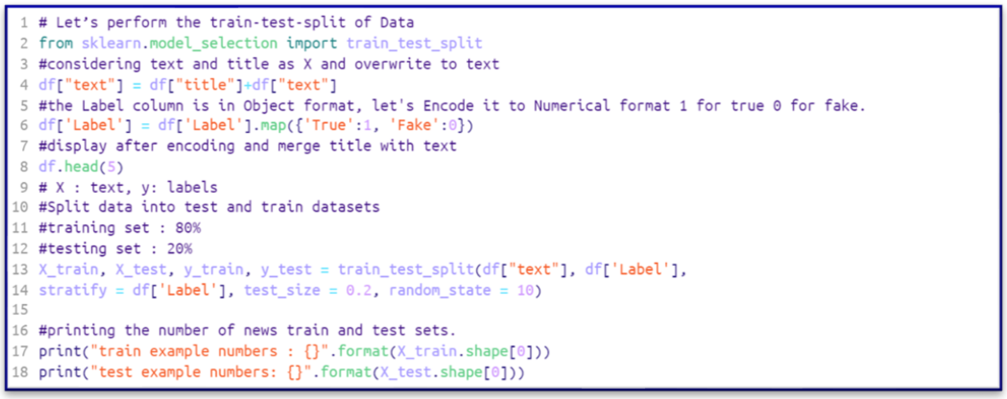


```python
#merging fake and true news DataFrame to one DataFrame
df = pd.concat([df_fake,df_true])
#after merging both the DataFrames and Shuffling it.
df = df.sample(frac=1).reset_index(drop=True)
#display after merging
df.head(5)
#Let's check whether we have any missing values
df.isnull().sum()
```

Figure 19 Merge data

4.2.5 Splitting the data into train and test

- Splitting dataset into two subsets train set and test set (80:20).

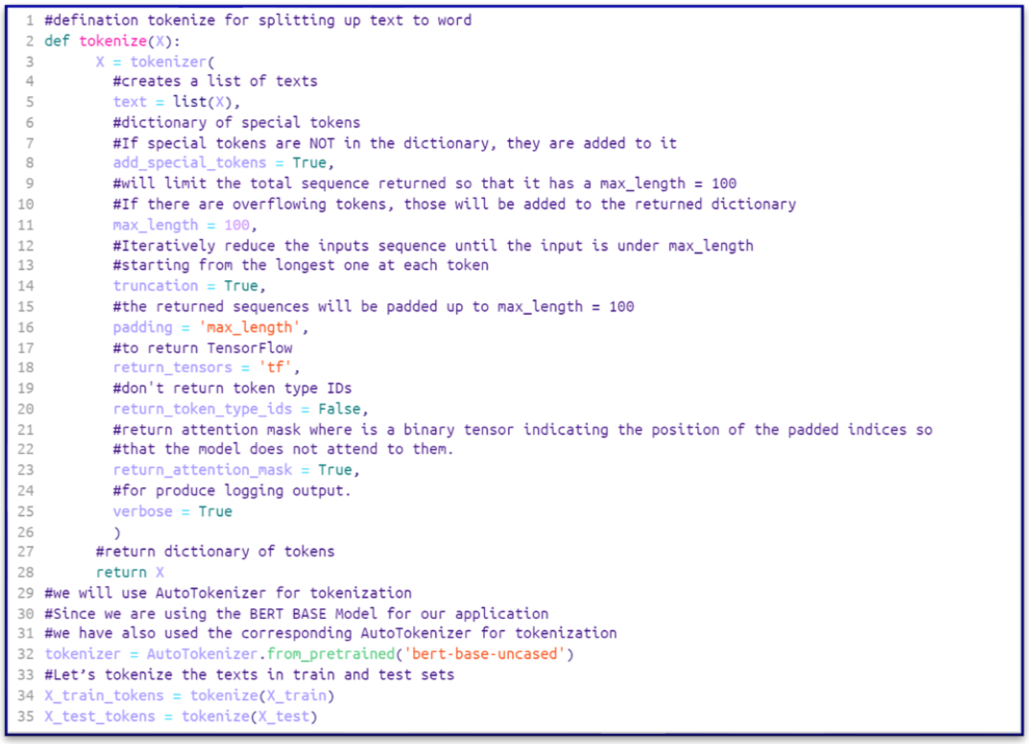


```python
# Let's perform the train-test-split of Data
from sklearn.model_selection import train_test_split
#considering text and title as X and overwrite to text
df["text"] = df["title"]+df["text"]
#the Label column is in Object format, let's Encode it to Numerical format 1 for true 0 for fake.
df['Label'] = df['Label'].map({'True':1, 'Fake':0})
#display after encoding and merge title with text
df.head(5)
# X : text, y: labels
#Split data into test and train datasets
#training set : 80%
#testing set : 20%
X_train, X_test, y_train, y_test = train_test_split(df["text"], df['Label'],
stratify = df['Label'], test_size = 0.2, random_state = 10)

#printing the number of news train and test sets.
print("train example numbers : {}".format(X_train.shape[0]))
print("test example numbers: {}".format(X_test.shape[0]))
```

Figure 20 Data splitting

4.2.6 Tokenization

- Tokenizing is splitting strings in sub-word token strings so we prepare definition to tokenizing content into token of word with using pre-trained *AutoTokenizer* that corresponding BERT pre-trained model.



```
#defination tokenize for splitting up text to word
def tokenize(X):
      X = tokenizer(
        #creates a list of texts
        text = list(X),
        #dictionary of special tokens
        #If special tokens are NOT in the dictionary, they are added to it
        add_special_tokens = True,
        #will limit the total sequence returned so that it has a max_length = 100
        #If there are overflowing tokens, those will be added to the returned dictionary
        max_length = 100,
        #Iteratively reduce the inputs sequence until the input is under max_length
        #starting from the longest one at each token
        truncation = True,
        #the returned sequences will be padded up to max_length = 100
        padding = 'max_length',
        #to return TensorFlow
        return_tensors = 'tf',
        #don't return token type IDs
        return_token_type_ids = False,
        #return attention mask where is a binary tensor indicating the position of the padded indices so
        #that the model does not attend to them.
        return_attention_mask = True,
        #for produce logging output.
        verbose = True
        )
      #return dictionary of tokens
      return X
#we will use AutoTokenizer for tokenization
#Since we are using the BERT BASE Model for our application
#we have also used the corresponding AutoTokenizer for tokenization
tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')
#Let’s tokenize the texts in train and test sets
X_train_tokens = tokenize(X_train)
X_test_tokens = tokenize(X_test)
```

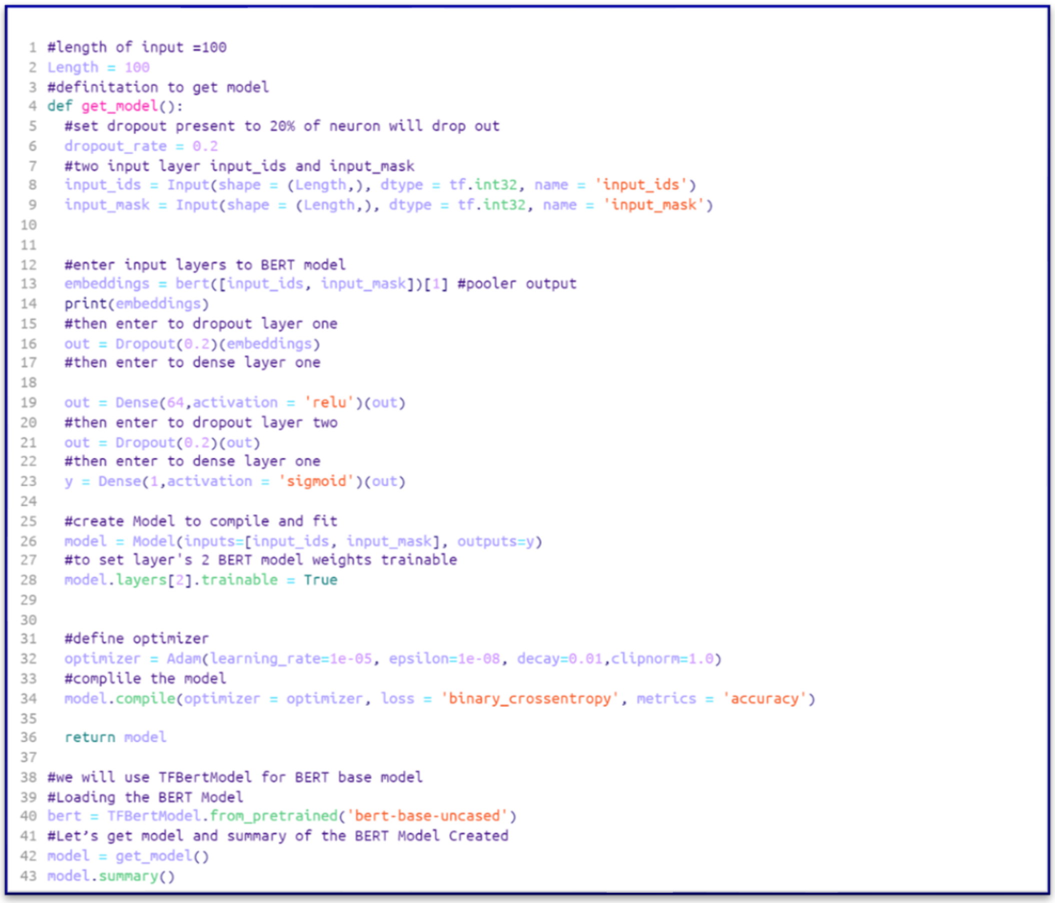
Figure 21 Data tokenization

4.3 Model development

The BERT deep learning model is widely used in NLP task such as question answering ,abstract summarization and sentence prediction. So in this section we explain how we implement the model with pre-trained BERT model. The workflow of our model is illustrated in Figure 8. Firstly, we load pre-trained *AutoTokenizer* to prepare input of model. Our model has two input the token and mask of token. Then we load pre-trained *TFBertModel* with *base* size that has 12 layers in the Encoder stack. Then we add two dropout layer with rate 0.2 and two dense layer. The model starts with wights from BERT model then enter to dropout layer and dense layer. So we benefit from the weight that are learned previously for training classifier.

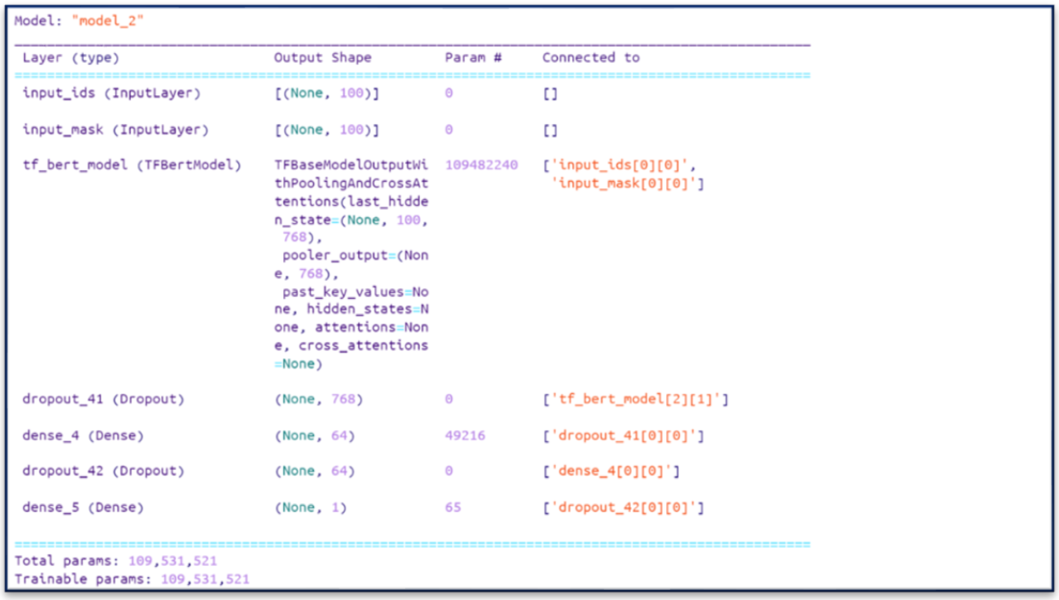
4.3.1 Design BERT Model

- Build model with using pre-trained BERT model .



```
#length of input =100
Length = 100
#definitation to get model
def get_model():
  #set dropout present to 20% of neuron will drop out
  dropout_rate = 0.2
  #two input layer input_ids and input_mask
  input_ids = Input(shape = (Length,), dtype = tf.int32, name = 'input_ids')
  input_mask = Input(shape = (Length,), dtype = tf.int32, name = 'input_mask')


  #enter input layers to BERT model
  embeddings = bert([input_ids, input_mask])[1] #pooler output
  print(embeddings)
  #then enter to dropout layer one
  out = Dropout(0.2)(embeddings)
  #then enter to dense layer one

  out = Dense(64,activation = 'relu')(out)
  #then enter to dropout layer two
  out = Dropout(0.2)(out)
  #then enter to dense layer one
  y = Dense(1,activation = 'sigmoid')(out)

  #create Model to compile and fit
  model = Model(inputs=[input_ids, input_mask], outputs=y)
  #to set layer's 2 BERT model weights trainable
  model.layers[2].trainable = True


  #define optimizer
  optimizer = Adam(learning_rate=1e-05, epsilon=1e-08, decay=0.01,clipnorm=1.0)
  #complile the model
  model.compile(optimizer = optimizer, loss = 'binary_crossentropy', metrics = 'accuracy')

  return model

#we will use TFBertModel for BERT base model
#Loading the BERT Model
bert = TFBertModel.from_pretrained('bert-base-uncased')
#Let's get model and summary of the BERT Model Created
model = get_model()
model.summary()
```

Figure 22 Build model

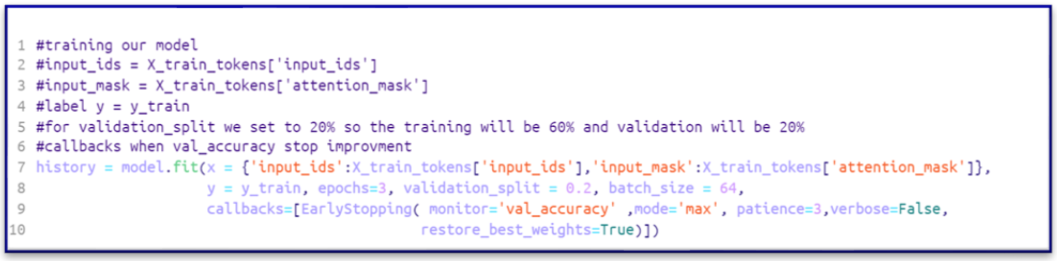


```
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_ids (InputLayer)         [(None, 100)]        0           []

 input_mask (InputLayer)        [(None, 100)]        0           []

 tf_bert_model (TFBertModel)    TFBaseModelOutputWi  109482240   ['input_ids[0][0]',
                                thPoolingAndCrossAt               'input_mask[0][0]']
                                tentions(last_hidde
                                n_state=(None, 100,
                                 768),
                                 pooler_output=(Non
                                e, 768),
                                 past_key_values=No
                                ne, hidden_states=N
                                one, attentions=Non
                                e, cross_attentions
                                =None)

 dropout_41 (Dropout)           (None, 768)          0           ['tf_bert_model[2][1]']

 dense_4 (Dense)                (None, 64)           49216       ['dropout_41[0][0]']

 dropout_42 (Dropout)           (None, 64)           0           ['dense_4[0][0]']

 dense_5 (Dense)                (None, 1)            65          ['dropout_42[0][0]']

==================================================================================================
Total params: 109,531,521
Trainable params: 109,531,521
```

Figure 23 model summery

4.3.2 FakeNews classification using BERT model

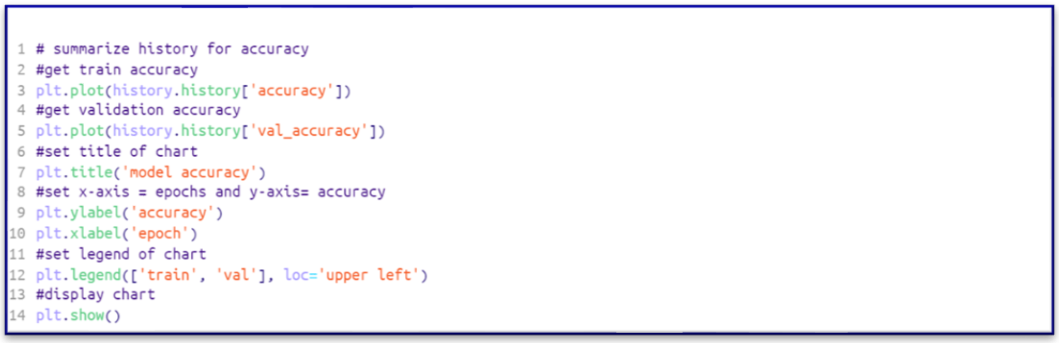
- Train model for train set where equal 80% of data also its split to validation set so the train will be 60% and validation 20%.



```
#training our model
#input_ids = X_train_tokens['input_ids']
#input_mask = X_train_tokens['attention_mask']
#label y = y_train
#for validation_split we set to 20% so the training will be 60% and validation will be 20%
#callbacks when val_accuracy stop improvment
history = model.fit(x = {'input_ids':X_train_tokens['input_ids'],'input_mask':X_train_tokens['attention_mask']},
                    y = y_train, epochs=3, validation_split = 0.2, batch_size = 64,
                    callbacks=[EarlyStopping( monitor='val_accuracy' ,mode='max', patience=3,verbose=False,
                                             restore_best_weights=True)])
```

Figure 24 Train model

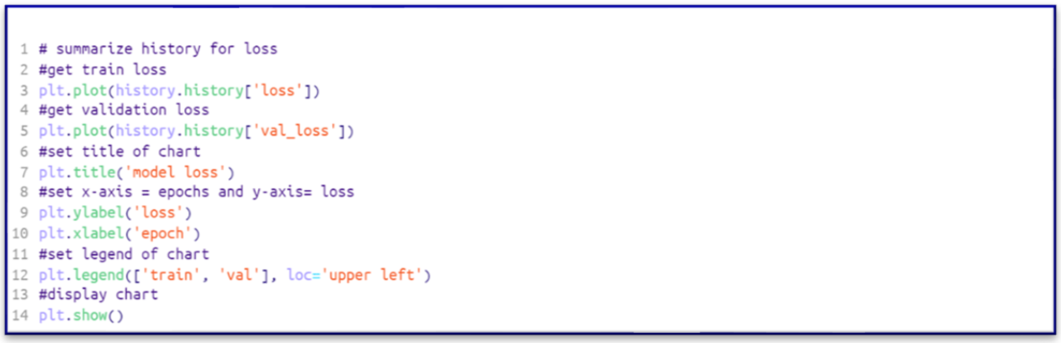
- Accuracy curves for training and validation



```
# summarize history for accuracy
#get train accuracy
plt.plot(history.history['accuracy'])
#get validation accuracy
plt.plot(history.history['val_accuracy'])
#set title of chart
plt.title('model accuracy')
#set x-axis = epochs and y-axis= accuracy
plt.ylabel('accuracy')
plt.xlabel('epoch')
#set legend of chart
plt.legend(['train', 'val'], loc='upper left')
#display chart
plt.show()
```

Figure 25 Train and validation accuracy

- Loss curves for training and validation

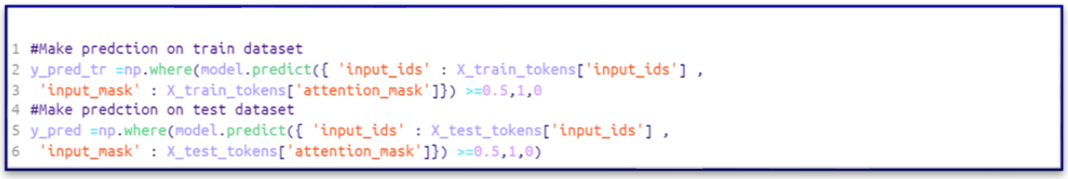


```
# summarize history for loss
#get train loss
plt.plot(history.history['loss'])
#get validation loss
plt.plot(history.history['val_loss'])
#set title of chart
plt.title('model loss')
#set x-axis = epochs and y-axis= loss
plt.ylabel('loss')
plt.xlabel('epoch')
#set legend of chart
plt.legend(['train', 'val'], loc='upper left')
#display chart
plt.show()
```

Figure 26 Train and validation loss

4.4 Model evaluation

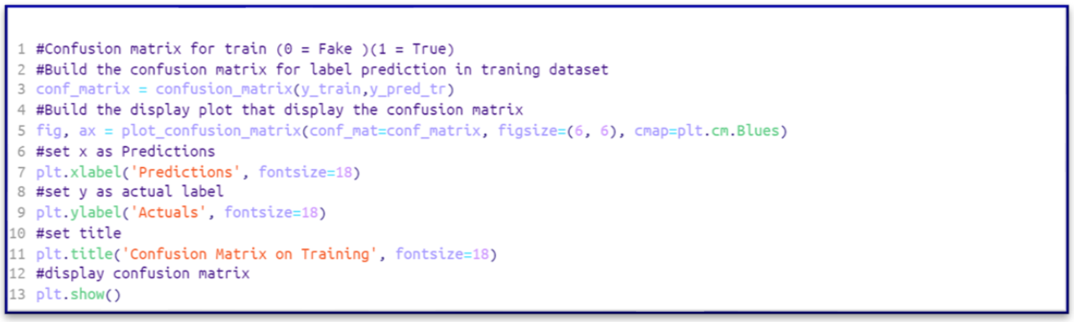
- Make prediction for train and test sets.



```
#Make predction on train dataset
y_pred_tr =np.where(model.predict({ 'input_ids' : X_train_tokens['input_ids'] ,
 'input_mask' : X_train_tokens['attention_mask']}) >=0.5,1,0)
#Make predction on test dataset
y_pred =np.where(model.predict({ 'input_ids' : X_test_tokens['input_ids'] ,
 'input_mask' : X_test_tokens['attention_mask']}) >=0.5,1,0)
```

Figure 27 Make prediction of data.

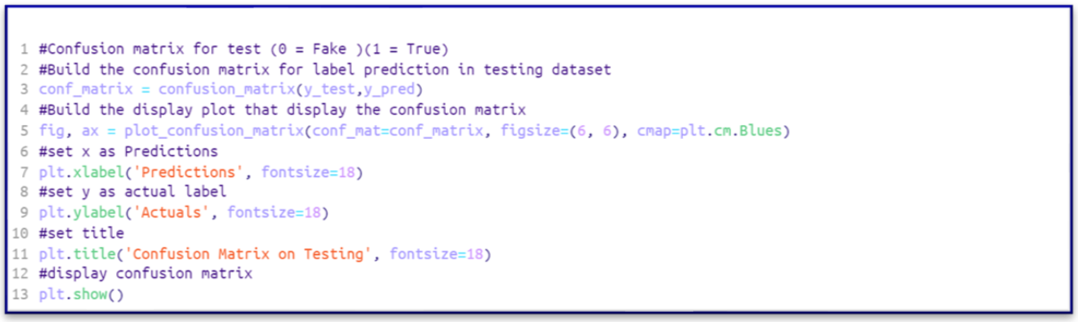
- Confusion matrix of training.



```python
#Confusion matrix for train (0 = Fake )(1 = True)
#Build the confusion matrix for label prediction in traning dataset
conf_matrix = confusion_matrix(y_train,y_pred_tr)
#Build the display plot that display the confusion matrix
fig, ax = plot_confusion_matrix(conf_mat=conf_matrix, figsize=(6, 6), cmap=plt.cm.Blues)
#set x as Predictions
plt.xlabel('Predictions', fontsize=18)
#set y as actual label
plt.ylabel('Actuals', fontsize=18)
#set title
plt.title('Confusion Matrix on Training', fontsize=18)
#display confusion matrix
plt.show()
```

Figure 28 Confusion matrix on training

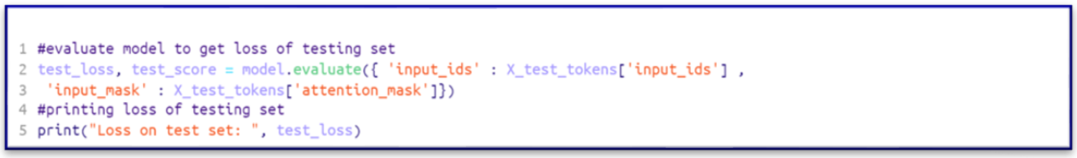
- Confusion matrix of testing.



```python
#Confusion matrix for test (0 = Fake )(1 = True)
#Build the confusion matrix for label prediction in testing dataset
conf_matrix = confusion_matrix(y_test,y_pred)
#Build the display plot that display the confusion matrix
fig, ax = plot_confusion_matrix(conf_mat=conf_matrix, figsize=(6, 6), cmap=plt.cm.Blues)
#set x as Predictions
plt.xlabel('Predictions', fontsize=18)
#set y as actual label
plt.ylabel('Actuals', fontsize=18)
#set title
plt.title('Confusion Matrix on Testing', fontsize=18)
#display confusion matrix
plt.show()
```

Figure 29 Confusion matrix on testing

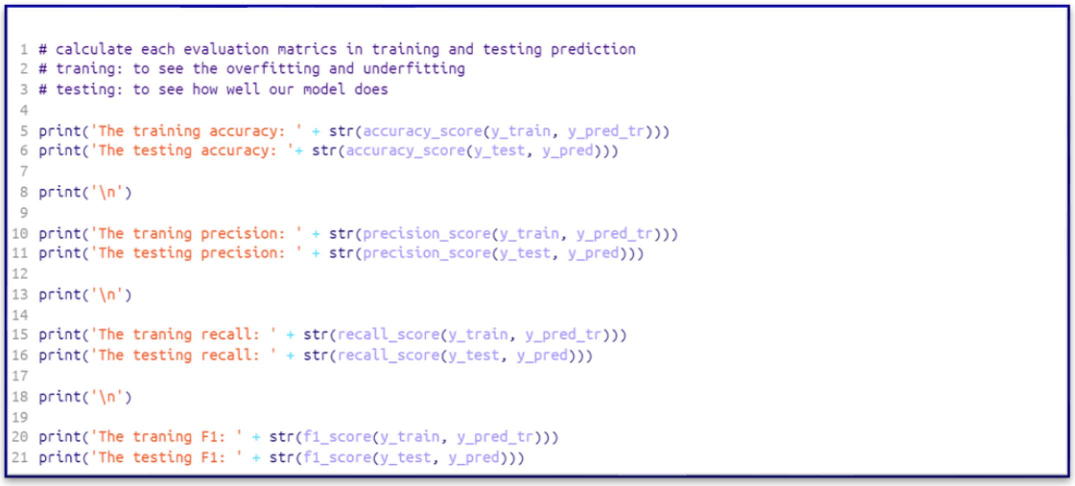
- Loss of test set.



```python
#evaluate model to get loss of testing set
test_loss, test_score = model.evaluate({ 'input_ids' : X_test_tokens['input_ids'] ,
 'input_mask' : X_test_tokens['attention_mask']})
#printing loss of testing set
print("Loss on test set: ", test_loss)
```

Figure 30 Loss of test

- Accuracy, precision, recall and F1 score on train and test dataset.

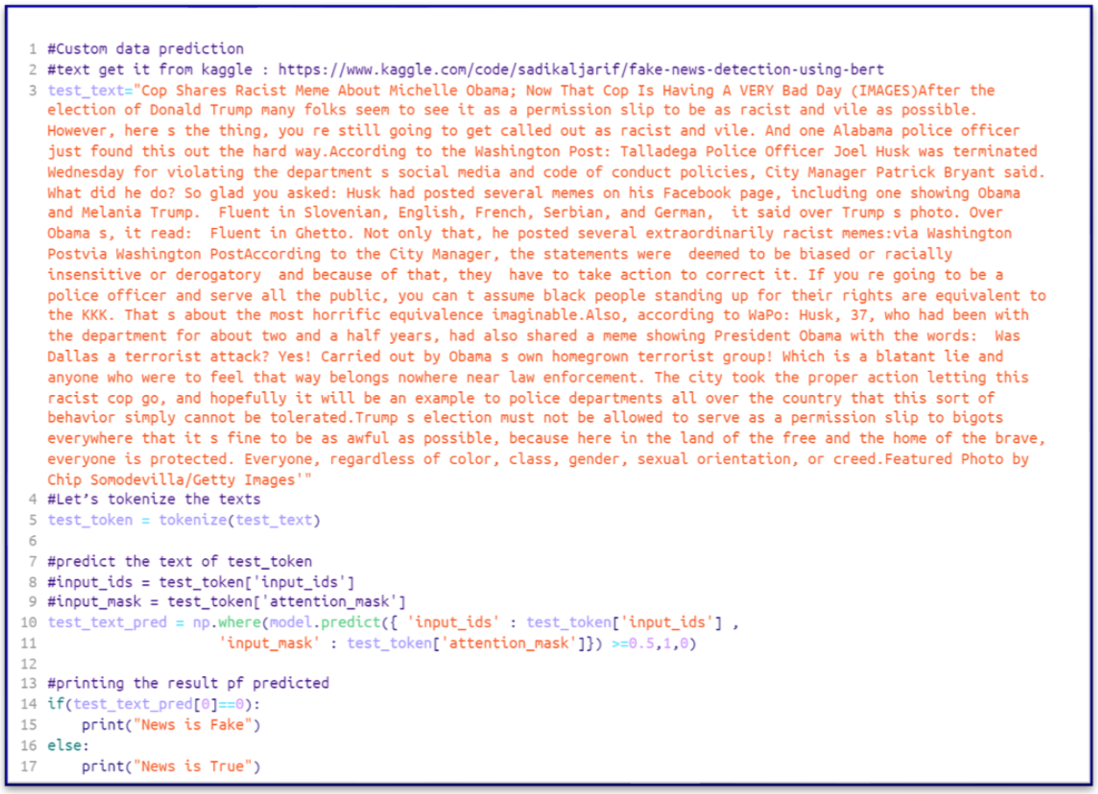


```python
# calculate each evaluation matrics in training and testing prediction
# traning: to see the overfitting and underfitting
# testing: to see how well our model does

print('The training accuracy: ' + str(accuracy_score(y_train, y_pred_tr)))
print('The testing accuracy: '+ str(accuracy_score(y_test, y_pred)))

print('\n')

print('The traning precision: ' + str(precision_score(y_train, y_pred_tr)))
print('The testing precision: ' + str(precision_score(y_test, y_pred)))

print('\n')

print('The traning recall: ' + str(recall_score(y_train, y_pred_tr)))
print('The testing recall: ' + str(recall_score(y_test, y_pred)))

print('\n')

print('The traning F1: ' + str(f1_score(y_train, y_pred_tr)))
print('The testing F1: ' + str(f1_score(y_test, y_pred)))
```

Figure 31 Accuracy, precision, recall and F1 score on train and test dataset.

4.5 Model prediction/inference

- Predict new sample for model inference.

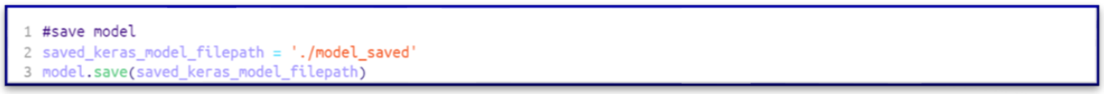


```
#Custom data prediction
#text get it from kaggle : https://www.kaggle.com/code/sadikaljarif/fake-news-detection-using-bert
test_text="Cop Shares Racist Meme About Michelle Obama; Now That Cop Is Having A VERY Bad Day (IMAGES)After the
election of Donald Trump many folks seem to see it as a permission slip to be as racist and vile as possible.
However, here s the thing, you re still going to get called out as racist and vile. And one Alabama police officer
just found this out the hard way.According to the Washington Post: Talladega Police Officer Joel Husk was terminated
Wednesday for violating the department s social media and code of conduct policies, City Manager Patrick Bryant said.
What did he do? So glad you asked: Husk had posted several memes on his Facebook page, including one showing Obama
and Melania Trump.  Fluent in Slovenian, English, French, Serbian, and German,  it said over Trump s photo. Over
Obama s, it read:  Fluent in Ghetto. Not only that, he posted several extraordinarily racist memes:via Washington
Postvia Washington PostAccording to the City Manager, the statements were  deemed to be biased or racially
insensitive or derogatory  and because of that, they  have to take action to correct it. If you re going to be a
police officer and serve all the public, you can t assume black people standing up for their rights are equivalent to
the KKK. That s about the most horrific equivalence imaginable.Also, according to WaPo: Husk, 37, who had been with
the department for about two and a half years, had also shared a meme showing President Obama with the words:  Was
Dallas a terrorist attack? Yes! Carried out by Obama s own homegrown terrorist group! Which is a blatant lie and
anyone who were to feel that way belongs nowhere near law enforcement. The city took the proper action letting this
racist cop go, and hopefully it will be an example to police departments all over the country that this sort of
behavior simply cannot be tolerated.Trump s election must not be allowed to serve as a permission slip to bigots
everywhere that it s fine to be as awful as possible, because here in the land of the free and the home of the brave,
everyone is protected. Everyone, regardless of color, class, gender, sexual orientation, or creed.Featured Photo by
Chip Somodevilla/Getty Images'"
#Let's tokenize the texts
test_token = tokenize(test_text)

#predict the text of test_token
#input_ids = test_token['input_ids']
#input_mask = test_token['attention_mask']
test_text_pred = np.where(model.predict({ 'input_ids' : test_token['input_ids'] ,
                    'input_mask' : test_token['attention_mask']}) >=0.5,1,0)

#printing the result pf predicted
if(test_text_pred[0]==0):
    print("News is Fake")
else:
    print("News is True")
```

Figure 32 Predict new sample.

4.6 Save the model

- We saved our model in a file



```
#save model
saved_keras_model_filepath = './model_saved'
model.save(saved_keras_model_filepath)
```

Figure 33 save model

5. Discussion and Analyze the Results

In this section we want to explain and describe about the main findings after implementation in sec 5.1 then discuss and interprets the results in sec 5.2.

5.1 Results:

First of all, the dataset was already cleaned and processed. Except that the fake set has punctuation. Therefore, we conducted two experiments in order to see the effectiveness of the model when and when not we remove the punctuation from the fake sets. Table 3 shows the results of both experiments. Note that, in both of these experiments, we formulate the input as title + content of each article. In addition, we perform the tokenization for preprocessing fake and real sets (with max length equal to 100).

EX #	Input	Preprocessing	Test Acc	Test loss	Test Precision	Test Recall	Test F1
EX1	Title + content	Tokenization	99.96%	0.0	1.0	0.99	0.99
EX2	Title + content	Tokenization + Remove punctuation from fake set only.	100%	0.0	1.0	1.0	1.0

Table 3 Experiments results

During training the model, we use the validation dataset to see how our model is doing in only three epochs. In the left of Figure 34, 36 show the loss between training and validation dataset. While in the right of Figure 34 and 36, the accuracy between training and validation dataset. In addition, we calculate the confusion matrix for the training data, as presented in the left of Figure 35, 37. After training the model, it has been evaluated using testing dataset. Table 3 shows the basic evaluation metrics for the classification tasks. In addition, we calculate the confusion matrix for the testing data, as presented in the right of Figure 35, 37.

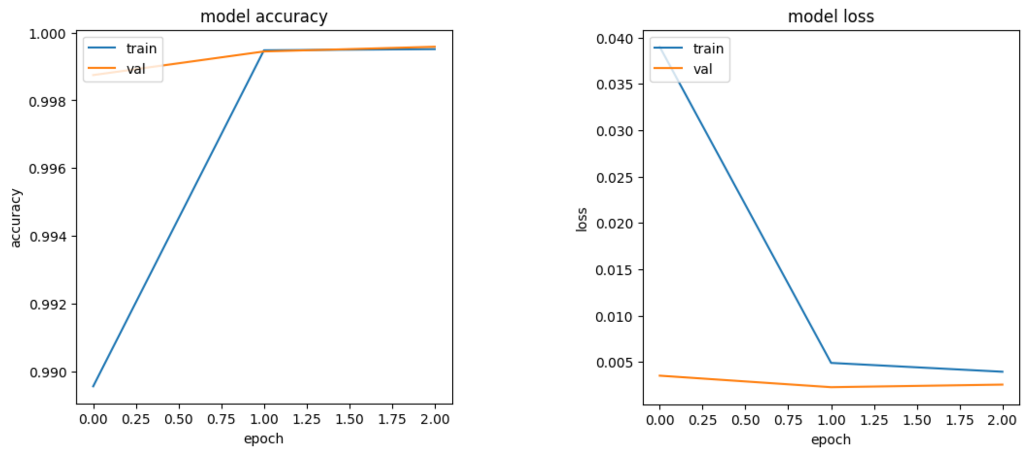


Figure 34 Accuracies and losses of experiments 1

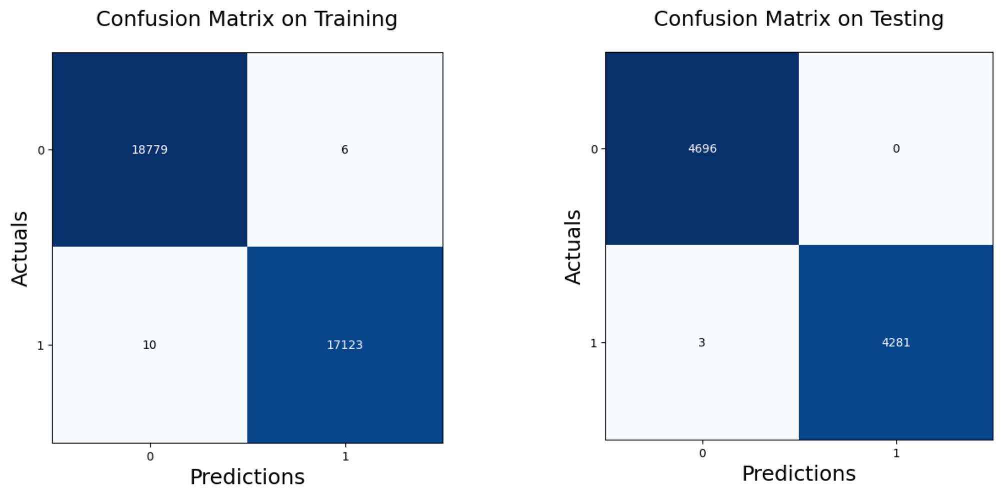


Figure 35 Confusion matrices of experiments 1

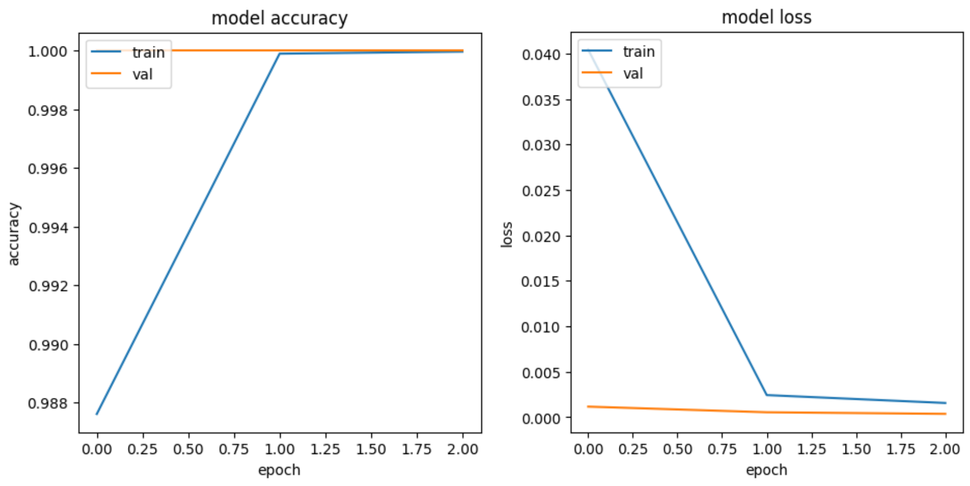


Figure 36 Accuracies and losses of experiments 2

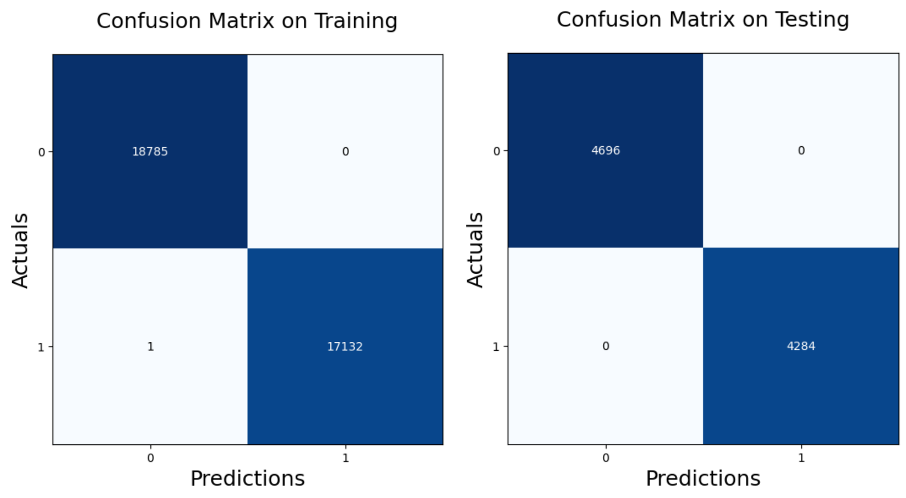


Figure 37 Confusion matrices of experiments 2

5.2 Discussions:

After we conduct two experiments, one with removing the punctuation from fake set and another without. As shown in Table3, we conclude that both of them are close to each other in terms of performance, with high accuracy. Since our task is Fake news classification, our attention will be on accuracy, precision and recall. Because this evaluation metrics are used mainly used in the literatures of fake news classification, especially when the dataset is balanced as in our case. As shown in Table3, the second experiment outperforms the first one in term of accuracy, precision and recall.

This is due to the use of pre-trained BERT model with suitable and enough size of dataset. BERT outperforms other pre-trained models to this dataset based on this repository [7], where a comparison between LSTM, Bidirectional LSTM, CNN-BiLSTM and BERT are conducted on the same datasets. The results between these four models are comparable, but BERT is the highest one.

Another reason may be influenced to this high performance is the input to the model, where the title and content of the articles are considered. However, some implementations of previous projects to the same dataset use only the content or only the title as input to the model in order to simplifying the complexity, which may give up important terms that help to make the final decision.

Regarding the learning process, as shown in Figures 34 and 36, it shown that the learning was so fast to reach to the best solution, even though the model trained with only three epochs, this is because we used the pre-trained weights as initial weights. The losses of train and validation dataset are increasingly decreasing, ending up with convergence between their values.

6. Conclusion

Fake news detection considered as one of the most currently problem that faced many people in social media. Thus, in this project, we aim to build a deep learning model that utilize BERT as pre-trained model to transfer the learning into our task. The model takes the title and content of a news article, then classifies this article as fake or real. After implementation, we achieve high performance, in term of accuracy, our model performs 100% on testing data with punctuation removal processing and 99.96% on testing data without punctuation removal processing. As a future work, we will implement different types of pre-trained model on same dataset and compare the results between them.

7. Challenges

1. Understanding the concepts related to NLP.
2. Choosing the best pre-trained model among many NLP models for given dataset.
3. Long run-time.
4. Choosing parameters values for tokenizer was challenging since its first-time deal with tokenization process.
5. Errors in save model step due to *TypeError* in package joblib.
6. Find the most influence performance measurement for our task between TP, TN, FP, FN where the parameters are used in the evaluation of specificity, sensitivity, and accuracy.

8.Tool and technologies

In our project, we use a lot of tools and technologies, we illustrate it in Figure 38.

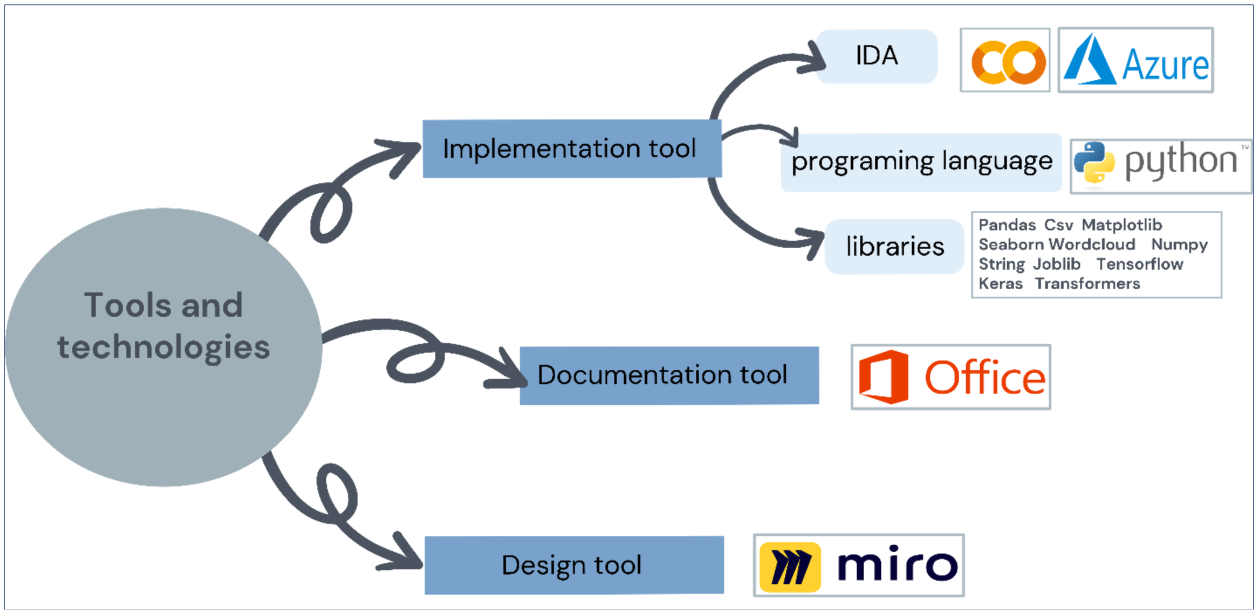


Figure 38 Tools and technologies used in project

9. References

[1] “Fake News Detection Datasets | ISOT Research Lab.” *Online Academic Community | Initiative of the University of Victoria*, https://onlineacademiccommunity.uvic.ca/isot/2022/11/27/fake-news-detection-datasets/. Accessed 11 Feb. 2023.

[2] Kr, Abhijit. “Fine-Tuning BERT with Masked Language Modeling -.” *Analytics Vidhya*, https://www.facebook.com/AnalyticsVidhya/, 24 Sept. 2022, https://www.analyticsvidhya.com/blog/2022/09/fine-tuning-bert-with-masked-language-modeling/.

[3] Rogel-Salazar, J. “Transformers Models in Machine Learning: Self-Attention to the Rescue.” *Domino Data Lab | Unleash Data Science at Scale*, Domino Data Lab, 25 May 2022, https://www.dominodatalab.com/blog/transformers-self-attention-to-the-rescue.

[4] Kafritsas, Nikos. “Block-Recurrent Transformer: LSTM and Transformer Combined.” Medium, 6 July 2022, https://towardsdatascience.com/block-recurrent-transformer-lstm-and-transformer-combined-ec3e64af971a.

[5] “Attention Is All You Need.” ArXiv.Org, https://doi.org/10.48550/arxiv.1706.03762. Accessed 11 Feb. 2023.

[6] Alammar, Jay. “The Illustrated BERT, ELMo, and Co. (How NLP Cracked Transfer Learning) – Jay Alammar – Visualizing Machine Learning One Concept at a Time.” Jay Alammar – Visualizing Machine Learning One Concept at a Time., https://jalammar.github.io/illustrated-bert/. Accessed 11 Feb. 2023.

[7] Wu, TonyHY. “Fake News Detection Using a BERT-Based Deep Learning Approach.” GitHub, 15 Jan. 2023, github.com/wutonytt/Fake-News-Detection. Accessed 10 Feb. 2023.

[8] Devlin, Jacob, et al. BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. 2019.

[9] face Hugging, “Tokenizer — transformers 2.11.0 documentation,” *huggingface.co*, 2020. https://huggingface.co/transformers/v2.11.0/main_classes/tokenizer.html (accessed Feb. 11, 2023).

[10] B. Lutkevich, “What is BERT (Language Model) and How Does It Work?,” *SearchEnterpriseAI*, Jan. 2020. https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model

[11] D. M. J. Lazer et al., “The Science of Fake News,” Science, vol. 359, no. 6380, pp. 1094–1096, Mar. 2018, https://10.1126/science.aao2998

[12] Arkvik, Isabel. “What Is Fake News?” Visma International Blog – Technology, Business & Life at Visma, 24 June 2021, www.visma.com/blog/what-is-fake-news/.

[13] “Fine-Tuning a Neural Network Explained.” Deeplizard.com, deeplizard.com/learn/video/5T-iXNNiwIs#:~:text=Fine%2Dtuning%20is%20a%20way.

[14] “BERT Cased vs BERT Uncased.” OpenGenus IQ: Computing Expertise & Legacy, iq.opengenus.org/bert-cased-vs-bert-uncased/#:~:text=In%20BERT%20uncased%2C%20the%20text. Accessed 10 Feb. 2023.

[15] “Bert-Base-Uncased · Hugging Face.” Huggingface.co, huggingface.co/bert-base-uncased.