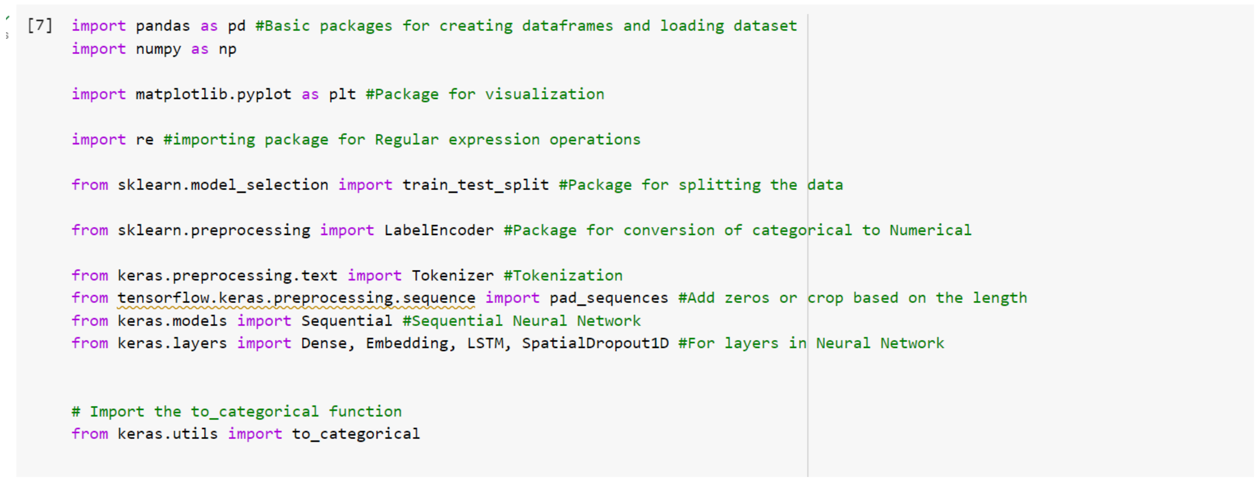
Fall 2023: CS5720 Neural Networks & Deep Learning - ICP-9 Assignment-9

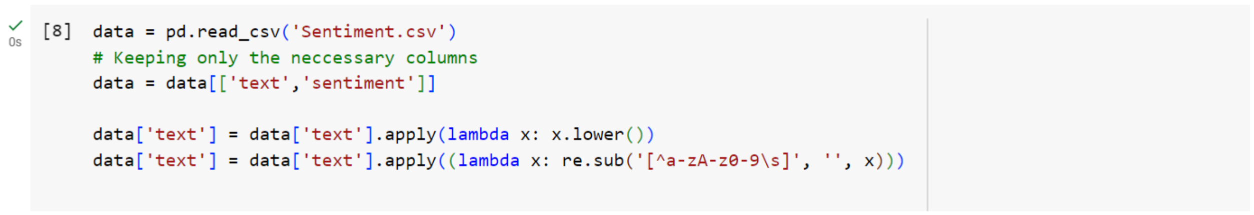
NAME:RAJYALAKSHMI GOTTIPATI

STUDENT ID:700745186

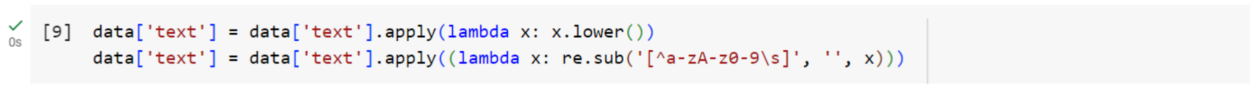
Github Link:https://github.com/rajigottipati/icp-9.git



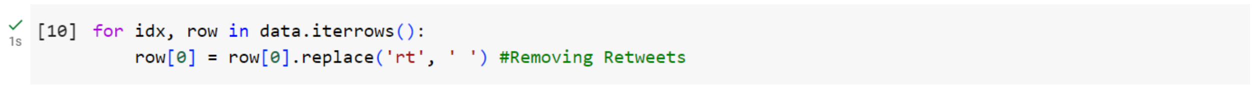
```
[7] import pandas as pd #Basic packages for creating dataframes and loading dataset
    import numpy as np

    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 tensorflow.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


    # Import the to_categorical function
    from keras.utils import to_categorical
```



```
[8] data = pd.read_csv('Sentiment.csv')
    # Keeping only the neccessary columns
    data = data[['text','sentiment']]

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



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



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

```
[11] 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
```

```
[12]
     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


     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())
```

```
[13] labelencoder = LabelEncoder()
     integer_encoded = labelencoder.fit_transform(data['sentiment'])

     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)

     batch_size = 32
     model = createmodel()
     model.fit(X_train, Y_train, epochs = 1, batch_size=batch_size, verbose = 2)
     score,acc = model.evaluate(X_test,Y_test,verbose=2,batch_size=batch_size)
     print(score)
     print(acc)
     print(model.metrics_names)
```

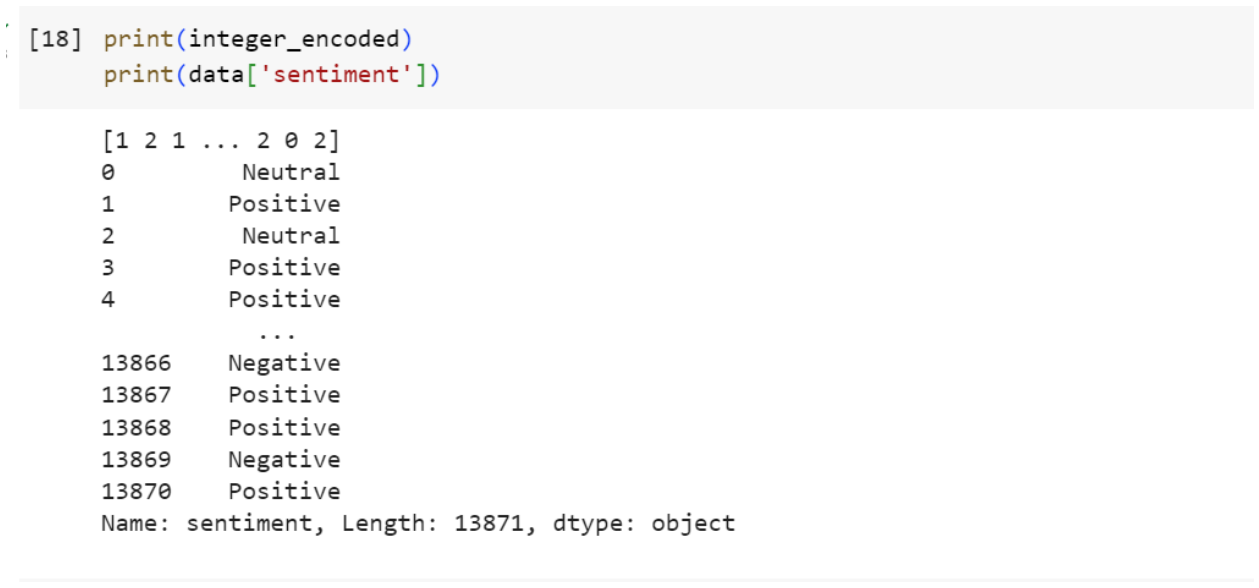
```
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running
291/291 - 61s - loss: 0.8183 - accuracy: 0.6451 - 61s/epoch - 208ms/step
144/144 - 2s - loss: 0.7441 - accuracy: 0.6791 - 2s/epoch - 15ms/step
0.7440932989120483
0.6791175007820129
['loss', 'accuracy']
```

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

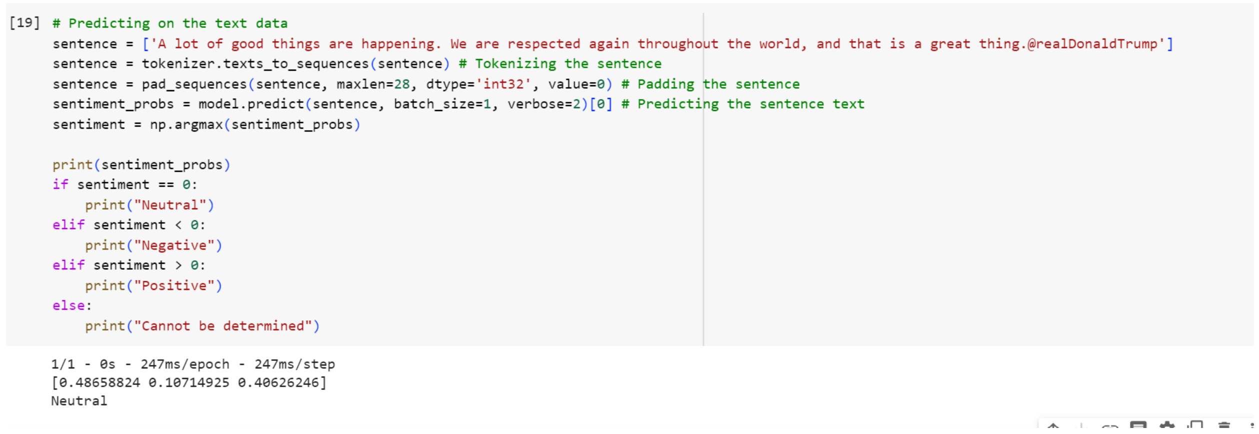
```
[17] from keras.models import load_model #Importing the package for importing the saved model
     model= load_model('sentimentAnalysis.h5') #loading the saved model
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a generic GPU kernel as fallback when running
```

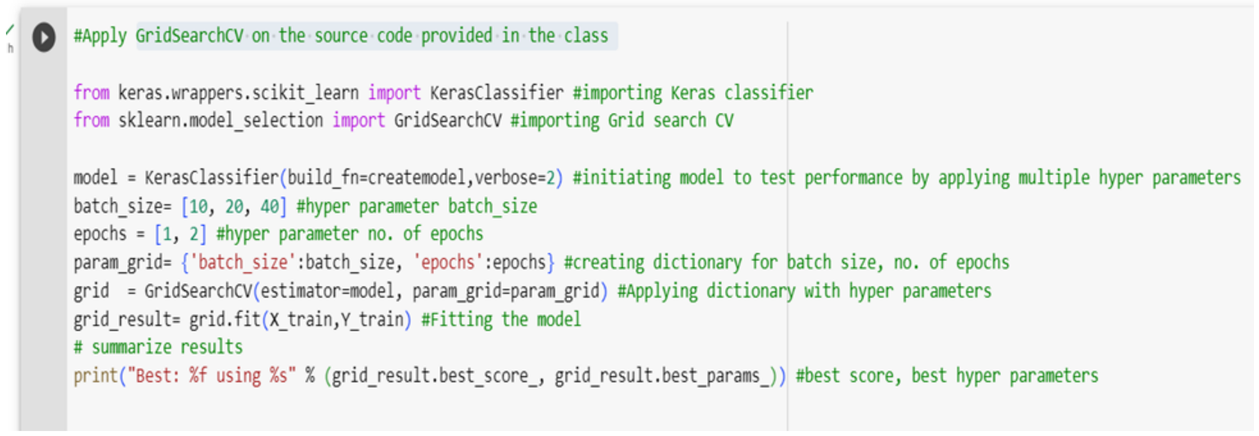
```
[18] print(integer_encoded)
     print(data['sentiment'])
```



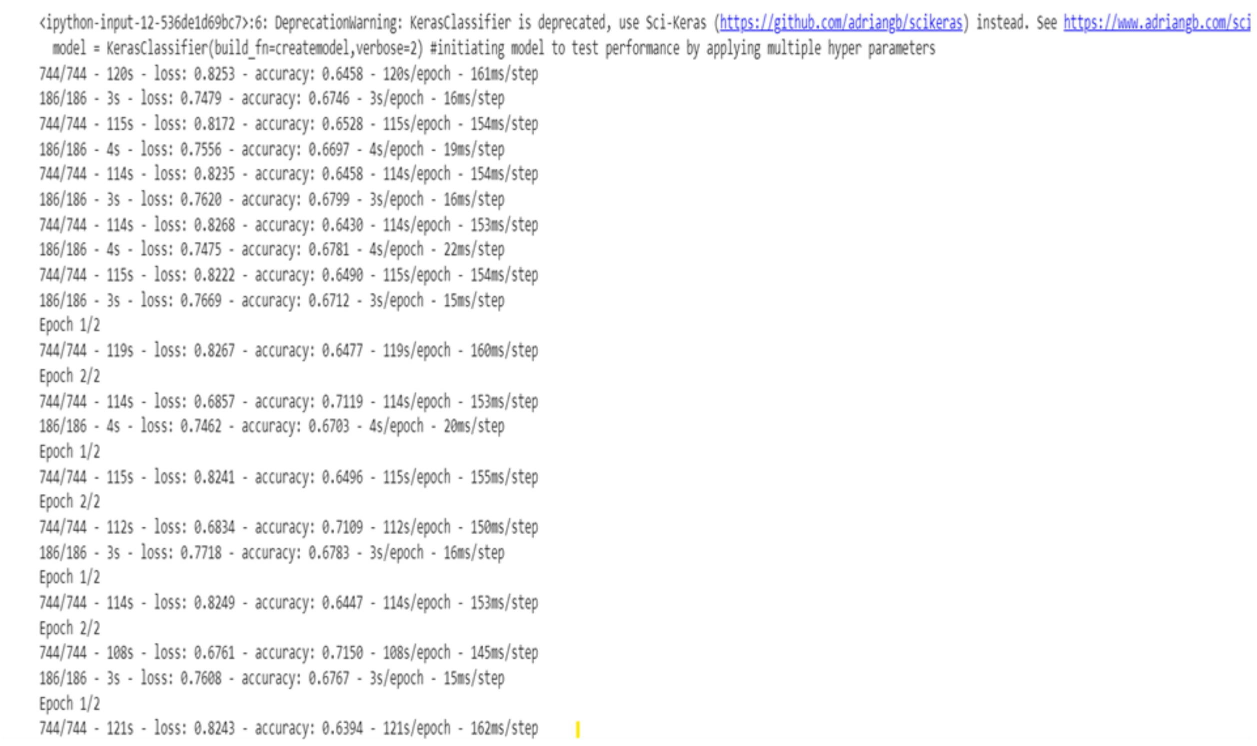
```
[18] print(integer_encoded)
     print(data['sentiment'])

     [1 2 1 ... 2 0 2]
     0          Neutral
     1         Positive
     2          Neutral
     3         Positive
     4         Positive
                 ...
     13866     Negative
     13867     Positive
     13868     Positive
     13869     Negative
     13870     Positive
     Name: sentiment, Length: 13871, dtype: object
```



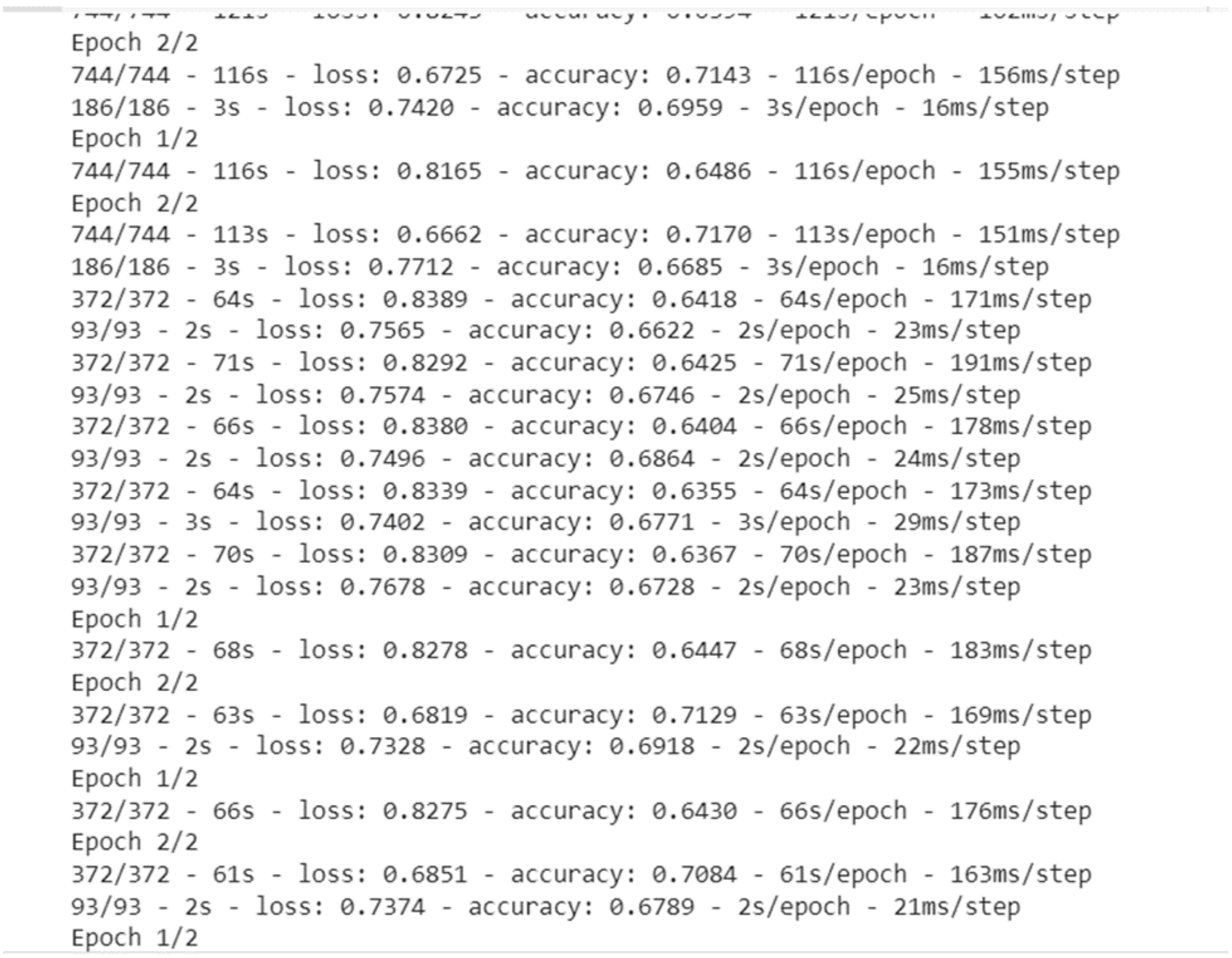
```
[19] # Predicting on the text data
     sentence = ['A lot of good things are happening. We are respected again throughout the world, and that is a great thing.@realDonaldTrump']
     sentence = tokenizer.texts_to_sequences(sentence) # Tokenizing the sentence
     sentence = pad_sequences(sentence, 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 - 247ms/epoch - 247ms/step
     [0.48658824 0.10714925 0.40626246]
     Neutral
```



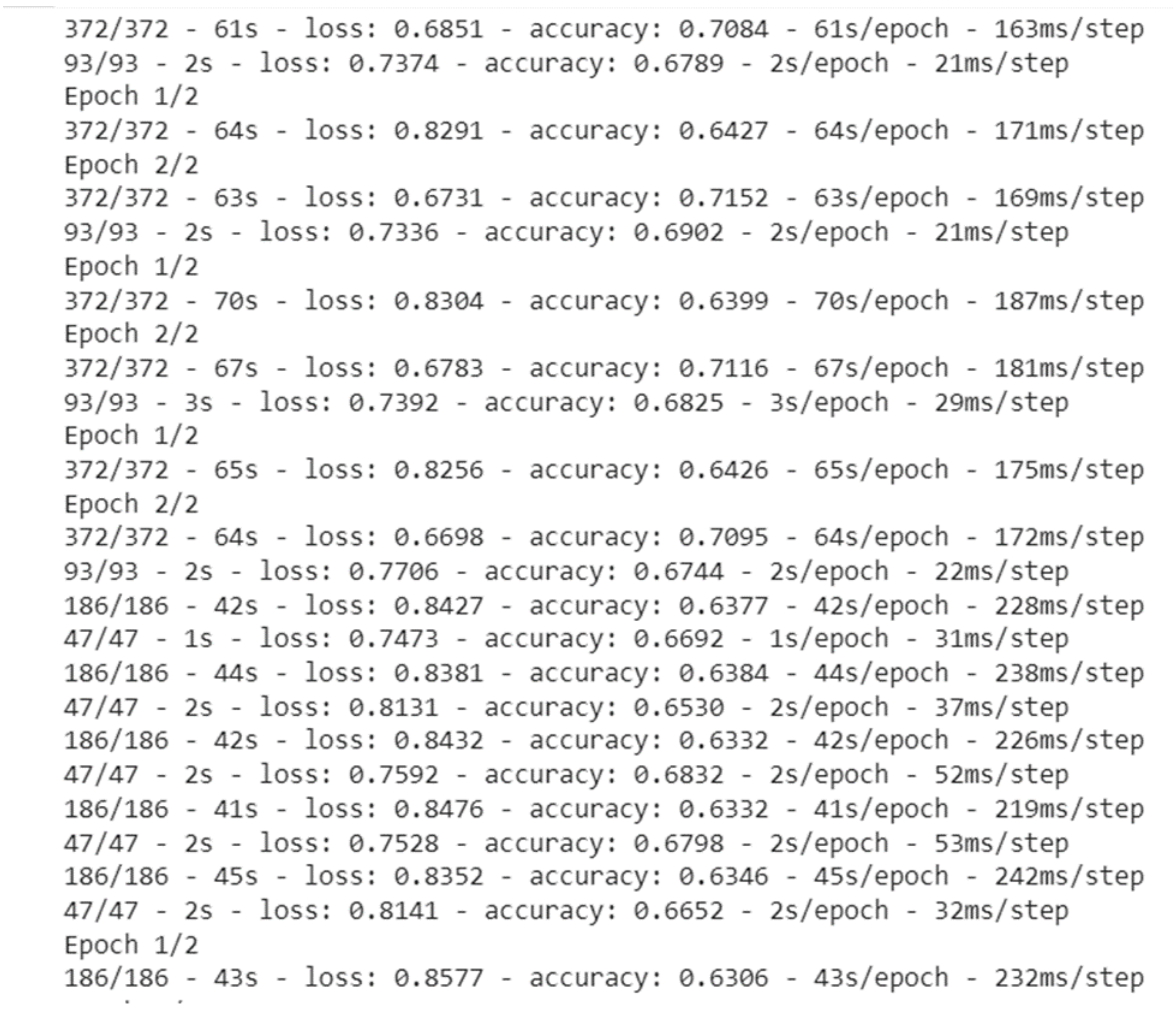
```
#Apply GridSearchCV on the source code provided in the class

from keras.wrappers.scikit_learn import KerasClassifier #importing Keras classifier
from sklearn.model_selection import GridSearchCV #importing Grid search CV

model = KerasClassifier(build_fn=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
```



```
<ipython-input-12-536de1d69bc7>:6: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead. See https://www.adriangb.com/sci
  model = KerasClassifier(build_fn=createmodel,verbose=2) #initiating model to test performance by applying multiple hyper parameters
744/744 - 120s - loss: 0.8253 - accuracy: 0.6458 - 120s/epoch - 161ms/step
186/186 - 3s - loss: 0.7479 - accuracy: 0.6746 - 3s/epoch - 16ms/step
744/744 - 115s - loss: 0.8172 - accuracy: 0.6528 - 115s/epoch - 154ms/step
186/186 - 4s - loss: 0.7556 - accuracy: 0.6697 - 4s/epoch - 19ms/step
744/744 - 114s - loss: 0.8235 - accuracy: 0.6458 - 114s/epoch - 154ms/step
186/186 - 3s - loss: 0.7620 - accuracy: 0.6799 - 3s/epoch - 16ms/step
744/744 - 114s - loss: 0.8268 - accuracy: 0.6430 - 114s/epoch - 153ms/step
186/186 - 4s - loss: 0.7475 - accuracy: 0.6781 - 4s/epoch - 22ms/step
744/744 - 115s - loss: 0.8222 - accuracy: 0.6490 - 115s/epoch - 154ms/step
186/186 - 3s - loss: 0.7669 - accuracy: 0.6712 - 3s/epoch - 15ms/step
Epoch 1/2
744/744 - 119s - loss: 0.8267 - accuracy: 0.6477 - 119s/epoch - 160ms/step
Epoch 2/2
744/744 - 114s - loss: 0.6857 - accuracy: 0.7119 - 114s/epoch - 153ms/step
186/186 - 4s - loss: 0.7462 - accuracy: 0.6703 - 4s/epoch - 20ms/step
Epoch 1/2
744/744 - 115s - loss: 0.8241 - accuracy: 0.6496 - 115s/epoch - 155ms/step
Epoch 2/2
744/744 - 112s - loss: 0.6834 - accuracy: 0.7109 - 112s/epoch - 150ms/step
186/186 - 3s - loss: 0.7718 - accuracy: 0.6783 - 3s/epoch - 16ms/step
Epoch 1/2
744/744 - 114s - loss: 0.8249 - accuracy: 0.6447 - 114s/epoch - 153ms/step
Epoch 2/2
744/744 - 108s - loss: 0.6761 - accuracy: 0.7150 - 108s/epoch - 145ms/step
186/186 - 3s - loss: 0.7608 - accuracy: 0.6767 - 3s/epoch - 15ms/step
Epoch 1/2
744/744 - 121s - loss: 0.8243 - accuracy: 0.6394 - 121s/epoch - 162ms/step
```



```
744/744 - 121s - loss: 0.8249 - accuracy: 0.6594 - 121s/epoch - 162ms/step
Epoch 2/2
744/744 - 116s - loss: 0.6725 - accuracy: 0.7143 - 116s/epoch - 156ms/step
186/186 - 3s - loss: 0.7420 - accuracy: 0.6959 - 3s/epoch - 16ms/step
Epoch 1/2
744/744 - 116s - loss: 0.8165 - accuracy: 0.6486 - 116s/epoch - 155ms/step
Epoch 2/2
744/744 - 113s - loss: 0.6662 - accuracy: 0.7170 - 113s/epoch - 151ms/step
186/186 - 3s - loss: 0.7712 - accuracy: 0.6685 - 3s/epoch - 16ms/step
372/372 - 64s - loss: 0.8389 - accuracy: 0.6418 - 64s/epoch - 171ms/step
93/93 - 2s - loss: 0.7565 - accuracy: 0.6622 - 2s/epoch - 23ms/step
372/372 - 71s - loss: 0.8292 - accuracy: 0.6425 - 71s/epoch - 191ms/step
93/93 - 2s - loss: 0.7574 - accuracy: 0.6746 - 2s/epoch - 25ms/step
372/372 - 66s - loss: 0.8380 - accuracy: 0.6404 - 66s/epoch - 178ms/step
93/93 - 2s - loss: 0.7496 - accuracy: 0.6864 - 2s/epoch - 24ms/step
372/372 - 64s - loss: 0.8339 - accuracy: 0.6355 - 64s/epoch - 173ms/step
93/93 - 3s - loss: 0.7402 - accuracy: 0.6771 - 3s/epoch - 29ms/step
372/372 - 70s - loss: 0.8309 - accuracy: 0.6367 - 70s/epoch - 187ms/step
93/93 - 2s - loss: 0.7678 - accuracy: 0.6728 - 2s/epoch - 23ms/step
Epoch 1/2
372/372 - 68s - loss: 0.8278 - accuracy: 0.6447 - 68s/epoch - 183ms/step
Epoch 2/2
372/372 - 63s - loss: 0.6819 - accuracy: 0.7129 - 63s/epoch - 169ms/step
93/93 - 2s - loss: 0.7328 - accuracy: 0.6918 - 2s/epoch - 22ms/step
Epoch 1/2
372/372 - 66s - loss: 0.8275 - accuracy: 0.6430 - 66s/epoch - 176ms/step
Epoch 2/2
372/372 - 61s - loss: 0.6851 - accuracy: 0.7084 - 61s/epoch - 163ms/step
93/93 - 2s - loss: 0.7374 - accuracy: 0.6789 - 2s/epoch - 21ms/step
Epoch 1/2
```



```
372/372 - 61s - loss: 0.6851 - accuracy: 0.7084 - 61s/epoch - 163ms/step
93/93 - 2s - loss: 0.7374 - accuracy: 0.6789 - 2s/epoch - 21ms/step
Epoch 1/2
372/372 - 64s - loss: 0.8291 - accuracy: 0.6427 - 64s/epoch - 171ms/step
Epoch 2/2
372/372 - 63s - loss: 0.6731 - accuracy: 0.7152 - 63s/epoch - 169ms/step
93/93 - 2s - loss: 0.7336 - accuracy: 0.6902 - 2s/epoch - 21ms/step
Epoch 1/2
372/372 - 70s - loss: 0.8304 - accuracy: 0.6399 - 70s/epoch - 187ms/step
Epoch 2/2
372/372 - 67s - loss: 0.6783 - accuracy: 0.7116 - 67s/epoch - 181ms/step
93/93 - 3s - loss: 0.7392 - accuracy: 0.6825 - 3s/epoch - 29ms/step
Epoch 1/2
372/372 - 65s - loss: 0.8256 - accuracy: 0.6426 - 65s/epoch - 175ms/step
Epoch 2/2
372/372 - 64s - loss: 0.6698 - accuracy: 0.7095 - 64s/epoch - 172ms/step
93/93 - 2s - loss: 0.7706 - accuracy: 0.6744 - 2s/epoch - 22ms/step
186/186 - 42s - loss: 0.8427 - accuracy: 0.6377 - 42s/epoch - 228ms/step
47/47 - 1s - loss: 0.7473 - accuracy: 0.6692 - 1s/epoch - 31ms/step
186/186 - 44s - loss: 0.8381 - accuracy: 0.6384 - 44s/epoch - 238ms/step
47/47 - 2s - loss: 0.8131 - accuracy: 0.6530 - 2s/epoch - 37ms/step
186/186 - 42s - loss: 0.8432 - accuracy: 0.6332 - 42s/epoch - 226ms/step
47/47 - 2s - loss: 0.7592 - accuracy: 0.6832 - 2s/epoch - 52ms/step
186/186 - 41s - loss: 0.8476 - accuracy: 0.6332 - 41s/epoch - 219ms/step
47/47 - 2s - loss: 0.7528 - accuracy: 0.6798 - 2s/epoch - 53ms/step
186/186 - 45s - loss: 0.8352 - accuracy: 0.6346 - 45s/epoch - 242ms/step
47/47 - 2s - loss: 0.8141 - accuracy: 0.6652 - 2s/epoch - 32ms/step
Epoch 1/2
186/186 - 43s - loss: 0.8577 - accuracy: 0.6306 - 43s/epoch - 232ms/step
```