Import here all the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import plotly.express as px
import random
from keras.utils import np_utils
from scipy.stats import multivariate_normal as mvn
from keras.callbacks import TensorBoard

import string
import re #regular expressions
import nltk
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
from __future__ import absolute_import, division, print_function, unicode_literals
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('punkt')

import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_datasets as tfds
from tensorflow import keras
from tensorflow.keras import regularizers

#Import .py file of general algorithms
from google.colab import files
#files.upload()
from general import accuracy, R2, OLS
from general import confusionMatrix
from general import SimpleLogisticRegression, ANN
from general import derivative, relu, linear, sigmoid, softmax, bin_cross_entropy, one_hot
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Unzipping corpora/omw-1.4.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
#Mount Google Drive Folders
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Load dataset

```
train = pd.read_csv('/content/drive/MyDrive/Datos/train.csv')
test = pd.read_csv('/content/drive/MyDrive/Datos/test.csv')
dev = pd.read_csv('/content/drive/MyDrive/Datos/dev.csv')
train.head()
```

	Utterance	Dialogue_Act	Emotion	Dialogue_ID
0	say , jim , how about going for a few beers af...	directive	no emotion	1
1	you know that is tempting but is really not go...	commissive	no emotion	1
2	what do you mean ? it will help us to relax .	question	no emotion	1
3	do you really think so ? i don't . it will jus...	question	no emotion	1
4	i guess you are right.but what shall we do ? i...	question	no emotion	1

```
def trans_y(x):
  if x == 'directive' or x == 'commissive':
    return 0
  elif x == 'inform':
    return 1
  elif x == 'question':
    return 2

train.Dialogue_Act = train. Dialogue_Act.apply(trans_y)
test.Dialogue_Act = test.   Dialogue_Act.apply(trans_y)
dev.Dialogue_Act = dev. Dialogue_Act.apply(trans_y)
train.head()
```

	Utterance	Dialogue_Act	Emotion	Dialogue_ID
0	say , jim , how about going for a few beers af...	0	no emotion	1
1	you know that is tempting but is really not go...	0	no emotion	1
2	what do you mean ? it will help us to relax .	2	no emotion	1

```
train.drop(['Emotion','Dialogue_ID'], inplace = True, axis = 1)
test.drop(['Emotion','Dialogue_ID'], inplace = True, axis = 1)
dev.drop(['Emotion','Dialogue_ID'], inplace = True, axis = 1)
train.head()
```

	Utterance	Dialogue_Act
0	say , jim , how about going for a few beers af...	0
1	you know that is tempting but is really not go...	0
2	what do you mean ? it will help us to relax .	2
3	do you really think so ? i don't . it will jus...	2
4	i guess you are right.but what shall we do ? i...	2

```
#Check the size of the data
print(f"Size of dataset: {train.shape}")

#Check and delete for duplicates
print(f"Number of duplicate rows: {train[train.duplicated()].shape}\n")
train.drop_duplicates(subset=None, keep="first", inplace=True)
print(f"Final size of dataset: {train.shape}")

train = train.reset_index()
```

```
Size of dataset: (87170, 2)
Number of duplicate rows: (14779, 2)

Final size of dataset: (72391, 2)
```

Data Preprocessing

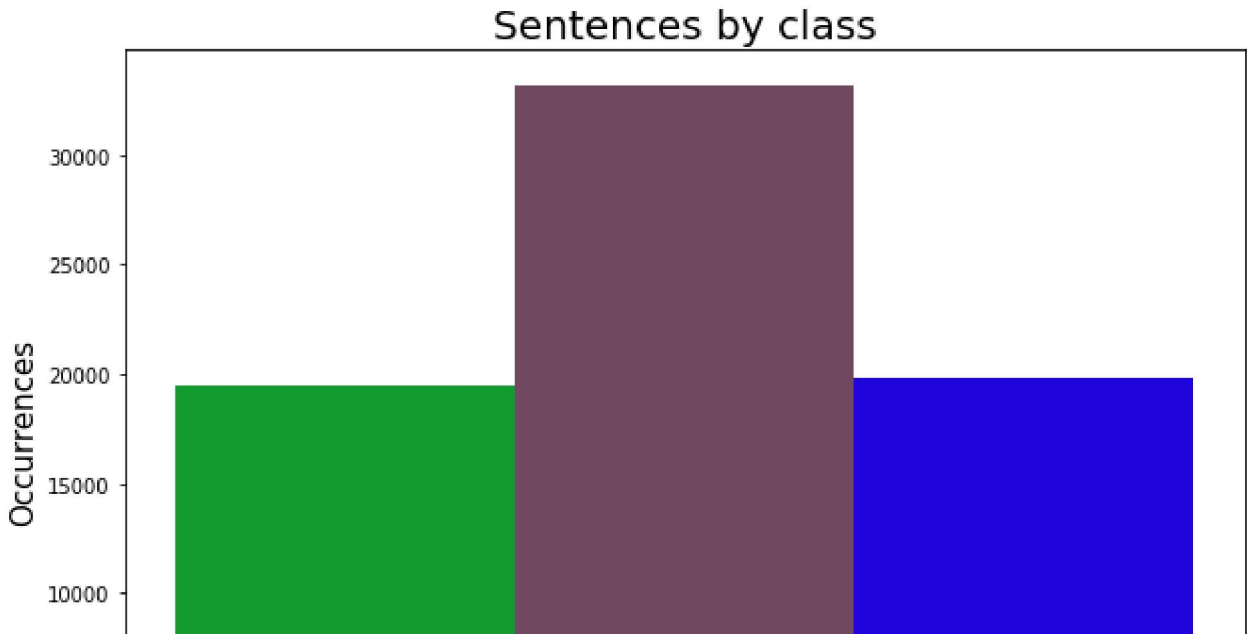
```
def preprocess(sentence,lemma=True):
  #Remove url links
  proc_sent = re.sub(r'https?:\/\/.*[\r\n]*','',sentence)

  #Delete non-ASCII values
  proc_sent = str(proc_sent.encode("ascii","ignore"))[1:]

  #Remove punctuation
```

```
    proc_sent = ''.join([char for char in proc_sent if char not in string.punctuation])

    #Parse to lower case
    proc_sent = proc_sent.lower()

    #Tokenize and remove stop words
    stop_words = stopwords.words('english')
    proc_sent = word_tokenize(proc_sent)
    #proc_sent = [word for word in proc_sent if word not in stop_words]

    #Remove single letters
    proc_sent = [word for word in proc_sent if len(word) != 1]

    #Stem or lemmatize (just 1)
    if lemma:
      lemmatizer = WordNetLemmatizer()
      proc_sent = [lemmatizer.lemmatize(word) for word in proc_sent]
    else:
      porter = PorterStemmer()
      proc_sent = [porter.stem(word) for word in proc_sent] #Stemming

    return proc_sent


train.Utterance = train.Utterance.apply(preprocess)
test.Utterance = test.Utterance.apply(preprocess)
dev.Utterance = dev.Utterance.apply(preprocess)
print(f"{len(train)} observations...")
```

```
     72391 observations...
```

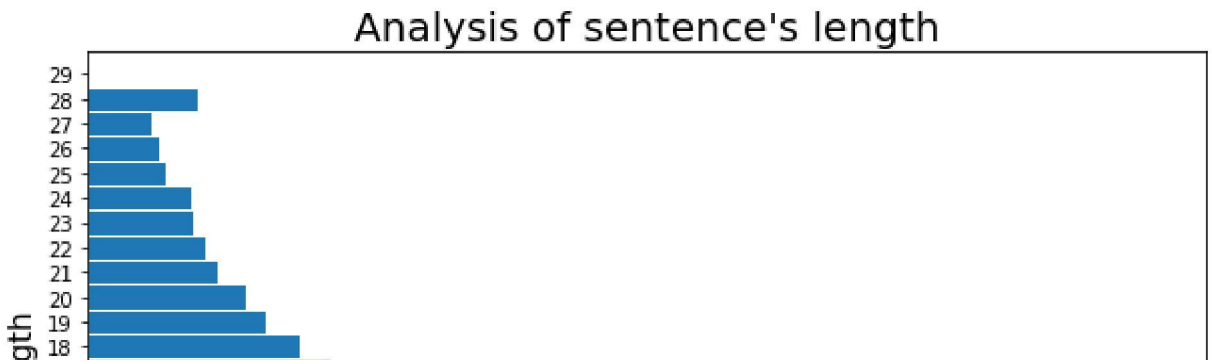
```
plt.figure()
fig, ax = plt.subplots(figsize=(10,7))
N, bins, patches = plt.hist(train['Dialogue_Act'], bins=3)
ax.set_xticks(np.arange(3))
ax.set_xticklabels(['Talk:0','Inform:1','Question:2'], size=13)
for i in range(len(N)):
  patches[i].set_facecolor("#" + ''.join(random.choices("ABCDEF" + string.digits, k=6)))

plt.ylabel("Occurrences", size=15)
plt.xlabel("Sentence class", size=15)
plt.title("Sentences by class", size=20)
```

```
Text(0.5, 1.0, 'Sentences by class')
<Figure size 432x288 with 0 Axes>
```



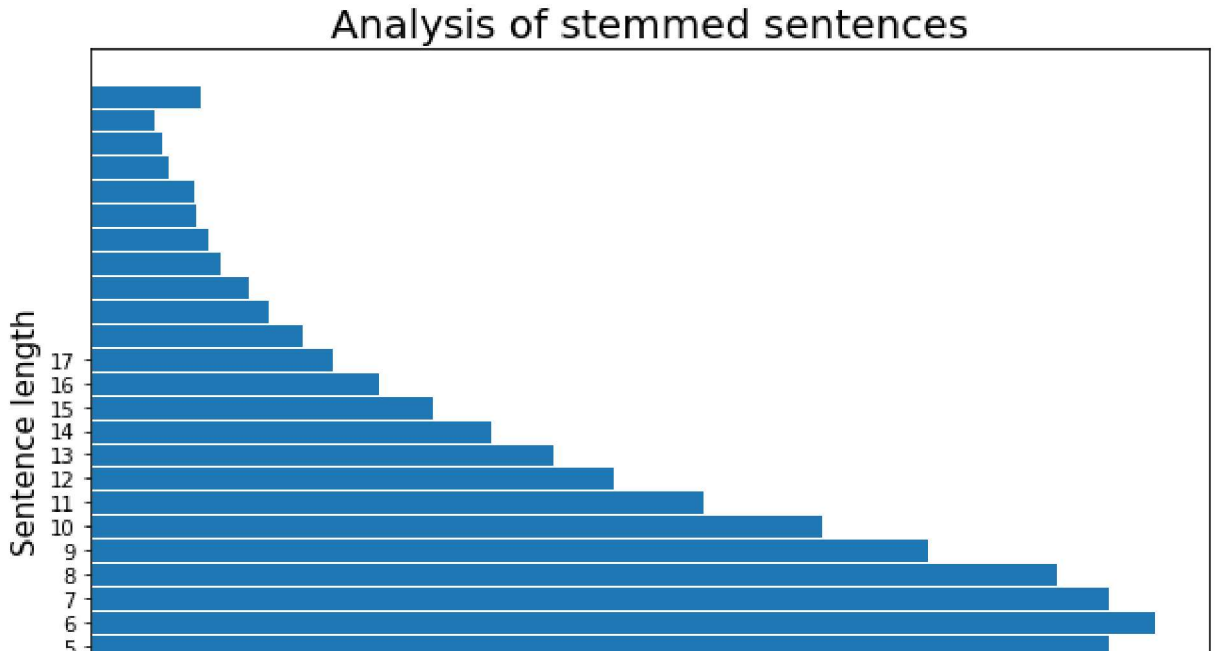

```
lengths = [len(i) for i in train['Utterance']]
uniques = np.unique(lengths)
bins = [i for i in range(1,30)]

plt.figure()
fig, ax = plt.subplots(figsize=(10,7))
N, bins, patches = plt.hist(lengths, bins=bins, orientation='horizontal', edgecolor='white',
ax.set_yticks(np.arange(30))
ax.set_yticklabels(uniques, size=10)
#for i in range(len(N)):
#  patches[i].set_facecolor("#" + ''.join(random.choices("ABCDEF" + string.digits, k=6)))
#patches.set_facecolor(, k = 10)

plt.ylabel("Sentence length", size=15)
plt.xlabel("Number of sentences", size=15)
plt.title("Analysis of sentence's length", size=20)
```

```
Text(0.5, 1.0, "Analysis of sentence's length")
<Figure size 432x288 with 0 Axes>
```




```
#Remove short sentences
#size = int(np.mean(np.arange(1,25)))

#for i in range(len(train['Utterance'])):
  #Save as is if the size is between the margins
  #Clip the long sentences to make them shorter
#  if len(train['Utterance'][i]) > 25:
#    train['Utterance'][i] = train['Utterance'][i][:size]
#  elif len(train['Utterance'][i]) < 1:
#    train.drop([i], axis=0, inplace=True)

#train = train.reset_index()
#print(f"{len(train)} observations...")
#print(f"Mean size for clipping sentences: {size}")


lengths = [len(i) for i in train['Utterance']]
uniques = np.unique(lengths)
print(uniques)
bins = [i for i in range(30)]

plt.figure()
fig, ax = plt.subplots(figsize=(10,7))
N, bins, patches = plt.hist(lengths, bins=bins, orientation='horizontal', edgecolor='white',
ax.set_yticks(np.arange(18))
#for i in range(len(N)):
#  patches[i].set_facecolor("#" + ''.join(random.choices("ABCDEF" + string.digits, k=6)))

plt.ylabel("Sentence length", size=15)
plt.xlabel("Number of sentences", size=15)
plt.title("Analysis of stemmed sentences", size=20)
```

```
[  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  83  84  85  86  87  88  89  90
  91  92  94  95  97  98  99 100 109 110 111 114 116 119 121 123 129 146
 199 201 239]
Text(0.5, 1.0, 'Analysis of stemmed sentences')
<Figure size 432x288 with 0 Axes>
```




```
def pd_to_tensor_data (data):
  new_tensor_data = tf.data.Dataset.from_tensor_slices(
    (
        tf.cast(data["Utterance"].values, tf.string),
        tf.cast(data["Dialogue_Act"].values, tf.int64)
        )
  )
  return new_tensor_data


def join_text (text):
  return " ".join(text)


train['Utterance'] = train['Utterance'].apply(join_text)
test['Utterance'] = test['Utterance'].apply(join_text)
dev['Utterance'] = dev['Utterance'].apply(join_text)
train.drop(['index'], inplace=True, axis=1)
train
```

	Utterance	Dialogue_Act
0	say jim how about going for few beer after dinner	0
1	you know that is tempting but is really not go...	0
2	what do you mean it will help u to relax	2
3	do you really think so dont it will just make ...	2
4	guess you are rightbut what shall we do dont f...	2
...	...	...
72386	want pair of locus	0
72387	take look at the one on display please	0
72388	need size 41	0
72390	okay ill just be minute	0

Make the training, validation, and test sets

```
print(f"Training set: {train.shape}")
print(f"Validation set: {dev.shape}")
print(f"Test set: {test.shape}")
```

```
Training set: (72391, 2)
Validation set: (8069, 2)
Test set: (7740, 2)
```

```
tensor_train = pd_to_tensor_data(train)
tensor_validation = pd_to_tensor_data(dev)
tensor_test = pd_to_tensor_data(test)
```

```
tensor_train
```

```
<TensorSliceDataset element_spec=(TensorSpec(shape=(), dtype=tf.string, name=None), Tens
```

Model Creation

Create the embeddings

```
#embedding = "https://tfhub.dev/google/tf2-preview/nnlm-en-dim128-with-normalization/1" #El m
#embedding = "https://tfhub.dev/google/universal-sentence-encoder/4"
#embedding = "https://tfhub.dev/google/tf2-preview/nnlm-en-dim50-with-normalization/1"
embedding = "https://tfhub.dev/google/tf2-preview/nnlm-en-dim50/1"
#embedding = 'https://tfhub.dev/google/nnlm-en-dim50/2'
#embedding = "https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim/1"
```

```
#embedding = "https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim-with-oov/1"
hub_layer = hub.KerasLayer(embedding,
                           input_shape=[],
                           dtype=tf.string,
                           trainable=True)


model = keras.Sequential()
model.add(hub_layer)  #Embedding layer
model.add(keras.layers.Dense(4, activation='relu',  kernel_regularizer = regularizers.L1(0.00
#model.add(keras.layers.Dense(3, activation='relu',  kernel_regularizer = regularizers.L1(0.0
model.add(keras.layers.Dropout(.3))
model.add(keras.layers.Dense(3, activation='softmax'))
model.summary()
```

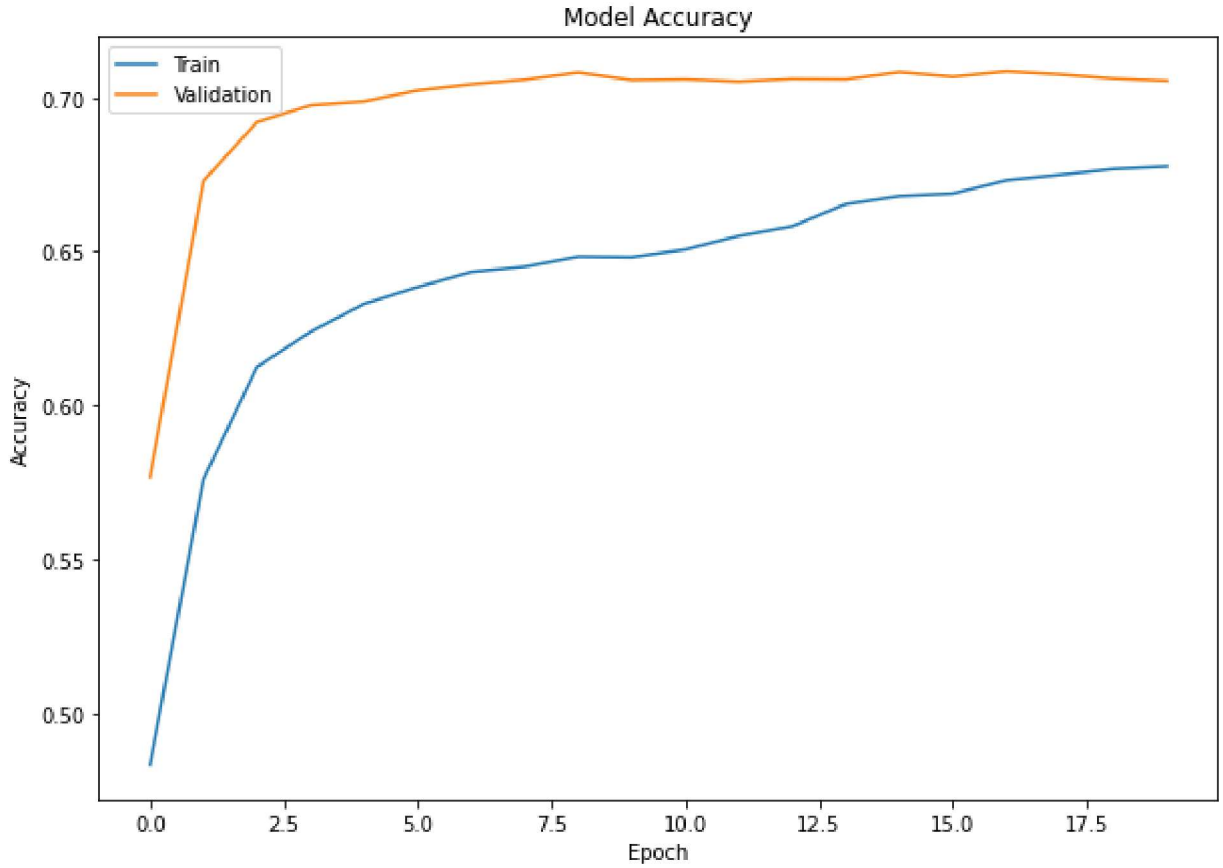
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 50)                48190600

 dense (Dense)               (None, 4)                 204

 dropout (Dropout)           (None, 4)                 0

 dense_1 (Dense)             (None, 3)                 15

=================================================================
Total params: 48,190,819
Trainable params: 48,190,819
Non-trainable params: 0
_________________________________________________________________
```

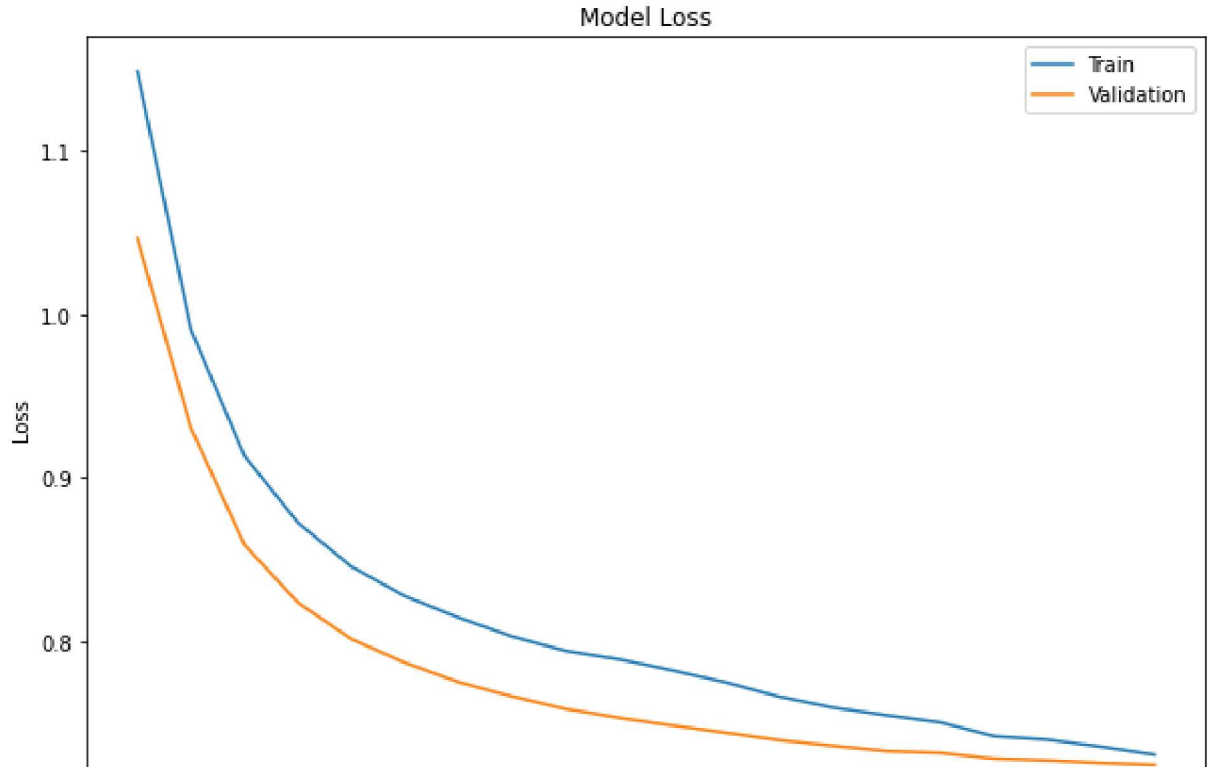
```
model.compile(optimizer = 'Adam',
              loss = keras.losses.SparseCategoricalCrossentropy(),
              metrics = ['accuracy'])


history = model.fit(tensor_train.batch(512),
                    epochs = 20,
                    validation_data = tensor_validation.batch(512))
#                    verbose = 2)
```

```
Epoch 1/20
142/142 [==============================] - 81s 562ms/step - loss: 1.1482 - accuracy: 0.4
Epoch 2/20
142/142 [==============================] - 77s 544ms/step - loss: 0.9905 - accuracy: 0.5
Epoch 3/20
142/142 [==============================] - 77s 540ms/step - loss: 0.9134 - accuracy: 0.6
Epoch 4/20
142/142 [==============================] - 78s 546ms/step - loss: 0.8727 - accuracy: 0.6
Epoch 5/20
142/142 [==============================] - 76s 537ms/step - loss: 0.8459 - accuracy: 0.6
```

```
Epoch 6/20
142/142 [==============================] - 76s 534ms/step - loss: 0.8278 - accuracy: 0.6
Epoch 7/20
142/142 [==============================] - 77s 541ms/step - loss: 0.8148 - accuracy: 0.6
Epoch 8/20
142/142 [==============================] - 76s 534ms/step - loss: 0.8034 - accuracy: 0.6
Epoch 9/20
142/142 [==============================] - 76s 534ms/step - loss: 0.7945 - accuracy: 0.6
Epoch 10/20
142/142 [==============================] - 77s 539ms/step - loss: 0.7894 - accuracy: 0.6
Epoch 11/20
142/142 [==============================] - 76s 533ms/step - loss: 0.7826 - accuracy: 0.6
Epoch 12/20
142/142 [==============================] - 76s 538ms/step - loss: 0.7752 - accuracy: 0.6
Epoch 13/20
142/142 [==============================] - 77s 543ms/step - loss: 0.7665 - accuracy: 0.6
Epoch 14/20
142/142 [==============================] - 76s 534ms/step - loss: 0.7599 - accuracy: 0.6
Epoch 15/20
142/142 [==============================] - 76s 534ms/step - loss: 0.7552 - accuracy: 0.6
Epoch 16/20
142/142 [==============================] - 77s 541ms/step - loss: 0.7509 - accuracy: 0.6
Epoch 17/20
142/142 [==============================] - 76s 535ms/step - loss: 0.7425 - accuracy: 0.6
Epoch 18/20
142/142 [==============================] - 76s 536ms/step - loss: 0.7404 - accuracy: 0.6
Epoch 19/20
142/142 [==============================] - 76s 534ms/step - loss: 0.7360 - accuracy: 0.6
Epoch 20/20
142/142 [==============================] - 77s 543ms/step - loss: 0.7313 - accuracy: 0.6
```

```
plt.figure(figsize=(10,7))
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

plt.figure(figsize=(10,7))
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 right')
plt.show()
```





https://huggingface.co/datasets/silicone

https://huggingface.co/datasets/silicone/tree/main/dummy/dyda_da/1.0.0

```
yhat = model.predict(np.array(test.Utterance))
```

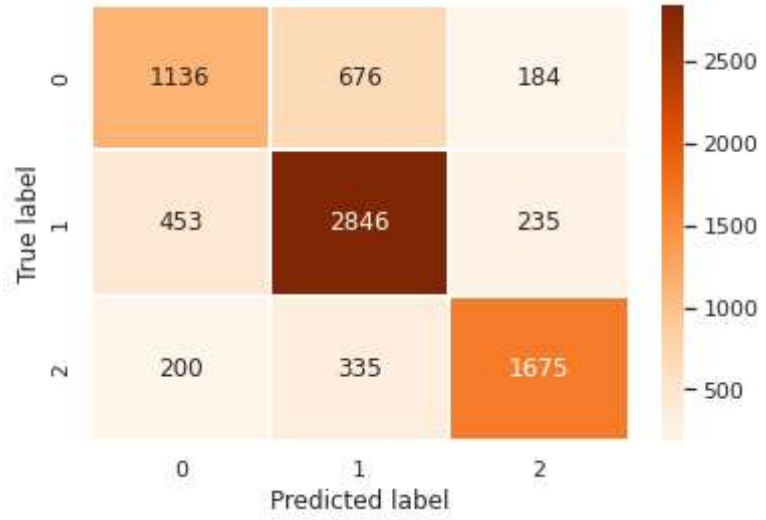
```
y2hat = np.argmax(yhat, axis = 1)
```

```
accuracy(test.Dialogue_Act,y2hat)
```

```
0.7308785529715762
```

```
cm = pd.crosstab(test.Dialogue_Act,y2hat)
sns.set(font_scale = 1)
ax = sns.heatmap(cm, annot=True, cmap='Oranges',  fmt='d', linewidths=1)
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 12.5, 'Predicted label')
```




```
from keras.models import load_model

model.save('my_model.h5')  # creates a HDF5 file 'my_model.h5'
```

```
model.get_weights()
```

```
[array([[-0.07078403,  0.2309921 , -0.04001201, ...,  0.006042  ,
          0.01584901, -0.23308909],
        [ 0.13103518,  0.17923635,  0.01063039, ..., -0.29381433,
          0.05539819,  0.05226879],
        [ 0.06475412, -0.12864912, -0.12485643, ..., -0.16776969,
          0.07435564,  0.19124678],
        ...,
        [ 0.4045343 , -0.17706504, -0.13003995, ..., -0.01742322,
          0.02578888,  0.29181632],
        [ 0.4186347 ,  0.0406054 , -0.14481059, ..., -0.05217805,
          0.03594268,  0.21899657],
        [ 0.53632677, -0.06234785, -0.07169826, ...,  0.11806542,
         -0.06521951,  0.25392768]], dtype=float32),
 array([[-8.64409842e-03, -1.35629415e-03,  2.39973539e-04,
          2.63674883e-06],
        [ 2.65687108e-01, -4.65952908e-05, -1.28384598e-03,
```

```
 -2.98594922e-01],
[-1.01601596e-04,  1.22161724e-01,  3.96659103e-04,
  1.58998650e-04],
[ 1.07205147e-03,  2.53814667e-01, -7.08013475e-02,
 -4.48130444e-03],
[-7.45285128e-04, -1.31420512e-02, -2.35818356e-04,
  4.29416163e-04],
[ 2.63953465e-04,  3.61502439e-01, -2.59811163e-01,
 -1.79079501e-03],
[ 1.25935383e-03,  3.97002965e-01, -8.10843054e-03,
 -2.80113192e-03],
[ 3.76312318e-03,  1.77863310e-03, -6.92195259e-04,
 -2.07327053e-01],
[-1.90509149e-04,  4.32586968e-02,  4.08421940e-04,
  2.19984664e-04],
[-4.57304064e-03, -1.24944961e-02, -6.64635887e-03,
 -1.06430112e-03],
[-1.99860008e-03, -1.11869024e-03,  1.30933162e-03,
  6.41831756e-02],
[-3.17298291e-05,  7.32671190e-03, -4.19729888e-01,
  5.61613182e-04],
[-3.11533618e-03, -7.33417459e-04,  8.27413984e-04,
  5.23001134e-01],
[-8.48840224e-04,  8.39261338e-04,  6.71804941e-04,
 -8.48954893e-04],
[ 7.29887834e-05, -8.10532924e-03, -1.57629675e-03,
  2.54075305e-04],
[ 1.50340365e-03,  1.24114104e-01, -7.96011009e-04,
 -9.90449917e-04],
[ 3.59227648e-03, -2.44457857e-03,  8.69443349e-04,
 -7.36772432e-04],
[ 2.08630256e-04,  3.33838922e-04,  2.88001681e-03,
  3.56190867e-05],
[ 2.35030195e-04,  5.32804697e-04,  3.48347763e-04,
 -7.90880949e-05],
[ 8.65432620e-03, -3.30979261e-03,  1.50300984e-04,
 -3.33289981e-01],
[ 1.92239985e-03,  9.92250396e-04, -2.16122111e-03,
 -3.36863875e-01],
[-6.95219496e-04, -5.30599849e-04,  1.64965272e-01,
  3.01737746e-04],
[ 9.46709793e-03,  2.65137409e-04, -1.60899549e-03,
```

Model implementation