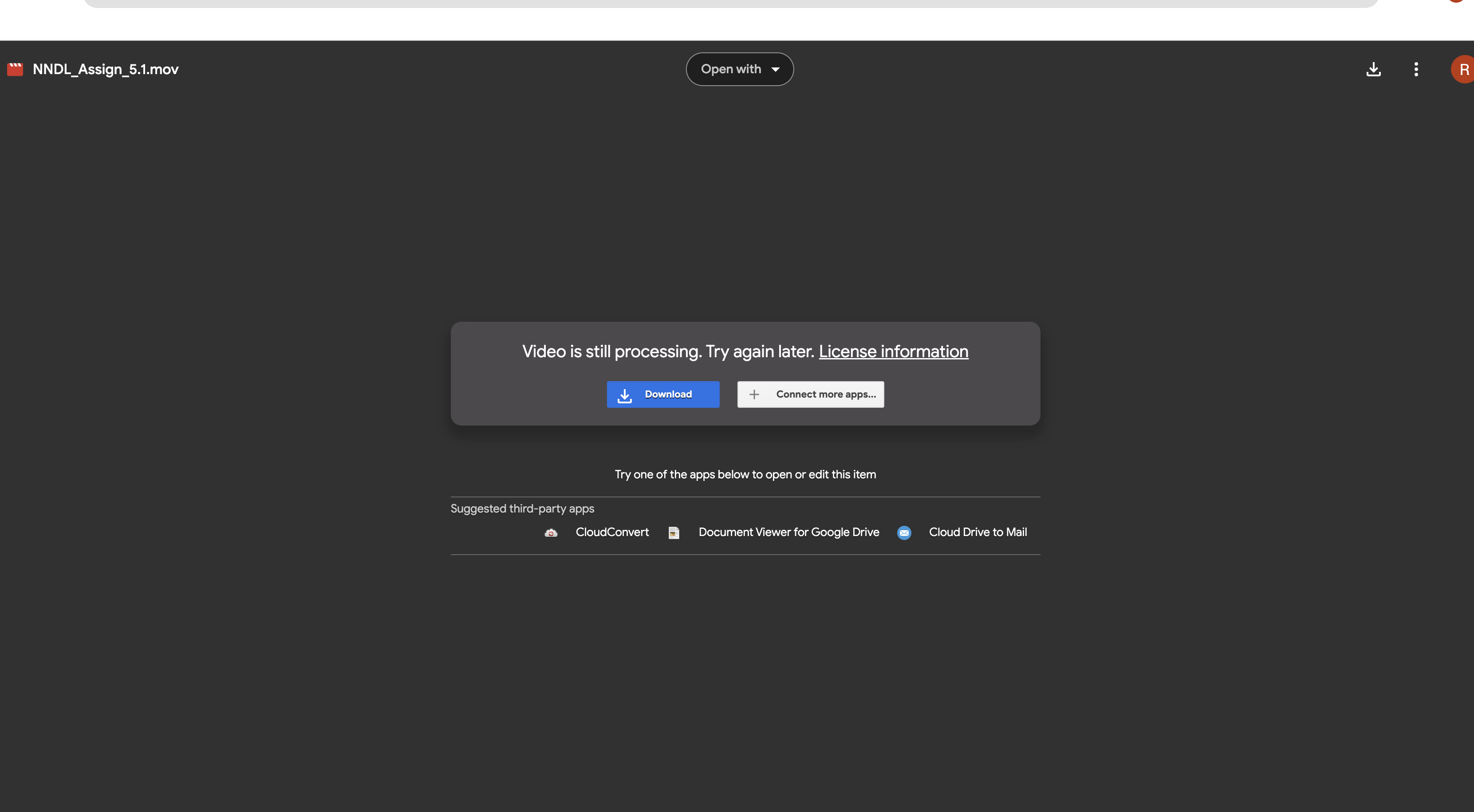
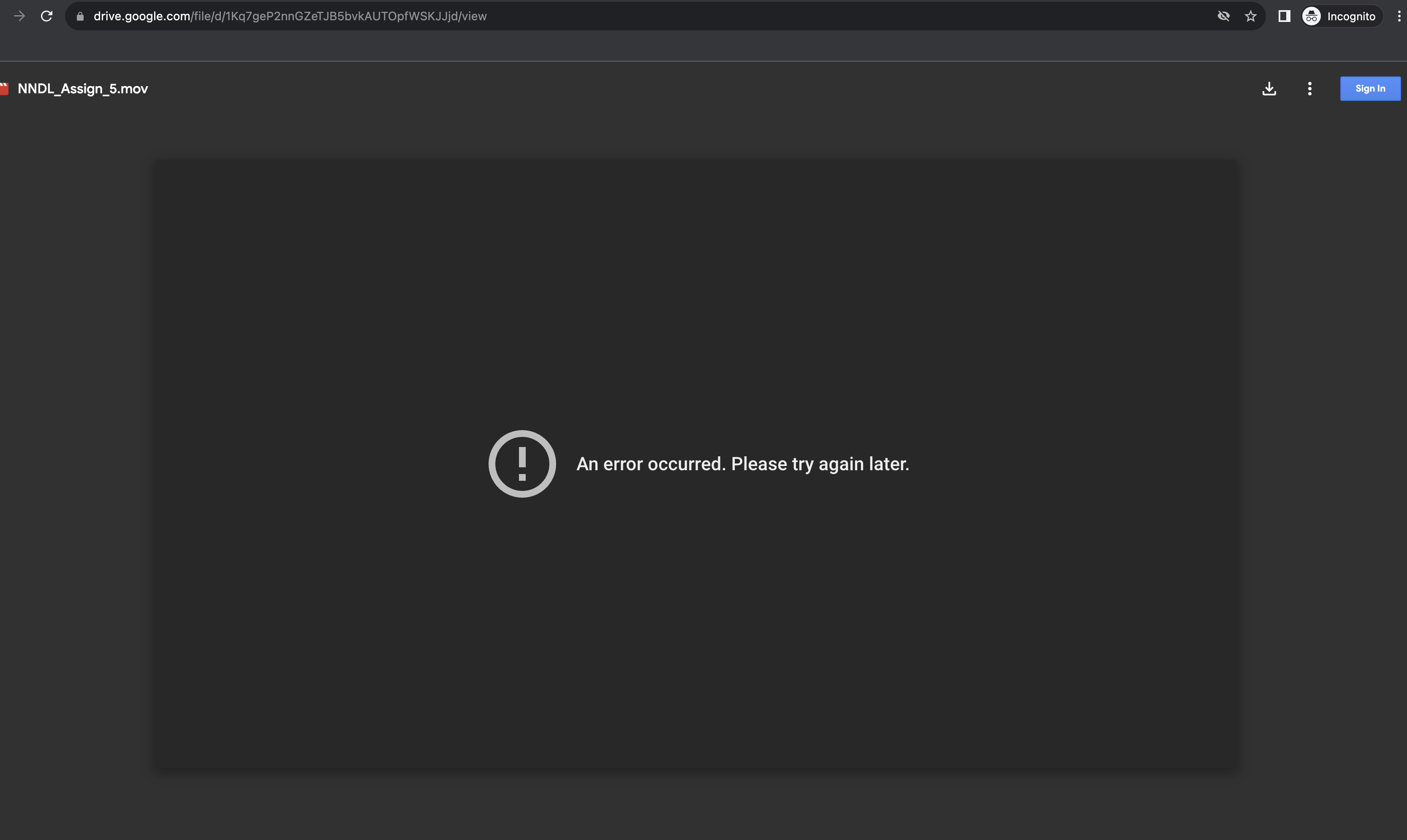
**Neural Networks and Deep Learning Assignment -5:**

**Id : 700742325**

**Name: Kunapareddy Jyothi Sai Rajesh**

**Crn : 31197**

**Note : Professor , I created 2 videos and tried multiple time but still video are not playing, But are getting Downloaded.Please consider downloading them ,sir.**

****

**Video Link 1:**

<https://drive.google.com/file/d/1CcjYCsA2mPXsx9qBtqSPa_kTZl33-qcw/view?usp=sharing>

**Video Link 2:**

<https://drive.google.com/file/d/1Kq7geP2nnGZeTJB5bvkAUTOpfWSKJJjd/view?usp=sharing>

**GitHub Link:**

<https://github.com/Rk-oo7/NNDL_Assign_5.git>

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](mailto:thing.@realDonaldTrump)”)**

A screenshot of a computer screen

Description automatically generated

1. **import pandas as pd**: This line is importing the pandas library as pd. Pandas is a popular Python library for data manipulation and analysis.
2. **import numpy as np**: This line is importing the numpy library as np. Numpy is a library in Python that is used for working with arrays.
3. **import matplotlib.pyplot as plt**: This line is importing the pyplot module from the matplotlib library as plt. Matplotlib is a plotting library, and pyplot is its easy-to-use module.
4. **import re**: This line is importing the 're' module, which stands for Regular Expressions. 're' is used for operations like searching, matching, finding, or replacing patterns in text.
5. **from sklearn.model\_selection import train\_test\_split**: This line is importing the function train\_test\_split from the module sklearn.model\_selection, which is used for splitting the data arrays into two subsets: for training data and for testing data.
6. **from sklearn.preprocessing import LabelEncoder**: This line is importing the LabelEncoder class from sklearn.preprocessing module. LabelEncoder is used to convert categorical data, or text data, into numbers, which our predictive model can better understand.
7. **from keras.preprocessing.text import Tokenizer**: This line is importing the Tokenizer class from the keras.preprocessing.text module. Tokenization is the process of taking text (like a sentence) and breaking it into individual terms (usually words).
8. **from tensorflow.keras.preprocessing.sequence import pad\_sequences**: This line is importing the pad\_sequences function from the tensorflow.keras.preprocessing.sequence module. pad\_sequences is used to ensure that all sequences in a list have the same length.
9. **from keras.models import Sequential**: This line is importing the Sequential model from keras.models. A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.
10. **from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D**: This line is importing several types of layers from keras.layers that can be added to a Sequential model: Dense (a regular densely-connected layer), Embedding (turns positive integers into dense vectors of fixed size), LSTM (Long Short-Term Memory layer), and SpatialDropout1D (Spatial 1D version of Dropout).
11. **from keras.utils.np\_utils import to\_categorical**: This line is importing the function to\_categorical from keras.utils.np\_utils. to\_categorical converts a class vector (integers) to binary class matrix.

A computer screen shot of text

Description automatically generated

1. Importing the pandas library under the alias pd.
2. Defining the path to a CSV file.
3. Reading the CSV file into a pandas DataFrame.
4. Creating a boolean mask by checking if any column names in the DataFrame are 'text' or 'sentiment'.
5. Applying the mask to the DataFrame, filtering it to only include the 'text' and 'sentiment' columns.

A screenshot of a computer code

Description automatically generated

1. **data['text'] = data['text'].apply(lambda x: x.lower())** - This line converts all the text in the 'text' column of the DataFrame 'data' to lowercase.
2. **data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))** - This line removes any characters that are not alphanumeric (letters and numbers) or whitespace from the 'text' column.
3. **for idx, row in data.iterrows():** - This line starts a loop that goes through each row in the DataFrame 'data'.
4. **row[0] = row[0].replace('rt', ' ')** - This line replaces any instances of 'rt' in the first column of the current row with a space, effectively removing Twitter retweet indicators.
5. **max\_fatures = 2000** - This line sets a variable named 'max\_fatures' to 2000, which will be used as the maximum number of words to consider when tokenizing the text.
6. **tokenizer = Tokenizer(num\_words=max\_fatures, split=' ')** - This line creates a Tokenizer object, which is used to convert text into sequences of integers, based on word frequency. The number of words to consider is defined by 'max\_fatures' and words are split by spaces.
7. **tokenizer.fit\_on\_texts(data['text'].values)** - This line fits the tokenizer on the texts in the 'text' column of the 'data' DataFrame, effectively learning the vocabulary of the text.
8. **X = tokenizer.texts\_to\_sequences(data['text'].values)** - This line uses the fitted tokenizer to convert the texts in the 'text' column of the 'data' DataFrame into sequences of integers, which can then be used for further analysis or model training.

A screenshot of a computer

Description automatically generated

1. **X = pad\_sequences(X)**: Padding the feature matrix **X** to ensure all sequences have the same length.
2. **embed\_dim = 128**: Setting the dimension of the embedded layer to 128.
3. **lstm\_out = 196**: Defining the number of neurons (units) in the LSTM layer.
4. **def createmodel():**: Function to create a sequential neural network model.
5. **model = Sequential()**: Creating a sequential neural network model.
6. **model.add(Embedding(max\_fatures, embed\_dim, input\_length=X.shape[1]))**: Adding an embedding layer with **max\_fatures** input neurons, **embed\_dim** output neurons, and input sequence length equal to the number of features in **X**.
7. **model.add(LSTM(lstm\_out, dropout=0.2, recurrent\_dropout=0.2))**: Adding an LSTM layer with **lstm\_out** neurons and dropout and recurrent dropout rates set to 0.2.
8. **model.add(Dense(3, activation='softmax'))**: Adding a fully connected dense layer with 3 output neurons (corresponding to positive, neutral, and negative classes) and using the softmax activation function.
9. **model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])**: Compiling the model with categorical cross-entropy loss, Adam optimizer, and accuracy as the evaluation metric.
10. **labelencoder = LabelEncoder()**: Creating a label encoder object.
11. **integer\_encoded = labelencoder.fit\_transform(data['sentiment'])**: Applying label encoding to the 'sentiment' column of the data.
12. **y = to\_categorical(integer\_encoded)**: Converting the integer-encoded labels to one-hot encoded categorical labels.
13. **X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)**: Splitting the data into training and testing sets with a test size of 33% and a fixed random seed of 42.
14. **batch\_size = 32**: Setting the batch size to 32 for training the model.
15. **model = createmodel()**: Creating the model using the defined **createmodel** function.
16. **model.fit(X\_train, Y\_train, epochs=1, batch\_size=batch\_size, verbose=2)**: Training the model for 1 epoch using the training data, with verbose set to 2 for intermediate progress messages.
17. **score, acc = model.evaluate(X\_test, Y\_test, verbose=2, batch\_size=batch\_size)**: Evaluating the model's performance on the test data and storing the loss and accuracy.
18. **print(score)**: Printing the loss (score) of the model on the test data.
19. **print(acc)**: Printing the accuracy of the model on the test data.
20. **print(model.metrics\_names)**: Printing the names of the metrics used to evaluate the model (in this case, ['loss', 'accuracy'])

A close-up of a computer screen

Description automatically generated

1. **model.save('sentimentAnalysis.h5')**: This line saves the current state of the machine learning model in a file named 'sentimentAnalysis.h5'.
2. **from keras.models import load\_model**: This line imports the **load\_model** function from the keras.models module, which is used to load previously saved models.
3. **model= load\_model('sentimentAnalysis.h5')**: This line uses the **load\_model** function to load the previously saved 'sentimentAnalysis.h5' model into the **model** variable.
4. **print(integer\_encoded)**: This line prints the content of the variable **integer\_encoded**, which likely contains some form of integer-encoded representation of your data.
5. **print(data['sentiment'])**: This line prints the 'sentiment' column of the **data** DataFrame, which presumably contains the sentiment labels for the data.

A screenshot of a computer

Description automatically generated

1. **from keras.wrappers.scikit\_learn import KerasClassifier**: This line imports the KerasClassifier wrapper from Keras, allowing the usage of scikit-learn's functionalities with Keras models.
2. **from sklearn.model\_selection import GridSearchCV**: This line imports GridSearchCV from scikit-learn, a tool that allows for exhaustive searching over specified parameter values for an estimator.
3. **model = KerasClassifier(build\_fn=createmodel,verbose=2)**: This line initializes a KerasClassifier object with a user-defined function **createmodel** as the model architecture to be trained, with verbosity level 2 (where the model outputs loss and accuracy for every epoch).
4. **batch\_size= [10, 20, 40]**: This line defines a list of batch sizes to be tested in the grid search.
5. **epochs = [1, 2]**: This line defines a list of epoch values (the number of times the learning algorithm will work through the entire training dataset) to be tested in the grid search.
6. **param\_grid= {'batch\_size':batch\_size, 'epochs':epochs}**: This line creates a dictionary where the keys are the names of parameters to optimize and the values are the lists of values for those parameters.
7. **grid = GridSearchCV(estimator=model, param\_grid=param\_grid)**: This line initializes a GridSearchCV object with the model and the parameter grid to use for the search.
8. **grid\_result= grid.fit(X\_train,Y\_train)**: This line fits the model with the training data, which will conduct the grid search over the parameter grid, testing all combinations of batch sizes and epochs defined earlier.
9. **print("Best: %f using %s" % (grid\_result.best\_score\_, grid\_result.best\_params\_))**: This line prints the highest validation score achieved during the grid search, and the parameters that yielded that score.