Configuration Manual

MSc Research Project
Data Analytics

Nikhil Vishnupant Deshmukh
Student ID: x21232946

School of Computing
National College of Ireland

Supervisor: Vikas Tomer

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Nikhil Vishnupant Deshmukh		
Student ID:	X21232946		
Programme:	Data Analytics	**Year:**	2022-23
Module:	MSc Research Project		
Lecturer:	Vikas Tomer		
Submission Due Date:	18th September 2023		
Project Title:	Transfer Learning for Identification of disaster tweets using fine-tuning DistilBert		
Word Count:	1169	**Page Count:**	14

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:	Nikhil Vishnupant Deshmukh
Date:	18th September 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

Nikhil Vishnupant Deshmukh
Student ID:x21232946

1. Introduction:

This is the research project configuration manual for "**Transfer Learning for Identification of disaster tweets using fine-tuning DistilBert**" This Configuration Manual compiles the necessary requirements for reproducing the research and their results in a particular scenario. A brief overview of the data source for data importation, exploratory data analysis (EDA), and data pre-processing is provided.

2. System Requirements:

This section goes into detail about the particular hardware and software requirements for using the research.

2.1 Hardware Requirements:

The system on which this research project is developed and carried out has the following hardware configuration:

Operating System	Windows 11
Processor	11th Gen Intel(R) Core(TM) i3-1125G4 @ 2.00GHz 2.00 GHz
Storage	500 GB
RAM	8.00 GB

2.2 Software Requirements:

To conduct the experiments, the following software is used**:**

Integrated Development Environment	Google Colab
Scripting Language	Python
Cloud Storage	Google Drive
Modelling Library	TensorFlow, Keras

3. Environment Setup:

3.1 Google Colaboratory:

The first thing need to do is setting up the IDE. First visit the official website of Google Colaboratory and configure.

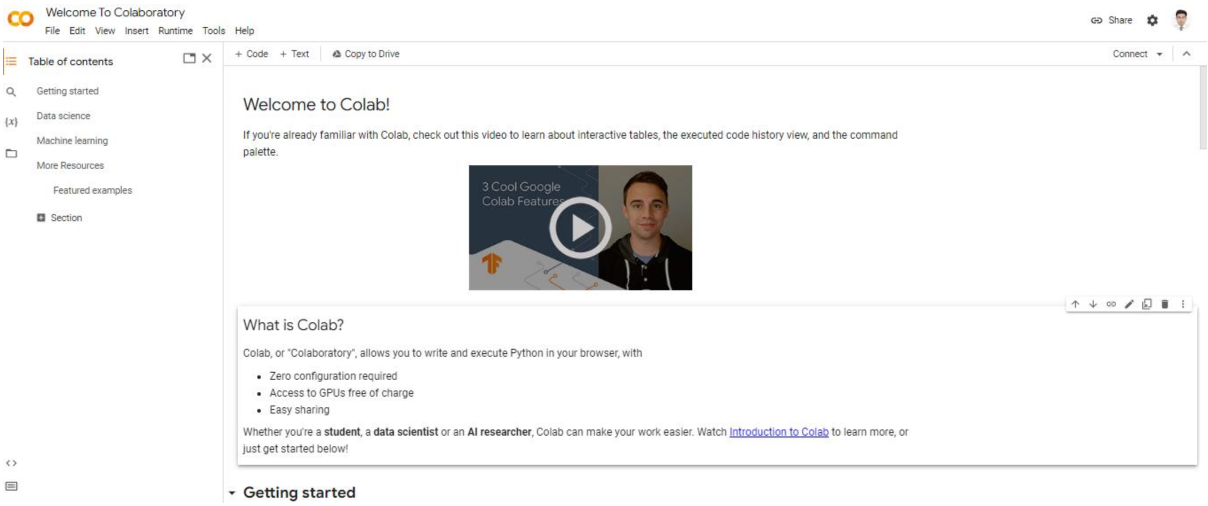

Figure 1: Google Colab Setup

4. Data Selection:

The information came from a publicly accessible Kaggle source. https://www.kaggle.com/competitions/nlp-getting-started/data?select=train.csv is the dataset used.

5. Data Exploration:

Figure 2 and Figure 3 shows the necessary libraries need to be installed to execute.

```
!pip install keras==2.10.0
!pip install tensorflow==2.10.0
!pip install h5py==3.7.0
```

Figure 2 . Necessary Python libraries

```
!pip install transformers
# Keras tuner
!pip install -q -U keras-tuner
```

Figure 3. Necessary Python libraries

Figure 4 indicates that all the libraries and packages that are required for data visualization to data processing.

```python
import numpy as np
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 keras.preprocessing.text import Tokenizer
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
from keras.layers import Conv1D, Activation
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
# Warning
import warnings
warnings.filterwarnings('ignore')
import keras_tuner as kt
from transformers import AutoTokenizer
from transformers import TFDistilBertModel, DistilBertConfig, RobertaConfig, TFRobertaModel
max_len=150
BATCH_SIZE = 64
```

Figure 4 . Python Packages

Figure 5 shows the code snippet that used to do the data visualization for the most common keywords.

```python
#Most common keywords
common_keywords=df["keyword"].value_counts()[:20].to_frame()
fig=plt.figure(figsize=(15,6))
sns.barplot(data=common_keywords,x=common_keywords.index,y="keyword",palette="viridis")
plt.title("Most common keywords",size=16)
plt.xticks(rotation=70,size=12);
plt.show()
```

Figure 5.Data Visualization for most common keywords.

Figure 6 is the barplot which shows the results for data visualization.

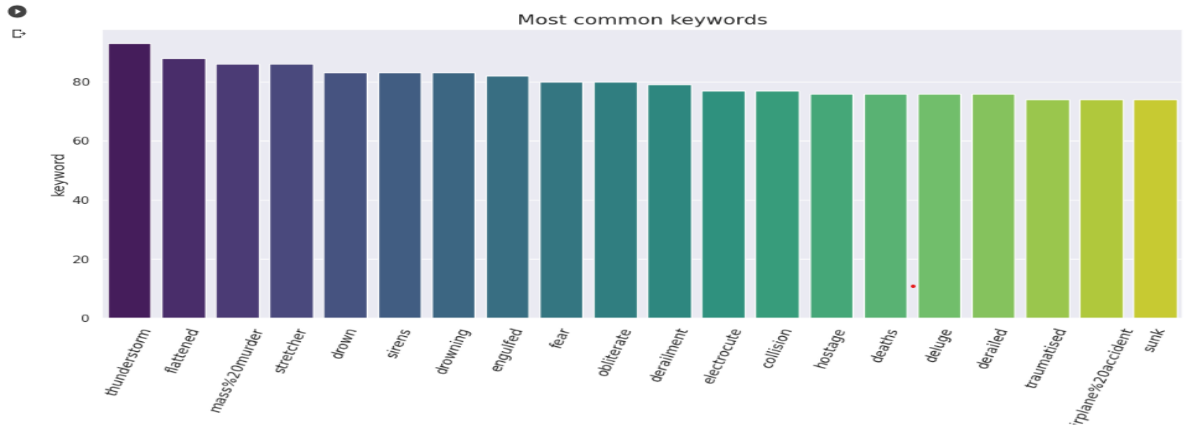

Figure 6. Barplot for Most Common Keywords.

Now, in the data visualization word cloud is important. Figure 7 shows the code snippet for generating the word clouds.

```
#Wordcloud
disaster_tweets = df[df['target']==1]['text']
non_disaster_tweets = df[df['target']==0]['text']

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=[16, 8])
wordcloud1 = WordCloud( background_color='white',
                        width=600,
                        height=400).generate(" ".join(disaster_tweets))
ax1.imshow(wordcloud1)
ax1.axis('off')
ax1.set_title('Disaster Tweets',fontsize=40);

wordcloud2 = WordCloud( background_color='white',
                        width=600,
                        height=400).generate(" ".join(non_disaster_tweets))
ax2.imshow(wordcloud2)
ax2.axis('off')
ax2.set_title('Non Disaster Tweets',fontsize=40);
plt.show()
```

Figure 7. Word Cloud Code Snippet.

Figure 8 and Figure 9 shows the visualization results obtained from above code.

Figure 8 . Word Cloud for Disaster Tweets

Figure 9 . Word Cloud for Non-Disaster Tweets

Now comes the text cleaning part. The unnecessary punctuations are removed from the data using text cleaing.Figure 10 shows the code snippet for the text cleaning part.

```
#Text Cleaning
# lowering the text
df.text=df.text.apply(lambda x:x.lower() )

#removing square brackets
df.text=df.text.apply(lambda x:re.sub('\[.*?\]', '', x) )
df.text=df.text.apply(lambda x:re.sub('<.*?>+', '', x) )

#removing hyperlink
df.text=df.text.apply(lambda x:re.sub('https?://\S+|www\.\S+', '', x) )

#removing puncuation
df.text=df.text.apply(lambda x:re.sub('[%s]' % re.escape(string.punctuation), '', x) )
df.text=df.text.apply(lambda x:re.sub('\n' , '', x) )

#remove words containing numbers
df.text=df.text.apply(lambda x:re.sub('\w*\d\w*' , '', x) )
emoji_pattern = re.compile("["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           "]+", flags=re.UNICODE)
df.text=df.text.apply(lambda x:emoji_pattern.sub(r'', x) )

print(df.text)
```

```
0        communal violence in bhainsa telangana stones ...
```

Figure 10 Text Cleaning Code Snippet.

After the text cleaning, to visualize the length of text code snippet used is shown in the figure 11. Figure 11 also shows the plot for the length of the Keywords in the total data.

```
[ ] text_data = df.text
```

```
text_lengths = []
for disaster_text in text_data:
 text_lengths.append(len(disaster_text))
plt.hist(text_lengths, bins=50)
plt.show()
```

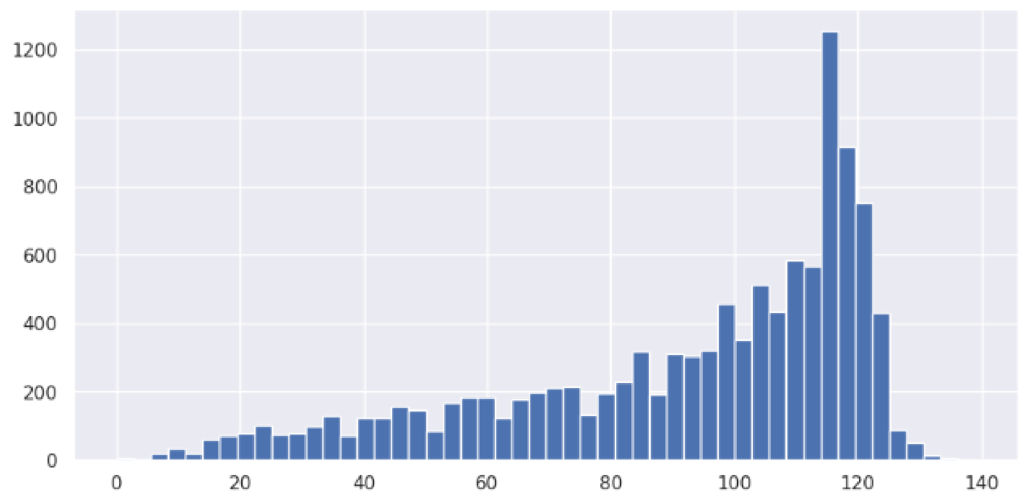

Figure 11 .Code Snippet and Plot for Length of text

6. Models:

In this section , the models applied in this research are shown.

6.1 CNN + LSTM with Glove Embedding:

The below figure shows the code snippet for the word embedding and tokenization done for the model in research CNN + LSTM with Glove Embedding.

```
[ ] embedding_dict={}
    with open('/content/drive/My Drive/disaster_tweets_using_distilbert/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(text_data)
    titles=tok.texts_to_sequences(text_data)
    titles = pad_sequences(titles,maxlen=140,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=140
```

Figure 12. Embedding and Tokenization for CNN+LSTM

Now, further in this research, the training data and testing data needs to be split.

Figure 13 below shows the code block used for the splitting of training data and testing data.

```
[ ] X_train, X_test, y_train, y_test = train_test_split( titles, labels, 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)
    print("Training Data size: ", X_train.shape[0])
    print("Testing data size: ", X_test.shape[0])
    embed_len = 300
```

Figure 13. Train and Test Data Split

Now to build the sequential model, the code block shown in the figure 14 is used. Figure 14 also shows the results for the sequential model.

```
model= Sequential()
model.add(Embedding(vocab_size,embed_len,embeddings_initializer=tf.keras.initializers.Constant(embedding_matrix),input_length=titles.shape[1],trainable=False))
model.add(Conv1D(14, 3, activation='relu'))
model.add(LSTM(12,return_sequences=True))
model.add(Dense(10,activation='relu'))
model.add(Flatten())
model.add(Dense(8,activation='relu'))
model.add(Dense(6,activation='relu'))
model.add(Dense(2,activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adagrad', metrics = ['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 140, 300)          7280100

 conv1d (Conv1D)             (None, 138, 14)           12614

 lstm (LSTM)                 (None, 138, 12)           1296

 dense (Dense)               (None, 138, 10)           130

 flatten (Flatten)           (None, 1380)              0

 dense_1 (Dense)             (None, 8)                 11048

 dense_2 (Dense)             (None, 6)                 54

 dense_3 (Dense)             (None, 2)                 14

=================================================================
```

Figure 14 . Code Snippet and Results for Sequential Model.

6.2 Roberta:

The next model in the research is Roberta. The first part in this is importing the Pretrained model. Figure 15 shows the code snippet for the import of pretrained model.

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

Figure 15. Pretrained Robert base

Now comes the part of data split which is shown the below figure no 16

```
X_train, X_test, y_train, y_test = train_test_split(df.text, df.target, 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)
```

Figure 16 .Data Splitting.

Now ,The figure 17 shows the encoding part of the Roberta, In this figure the results obtained are also captured.

```
train_encodings = tokenizer(X_train.tolist(), truncation=True, max_length=max_len, padding="max_length", return_tensors='tf')
val_encodings = tokenizer(X_val.tolist(), truncation=True, max_length=max_len, padding="max_length", return_tensors='tf')
test_encodings = tokenizer(X_test.tolist(), truncation=True, max_length=max_len, padding="max_length", return_tensors='tf')
print(train_encodings)
```

```
{'input_ids': <tf.Tensor: shape=(7276, 140), dtype=int32, numpy=
array([[    0,  8877,    30, ...,     1,     1,     1],
       [    0,   627,   433, ...,     1,     1,     1],
       [    0,   102,  7917, ...,     1,     1,     1],
       ...,
       [    0,   118,   120, ...,     1,     1,     1],
       [    0,  1594,    52, ...,     1,     1,     1],
       [    0, 10393,     8, ...,     1,     1,     1]], dtype=int32)>, 'attention_mask': <tf.Tensor: shape=(7276, 140), dtype=int32, numpy=
array([[1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       ...,
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0]], dtype=int32)>}
```

```
# Using for Keras-tuner
df_text_encodings_keras = tokenizer(df.text.tolist(), truncation=True, max_length=max_len, padding="max_length", return_tensors="tf")
df_text_encodings_keras
```

```
{'input_ids': <tf.Tensor: shape=(11370, 140), dtype=int32, numpy=
array([[    0, 27217,   337, ...,     1,     1,     1],
       [    0, 29714,  1097, ...,     1,     1,     1],
       [    0, 24858,   661, ...,     1,     1,     1],
       ...,
       [    0,   118,   619, ...,     1,     1,     1],
       [    0,  1638,    54, ...,     1,     1,     1],
       [    0,   267,  5113, ...,     1,     1,     1]], dtype=int32)>, 'attention_mask': <tf.Tensor: shape=(11370, 140), dtype=int32, numpy=
array([[1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
```

Figure 17. Encodings in Roberta.

Now comes tha part of Roberta transformer model creation. Figure 18 shows the code block for the model creation.

```
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_len,),
                                  dtype=tf.int32,
                                  name='input_ids')
    input_attention_layer = keras.Input(shape=(max_len,),
                                        dtype=tf.int32,
                                        name='attention_mask')

    # DistilBERT 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 DistilBERT'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(10,
                                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(10, activation = 'relu')(output)
    # Output layer
    output = keras.layers.Dense(1, 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
```

Figure 18 .Roberta transformer Model Creation.

```
Robertamodel = TFRobertaModel.from_pretrained('roberta-base')
model = create_model(Robertamodel)
#model.summary()
# Compile the model
model.compile(optimizer="rmsprop", loss="binary_crossentropy", metrics=['accuracy'])

history = model.fit(train_tf_dataset, epochs=5, batch_size=BATCH_SIZE, validation_data=val_tf_dataset,verbose=2)
```

Figure 19. Roberta Model

6.3 FineTune Distilbert Transformer:

The next model in the research is Finetuned Distilbert transformer. It also starts with the same process like Roberta. Which is importing the pretrained models.

Figure 20 shows the code snippet for pretrained import.

```
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")

Downloading (…)okenizer_config.json:   0%|          | 0.00/28.0 [00:00<?, ?B/s]
Downloading (…)lve/main/config.json:   0%|          | 0.00/483 [00:00<?, ?B/s]
Downloading (…)solve/main/vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
Downloading (…)/main/tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
```

Figure 20. Pretrained Distilbert

Now for the data spit, the code snippet used in the figure 21 is used.

```
[ ] X_train, X_test, y_train, y_test = train_test_split(df.text, df.target, 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)
```

Figure 21. Data Split

After data splitting part, the model needs to do encoding. And code block in the Figure 22 below shows the encoding and the results of it.

```
train_encodings = tokenizer(X_train.tolist(), truncation=True, max_length=max_len, padding="max_length", return_tensors='tf')
val_encodings = tokenizer(X_val.tolist(), truncation=True, max_length=max_len, padding="max_length", return_tensors='tf')
test_encodings = tokenizer(X_test.tolist(), truncation=True, max_length=max_len, padding="max_length", return_tensors='tf')
print(train_encodings)

{'input_ids': <tf.Tensor: shape=(7276, 140), dtype=int32, numpy=
array([[  101,  2016,  2011, ...,     0,     0,     0],
       [  101,  1996,  2865, ...,     0,     0,     0],
       [  101,  1037, 17772, ...,     0,     0,     0],
       ...,
       [  101,  1045,  2131, ...,     0,     0,     0],
       [  101,  2065,  2057, ...,     0,     0,     0],
       [  101,  2915,  1998, ...,     0,     0,     0]], dtype=int32)>, 'attention_mask': <tf.Tensor: shape=(7276, 140), dtype=int32, numpy=
array([[1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       ...,
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0]], dtype=int32)>}
```

```
[ ] # Using for Keras-tuner
    df_text_encodings_keras = tokenizer(df.text.tolist(), truncation=True, max_length=max_len, padding="max_length", return_tensors="tf")
    df_text_encodings_keras

{'input_ids': <tf.Tensor: shape=(11370, 140), dtype=int32, numpy=
array([[  101, 15029,  4808, ...,     0,     0,     0],
       [  101, 23764,  2930, ...,     0,     0,     0],
       [  101, 24912,  2923, ...,     0,     0,     0],
       ...,
       [  101,  1045,  2514, ...,     0,     0,     0],
       [  101,  7929,  2040, ...,     0,     0,     0],
       [  101,  5180,  2522, ...,     0,     0,     0]], dtype=int32)>, 'attention_mask': <tf.Tensor: shape=(11370, 140), dtype=int32, numpy=
array([[1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       ...,
```

Figure 22. Encoding Code Snippet and results.

Now , to encode the training data. Figure no 23 shows the code snippet for the encoding the training data.

```
# Encode Training Data
train_encodings_ids = train_encodings['input_ids'].numpy()
train_encodings_attention = train_encodings['attention_mask'].numpy()

val_encodings_ids = val_encodings['input_ids'].numpy()
val_encodings_attention = val_encodings['attention_mask'].numpy()
```

```
train_tf_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings), y_train))
train_tf_dataset = train_tf_dataset.shuffle(len(train_encodings)).batch(BATCH_SIZE)

val_tf_dataset = tf.data.Dataset.from_tensor_slices((dict(val_encodings), y_val))
val_tf_dataset = val_tf_dataset.batch(BATCH_SIZE)
```

```
BERT_DROPOUT = 0.1
BERT_ATT_DROPOUT = 0.1

# Configure DistilBERT's initialization
config = DistilBertConfig(dropout=BERT_DROPOUT,
                          attention_dropout=BERT_ATT_DROPOUT,
                          output_hidden_states=True)
```

Figure 23 code snippet for training data encoding.

Now for the modelling of the distilbert, the figure 24 shows the code block for it.

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

# DistilBERT 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 DistilBERT'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(1, 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
```

```
DistilBERTmodel = TFDistilBertModel.from_pretrained('distilbert-base-uncased', config=config)
model = create_model(DistilBERTmodel)
# Compile the model
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=['accuracy'])

history = model.fit(train_tf_dataset, epochs=5, batch_size=BATCH_SIZE, validation_data=val_tf_dataset,verbose=2)
```

Figure 24. DistilBERT Model

7. Model Evaluation:

7.1 CNN + LSTM with Glove Embedding:

Figure 25 shows the classification report and sensitivity and specificity report for CNN+LSTM model.

```
#classification report
print(classification_report(y_test_new, prediction))
```

```
              precision    recall  f1-score   support

           0       0.83      1.00      0.90      1878
           1       0.00      0.00      0.00       396

    accuracy                           0.83      2274
   macro avg       0.41      0.50      0.45      2274
weighted avg       0.68      0.83      0.75      2274
```

```
[22] from sklearn.metrics import precision_recall_fscore_support
     res = []
     for l in range(2):
         prec,recall,_,_ = precision_recall_fscore_support(y_test_new==l,
                                                           prediction==l,
                                                           pos_label=True,average=None)
         res.append([l,recall[0],recall[1]])

     pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

	class	sensitivity	specificity
0	0	0.0	1.0
1	1	1.0	0.0

Figure 25 . Results for CNN+LSTM

Similarly for Roberta Figure 26 shows the results and code snippet.

```
#classification report
print(classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.89      0.97      0.93      1878
           1       0.75      0.45      0.56       396

    accuracy                           0.88      2274
   macro avg       0.82      0.71      0.75      2274
weighted avg       0.87      0.88      0.87      2274
```

```
[35] from sklearn.metrics import precision_recall_fscore_support
     res = []
     for l in range(2):
         prec,recall,_,_ = precision_recall_fscore_support(y_test==l,
                                                           np.array(y_preds)==l,
                                                           pos_label=True,average=None)
         res.append([l,recall[0],recall[1]])

     pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

	class	sensitivity	specificity
0	0	0.449495	0.968584
1	1	0.968584	0.449495

Figure 26 Results for Roberta.

Now, for our last model, the figure 27 shows the results for FineTuned DistilBERT model.

```
[48] #classification report
print(classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.92      0.94      0.93      1878
           1       0.70      0.60      0.65       396

    accuracy                           0.89      2274
   macro avg       0.81      0.77      0.79      2274
weighted avg       0.88      0.89      0.88      2274
```

```
[49] from sklearn.metrics import precision_recall_fscore_support
res = []
for l in range(2):
    prec,recall,_,_ = precision_recall_fscore_support(y_test==l,
                                                      np.array(y_preds)==l,
                                                      pos_label=True,average=None)
    res.append([l,recall[0],recall[1]])

pd.DataFrame(res,columns = ['class','sensitivity','specificity'])
```

	class	sensitivity	specificity
0	0	0.603535	0.944622
1	1	0.944622	0.603535

Figure 27 Results for FineTuned distilbert.

Conclusion:

This configuration manual covers all the required specifications and the approach step by step to rebuild this research work.