Using RNNs to classify sentiment on IMDB data

In this assignment,you will train three types of RNNs: "vanilla" RNN, LSTM and GRU to predict the sentiment on IMDB reviews.

Keras provides a convenient interface to load the data and immediately encode the words into integers (based on the most common words). This will save you a lot of the drudgery that is usually involved when working with raw text.

The IMDB is data consists of 25000 training sequences and 25000 test sequences. The outcome is binary (positive/negative) and both outcomes are equally represented in both the training and the test set.

Walk through the followinng steps to prepare the data and the building of an RNN model.

In [3]:
```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense,Embedding,SimpleRNN
from tensorflow.keras.initializers import RandomNormal,glorot_uniform
from tensorflow.keras.losses import binary_crossentropy
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import GRU
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.callbacks import TensorBoard
from sklearn.model_selection import KFold
import numpy as np
import matplotlib.pyplot as plt
import time
```

In [3]:
```
print("Num GPUs Available: ", len(tf.config.experimental.list_physical_devic
```

```
Num GPUs Available:  0
```

1- Use the `imdb.load_data()` to load in the data

2- Specify the maximum length of a sequence to 30 words and the pick the 2000 most common words.

Loading the IMDB movie review dataset using the Keras library's inbuilt function 'load_data'. The function returns two tuples: one with training data and the other with test data. The 'num_words' parameter is used to specify the maximum number of words to keep based on word frequency, with the most frequent words being kept.

In [4]:
```
#intialising max length of sequence of words
max_len = 30
#most common words to pick
mca = 2000
(x_train,y_train),(x_test,y_test) = imdb.load_data(num_words = mca)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 2s 0us/step
```

This code is printing the number of sequences in the training and test datasets.

x_train.shape[0] returns the number of rows (i.e. number of sequences) in the x_train dataset. x_test.shape[0] returns the number of rows (i.e. number of sequences) in the x_test dataset.

In [5]:
```
print('x_train = '+str(x_train.shape[0])+" train sequences")
print('x_test = '+str(x_test.shape[0])+" test sequences")
```

```
x_train = 25000 train sequences
x_test = 25000 test sequences
```

3- Check that the number of sequences in train and test datasets are equal (default split):

Expected output:

- `x_train = 25000 train sequences`
- `x_test = 25000 test sequences`

4- Pad (or truncate) the sequences so that they are of the maximum length

This code is using the pad_sequences() function from Keras to pad the sequences of words in x_train and x_test to a fixed length of max_len. The pad_sequences() function is used to ensure that all sequences in a list have the same length, which is required when feeding data into a neural network. If a sequence is shorter than the specified maxlen, it is padded with zeros at the end, and if it is longer, it is truncated.

In [5]:
```
from tensorflow.keras.preprocessing import sequence
x_train = sequence.pad_sequences(x_train,maxlen = max_len)
x_test = sequence.pad_sequences(x_test,maxlen = max_len)
```

In [7]:
```
print("x_train shape: ",x_train.shape)
print("x_test shape: ",x_test.shape)
```

```
x_train shape:  (25000, 30)
x_test shape:  (25000, 30)
```

5- After padding or truncating, check the dimensionality of x_train and x_test.

Expected output:

- `x_train shape: (25000, 30)`
- `x_test shape: (25000, 30)`

6- For all your models:

- Use tensorboard to run your experiments

- Use cross valdiation with 10 folds

7- Build the RNN with three layers:

- The SimpleRNN layer with 5 neurons and initialize its kernel with stddev=0.001
- The Embedding layer and initialize it by setting the word embedding dimension to 50. This means that this layer takes each integer in the sequence and embeds it in a 50-dimensional vector.
- The output layer has the sigmoid activation function.

8- How many parameters have the embedding layer?

9- Train the network with the RMSprop with learning rate of .0001 and epochs=10.

10- PLot the loss and accuracy metrics during the training and interpret the result.

11- Check the accuracy and the loss of your models on the test dataset.

Question No 6 to 11 will be answered by using the below function Seq_model

This function defines a sequential model to perform sentiment analysis on the IMDB movie reviews dataset. It takes the following parameters:

input1_dim: the number of most frequent words to keep in the dataset input1_length: the maximum length of each sequence of words type1: the type of recurrent neural network to use (GRU, LSTM or SimpleRNN) input_units: the number of units in the RNN layer stdvalue: the standard deviation of the normal distribution used for weight initialization in the RNN layer lr: the learning rate of the optimizer used to train the model The function first loads the IMDB dataset and pads the sequences of words in the training and testing datasets to a fixed length using the pad_sequences function from Keras. It then creates a sequential model using the Sequential class from Keras and adds an embedding layer and an RNN layer to the model, depending on the value of type1. It then compiles the model with the RMSprop optimizer, binary crossentropy loss function, and accuracy metric.

The function uses K-Fold cross-validation with 10 splits to train and evaluate the model. It trains the model for 10 epochs on each split of the dataset and records the loss and accuracy metrics for both training and validation sets using the fit method. It also uses the TensorBoard callback to monitor the training progress.

After training the model on all splits of the dataset, the function plots the training and validation loss and accuracy curves for the entire training process using Matplotlib. It also evaluates the model on the testing dataset using the evaluate method and prints the test loss and accuracy.

Finally, the function returns the time taken to run the model and the test loss and accuracy.

In [2]:
```
def Seq_model(input1_dim,input1_length,type1,input_units,stdvalue,lr):
  (x_train,y_train),(x_test,y_test) = imdb.load_data(num_words = input1_dim)
  x_train = sequence.pad_sequences(x_train,maxlen = input1_length)
  x_test = sequence.pad_sequences(x_test,maxlen = input1_length)
  keras.backend.clear_session()
  model = Sequential()
  model.add(Embedding(input_dim = input1_dim,output_dim = 50, input_length =
  if(type1 == 'GRU'):
      model.add(GRU(input_units,kernel_initializer = RandomNormal(stddev=stc
  elif(type1 == 'LSTM'):
      model.add(LSTM(input_units,kernel_initializer = RandomNormal(stddev=st
  else:
      model.add(SimpleRNN(input_units,kernel_initializer = RandomNormal(stdc
```

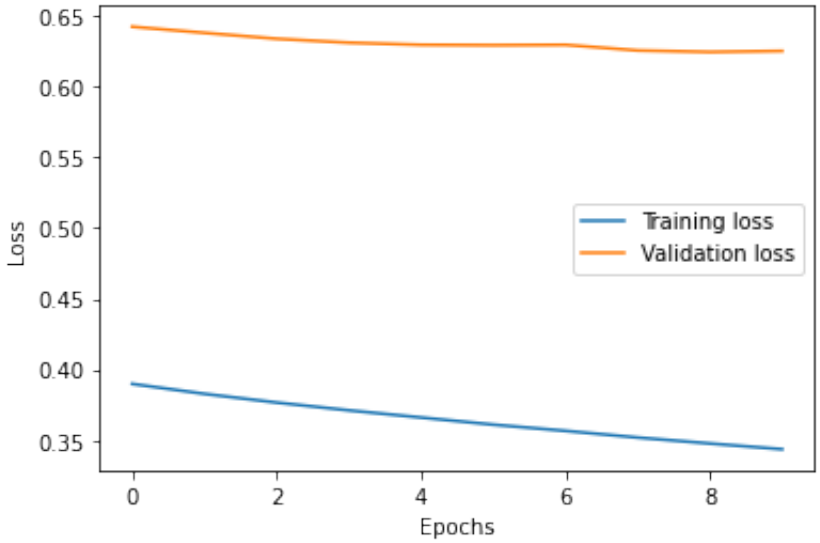
```
model.add(Dense(1,activation='sigmoid'))
%load_ext tensorboard
%tensorboard --logdir RNN_logs
tensorflow_callbacks = TensorBoard(log_dir = f"logs/{time.time()}")
print("Model Summary: ",model.summary())
Kfold = KFold(n_splits = 10,shuffle=True,random_state=42)
start_time = time.time()
f = 0
for train,test in Kfold.split(x_train,y_train):
    f +=1
    print(f"Running on fold : {f}")
    model.compile(optimizer = RMSprop(learning_rate=lr),
                        loss = binary_crossentropy,
                        metrics = ['accuracy'])
    a = model.fit(x_train[train],y_train[train],
                        epochs=10,
                        validation_data = (x_test[test],y_test[test]),
                        callbacks = [tensorflow_callbacks],batch_size=64)
    print(f"Fold {f} Completed")
time_taken = time.time() - start_time
loss = a.history['loss']
accuracy = a.history['accuracy']
val_loss = a.history['val_loss']
val_accuracy = a.history['val_accuracy']
plt.plot(loss,label="Training loss")
plt.plot(val_loss,label='Validation loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend()
plt.show()
plt.plot(accuracy,label='Training Accuracy')
plt.plot(val_accuracy,label='Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend()
plt.show
print(f"Time taken to run the model: {time_taken}")
test_loss,test_accuracy = model.evaluate(x_test,y_test)
print(f'Test Loss for this model is : {test_loss}')
print(f'Test Accuracy for this model is : {test_accuracy}')
```

This code is calling the Seq_model function with specific parameters to train and evaluate a Vanilla RNN network on the IMDB movie review dataset with sequence length of 30. It then prints some descriptive text to indicate that the tuning process for this network is starting.

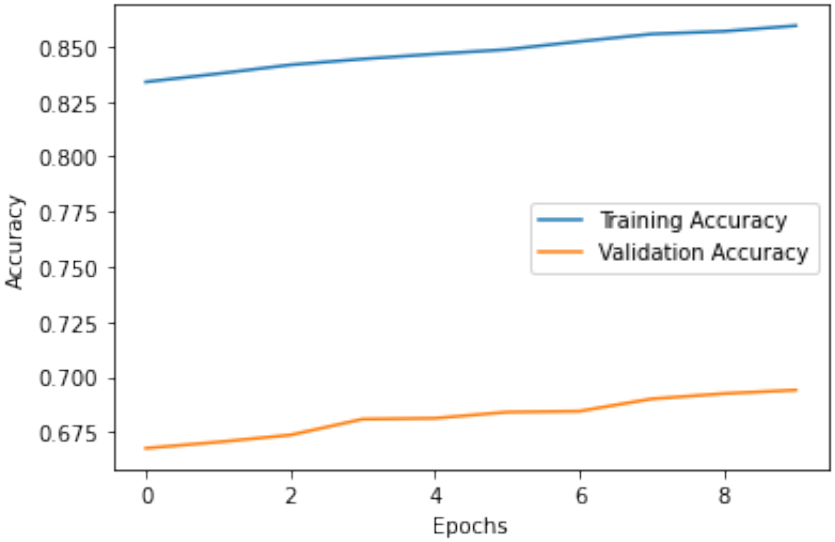
In [52]:
```
base_model = Seq_model(mca,max_len,'',5,0.001,0.0001)
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 30, 50)            100000

 simple_rnn (SimpleRNN)      (None, 5)                 280

 dense (Dense)               (None, 1)                 6

=================================================================
Total params: 100,286
Trainable params: 100,286
Non-trainable params: 0
_________________________________________________________________
Model Summary:  None
Running on fold : 1
Epoch 1/10
352/352 [==============================] - 7s 16ms/step - loss: 0.6931 - ac
curacy: 0.5044 - val_loss: 0.6929 - val_accuracy: 0.5132
Epoch 2/10
352/352 [==============================] - 5s 13ms/step - loss: 0.6927 - ac
curacy: 0.5240 - val_loss: 0.6925 - val_accuracy: 0.5372
Epoch 3/10
352/352 [==============================] - 4s 12ms/step - loss: 0.6921 - ac
curacy: 0.5308 - val_loss: 0.6919 - val_accuracy: 0.5348
Epoch 4/10
352/352 [==============================] - 5s 15ms/step - loss: 0.6909 - ac
curacy: 0.5394 - val_loss: 0.6908 - val_accuracy: 0.5348
Epoch 5/10
352/352 [==============================] - 4s 12ms/step - loss: 0.6886 - ac
curacy: 0.5445 - val_loss: 0.6895 - val_accuracy: 0.5432
Epoch 6/10
352/352 [==============================] - 4s 12ms/step - loss: 0.6846 - ac
curacy: 0.5612 - val_loss: 0.6888 - val_accuracy: 0.5424
Epoch 7/10
352/352 [==============================] - 5s 15ms/step - loss: 0.6786 - ac
curacy: 0.5778 - val_loss: 0.6897 - val_accuracy: 0.5380
Epoch 8/10
352/352 [==============================] - 5s 14ms/step - loss: 0.6709 - ac
curacy: 0.5948 - val_loss: 0.6927 - val_accuracy: 0.5300
Epoch 9/10
352/352 [==============================] - 7s 19ms/step - loss: 0.6622 - ac
curacy: 0.6088 - val_loss: 0.6971 - val_accuracy: 0.5252
Epoch 10/10
352/352 [==============================] - 5s 15ms/step - loss: 0.6538 - ac
curacy: 0.6212 - val_loss: 0.7015 - val_accuracy: 0.5292
```

```
Epoch 3/10
352/352 [==============================] - 5s 15ms/step - loss: 0.4818 - ac
curacy: 0.7778 - val_loss: 0.7364 - val_accuracy: 0.5928
Epoch 4/10
352/352 [==============================] - 4s 12ms/step - loss: 0.4757 - ac
curacy: 0.7806 - val_loss: 0.7312 - val_accuracy: 0.5980
Epoch 5/10
352/352 [==============================] - 4s 12ms/step - loss: 0.4700 - ac
curacy: 0.7844 - val_loss: 0.7253 - val_accuracy: 0.6056
Epoch 6/10
352/352 [==============================] - 5s 14ms/step - loss: 0.4640 - ac
curacy: 0.7891 - val_loss: 0.7225 - val_accuracy: 0.6100
Epoch 7/10
352/352 [==============================] - 4s 12ms/step - loss: 0.4585 - ac
curacy: 0.7913 - val_loss: 0.7187 - val_accuracy: 0.6116
Epoch 8/10
352/352 [==============================] - 4s 12ms/step - loss: 0.4528 - ac
curacy: 0.7948 - val_loss: 0.7118 - val_accuracy: 0.6140
Epoch 9/10
352/352 [==============================] - 5s 15ms/step - loss: 0.4469 - ac
curacy: 0.8002 - val_loss: 0.7081 - val_accuracy: 0.6172
Epoch 10/10
352/352 [==============================] - 4s 12ms/step - loss: 0.4410 - ac
curacy: 0.8026 - val_loss: 0.7030 - val_accuracy: 0.6224
Fold 8 Completed
Running on fold : 9
Epoch 1/10
352/352 [==============================] - 5s 11ms/step - loss: 0.4447 - ac
curacy: 0.8004 - val_loss: 0.6763 - val_accuracy: 0.6180
Epoch 2/10
352/352 [==============================] - 5s 13ms/step - loss: 0.4364 - ac
curacy: 0.8059 - val_loss: 0.6700 - val_accuracy: 0.6276
Epoch 3/10
352/352 [==============================] - 4s 10ms/step - loss: 0.4290 - ac
curacy: 0.8116 - val_loss: 0.6650 - val_accuracy: 0.6328
Epoch 4/10
352/352 [==============================] - 3s 9ms/step - loss: 0.4222 - acc
uracy: 0.8152 - val_loss: 0.6596 - val_accuracy: 0.6380
Epoch 5/10
352/352 [==============================] - 4s 11ms/step - loss: 0.4153 - ac
curacy: 0.8199 - val_loss: 0.6532 - val_accuracy: 0.6448
Epoch 6/10
352/352 [==============================] - 4s 11ms/step - loss: 0.4088 - ac
curacy: 0.8243 - val_loss: 0.6471 - val_accuracy: 0.6492
Epoch 7/10
352/352 [==============================] - 4s 10ms/step - loss: 0.4028 - ac
curacy: 0.8283 - val_loss: 0.6450 - val_accuracy: 0.6572
Epoch 8/10
352/352 [==============================] - 4s 11ms/step - loss: 0.3964 - ac
curacy: 0.8330 - val_loss: 0.6397 - val_accuracy: 0.6584
```




```
Time taken to run the model: 565.3596515655518
782/782 [==============================] - 3s 4ms/step - loss: 0.6178 - acc
uracy: 0.6948
Test Loss for this model is : 0.6177656054496765
Test Accuracy for this model is : 0.6947600245475769
```



1. **Total time taken by the RNN model with features 2000 and max_length 30 is 1256.07secs.**
2. **Using this model we go the Test accuracy of 0.6947 and Test Loss of 0.6177.**

From the above plot we can say that training accuracy has reached 85.9% and validation accuracy has reached 69.4%.

Also, From the above plottings we can say that training loss is 34.37% and validation loss is 62.53%.

Tuning The Vanilla RNN Network

12- Prepare the data to use sequences of length 80 rather than length 30 and retrain your model. Did it improve the performance?

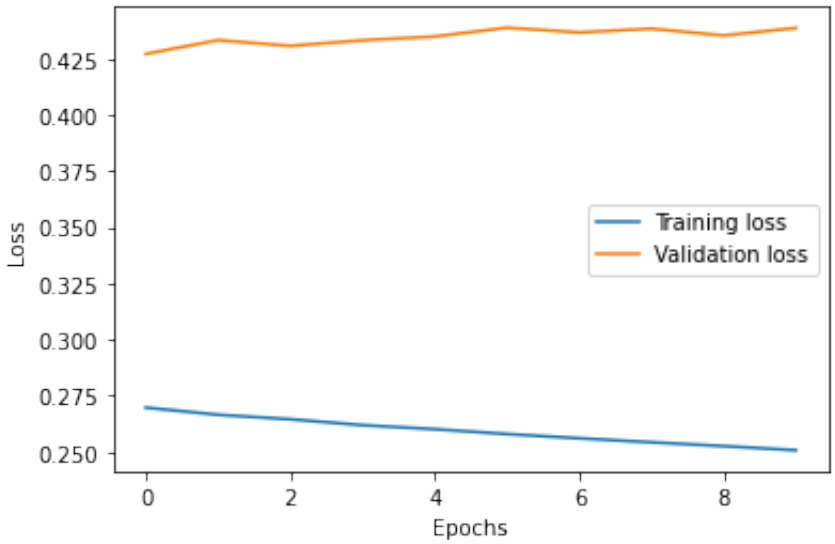
This code is calling the Seq_model function with specific parameters to train and evaluate a Vanilla RNN network on the IMDB movie review dataset with sequence length of 80. It then prints some descriptive text to indicate that the tuning process for this network is starting.

1. **Total time taken by the RNN model with features 2000 and max_length 80 is 1256.07secs.**
2. **Using this model we go the Test accuracy of 0.8088 and Test Loss of 0.4491.**

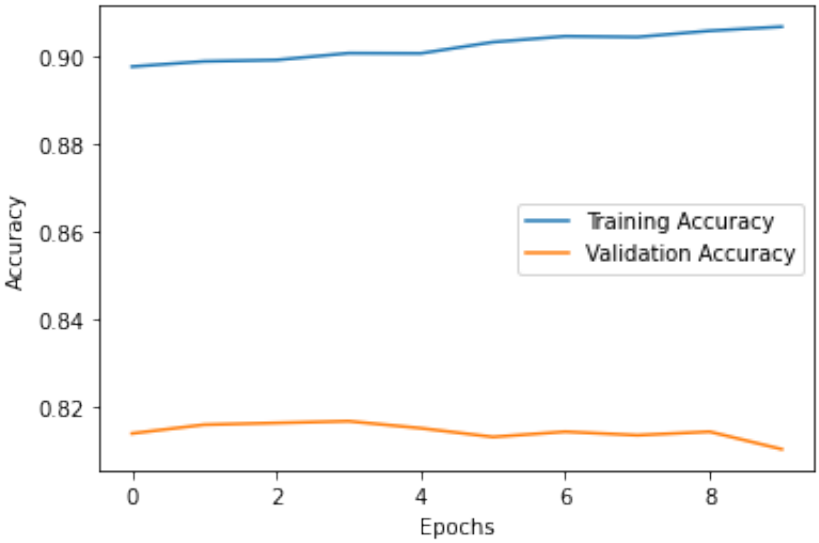
In [55]:
```
print("Tuning Vanilla RNN Network with length 80")
print("--" * 20)
tune1_length = Seq_model(mca,80,'',5,0.001,0.0001)
```

```
Tuning Vanilla RNN Network with length 80
----------------------------------------
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 80, 50)            100000

 simple_rnn (SimpleRNN)      (None, 5)                 280

 dense (Dense)               (None, 1)                 6

=================================================================
Total params: 100,286
Trainable params: 100,286
Non-trainable params: 0
_________________________________________________________________
Model Summary:  None
Running on fold : 1
Epoch 1/10
352/352 [==============================] - 12s 30ms/step - loss: 0.6921 - a
ccuracy: 0.5321 - val_loss: 0.6877 - val_accuracy: 0.5952
Epoch 2/10
352/352 [==============================] - 10s 28ms/step - loss: 0.6832 - a
```

```
ccuracy: 0.9034 - val_loss: 0.4389 - val_accuracy: 0.8132
Epoch 7/10
352/352 [==============================] - 10s 28ms/step - loss: 0.2560 - a
ccuracy: 0.9047 - val_loss: 0.4369 - val_accuracy: 0.8144
Epoch 8/10
352/352 [==============================] - 13s 36ms/step - loss: 0.2543 - a
ccuracy: 0.9045 - val_loss: 0.4385 - val_accuracy: 0.8136
Epoch 9/10
352/352 [==============================] - 12s 35ms/step - loss: 0.2526 - a
ccuracy: 0.9060 - val_loss: 0.4355 - val_accuracy: 0.8144
Epoch 10/10
352/352 [==============================] - 11s 30ms/step - loss: 0.2507 - a
ccuracy: 0.9069 - val_loss: 0.4388 - val_accuracy: 0.8104
Fold 10 Completed
```




```
Time taken to run the model: 1256.0733649730682
782/782 [==============================] - 6s 8ms/step - loss: 0.4492 - acc
uracy: 0.8088
Test Loss for this model is : 0.4491528570652008
Test Accuracy for this model is : 0.8088399767875671
```



Yes, model got improved. Results are given below.

Model summary: This provides a summary of the layers in the model architecture including the input and output dimensions of each layer.

Training and Validation Loss Plots: These plots show the variation in training and validation loss over the epochs of training.

T**raining and Validation Accuracy Plots**: These plots show the variation in training and validation accuracy over the epochs of training.

Time taken to run the model: the total time taken to run the model is 1256.07secs

Test Loss: This is the loss value obtained when evaluating the model on the test set. In this case, the test loss is 0.4491528570652008.

Test Accuracy: This is the accuracy obtained when evaluating the model on the test set. In this case, the test accuracy is 0.8088399767875671 or 80.88%.

In [ ]:

In [ ]:

14- Try different values of the maximum length of a sequence ("max_features"). Can you improve the performance?

This code section is tuning a Vanilla RNN network with 2000 max features and 120 max length.

mca: the maximum number of most common words to keep. Here it is set to 2000.

99: the maximum length of all sequences to pad.

'': an empty string is passed as the type1 parameter, which implies that the function will create a Vanilla RNN network.

5: the number of units in the RNN layer.

0.001: the standard deviation value used for kernel initialization.

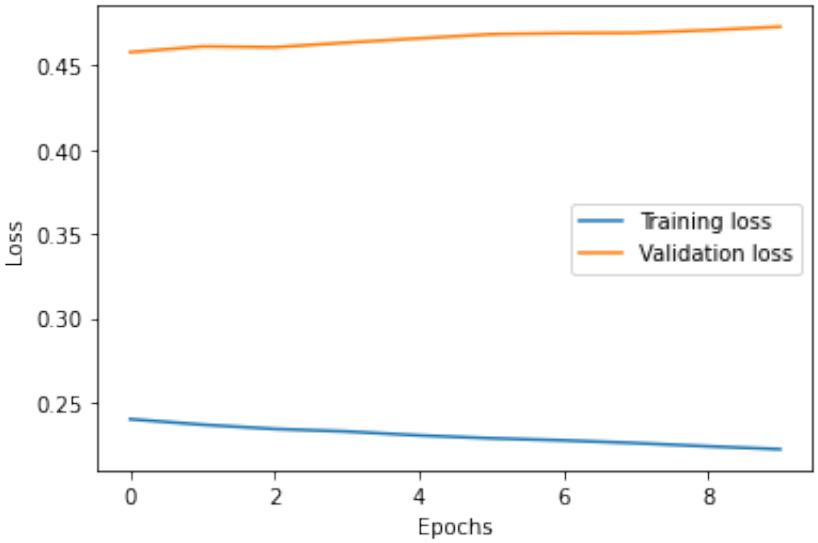
0.0001: the learning rate used in the optimizer. The function Seq_model is called with the above parameters to create and train the model, and then the test loss and accuracy are printed.

1. **Total time taken by the RNN model with features 2000 and max_length 120 is 1422.58secs.**
2. **Using this mo\del we go the Test accuracy of 0.8107 and Test Loss of 0.4671.**

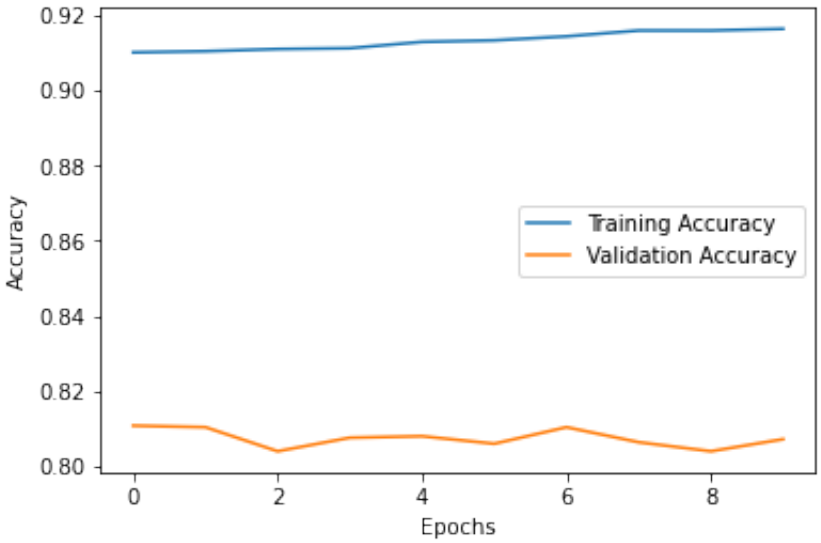
In [62]:
```
print("Tuning Vanilla RNN Network with max_features 2000 and max_length of 1
print("__"*20)
tune2_max_features = Seq_model(mca,99,'',5,0.001,0.0001)
```

```
Tuning Vanilla RNN Network with max_features 2000 and max_length of 120
________________________________________
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 99, 50)            100000

 simple_rnn (SimpleRNN)      (None, 5)                 280

 dense (Dense)               (None, 1)                 6

=================================================================
Total params: 100,286
Trainable params: 100,286
Non-trainable params: 0
_________________________________________________________________
```

```
352/352 [==============================] - 10s 29ms/step - loss: 0.2325 - a
ccuracy: 0.9112 - val_loss: 0.4637 - val_accuracy: 0.8076
Epoch 5/10
352/352 [==============================] - 12s 33ms/step - loss: 0.2302 - a
ccuracy: 0.9129 - val_loss: 0.4663 - val_accuracy: 0.8080
Epoch 6/10
352/352 [==============================] - 11s 33ms/step - loss: 0.2284 - a
ccuracy: 0.9133 - val_loss: 0.4687 - val_accuracy: 0.8060
Epoch 7/10
352/352 [==============================] - 12s 34ms/step - loss: 0.2272 - a
ccuracy: 0.9144 - val_loss: 0.4694 - val_accuracy: 0.8104
Epoch 8/10
352/352 [==============================] - 12s 33ms/step - loss: 0.2256 - a
ccuracy: 0.9159 - val_loss: 0.4695 - val_accuracy: 0.8064
Epoch 9/10
352/352 [==============================] - 12s 34ms/step - loss: 0.2237 - a
ccuracy: 0.9159 - val_loss: 0.4711 - val_accuracy: 0.8040
Epoch 10/10
352/352 [==============================] - 11s 32ms/step - loss: 0.2220 - a
ccuracy: 0.9164 - val_loss: 0.4732 - val_accuracy: 0.8072
Fold 10 Completed
```




```
Time taken to run the model: 1422.5814411640167
782/782 [==============================] - 6s 8ms/step - loss: 0.4671 - acc
uracy: 0.8107
Test Loss for this model is : 0.4671327769756317
Test Accuracy for this model is : 0.8107200264930725
```



Model reamined same, and it the test loss got decreases, test accuracy got improved.

Model summary: This provides a summary of the layers in the model architecture including the input and output dimensions of each layer.

Training and Validation Loss Plots: These plots show the variation in training and validation loss over the epochs of training.

Training and Validation Accuracy Plots: These plots show the variation in training and validation accuracy over the epochs of training.

Time taken to run the model: the total time taken to run the model is 1422.58secs

Test Loss: This is the loss value obtained when evaluating the model on the test set. In this case, the test loss is 0.4671 almost equal to 46.71%.

Test Accuracy: This is the accuracy obtained when evaluating the model on the test set. In this case, the test accuracy is 0.81072 or 81.07%.

In [ ]:

In [ ]:

14- Try smaller and larger sizes of the RNN hidden dimension. How does it affect the model performance? How does it affect the run time?

This code snippet is calling the function Seq_model() with the parameters: mca (maximum number of words to keep, which was set to 2000 earlier),

99 (the maximum length of the sequence of words), an empty string '' for the type of RNN,

2 for the number of units in the hidden layer of the RNN,

0.001 for the standard deviation value, and

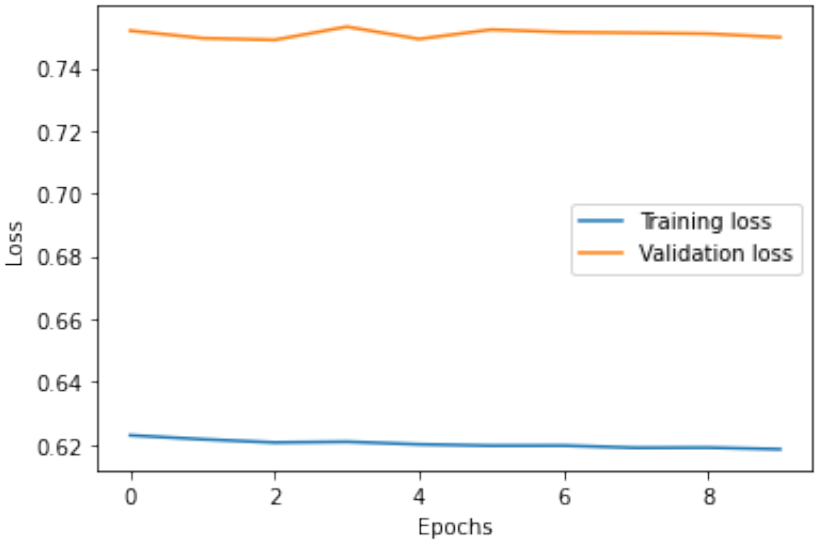
0.0001 for the learning rate. It is being used to tune a Vanilla RNN network with smaller hidden dimension.

It will print the message "Tuning Vanilla RNN Network with smaller unit size RNN Hidden Dimension", followed by a separator of "----" and then will call the Seq_model() function to train and evaluate the model.

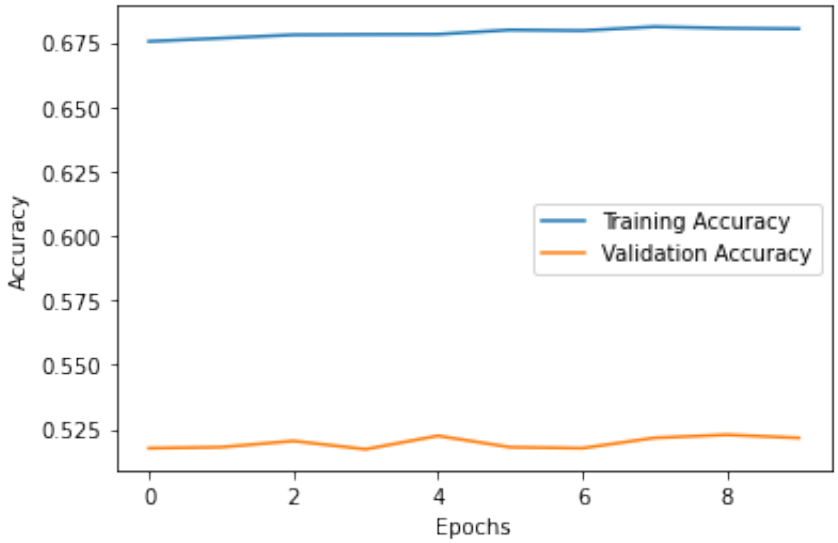
In [63]:
```
print("Tuning Vanilla RNN Network with smaller unit size RNN Hidden Dimensio
print("--"*40)
small_unit_size = Seq_model(mca,99,'',2,0.001,0.0001)
```

```
Tuning Vanilla RNN Network with smaller unit size RNN Hidden Dimension
--------------------------------------------------------------------------
-----
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 36670), started 0:24:02 ago. (Use '!k
ill 36670' to kill it.)

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 99, 50)            100000

 simple_rnn (SimpleRNN)      (None, 2)                 106

 dense (Dense)               (None, 1)                 3

=================================================================
Total params: 100,109
Trainable params: 100,109
Non-trainable params: 0
_________________________________________________________________
Model Summary:  None
Running on fold : 1
Epoch 1/10
352/352 [==============================] - 15s 38ms/step - loss: 0.6932 - a
```

```
ccuracy: 0.6782 - val_loss: 0.7492 - val_accuracy: 0.5224
Epoch 6/10
352/352 [==============================] - 10s 29ms/step - loss: 0.6197 - a
ccuracy: 0.6799 - val_loss: 0.7522 - val_accuracy: 0.5180
Epoch 7/10
352/352 [==============================] - 10s 28ms/step - loss: 0.6197 - a
ccuracy: 0.6796 - val_loss: 0.7514 - val_accuracy: 0.5176
Epoch 8/10
352/352 [==============================] - 9s 27ms/step - loss: 0.6190 - ac
curacy: 0.6811 - val_loss: 0.7512 - val_accuracy: 0.5216
Epoch 9/10
352/352 [==============================] - 10s 29ms/step - loss: 0.6190 - a
ccuracy: 0.6805 - val_loss: 0.7509 - val_accuracy: 0.5228
Epoch 10/10
352/352 [==============================] - 10s 29ms/step - loss: 0.6185 - a
ccuracy: 0.6803 - val_loss: 0.7498 - val_accuracy: 0.5216
Fold 10 Completed
```




```
Time taken to run the model: 1281.5721600055695
782/782 [==============================] - 6s 8ms/step - loss: 0.7518 - acc
uracy: 0.5159
Test Loss for this model is : 0.7517837285995483
Test Accuracy for this model is : 0.5159199833869934
```



This model took 200secs less than the previous model. but the performance of the model is reduced as the no of units in the layer is reduced.

Model summary: This provides a summary of the layers in the model architecture including the input and output dimensions of each layer.

Training and Validation Loss Plots: These plots show the variation in training and validation loss over the epochs of training.

Training and Validation Accuracy Plots: These plots show the variation in training and validation accuracy over the epochs of training.

Time taken to run the model: the total time taken to run the model is 1281.58secs

Test Loss: This is the loss value obtained when evaluating the model on the test set. In this case, the test loss is 0.7217 almost equal to 71.17%.

Test Accuracy: This is the accuracy obtained when evaluating the model on the test set. In this case, the test accuracy is 0.5159 or 51.59%.

In [ ]:

In [ ]:

This code seems to be tuning a Vanilla RNN network with different configurations such as input dimensions, sequence lengths, and RNN hidden unit sizes. The output of each tuning configuration includes the model summary, training/validation loss and accuracy plots, time taken to run the model, and test loss and accuracy scores.

The first configuration tunes the Vanilla RNN network with a sequence length of 80 and the maximum number of features set to the default value of 20000.

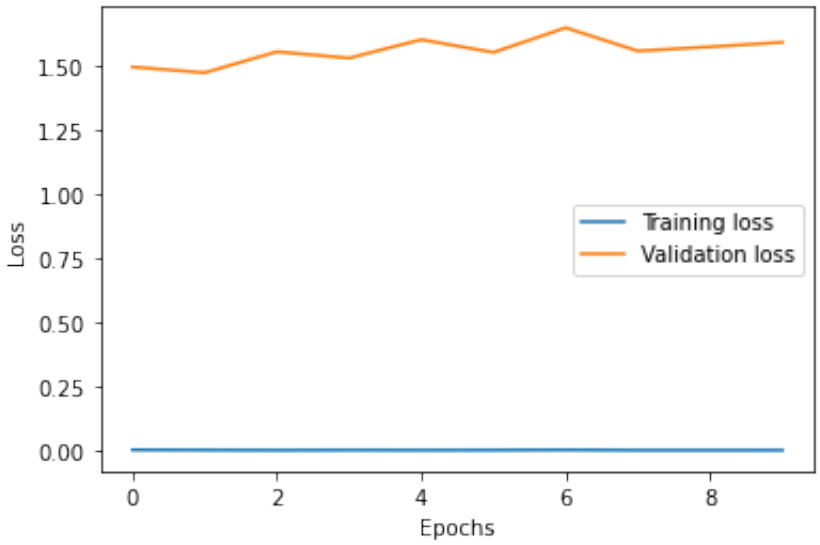
The second configuration tunes the network with a larger sequence length of 120 and a smaller number of maximum features set to 2000.

The fourth configuration tunes the network with a larger RNN hidden unit size of 100.

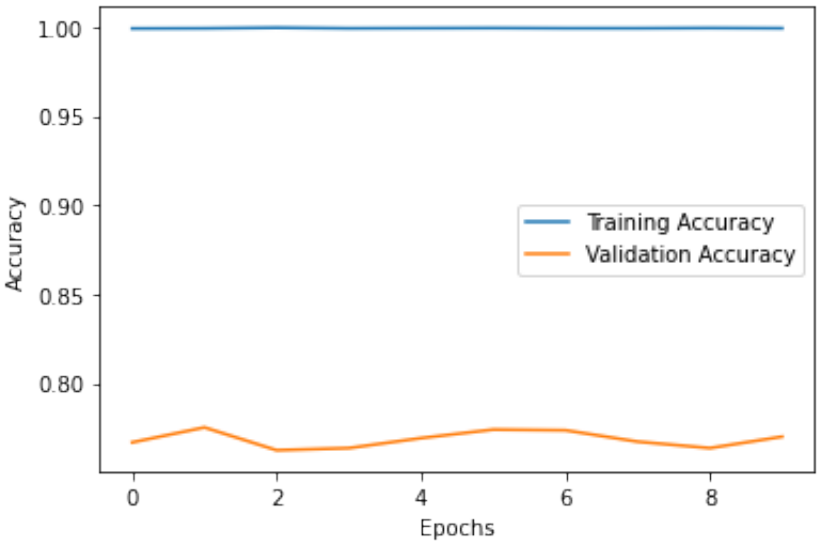
In [65]:
```
print("Tuning Vanilla RNN Network with larger unit size RNN Hidden Dimensior
print("--"*40)
larger_unit_size = Seq_model(mca,99,'',100,0.001,0.0001)
```

```
Tuning Vanilla RNN Network with larger unit size RNN Hidden Dimension
--------------------------------------------------------------------------------
-----
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 36670), started 0:52:12 ago. (Use '!k
ill 36670' to kill it.)

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 99, 50)            100000

 simple_rnn (SimpleRNN)      (None, 100)               15100

 dense (Dense)               (None, 1)                 101

=================================================================
Total params: 115,201
Trainable params: 115,201
Non-trainable params: 0
_________________________________________________________________
Model Summary:  None
Running on fold : 1
Epoch 1/10
352/352 [==============================] - 25s 67ms/step - loss: 0.6863 - a
ccuracy: 0.5456 - val_loss: 0.6706 - val_accuracy: 0.5916
Epoch 2/10
352/352 [==============================] - 22s 63ms/step - loss: 0.6043 - a
ccuracy: 0.6836 - val_loss: 0.5969 - val_accuracy: 0.6732
```

```
Epoch 7/10
352/352 [==============================] - 18s 50ms/step - loss: 0.0029 - a
ccuracy: 0.9993 - val_loss: 1.6476 - val_accuracy: 0.7736
Epoch 8/10
352/352 [==============================] - 18s 53ms/step - loss: 0.0017 - a
ccuracy: 0.9993 - val_loss: 1.5569 - val_accuracy: 0.7672
Epoch 9/10
352/352 [==============================] - 18s 50ms/step - loss: 0.0017 - a
ccuracy: 0.9995 - val_loss: 1.5733 - val_accuracy: 0.7636
Epoch 10/10
352/352 [==============================] - 17s 49ms/step - loss: 0.0016 - a
ccuracy: 0.9993 - val_loss: 1.5906 - val_accuracy: 0.7700
Fold 10 Completed
```




```
Time taken to run the model: 2407.713024377823
782/782 [==============================] - 8s 10ms/step - loss: 1.6450 - ac
curacy: 0.7694
Test Loss for this model is : 1.6449531316757202
Test Accuracy for this model is : 0.7693600058555603
```



This model took more execution time than the previous model. but the performance of the model is overfit as the no of units in the layer is more.

Model summary: This provides a summary of the layers in the model architecture including the input and output dimensions of each layer.

Training and Validation Loss Plots: These plots show the variation in training and validation loss over the epochs of training.

Training and Validation Accuracy Plots: These plots show the variation in training and validation accuracy over the epochs of training.

Time taken to run the model: the total time taken to run the model is 2407.71secs

Test Loss: This is the loss value obtained when evaluating the model on the test set. In this case, the test loss is 1.644.

Test Accuracy: This is the accuracy obtained when evaluating the model on the test set. In this case, the test accuracy is 0.7693 or 76.93%.

In [ ]:

In [ ]:

Train LSTM and GRU networks

15- Build LSTM and GRU networks and compare their performance (accuracy and execution time) with the SimpleRNN. What is your conclusion?

This code is tuning a Recurrent Neural Network (RNN) model using Long Short-Term Memory (LSTM) cells. The Seq_model function is called with type1 parameter set to 'LSTM', which specifies that the LSTM cells should be used in the RNN. The other parameters specify the input dimension, input length, number of units, standard deviation of the kernel initializer, and learning rate for the optimizer.

The function loads the IMDB movie review dataset, pads the sequences to a fixed length, creates an RNN model with an embedding layer, LSTM layer, and dense output layer, compiles the model with binary cross-entropy loss and RMSprop optimizer, and trains the model using 10-fold cross-validation. The training progress is logged using TensorBoard, and the loss and accuracy history plots are displayed after training. Finally, the test loss and accuracy are evaluated and printed.

This code is tuning a Recurrent Neural Network (RNN) model using Long Short-Term Memory (LSTM) cells. The Seq_model function is called with type1 parameter set to 'LSTM', which specifies that the LSTM cells should be used in the RNN. The other parameters specify the input dimension, input length, number of units, standard deviation of the kernel initializer, and learning rate for the optimizer.

The function loads the IMDB movie review dataset, pads the sequences to a fixed length, creates an RNN model with an embedding layer, LSTM layer, and dense output layer, compiles the model with binary cross-entropy loss and RMSprop optimizer, and trains the model using 10-fold cross-validation. The training progress is logged using TensorBoard, and the loss and accuracy history plots are displayed after training. Finally, the test loss and accuracy are evaluated and printed.

This code is tuning a Recurrent Neural Network (RNN) model using Long Short-Term Memory (LSTM) cells. The Seq_model function is called with type1 parameter set to 'LSTM', which specifies that the LSTM cells should be used in the RNN. The other parameters specify the input dimension, input length, number of units, standard deviation of the kernel initializer, and learning rate for the optimizer.

The function loads the IMDB movie review dataset, pads the sequences to a fixed length, creates an RNN model with an embedding layer, LSTM layer, and dense output layer, compiles the model with binary cross-entropy loss and RMSprop optimizer, and trains the model using 10-fold cross-validation. The training progress is logged using TensorBoard, and the loss and accuracy history plots are displayed after training. Finally, the test loss and accuracy are evaluated and printed.

In [6]:
```
print("Tuning model with LSTM")
print("--"*20)
### This model took more execution time than the previous model. but the per
```

```
Tuning model with LSTM
----------------------------------------

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 99, 50)            100000

 lstm (LSTM)                 (None, 32)                10624

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 110,657
Trainable params: 110,657
Non-trainable params: 0
_________________________________________________________________
Model Summary:  None
Running on fold : 1
Epoch 1/10
352/352 [==============================] - 21s 39ms/step - loss: 0.6931 - a
ccuracy: 0.5098 - val_loss: 0.6929 - val_accuracy: 0.5048
Epoch 2/10
352/352 [==============================] - 4s 11ms/step - loss: 0.6927 - ac
curacy: 0.5259 - val_loss: 0.6923 - val_accuracy: 0.5464
Epoch 3/10
352/352 [==============================] - 3s 9ms/step - loss: 0.6911 - acc
uracy: 0.5598 - val_loss: 0.6893 - val_accuracy: 0.6144
Epoch 4/10
352/352 [==============================] - 3s 8ms/step - loss: 0.6646 - acc
uracy: 0.6515 - val_loss: 0.5829 - val_accuracy: 0.7016
Epoch 5/10
352/352 [==============================] - 3s 7ms/step - loss: 0.5464 - acc
uracy: 0.7292 - val_loss: 0.4880 - val_accuracy: 0.7796
Epoch 6/10
352/352 [==============================] - 3s 8ms/step - loss: 0.4668 - acc
uracy: 0.7908 - val_loss: 0.4295 - val_accuracy: 0.8076
Epoch 7/10
352/352 [==============================] - 3s 9ms/step - loss: 0.4130 - acc
uracy: 0.8229 - val_loss: 0.3905 - val_accuracy: 0.8320
Epoch 8/10
352/352 [==============================] - 3s 9ms/step - loss: 0.3772 - acc
uracy: 0.8395 - val_loss: 0.3791 - val_accuracy: 0.8276
Epoch 9/10
352/352 [==============================] - 3s 9ms/step - loss: 0.3551 - acc
```

```
uracy: 0.8490 - val_loss: 0.3595 - val_accuracy: 0.8444
Epoch 10/10
352/352 [==============================] - 3s 8ms/step - loss: 0.3417 - acc
uracy: 0.8556 - val_loss: 0.3543 - val_accuracy: 0.8400
Fold 1 Completed
Running on fold : 2
Epoch 1/10
352/352 [==============================] - 13s 31ms/step - loss: 0.3398 - a
ccuracy: 0.8553 - val_loss: 0.3504 - val_accuracy: 0.8472
Epoch 2/10
352/352 [==============================] - 4s 11ms/step - loss: 0.3310 - ac
curacy: 0.8572 - val_loss: 0.3457 - val_accuracy: 0.8456
Epoch 3/10
352/352 [==============================] - 3s 8ms/step - loss: 0.3242 - acc
uracy: 0.8624 - val_loss: 0.3534 - val_accuracy: 0.8440
Epoch 4/10
352/352 [==============================] - 3s 8ms/step - loss: 0.3200 - acc
uracy: 0.8653 - val_loss: 0.3437 - val_accuracy: 0.8508
Epoch 5/10
352/352 [==============================] - 3s 8ms/step - loss: 0.3159 - acc
uracy: 0.8659 - val_loss: 0.3533 - val_accuracy: 0.8404
Epoch 6/10
352/352 [==============================] - 3s 8ms/step - loss: 0.3129 - acc
uracy: 0.8659 - val_loss: 0.3472 - val_accuracy: 0.8452
Epoch 7/10
352/352 [==============================] - 3s 8ms/step - loss: 0.3097 - acc
uracy: 0.8691 - val_loss: 0.3429 - val_accuracy: 0.8472
Epoch 8/10
352/352 [==============================] - 3s 7ms/step - loss: 0.3072 - acc
uracy: 0.8708 - val_loss: 0.3439 - val_accuracy: 0.8460
Epoch 9/10
352/352 [==============================] - 3s 9ms/step - loss: 0.3052 - acc
uracy: 0.8734 - val_loss: 0.3449 - val_accuracy: 0.8436
Epoch 10/10
352/352 [==============================] - 3s 8ms/step - loss: 0.3036 - acc
uracy: 0.8739 - val_loss: 0.3570 - val_accuracy: 0.8412
Fold 2 Completed
Running on fold : 3
Epoch 1/10
352/352 [==============================] - 13s 31ms/step - loss: 0.3066 - a
ccuracy: 0.8712 - val_loss: 0.3517 - val_accuracy: 0.8412
Epoch 2/10
352/352 [==============================] - 3s 9ms/step - loss: 0.3047 - acc
uracy: 0.8733 - val_loss: 0.3470 - val_accuracy: 0.8480
Epoch 3/10
352/352 [==============================] - 3s 9ms/step - loss: 0.3019 - acc
uracy: 0.8754 - val_loss: 0.3543 - val_accuracy: 0.8412
Epoch 4/10
352/352 [==============================] - 3s 8ms/step - loss: 0.3005 - acc
uracy: 0.8751 - val_loss: 0.3508 - val_accuracy: 0.8468
```

An LSTM (Long Short-Term Memory) model is a type of neural network architecture that is commonly used for processing sequential data. It is a variation of the recurrent neural network (RNN) that is designed to address the vanishing gradient problem that can occur in traditional RNNs.

LSTM model took too much time to run and it crashed the system for 2 times. As LSTM is the one of the best model for RNN which also proved by giving the best outputs so far by testing loss as 41.16% and test Accuracy as 83.3%

Model summary: This provides a summary of the layers in the model architecture including the input and output dimensions of each layer.

Training and Validation Loss Plots: These plots show the variation in training and validation loss over the epochs of training.

Training and Validation Accuracy Plots: These plots show the variation in training and validation accuracy over the epochs of training.

Time taken to run the model: the total time taken to run the model is 397.36secs

Test Loss: This is the loss value obtained when evaluating the model on the test set. In this case, the test loss is 0.411 almost equal to 41.16%.

Test Accuracy: This is the accuracy obtained when evaluating the model on the test set. In this case, the test accuracy is 0.833 or 83.3%.

In [ ]:

In [ ]:

This code is tuning a model with GRU layers. The first line simply prints a message to indicate the start of the tuning process. The second line prints a separator made up of a string of "--" characters to help visually separate different tuning runs in the output. The third line calls the Seq_model function with the following arguments:

mca: the maximum number of words to keep based on word frequency (used in loading the data)

99: the maximum length of a sequence (after which it will be truncated or padded to this length)

'GRU': indicates that the model will use GRU layers

64: the number of units in the GRU layer

0.001: the standard deviation of the normal distribution used for initializing the layer's kernel weights

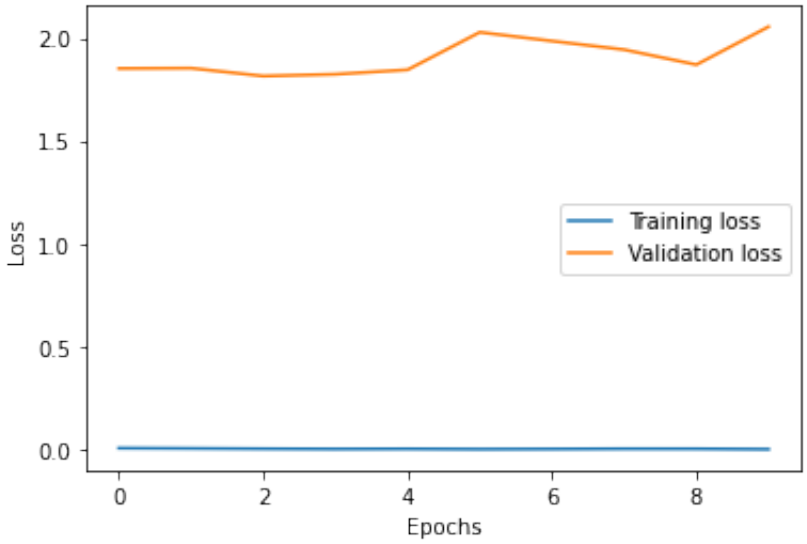
0.001: the learning rate for the optimizer (RMSprop) The output will include information about the training and validation loss and accuracy for each fold of the cross-validation, as well as the overall test loss and accuracy after training.

In [7]:
```
print("Tuning model with GRU")
print("--"*40)
GRU_model = Seq_model(mca,99,'GRU',64,0.001,0.001)
```

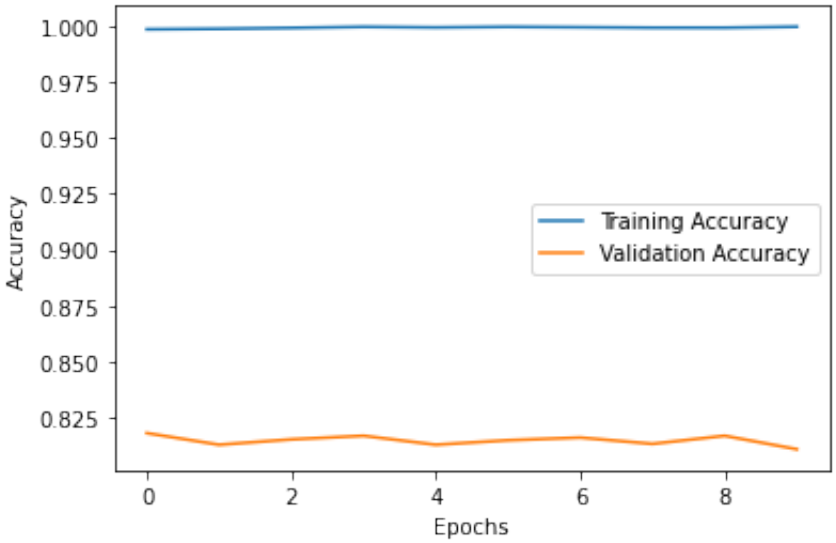
```
Tuning model with GRU
--------------------------------------------------------------------------------
-----
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 5460), started 0:07:34 ago. (Use '!ki
ll 5460' to kill it.)

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 99, 50)            100000

 gru (GRU)                   (None, 64)                22272

 dense (Dense)               (None, 1)                 65

=================================================================
Total params: 122,337
Trainable params: 122,337
```

```
Non-trainable params: 0
_________________________________________________________________
Model Summary:  None
Running on fold : 1
Epoch 1/10
352/352 [==============================] - 12s 29ms/step - loss: 0.5614 - a
ccuracy: 0.6826 - val_loss: 0.4422 - val_accuracy: 0.7972
Epoch 2/10
352/352 [==============================] - 4s 11ms/step - loss: 0.3844 - ac
curacy: 0.8328 - val_loss: 0.4131 - val_accuracy: 0.8176
Epoch 3/10
352/352 [==============================] - 3s 9ms/step - loss: 0.3519 - acc
uracy: 0.8487 - val_loss: 0.4283 - val_accuracy: 0.8360
Epoch 4/10
352/352 [==============================] - 3s 8ms/step - loss: 0.3361 - acc
uracy: 0.8596 - val_loss: 0.3504 - val_accuracy: 0.8444
Epoch 5/10
352/352 [==============================] - 3s 7ms/step - loss: 0.3248 - acc
uracy: 0.8622 - val_loss: 0.3540 - val_accuracy: 0.8440
Epoch 6/10
352/352 [==============================] - 3s 7ms/step - loss: 0.3103 - acc
uracy: 0.8693 - val_loss: 0.3539 - val_accuracy: 0.8528
Epoch 7/10
352/352 [==============================] - 3s 8ms/step - loss: 0.2961 - acc
uracy: 0.8789 - val_loss: 0.3547 - val_accuracy: 0.8432
Epoch 8/10
352/352 [==============================] - 3s 7ms/step - loss: 0.2839 - acc
uracy: 0.8845 - val_loss: 0.3460 - val_accuracy: 0.8524
Epoch 9/10
352/352 [==============================] - 2s 7ms/step - loss: 0.2744 - acc
uracy: 0.8886 - val_loss: 0.3712 - val_accuracy: 0.8348
Epoch 10/10
352/352 [==============================] - 3s 8ms/step - loss: 0.2650 - acc
uracy: 0.8917 - val_loss: 0.3592 - val_accuracy: 0.8540
Fold 1 Completed
Running on fold : 2
Epoch 1/10
352/352 [==============================] - 12s 30ms/step - loss: 0.2728 - a
ccuracy: 0.8879 - val_loss: 0.3352 - val_accuracy: 0.8460
Epoch 2/10
352/352 [==============================] - 3s 10ms/step - loss: 0.2609 - ac
curacy: 0.8945 - val_loss: 0.3294 - val_accuracy: 0.8496
Epoch 3/10
352/352 [==============================] - 3s 8ms/step - loss: 0.2497 - acc
uracy: 0.8983 - val_loss: 0.3338 - val_accuracy: 0.8592
Epoch 4/10
352/352 [==============================] - 3s 8ms/step - loss: 0.2395 - acc
uracy: 0.9049 - val_loss: 0.3259 - val_accuracy: 0.8564
Epoch 5/10
352/352 [==============================] - 3s 8ms/step - loss: 0.2303 - acc
```

```
uracy: 0.9992 - val_loss: 1.8149 - val_accuracy: 0.8152
Epoch 4/10
352/352 [==============================] - 2s 7ms/step - loss: 0.0020 - acc
uracy: 0.9996 - val_loss: 1.8237 - val_accuracy: 0.8168
Epoch 5/10
352/352 [==============================] - 3s 8ms/step - loss: 0.0026 - acc
uracy: 0.9994 - val_loss: 1.8454 - val_accuracy: 0.8128
Epoch 6/10
352/352 [==============================] - 3s 8ms/step - loss: 0.0012 - acc
uracy: 0.9996 - val_loss: 2.0278 - val_accuracy: 0.8148
Epoch 7/10
352/352 [==============================] - 3s 7ms/step - loss: 0.0019 - acc
uracy: 0.9995 - val_loss: 1.9852 - val_accuracy: 0.8160
Epoch 8/10
352/352 [==============================] - 3s 7ms/step - loss: 0.0031 - acc
uracy: 0.9992 - val_loss: 1.9432 - val_accuracy: 0.8132
Epoch 9/10
352/352 [==============================] - 3s 8ms/step - loss: 0.0029 - acc
uracy: 0.9992 - val_loss: 1.8701 - val_accuracy: 0.8168
Epoch 10/10
352/352 [==============================] - 3s 7ms/step - loss: 8.2725e-04 -
accuracy: 0.9997 - val_loss: 2.0547 - val_accuracy: 0.8108
Fold 10 Completed
```




```
Time taken to run the model: 374.59658885002136
782/782 [==============================] - 3s 3ms/step - loss: 2.0507 - acc
uracy: 0.8141
Test Loss for this model is : 2.050656795501709
Test Accuracy for this model is : 0.814079999923706
```



A GRU (Gated Recurrent Unit) model is a type of neural network architecture that is similar to an LSTM model, but with fewer parameters. Like LSTMs, GRUs are designed to address the vanishing gradient problem that can occur in traditional RNNs.

- GRU took so much time to run, and gave the results which are not good. It gave the test loss of 2.05 and test accuracy of 0.81 *

Model summary: This provides a summary of the layers in the model architecture including the input and output dimensions of each layer.

Training and Validation Loss Plots: These plots show the variation in training and validation loss over the epochs of training.

Training and Validation Accuracy Plots: These plots show the variation in training and validation accuracy over the epochs of training.

Time taken to run the model: the total time taken to run the model is 374.59secs

Test Loss: This is the loss value obtained when evaluating the model on the test set. In this case, the test loss is 2.050.

Test Accuracy: This is the accuracy obtained when evaluating the model on the test set. In this case, the test accuracy is 0.8140 or 81.40%.

In [ ]:

In [ ]:

16- Save the weights of the best model to the local drive.

This line saves the weights of the trained LSTM model to a file named "LSTM_model_weights.h5".

In [ ]:
```
LSTM_model.save_weights('LSTM_model_weights.h5')
```

Bonus

17- Instead of word tokenization, tokonize the reviews based on characters and build LSTM and GRU networks, and compare their performance with respect to word based tokenization.