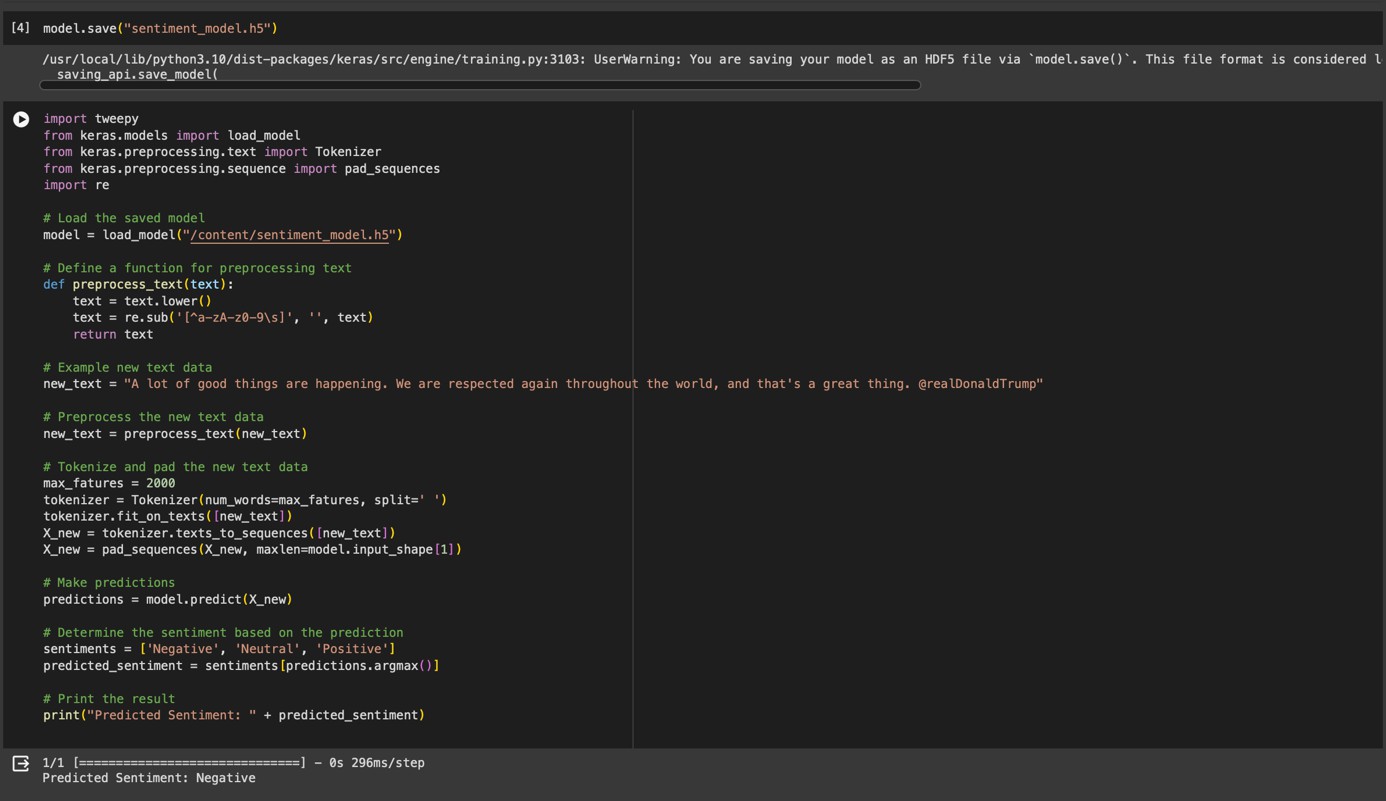
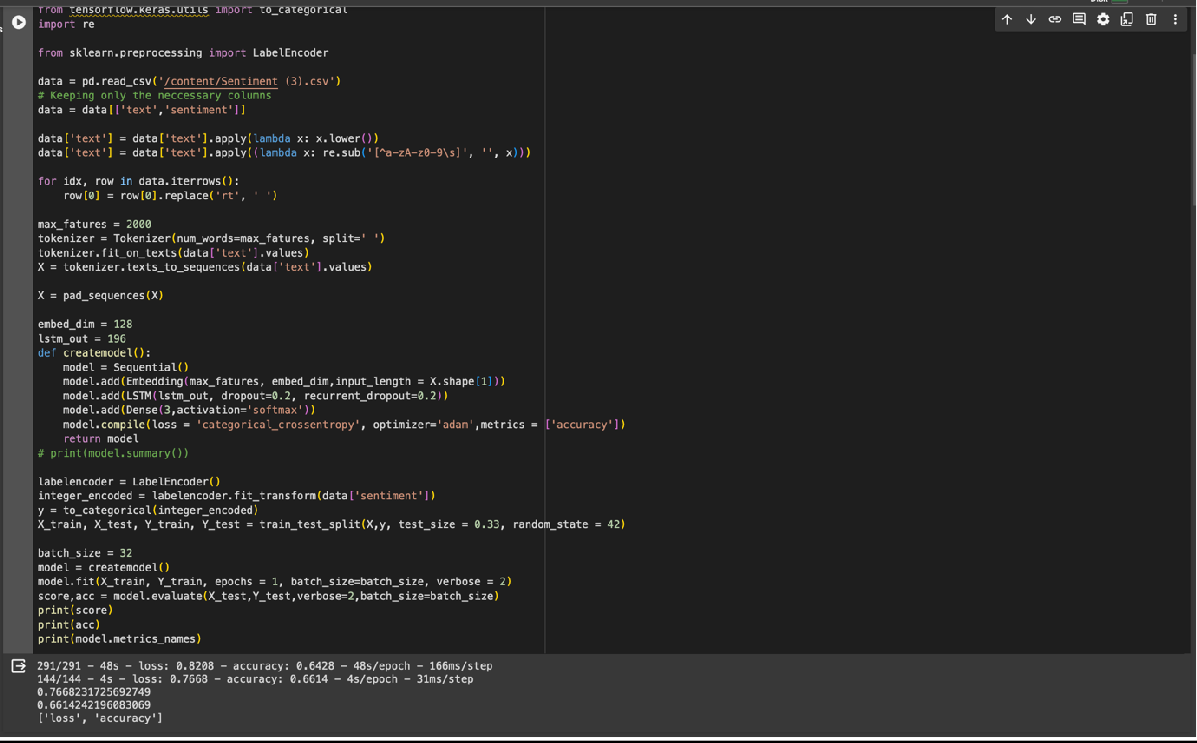
Bhavana Parasa 700761185

GitHub Link: https://github.com/BXP11850/Neural-Networks-Deep-Learning-Assignments-BXP11850

Video Link: https://github.com/BXP11850/Neural-Networks-Deep-Learning-Assignments-BXP11850/blob/main/Assignment%20-7/700761185.mp4

**Q1:**

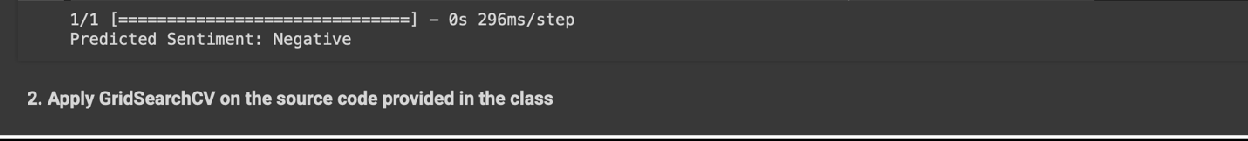
**Code:**



**Explanation:**

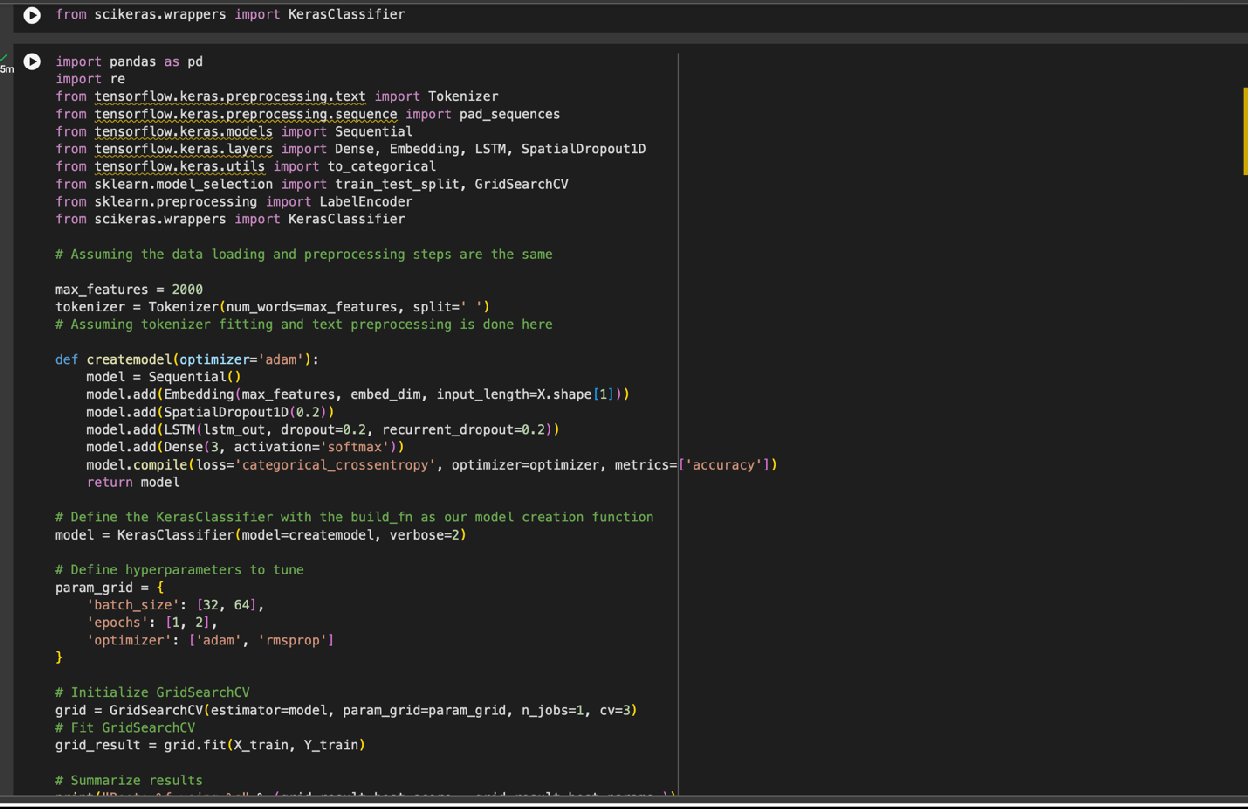
1. Import Libraries:It starts by importing necessary libraries. `tweepy` is used for accessing the Twitter API, `keras` is used for building and loading the neural network model, and `re` for regular expression operations.
2. Load Pre-trained Model:The pre-trained sentiment analysis model is loaded from a saved file (`sentiment\_model.h5`). This model is assumed to be trained to classify text into sentiments.
3. Preprocess Text:The `preprocess\_text` function is defined to clean the input text by converting it to lowercase and removing non-alphanumeric characters. This ensures the model receives the text in the format it expects.
4. Example Text:A sample tweet is provided as `new\_text`. This text is then preprocessed to remove unwanted characters and format it properly.
5. Tokenize and Pad the Text: The text is tokenized using Keras' `Tokenizer`, which converts the text into a sequence of integers where each integer represents a specific word in a dictionary. The sequence is then padded to ensure it has a fixed length, matching the model's input requirements.
6. Make Predictions:The preprocessed and formatted text is fed into the model to predict its sentiment. The model outputs a probability distribution across the possible sentiment classes (Negative, Neutral, Positive).
7. Determine Sentiment: The sentiment with the highest probability is selected as the predicted sentiment for the input text.

**Output:**



**Q2:**

**Code:**



**Explanation:**

1. Library Imports: It starts by importing necessary libraries. `pandas` for data manipulation, `re` for regular expressions, `tensorflow.keras` for building and training the neural network model, `sklearn.model\_selection` for splitting the dataset and conducting grid search, and `scikeras.wrappers` to wrap Keras models for use with scikit- learn.
2. Model Building Function:The `createmodel` function defines the architecture of the neural network using Keras' Sequential API. It includes an Embedding layer for text input, a SpatialDropout1D layer to reduce overfitting, an LSTM layer for learning from the sequence data, and a Dense output layer with a softmax activation function for classification. The optimizer for compiling the model can be adjusted, making the model flexible for hyperparameter tuning.
3. KerasClassifier Wrapper: A `KerasClassifier` wrapper is used to make the Keras model compatible with scikit- learn's grid search functionality. This allows the use of scikit-learn's `GridSearchCV` for hyperparameter tuning.
4. Hyperparameter Tuning:A parameter grid is defined with different values for batch size, number of epochs, and optimizer type. `GridSearchCV` is then used to exhaustively search through the parameter grid for the best model configuration based on cross-validation performance. It evaluates model performance for each combination of parameters across a specified number of folds of the training data.
5. Model Training and Selection: `grid.fit(X\_train, Y\_train)` trains the model using the training data across all combinations of parameters specified in `param\_grid`, using cross-validation. After fitting, it identifies the combination of parameters that resulted in the best model performance.
6. Results Summary: Finally, the best performance score and the hyperparameters that led to this best score are printed. This provides insights into which settings worked best for the given text classification task.

**Output:**

