Text and Sequence Data Assignment

Report by Sudarshan Rayapati

Introduction: Recurrent Neural Networks (RNNs) are widely used for sequence data processing tasks such as sentiment analysis, where input sequences represent sentences or texts. In this report, I evaluate the performance of RNN models on the IMDB sentiment analysis dataset. Specifically, I compared RNN models trained from scratch with those using pretrained word embeddings.

Data Preparation: I used the IMDB dataset, consisting of movie reviews labeled as positive or negative. The dataset was split into training, validation, and test sets. For the purpose of this report, I used different sizes of training data: 100, 500, and 1000 samples.

Model Implementation: I built RNN models with and without pretrained word embeddings.

The RNN architecture consists of an input layer, an embedding layer to convert words into dense vectors, an RNN layer with LSTM cells for sequence processing, and a dense output layer for sentiment classification.

```
RNN Model Architecture :
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 150, 32)           320000

 simple_rnn_4 (SimpleRNN)    (None, 16)                784

 dense_4 (Dense)             (None, 1)                 17

 activation_2 (Activation)   (None, 1)                 0

=================================================================
Total params: 320801 (1.22 MB)
Trainable params: 320801 (1.22 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

The pretrained word embeddings Ire loaded from a pre-trained word embedding model, such as Word2Vec or GloVe, which captures semantic meanings of words. The RNN architecture includes an embedding layer initialized with pretrained word embeddings, an RNN layer with LSTM cells, and a dense output layer.

```
RNN Model Architecture with Pretrained Embeddings :
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_5 (Embedding)     (None, 150, 100)          1000000

 simple_rnn_5 (SimpleRNN)    (None, 16)                1872

 dense_5 (Dense)             (None, 1)                 17

=================================================================
Total params: 1001889 (3.82 MB)
Trainable params: 1889 (7.38 KB)
Non-trainable params: 1000000 (3.81 MB)
_________________________________________________________________
None
```

Results:

	RNN Accuracies	Pretrained RNN Accuracies	RNN Losses	Pretrained RNN Losses
100	0.5170	0.5192	0.692560	0.750598
500	0.5554	0.5088	0.685657	0.708162
1000	0.7144	0.5302	0.737485	0.722393

Conclusion:

Training Data Size: Larger training datasets generally lead to better performance, as observed in both models.

Pretrained Embeddings: Although pretrained word embeddings shoId minimal improvement in some cases, they didn't consistently enhance the model's performance. Further fine-tuning or using different embeddings might yield better results.

Data Preparation

In [1]:
```
from keras.datasets import imdb

# Load the IMDB dataset
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

In [2]:
```
from keras.preprocessing.sequence import pad_sequences

# Truncate or pad the reviews to a length of 150 words
maxlen = 150
train_data = pad_sequences(train_data, maxlen=maxlen)
test_data = pad_sequences(test_data, maxlen=maxlen)

# Select 5000 samples for testing
test_data = test_data[:5000]
test_labels = test_labels[:5000]

# Select 10,000 samples for validation
val_data = test_data[:10000]
val_labels = test_labels[:10000]
```

For 100 Training Samples

In [3]:
```
# Select the first 100 samples for training
train_data_100 = train_data[:100]
train_labels_100 = train_labels[:100]
```

Model Implementation

RNN

In [4]:
```
from keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense,Activation

# Build The RNN model
rnn_model_100 = Sequential()

rnn_model_100.add(Embedding(10000,32,input_length =len(train_data_100[0])))
rnn_model_100.add(SimpleRNN(16,input_shape = (10000,maxlen), return_sequences=False,ac
rnn_model_100.add(Dense(1)) #flatten
rnn_model_100.add(Activation("sigmoid")) #using sigmoid for binary classification
rnn_model_100.compile(loss="binary_crossentropy",optimizer="rmsprop",metrics=["accurac

print(" ")
print("RNN Model Architecture : ")
print(rnn_model_100.summary())
print(" ")
# Train the RNN model
rnn_history_100 = rnn_model_100.fit(train_data_100, train_labels_100, epochs=10, batch
```

```
# Evaluate the model
rnn100_test_loss, rnn100_test_accuracy = rnn_model_100.evaluate(test_data, test_labels

print("Test Loss : ", rnn100_test_loss)
print("Test Accuracy : ", rnn100_test_accuracy)


#Model Perfomance Evaluation
import matplotlib.pyplot as plt

print(" ")
print("Perfomance of RNN Model for 100 Training Samples : ")
print(" ")
# Plot training and validation accuracy
print("Accuracy : ")
print(" ")
plt.plot(rnn_history_100.history['accuracy'], label='Training Accuracy')
plt.plot(rnn_history_100.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
print(" ")
print("Loss : ")
print(" ")
plt.plot(rnn_history_100.history['loss'], label='Training Loss')
plt.plot(rnn_history_100.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

RNN Model Architecture :
Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 150, 32)	320000
simple_rnn (SimpleRNN)	(None, 16)	784
dense (Dense)	(None, 1)	17
activation (Activation)	(None, 1)	0

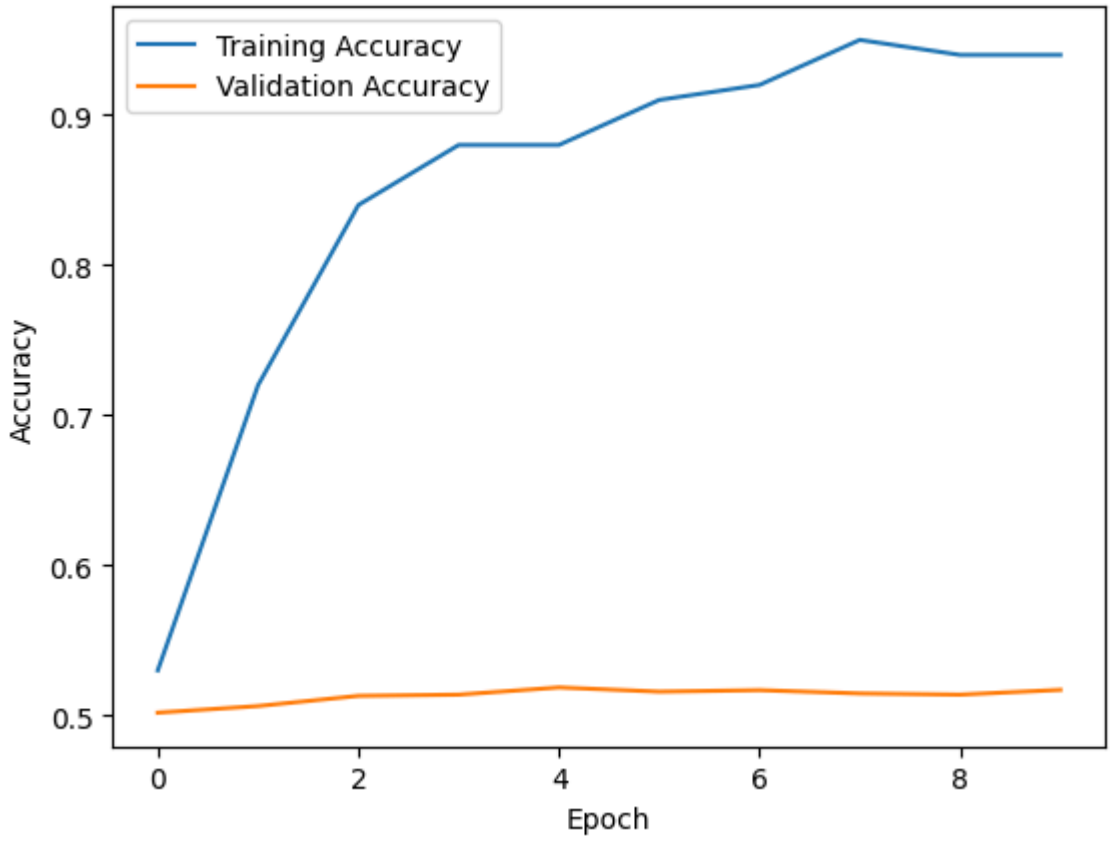
Total params: 320801 (1.22 MB)
Trainable params: 320801 (1.22 MB)
Non-trainable params: 0 (0.00 Byte)

None

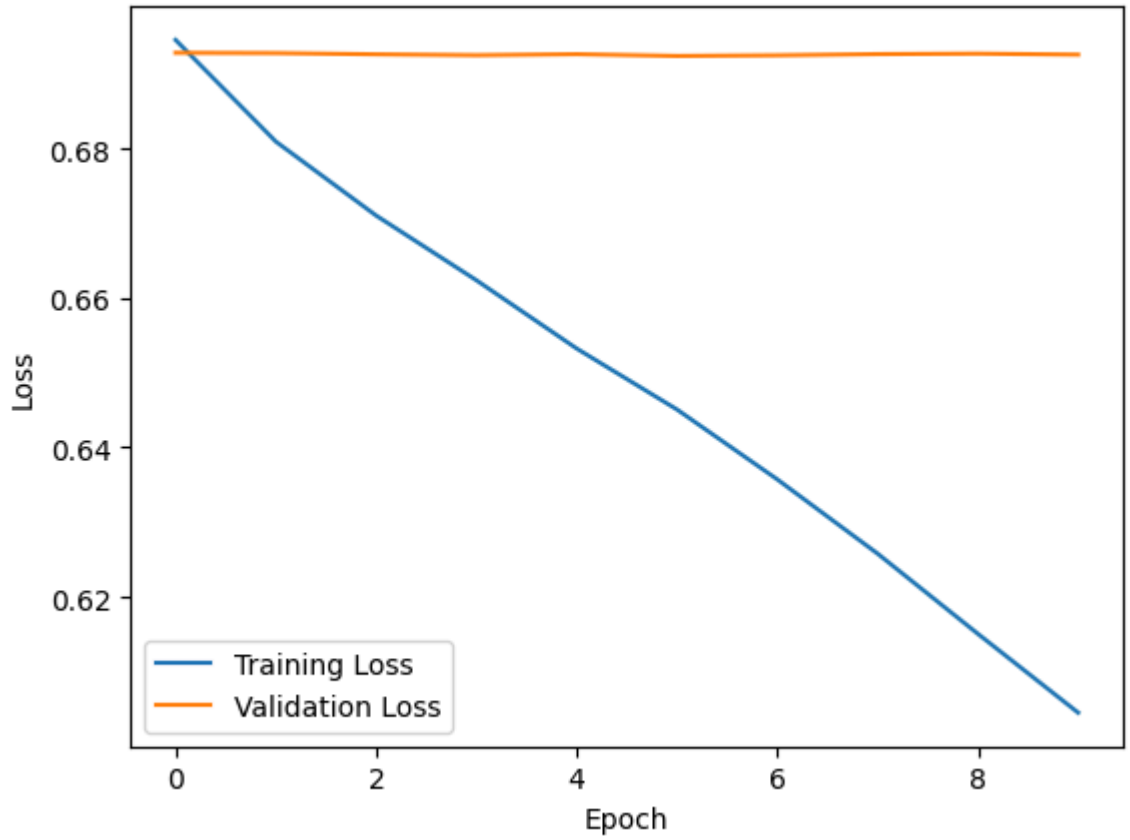
```
Epoch 1/10
4/4 [==============================] - 16s 2s/step - loss: 0.6945 - accuracy: 0.5300
- val_loss: 0.6928 - val_accuracy: 0.5018
Epoch 2/10
4/4 [==============================] - 3s 864ms/step - loss: 0.6809 - accuracy: 0.720
0 - val_loss: 0.6928 - val_accuracy: 0.5062
Epoch 3/10
4/4 [==============================] - 3s 865ms/step - loss: 0.6710 - accuracy: 0.840
0 - val_loss: 0.6926 - val_accuracy: 0.5130
Epoch 4/10
4/4 [==============================] - 3s 1s/step - loss: 0.6624 - accuracy: 0.8800 -
val_loss: 0.6925 - val_accuracy: 0.5138
Epoch 5/10
4/4 [==============================] - 7s 2s/step - loss: 0.6532 - accuracy: 0.8800 -
val_loss: 0.6926 - val_accuracy: 0.5186
Epoch 6/10
4/4 [==============================] - 7s 2s/step - loss: 0.6451 - accuracy: 0.9100 -
val_loss: 0.6924 - val_accuracy: 0.5158
Epoch 7/10
4/4 [==============================] - 3s 1s/step - loss: 0.6357 - accuracy: 0.9200 -
val_loss: 0.6925 - val_accuracy: 0.5168
Epoch 8/10
4/4 [==============================] - 3s 866ms/step - loss: 0.6258 - accuracy: 0.950
0 - val_loss: 0.6926 - val_accuracy: 0.5146
Epoch 9/10
4/4 [==============================] - 3s 1s/step - loss: 0.6151 - accuracy: 0.9400 -
val_loss: 0.6927 - val_accuracy: 0.5138
Epoch 10/10
4/4 [==============================] - 7s 2s/step - loss: 0.6045 - accuracy: 0.9400 -
val_loss: 0.6926 - val_accuracy: 0.5170
157/157 [==============================] - 2s 14ms/step - loss: 0.6926 - accuracy: 0.
5170
Test Loss :  0.6925599575042725
Test Accuracy :  0.5170000195503235
```

Perfomance of RNN Model for 100 Training Samples :

Accuracy :



Loss :



Pre Trained Embedded Layred RNN Model

In [5]:

```
import numpy as np

# Load GloVe word embeddings
embeddings_index = {}
with open('glove.6B.100d.txt') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs

# Prepare the GloVe word embeddings matrix
embedding_dim = 100  # Based on the dimension of GloVe embeddings used
embedding_matrix = np.zeros((10000, embedding_dim))  # Assuming 10000 words
for i, word in enumerate(embeddings_index.keys()):  # Iterate through GloVe words
    if i < 10000:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector

# Define the model with pretrained word embeddings
rnn_model_pretrained_100 = Sequential()
rnn_model_pretrained_100.add(Embedding(10000, embedding_dim, input_length=maxlen, trai
rnn_model_pretrained_100.add(SimpleRNN(16, activation="relu"))
rnn_model_pretrained_100.add(Dense(1, activation='sigmoid'))

# Set the pretrained word embeddings
rnn_model_pretrained_100.layers[0].set_weights([embedding_matrix])

# Compile the model
rnn_model_pretrained_100.compile(loss='binary_crossentropy', optimizer='rmsprop', metr

# Print model summary
print("RNN Model Architecture with Pretrained Embeddings : ")
print(rnn_model_pretrained_100.summary())
print(" ")

# Train the RNN model with pretrained embeddings
rnn_history_pretrained_100 = rnn_model_pretrained_100.fit(train_data_100, train_labels

# Evaluate the model on the test data
pre_trained_rnn100_test_loss, pre_trained_rnn100_test_accuracy = rnn_model_pretrained_

print("Test Loss : ", pre_trained_rnn100_test_loss)
print("Test Accuracy : ", pre_trained_rnn100_test_accuracy)

# Plot training and validation accuracy
print("Perfomance of Pre Trained RNN Model for 100 Training Samples : ")
print(" ")
print("Accuracy : ")
print(" ")
plt.plot(rnn_history_pretrained_100.history['accuracy'], label='Training Accuracy (Pre
plt.plot(rnn_history_pretrained_100.history['val_accuracy'], label='Validation Accurac
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

print(" ")
```

```
print("Loss : ")
print(" ")
# Plot training and validation loss
plt.plot(rnn_history_pretrained_100.history['loss'], label='Training Loss (Pretrained)
plt.plot(rnn_history_pretrained_100.history['val_loss'], label='Validation Loss (Pretr
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

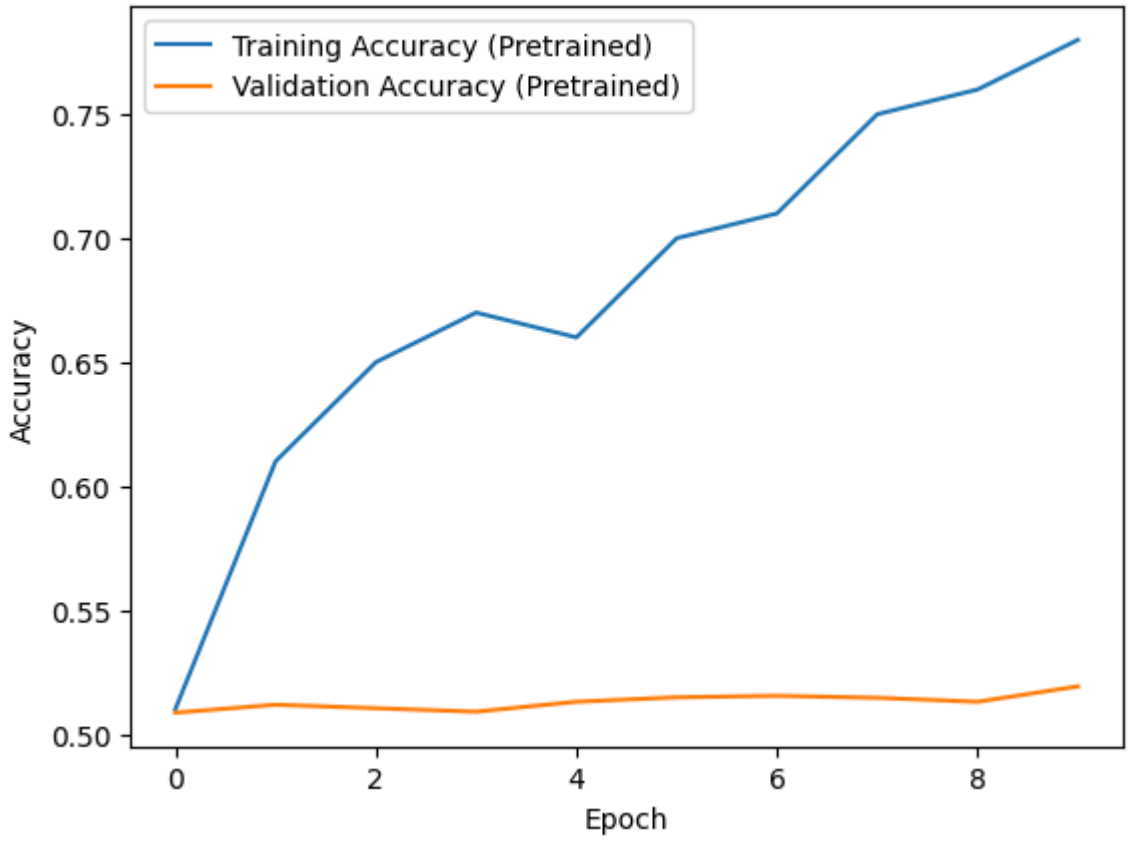
RNN Model Architecture with Pretrained Embeddings :
Model: "sequential_1"

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 150, 100)	1000000
simple_rnn_1 (SimpleRNN)	(None, 16)	1872
dense_1 (Dense)	(None, 1)	17

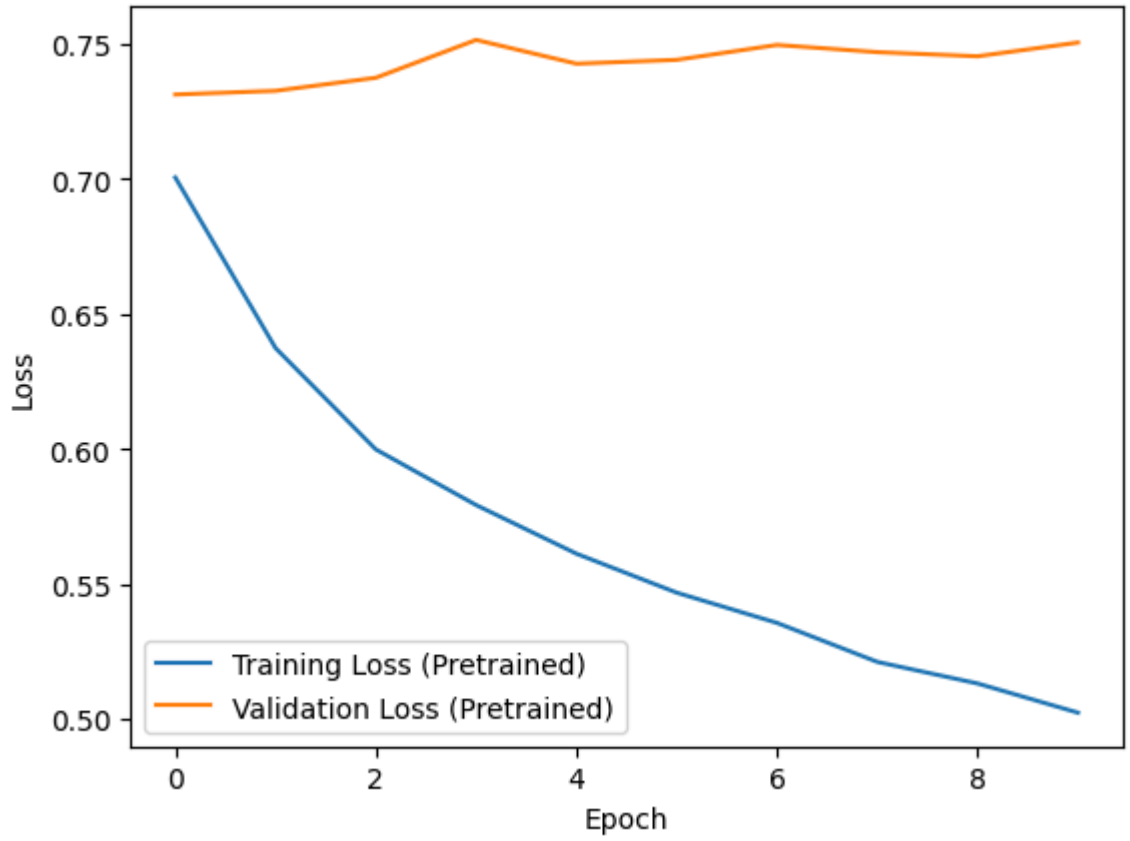
Total params: 1001889 (3.82 MB)
Trainable params: 1889 (7.38 KB)
Non-trainable params: 1000000 (3.81 MB)

None

```
Epoch 1/10
4/4 [==============================] - 7s 1s/step - loss: 0.7006 - accuracy: 0.5100 -
val_loss: 0.7313 - val_accuracy: 0.5086
Epoch 2/10
4/4 [==============================] - 2s 762ms/step - loss: 0.6374 - accuracy: 0.610
0 - val_loss: 0.7327 - val_accuracy: 0.5118
Epoch 3/10
4/4 [==============================] - 2s 752ms/step - loss: 0.5998 - accuracy: 0.650
0 - val_loss: 0.7376 - val_accuracy: 0.5104
Epoch 4/10
4/4 [==============================] - 2s 748ms/step - loss: 0.5792 - accuracy: 0.670
0 - val_loss: 0.7516 - val_accuracy: 0.5090
Epoch 5/10
4/4 [==============================] - 3s 917ms/step - loss: 0.5612 - accuracy: 0.660
0 - val_loss: 0.7428 - val_accuracy: 0.5130
Epoch 6/10
4/4 [==============================] - 6s 2s/step - loss: 0.5467 - accuracy: 0.7000 -
val_loss: 0.7442 - val_accuracy: 0.5148
Epoch 7/10
4/4 [==============================] - 6s 2s/step - loss: 0.5355 - accuracy: 0.7100 -
val_loss: 0.7497 - val_accuracy: 0.5154
Epoch 8/10
4/4 [==============================] - 3s 955ms/step - loss: 0.5211 - accuracy: 0.750
0 - val_loss: 0.7470 - val_accuracy: 0.5146
Epoch 9/10
4/4 [==============================] - 2s 753ms/step - loss: 0.5131 - accuracy: 0.760
0 - val_loss: 0.7455 - val_accuracy: 0.5130
Epoch 10/10
4/4 [==============================] - 2s 764ms/step - loss: 0.5023 - accuracy: 0.780
0 - val_loss: 0.7506 - val_accuracy: 0.5192
157/157 [==============================] - 3s 18ms/step - loss: 0.7506 - accuracy: 0.
5192
Test Loss :  0.7505980730056763
Test Accuracy :  0.5192000269889832
Perfomance of Pre Trained RNN Model for 100 Training Samples :

Accuracy :
```



Loss :

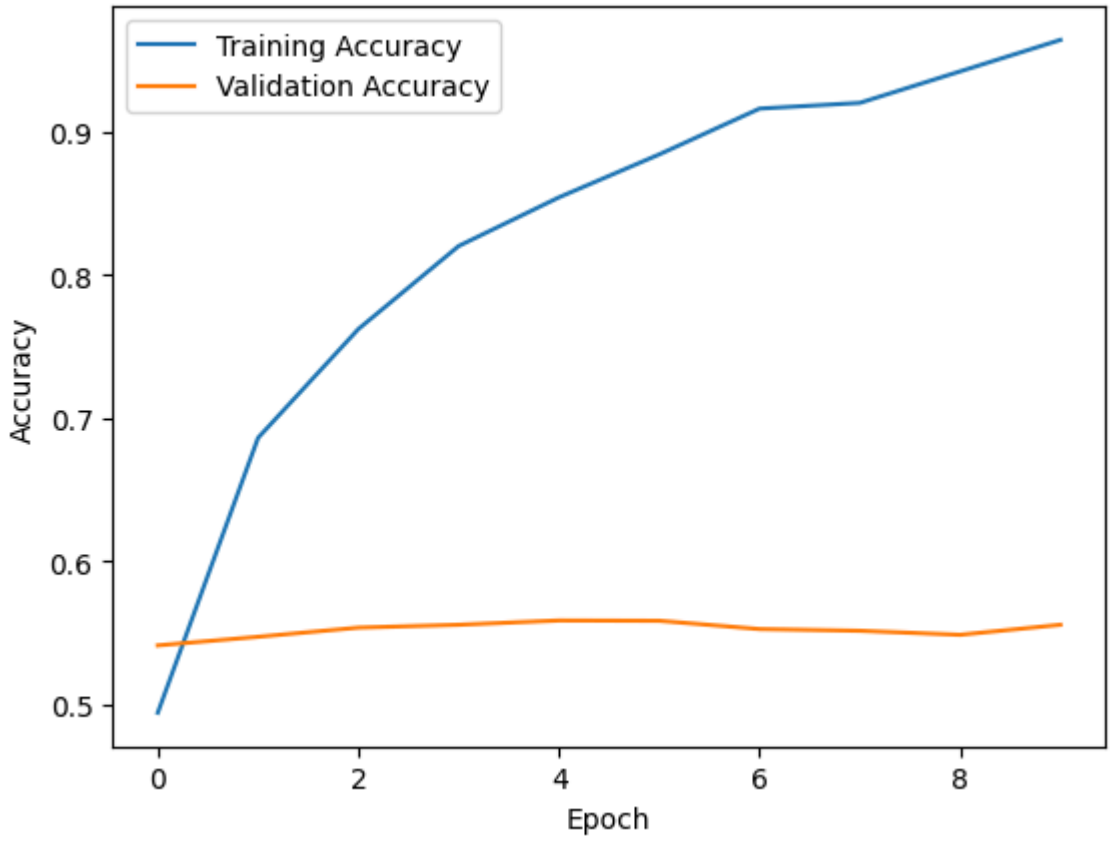


For Training Samples 500

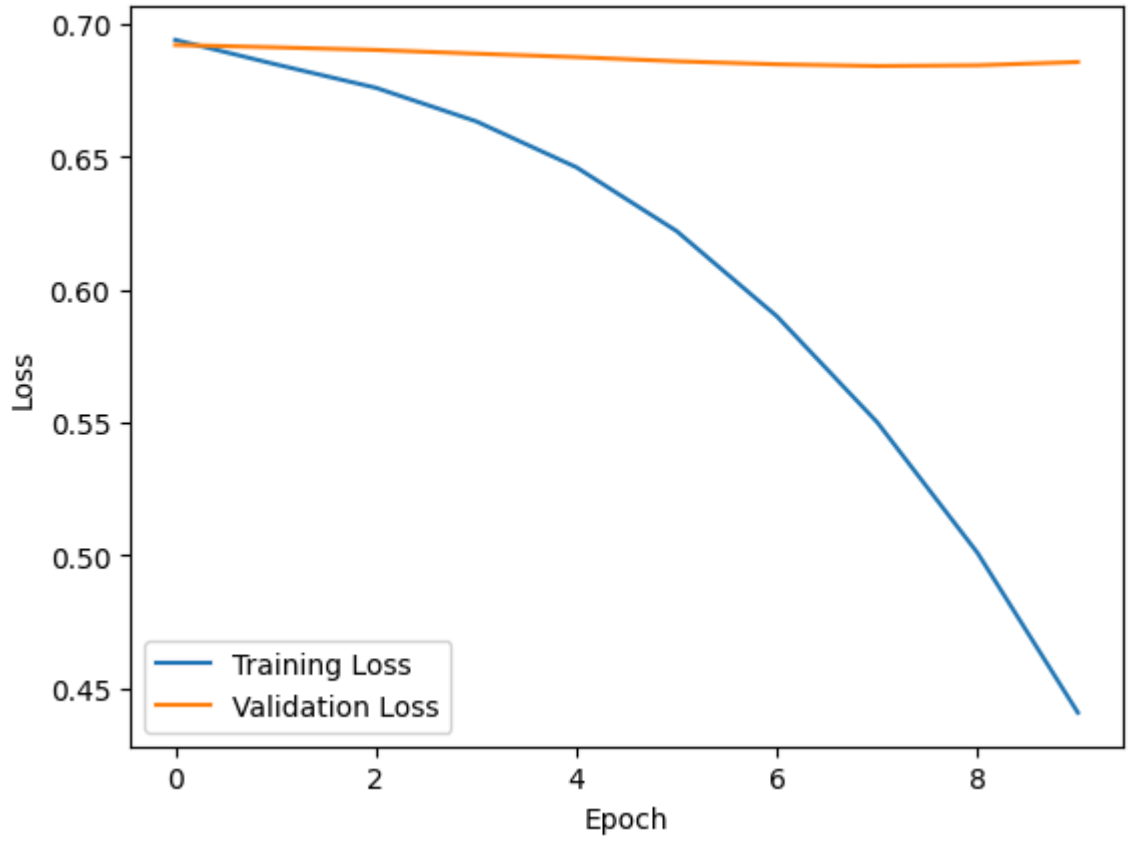
In [6]:
```
# Select the first 100 samples for training
train_data_500 = train_data[:500]
train_labels_500 = train_labels[:500]
```

In [7]:
```
# Build The RNN model
rnn_model_500 = Sequential()

rnn_model_500.add(Embedding(10000,32,input_length =len(train_data_500[0])))
rnn_model_500.add(SimpleRNN(16,input_shape = (10000,maxlen), return_sequences=False,ac
rnn_model_500.add(Dense(1)) #flatten
rnn_model_500.add(Activation("sigmoid")) #using sigmoid for binary classification
rnn_model_500.compile(loss="binary_crossentropy",optimizer="rmsprop",metrics=["accurac

print(" ")
print("RNN Model Architecture : ")
print(rnn_model_500.summary())
print(" ")
# Train the RNN model
rnn_history_500 = rnn_model_500.fit(train_data_500, train_labels_500, epochs=10, batch

# Evaluate the model
rnn500_test_loss, rnn500_test_accuracy = rnn_model_500.evaluate(test_data, test_labels

print("Test Loss : ", rnn500_test_loss)
print("Test Accuracy : ", rnn500_test_accuracy)


#Model Perfomance Evaluation
print(" ")
print("Perfomance of RNN Model for 500 Training Samples : ")
print(" ")
# Plot training and validation accuracy
print("Accuracy : ")
print(" ")
plt.plot(rnn_history_500.history['accuracy'], label='Training Accuracy')
plt.plot(rnn_history_500.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
print(" ")
print("Loss : ")
print(" ")
plt.plot(rnn_history_500.history['loss'], label='Training Loss')
plt.plot(rnn_history_500.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
RNN Model Architecture :
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 150, 32)           320000

 simple_rnn_2 (SimpleRNN)    (None, 16)                784

 dense_2 (Dense)             (None, 1)                 17

 activation_1 (Activation)   (None, 1)                 0

=================================================================
Total params: 320801 (1.22 MB)
Trainable params: 320801 (1.22 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None

Epoch 1/10
16/16 [==============================] - 16s 859ms/step - loss: 0.6940 - accuracy: 0.
4940 - val_loss: 0.6921 - val_accuracy: 0.5410
Epoch 2/10
16/16 [==============================] - 9s 593ms/step - loss: 0.6848 - accuracy: 0.6
860 - val_loss: 0.6912 - val_accuracy: 0.5470
Epoch 3/10
16/16 [==============================] - 6s 368ms/step - loss: 0.6760 - accuracy: 0.7
620 - val_loss: 0.6902 - val_accuracy: 0.5534
Epoch 4/10
16/16 [==============================] - 9s 602ms/step - loss: 0.6634 - accuracy: 0.8
200 - val_loss: 0.6888 - val_accuracy: 0.5554
Epoch 5/10
16/16 [==============================] - 9s 565ms/step - loss: 0.6461 - accuracy: 0.8
540 - val_loss: 0.6875 - val_accuracy: 0.5584
Epoch 6/10
16/16 [==============================] - 5s 334ms/step - loss: 0.6220 - accuracy: 0.8
840 - val_loss: 0.6859 - val_accuracy: 0.5582
Epoch 7/10
16/16 [==============================] - 8s 533ms/step - loss: 0.5900 - accuracy: 0.9
160 - val_loss: 0.6848 - val_accuracy: 0.5524
Epoch 8/10
16/16 [==============================] - 12s 778ms/step - loss: 0.5501 - accuracy: 0.
9200 - val_loss: 0.6842 - val_accuracy: 0.5512
Epoch 9/10
16/16 [==============================] - 5s 310ms/step - loss: 0.5010 - accuracy: 0.9
420 - val_loss: 0.6844 - val_accuracy: 0.5484
Epoch 10/10
16/16 [==============================] - 8s 514ms/step - loss: 0.4409 - accuracy: 0.9
640 - val_loss: 0.6857 - val_accuracy: 0.5554
157/157 [==============================] - 2s 14ms/step - loss: 0.6857 - accuracy: 0.
5554
Test Loss :  0.6856571435928345
Test Accuracy :  0.555400013923645

Perfomance of RNN Model for 500 Training Samples :

Accuracy :
```



Loss :



In [8]:
```
# Define the model with pretrained word embeddings
rnn_model_pretrained_500 = Sequential()
```

```
rnn_model_pretrained_500.add(Embedding(10000, embedding_dim, input_length=maxlen, trai
rnn_model_pretrained_500.add(SimpleRNN(16, activation="relu"))
rnn_model_pretrained_500.add(Dense(1, activation='sigmoid'))

# Set the pretrained word embeddings
rnn_model_pretrained_500.layers[0].set_weights([embedding_matrix])

# Compile the model
rnn_model_pretrained_500.compile(loss='binary_crossentropy', optimizer='rmsprop', metr

# Print model summary
print("RNN Model Architecture with Pretrained Embeddings : ")
print(rnn_model_pretrained_500.summary())
print(" ")

# Train the RNN model with pretrained embeddings
rnn_history_pretrained_500 = rnn_model_pretrained_500.fit(train_data_500, train_labels

# Evaluate the model on the test data
pre_trained_rnn500_test_loss, pre_trained_rnn500_test_accuracy = rnn_model_pretrained_

print("Test Loss : ", pre_trained_rnn500_test_loss)
print("Test Accuracy : ", pre_trained_rnn500_test_accuracy)

# Plot training and validation accuracy
print("Perfomance of Pre Trained RNN Model for 500 Training Samples : ")
print(" ")
print("Accuracy : ")
print(" ")
plt.plot(rnn_history_pretrained_500.history['accuracy'], label='Training Accuracy (Pre
plt.plot(rnn_history_pretrained_500.history['val_accuracy'], label='Validation Accurac
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

print(" ")
print("Loss : ")
print(" ")
# Plot training and validation loss
plt.plot(rnn_history_pretrained_500.history['loss'], label='Training Loss (Pretrained)
plt.plot(rnn_history_pretrained_500.history['val_loss'], label='Validation Loss (Pretr
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

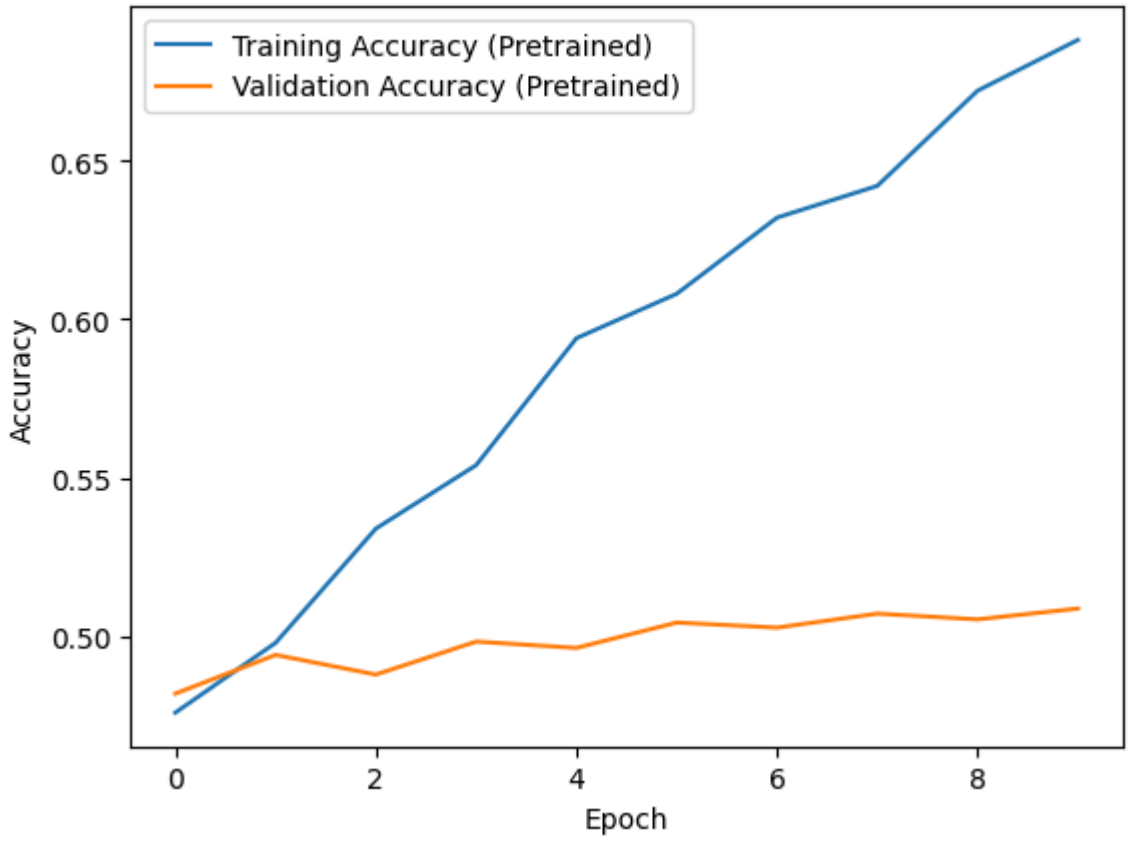
RNN Model Architecture with Pretrained Embeddings :
Model: "sequential_3"

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 150, 100)	1000000
simple_rnn_3 (SimpleRNN)	(None, 16)	1872
dense_3 (Dense)	(None, 1)	17

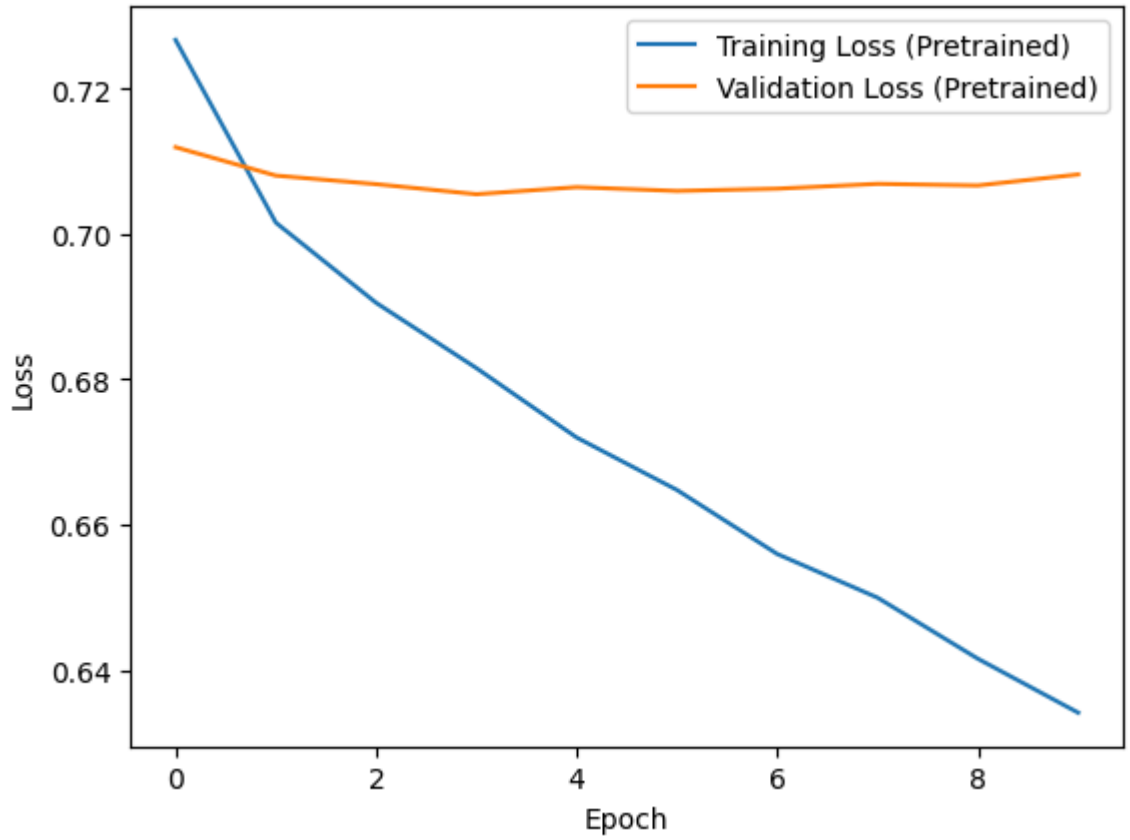
Total params: 1001889 (3.82 MB)
Trainable params: 1889 (7.38 KB)
Non-trainable params: 1000000 (3.81 MB)

None

```
Epoch 1/10
16/16 [==============================] - 9s 512ms/step - loss: 0.7267 - accuracy: 0.4
760 - val_loss: 0.7119 - val_accuracy: 0.4820
Epoch 2/10
16/16 [==============================] - 9s 593ms/step - loss: 0.7015 - accuracy: 0.4
980 - val_loss: 0.7080 - val_accuracy: 0.4942
Epoch 3/10
16/16 [==============================] - 9s 579ms/step - loss: 0.6905 - accuracy: 0.5
340 - val_loss: 0.7068 - val_accuracy: 0.4880
Epoch 4/10
16/16 [==============================] - 5s 296ms/step - loss: 0.6815 - accuracy: 0.5
540 - val_loss: 0.7054 - val_accuracy: 0.4984
Epoch 5/10
16/16 [==============================] - 9s 562ms/step - loss: 0.6720 - accuracy: 0.5
940 - val_loss: 0.7064 - val_accuracy: 0.4964
Epoch 6/10
16/16 [==============================] - 6s 372ms/step - loss: 0.6648 - accuracy: 0.6
080 - val_loss: 0.7059 - val_accuracy: 0.5044
Epoch 7/10
16/16 [==============================] - 4s 269ms/step - loss: 0.6560 - accuracy: 0.6
320 - val_loss: 0.7062 - val_accuracy: 0.5028
Epoch 8/10
16/16 [==============================] - 4s 250ms/step - loss: 0.6500 - accuracy: 0.6
420 - val_loss: 0.7069 - val_accuracy: 0.5072
Epoch 9/10
16/16 [==============================] - 9s 577ms/step - loss: 0.6416 - accuracy: 0.6
720 - val_loss: 0.7067 - val_accuracy: 0.5054
Epoch 10/10
16/16 [==============================] - 7s 427ms/step - loss: 0.6341 - accuracy: 0.6
880 - val_loss: 0.7082 - val_accuracy: 0.5088
157/157 [==============================] - 4s 25ms/step - loss: 0.7082 - accuracy: 0.
5088
Test Loss :  0.7081617712974548
Test Accuracy :  0.5088000297546387
Perfomance of Pre Trained RNN Model for 500 Training Samples :

Accuracy :
```



Loss :

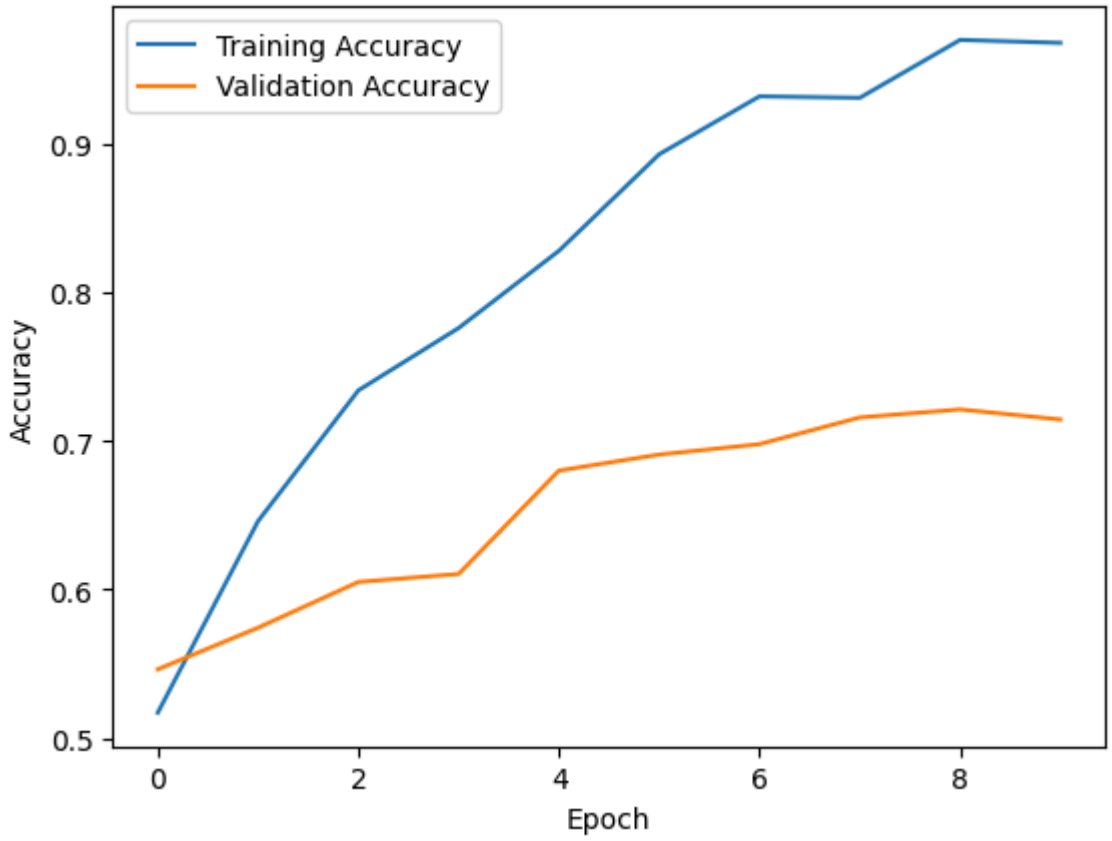


For Training Samples 1000

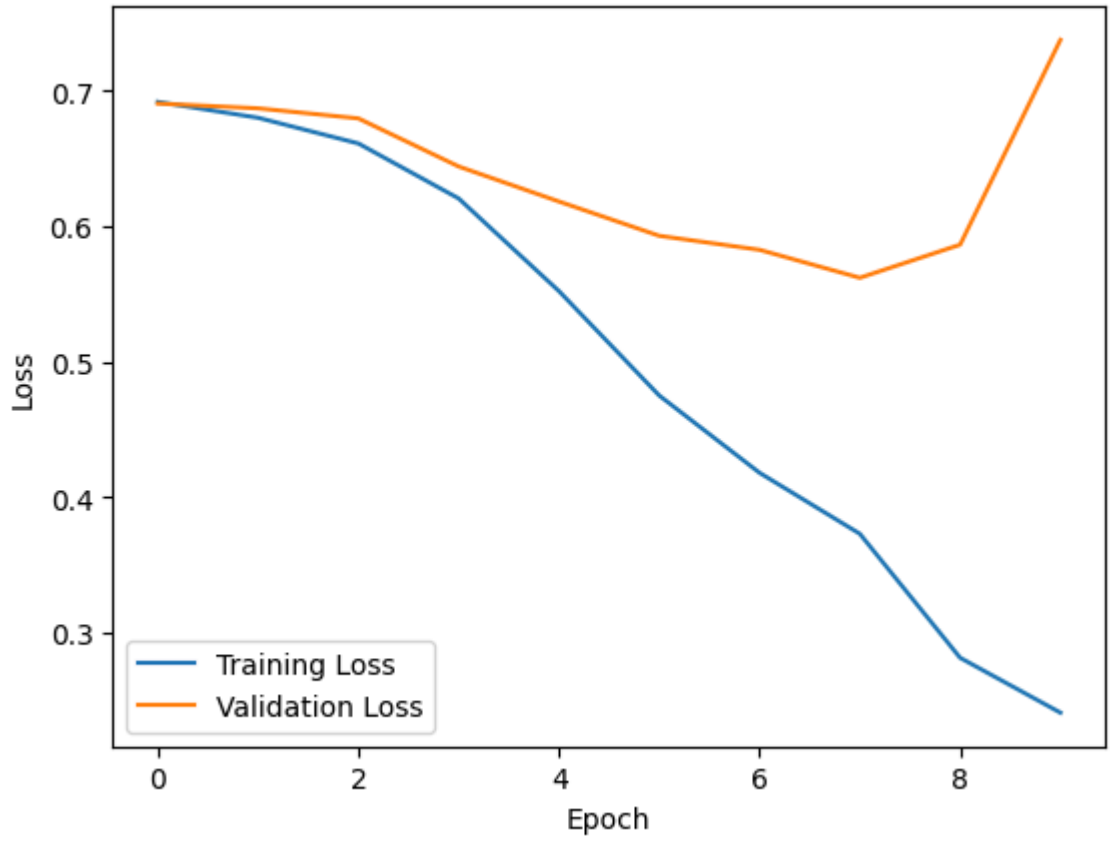
In [9]:
```
# Select the first 100 samples for training
train_data_1000 = train_data[:1000]
train_labels_1000 = train_labels[:1000]
```

In [10]:
```
# Build The RNN model
rnn_model_1000 = Sequential()

rnn_model_1000.add(Embedding(10000,32,input_length =len(train_data_500[0])))
rnn_model_1000.add(SimpleRNN(16,input_shape = (10000,maxlen), return_sequences=False,a
rnn_model_1000.add(Dense(1)) #flatten
rnn_model_1000.add(Activation("sigmoid")) #using sigmoid for binary classification
rnn_model_1000.compile(loss="binary_crossentropy",optimizer="rmsprop",metrics=["accura

print(" ")
print("RNN Model Architecture : ")
print(rnn_model_1000.summary())
print(" ")
# Train the RNN model
rnn_history_1000 = rnn_model_1000.fit(train_data_1000, train_labels_1000, epochs=10, b

# Evaluate the model
rnn1000_test_loss, rnn1000_test_accuracy = rnn_model_1000.evaluate(test_data, test_lab

print("Test Loss : ", rnn1000_test_loss)
print("Test Accuracy : ", rnn1000_test_accuracy)


#Model Perfomance Evaluation
print(" ")
print("Perfomance of RNN Model for 1000 Training Samples : ")
print(" ")
# Plot training and validation accuracy
print("Accuracy : ")
print(" ")
plt.plot(rnn_history_1000.history['accuracy'], label='Training Accuracy')
plt.plot(rnn_history_1000.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
print(" ")
print("Loss : ")
print(" ")
plt.plot(rnn_history_1000.history['loss'], label='Training Loss')
plt.plot(rnn_history_1000.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
RNN Model Architecture :
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 150, 32)           320000

 simple_rnn_4 (SimpleRNN)    (None, 16)                784

 dense_4 (Dense)             (None, 1)                 17

 activation_2 (Activation)   (None, 1)                 0

=================================================================
Total params: 320801 (1.22 MB)
Trainable params: 320801 (1.22 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None

Epoch 1/10
32/32 [==============================] - 11s 286ms/step - loss: 0.6917 - accuracy: 0.
5170 - val_loss: 0.6905 - val_accuracy: 0.5462
Epoch 2/10
32/32 [==============================] - 14s 451ms/step - loss: 0.6800 - accuracy: 0.
6460 - val_loss: 0.6870 - val_accuracy: 0.5742
Epoch 3/10
32/32 [==============================] - 8s 251ms/step - loss: 0.6611 - accuracy: 0.7
340 - val_loss: 0.6796 - val_accuracy: 0.6050
Epoch 4/10
32/32 [==============================] - 13s 414ms/step - loss: 0.6205 - accuracy: 0.
7760 - val_loss: 0.6441 - val_accuracy: 0.6104
Epoch 5/10
32/32 [==============================] - 10s 296ms/step - loss: 0.5521 - accuracy: 0.
8280 - val_loss: 0.6182 - val_accuracy: 0.6800
Epoch 6/10
32/32 [==============================] - 8s 263ms/step - loss: 0.4751 - accuracy: 0.8
930 - val_loss: 0.5929 - val_accuracy: 0.6908
Epoch 7/10
32/32 [==============================] - 14s 425ms/step - loss: 0.4181 - accuracy: 0.
9320 - val_loss: 0.5826 - val_accuracy: 0.6978
Epoch 8/10
32/32 [==============================] - 16s 492ms/step - loss: 0.3730 - accuracy: 0.
9310 - val_loss: 0.5619 - val_accuracy: 0.7158
Epoch 9/10
32/32 [==============================] - 13s 404ms/step - loss: 0.2815 - accuracy: 0.
9700 - val_loss: 0.5864 - val_accuracy: 0.7212
Epoch 10/10
32/32 [==============================] - 7s 218ms/step - loss: 0.2410 - accuracy: 0.9
680 - val_loss: 0.7375 - val_accuracy: 0.7144
157/157 [==============================] - 4s 25ms/step - loss: 0.7375 - accuracy: 0.
7144
Test Loss :  0.7374849915504456
Test Accuracy :  0.7143999934196472

Perfomance of RNN Model for 1000 Training Samples :

Accuracy :
```



Loss :



In [11]:
```
# Define the model with pretrained word embeddings
rnn_model_pretrained_1000 = Sequential()
```

```
rnn_model_pretrained_1000.add(Embedding(10000, embedding_dim, input_length=maxlen, tra
rnn_model_pretrained_1000.add(SimpleRNN(16, activation="relu"))
rnn_model_pretrained_1000.add(Dense(1, activation='sigmoid'))

# Set the pretrained word embeddings
rnn_model_pretrained_1000.layers[0].set_weights([embedding_matrix])

# Compile the model
rnn_model_pretrained_1000.compile(loss='binary_crossentropy', optimizer='rmsprop', met

# Print model summary
print("RNN Model Architecture with Pretrained Embeddings : ")
print(rnn_model_pretrained_1000.summary())
print(" ")

# Train the RNN model with pretrained embeddings
rnn_history_pretrained_1000 = rnn_model_pretrained_1000.fit(train_data_1000, train_lab

# Evaluate the model on the test data
pre_trained_rnn1000_test_loss, pre_trained_rnn1000_test_accuracy = rnn_model_pretraine

print("Test Loss : ", pre_trained_rnn1000_test_loss)
print("Test Accuracy : ", pre_trained_rnn1000_test_accuracy)

# Plot training and validation accuracy
print("Perfomance of Pre Trained RNN Model for 1000 Training Samples : ")
print(" ")
print("Accuracy : ")
print(" ")
plt.plot(rnn_history_pretrained_1000.history['accuracy'], label='Training Accuracy (Pr
plt.plot(rnn_history_pretrained_1000.history['val_accuracy'], label='Validation Accura
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

print(" ")
print("Loss : ")
print(" ")
# Plot training and validation loss
plt.plot(rnn_history_pretrained_1000.history['loss'], label='Training Loss (Pretrained
plt.plot(rnn_history_pretrained_1000.history['val_loss'], label='Validation Loss (Pret
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

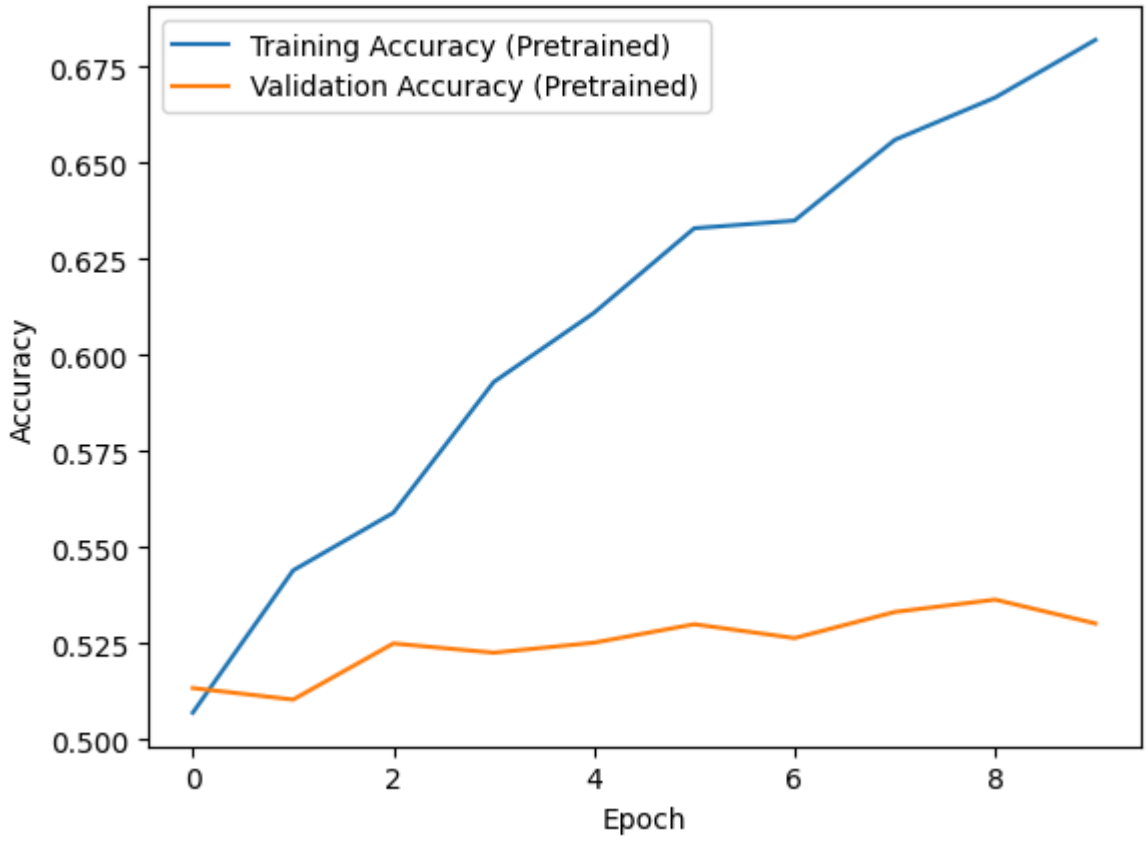
RNN Model Architecture with Pretrained Embeddings :
Model: "sequential_5"

Layer (type)	Output Shape	Param #
embedding_5 (Embedding)	(None, 150, 100)	1000000
simple_rnn_5 (SimpleRNN)	(None, 16)	1872
dense_5 (Dense)	(None, 1)	17

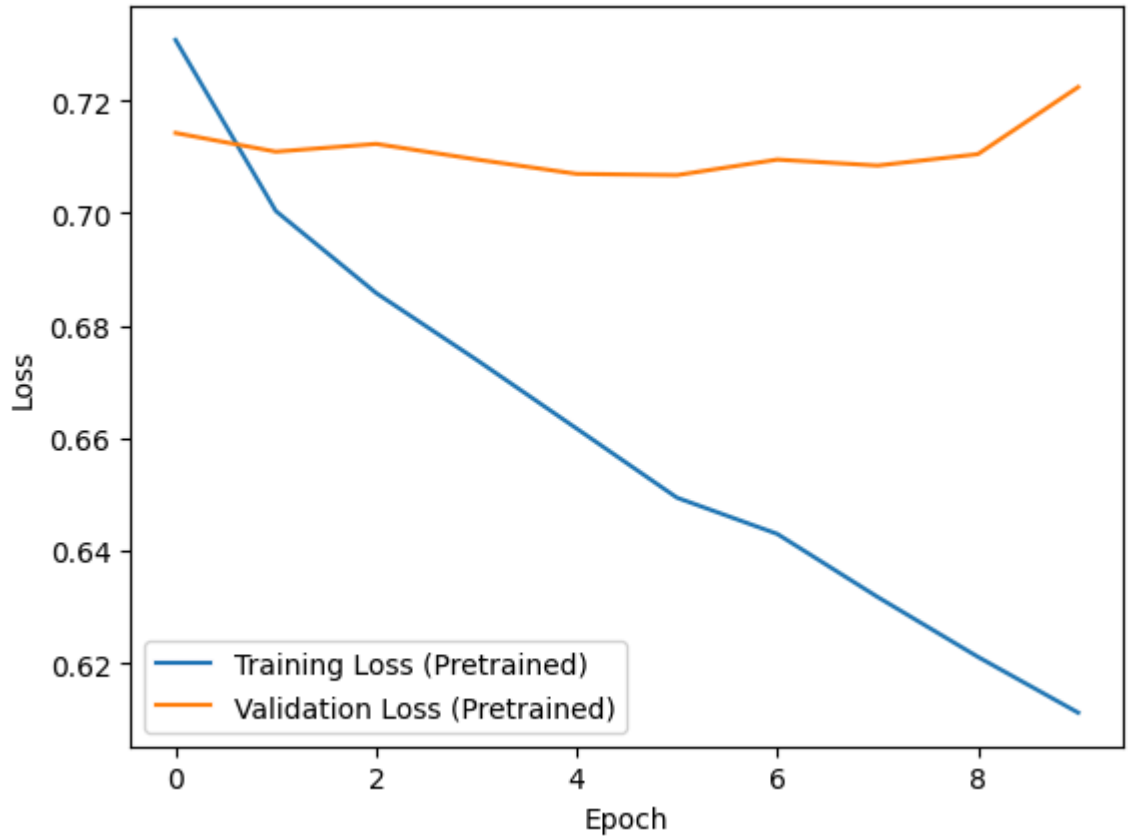
Total params: 1001889 (3.82 MB)
Trainable params: 1889 (7.38 KB)
Non-trainable params: 1000000 (3.81 MB)

None

```
Epoch 1/10
32/32 [==============================] - 7s 174ms/step - loss: 0.7308 - accuracy: 0.5
070 - val_loss: 0.7142 - val_accuracy: 0.5134
Epoch 2/10
32/32 [==============================] - 8s 266ms/step - loss: 0.7004 - accuracy: 0.5
440 - val_loss: 0.7109 - val_accuracy: 0.5104
Epoch 3/10
32/32 [==============================] - 11s 343ms/step - loss: 0.6858 - accuracy: 0.
5590 - val_loss: 0.7123 - val_accuracy: 0.5250
Epoch 4/10
32/32 [==============================] - 5s 163ms/step - loss: 0.6739 - accuracy: 0.5
930 - val_loss: 0.7095 - val_accuracy: 0.5226
Epoch 5/10
32/32 [==============================] - 9s 277ms/step - loss: 0.6617 - accuracy: 0.6
110 - val_loss: 0.7070 - val_accuracy: 0.5252
Epoch 6/10
32/32 [==============================] - 10s 330ms/step - loss: 0.6494 - accuracy: 0.
6330 - val_loss: 0.7067 - val_accuracy: 0.5300
Epoch 7/10
32/32 [==============================] - 5s 164ms/step - loss: 0.6430 - accuracy: 0.6
350 - val_loss: 0.7095 - val_accuracy: 0.5264
Epoch 8/10
32/32 [==============================] - 9s 283ms/step - loss: 0.6318 - accuracy: 0.6
560 - val_loss: 0.7085 - val_accuracy: 0.5332
Epoch 9/10
32/32 [==============================] - 11s 334ms/step - loss: 0.6211 - accuracy: 0.
6670 - val_loss: 0.7105 - val_accuracy: 0.5364
Epoch 10/10
32/32 [==============================] - 6s 183ms/step - loss: 0.6112 - accuracy: 0.6
820 - val_loss: 0.7224 - val_accuracy: 0.5302
157/157 [==============================] - 4s 25ms/step - loss: 0.7224 - accuracy: 0.
5302
Test Loss :  0.7223929762840271
Test Accuracy :  0.5302000045776367
Perfomance of Pre Trained RNN Model for 1000 Training Samples :

Accuracy :
```

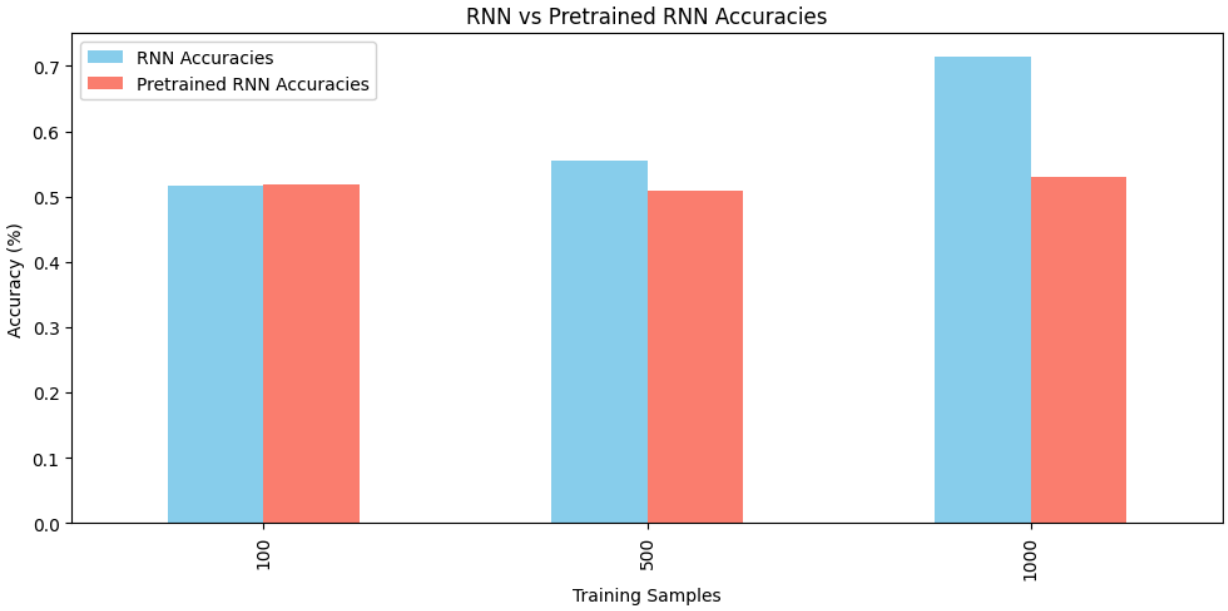


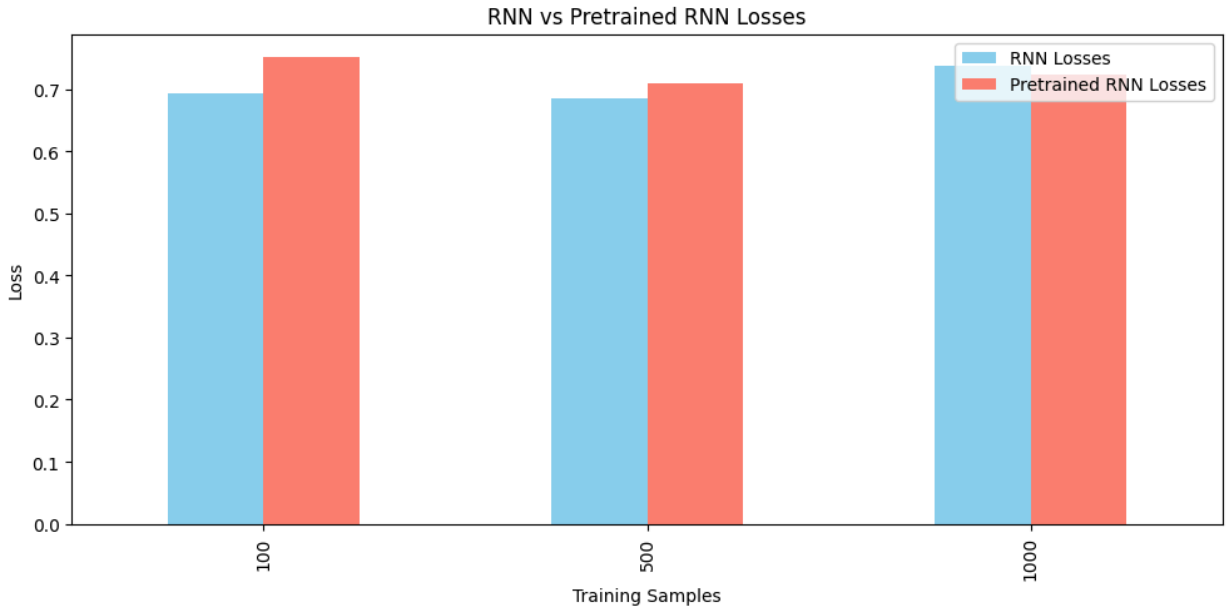
Loss :



In [12]:
```
import pandas as pd
```

```
# Define RNN and pretrained RNN accuracies and losses
rnn_accuracies = [rnn100_test_accuracy, rnn500_test_accuracy, rnn1000_test_accuracy]
pretrained_accuracies = [pre_trained_rnn100_test_accuracy, pre_trained_rnn500_test_acc

rnn_losses = [rnn100_test_loss, rnn500_test_loss, rnn1000_test_loss]  # RNN test losse
pretrained_losses = [pre_trained_rnn100_test_loss, pre_trained_rnn500_test_loss, pre_t

# Create a pandas DataFrame to store accuracies and losses
data = {
    "RNN Accuracies": rnn_accuracies,
    "Pretrained RNN Accuracies": pretrained_accuracies,
    "RNN Losses": rnn_losses,
    "Pretrained RNN Losses": pretrained_losses
}

index = ["100", "500", "1000"]  # Training samples

df = pd.DataFrame(data, index=index)

# Plotting
fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(10, 10))

# Accuracy comparison
df[["RNN Accuracies", "Pretrained RNN Accuracies"]].plot(kind='bar', ax=axes[0], color
axes[0].set_title("RNN vs Pretrained RNN Accuracies")
axes[0].set_xlabel("Training Samples")
axes[0].set_ylabel("Accuracy (%)")

# Loss comparison
df[["RNN Losses", "Pretrained RNN Losses"]].plot(kind='bar', ax=axes[1], color=['skybl
axes[1].set_title("RNN vs Pretrained RNN Losses")
axes[1].set_xlabel("Training Samples")
axes[1].set_ylabel("Loss")

plt.tight_layout()
plt.show()
```


RNN vs Pretrained RNN Accuracies
RNN Accuracies
Pretrained RNN Accuracies
Accuracy (%)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
100
500
1000
Training Samples


RNN vs Pretrained RNN Losses
RNN Losses
Pretrained RNN Losses
Loss
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
100
500
1000
Training Samples