**IMPORT LIBRARIES**

In [2]:

**import** pandas **as** pd

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.preprocessing **import** LabelEncoder

**from** keras.models **import** Model

**from** keras.layers **import** LSTM, Activation, Dense, Dropout, Input, Embedding

**from** keras.optimizers **import** Adam

**from** keras.preprocessing.text **import** Tokenizer

**from** keras.preprocessing **import** sequence

**from** keras.utils **import** pad\_sequences

**from** keras.utils **import** to\_categorical

**from** keras.callbacks **import** EarlyStopping

**READING DATASET**

In [4]:

**from** google.colab **import** drive

drive**.**mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True).

In [6]:

df **=** pd**.**read\_csv('/content/drive/MyDrive/IBM PROJECT/assignment 4/spam.csv', delimiter**=**',',encoding**=**'latin-1')

df**.**head()

Out[6]:

|  | **v1** | **v2** | **Unnamed: 2** | **Unnamed: 3** | **Unnamed: 4** |
| --- | --- | --- | --- | --- | --- |
| **0** | ham | Go until jurong point, crazy.. Available only ... | NaN | NaN | NaN |
| **1** | ham | Ok lar... Joking wif u oni... | NaN | NaN | NaN |
| **2** | spam | Free entry in 2 a wkly comp to win FA Cup fina... | NaN | NaN | NaN |
| **3** | ham | U dun say so early hor... U c already then say... | NaN | NaN | NaN |
| **4** | ham | Nah I don't think he goes to usf, he lives aro... | NaN | NaN | NaN |

**PRE-PROCESSING THE DATA**

In [7]:

df**.**drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'],axis**=**1,inplace**=True**)

**from** wordcloud **import** WordCloud, STOPWORDS, ImageColorGenerator

X **=** df**.**v2

Y **=** df**.**v1

le **=** LabelEncoder()

Y **=** le**.**fit\_transform(Y)

Y **=** Y**.**reshape(**-**1,1)

X\_train,X\_test,Y\_train,Y\_test **=** train\_test\_split(X,Y,test\_size**=**0.25)

max\_words **=** 1000

max\_len **=** 150

tok **=** Tokenizer(num\_words**=**max\_words)

tok**.**fit\_on\_texts(X\_train)

sequences **=** tok**.**texts\_to\_sequences(X\_train)

sequences\_matrix **=** pad\_sequences(sequences,maxlen**=**max\_len)

**CREATING MODEL**

In [8]:

inputs **=** Input(shape**=**[max\_len])

layer **=** Embedding(max\_words,50,input\_length**=**max\_len)(inputs)

**ADDING LAYERS**

In [9]:

layer **=** LSTM(128)(layer)

layer **=** Dense(128)(layer)

layer **=** Activation('relu')(layer)

layer **=** Dropout(0.5)(layer)

layer **=** Dense(1.5)(layer)

layer **=** Activation('sigmoid')(layer)

model **=** Model(inputs**=**inputs,outputs**=**layer)

In [10]:

model**.**summary()

Model: "model"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

input\_1 (InputLayer) [(None, 150)] 0

embedding (Embedding) (None, 150, 50) 50000

lstm (LSTM) (None, 128) 91648

dense (Dense) (None, 128) 16512

activation (Activation) (None, 128) 0

dropout (Dropout) (None, 128) 0

dense\_1 (Dense) (None, 1) 129

activation\_1 (Activation) (None, 1) 0

=================================================================

Total params: 158,289

Trainable params: 158,289

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**COMPILE THE MODEL**

In [11]:

model**.**compile(loss**=**'binary\_crossentropy',optimizer**=**Adam(),metrics**=**['accuracy'])

**FIT THE MODEL**

In [12]:

history **=** model**.**fit(sequences\_matrix,Y\_train,batch\_size**=**20,epochs**=**15,validation\_split**=**0.2)

Epoch 1/15

168/168 [==============================] - 34s 183ms/step - loss: 0.1790 - accuracy: 0.9402 - val\_loss: 0.0514 - val\_accuracy: 0.9833

Epoch 2/15

168/168 [==============================] - 30s 180ms/step - loss: 0.0387 - accuracy: 0.9886 - val\_loss: 0.0400 - val\_accuracy: 0.9880

Epoch 3/15

168/168 [==============================] - 33s 195ms/step - loss: 0.0198 - accuracy: 0.9943 - val\_loss: 0.0390 - val\_accuracy: 0.9892

Epoch 4/15

168/168 [==============================] - 31s 185ms/step - loss: 0.0097 - accuracy: 0.9973 - val\_loss: 0.0583 - val\_accuracy: 0.9785

Epoch 5/15

168/168 [==============================] - 32s 190ms/step - loss: 0.0058 - accuracy: 0.9988 - val\_loss: 0.0628 - val\_accuracy: 0.9833

Epoch 6/15

168/168 [==============================] - 33s 194ms/step - loss: 0.0077 - accuracy: 0.9979 - val\_loss: 0.0537 - val\_accuracy: 0.9833

Epoch 7/15

168/168 [==============================] - 31s 184ms/step - loss: 0.0041 - accuracy: 0.9985 - val\_loss: 0.0656 - val\_accuracy: 0.9844

Epoch 8/15

168/168 [==============================] - 31s 186ms/step - loss: 0.0089 - accuracy: 0.9988 - val\_loss: 0.0590 - val\_accuracy: 0.9833

Epoch 9/15

168/168 [==============================] - 32s 189ms/step - loss: 0.1862 - accuracy: 0.9710 - val\_loss: 0.0513 - val\_accuracy: 0.9868

Epoch 10/15

168/168 [==============================] - 37s 222ms/step - loss: 0.0053 - accuracy: 0.9985 - val\_loss: 0.0674 - val\_accuracy: 0.9821

Epoch 11/15

168/168 [==============================] - 49s 291ms/step - loss: 0.0052 - accuracy: 0.9985 - val\_loss: 0.0813 - val\_accuracy: 0.9844

Epoch 12/15

168/168 [==============================] - 39s 232ms/step - loss: 0.0014 - accuracy: 0.9997 - val\_loss: 0.0774 - val\_accuracy: 0.9809

Epoch 13/15

168/168 [==============================] - 34s 203ms/step - loss: 7.5460e-04 - accuracy: 0.9997 - val\_loss: 0.0799 - val\_accuracy: 0.9821

Epoch 14/15

168/168 [==============================] - 34s 203ms/step - loss: 3.5156e-04 - accuracy: 1.0000 - val\_loss: 0.0833 - val\_accuracy: 0.9833

Epoch 15/15

168/168 [==============================] - 31s 185ms/step - loss: 3.1628e-04 - accuracy: 1.0000 - val\_loss: 0.0831 - val\_accuracy: 0.9844

In [13]:

metrics **=** pd**.**DataFrame(history**.**history)

metrics**.**rename(columns **=** {'loss': 'Training\_Loss', 'accuracy': 'Training\_Accuracy', 'val\_loss': 'Validation\_Loss', 'val\_accuracy': 'Validation\_Accuracy'}, inplace **=** **True**)

**def** plot\_graphs1(var1, var2, string):

metrics[[var1, var2]]**.**plot()

plt**.**title('Training and Validation ' **+** string)

plt**.**xlabel ('Number of epochs')

plt**.**ylabel(string)

plt**.**legend([var1, var2])

In [14]:

plot\_graphs1('Training\_Accuracy', 'Validation\_Accuracy', 'accuracy')

**SAVE THE MODEL**

In [15]:

model**.**save('Spam\_sms\_classifier.h5')

**TEST THE MODEL**

In [16]:

test\_sequences **=** tok**.**texts\_to\_sequences(X\_test)

test\_sequences\_matrix **=** pad\_sequences(test\_sequences,maxlen**=**max\_len)

In [17]:

accuracy1 **=** model**.**evaluate(test\_sequences\_matrix,Y\_test)

44/44 [==============================] - 3s 78ms/step - loss: 0.1372 - accuracy: 0.9821

In [18]:

print(' Accuracy: {:0.5f}'**.**format(accuracy1[0],accuracy1[1]))

Accuracy: 0.13715