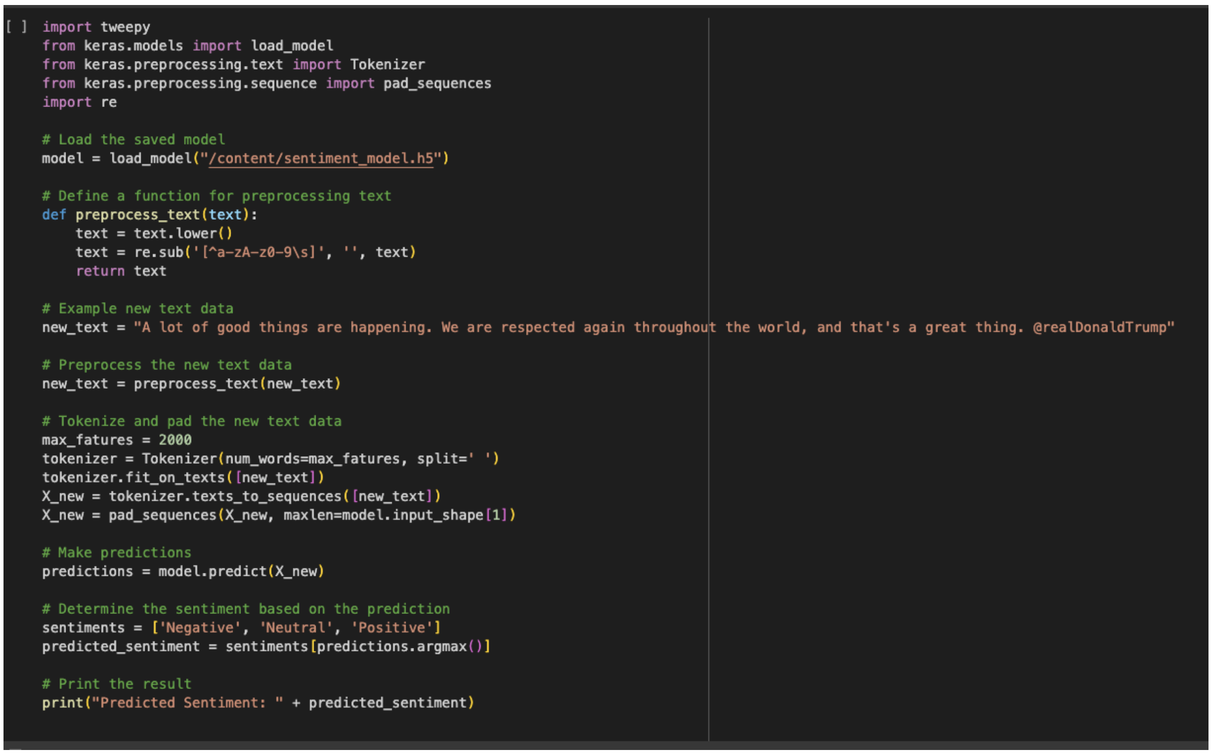
CS5720
Neural Networks & Deep Learning
Rishma Reddy Nalla
700752916

Github Link:

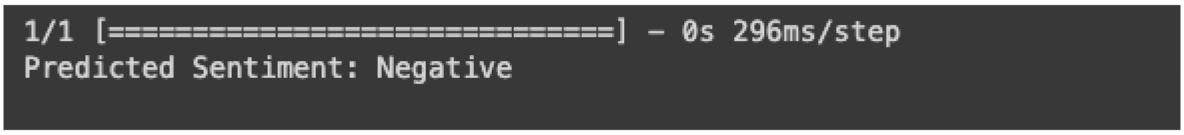
Question 1:



```
[ ] import tweepy
    from keras.models import load_model
    from keras.preprocessing.text import Tokenizer
    from keras.preprocessing.sequence import pad_sequences
    import re

    # Load the saved model
    model = load_model("/content/sentiment_model.h5")

    # Define a function for preprocessing text
    def preprocess_text(text):
        text = text.lower()
        text = re.sub('[^a-zA-z0-9\s]', '', text)
        return text

    # Example new text data
    new_text = "A lot of good things are happening. We are respected again throughout the world, and that's a great thing. @realDonaldTrump"

    # Preprocess the new text data
    new_text = preprocess_text(new_text)

    # Tokenize and pad the new text data
    max_fatures = 2000
    tokenizer = Tokenizer(num_words=max_fatures, split=' ')
    tokenizer.fit_on_texts([new_text])
    X_new = tokenizer.texts_to_sequences([new_text])
    X_new = pad_sequences(X_new, maxlen=model.input_shape[1])

    # Make predictions
    predictions = model.predict(X_new)

    # Determine the sentiment based on the prediction
    sentiments = ['Negative', 'Neutral', 'Positive']
    predicted_sentiment = sentiments[predictions.argmax()]

    # Print the result
    print("Predicted Sentiment: " + predicted_sentiment)
```

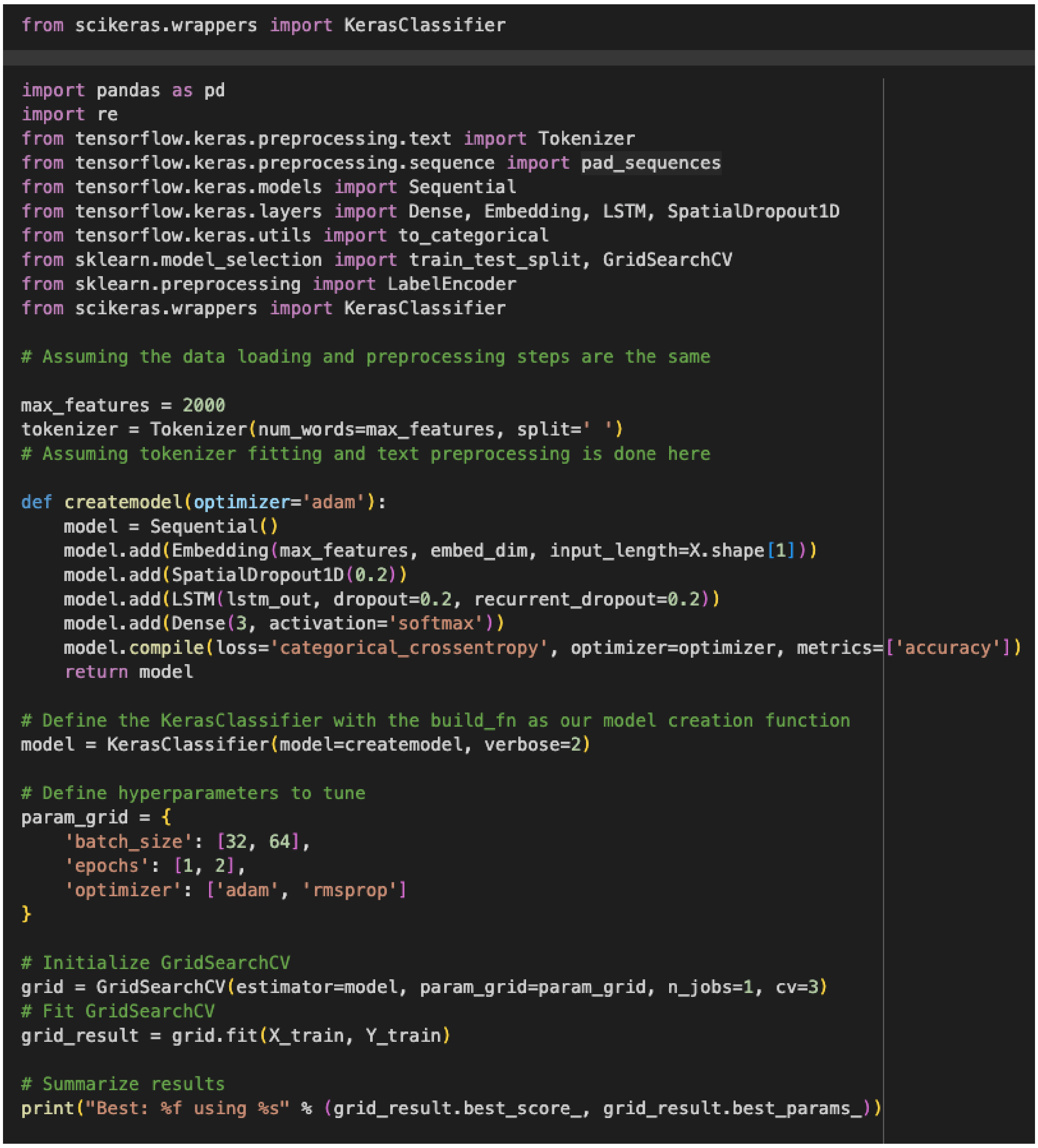
Output:

By saving the previous model and applying new text to the model for prediction, here the text will be processed using the Keras text Tokenizer to divide the sentence into chunks and feed it to the model for prediction. The Model predicted the text as Negative tone as per the training.



```
1/1 [==============================] - 0s 296ms/step
Predicted Sentiment: Negative
```

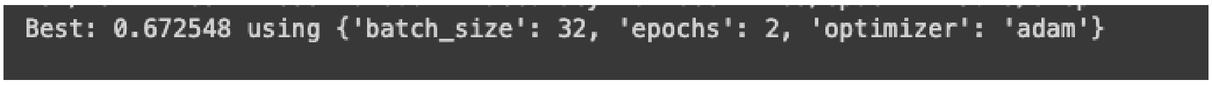
Question 2:



```python
from scikeras.wrappers import KerasClassifier
```

```python
import pandas as pd
import re
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from scikeras.wrappers import KerasClassifier

# Assuming the data loading and preprocessing steps are the same

max_features = 2000
tokenizer = Tokenizer(num_words=max_features, split=' ')
# Assuming tokenizer fitting and text preprocessing is done here

def createmodel(optimizer='adam'):
    model = Sequential()
    model.add(Embedding(max_features, embed_dim, input_length=X.shape[1]))
    model.add(SpatialDropout1D(0.2))
    model.add(LSTM(lstm_out, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model

# Define the KerasClassifier with the build_fn as our model creation function
model = KerasClassifier(model=createmodel, verbose=2)

# Define hyperparameters to tune
param_grid = {
    'batch_size': [32, 64],
    'epochs': [1, 2],
    'optimizer': ['adam', 'rmsprop']
}

# Initialize GridSearchCV
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1, cv=3)
# Fit GridSearchCV
grid_result = grid.fit(X_train, Y_train)

# Summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

Output:



```
Best: 0.672548 using {'batch_size': 32, 'epochs': 2, 'optimizer': 'adam'}
```

Explanation:

The create_model function defines the neural network architecture with an Embedding layer, SpatialDropout1D layer, LSTM layer, and a Dense output layer with a softmax activation function. A KerasClassifier wrapper is used to make the model compatible with scikit-learn's grid search functionality for hyperparameter tuning. A parameter grid is defined for batch size, number of epochs, and optimizer type, and GridSearchCV is used to find the best model configuration based on cross-validation performance. The model is trained using grid.fit(X_train, Y_train), and the best performance score along with the optimal hyperparameters are identified and printed, providing insights into the most effective settings for the text classification task.