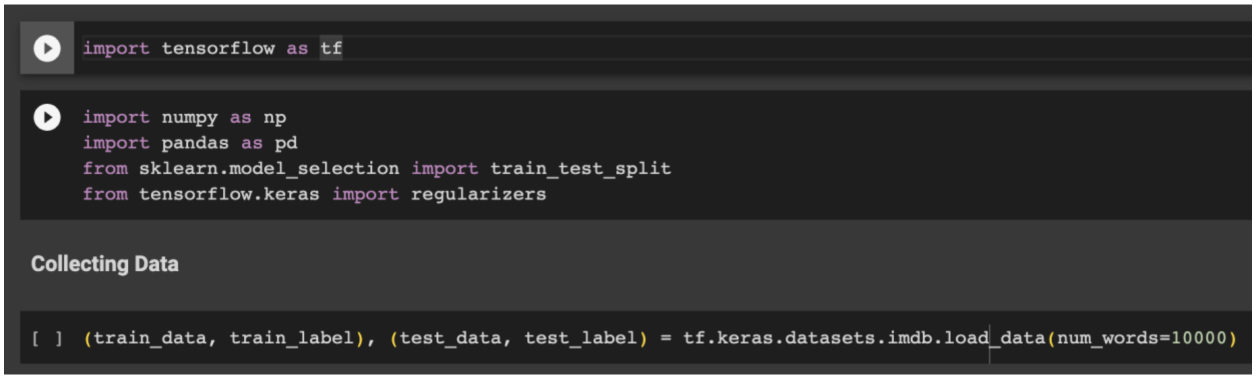
Sentiment Detection Model

Description:

The goal of this project was to build a machine learning model than can predict the sentiment analysis of a movie review to be either labelled negative or positive. There were a few requirements that had to be met: the training accuracy of the model should not be less than 87% and difference between the training and test accuracy should not exceed 2.5%. Also, only 10,000 words was allowed to be used for the model's input.

Since, the output label is the sentiment value which can either be positive or negative, this machine learning problem is called classification. This is because the output can only be the two different values meaning it is a discrete output and not continuous. To solve this classification problem, I chose to implement deep learning neutral networks. The main reason I chose to go with deep learning over traditional classification is because NN allows the model to extract its own features, capturing many complex relationships within the movie reviews, and ultimately enhancing the task of sentiment analysis.

The first step that was done was reading in the data which was pulled from "tf.keras.datasets". The dataset was the "IMDB dataset" which consisted of 50,000 movie reviews. When loaded the data, the data was split into two sets: 25,000 reviews for the training set and 25,000 for the testing set. Each set was divided into two sections: one being data and other being label. Additionally, there was a word limit used (num_words parameter was set to 10,000) when pulling the data to meet the requirement of this project of only using 10,000 for the model's input.



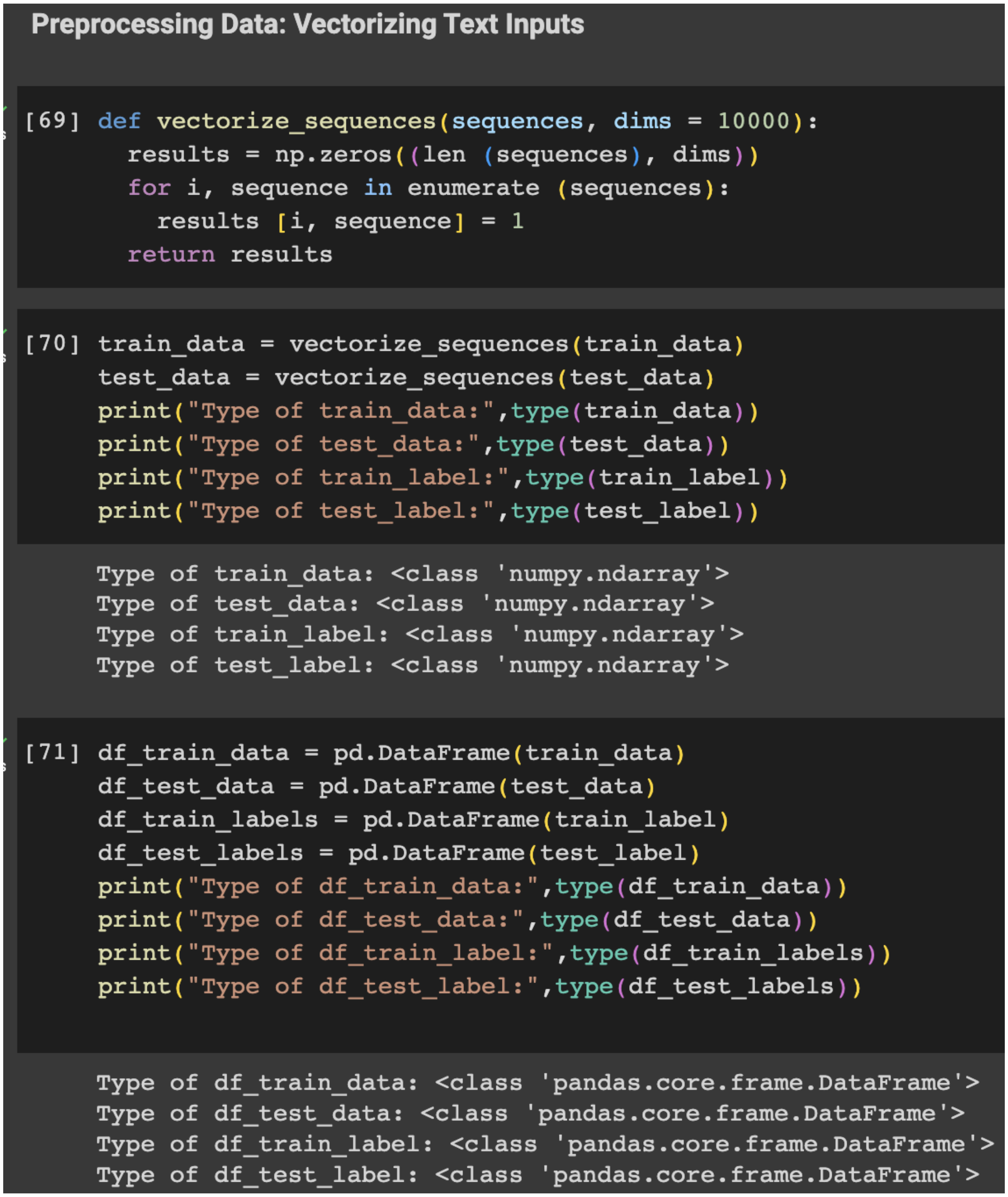
```
import tensorflow as tf
```

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras import regularizers
```

Collecting Data

```
[ ] (train_data, train_label), (test_data, test_label) = tf.keras.datasets.imdb.load_data(num_words=10000)
```

The next step was the preprocessing step, getting the data ready for deep learning. To use the train and test data in the NN model, I first had to vectorize the text inputs. This was achieved using the "vectorize_sequences" function. I also then checked the type of the data and labels, it was of numpy.ndarray. This was an issue for the model, as it kept giving errors when feeding in a ndarray. So, I then converted the data and labels to the pandas DataFrame data structure to fix this error.


Preprocessing Data: Vectorizing Text Inputs

```
[69] def vectorize_sequences(sequences, dims = 10000):
       results = np.zeros((len (sequences), dims))
       for i, sequence in enumerate (sequences):
         results [i, sequence] = 1
       return results
```

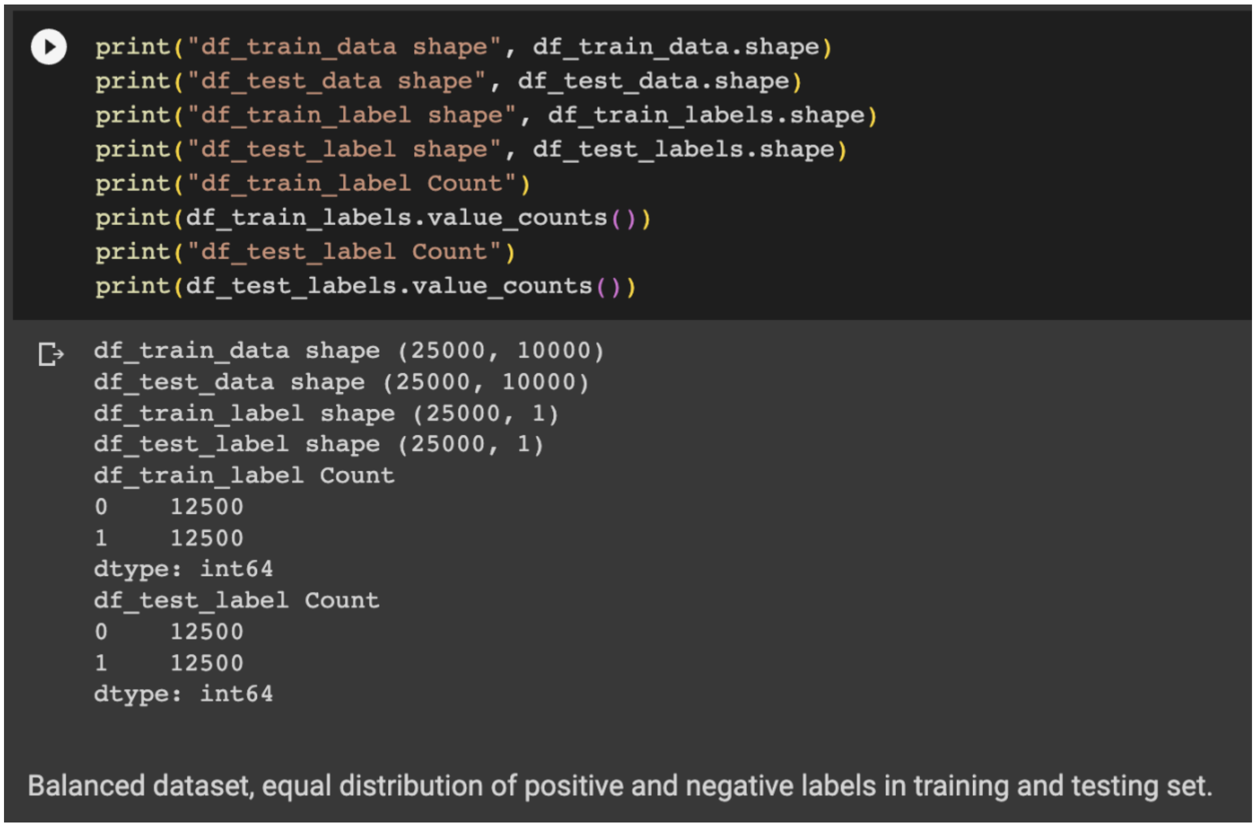
```
[70] train_data = vectorize_sequences(train_data)
     test_data = vectorize_sequences(test_data)
     print("Type of train_data:",type(train_data))
     print("Type of test_data:",type(test_data))
     print("Type of train_label:",type(train_label))
     print("Type of test_label:",type(test_label))
```

```
     Type of train_data: <class 'numpy.ndarray'>
     Type of test_data: <class 'numpy.ndarray'>
     Type of train_label: <class 'numpy.ndarray'>
     Type of test_label: <class 'numpy.ndarray'>
```

```
[71] df_train_data = pd.DataFrame(train_data)
     df_test_data = pd.DataFrame(test_data)
     df_train_labels = pd.DataFrame(train_label)
     df_test_labels = pd.DataFrame(test_label)
     print("Type of df_train_data:",type(df_train_data))
     print("Type of df_test_data:",type(df_test_data))
     print("Type of df_train_label:",type(df_train_labels))
     print("Type of df_test_label:",type(df_test_labels))
```

```
     Type of df_train_data: <class 'pandas.core.frame.DataFrame'>
     Type of df_test_data: <class 'pandas.core.frame.DataFrame'>
     Type of df_train_label: <class 'pandas.core.frame.DataFrame'>
     Type of df_test_label: <class 'pandas.core.frame.DataFrame'>
```

I checked whether this dataset was balanced as I was later using the accuracy metric for testing the performance of the model with the training and testing sets. For the accuracy to be a good choice of metric, the dataset should be balanced. This dataset proved to be balanced as shown below. The number of positive and negative labels within the training and testing label sets are equal.



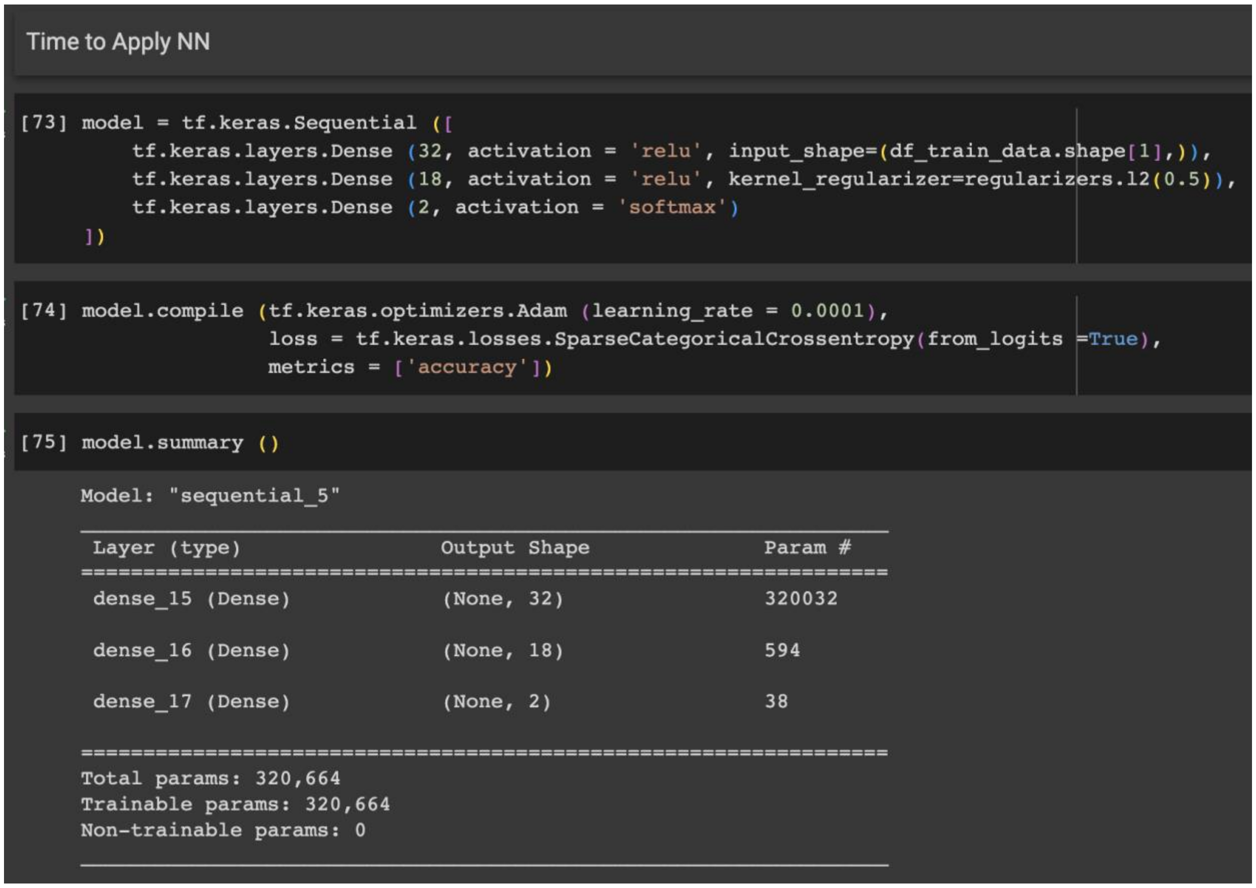
```
print("df_train_data shape", df_train_data.shape)
print("df_test_data shape", df_test_data.shape)
print("df_train_label shape", df_train_labels.shape)
print("df_test_label shape", df_test_labels.shape)
print("df_train_label Count")
print(df_train_labels.value_counts())
print("df_test_label Count")
print(df_test_labels.value_counts())
```

```
df_train_data shape (25000, 10000)
df_test_data shape (25000, 10000)
df_train_label shape (25000, 1)
df_test_label shape (25000, 1)
df_train_label Count
0    12500
1    12500
dtype: int64
df_test_label Count
0    12500
1    12500
dtype: int64
```

Balanced dataset, equal distribution of positive and negative labels in training and testing set.

Now, that the data is preprocessed. It is time to build the model. Selecting the architecture for the neutral network such as manipulating the number of layers or neurons impact how the NN learns, extract its own features, and therefore affects the performance of the model. I had initially chosen to incorporate 5 layers in my NN architecture with various number of neurons which starting from 256 and then reduced from layer to layer. This achieved a high training score but poor testing score which meant the model was very overfitted. To deal with this overfitted problem, I kept manipulating the NN architecture, reducing the number of layers and the number of neutrons in each layer to reduce the difference between the training and testing scores. Eventually, I got to the number of layers and neutrons in the below photo, this architecture helped me achieve a training score higher than 87% but I still had a difference greater than 2.5% between my training and testing score. So, I decided to use L2 regularization to help reduce the overfitting even more. This final architecture resulted in my training accuracy score being 90% and testing accuracy score being 89%.

Therefore, all requirements were met. I used 10000 words for my model input. The final model training score was above 87%. The difference between my final model's training and testing score was less than 2.5%.


Time to Apply NN

```
[73] model = tf.keras.Sequential ([
         tf.keras.layers.Dense (32, activation = 'relu', input_shape=(df_train_data.shape[1],)),
         tf.keras.layers.Dense (18, activation = 'relu', kernel_regularizer=regularizers.l2(0.5)),
         tf.keras.layers.Dense (2, activation = 'softmax')
     ])
```

```
[74] model.compile (tf.keras.optimizers.Adam (learning_rate = 0.0001),
                    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits =True),
                    metrics = ['accuracy'])
```

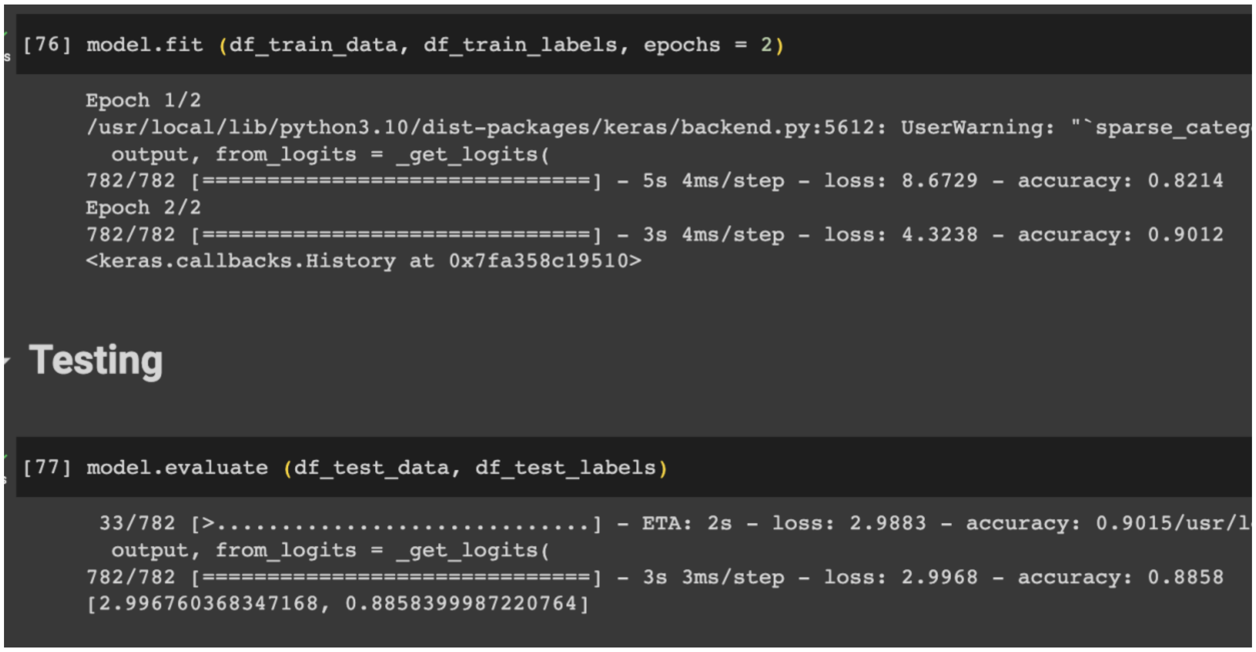
```
[75] model.summary ()

     Model: "sequential_5"
     _________________________________________________________________
      Layer (type)                Output Shape              Param #
     =================================================================
      dense_15 (Dense)            (None, 32)                320032

      dense_16 (Dense)            (None, 18)                594

      dense_17 (Dense)            (None, 2)                 38

     =================================================================
     Total params: 320,664
     Trainable params: 320,664
     Non-trainable params: 0
     _________________________________________________________________
```



```
[76] model.fit (df_train_data, df_train_labels, epochs = 2)

     Epoch 1/2
     /usr/local/lib/python3.10/dist-packages/keras/backend.py:5612: UserWarning: "`sparse_categ
       output, from_logits = _get_logits(
     782/782 [==============================] - 5s 4ms/step - loss: 8.6729 - accuracy: 0.8214
     Epoch 2/2
     782/782 [==============================] - 3s 4ms/step - loss: 4.3238 - accuracy: 0.9012
     <keras.callbacks.History at 0x7fa358c19510>
```

Testing

```
[77] model.evaluate (df_test_data, df_test_labels)

      33/782 [>.............................] - ETA: 2s - loss: 2.9883 - accuracy: 0.9015/usr/l
       output, from_logits = _get_logits(
     782/782 [==============================] - 3s 3ms/step - loss: 2.9968 - accuracy: 0.8858
     [2.996760368347168, 0.8858399987220764]
```