NN and Neural Networks Assignment 1 Report

Name: Danish Mehra

Student ID: 200675596

Part A

1. Sanity Check after pre-processing:

```
[['[', 'Sense', 'and', 'Sensibility', 'by', 'Jane', 'Austen', '1811', ']'], ['CHAPTER', '1'], ...]
The new length of the preprocessed output
13660
[['Sense', 'Sensibility', 'Jane', 'Austen'], ['The', 'family', 'Dashwood', 'long', 'settled', 'Sussex'],
```

2. Sanity Check after Creating the Corpus Vocabulary and Preparing the Data:

```
print('Number of unique words:', len(word2idx))
```

```
Number of unique words: 10808
```

```
[ ] print('\nSample word2idx: ', list(word2idx.items())[:10])
```

```
Sample word2idx:  [('Sense', 1), ('Sensibility', 2), ('Jane', 3), ('Austen', 4), ('The', 5), ('family', 6)
```

```
[ ] print('\nSample idx2word:', list(idx2word.items())[:10])
```

```
Sample idx2word: [(1, 'Sense'), (2, 'Sensibility'), (3, 'Jane'), (4, 'Austen'), (5, 'The'), (6, 'family'),
```

```
[ ] print('\nSample sents_as_id:', prepareSentsAsId(preprocessed_sample))
```

```
Sample sents_as_id: [[1, 2, 3, 4], [5, 6, 7, 8, 9, 10, 11, 12, 13]]
```

3. model.summary Sanity Check:

```
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_3 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
target_embed_layer (Embedding)  (None, 1, 100)       1080800     input_2[0][0]
__________________________________________________________________________________________________
context_embed_layer (Embedding) (None, 1, 100)       1080800     input_3[0][0]
__________________________________________________________________________________________________
reshape (Reshape)               (None, 100)          0           target_embed_layer[0][0]
__________________________________________________________________________________________________
reshape_1 (Reshape)             (None, 100)          0           context_embed_layer[0][0]
__________________________________________________________________________________________________
dot (Dot)                       (None, 1)            0           reshape[0][0]
                                                                 reshape_1[0][0]
__________________________________________________________________________________________________
activation (Activation)         (None, 1)            0           dot[0][0]
==================================================================================================
Total params: 2,161,600
Trainable params: 2,161,600
Non-trainable params: 0
```

4. What would be the Inputs and Outputs to the model be?

Inputs are wordtoindex of context word and target word. Outputs are probability of pair is positive or negative

How would you use the Keras framework to create this architecture?

Define the layers in the model. Connect the flow of input and outputs through each layer. Fit the model and then evaluate on test data to get fast and accurate results.

What are the reasons this training approach is considered inefficient?

Sometimes when we use two different embedding Matrices, same word may have different vectors.

5. **Printing the dataframe for word embeddings**

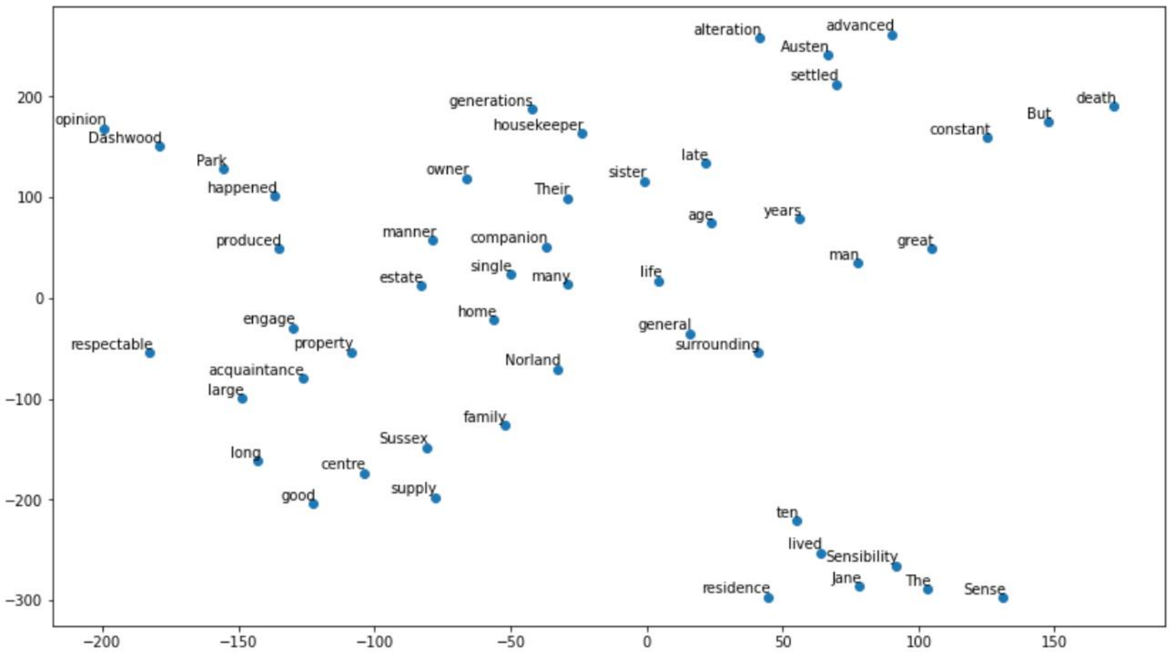
```
[ ] from pandas import DataFrame

    print(DataFrame(word_embeddings, index=idx2word.values()).head(10))
```

```
                    0         1         2  ...        97        98        99
Sense        0.010083 -0.012033 -0.016542  ...  0.003729 -0.014503  0.019011
Sensibility  0.010376 -0.023291  0.008540  ...  0.030873 -0.024183  0.011487
Jane         0.016210 -0.014352  0.023118  ...  0.020753 -0.014591  0.034952
Austen       0.011112 -0.078037  0.038333  ...  0.008071  0.010603  0.035187
The          0.005096 -0.014580 -0.007197  ...  0.015193 -0.015456  0.022403
family       0.027164 -0.017446 -0.005271  ... -0.011948  0.054367  0.092967
Dashwood    -0.145946 -0.032238  0.018873  ...  0.052960 -0.162810  0.091453
long        -0.236915 -0.052849  0.114623  ...  0.102109 -0.154706 -0.048957
settled      0.067867 -0.000789 -0.019915  ...  0.061657 -0.019171  0.089344
Sussex       0.053522 -0.021792  0.059648  ...  0.058734 -0.012536  0.010037

[10 rows x 100 columns]
```

6. Plt.annotate Output:



Part B

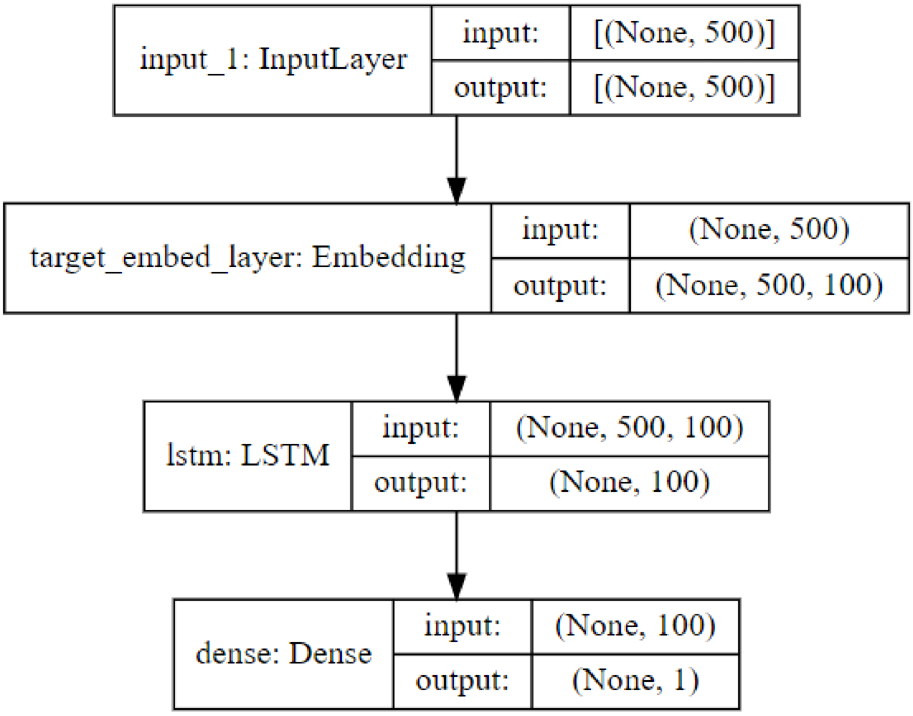
1. Readying the inputs for the LSTM Sanity Check:

```
Length of sample train_data before preprocessing: 218
Length of sample train_data after preprocessing: 500
Sample train data: [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    1   13   21   15   42  529  972 1621 1384   64  457 4467
   65 3940    3  172   35  255    4   24   99   42  837  111   49  669
    2    8   34  479  283    4  149    3  171  111  166    2  335  384
   38    3  171 4535 1110   16  545   37   12  446    3  191   49   15
    5  146 2024   18   13   21    3 1919 4612  468    3   21   70   86
```

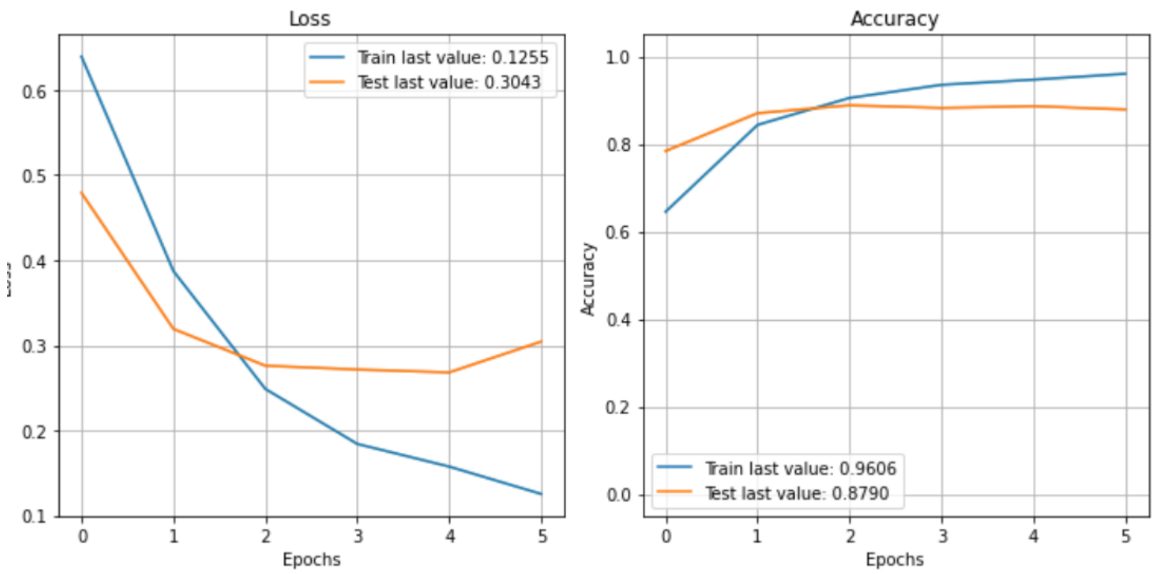
2. Structure of the model obtained:

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 500)]             0
_________________________________________________________________
target_embed_layer (Embeddin (None, 500, 100)          1000000
_________________________________________________________________
lstm (LSTM)                  (None, 100)               80400
_________________________________________________________________
dense (Dense)                (None, 1)                 101
=================================================================
Total params: 1,080,501
Trainable params: 1,080,501
Non-trainable params: 0
_________________________________________________________________
```



3. **Plot of training and validation accuracy and loss data:**



The optimal stopping point should be about 2 epochs where both validation and training data converge so that we don't get any overfitting.

4. **Output of the test accuracy and test loss:**

```
[15] # YOUR CODE TO EVALUATE THE MODEL ON TEST DATA GOES HERE
     results = model.evaluate(validation_x, validation_y)
     print('test_loss:', results[0], 'test_accuracy:', results[1])
```

```
63/63 [==============================] - 9s 126ms/step - loss: 0.3043 - accuracy: 0.8790
test_loss: 0.30432361364364624 test_accuracy: 0.8790000081062317
```

5. model.summary

```
model.summary()
```

```
(10000, 100)
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 500)]             0
_________________________________________________________________
target_embed_layer (Embeddin (None, 500, 100)          1000000
_________________________________________________________________
lstm_2 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 1,080,501
Trainable params: 1,080,501
Non-trainable params: 0
_________________________________________________________________
```

Sanity Check

Print the shape of the word embeddings using the line of code below. It should return (VOCAB

```
[17] print('Shape of word_embeddings:', word_embeddings.shape)
```

```
Shape of word_embeddings: (10000, 100)
```

6. Visualizing the reviews:

Step 3: Visualize sample review

View a sample review text using the lines of code below:

```
[20] print(' '.join(idx2word[idx] for idx in train_data[0]))
```

```
<START> this film was just brilliant casting location scenery story direction everyone's really suited the part they played and
```

7. Visualizing the word embeddings:

8. Visualizing the Word_Embeddings

Visualize the word embeddings for 10 of the words using pandas DataFrame like we did in lab 3

```
[21] from pandas import DataFrame

     print(DataFrame(word_embeddings, index=idx2word.values()).head(10))
```

```
                  0         1         2  ...        97        98        99
woods     -0.018422 -0.028894  0.044005  ...  0.042796 -0.003599  0.045664
hanging   -0.050355  0.019669 -0.023673  ...  0.050463  0.028757  0.005433
woody     -0.032700 -0.030171 -0.001717  ... -0.027080  0.004520 -0.017445
arranged  -0.045771 -0.032585  0.026716  ... -0.035090  0.019204  0.026020
bringing  -0.024623  0.004106 -0.001668  ...  0.027033 -0.022006  0.025479
wooden     0.032191  0.002833  0.033943  ... -0.006398 -0.003582  0.011662
errors    -0.005533  0.043958 -0.036572  ... -0.009391 -0.009012 -0.050597
dialogs   -0.011855 -0.045498 -0.023849  ... -0.029265 -0.040318 -0.026097
kids      -0.022314 -0.028868 -0.038487  ... -0.016460  0.025459  0.007085
uplifting -0.028250  0.007414  0.048871  ... -0.047134  0.028057  0.025444

[10 rows x 100 columns]
```

8. **Create a new model that is a copy of the model step 3. To this new model,add two dropout layers, one between the embedding layer and the LSTM layer and another between the LSTM layer and the output layer. Repeat steps 4 and 5 for this model. What do you observe?**

Answer: Dropout is a technique used to prevent a model from overfitting. Dropout works by randomly setting the outgoing edges of hidden units (neurons that make up hidden layers) to 0 at each update of the training phase. I observed that the validation and training accuracy came out to be similar about around 51%. I can clearly observe that there is no overfitting now and adding these dropout layers increased the reliability of the model.

Q: Experiment with training the model with batch sizes of 1, 32, len(training_data). What do you observe?

Answer: Although with smaller batch size, I got a much more accurate test result, it took so much computational power and time as the batch size was decreased. Another downside of using

a smaller batch size is that the model is not guaranteed to converge to the global optima. It will bounce around the global optima.

With Batch size = 32

```
Total params: 1,080,501
Trainable params: 1,080,501
Non-trainable params: 0
_________________________________________________________________
Epoch 1/6
719/719 [==============================] - 423s 585ms/step - loss: 0.7811 - accuracy: 0.6060 - val_loss: 0.5487 - val_accuracy: 0.7145
Epoch 2/6
317/719 [============>.................] - ETA: 3:43 - loss: 0.7874 - accuracy: 0.6555
---------------------------------------------------------------------------
```

With Batch Size = 1

```
from sklearn.model_selection import train_test_split
train_x, validation_x, train_y, validation_y = train_test_split(padded_train_data, train_labels, test_size=0.08)
history = model.fit(train_x, train_y, epochs=6, batch_size=1, validation_data=(validation_x, validation_y))

Epoch 1/6
   46/23000 [..............................] - ETA: 1:15:32 - loss: 0.6610 - accuracy: 0.7609
```

As you can see with a smaller batch size, it took even longer for a single epoch.

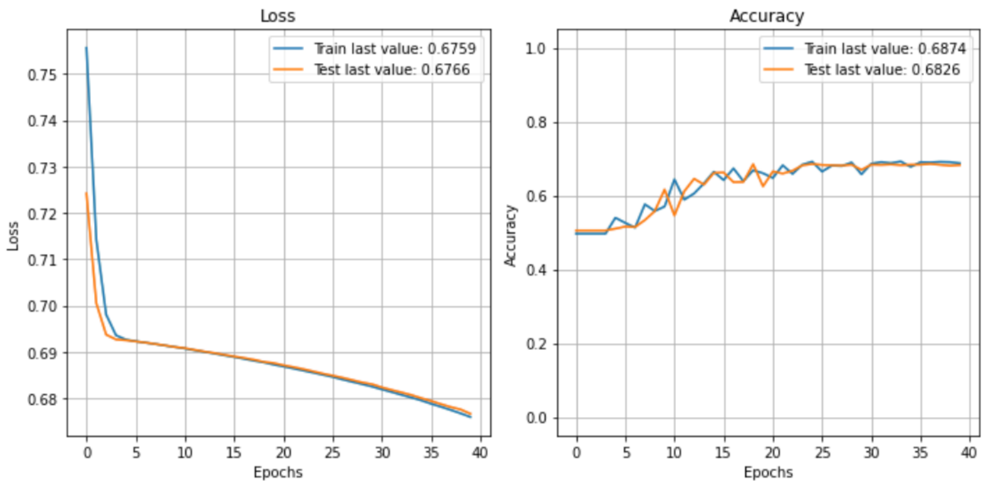
Part C

1. model.summary and plotting of the graph:

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 256)]             0
_________________________________________________________________
lambda_3 (Lambda)            (None, 256, 10000)        0
_________________________________________________________________
global_average_pooling1d_mas (None, 10000)             0
_________________________________________________________________
dense (Dense)                (None, 16)                160016
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 17
=================================================================
Total params: 160,033
Trainable params: 160,033
Non-trainable params: 0
_________________________________________________________________
```

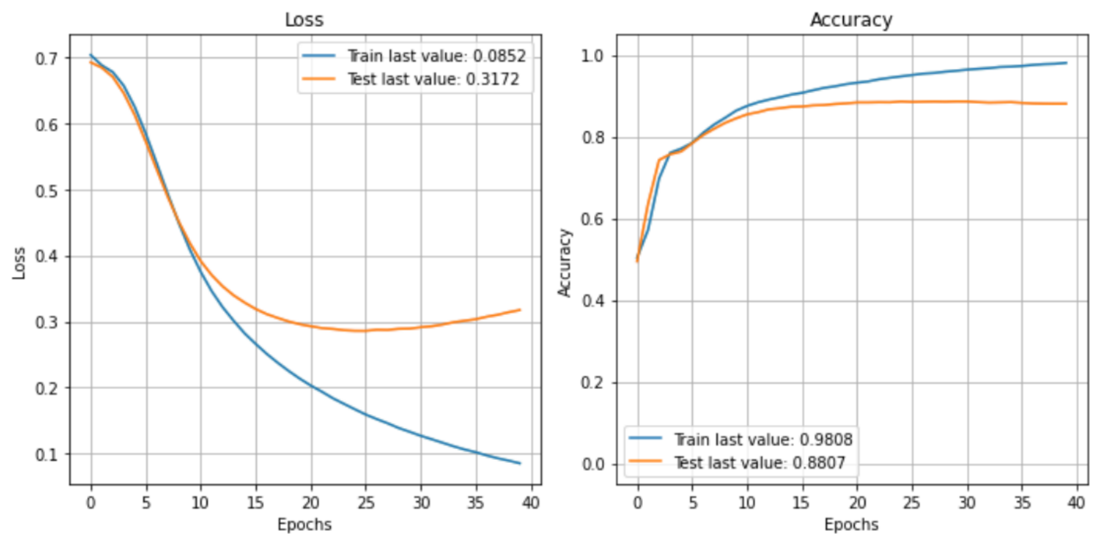
```
# Plotted the graph for accuracy and loss over training and validation data
from plot_keras_history import plot_history
import matplotlib.pyplot as plt
plot_history(history.history, path="standard.png")
plt.show()
```



2. model2.summary and plotting of training and validation loss:

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         [(None, 256)]             0
_________________________________________________________________
target_embed_layer (Embeddin (None, 256, 100)          1000000
_________________________________________________________________
global_average_pooling1d_mas (None, 100)               0
_________________________________________________________________
dense_4 (Dense)              (None, 16)                1616
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 17
=================================================================
Total params: 1,001,633
Trainable params: 1,001,633
Non-trainable params: 0
```

```
plt.show()
```



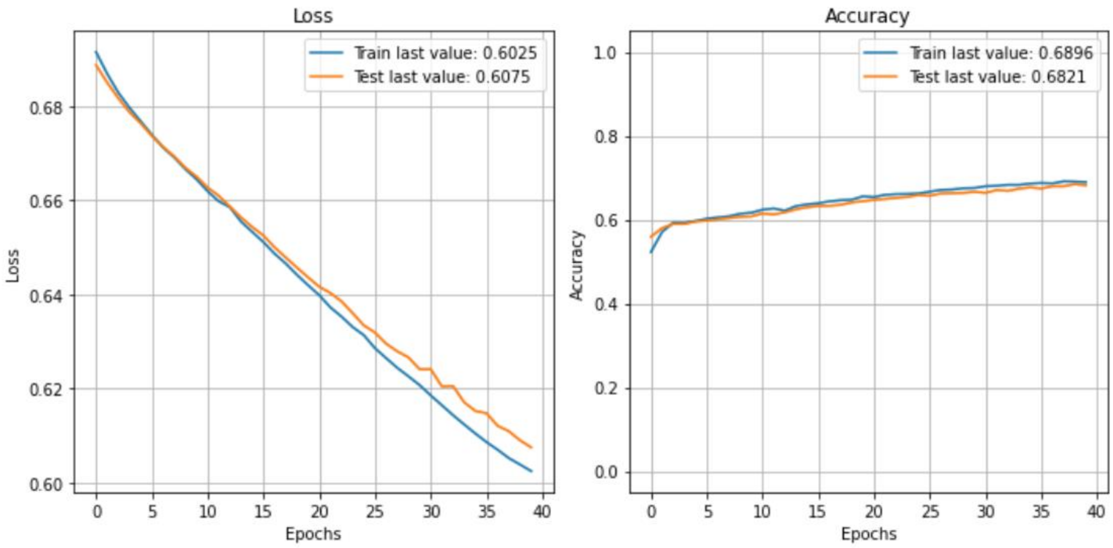
3. model3.summary and plotting the validation and training loss:

With freezing the weights:

```
model = Model (inputs=[input_review], outputs=[label])
model.summary()
```

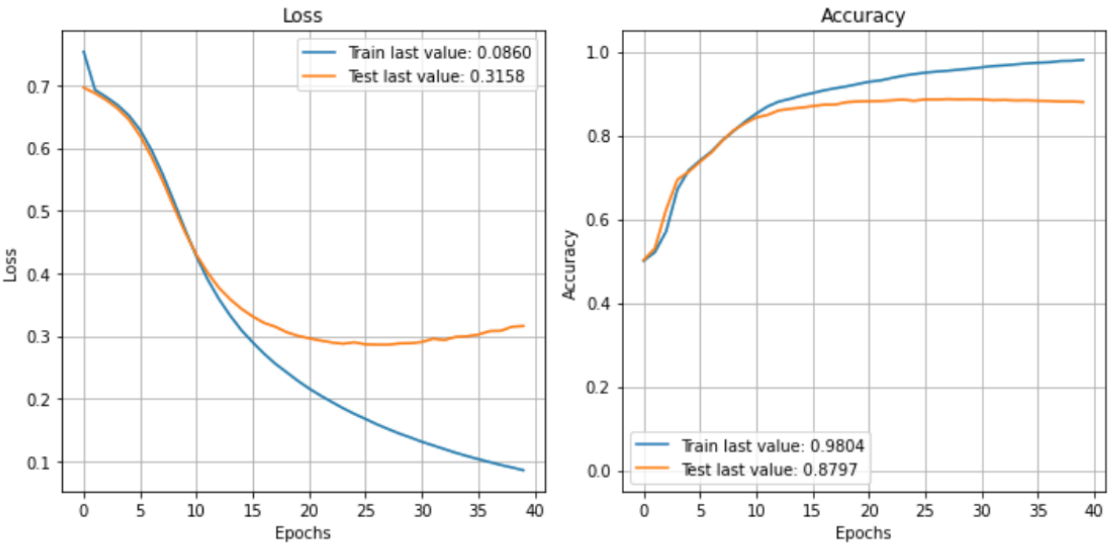
```
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_9 (InputLayer)         [(None, 256)]             0
_________________________________________________________________
GloVe_Embeddings (Embedding) (None, 256, 300)          120000300
_________________________________________________________________
global_average_pooling1d_mas (None, 300)               0
_________________________________________________________________
dense_8 (Dense)              (None, 16)                4816
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 17
=================================================================
Total params: 120,005,133
Trainable params: 4,833
Non-trainable params: 120,000,300
```

```
plt.show()
```



With fine tuning:

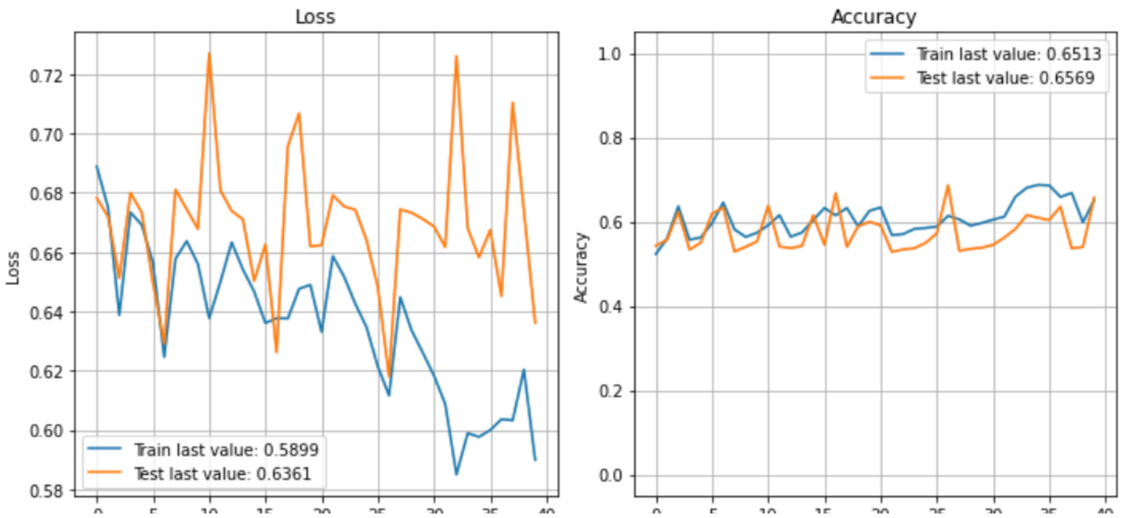
```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 256)]             0
_________________________________________________________________
GloVe_Embeddings (Embedding) (None, 256, 300)          120000300
_________________________________________________________________
global_average_pooling1d_mas (None, 300)               0
_________________________________________________________________
dense_6 (Dense)              (None, 16)                4816
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 17
=================================================================
Total params: 120,005,133
Trainable params: 120,005,133
Non-trainable params: 0
_________________________________________________________________
```



LSTM with pre-trained word embeddings:

```
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 256)]             0
_________________________________________________________________
GloVe_Embeddings (Embedding) (None, 256, 300)          120000300
_________________________________________________________________
lstm (LSTM)                  (None, 100)               160400
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 101
=================================================================
Total params: 120,160,801
Trainable params: 160,501
Non-trainable params: 120,000,300
_________________________________________________________________
```

```
from plot_keras_history import plot_history
import matplotlib.pyplot as plt
plot_history(history.history, path="standard.png")
plt.show()
```

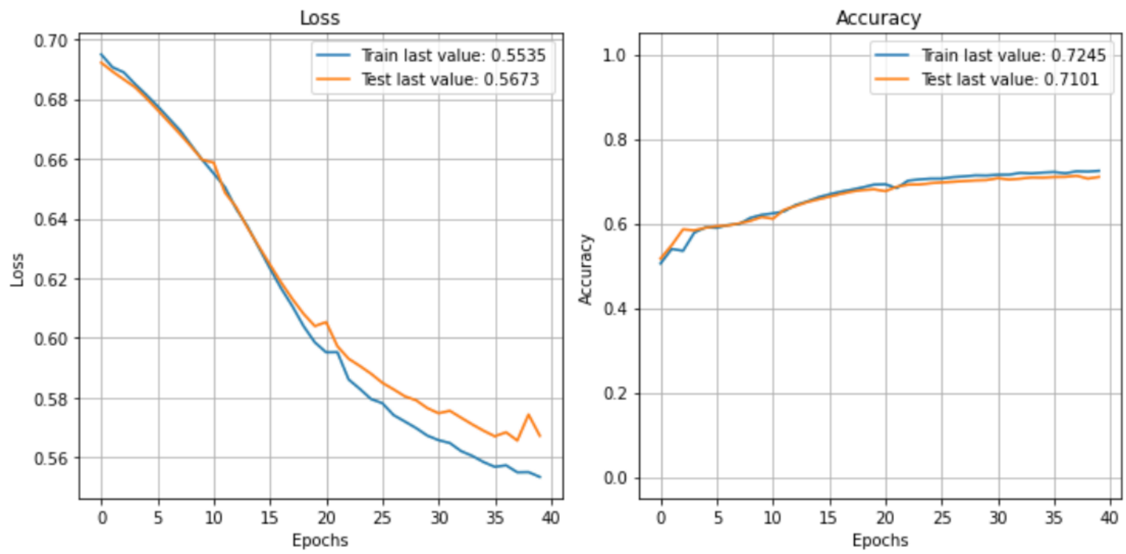


4. **model.summary and plotting of the validation and training loss:**

Adding a dense layer:

```
model4.summary()
```

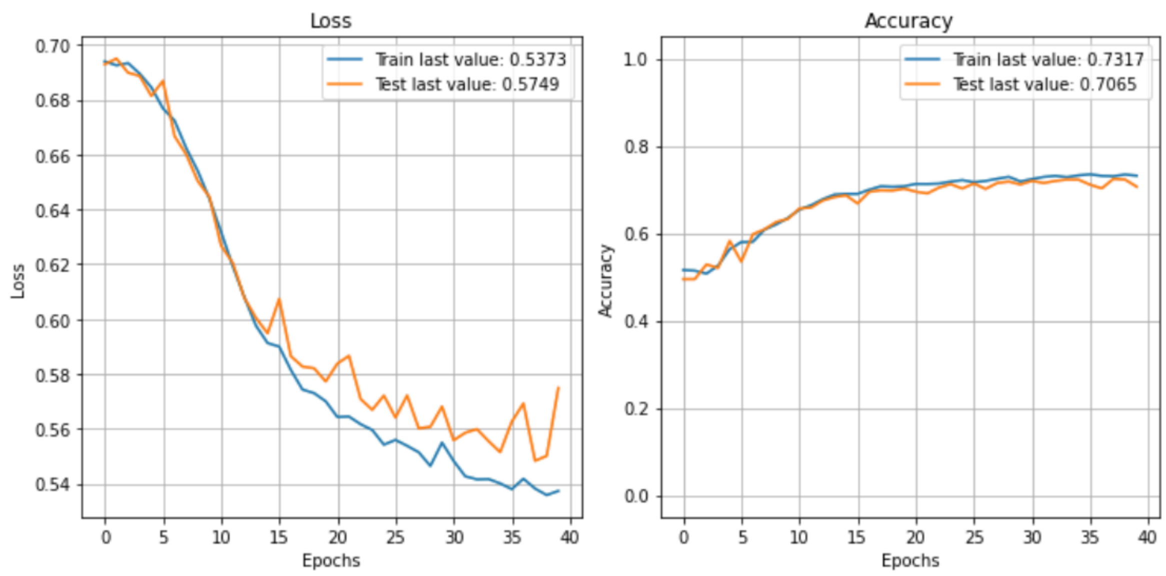
```
Model: "model_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         [(None, 256)]             0
_________________________________________________________________
GloVe_Embeddings (Embedding) (None, 256, 300)          120000300
_________________________________________________________________
global_average_pooling1d_mas (None, 300)               0
_________________________________________________________________
dense_9 (Dense)              (None, 100)               30100
_________________________________________________________________
dense_10 (Dense)             (None, 16)                1616
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 17
=================================================================
Total params: 120,032,033
Trainable params: 31,733
Non-trainable params: 120,000,300
_________________________________________________________________
```



Adding another dense layer:

```
Model: "model_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         [(None, 256)]             0
_________________________________________________________________
GloVe_Embeddings (Embedding) (None, 256, 300)          120000300
_________________________________________________________________
global_average_pooling1d_mas (None, 300)               0
_________________________________________________________________
dense_12 (Dense)             (None, 300)               90300
_________________________________________________________________
dense_13 (Dense)             (None, 100)               30100
_________________________________________________________________
dense_14 (Dense)             (None, 16)                1616
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 17
=================================================================
Total params: 120,122,333
Trainable params: 122,033
Non-trainable params: 120,000,300
_________________________________________________________________
```

```
plt.show()
```



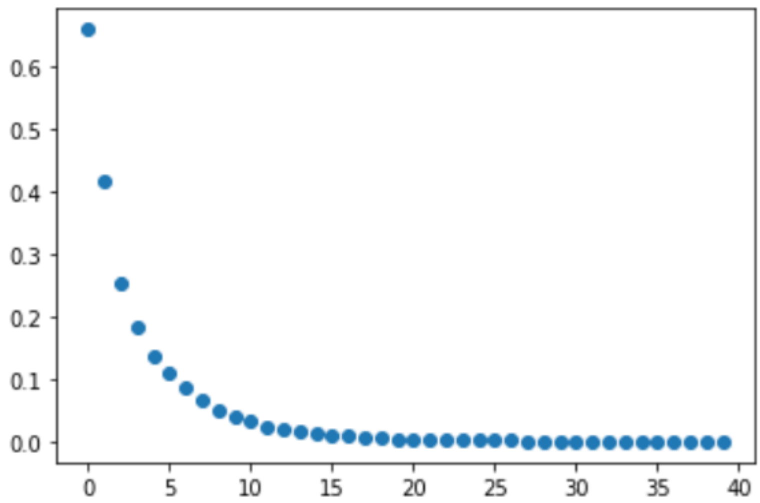
After comparing we can see that there is improvement in performance when adding an extra dense layer. The loss value decreases and accuracy increases.

These two experiments show that adding extra dense layers can slightly improve accuracy over model 3-1.

5. Model.summary and plotting the validation and training loss:

```
Model: "model_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_8 (InputLayer)         [(None, 256)]             0
_________________________________________________________________
embed_layer (Embedding)      (None, 256, 300)          3000000
_________________________________________________________________
conv1d (Conv1D)              (None, 251, 100)          180100
_________________________________________________________________
global_average_pooling1d_mas (None, 100)               0
_________________________________________________________________
dense_16 (Dense)             (None, 1)                 101
=================================================================
Total params: 3,180,201
Trainable params: 3,180,201
Non-trainable params: 0
_________________________________________________________________
```

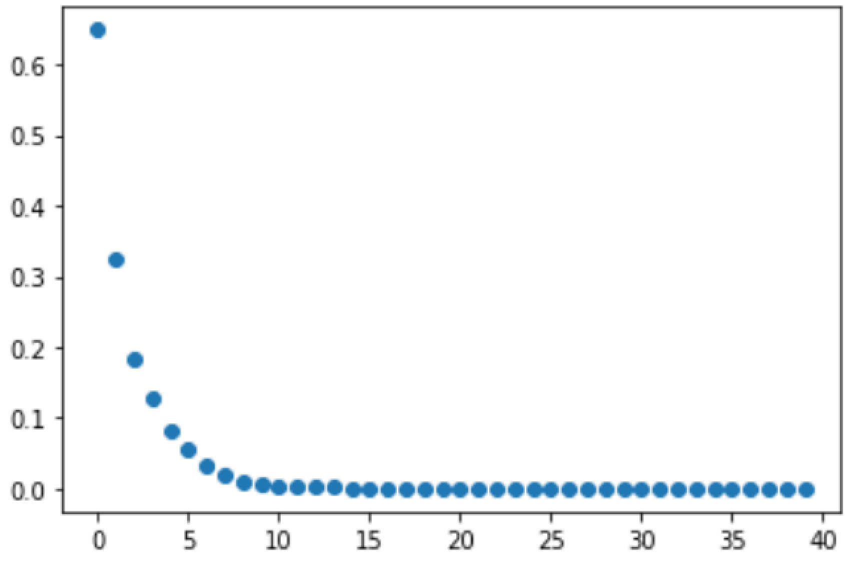
```
# your code goes here
plt.scatter(history.epoch,history.history['loss'])
plt.show()
```



Adding extra convolutional layer:

```
Model: "model_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_9 (InputLayer)         [(None, 256)]             0
_________________________________________________________________
embed_layer (Embedding)      (None, 256, 300)          3000000
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 251, 100)          180100
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 246, 100)          60100
_________________________________________________________________
global_average_pooling1d_mas (None, 100)               0
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 101
=================================================================
Total params: 3,240,301
Trainable params: 3,240,301
Non-trainable params: 0
_________________________________________________________________
```

[49]
```
# your code goes here
plt.scatter(history.epoch,history.history['loss'])
plt.show()
```



I observed that adding an extra convolutional layer here reduces the training loss but the evaluation accuracy is worse than the model without the extra convolutional layer.

Adding more layers can help to extract more features. But we can do that up to a certain extent. After some point, instead of extracting features, we tend to overfit the data. Overfitting can lead to errors in one form or another, such as false positives.

Part D

1. Preprocessing the data:

```
x_dev_tweet[0]:
['jay', 'z', 'joins', 'instagram', 'with', 'nostalgic', 'tribute', 'to', 'michael', 'jackson', 'jay', 'z', 'apparently', 'joined', 'i
x_dev_topic[0]:
['michael', 'jackson']
x_dev_tweet_int[0]:
[7269, 7022, 1054, 25880, 24821, 28295, 25462, 16889, 15313, 7882, 7269, 7022, 25999, 14715, 25880, 25333, 16110, 4229, 21320, 21173,
x_dev_topic_int[0]:
[15313, 7882]
```

```
Before paded:
['microsoft', '<START>', 'dear', 'microsoft', 'the', 'newooffice', 'for', 'mac', 'is', 'great', 'and', 'all', 'but', '
[31463, 1, 3310, 31463, 29114, 6170, 20972, 9602, 30518, 5879, 4229, 18197, 5883, 13993, 8537, 19463, 16854]
After paded:
[31463     1  3310 31463 29114  6170 20972  9602 30518  5879  4229 18197
  5883 13993  8537 19463 16854     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0]
```

2. model.summary() command to show your model structure, and training epochs, and evaluation results:

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 128)]             0
_________________________________________________________________
target_embed_layer (Embeddin (None, 128, 100)          3257500
_________________________________________________________________
global_average_pooling1d_mas (None, 100)               0
_________________________________________________________________
dense (Dense)                (None, 16)                1616
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 17
=================================================================
Total params: 3,259,133
Trainable params: 3,259,133
Non-trainable params: 0
_________________________________________________________________
```

Epochs:

```
Epoch 1/30
35/35 [==============================] - 2s 37ms/step - loss: 0.5517 - accuracy: 0.7895 - val_loss: 0.5680 - val_accuracy: 0.7442
Epoch 2/30
35/35 [==============================] - 1s 27ms/step - loss: 0.5165 - accuracy: 0.7876 - val_loss: 0.5748 - val_accuracy: 0.7442
Epoch 3/30
35/35 [==============================] - 1s 27ms/step - loss: 0.5043 - accuracy: 0.7943 - val_loss: 0.5719 - val_accuracy: 0.7442
Epoch 4/30
35/35 [==============================] - 1s 27ms/step - loss: 0.5107 - accuracy: 0.7875 - val_loss: 0.5714 - val_accuracy: 0.7442
Epoch 5/30
35/35 [==============================] - 1s 27ms/step - loss: 0.4984 - accuracy: 0.7937 - val_loss: 0.5657 - val_accuracy: 0.7442
Epoch 6/30
35/35 [==============================] - 1s 27ms/step - loss: 0.4940 - accuracy: 0.7920 - val_loss: 0.5593 - val_accuracy: 0.7442
Epoch 7/30
35/35 [==============================] - 1s 27ms/step - loss: 0.4883 - accuracy: 0.7880 - val_loss: 0.5515 - val_accuracy: 0.7442
Epoch 8/30
35/35 [==============================] - 1s 27ms/step - loss: 0.4723 - accuracy: 0.7874 - val_loss: 0.5390 - val_accuracy: 0.7442
Epoch 9/30
35/35 [==============================] - 1s 27ms/step - loss: 0.4450 - accuracy: 0.7917 - val_loss: 0.5250 - val_accuracy: 0.7442
Epoch 10/30
35/35 [==============================] - 1s 28ms/step - loss: 0.4147 - accuracy: 0.7946 - val_loss: 0.5047 - val_accuracy: 0.7449
Epoch 11/30
35/35 [==============================] - 1s 27ms/step - loss: 0.3888 - accuracy: 0.8068 - val_loss: 0.4889 - val_accuracy: 0.7570
Epoch 12/30
35/35 [==============================] - 1s 26ms/step - loss: 0.3552 - accuracy: 0.8378 - val_loss: 0.4778 - val_accuracy: 0.7570
```

Evaluation:

```
model.evaluate(x_test_pad,y_test)
```

```
194/194 [==============================] - 0s 2ms/step - loss: 0.5256 - accuracy: 0.7450
[0.5256087779998779, 0.7450283169746399]
```

CNN or LSTM without pre-trained word embeddings:

Summary:

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 128)]             0
_________________________________________________________________
embed_layer (Embedding)      (None, 128, 100)          3257500
_________________________________________________________________
conv1d (Conv1D)              (None, 123, 100)          60100
_________________________________________________________________
global_average_pooling1d (Gl (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 16)                1616
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 17
=================================================================
Total params: 3,319,233
Trainable params: 3,319,233
Non-trainable params: 0
_________________________________________________________________
```

```
35/35 [==============================] - 1s 41ms/step - loss: 0.0192 - accuracy: 0.9981 - val_loss: 0.8725 - val_accuracy: 0.7562
Epoch 20/30
35/35 [==============================] - 1s 40ms/step - loss: 0.0174 - accuracy: 0.9983 - val_loss: 0.8923 - val_accuracy: 0.7434
Epoch 21/30
35/35 [==============================] - 1s 42ms/step - loss: 0.0167 - accuracy: 0.9984 - val_loss: 0.9173 - val_accuracy: 0.7608
Epoch 22/30
35/35 [==============================] - 1s 42ms/step - loss: 0.0154 - accuracy: 0.9985 - val_loss: 0.9253 - val_accuracy: 0.7449
Epoch 23/30
35/35 [==============================] - 1s 41ms/step - loss: 0.0138 - accuracy: 0.9989 - val_loss: 0.9512 - val_accuracy: 0.7434
Epoch 24/30
35/35 [==============================] - 1s 42ms/step - loss: 0.0151 - accuracy: 0.9986 - val_loss: 0.9591 - val_accuracy: 0.7457
Epoch 25/30
35/35 [==============================] - 1s 41ms/step - loss: 0.0129 - accuracy: 0.9986 - val_loss: 0.9781 - val_accuracy: 0.7374
Epoch 26/30
35/35 [==============================] - 1s 41ms/step - loss: 0.0141 - accuracy: 0.9984 - val_loss: 0.9967 - val_accuracy: 0.7366
Epoch 27/30
35/35 [==============================] - 1s 42ms/step - loss: 0.0128 - accuracy: 0.9987 - val_loss: 1.0107 - val_accuracy: 0.7419
Epoch 28/30
35/35 [==============================] - 1s 42ms/step - loss: 0.0126 - accuracy: 0.9986 - val_loss: 1.0149 - val_accuracy: 0.7502
Epoch 29/30
35/35 [==============================] - 1s 41ms/step - loss: 0.0122 - accuracy: 0.9986 - val_loss: 1.0316 - val_accuracy: 0.7411
Epoch 30/30
35/35 [==============================] - 1s 41ms/step - loss: 0.0105 - accuracy: 0.9990 - val_loss: 1.0561 - val_accuracy: 0.7358
```

Evaluation:

```
model2.evaluate(x_test_pad,y_test)
```

```
194/194 [==============================] - 1s 3ms/step - loss: 1.0615 - accuracy: 0.7258
[1.0615345239639282, 0.7257881760597229]
```

3. model.summary, epochs and evaluation for Model 2:

Using pre-trained word embeddings:

Preprocess Part:

```
x_dev_tweet_glove[0]:
[196237, 395262, 198486, 190716, 388711, 264529, 365027, 360915, 242891, 194733,
x_dev_topic_glove[0]:
[242891, 194733]
```

```
Before paded:
[243317, 1, 118309, 243317, 357266, 2, 151349, 229153, 192973, 166369, 54718, 51582, 87775, 262350, 228306, 373375, 88558]
After paded:
[243317      1 118309 243317 357266      2 151349 229153 192973 166369
  54718  51582  87775 262350 228306 373375  88558      0      0      0
      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0]
```

Summary:

```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 128)]             0
_________________________________________________________________
GloVe_Embeddings (Embedding) (None, 128, 300)          120000300
_________________________________________________________________
global_average_pooling1d_mas (None, 300)               0
_________________________________________________________________
dense_8 (Dense)              (None, 16)                4816
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 17
=================================================================
Total params: 120,005,133
Trainable params: 4,833
Non-trainable params: 120,000,300
_________________________________________________________________
```

Epochs:

```
Epoch 189/200
35/35 [==============================] - 1s 23ms/step - loss: 0.3139 - accuracy: 0.8689 - val_loss: 0.4025 - val_accuracy: 0.8181
Epoch 190/200
35/35 [==============================] - 1s 23ms/step - loss: 0.3088 - accuracy: 0.8736 - val_loss: 0.4022 - val_accuracy: 0.8219
Epoch 191/200
35/35 [==============================] - 1s 23ms/step - loss: 0.3077 - accuracy: 0.8731 - val_loss: 0.4024 - val_accuracy: 0.8196
Epoch 192/200
35/35 [==============================] - 1s 24ms/step - loss: 0.3101 - accuracy: 0.8721 - val_loss: 0.4024 - val_accuracy: 0.8189
Epoch 193/200
35/35 [==============================] - 1s 23ms/step - loss: 0.3052 - accuracy: 0.8738 - val_loss: 0.4024 - val_accuracy: 0.8189
Epoch 194/200
35/35 [==============================] - 1s 23ms/step - loss: 0.3058 - accuracy: 0.8705 - val_loss: 0.4024 - val_accuracy: 0.8189
Epoch 195/200
35/35 [==============================] - 1s 24ms/step - loss: 0.3012 - accuracy: 0.8735 - val_loss: 0.4026 - val_accuracy: 0.8219
Epoch 196/200
35/35 [==============================] - 1s 23ms/step - loss: 0.3046 - accuracy: 0.8711 - val_loss: 0.4026 - val_accuracy: 0.8219
Epoch 197/200
35/35 [==============================] - 1s 23ms/step - loss: 0.3050 - accuracy: 0.8732 - val_loss: 0.4027 - val_accuracy: 0.8204
Epoch 198/200
35/35 [==============================] - 1s 23ms/step - loss: 0.3050 - accuracy: 0.8713 - val_loss: 0.4033 - val_accuracy: 0.8189
Epoch 199/200
35/35 [==============================] - 1s 23ms/step - loss: 0.3082 - accuracy: 0.8716 - val_loss: 0.4025 - val_accuracy: 0.8211
Epoch 200/200
35/35 [==============================] - 1s 24ms/step - loss: 0.3023 - accuracy: 0.8724 - val_loss: 0.4025 - val_accuracy: 0.8189
```

Evaluation:

```
results2 = model3.evaluate(x_test_pad_glove, y_test)
```

```
194/194 [==============================] - 1s 3ms/step - loss: 0.4560 - accuracy: 0.7888
```

Model 2-2: CNN or LSTM with pre-trained word embeddings:

Summary:

```
Model: "model_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_21 (InputLayer)        [(None, 128)]             0
_________________________________________________________________
GloVe_Embeddings (Embedding) (None, 128, 300)          120000300
_________________________________________________________________
conv1d_11 (Conv1D)           (None, 123, 100)          180100
_________________________________________________________________
global_average_pooling1d_11  (None, 100)               0
_________________________________________________________________
dense_28 (Dense)             (None, 1)                 101
=================================================================
Total params: 120,180,501
Trainable params: 180,201
Non-trainable params: 120,000,300
_________________________________________________________________
```

Epochs:

```
Epoch 189/200
35/35 [==============================] - 1s 32ms/step - loss: 0.2687 - accuracy: 0.8897 - val_loss: 0.4316 - val_accuracy: 0.8143
Epoch 190/200
35/35 [==============================] - 1s 32ms/step - loss: 0.2635 - accuracy: 0.8921 - val_loss: 0.4436 - val_accuracy: 0.8098
Epoch 191/200
35/35 [==============================] - 1s 32ms/step - loss: 0.2690 - accuracy: 0.8858 - val_loss: 0.4364 - val_accuracy: 0.8008
Epoch 192/200
35/35 [==============================] - 1s 32ms/step - loss: 0.2664 - accuracy: 0.8918 - val_loss: 0.4351 - val_accuracy: 0.8113
Epoch 193/200
35/35 [==============================] - 1s 32ms/step - loss: 0.2576 - accuracy: 0.8954 - val_loss: 0.4360 - val_accuracy: 0.8023
Epoch 194/200
35/35 [==============================] - 1s 32ms/step - loss: 0.2630 - accuracy: 0.8933 - val_loss: 0.4350 - val_accuracy: 0.8008
Epoch 195/200
35/35 [==============================] - 1s 32ms/step - loss: 0.2727 - accuracy: 0.8887 - val_loss: 0.4313 - val_accuracy: 0.8106
Epoch 196/200
35/35 [==============================] - 1s 32ms/step - loss: 0.2696 - accuracy: 0.8883 - val_loss: 0.4379 - val_accuracy: 0.7970
Epoch 197/200
35/35 [==============================] - 1s 32ms/step - loss: 0.2674 - accuracy: 0.8896 - val_loss: 0.4376 - val_accuracy: 0.7992
Epoch 198/200
35/35 [==============================] - 1s 32ms/step - loss: 0.2611 - accuracy: 0.8919 - val_loss: 0.4356 - val_accuracy: 0.8053
Epoch 199/200
35/35 [==============================] - 1s 32ms/step - loss: 0.2632 - accuracy: 0.8920 - val_loss: 0.4379 - val_accuracy: 0.8075
Epoch 200/200
35/35 [==============================] - 1s 32ms/step - loss: 0.2630 - accuracy: 0.8941 - val_loss: 0.4512 - val_accuracy: 0.7857
```

Evaluation:

```
results_4 = model4.evaluate(x_test_pad_glove, y_test)
print(results_4)
```

```
194/194 [==============================] - 1s 3ms/step - loss: 0.4661 - accuracy: 0.7900
[0.4660947024822235, 0.7899757623672485]
```

4. model.summary() command to show your model structure, and training epochs, and evaluation results:

Neural bag of words model with multiple-input:

Summary:

```
Model: "model_6"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_10 (InputLayer)           [(None, 16)]         0
__________________________________________________________________________________________________
input_11 (InputLayer)           [(None, 128)]        0
__________________________________________________________________________________________________
GloVe_Embeddings (Embedding)    multiple             120000300   input_10[0][0]
                                                                 input_11[0][0]
__________________________________________________________________________________________________
global_average_pooling1d_masked (None, 300)          0           GloVe_Embeddings[6][0]
__________________________________________________________________________________________________
global_average_pooling1d_masked (None, 300)          0           GloVe_Embeddings[7][0]
__________________________________________________________________________________________________
dense_14 (Dense)                (None, 16)           4816        global_average_pooling1d_masked_6
__________________________________________________________________________________________________
dense_15 (Dense)                (None, 16)           4816        global_average_pooling1d_masked_7
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 32)           0           dense_14[0][0]
                                                                 dense_15[0][0]
__________________________________________________________________________________________________
dense_16 (Dense)                (None, 1)            33          concatenate_1[0][0]
==================================================================================================
Total params: 120,009,965
Trainable params: 9,665
Non-trainable params: 120,000,300
```

Epochs:

```
Epoch 188/200
35/35 [==============================] - 1s 25ms/step - loss: 0.2705 - accuracy: 0.8879 - val_loss: 0.4602 - val_accuracy: 0.7789
Epoch 189/200
35/35 [==============================] - 1s 25ms/step - loss: 0.2675 - accuracy: 0.8885 - val_loss: 0.4603 - val_accuracy: 0.7804
Epoch 190/200
35/35 [==============================] - 1s 25ms/step - loss: 0.2769 - accuracy: 0.8860 - val_loss: 0.4602 - val_accuracy: 0.7811
Epoch 191/200
35/35 [==============================] - 1s 25ms/step - loss: 0.2729 - accuracy: 0.8875 - val_loss: 0.4624 - val_accuracy: 0.7766
Epoch 192/200
35/35 [==============================] - 1s 25ms/step - loss: 0.2771 - accuracy: 0.8851 - val_loss: 0.4628 - val_accuracy: 0.7789
Epoch 193/200
35/35 [==============================] - 1s 25ms/step - loss: 0.2692 - accuracy: 0.8890 - val_loss: 0.4600 - val_accuracy: 0.7804
Epoch 194/200
35/35 [==============================] - 1s 25ms/step - loss: 0.2736 - accuracy: 0.8878 - val_loss: 0.4623 - val_accuracy: 0.7774
Epoch 195/200
35/35 [==============================] - 1s 25ms/step - loss: 0.2768 - accuracy: 0.8859 - val_loss: 0.4607 - val_accuracy: 0.7789
Epoch 196/200
35/35 [==============================] - 1s 25ms/step - loss: 0.2752 - accuracy: 0.8859 - val_loss: 0.4624 - val_accuracy: 0.7774
Epoch 197/200
35/35 [==============================] - 1s 25ms/step - loss: 0.2814 - accuracy: 0.8855 - val_loss: 0.4606 - val_accuracy: 0.7804
Epoch 198/200
35/35 [==============================] - 1s 25ms/step - loss: 0.2802 - accuracy: 0.8810 - val_loss: 0.4628 - val_accuracy: 0.7781
Epoch 199/200
35/35 [==============================] - 1s 25ms/step - loss: 0.2753 - accuracy: 0.8859 - val_loss: 0.4639 - val_accuracy: 0.7766
Epoch 200/200
35/35 [==============================] - 1s 25ms/step - loss: 0.2738 - accuracy: 0.8867 - val_loss: 0.4640 - val_accuracy: 0.7774
```

Evaluation:

```python
results_5 = model_5.evaluate([x_test_topic_pad_glove, x_test_tweet_pad_glove], y_test)
print(results_5)
```

```
194/194 [==============================] - 1s 3ms/step - loss: 0.5301 - accuracy: 0.7639
[0.5300872325897217, 0.7639450430870056]
```

Model 3-2: **CNN or LSTM model with multiple-input**

Summary:

```
Model: "model_7"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_12 (InputLayer)           [(None, 16)]         0
__________________________________________________________________________________________________
input_13 (InputLayer)           [(None, 128)]        0
__________________________________________________________________________________________________
GloVe_Embeddings (Embedding)    multiple             120000300   input_12[0][0]
                                                                 input_13[0][0]
__________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 11, 100)      180100      GloVe_Embeddings[8][0]
__________________________________________________________________________________________________
conv1d_3 (Conv1D)               (None, 123, 100)     180100      GloVe_Embeddings[9][0]
__________________________________________________________________________________________________
global_average_pooling1d_2 (Glo (None, 100)          0           conv1d_2[0][0]
__________________________________________________________________________________________________
global_average_pooling1d_3 (Glo (None, 100)          0           conv1d_3[0][0]
__________________________________________________________________________________________________
dense_17 (Dense)                (None, 16)           1616        global_average_pooling1d_2[0][0]
__________________________________________________________________________________________________
dense_18 (Dense)                (None, 16)           1616        global_average_pooling1d_3[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 32)           0           dense_17[0][0]
                                                                 dense_18[0][0]
__________________________________________________________________________________________________
dense_19 (Dense)                (None, 1)            33          concatenate_2[0][0]
==================================================================================================
Total params: 120,363,765
Trainable params: 363,465
Non-trainable params: 120,000,300
__________________________________________________________________________________________________
```

Epochs:

```
Epoch 188/200
35/35 [==============================] - 1s 35ms/step - loss: 0.2047 - accuracy: 0.9199 - val_loss: 0.5526 - val_accuracy: 0.7638
Epoch 189/200
35/35 [==============================] - 1s 36ms/step - loss: 0.2050 - accuracy: 0.9194 - val_loss: 0.5362 - val_accuracy: 0.7811
Epoch 190/200
35/35 [==============================] - 1s 36ms/step - loss: 0.2094 - accuracy: 0.9191 - val_loss: 0.5465 - val_accuracy: 0.7721
Epoch 191/200
35/35 [==============================] - 1s 35ms/step - loss: 0.2071 - accuracy: 0.9174 - val_loss: 0.5466 - val_accuracy: 0.7766
Epoch 192/200
35/35 [==============================] - 1s 35ms/step - loss: 0.2076 - accuracy: 0.9170 - val_loss: 0.5623 - val_accuracy: 0.7736
Epoch 193/200
35/35 [==============================] - 1s 35ms/step - loss: 0.2117 - accuracy: 0.9161 - val_loss: 0.5523 - val_accuracy: 0.7736
Epoch 194/200
35/35 [==============================] - 1s 36ms/step - loss: 0.1976 - accuracy: 0.9225 - val_loss: 0.5572 - val_accuracy: 0.7698
Epoch 195/200
35/35 [==============================] - 1s 35ms/step - loss: 0.2091 - accuracy: 0.9159 - val_loss: 0.5506 - val_accuracy: 0.7743
Epoch 196/200
35/35 [==============================] - 1s 36ms/step - loss: 0.2010 - accuracy: 0.9231 - val_loss: 0.5568 - val_accuracy: 0.7758
Epoch 197/200
35/35 [==============================] - 1s 35ms/step - loss: 0.2066 - accuracy: 0.9154 - val_loss: 0.5556 - val_accuracy: 0.7736
Epoch 198/200
35/35 [==============================] - 1s 36ms/step - loss: 0.2011 - accuracy: 0.9218 - val_loss: 0.5596 - val_accuracy: 0.7706
Epoch 199/200
35/35 [==============================] - 1s 36ms/step - loss: 0.2000 - accuracy: 0.9228 - val_loss: 0.5623 - val_accuracy: 0.7713
Epoch 200/200
35/35 [==============================] - 1s 35ms/step - loss: 0.2045 - accuracy: 0.9210 - val_loss: 0.5621 - val_accuracy: 0.7691
```

Evaluation:

```
results_6 = model_6.evaluate([x_test_topic_pad_glove, x_test_tweet_pad_glove], y_test)
print(results_6)
```

```
194/194 [==============================] - 1s 3ms/step - loss: 0.7018 - accuracy: 0.7378
[0.7018463015556335, 0.7377526164054871]
```

5. **Try to improve your overall accuracy over 81% on this task. You could try one or more of: using a CNN instead; using a different classifier; modify embeddings, loss function etc - see ideas in Week 5 & 6 lectures. Please discuss your model and results in your report. [6 marks].**

Summary of the best model:

```
Model: "model_3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            [(None, 16)]         0
__________________________________________________________________________________________________
input_4 (InputLayer)            [(None, 128)]        0
__________________________________________________________________________________________________
GloVe_Embeddings (Embedding)    multiple             120000300   input_3[0][0]
                                                                 input_4[0][0]
__________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 11, 100)      180100      GloVe_Embeddings[2][0]
__________________________________________________________________________________________________
conv1d_3 (Conv1D)               (None, 123, 100)     180100      GloVe_Embeddings[3][0]
__________________________________________________________________________________________________
global_max_pooling1d (GlobalMax (None, 100)          0           conv1d_2[0][0]
__________________________________________________________________________________________________
global_max_pooling1d_1 (GlobalM (None, 100)          0           conv1d_3[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 16)           1616        global_max_pooling1d[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 16)           1616        global_max_pooling1d_1[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 32)           0           dense_5[0][0]
                                                                 dense_6[0][0]
__________________________________________________________________________________________________
dense_7 (Dense)                 (None, 1)            33          concatenate_1[0][0]
--------------------------------------------------------------------------------------------------
==================================================================================================
Total params: 120,363,765
Trainable params: 363,465
Non-trainable params: 120,000,300
__________________________________________________________________________________________________
```

Epochs:

```
Epoch 188/200
35/35 [==============================] - 22s 637ms/step - loss: 0.0018 - accuracy: 0.9982 - val_loss: 0.1705 - val_accuracy: 0.8121
Epoch 189/200
35/35 [==============================] - 22s 640ms/step - loss: 0.0022 - accuracy: 0.9978 - val_loss: 0.1705 - val_accuracy: 0.8121
Epoch 190/200
35/35 [==============================] - 22s 641ms/step - loss: 0.0024 - accuracy: 0.9976 - val_loss: 0.1706 - val_accuracy: 0.8113
Epoch 191/200
35/35 [==============================] - 23s 644ms/step - loss: 0.0024 - accuracy: 0.9976 - val_loss: 0.1705 - val_accuracy: 0.8128
Epoch 192/200
35/35 [==============================] - 23s 644ms/step - loss: 0.0019 - accuracy: 0.9981 - val_loss: 0.1705 - val_accuracy: 0.8128
Epoch 193/200
35/35 [==============================] - 23s 646ms/step - loss: 0.0016 - accuracy: 0.9984 - val_loss: 0.1706 - val_accuracy: 0.8113
Epoch 194/200
35/35 [==============================] - 23s 644ms/step - loss: 0.0021 - accuracy: 0.9979 - val_loss: 0.1706 - val_accuracy: 0.8128
Epoch 195/200
35/35 [==============================] - 22s 641ms/step - loss: 0.0017 - accuracy: 0.9983 - val_loss: 0.1706 - val_accuracy: 0.8128
Epoch 196/200
35/35 [==============================] - 23s 645ms/step - loss: 0.0021 - accuracy: 0.9979 - val_loss: 0.1705 - val_accuracy: 0.8128
Epoch 197/200
35/35 [==============================] - 22s 640ms/step - loss: 0.0014 - accuracy: 0.9986 - val_loss: 0.1706 - val_accuracy: 0.8128
Epoch 198/200
35/35 [==============================] - 23s 643ms/step - loss: 0.0020 - accuracy: 0.9980 - val_loss: 0.1706 - val_accuracy: 0.8113
Epoch 199/200
35/35 [==============================] - 22s 638ms/step - loss: 0.0019 - accuracy: 0.9981 - val_loss: 0.1706 - val_accuracy: 0.8128
Epoch 200/200
35/35 [==============================] - 23s 646ms/step - loss: 0.0015 - accuracy: 0.9985 - val_loss: 0.1706 - val_accuracy: 0.8128
```

Evaluation:

```
results_7 = final_model.evaluate([x_test_topic_pad_glove, x_test_tweet_pad_glove], y_test)
print(results_7)
```

```
194/194 [==============================] - 4s 19ms/step - loss: 0.1724 - accuracy: 0.8087
[0.17236223816871643, 0.8087307810783386]
```