Ahsanullah University of Science and Technology

Department of Computer Science and Engineering

Course No. : CSE4238

Course Name : Soft Computing Lab

Assignment No. : 03

Submitted By:

Name : Md. Mainul Ahsan

ID No. : 17 01 04 020

Session : Fall - 2020

Section : A (A1)

My ID is 170104020. So, dataset number is 3 and model name is RNN.

In this notebook you will see text precessing on twitter data set and after that I have performed different Machine Learning Algorithms on the data such as Logistic Regression, RandomForestClassifier, SVC, Naive Bayes to classifiy positive and negative tweets. After that I have also built a RNN network which is the best fit for such textual sentiment analysis, since it's a Sequential Dataset which is requirement for RNN network.

Let's Dive into it.

Importing Libraries:

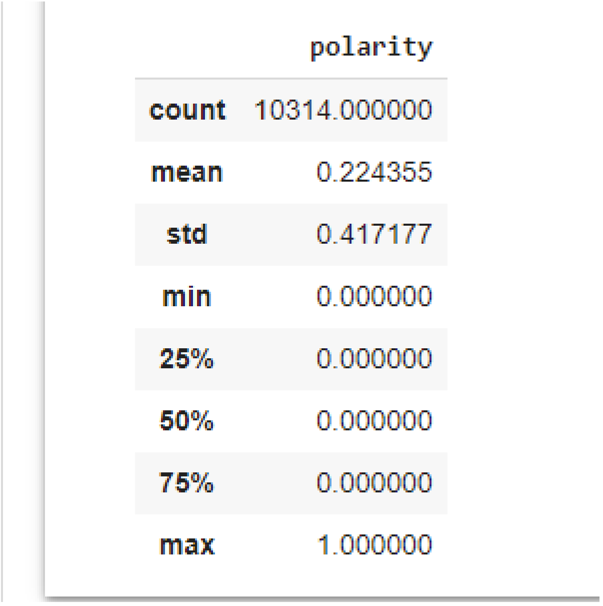
Several libraries are used here such as dataFrame, plotting, nltk, sklearn, tensorflow and some utility libraries.

Dataset:

The given Dataset 3 is used here.

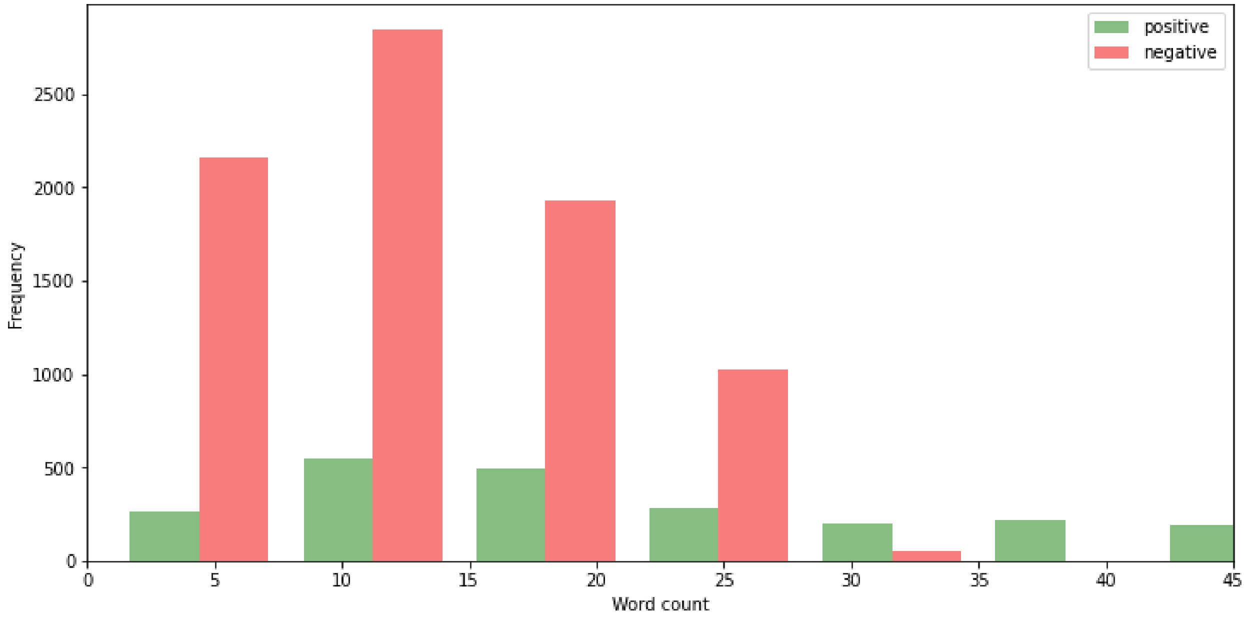
Data Visualization:

Count, mean, std, min, 25%, 50%, 75%, max values from the dataset are shown below:

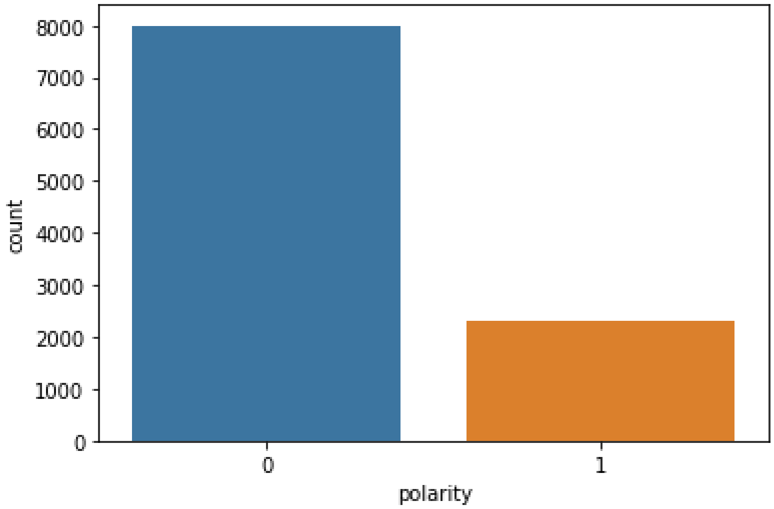


	polarity
count	10314.000000
mean	0.224355
std	0.417177
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	1.000000

Word count distribution for both positive and negative words are shown below:



Sentiment count chart is shown below:

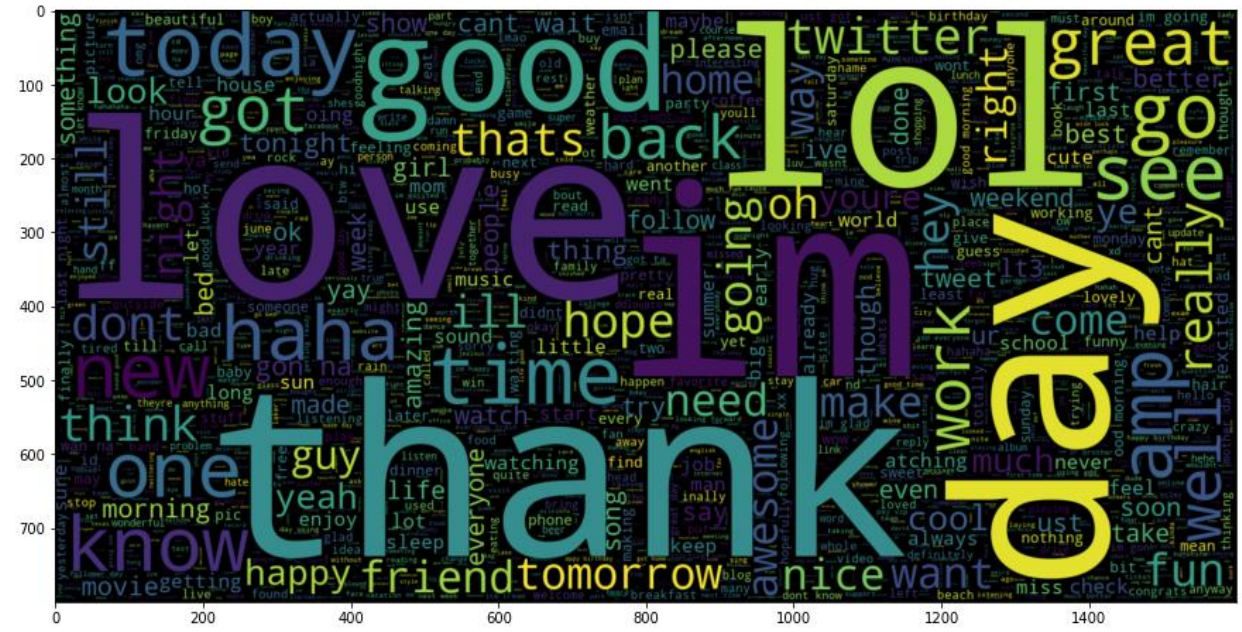


The Preprocessing steps taken are:

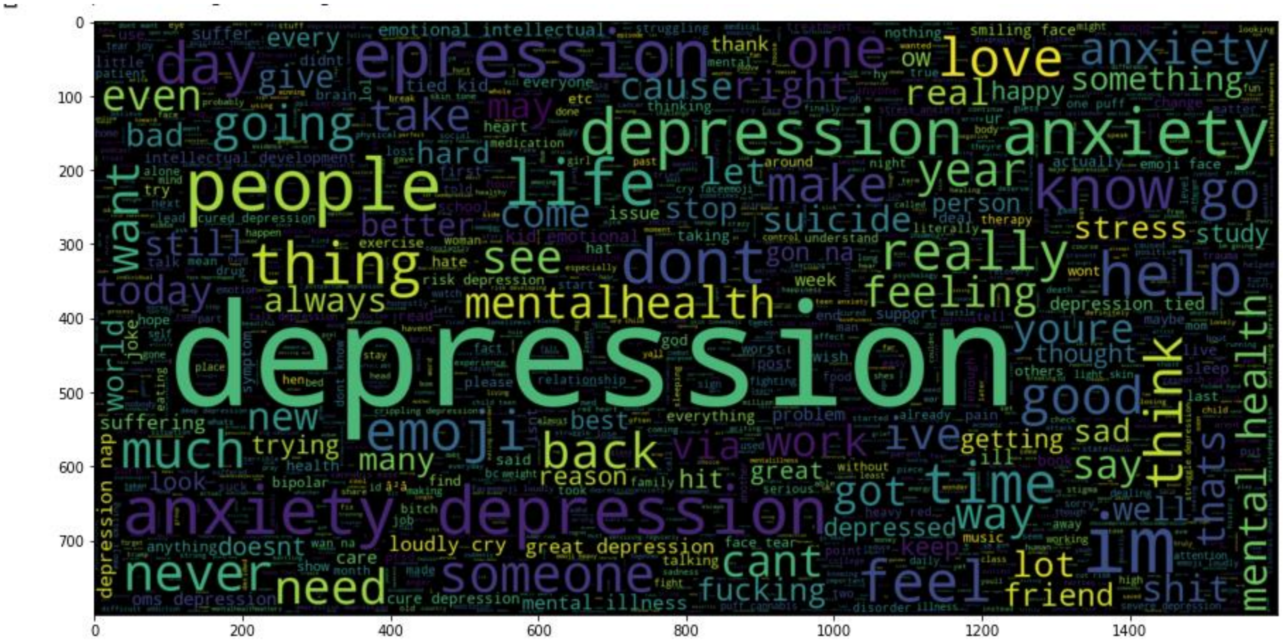
- Lower Casing: Each text is converted to lowercase.
- Removing URLs: Links starting with "http" or "https" or "www" are replaced by "".
- Removing Usernames: Replace @Usernames with word "". (eg: "@XYZ" to "")
- Removing Short Words: Words with length less than 2 are removed.
- Removing Stopwords: Stopwords are the English words which does not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence. (eg: "the", "he", "have")
- Lemmatizing: Lemmatization is the process of converting a word to its base form. (e.g: “wolves” to “wolf”)

Analyzing the data:

Word-cloud for Negative tweets:



<u>Word-cloud for Positive tweets:</u>



<u>Convert text to word frequency vectors:</u>

TF-IDF

This is an acronym than stands for **Term Frequency – Inverse Document** Frequency which are the components of the resulting scores assigned to each word.

- Term Frequency: This summarizes how often a given word appears within a document.
- Inverse Document Frequency: This downscales words that appear a lot across documents.

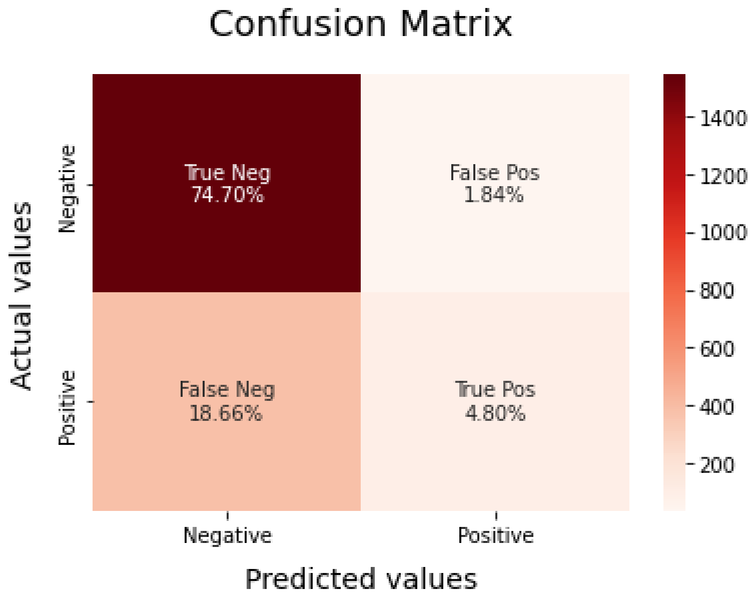
<u>Split train and test:</u>

The Preprocessed Data is divided into 2 sets of data:

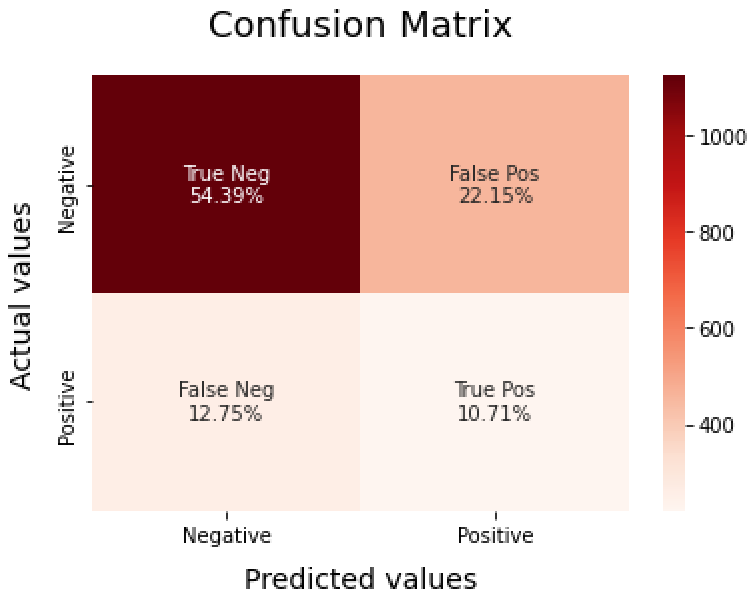
- Training Data: The dataset upon which the model would be trained on. Contains 80% data.
- Test Data: The dataset upon which the model would be tested against. Contains 20% data.

Accuracy of model:

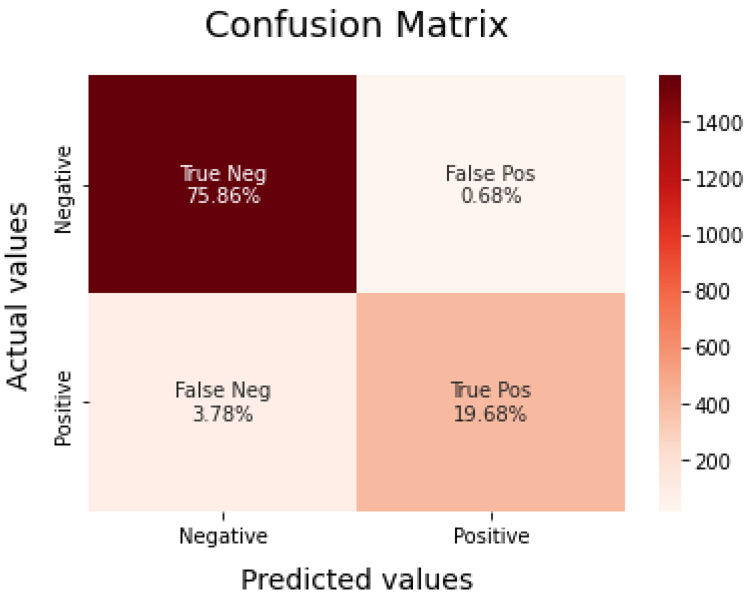
Logistic Regression confusion matrix:



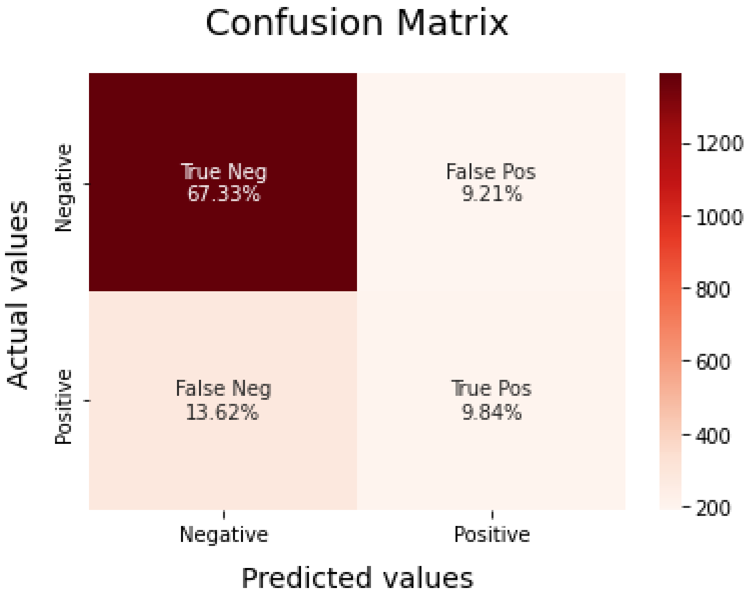
Linear SVM confusion matrix:



Random Forest confusion matrix:



Naive Bayes confusion matrix



What is RNN?

Recurrent neural networks (RNN) are the state of the art algorithm for sequential data and are used by Apple's Siri and and Google's voice search. It is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for machine learning problems that involve sequential data

Embedding Layer

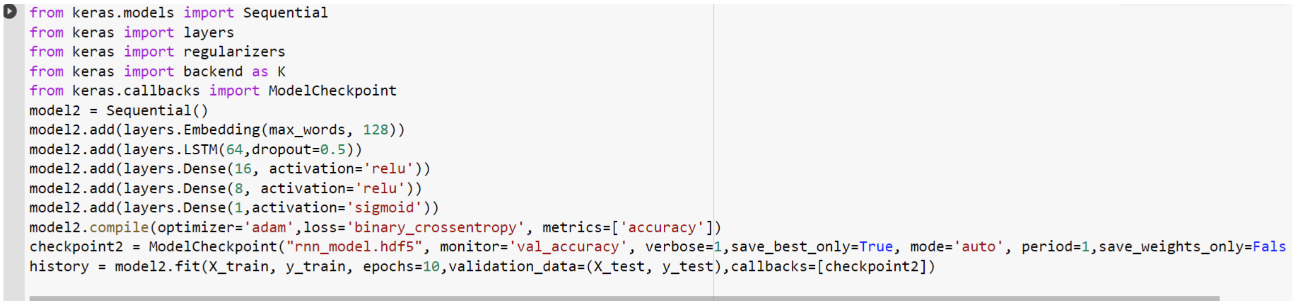
Embedding layer is one of the available layers in Keras. This is mainly used in Natural Language Processing related applications such as language modeling, but it can also be used with other tasks that involve neural networks. While dealing with NLP problems, we can use pre-trained word embeddings such as GloVe. Alternatively we can also train our own embeddings using Keras embedding layer.

LSTM layer

Long Short Term Memory networks, usually called “LSTMs”, were introduced by Hochreiter and Schmiduber. These have widely been used for speech recognition, language modeling, sentiment analysis and text prediction. Before going deep into LSTM, we should first understand the need of LSTM which can be explained by the drawback of practical use of Recurrent Neural Network (RNN). So, let's start with RNN.

Building a Recurrent Neural Network:

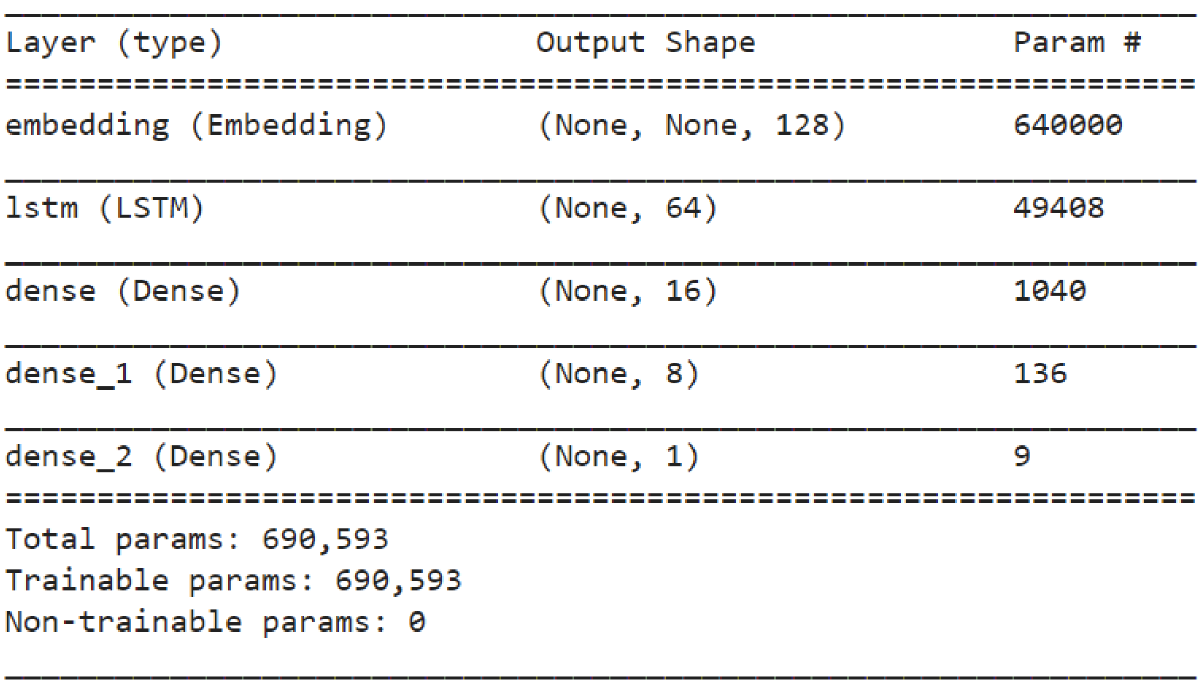
Keras is an incredible library: it allows us to build state-of-the-art models in a few lines of understandable Python code. Although other neural network libraries may be faster or allow more flexibility, nothing can beat Keras for development time and ease-of-use.



```
from keras.models import Sequential
from keras import layers
from keras import regularizers
from keras import backend as K
from keras.callbacks import ModelCheckpoint
model2 = Sequential()
model2.add(layers.Embedding(max_words, 128))
model2.add(layers.LSTM(64,dropout=0.5))
model2.add(layers.Dense(16, activation='relu'))
model2.add(layers.Dense(8, activation='relu'))
model2.add(layers.Dense(1,activation='sigmoid'))
model2.compile(optimizer='adam',loss='binary_crossentropy', metrics=['accuracy'])
checkpoint2 = ModelCheckpoint("rnn_model.hdf5", monitor='val_accuracy', verbose=1,save_best_only=True, mode='auto', period=1,save_weights_only=Fals
history = model2.fit(X_train, y_train, epochs=10,validation_data=(X_test, y_test),callbacks=[checkpoint2])
```

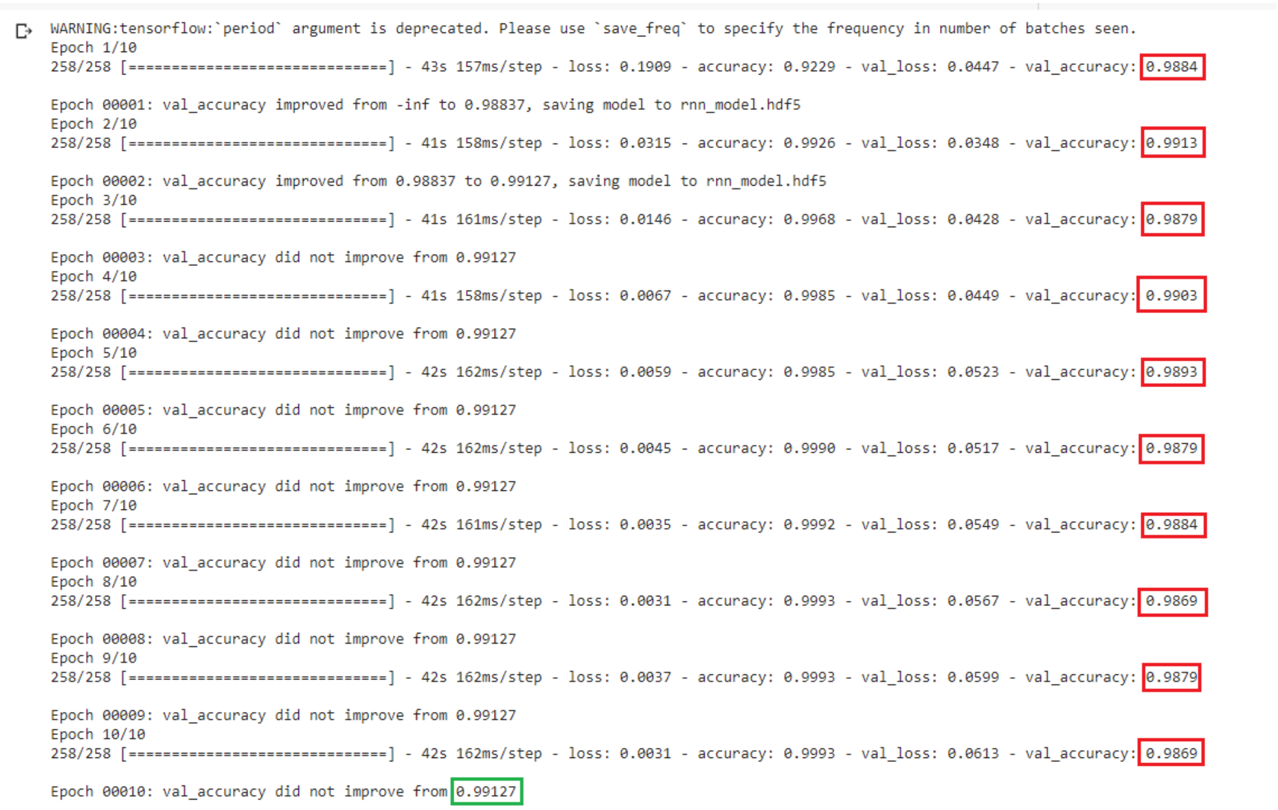
The model is compiled with the adam optimizer (a variant on Stochastic Gradient Descent) and trained using the binary_crossentropy loss. During training, the network will try to minimize the log loss by adjusting the trainable parameters (weights). As always, the gradients of the parameters are calculated using back-propagation and updated with the optimizer. Since we are using Keras.

Layer types, output shapes and parameters are given below:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding (Embedding)        (None, None, 128)         640000    
_________________________________________________________________
lstm (LSTM)                  (None, 64)                49408     
_________________________________________________________________
dense (Dense)                (None, 16)                1040      
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 136       
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 9         
=================================================================
Total params: 690,593
Trainable params: 690,593
Non-trainable params: 0
_________________________________________________________________
```

Here, I am showing the accuracy of 10 epochs and the best accuracy among them:



```
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specify the frequency in number of batches seen.
Epoch 1/10
258/258 [==============================] - 43s 157ms/step - loss: 0.1909 - accuracy: 0.9229 - val_loss: 0.0447 - val_accuracy: 0.9884

Epoch 00001: val_accuracy improved from -inf to 0.98837, saving model to rnn_model.hdf5
Epoch 2/10
258/258 [==============================] - 41s 158ms/step - loss: 0.0315 - accuracy: 0.9926 - val_loss: 0.0348 - val_accuracy: 0.9913

Epoch 00002: val_accuracy improved from 0.98837 to 0.99127, saving model to rnn_model.hdf5
Epoch 3/10
258/258 [==============================] - 41s 161ms/step - loss: 0.0146 - accuracy: 0.9968 - val_loss: 0.0428 - val_accuracy: 0.9879

Epoch 00003: val_accuracy did not improve from 0.99127
Epoch 4/10
258/258 [==============================] - 41s 158ms/step - loss: 0.0067 - accuracy: 0.9985 - val_loss: 0.0449 - val_accuracy: 0.9903

Epoch 00004: val_accuracy did not improve from 0.99127
Epoch 5/10
258/258 [==============================] - 42s 162ms/step - loss: 0.0059 - accuracy: 0.9985 - val_loss: 0.0523 - val_accuracy: 0.9893

Epoch 00005: val_accuracy did not improve from 0.99127
Epoch 6/10
258/258 [==============================] - 42s 162ms/step - loss: 0.0045 - accuracy: 0.9990 - val_loss: 0.0517 - val_accuracy: 0.9879

Epoch 00006: val_accuracy did not improve from 0.99127
Epoch 7/10
258/258 [==============================] - 42s 161ms/step - loss: 0.0035 - accuracy: 0.9992 - val_loss: 0.0549 - val_accuracy: 0.9884

Epoch 00007: val_accuracy did not improve from 0.99127
Epoch 8/10
258/258 [==============================] - 42s 162ms/step - loss: 0.0031 - accuracy: 0.9993 - val_loss: 0.0567 - val_accuracy: 0.9869

Epoch 00008: val_accuracy did not improve from 0.99127
Epoch 9/10
258/258 [==============================] - 42s 162ms/step - loss: 0.0037 - accuracy: 0.9993 - val_loss: 0.0599 - val_accuracy: 0.9879

Epoch 00009: val_accuracy did not improve from 0.99127
Epoch 10/10
258/258 [==============================] - 42s 162ms/step - loss: 0.0031 - accuracy: 0.9993 - val_loss: 0.0613 - val_accuracy: 0.9869

Epoch 00010: val_accuracy did not improve from 0.99127
```

So, we can see that, among all these accuracies the best accuracy from this model is **99.127%.**

GitHub repo: https://github.com/Swaad/170104020_assignment3