Configuration Manual

MSc Research Project
MSc. In Data Analytics

Jigar Bhatt
Student ID: x18179959

School of Computing
National College of Ireland

Supervisor: Prof. Paul Stynes

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Jigar Sanjay Bhatt
Student ID:	X18179959
Programme:	MSc. In Data Analytics **Year:** 2019-20
Module:	Research in Computing
Supervisor:	Prof. Paul Stynes
Submission Due Date:	17-08-2020
Project Title:	Using hybrid deep learning and word embedding based approach for advance cyberbullying detection
Word Count:	1109 **Page Count:** 10

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: Jigar Sanjay Bhatt

Date: 16-08-2020

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

Jigar Bhatt (x18179959)

1. Introduction

This configuration manual provides detailed instructions on the steps required to replicate the work done in research study and achieved the desired results. The manual includes the minimum system requirements, configuration and the procedure to perform the pre-processing, transformation, training, testing and evaluation.

2. Pre-requisites and configuration

Since the techniques used in this study are processing intensive, normal CPU cannot be able to cope up with the resource and memory required in order to perform the experiments. So entire implementation of the code was performed on an online platform Google Collaboratory. Google Collaboratory is an online resource that provides additional processing capabilities like TPU and GPU in a Jupyter notebooks fashioned environment. Google Colab provides 12 hours of uninterrupted processing availability for implementing data analytics projects. The specifications provided by Google Colab is as follows: -

CPU	GPU	TPU
Intel Xeon Processor with two cores @ 2.30 GHz and 13 GB RAM	Up to Tesla K80 with 12 GB of GDDR5 VRAM, Intel Xeon Processor with two cores @ 2.20 GHz and 13 GB RAM	Cloud TPU with 180 teraflops of computation, Intel Xeon Processor with two cores @ 2.30 GHz and 13 GB RAM

Figure 1: Google Colab Specifications[1]

This project made use of GPU in order to perform the implementation.
Before running the code, click on the **Runtime** option on the menu bar and click on the **Change runtime type**. Change the Hardware Accelerator setting to **GPU.**

The Programming language used for writing the entire code was Python. Python was used throughout the research for data cleaning, processing, transformation and training the models. For coding, inspiration has been taken from (Agrawal and Awekar, 2018) in order to create a similar experimental setup.[2]

3. Datasets and other supporting files

[1] https://www.analyticsvidhya.com/blog/2020/03/google-colab-machine-learning-deep-learning/
[2] https://github.com/sweta20/Detecting-Cyberbullying-Across-SMPs

The datasets were requested from (Agrawal and Awekar, 2018) for carrying out the research study. [3] Two primary datasets were used for the purpose of the study i.e. Wikipedia and Formspring. The datasets consisted of 2 columns in which 1 column consisted of texts and other column consisted of labels annotated by experts as cyberbullying or not cyberbullying.

This study incorporates the use of fastText embedding. FastText embedding have pre-trained vector files in which consists of database of words with its associated vector representations. The vector file can be found and downloaded online on the fast text official website.[4]

4. Uploading and Authenticating the drive for data retrieval

All the files required for the implementation of the code have to be first uploaded on the google drive of the user who is performing the implementation. Inside google drive the user will have to create the same path in his drive as used in the code. For this, the user has to create a folder named 'Colab Notebooks' and upload all the relevant files relating to the study in that folder. For implementing the code, the notebook has to be uploaded on google colab.
For accessing the files from the drive, the authentication needs to be completed.

Mounting the drive to the notebook to import relevant files

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg

Enter your authorization code:
```

The user will have to click on the link that will take him to the drive authorization page where he will have to allow the colab to have access to the drive. The drive will then provide an authorization code that the user has to type in the dialog box below the link.

5. Importing the required libraries and setting up the environment

In this step the prerequisite steps like setting up the tensor flow version, installing and importing the required libraries is done as can be seen in the following screenshots.

Setting up the tensorflow version

```
[2] %tensorflow_version 1.x
```

```
TensorFlow 1.x selected.
```

[3] https://drive.google.com/file/d/11RMLCSIAO3dWk9ejSkVYc5tQwwK5pquG/view
[4] https://dl.fbaipublicfiles.com/fasttext/vectors-english/wiki-news-300d-1M.vec.zip

Installing the required libraries

```
[3] ! pip install tweet-preprocessor
```

```
Collecting tweet-preprocessor
  Downloading https://files.pythonhosted.org/packages/17/9d/71bd016a9edcef8860c607e531f30bd09b13103c7951ae73dd2bf174163c/tweet_preprocessor-0.6.0-py3-none-any.whl
Installing collected packages: tweet-preprocessor
Successfully installed tweet-preprocessor-0.6.0
```

Importing the required libraries

```python
import sys
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import re
from urllib.parse import urlparse
from math import floor, ceil
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
import string
import preprocessor as p
from preprocessor import tokenize
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
from tqdm.notebook import tqdm
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import KFold
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.multiclass import OneVsRestClassifier
from scipy.stats import spearmanr, rankdata
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
import tensorflow as tf
from keras.preprocessing import text, sequence
from keras.models import Model
from keras.layers import Input, LSTM, Bidirectional, Conv1D, Dense, Dropout, GlobalAveragePooling1D, Embedding, Activation, Lambda, BatchNormalization
from keras.optimizers import Adam, RMSprop
from keras.callbacks import Callback
import keras.backend as K
import argparse
import pickle
import string
import numpy as np
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import roc_auc_score
from collections import Counter
import os
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.contrib import learn
from tflearn.data_utils import to_categorical, pad_sequences
from scipy import stats
import tflearn
import json
from sklearn.linear_model import MultiTaskElasticNet
from sklearn.datasets import make_classification
from matplotlib import pyplot
from numpy import where
from sklearn.utils import resample
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
```

6. Setting up the parameters

This step involves in setting the parameters like selecting the train data, type of embedding method, model type and test data out of the options described in the markdown. The user can select any combination of parameters in order to perform the experiment.

Setting up the parameters

1. Choose one of the dataset for training the data.

- formspring dataset - "formspring"
- wikipedia dataset - "wiki"

2. Choose one the embedding type.

- fastText - "fasttext"
- ELMo - "ELMO"
- Stacked(Flair forward + Flair Backward + GloVe) - "stacked"

3. Choose one of the model type.

- BLSTM - "BLSTM"
- CNN - "CNN"
- LSTM - "LSTM"

4. Choose the testing dataset for transfer learning.

- formspring dataset - "formspring"
- wikipedia dataset - "wiki"

```
[5]  train_data = "formspring"
     test_data = "wiki"
     embedding = "stacked"
     model_type = "CNN"
```

7. Importing and basic pre-processing of train data

This section includes importing the datasets into the notebook and applying some basic pre-processing. Steps like checking and dealing with missing values. If the dataset is Wikipedia and embedding method is ELMo or Stacked embedding, the data is sliced in order to avoid out of memory issues in the later stages.

Importing the train dataset

```
[25] if(train_data == "wiki"):
       train_df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/cyberbullying_wiki.csv')
     if(train_data == "formspring"):
       train_df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/data.csv')

     #Checking the missing values in data
     missing_values = train_df.isna().sum()
     print("Missing values in data", missing_values)

     #Dropping the rows with missing values
     nan_value = float("NaN")
     train_df.replace("", nan_value, inplace=True)
     train_df.dropna(subset = ["col1"], inplace=True)
     missing_values = train_df.isna().sum()
     print("Missing values after dropping the empty rows", missing_values)

     #The wikipedia dataset is very large with more than 100000 rows. In order to avoid Out of memory issues, the data has been sliced.
     if(train_data == "wiki" and (embedding == "stacked" or embedding== "ELMO" )):
       train_df = train_df[:6000]
       print(train_df.col2.value_counts())
```

```
Missing values in data col1     13
col2     0
dtype: int64
Missing values after dropping the empty rows col1     0
col2     0
```

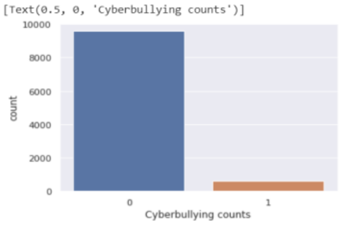
Performing the train-test split

```
[8]  X_train, X_test, Y_train, Y_test = train_test_split(train_df['col1'], train_df['col2'], random_state=42, test_size=0.20) #Train-Test Split
     train_df = pd.DataFrame()
     train_df['text'] = X_train
     train_df['label'] = Y_train
```

Plotting the class distribution

```
[9] import seaborn as sns
    sns.set(style="darkgrid")
    ax = sns.countplot(x="label", data=train_df)
    ax.set(xlabel="Cyberbullying counts")
```

```
[Text(0.5, 0, 'Cyberbullying counts')]
```



Upsampling the data to balance the data

```
[10] print("Distribution of class before upsampling:\n",train_df.label.value_counts())

     # Separate majority and minority classes
     df_majority = train_df[train_df.label==0]
     df_minority = train_df[train_df.label==1]
     # Upsample minority class
     df_minority_upsampled = resample(df_minority,
                                      replace=True,     # sample with replacement
                                      n_samples=len(df_majority),    # to match majority class
                                      random_state=123) # reproducible results

     # Combine majority class with upsampled minority class
     train_df = pd.concat([df_majority, df_minority_upsampled])
     # Display new class counts
     print("Distribution of data after upsampling: \n",train_df.label.value_counts())
     ax = sns.countplot(x="label", data=train_df)
     ax.set(xlabel="Cyberbullying counts")
```

```
Distribution of class before upsampling:
 0    9585
1     633
Name: label, dtype: int64
Distribution of data after upsampling:
 1    9585
0    9585
Name: label, dtype: int64
[Text(0.5, 0, 'Cyberbullying counts')]
```

Setting the data type of data for compatibility

```
[11] labels = train_df['label']
     labels = labels.astype('category')
     train_df['text']=train_df['text'].astype('str')
     x_text=list(train_df['text'])
```

8. Processing Word Embeddings: ELMo and Flair

This section includes initializing of word embedding of ELMo and Flair embeddings.

Initializing of ELMO and Flair embeddings

```
[12] if(embedding == "ELMO" or embedding == "stacked"):
       import torch
       !pip install flair
       import flair
       !pip install allennlp==0.9.0
       from flair.models import TextClassifier
       from flair.data import Sentence
       ## Importing the Embeddings ##
       from flair.embeddings import WordEmbeddings
       from flair.embeddings import CharacterEmbeddings
       from flair.embeddings import StackedEmbeddings
       from flair.embeddings import FlairEmbeddings
       from flair.embeddings import ELMoEmbeddings
       from flair.embeddings import FlairEmbeddings
     ### Initialising embeddings (un-comment to use others) ###
       glove_embedding = WordEmbeddings('glove')
     #character_embeddings = CharacterEmbeddings()
       flair_forward  = FlairEmbeddings('news-forward-fast')
       flair_backward = FlairEmbeddings('news-backward-fast')
       elmo_embedding = ELMoEmbeddings()
       if(embedding == "ELMO"):
         stacked_embeddings = StackedEmbeddings( embeddings = [elmo_embedding])
       if(embedding == "stacked"):
         stacked_embeddings = StackedEmbeddings( embeddings = [flair_forward,flair_backward,glove_embedding])
```

Basic preprocessing and transformation of data for Flair stacked and ELMO embedding

```
[13] if(embedding == "stacked" or embedding == "ELMO"):

     # create a sentence #
       CUDA_LAUNCH_BLOCKING=1
       sentence = Sentence('This code is used for initializing of embedding')
     # embed words in sentence #
       stacked_embeddings.embed(sentence)
       for token in sentence:
         print(token.embedding)
     # data type and size of embedding #
       print(type(token.embedding))
     # storing size (length) #
       z = token.embedding.size()[0]
       print(z)
```

```
[13]   from tqdm import tqdm ## tracks progress of loop ##
     # creating a tensor for storing sentence embeddings #
       s = torch.zeros(0,z)
       if torch.cuda.is_available():
         s = s.cuda()
     # iterating Sentence (tqdm tracks progress) #
       for tweet in tqdm(x_text):
       # empty tensor for words #
         w = torch.zeros(0,z)
         if torch.cuda.is_available():
           w = w.cuda()
         sentence = Sentence(tweet)
         #sentence = sentence[:10]
         #print(sentence)

         stacked_embeddings.embed(sentence)
         for token in sentence:
         # storing word Embeddings of each word in a sentence #
           w = torch.cat((w,token.embedding.view(-1,z)),0)
       # storing sentence Embeddings (mean of embeddings of all words)   #
         s = torch.cat((s, w.mean(dim = 0).view(-1, z)),0)
```

9. Tokenizing and mapping the word embeddings

This section performs the tokenizing and mapping of the embeddings

```
    #Initializing the matrix by creating an empty matrix of 0's.
    embedding_matrix = np.zeros((vocab, emb_dim))
    #Mapping of word embeddings
    if(emb_type == "stacked" or emb_type == "ELMO"):

        for word,i in word_index.items():
          try:
            word_sent = Sentence(word)
            stacked_embeddings.embed(word_sent)
            embedding_vector = word_sent[0].embedding.cpu().detach().numpy()
            embedding_matrix[i] = embedding_vector
          except IndexError:
            embedding_matrix[i] = np.random.normal(0,np.sqrt(0.25),emb_dim)
        return embedding_matrix


    elif(emb_type == "fasttext"):
      embeddings_index = {}
      #Loading the pretrained word vector file of ELMo.
      f = open('/content/drive/My Drive/Colab Notebooks/wiki-news-300d-1M.vec')
      for line in f:
          values = line.split()
          word = values[0]
          coefs = np.asarray(values[1:], dtype='float32')
          embeddings_index[word] = coefs
      f.close()
      print('Found %s word vectors.' % len(embeddings_index))
      #Mapping the word embeddings from the vector file
      k=0
      for word, i in word_index.items():
          embedding_vector = embeddings_index.get(word)
          if embedding_vector is not None:
          # we found the word - add that words vector to the matrix
              embedding_matrix[i] = embedding_vector
          else:
          # doesn't exist, assign a random vector
              k=k+1
              embedding_matrix[i] = np.random.randn(emb_dim)
      return embedding_matrix

  embedding_weight = map_embeddings(embedding)
  print("Embedding matrix after tokenization and mapping:", embedding_weight)
```

```
Embedding matrix after tokenization and mapping: [[ 0.00000000e+00  0.00000000e+00  0.00000000e+00 ...  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [ 2.40953907e-01 -1.04704559e-05  1.27468466e-05 ... -4.11790013e-01
   4.05389994e-01  7.85040021e-01]
 [ 3.71650071e-03 -9.14466455e-06  1.02699604e-02 ... -3.76159996e-01
  -3.25019993e-02  8.06200027e-01]
 ...
 [-4.05321596e-03 -1.18993332e-04  1.41220279e-02 ...  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [-2.61018774e-03 -2.80036929e-05 -2.75324658e-03 ...  0.00000000e+00
```

10. Preparing the same test data for evaluation

The following section aims at processing the same domain test data and also importing and processing the data for transfer learning

Basic pre-processing of the test data obtained from the train-test split

```
[ ] #Same domain testing data preparation
    X_test = X_test.astype('str')
    temp=list(X_test)
    len(temp)
    tok = text.Tokenizer()
    tok.fit_on_texts(temp)
    test_seq = tok.texts_to_sequences(temp)
    test_x1 = sequence.pad_sequences(test_seq, 300)
    test_y1 = Y_test
```

Loading the test data for transfer learning and applying some basic pre-processing required for testing

```
[ ] def load_testdata(test_data):
    #One test data
      if(test_data == "wiki"):
        test_df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/cyberbullying_wiki.csv')
      if(test_data == "formspring"):
        test_df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/data.csv')
      test_df.shape
      test_df['col1']=test_df['col1'].astype('str')
      temp=list(test_df['col1'])
      len(temp)
      tok = text.Tokenizer()
      tok.fit_on_texts(temp)
      test_seq = tok.texts_to_sequences(temp)
      test_x = sequence.pad_sequences(test_seq, 300)
      test_y = test_df['col2']
      return test_x,test_y

    test_x,test_y = load_testdata(test_data)
```

11. Defining the evaluation function

Evaluation function

```
[ ] def evaluate_model(model, testX, testY):
        temp = model.predict(testX)
        y_pred  = np.argmax(temp, 1)
        y_true = testY
        precision = precision_score(y_true, y_pred, average='weighted')
        recall = recall_score(y_true, y_pred, average='weighted')
        F1_score = f1_score(y_true, y_pred, average='weighted')
        print("Precision: " + str(precision) + "\n")
        print("Recall: " + str(recall) + "\n")
        print("f1 score: " + str(F1_score) + "\n")
        return precision, recall, f1_score
```

12. Model training and testing

This step involves defining various DNN models and configuration of layers.

CNN

```
def model_training(model_type,emb_dim):

  Y_train_dm = pd.get_dummies(labels)
  units = 2

  if(model_type == "CNN"):
    #Defining and configuring the layers
    model = Sequential()
    embedding_layer = Embedding(vocab, emb_dim, weights=[embedding_weight], input_length=300, trainable=False)
    model.add(embedding_layer)
    model.add(Conv1D(128, 5, activation='relu'))
    model.add(GlobalMaxPooling1D())
    model.add(Dropout(0.25))
    model.add(Dense(units, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
    print(model.summary())
    history = model.fit(X_ques, Y_train_dm, batch_size=128, epochs=10, verbose=1, validation_split=0.2)
    print('\033[1m'+'\nResults on Evaluation of same dataset:-\n'+'\033[0m ')
    evaluate_model(model,test_x1, test_y1)
    print('\033[1m'+'Results on Evaluation of transfer learning:- \n '+'\033[0m')
    evaluate_model(model,test_x, test_y)
```

BLSTM

```
if(model_type == "BLSTM"):
  model = Sequential()
  model.add(Embedding(vocab,
                emb_dim,
                embeddings_initializer=Constant(embedding_weight),
                input_length=300,
                trainable=True))

  model.add(SpatialDropout1D(0.2))
  model.add(Bidirectional(CuDNNLSTM(64, return_sequences=True)))
  model.add(Bidirectional(CuDNNLSTM(32)))
  model.add(Dropout(0.25))
  model.add(Dense(units, activation='sigmoid'))
  model.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics = ['accuracy'])
  print(model.summary())
  history = model.fit(X_ques, Y_train_dm, epochs=10, batch_size=128,verbose = 1,validation_split = 0.2)
  print('\033[1m'+'\nResults on Evaluation of same dataset:-\n'+'\033[0m ')
  evaluate_model(model,test_x1, test_y1)
  print('\033[1m'+'Results on Evaluation of transfer learning:-\n '+'\033[0m')
  evaluate_model(model,test_x, test_y)
```

LSTM

```
if(model_type == "LSTM"):
  model = Sequential()
  model.add(Embedding(vocab, emb_dim,weights = [embedding_weight], input_length=300, trainable=True))
  model.add(Dropout(0.25))
  model.add(LSTM(300))
  model.add(Dropout(0.50))
  model.add(Dense(units, activation='sigmoid'))
  model.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])
  print(model.summary())
  history = model.fit(X_ques, Y_train_dm, batch_size=128, epochs=10, verbose=1, validation_split=0.2)
  print('\033[1m'+'\nResults on Evaluation of same dataset:-\n'+'\033[0m ')
  evaluate_model(model,test_x1, test_y1)
  print('\033[1m'+'Results on Evaluation of transfer learning:-\n '+'\033[0m ')
  evaluate_model(model,test_x, test_y)
return history
```

The following image shows the selection of the embedding dimensions as per the embedding method used. The functions for training is passed to the model training function and finally the parameters for model loss graph is set.

```
#Setting the embedding dimensions as per the type of word embedding method used.
if(embedding == "fasttext"):
  emb_dim = 300
if(embedding == "ELMO"):
  emb_dim = 3072
if(embedding == "stacked"):
  emb_dim = 2148
print(emb_dim)
print(model_type)
#Finally training the model
history = model_training(model_type,emb_dim)

#Plotting the model training and validation loss.
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

Output of the model training and testing

```
2148
CNN
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resou
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow_core/python/ops/nn_impl.py:183: where (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a fu
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 300, 2148)         35371116
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 296, 128)          1374848
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 258
=================================================================
Total params: 36,746,222
Trainable params: 1,375,106
Non-trainable params: 35,371,116
```

```
Train on 15336 samples, validate on 3834 samples
Epoch 1/10
15336/15336 [==============================] - 15s 993us/step - loss: 0.3744 - acc: 0.8355 - val_loss: 0.2366 - val_acc: 0.9429
Epoch 2/10
15336/15336 [==============================] - 7s 467us/step - loss: 0.1248 - acc: 0.9682 - val_loss: 0.1245 - val_acc: 0.9853
Epoch 3/10
15336/15336 [==============================] - 7s 465us/step - loss: 0.0537 - acc: 0.9903 - val_loss: 0.0288 - val_acc: 1.0000
Epoch 4/10
15336/15336 [==============================] - 7s 468us/step - loss: 0.0301 - acc: 0.9957 - val_loss: 0.0111 - val_acc: 1.0000
Epoch 5/10
15336/15336 [==============================] - 7s 469us/step - loss: 0.0162 - acc: 0.9981 - val_loss: 0.0069 - val_acc: 1.0000
Epoch 6/10
15336/15336 [==============================] - 7s 469us/step - loss: 0.0116 - acc: 0.9987 - val_loss: 0.0046 - val_acc: 1.0000
Epoch 7/10
15336/15336 [==============================] - 7s 470us/step - loss: 0.0092 - acc: 0.9992 - val_loss: 0.0031 - val_acc: 1.0000
Epoch 8/10
15336/15336 [==============================] - 7s 470us/step - loss: 0.0075 - acc: 0.9993 - val_loss: 0.0017 - val_acc: 1.0000
Epoch 9/10
15336/15336 [==============================] - 7s 470us/step - loss: 0.0070 - acc: 0.9988 - val_loss: 0.0028 - val_acc: 1.0000
Epoch 10/10
15336/15336 [==============================] - 7s 470us/step - loss: 0.0072 - acc: 0.9993 - val_loss: 0.0040 - val_acc: 1.0000
```

Results on Evaluation of same dataset:-

Precision: 0.8956899388718396

Recall: 0.8998043052837573

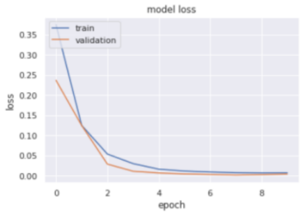
f1 score: 0.8977345512524254

Results on Evaluation of transfer learning:-

Precision: 0.7844569152142045

Recall: 0.7884847752537457

f1 score: 0.7864600661161946



References

Agrawal, S. and Awekar, A. (2018) ‘Deep learning for detecting cyberbullying across multiple social media platforms’, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10772 LNCS(Table 2), pp. 141–153. doi: 10.1007/978-3-319-76941-7_11.