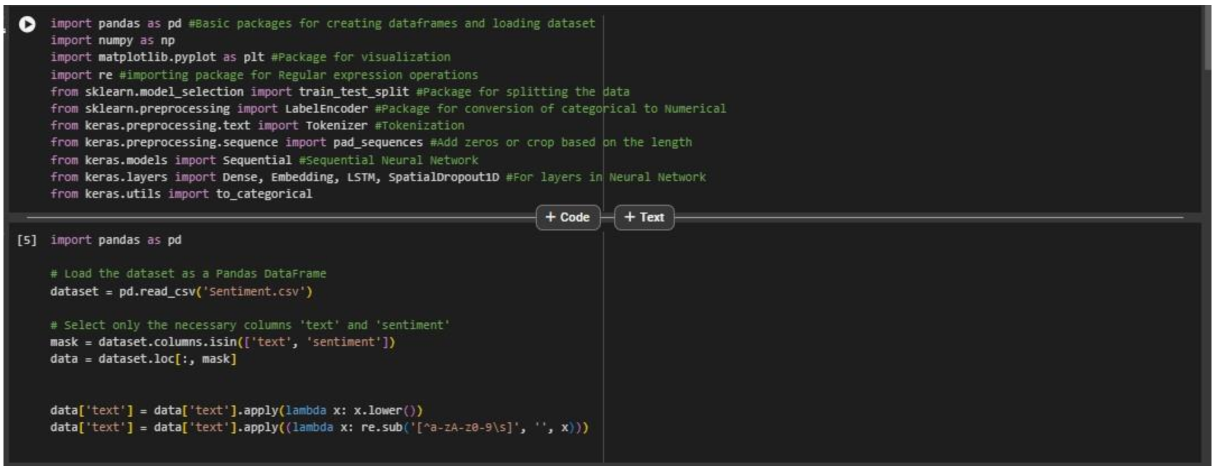
NEURAL NETWORKING & DEEP LEARNING ASSIGNMENT-9

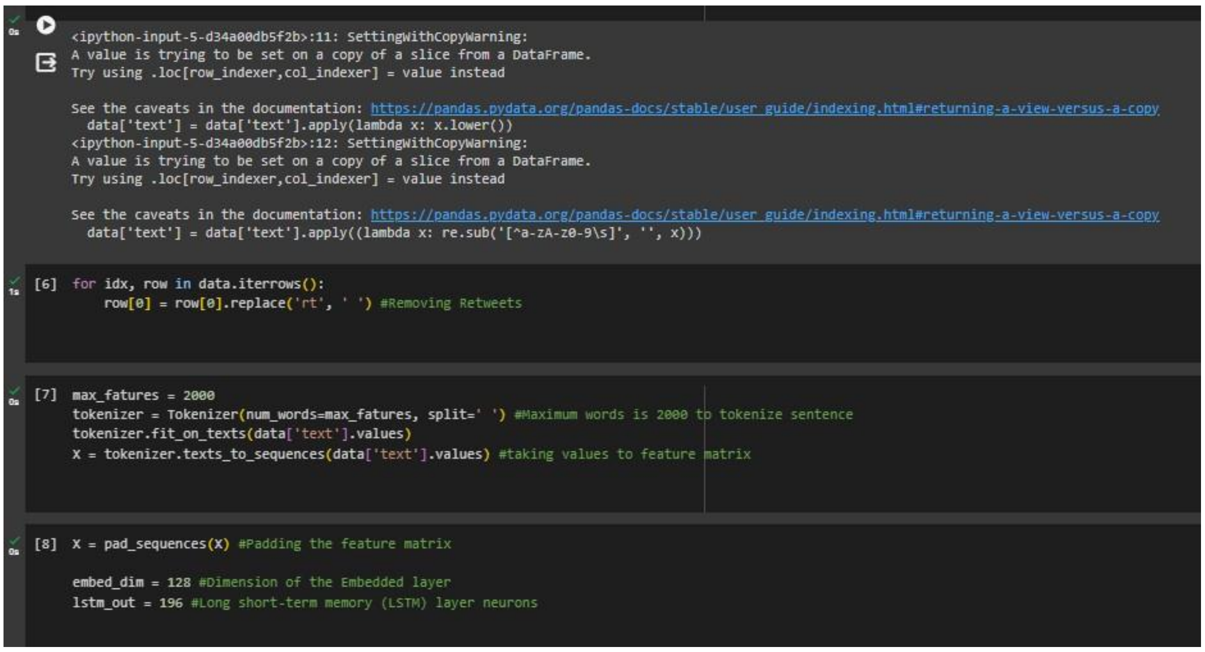
CHARITHA GONGATI (700756538)

GITHUB LINK: https://github.com/CharithaGongati



```
import pandas as pd #Basic packages for creating dataframes and loading dataset
import numpy as np
import matplotlib.pyplot as plt #Package for visualization
import re #importing package for Regular expression operations
from sklearn.model_selection import train_test_split #Package for splitting the data
from sklearn.preprocessing import LabelEncoder #Package for conversion of categorical to Numerical
from keras.preprocessing.text import Tokenizer #Tokenization
from keras.preprocessing.sequence import pad_sequences #Add zeros or crop based on the length
from keras.models import Sequential #Sequential Neural Network
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D #For layers in Neural Network
from keras.utils import to_categorical
```

```
[5] import pandas as pd

    # Load the dataset as a Pandas DataFrame
    dataset = pd.read_csv('Sentiment.csv')

    # Select only the necessary columns 'text' and 'sentiment'
    mask = dataset.columns.isin(['text', 'sentiment'])
    data = dataset.loc[:, mask]


    data['text'] = data['text'].apply(lambda x: x.lower())
    data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))
```

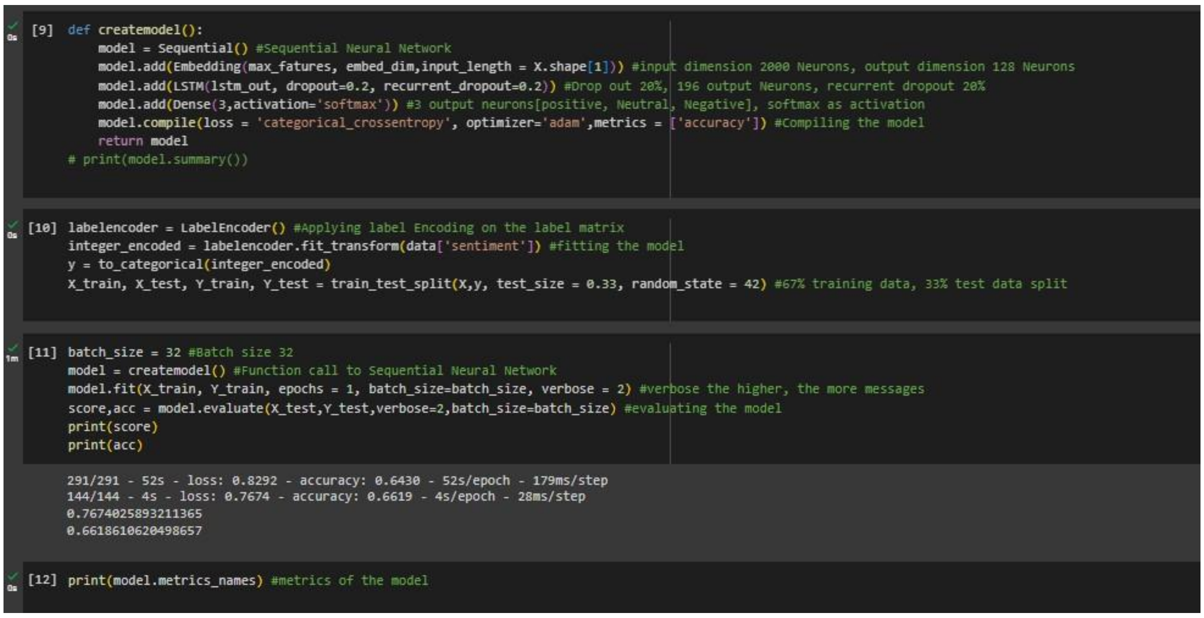


```
<ipython-input-5-d34a00db5f2b>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['text'] = data['text'].apply(lambda x: x.lower())
<ipython-input-5-d34a00db5f2b>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))
```

```
[6] for idx, row in data.iterrows():
        row[0] = row[0].replace('rt', ' ') #Removing Retweets
```

```
[7] max_fatures = 2000
    tokenizer = Tokenizer(num_words=max_fatures, split=' ') #Maximum words is 2000 to tokenize sentence
    tokenizer.fit_on_texts(data['text'].values)
    X = tokenizer.texts_to_sequences(data['text'].values) #taking values to feature matrix
```

```
[8] X = pad_sequences(X) #Padding the feature matrix

    embed_dim = 128 #Dimension of the Embedded layer
    lstm_out = 196 #Long short-term memory (LSTM) layer neurons
```



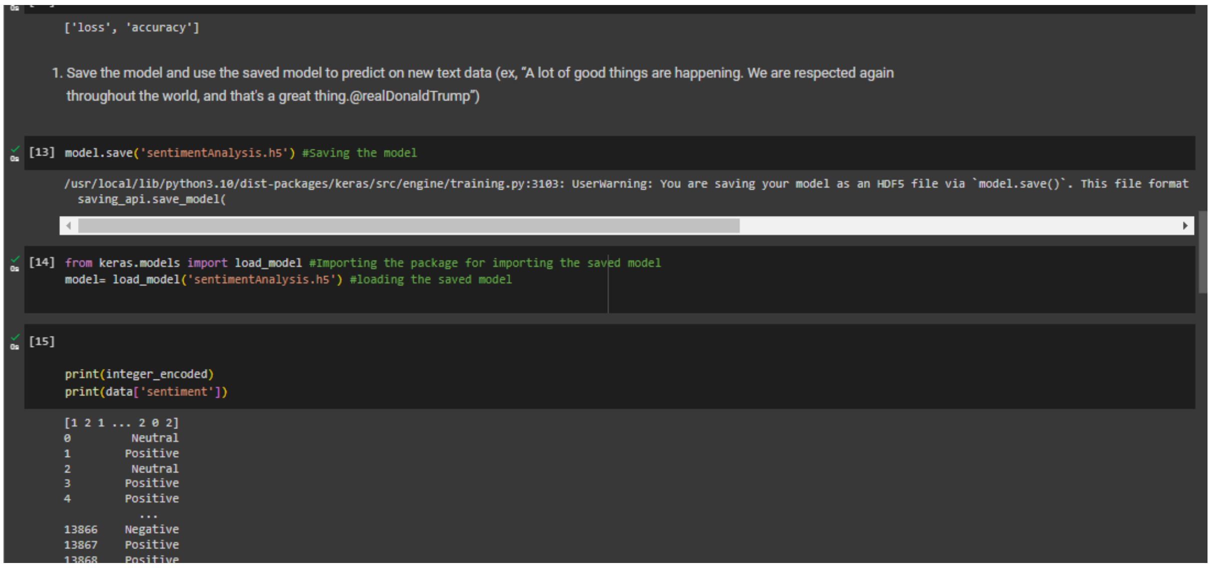
```
[9]  def createmodel():
         model = Sequential() #Sequential Neural Network
         model.add(Embedding(max_fatures, embed_dim,input_length = X.shape[1])) #input dimension 2000 Neurons, output dimension 128 Neurons
         model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2)) #Drop out 20%, 196 output Neurons, recurrent dropout 20%
         model.add(Dense(3,activation='softmax')) #3 output neurons[positive, Neutral, Negative], softmax as activation
         model.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics = ['accuracy']) #Compiling the model
         return model
     # print(model.summary())
```

```
[10] labelencoder = LabelEncoder() #Applying label Encoding on the label matrix
     integer_encoded = labelencoder.fit_transform(data['sentiment']) #fitting the model
     y = to_categorical(integer_encoded)
     X_train, X_test, Y_train, Y_test = train_test_split(X,y, test_size = 0.33, random_state = 42) #67% training data, 33% test data split
```

```
[11] batch_size = 32 #Batch size 32
     model = createmodel() #Function call to Sequential Neural Network
     model.fit(X_train, Y_train, epochs = 1, batch_size=batch_size, verbose = 2) #verbose the higher, the more messages
     score,acc = model.evaluate(X_test,Y_test,verbose=2,batch_size=batch_size) #evaluating the model
     print(score)
     print(acc)
```

```
291/291 - 52s - loss: 0.8292 - accuracy: 0.6430 - 52s/epoch - 179ms/step
144/144 - 4s - loss: 0.7674 - accuracy: 0.6619 - 4s/epoch - 28ms/step
0.7674025893211365
0.6618610620498657
```

```
[12] print(model.metrics_names) #metrics of the model
```



```
['loss', 'accuracy']
```

1. Save the model and use the saved model to predict on new text data (ex, "A lot of good things are happening. We are respected again throughout the world, and that's a great thing.@realDonaldTrump")

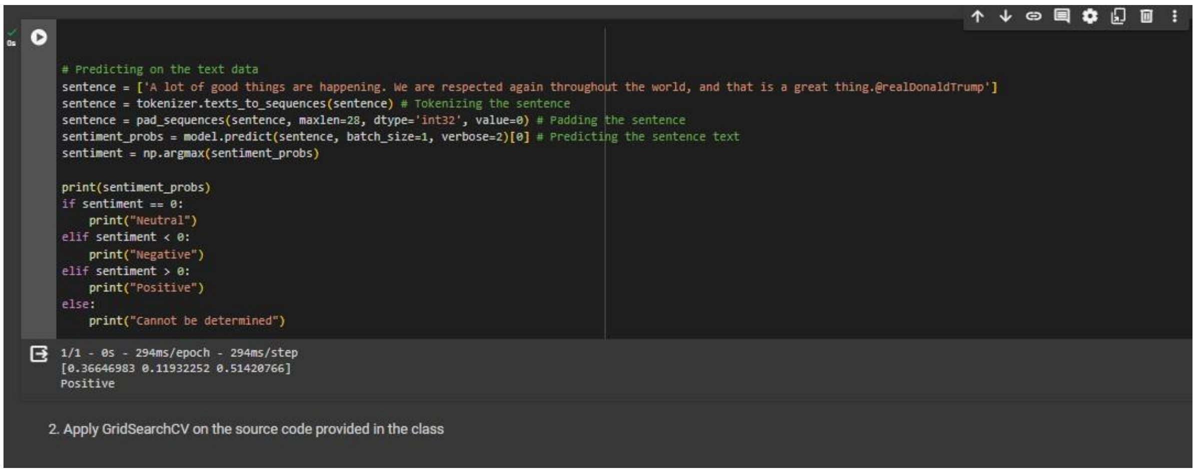
```
[13] model.save('sentimentAnalysis.h5') #Saving the model
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format
  saving_api.save_model(
```

```
[14] from keras.models import load_model #Importing the package for importing the saved model
     model= load_model('sentimentAnalysis.h5') #loading the saved model
```

```
[15]

     print(integer_encoded)
     print(data['sentiment'])
```

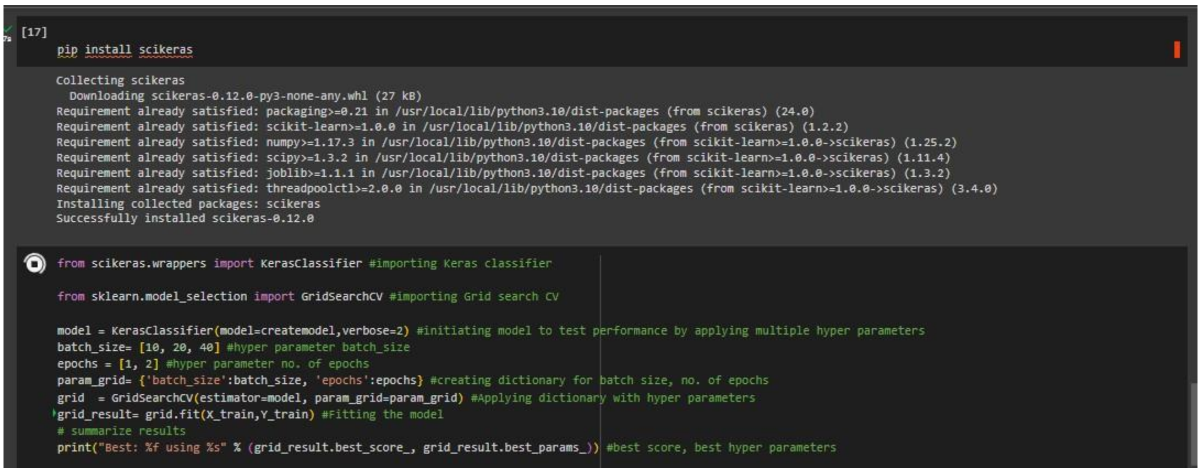
```
[1 2 1 ... 2 0 2]
0         Neutral
1        Positive
2         Neutral
3        Positive
4        Positive
           ...
13866    Negative
13867    Positive
13868    Positive
```



```
# Predicting on the text data
sentence = ['A lot of good things are happening. We are respected again throughout the world, and that is a great thing.@realDonaldTrump']
sentence = tokenizer.texts_to_sequences(sentence) # Tokenizing the sentence
sentence = pad_sequences(sentence, maxlen=28, dtype='int32', value=0) # Padding the sentence
sentiment_probs = model.predict(sentence, batch_size=1, verbose=2)[0] # Predicting the sentence text
sentiment = np.argmax(sentiment_probs)

print(sentiment_probs)
if sentiment == 0:
    print("Neutral")
elif sentiment < 0:
    print("Negative")
elif sentiment > 0:
    print("Positive")
else:
    print("Cannot be determined")
```

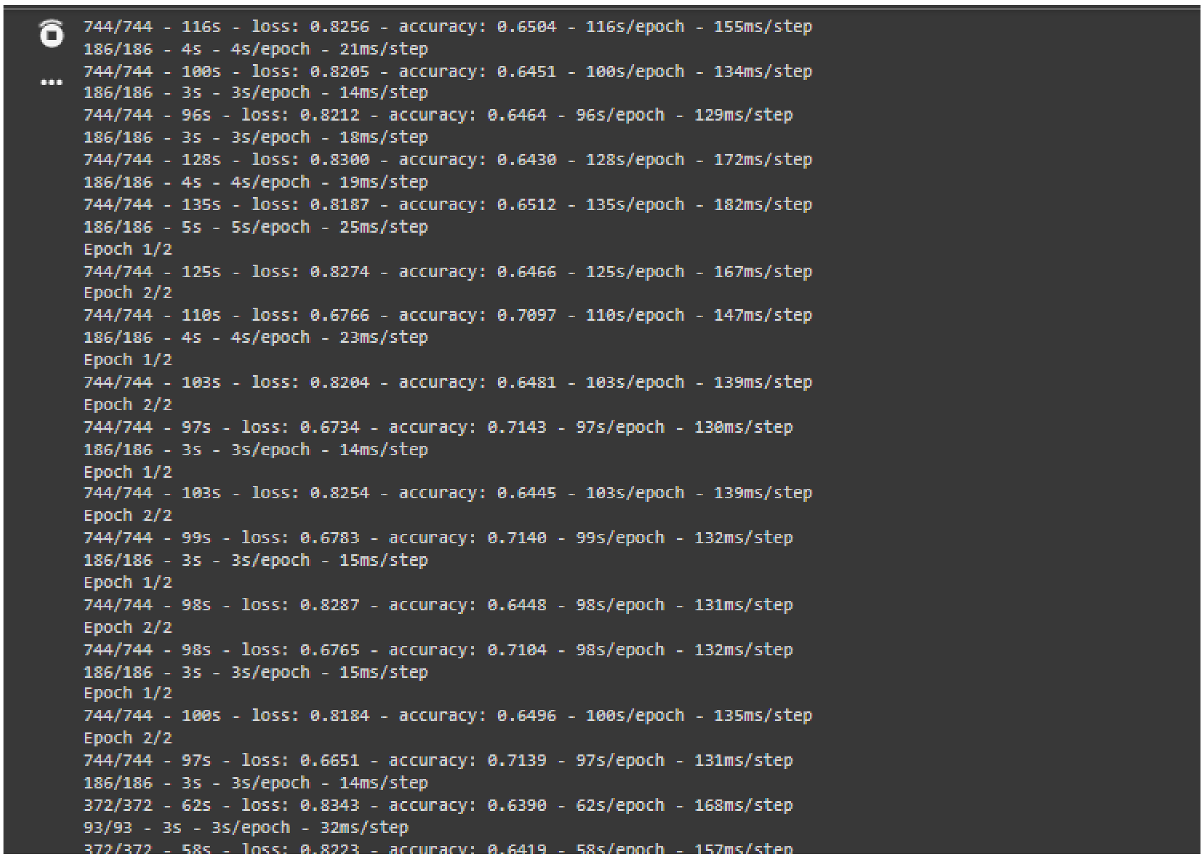
```
1/1 - 0s - 294ms/epoch - 294ms/step
[0.36646983 0.11932252 0.51420766]
Positive
```

2. Apply GridSearchCV on the source code provided in the class



```
[17]
pip install scikeras

Collecting scikeras
  Downloading scikeras-0.12.0-py3-none-any.whl (27 kB)
Requirement already satisfied: packaging>=0.21 in /usr/local/lib/python3.10/dist-packages (from scikeras) (24.0)
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikeras) (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (3.4.0)
Installing collected packages: scikeras
Successfully installed scikeras-0.12.0
```

```python
from scikeras.wrappers import KerasClassifier #importing Keras classifier

from sklearn.model_selection import GridSearchCV #importing Grid search CV

model = KerasClassifier(model=createmodel,verbose=2) #initiating model to test performance by applying multiple hyper parameters
batch_size= [10, 20, 40] #hyper parameter batch_size
epochs = [1, 2] #hyper parameter no. of epochs
param_grid= {'batch_size':batch_size, 'epochs':epochs} #creating dictionary for batch size, no. of epochs
grid  = GridSearchCV(estimator=model, param_grid=param_grid) #Applying dictionary with hyper parameters
grid_result= grid.fit(X_train,Y_train) #Fitting the model
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_)) #best score, best hyper parameters
```



```
744/744 - 116s - loss: 0.8256 - accuracy: 0.6504 - 116s/epoch - 155ms/step
186/186 - 4s - 4s/epoch - 21ms/step
744/744 - 100s - loss: 0.8205 - accuracy: 0.6451 - 100s/epoch - 134ms/step
186/186 - 3s - 3s/epoch - 14ms/step
744/744 - 96s - loss: 0.8212 - accuracy: 0.6464 - 96s/epoch - 129ms/step
186/186 - 3s - 3s/epoch - 18ms/step
744/744 - 128s - loss: 0.8300 - accuracy: 0.6430 - 128s/epoch - 172ms/step
186/186 - 4s - 4s/epoch - 19ms/step
744/744 - 135s - loss: 0.8187 - accuracy: 0.6512 - 135s/epoch - 182ms/step
186/186 - 5s - 5s/epoch - 25ms/step
Epoch 1/2
744/744 - 125s - loss: 0.8274 - accuracy: 0.6466 - 125s/epoch - 167ms/step
Epoch 2/2
744/744 - 110s - loss: 0.6766 - accuracy: 0.7097 - 110s/epoch - 147ms/step
186/186 - 4s - 4s/epoch - 23ms/step
Epoch 1/2
744/744 - 103s - loss: 0.8204 - accuracy: 0.6481 - 103s/epoch - 139ms/step
Epoch 2/2
744/744 - 97s - loss: 0.6734 - accuracy: 0.7143 - 97s/epoch - 130ms/step
186/186 - 3s - 3s/epoch - 14ms/step
Epoch 1/2
744/744 - 103s - loss: 0.8254 - accuracy: 0.6445 - 103s/epoch - 139ms/step
Epoch 2/2
744/744 - 99s - loss: 0.6783 - accuracy: 0.7140 - 99s/epoch - 132ms/step
186/186 - 3s - 3s/epoch - 15ms/step
Epoch 1/2
744/744 - 98s - loss: 0.8287 - accuracy: 0.6448 - 98s/epoch - 131ms/step
Epoch 2/2
744/744 - 98s - loss: 0.6765 - accuracy: 0.7104 - 98s/epoch - 132ms/step
186/186 - 3s - 3s/epoch - 15ms/step
Epoch 1/2
744/744 - 100s - loss: 0.8184 - accuracy: 0.6496 - 100s/epoch - 135ms/step
Epoch 2/2
744/744 - 97s - loss: 0.6651 - accuracy: 0.7139 - 97s/epoch - 131ms/step
186/186 - 3s - 3s/epoch - 14ms/step
372/372 - 62s - loss: 0.8343 - accuracy: 0.6390 - 62s/epoch - 168ms/step
93/93 - 3s - 3s/epoch - 32ms/step
372/372 - 58s - loss: 0.8223 - accuracy: 0.6419 - 58s/epoch - 157ms/step
```