In [0]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Dense, Input, GlobalMaxPooling1D
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Embedding
from tensorflow.keras.layers import LSTM, Embedding
from tensorflow.keras.models import Model
```

In [111]:
```
df = pd.read_csv('spam.csv', encoding= 'ISO-8859-1')
```

In [113]:
```
df.head()
```

Out[113]:

	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

In [114]:
```
# drop unwanted columns

df = df.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis=1)
```

In [115]:
```
df.head()
```

Out[115]:

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

In [121]:
```
# rename column

df.columns = ['labels', 'data']
```

In [122]:
```
df.head()
```

Out[122]:

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

In [124]:
```
# Create binary label (0 and 1)

df['b_labels'] = df['labels'].map({'ham': 0, 'spam': 1})
y = df['b_labels'].values
```

In [125]:
```
# Split the data

x_train, x_test, y_train, y_test = train_test_split(df['data'], y, test_size = 0.33)
```

In [130]:
```
# Convert sentences to sequences

max_vocab_size = 20000
tokenizer = Tokenizer (num_words=max_vocab_size)
tokenizer.fit_on_texts(x_train)
sequences_train = tokenizer.texts_to_sequences (x_train)
sequences_test = tokenizer.texts_to_sequences (x_test)
```

In [131]:
```
# Check word index mapping (to check the number of words in Vocabulary)

word2idx = tokenizer.word_index
V = len(word2idx)
print ("Total number of unique tokens are: %s" % V)
```

```
Total number of unique tokens are: 7218
```

In [132]:
```
# pad sequences (to get N x T matrix)

data_train = pad_sequences(sequences_train)
print('shape of data train tensor:', data_train.shape)
```

```
shape of data train tensor: (3733, 162)
```

In [288]:
```
# Get the value of T to get sequence length

T = data_train.shape[1]
```

In [139]:
```
print(T)
```

```
162
```

In [450]:
```
# pad the test set
data_test = pad_sequences(sequences_test, maxlen=T)
# maxlen = T, to truncate longer sentences in test set
print('Shape of data test tensor:', data_test.shape)
```

```
Shape of data test tensor: (1839, 162)
```

In [ ]:
```

```

BUILDING THE MODEL (CNN)

In [145]:
```
# creating the model

D = 20    # This is a hyperparameter, any vector size can be choosen.

# Input layer
i = Input(shape=(T,))    # Input layer takes in sequences of integers and returns word vectors

# Embedding layer
x = Embedding(V + 1, D)(i)

# CNN Layer
x = Conv1D(32, 3, activation='relu')(x)
x = MaxPooling1D(3)(x)

# Second CNN layer
x = Conv1D(64, 3, activation='relu')(x)
x = MaxPooling1D(3)(x)

# Third CNN layer
x = Conv1D(128, 3, activation='relu')(x)
x = GlobalMaxPooling1D()(x)


# Dense layer
x = Dense (1, activation='sigmoid')(x)

model = Model(i, x)
```

In [146]:
```
# compile model
r = model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [147]:
```
# train model
r = model.fit (x = data_train, y = y_train, epochs = 5, validation_data=(data_test, y_test))
```

```
Epoch 1/5
117/117 [==============================] - 2s 10ms/step - loss: 0.3835 - accuracy: 0.8636 - val_loss: 0.2902 - v
al_accuracy: 0.8586
Epoch 2/5
117/117 [==============================] - 1s 9ms/step - loss: 0.1264 - accuracy: 0.9443 - val_loss: 0.0815 - va
l_accuracy: 0.9766
Epoch 3/5
117/117 [==============================] - 1s 9ms/step - loss: 0.0147 - accuracy: 0.9962 - val_loss: 0.0631 - va
l_accuracy: 0.9810
Epoch 4/5
117/117 [==============================] - 1s 10ms/step - loss: 0.0048 - accuracy: 0.9989 - val_loss: 0.0790 -
val_accuracy: 0.9815
Epoch 5/5
117/117 [==============================] - 1s 10ms/step - loss: 0.0026 - accuracy: 0.9995 - val_loss: 0.0789 -
val_accuracy: 0.9810
```

In [148]:
```
# loss
import matplotlib.pyplot as plt

plt.plot(r.history['loss'], label='Loss')
plt.plot(r.history['val_loss'], label='validation Loss')
plt.legend()
plt.show()
```

In [149]:
```
plt.plot(r.history['accuracy'], label='Accuracy')
plt.plot(r.history['val_accuracy'], label='validation Accuracy')
plt.legend()
plt.show()
```

In [ ]:
```

```

BUILDING THE MODEL (RNN)

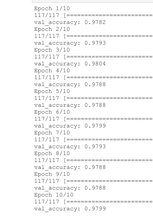
In [155]:
```
# creating the model

D1 = 20    # This is a hyperparameter, any vector size can be choosen.

# hidden state vectorize size
M = 15

# Input layer
i1 = Input(shape=(T,))    # Input layer takes in sequences of integers and returns word vectors

# Embedding layer
x1 = Embedding(V + 1, D1)(i1)

x1 = LSTM(M, return_sequences=True) (x1)
x1 = GlobalMaxPooling1D()(x1)

# Dense layer
x1 = Dense (1, activation='sigmoid')(x1)  # using activation 'sigmoid' because of it binary classification prob

model = Model(i1, x1)
```

In [156]:
```
# compile model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

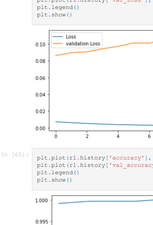
In [163]:
```
# train model
r1 = model.fit (x = data_train, y = y_train, epochs = 10, validation_data=(data_test, y_test))
```

```
Epoch 1/10
117/117 [==============================] - 7s 61ms/step - loss: 0.0072 - accuracy: 0.9992 - val_loss: 0.0861 -
val_accuracy: 0.9782
Epoch 2/10
117/117 [==============================] - 7s 61ms/step - loss: 0.0062 - accuracy: 0.9995 - val_loss: 0.0896 -
val_accuracy: 0.9793
Epoch 3/10
117/117 [==============================] - 7s 61ms/step - loss: 0.0052 - accuracy: 0.9997 - val_loss: 0.0906 -
val_accuracy: 0.9804
Epoch 4/10
117/117 [==============================] - 7s 60ms/step - loss: 0.0044 - accuracy: 0.9997 - val_loss: 0.0962 -
val_accuracy: 0.9788
Epoch 5/10
117/117 [==============================] - 7s 60ms/step - loss: 0.0038 - accuracy: 0.9997 - val_loss: 0.0971 -
val_accuracy: 0.9788
Epoch 6/10
117/117 [==============================] - 7s 61ms/step - loss: 0.0035 - accuracy: 0.9997 - val_loss: 0.1011 -
val_accuracy: 0.9739
Epoch 7/10
117/117 [==============================] - 7s 60ms/step - loss: 0.0031 - accuracy: 1.0000 - val_loss: 0.1008 -
val_accuracy: 0.9793
Epoch 8/10
117/117 [==============================] - 7s 60ms/step - loss: 0.0027 - accuracy: 1.0000 - val_loss: 0.1059 -
val_accuracy: 0.9788
Epoch 9/10
117/117 [==============================] - 7s 62ms/step - loss: 0.0026 - accuracy: 1.0000 - val_loss: 0.1077 -
val_accuracy: 0.9788
Epoch 10/10
117/117 [==============================] - 7s 61ms/step - loss: 0.0022 - accuracy: 1.0000 - val_loss: 0.1108 -
val_accuracy: 0.9739
```

In [164]:
```
plt.plot(r1.history['loss'], label='Loss')
plt.plot(r1.history['val_loss'], label='validation Loss')
plt.legend()
plt.show()
```



In [165]:
```
plt.plot(r1.history['accuracy'], label='Accuracy')
plt.plot(r1.history['val_accuracy'], label='validation Accuracy')
plt.legend()
plt.show()
```



In [ ]:
```

```