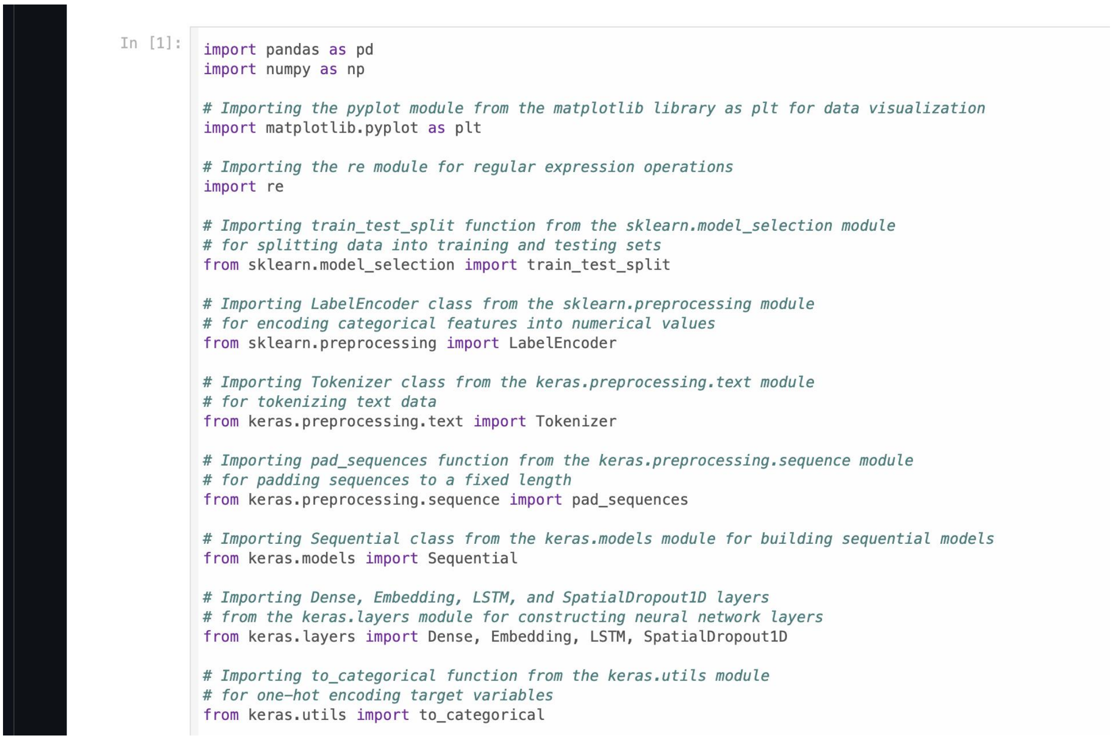
Spring 2024: CS5720

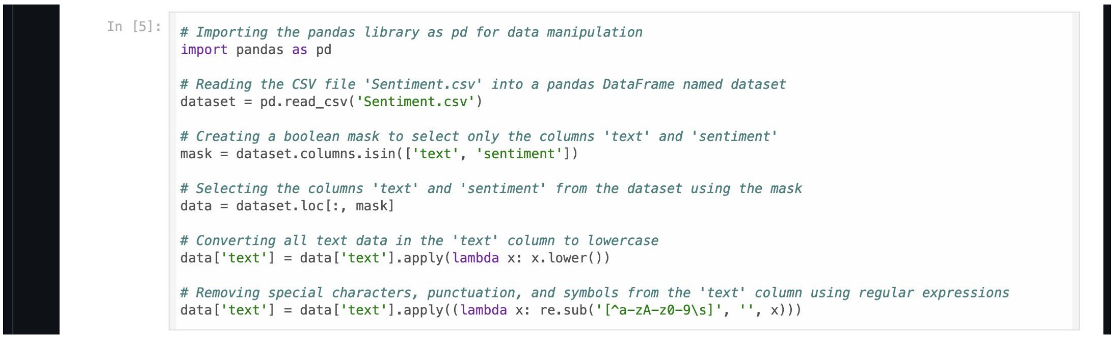
Neural Networks and Deep Learning - ICP10

Student Name: Vegi Akhilandeswari
Student ID: 700758173

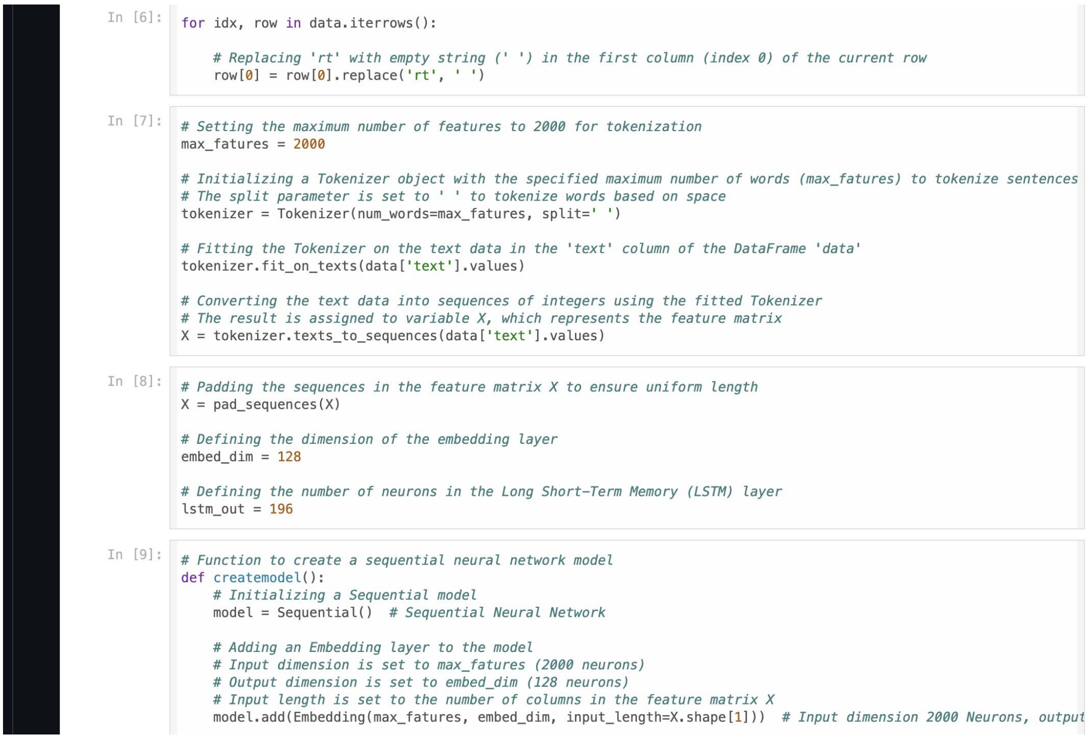
Github: https://github.com/akhilandeswariVegi/Assignment9_NNDL



```
In [1]:
import pandas as pd
import numpy as np

# Importing the pyplot module from the matplotlib library as plt for data visualization
import matplotlib.pyplot as plt

# Importing the re module for regular expression operations
import re

# Importing train_test_split function from the sklearn.model_selection module
# for splitting data into training and testing sets
from sklearn.model_selection import train_test_split

# Importing LabelEncoder class from the sklearn.preprocessing module
# for encoding categorical features into numerical values
from sklearn.preprocessing import LabelEncoder

# Importing Tokenizer class from the keras.preprocessing.text module
# for tokenizing text data
from keras.preprocessing.text import Tokenizer

# Importing pad_sequences function from the keras.preprocessing.sequence module
# for padding sequences to a fixed length
from keras.preprocessing.sequence import pad_sequences

# Importing Sequential class from the keras.models module for building sequential models
from keras.models import Sequential

# Importing Dense, Embedding, LSTM, and SpatialDropout1D layers
# from the keras.layers module for constructing neural network layers
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D

# Importing to_categorical function from the keras.utils module
# for one-hot encoding target variables
from keras.utils import to_categorical
```



```
In [5]:
# Importing the pandas library as pd for data manipulation
import pandas as pd

# Reading the CSV file 'Sentiment.csv' into a pandas DataFrame named dataset
dataset = pd.read_csv('Sentiment.csv')

# Creating a boolean mask to select only the columns 'text' and 'sentiment'
mask = dataset.columns.isin(['text', 'sentiment'])

# Selecting the columns 'text' and 'sentiment' from the dataset using the mask
data = dataset.loc[:, mask]

# Converting all text data in the 'text' column to lowercase
data['text'] = data['text'].apply(lambda x: x.lower())

# Removing special characters, punctuation, and symbols from the 'text' column using regular expressions
data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))
```

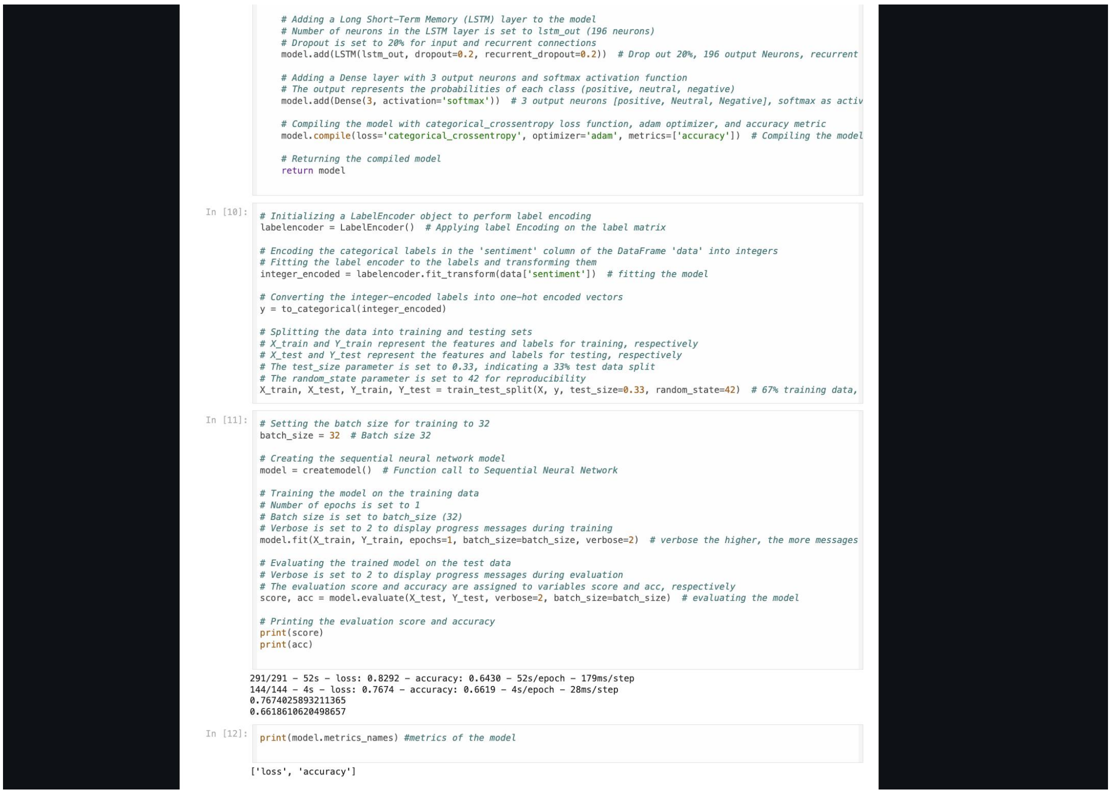


```
In [6]:  for idx, row in data.iterrows():

             # Replacing 'rt' with empty string (' ') in the first column (index 0) of the current row
             row[0] = row[0].replace('rt', ' ')
```

```
In [7]:  # Setting the maximum number of features to 2000 for tokenization
         max_fatures = 2000

         # Initializing a Tokenizer object with the specified maximum number of words (max_fatures) to tokenize sentences
         # The split parameter is set to ' ' to tokenize words based on space
         tokenizer = Tokenizer(num_words=max_fatures, split=' ')

         # Fitting the Tokenizer on the text data in the 'text' column of the DataFrame 'data'
         tokenizer.fit_on_texts(data['text'].values)

         # Converting the text data into sequences of integers using the fitted Tokenizer
         # The result is assigned to variable X, which represents the feature matrix
         X = tokenizer.texts_to_sequences(data['text'].values)
```

```
In [8]:  # Padding the sequences in the feature matrix X to ensure uniform length
         X = pad_sequences(X)

         # Defining the dimension of the embedding layer
         embed_dim = 128

         # Defining the number of neurons in the Long Short-Term Memory (LSTM) layer
         lstm_out = 196
```

```
In [9]:  # Function to create a sequential neural network model
         def createmodel():
             # Initializing a Sequential model
             model = Sequential()  # Sequential Neural Network

             # Adding an Embedding layer to the model
             # Input dimension is set to max_fatures (2000 neurons)
             # Output dimension is set to embed_dim (128 neurons)
             # Input length is set to the number of columns in the feature matrix X
             model.add(Embedding(max_fatures, embed_dim, input_length=X.shape[1]))  # Input dimension 2000 Neurons, output
```



```
             # Adding a Long Short-Term Memory (LSTM) layer to the model
             # Number of neurons in the LSTM layer is set to lstm_out (196 neurons)
             # Dropout is set to 20% for input and recurrent connections
             model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))  # Drop out 20%, 196 output Neurons, recurrent

             # Adding a Dense layer with 3 output neurons and softmax activation function
             # The output represents the probabilities of each class (positive, neutral, negative)
             model.add(Dense(3, activation='softmax'))  # 3 output neurons [positive, Neutral, Negative], softmax as activ

             # Compiling the model with categorical_crossentropy loss function, adam optimizer, and accuracy metric
             model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])  # Compiling the model

             # Returning the compiled model
             return model
```

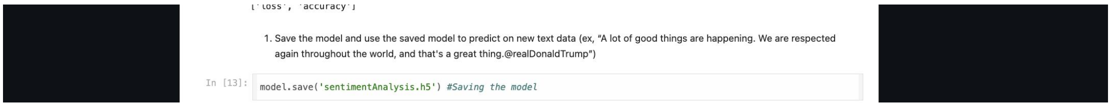
```
In [10]: # Initializing a LabelEncoder object to perform label encoding
         labelencoder = LabelEncoder()  # Applying label Encoding on the label matrix

         # Encoding the categorical labels in the 'sentiment' column of the DataFrame 'data' into integers
         # Fitting the label encoder to the labels and transforming them
         integer_encoded = labelencoder.fit_transform(data['sentiment'])  # fitting the model

         # Converting the integer-encoded labels into one-hot encoded vectors
         y = to_categorical(integer_encoded)

         # Splitting the data into training and testing sets
         # X_train and Y_train represent the features and labels for training, respectively
         # X_test and Y_test represent the features and labels for testing, respectively
         # The test_size parameter is set to 0.33, indicating a 33% test data split
         # The random_state parameter is set to 42 for reproducibility
         X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.33, random_state=42)  # 67% training data,
```

```
In [11]: # Setting the batch size for training to 32
         batch_size = 32  # Batch size 32

         # Creating the sequential neural network model
         model = createmodel()  # Function call to Sequential Neural Network

         # Training the model on the training data
         # Number of epochs is set to 1
         # Batch size is set to batch_size (32)
         # Verbose is set to 2 to display progress messages during training
         model.fit(X_train, Y_train, epochs=1, batch_size=batch_size, verbose=2)  # verbose the higher, the more messages

         # Evaluating the trained model on the test data
         # Verbose is set to 2 to display progress messages during evaluation
         # The evaluation score and accuracy are assigned to variables score and acc, respectively
         score, acc = model.evaluate(X_test, Y_test, verbose=2, batch_size=batch_size)  # evaluating the model

         # Printing the evaluation score and accuracy
         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
```

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

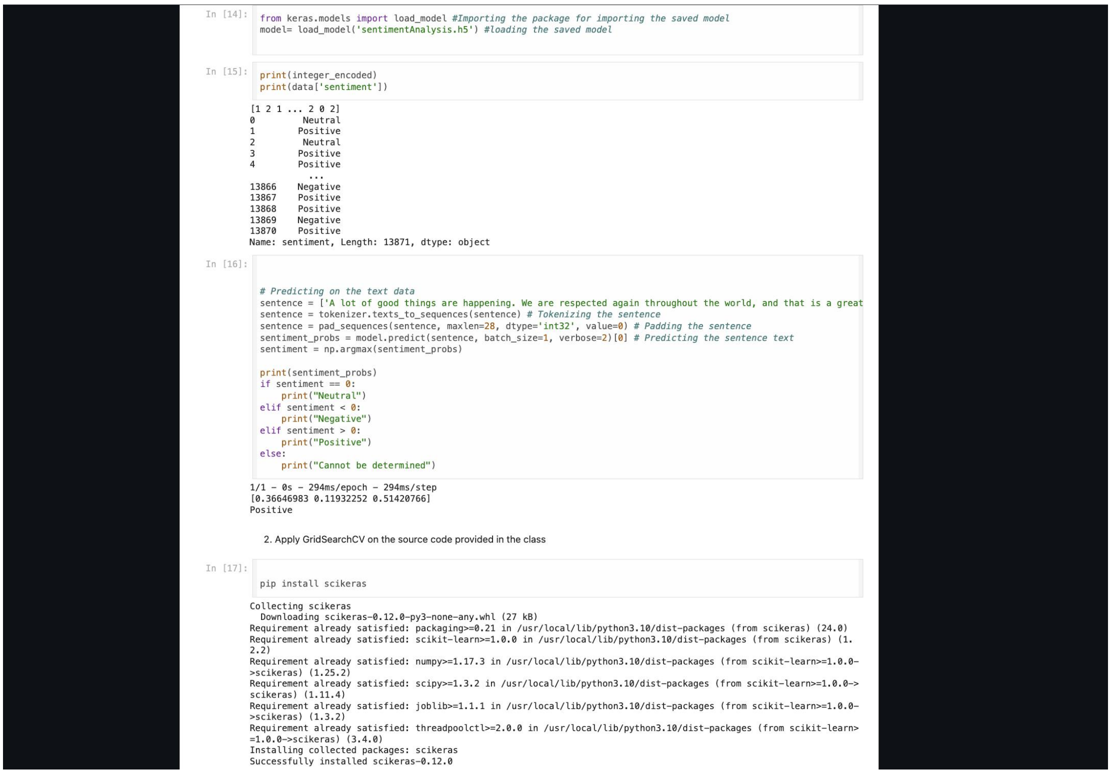
```
['loss', 'accuracy']
```



['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”)

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



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

```
In [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
13869    Negative
13870    Positive
Name: sentiment, Length: 13871, dtype: object
```

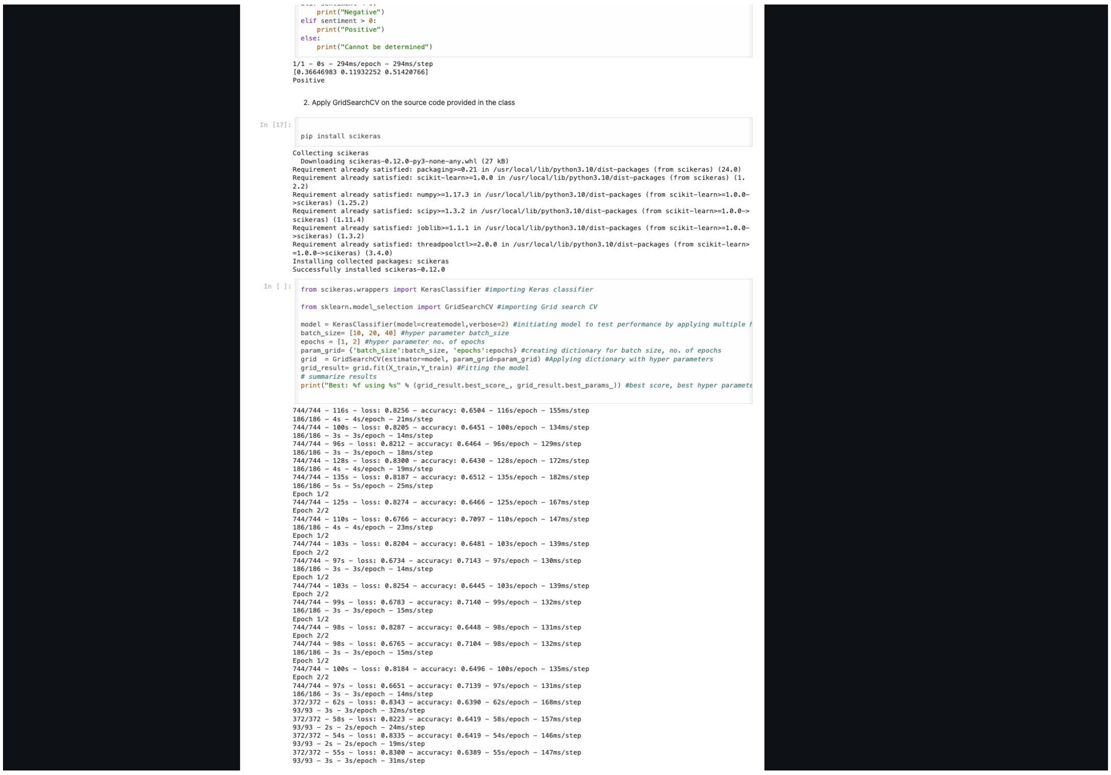
```
In [16]:


# 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
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

```
In [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
```



```
    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

```
In [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
```

```
In [ ]:
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 h
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 paramete
```

```
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
93/93 - 2s - 2s/epoch - 24ms/step
372/372 - 54s - loss: 0.8335 - accuracy: 0.6419 - 54s/epoch - 146ms/step
93/93 - 2s - 2s/epoch - 19ms/step
372/372 - 55s - loss: 0.8300 - accuracy: 0.6389 - 55s/epoch - 147ms/step
93/93 - 3s - 3s/epoch - 31ms/step
```