Deep Learning to Filter SMS Spam

A Project Report

Submitted for Machine Learning Class Project

By

Gaurav Rai, Roll No.1706019

Shubham Vasnik, Roll No.1706083

Department of Computer Science and Engineering

NATIONAL INSTITUTE OF TECHNOLOGY, PATNA

PATNA – 800005

JAN - JUN 2020

राष्ट्रीय प्रौद्योगिकी संस्थान पटना
NATIONAL INSTITUTE OF TECHNOLOGY PATNA

Contents

Introduction

- **Increase in popularity of short message service (SMS) :**

  People are increasingly using mobile text messages as a way of communication. The popularity of short message service (SMS) has been growing over the last decade. The volume of SMS sent per month on average has increased by a whopping 7700% from 2008 to 2018. For businesses, text messages are more effective than even emails. This is because while 98% of mobile users read their SMS by the end of the day, about 80% of the emails remain unopened (SMS Comparison, 2018) . Hence, it is easy to understand why SMS has grown into a multi-billion dollar commercial industry

- **Increasing Spam Problem With SMS**

  Unfortunately, over the years, mobile phones have also become the target for what is known as SMS Spam. SMS Spam refers to any irrelevant text messages delivered using mobile networks. They are severely annoying to users. An example of an annoying spam text message is as follows “CONGRATS: YOUR MOBILE NO HAVE WON YOU 500,000 IN — MOBILE DRAW UK, TO CLAIM PRIZE SEND BANK DETAIL, NAME, ADDRESS, MOBILE NO, SEX, AGE, TO –”. Such messages come not only from domestic but also international senders. A survey revealed that 68% of mobile phone users are affected by SMS Spam, with teenagers being the worst affected community.

- **Reasons for Popularity of SMS**

  SMS Spam has become popular due to variety of reasons. For one, the cost of sending SMS has decreased dramatically and has even become zero in some cases. In addition, the mobile phone user-base is continually growing. The number of mobile phone users in countries such as India have increased to 775 million in 2018 and by following the patterns it may reach 813 million by 2019 as shown in Fig. 1.

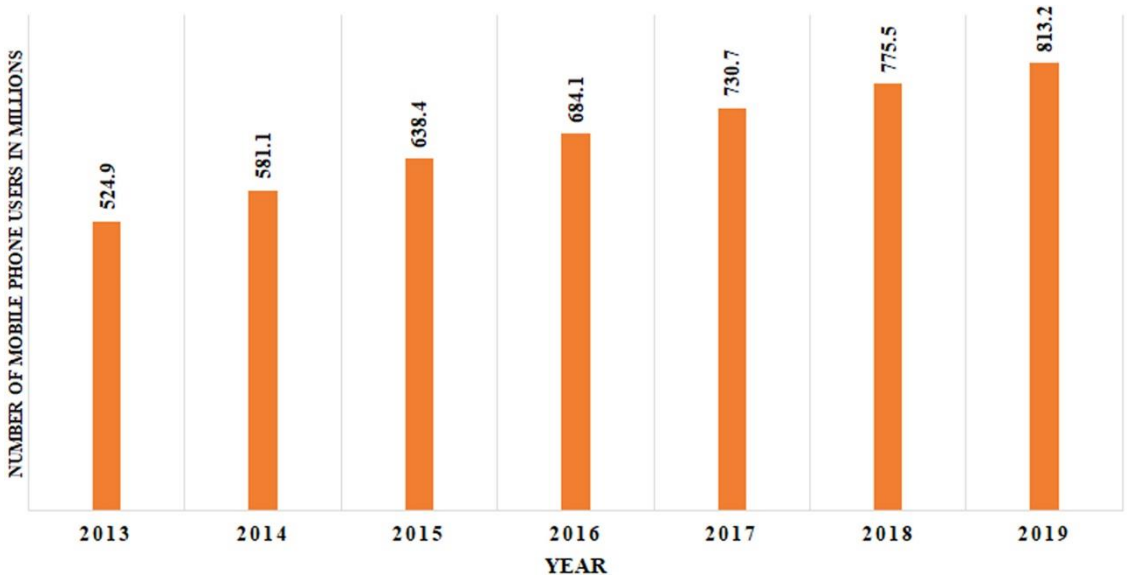


Fig 1

- **Email vs SMS spam filters**
  Moreover, unlike emails that are supported with sophisticated spam filtering, SMS spam filtering is still not very robust. This is because most works that classify SMS spam suffer from the limitation of manual feature engineering, which is not an efficient approach. Identifying prospective features for accurate classification requires prior knowledge and domain expertise. Even then, the selection of the features needs to be reassessed based on criteria such as information gain and feature importance graph. Only then, it is possible to identify features that are helpful for the classification, and those that are not. Such an iterative trial-and-error process is expectedly time-consuming

- **Ways to Improve SMS Spam Filter's**
  One way to obviate this inefficient feature engineering process lies in the use of deep neural networks. Deep learning is a class of machine learning techniques in which several layers of information processing stages are exploited for automatic pattern classification as well as unsupervised feature learning. The components of a deep neural network work together to self-train itself iteratively in order to minimize

classification errors. Over the years, Deep Neural Network based architecture such as Convolutional Neural Network, Recurrent Neural Network , Long Short-Term Memory and their variations have been shown to be helpful in obviating manual feature engineering. These networks have been successfully adopted in various research domains such as speech recognition, sentence modelling and image processing. However, deep neural networks have not yet been applied more in the classification of SMS Spam.

To address the research gap, this report seeks to harness the strength of deep neural networks for classifying SMS Spam. It leverages the Convolutional Neural Network (CNN) model and Long short-term memory (LSTM) model.

Problem with Traditional Implementation & Need for Deep Learning Approach

- We need to identify various features from SMS to be used as input to the machine learning model for using a supervised model
- The feature extraction in machine learning based models are manual which required domain knowledge
- The accuracy of the classifier is highly dependent on these features
- Deep learning eliminates the need of these manual feature extraction by identifying the hidden features from the data.

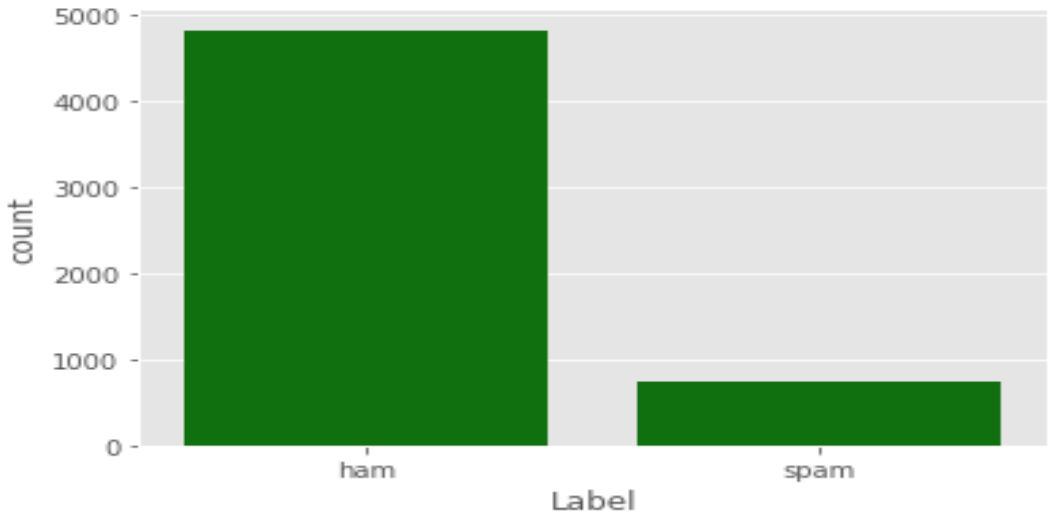
Dataset

- Dataset used: **UCI ML REPO SMS Spam Collection Data**
- Dataset has a total of 4,825 SMS legitimate messages (86.6%) and a total of 747 (13.4%) spam messages.
- Each line is composed by two columns: one with label (ham or spam) and other with the raw text

```
df.head()
```

	Label	Text
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...

```
plt.style.use('ggplot')
sns_plot = sns.countplot(df['Label'], data=df, color="g")
sns_plot.get_figure().savefig('Spam-Ham-Distribution.png')
```



Data Preprocessing

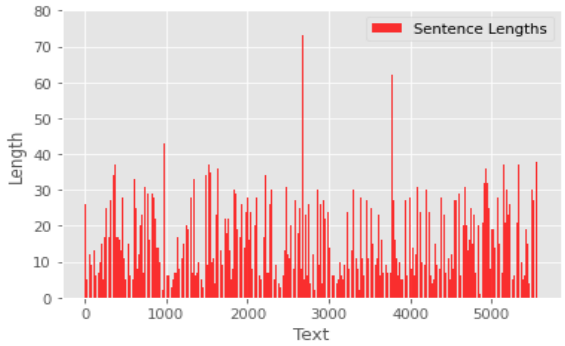
- 'ham' and 'spam' were replaced by 0 and 1 respectively

- We then change the labels 0/1 to categorical values.
- All the punctuations are removed and text is converted into sequences of tokens.
- Plotting distribution of sentence length showed that more than 95% of the sentences has less than 32 words.
- So, we converted all sentences to vectors of dimension 32 and post padded the shorter sentences with zeros.
- We then embedded each sentence into 25d vector space with the help of pretrained Glove weights.
- Those words which were not in Glove were assigned weights randomly.
- We tried both Wikipedia crawled & twitter crawled text and found twitter one was giving better accuracy.
- So now each sentence was converted to 32*25 matrix.

In [0]:
```
tokenize = Tokenizer()
tokenize.fit_on_texts(text)
text_seq = tokenize.texts_to_sequences(text)
```

In [0]:
```
f = lambda sen : len(sen.split(' '))
len_of_sent = text.apply(f).to_list()
```

In [18]:
```
plt.ylim(0, 80)
plt.bar(np.arange(0,len(len_of_sent),1),len_of_sent,align='center',alpha=0.8,color='red')
plt.legend(['Sentence Lengths'])
plt.xlabel('Text')
plt.ylabel('Length')
plt.show()
plt.savefig('Senetence_Length_distribution.png')
```




```
<Figure size 432x288 with 0 Axes>
```

In [0]:
```
n_timesteps = 32
x_pad = pad_sequences(text_seq,padding='post',maxlen = n_timesteps)
```

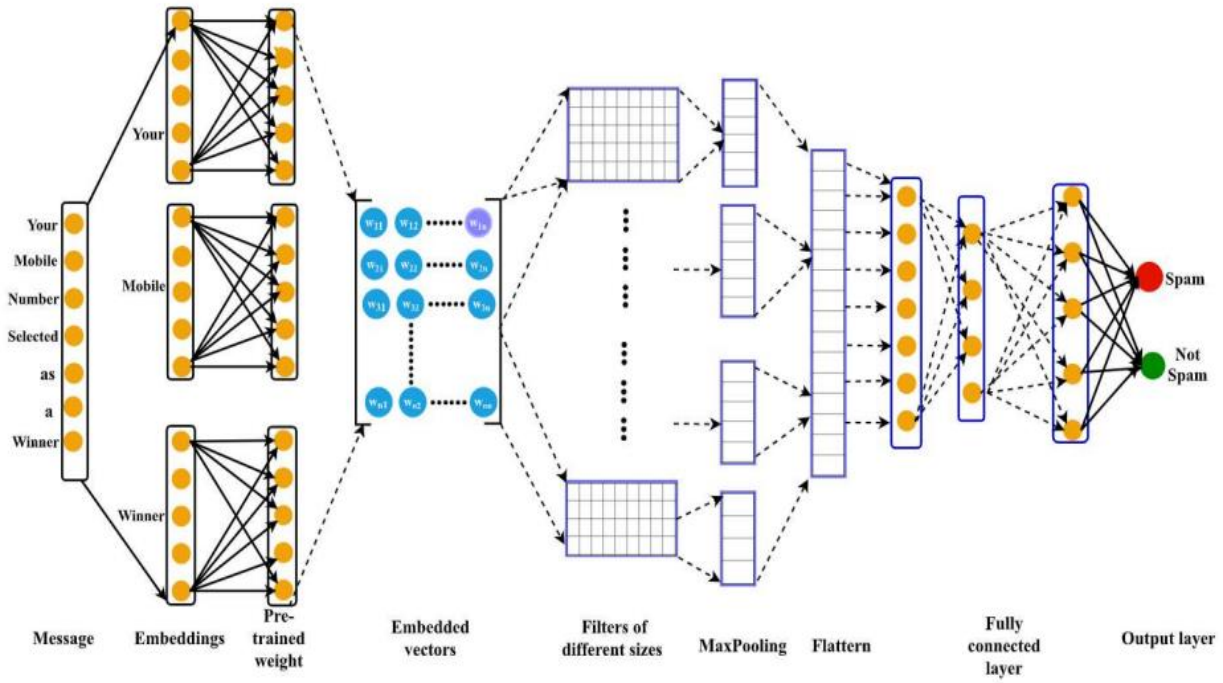
Proposed Solution and its Implementation

Convolutional neural network (CNN)

- CNN is one of the popular deep learning models which is able to extract the relevant features from the data.
- CNN mainly works in three phases:

1. creation of word matrix,
2. identifying the hidden features from the text, and
3. classify them into predefined classes.

Framework of Convolutional Neural Network



Working Of CNN

I. **Creation of word matrix**:

Every message *M* is the sequence of the words: w_1, w_2, w_3, …, w_n . From the given dataset, all unique word are bagged together to create a vocabulary *(V)* set. Each word of the V is assigned a unique integer value. From the pre-trained word vector called Glove , the word vector is extracted for every word present in the *V,* the word which is not present in the Glove are assigned random values. The word vector extracted from the Glove is represented as:

E(m) = e(w_1), e(w_2), e(w_3), … , *e(w_3,),* where e(w_1), e(w_2), e(w_3), … , *e(w_n)* are the individual word vectors of the words w_1, w_2, w_3, … , w_n. Finally, the word vectors e(w1), e(w2), e(w3), …. , *e(w,i)* are concatenated to create a complete SMS word matrix

M = e(w_1) • e(w_2) • e(w_3) ……..• e(w_n)

where • is the sign of concatenation. In general the SMS word matrix is represented as M1:n for the messages of the word 1 to n.

In this way an SMS word matrix $M \in \mathbb{R}^{d \times |n|}$ is created from every SMS.

$$\mathbf{M} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1d} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2d} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{n1} & w_{n2} & w_{n3} & \dots & w_{nd} \end{bmatrix}$$

II. Context dependent feature extraction phase:

A CNN network consists of a series of convolution and pooling layers. For convolution we have used different sizes of kernels: *F ∑* 2, 3, 4, 5 (i.e., 2-grams, 3-grams, 4-grams and 5-grams). A convolution operation over the SMS matrix *M* $M \in \mathbb{R}^{d \times |n|}$ and kernel F, F $\in \mathbb{R}^{d \times |n|}$

(where m = 2 is the region size of the kernel and *d* is the dimension) yields a feature matrix of dimension (|n| — F_m + 1) The process of finding the feature vector is show below:

$$\mathbf{M} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1d} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2d} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{n1} & w_{n2} & w_{n3} & \dots & w_{nd} \end{bmatrix} \odot \mathbf{Fm} = \begin{bmatrix} fm_{11} & fm_{21} \\ fm_{12} & fm_{22} \\ \vdots & \vdots \\ fm_{1d} & fm_{2d} \end{bmatrix}$$

The left matrix *M* is the message matrix having the n number of words, each word represented in *d* dimensional embedding vector. Initially, the messages are of different sizes as some of the messages have more number of words and some of them have fewer. However, the CNN network does not accept the inputs having different lengths. Therefore, before creating the message matrix (M), we used the post padding technique to make the messages of equal length. The right matrix is the (Fm) is the 2-gram kernel which slides vertically over the message matrix (M) and finds the features O. The features O_i is calculated as follows:

$$\text{Where } O_1 = w_{11}fm_{11} + w_{12}fm_{12} + \cdots + w_{1d}fm_{1d} + w_{21}fm_{21} + w_{22}fm_{22} + \cdots + w_{2d}fm_{2d}$$

$$O_2 = w_{21}fm_{11} + w_{22}fm_{12} + \cdots + w_{2d}fm_{1d} + w_{31}fm_{21} + w_{32}fm_{22} + \cdots\cdots + w_{3d}fm_{2d}$$

and

$$O_n = w_{(n-1)1}fm_{11} + w_{(n-1)2}fm_{12} + \cdots + w_{n1}fm_{21} + w_{n2}fm_{22} + \cdots\cdots\cdots + w_{nd}fm_{2d}$$

These features were stored in a matrix K, the dimension of the matrix K is *(n* — 2 + 1) x 1, the features 01, 02, 03, ..„0„ are of context-dependent features extracted using the convolution operation.

$$\mathbf{C} = \begin{bmatrix} O_1 \\ O_2 \\ \vdots \\ O_n \end{bmatrix}$$

The feature matrix K is passed through an activation function called Rectified linear unit (ReLU) ReLU is defined as follows (Eq (3)).

$\sigma(u) = max(0, u)$ (3)

Here u is a positive value. ReLU activation function returns the positive value for all positive and 0 for others. The resulting values are stored in a separate matrix *K'.* In *K'* only positive values are present, the dimension of the K' is also (n — 2 +1) x 1. We identified the hidden features from the text using the convolution operations, but all the features may not be equally important. Hence, to identify the important ones, we used another function called pooling. Pooling has different variants such as: max-pooling and min-pooling and average pooling. We checked all the three variants and found that max-pooling operation yielded the most promising performance. Hence we have used the max-pooling operation with the window size k. Window size k defines the number of elements out of which a value is pulled out (Eq. (4)). For example, if the window size *k* = 5, then out of 5 features, a value (maximum) is pulled out in max-pooling operation.

$0_i = max(O_1, 0_2, 0_3, \ldots, O_k)$

$$\tilde{O}_i = max(O_1, O_2, O_3, \ldots, O_k) \quad (4)$$

In such way, we get the vector of important features: $\tilde{O} = [\tilde{O}_1, \tilde{O}_2, \tilde{O}_3, \ldots \tilde{O}_L]$ where L is defined as:

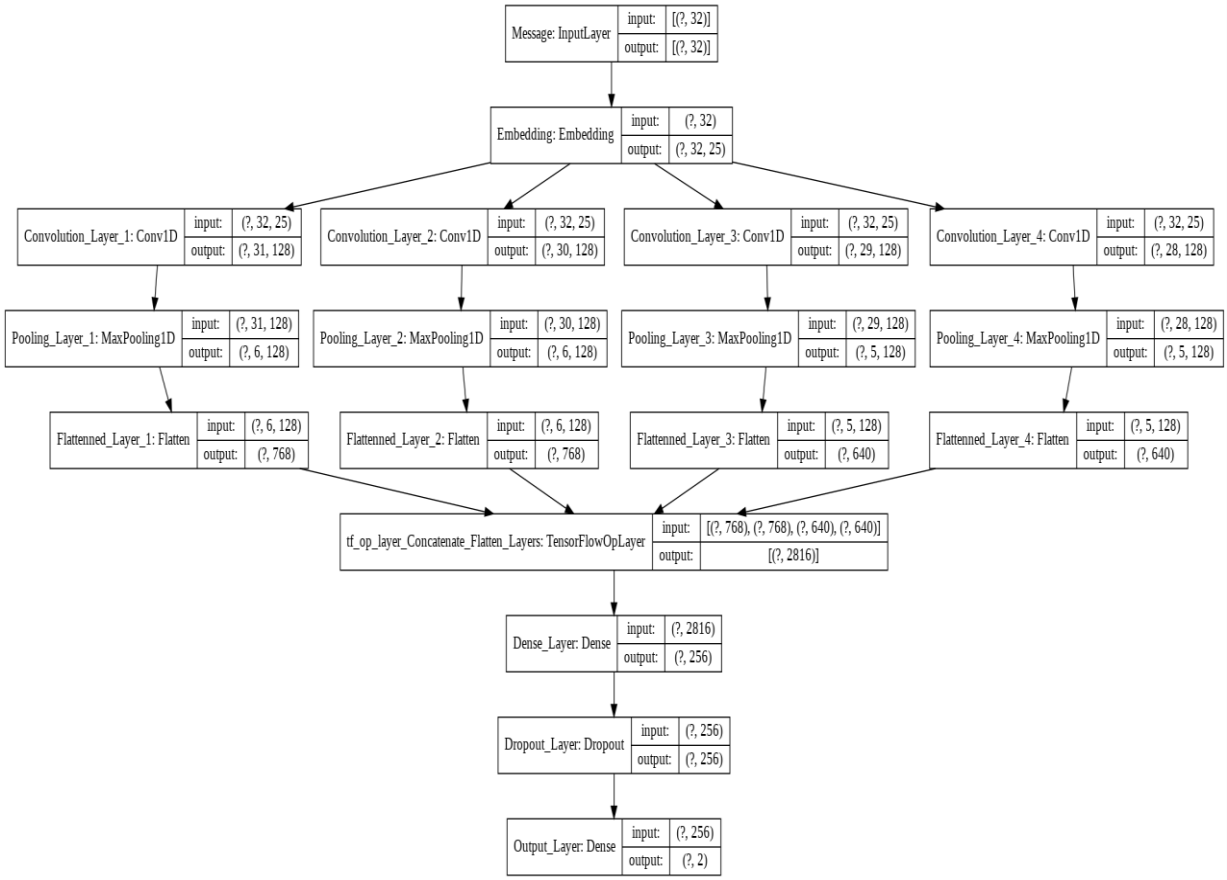
$$L = \lfloor \frac{\tilde{O}_n}{k} \rfloor \quad (5)$$

At the end of proposed CNN model, a fully connected multi-layer perceptron performed the classification task. The features identified using the convolution and pooling layer i.e.: $\tilde{O} = [\tilde{O}_1, \tilde{O}_2, \tilde{O}_3, \ldots \tilde{O}_L]$ is given to this dense layer to classify the messages into predefined classes. On output layer, a Softmax function [66] is applied to decide the probability of each message for the classes present at output layer. The Softmax functions is defined as (Eq. (6)):

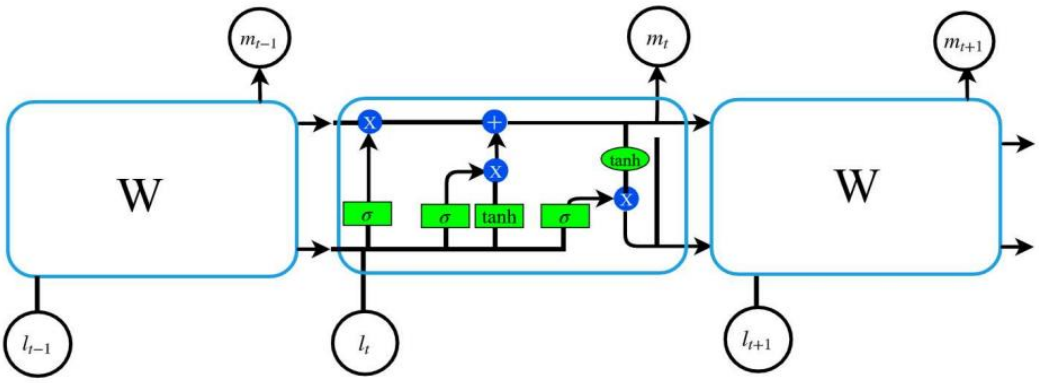
$$\sigma(w)_j = \frac{e^{w_j}}{\sum_{class=1}^{N} e^{w_{class}}} \quad (6)$$

- **Kernel size:** CNN support the multi-gram of kernel sizes, we used combinations of all types of kernel together i.e., the kernel size of 2, 3, 4, 5 together gives the best performance for our case as mentioned in the source paper.
- **Dropout** : CNN support the regularization operator called dropout. Dropout are generally used to reduce the complexity between the links present in the fully connected dense layer We have used a value of 0.3 as mentioned in the paper.
- **Optimizer:** The role of the optimizer is to improve the accuracy of the model by reducing the error rate and we have used Adam optimizer.
- **Activation Function :** At the output layer, an activation function is used to decide the probability of the message and We have used softmax activation function as mentioned in the source paper.

Our CNN Implementation



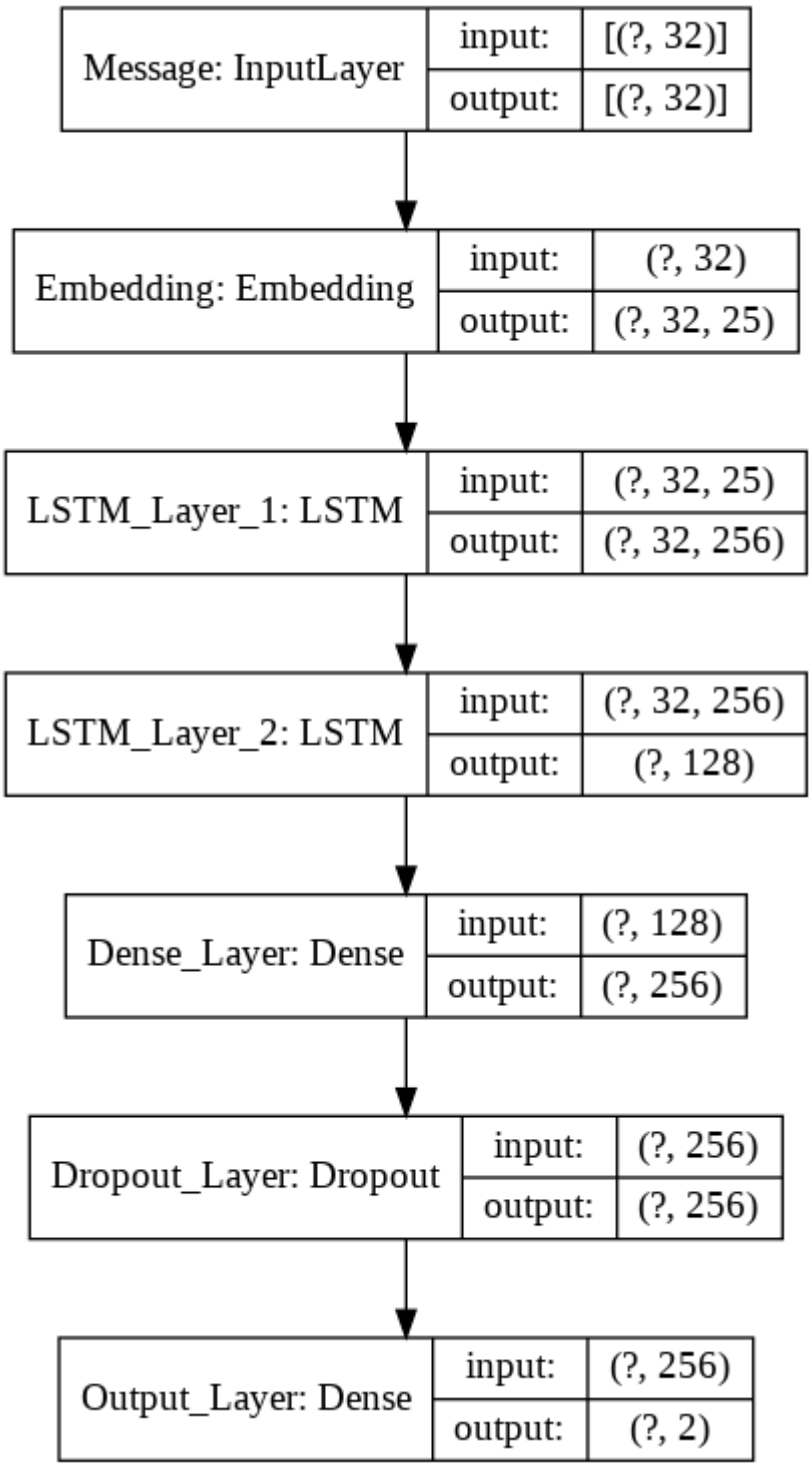
Long Short Term Memory (LSTM)



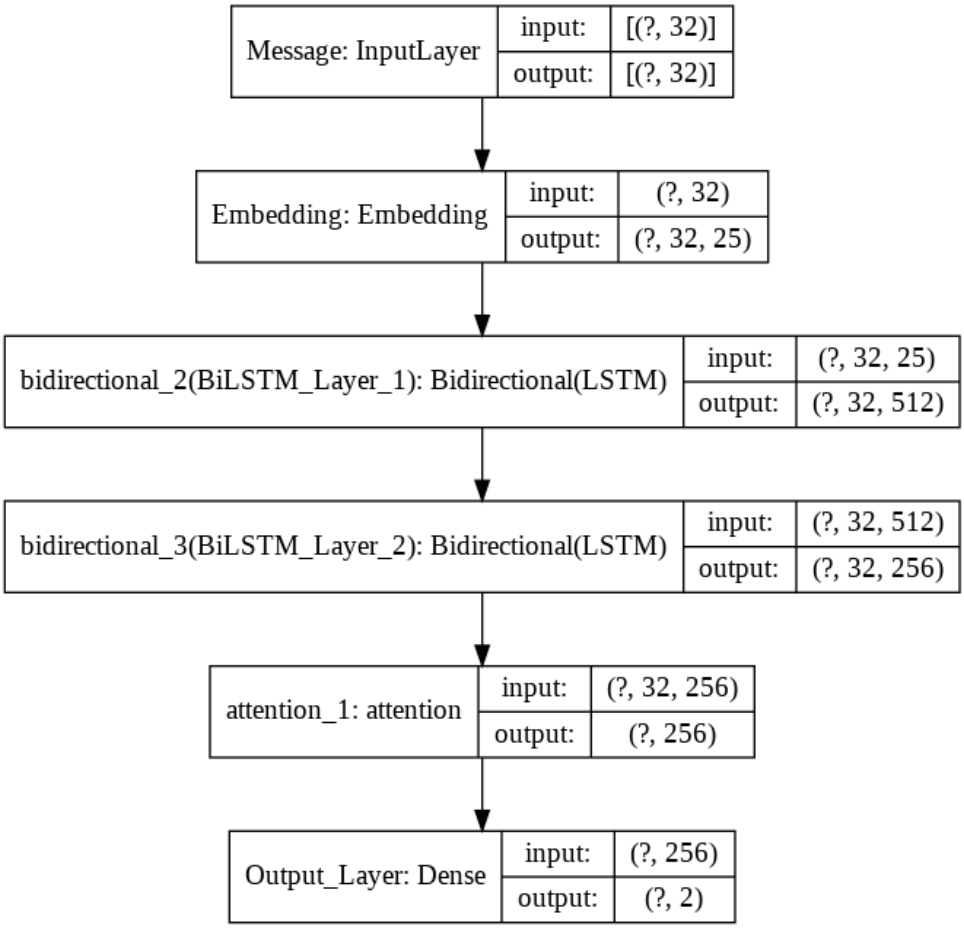
- CNN can extract hidden features from the text. However, it is unable to remember the long sequences of the text.
- The LSTM network is able to do so. As shown in Fig. in the previous slide. The LSTM network mainly has four different gates namely, input gate *(i_t),* an output gate (o_t), a memory cell m_t, a forget gate *(f_t)* and a hidden state *(h_t).* At every time stamp t, a word vector it is given to the LSTM network which processes it and yields an output m_i as shown in Fig. in the previous slide. The first step of the LSTM network is to find the information that is not relevant and throw away from the cell state. This decision is taken by the very first layer of the network i.e., Sigmoid layer which is called forget gate layer *(f_t)* .

We use the LSTM network to predict whether the message is a Spam or Not-Spam using the simple model and with the regularization parameter i.e., Dropout.

Simple LSTM Based Implementation



Attention Based BiLSTM Implementation



Hyper-parameter tuning and training of the model

- All the hyper-parameters were taken according to as mentioned in the paper so not much tweaking was done.
- Though we tried with Glove embeddings because there was no information about which embedding and of what dimension to be used, after trying we found that twitter crawled embeddings were working better than Wikipedia crawled pretrained embeddings.

Performance Evaluation

- To evaluate the performance of the proposed model, we used the well-known metrics for the classification techniques such as, Precision (P), Recall (R), F1-Score (F1), Accuracy and Area Under Receiver Operating Curve.

Precision (P): It is defined as the fraction of circumstances in which the correct SMS Spams is returned (Eq. (13)).

$$Precision\ (P) = \frac{T_p}{T_p + F_p} \tag{13}$$

Recall (R): It is defined as proportion of actual SMS Spams is predicted correctly. Mathematically it is defined in Eq. (14).

$$Recall\ (R) = \frac{T_p}{T_p + F_n} \tag{14}$$

F1-Score (F1): It is defined as harmonic mean of the precision and recall as given in Eq. (15).

$$F1\text{-}Score\ (F1) = 2 * \frac{P * R}{P + R} \tag{15}$$

Accuracy: It is the fraction of SMS Spams Messages that were correctly predicted among the SMS Messages (Eq. (16)).

$$Accuracy\ (A) = \frac{T_p + T_n}{T_p + F_p + T_n + F_n} \tag{16}$$

We also find the Receiver Operating Characteristic (ROC) curve and find the area under the ROC curve. ROC curve is the plot between True positive rate (TPR) (Eq. (17)) and False positive rate (FPR) (Eq. (18)). The area under the ROC curve is used to measure the accuracy of the classifier. Greater the AUC value greater is the accuracy of the model.

$$TPR = \frac{T_p}{T_p + F_n} \tag{17}$$

$$FPR = \frac{F_p}{F_p + T_n} \tag{18}$$

Comparison of our results with other Models

Using Traditional Machine Learning Models (taken from paper)

Classifier	Class	Precision (P)	Recall (R)	F1-Score (F1)
NB	Spam	0.549	0.651	0.596
	Not-Spam	0.550	0.469	0.506
RF	Spam	0.962	0.763	0.851
	Not-Spam	0.804	0.971	0.880
GB	Spam	0.993	0.809	0.891
	Not-Spam	0.839	0.994	0.910
LR	Spam	0.581	0.449	0.507
	Not-Spam	0.551	0.676	0.607
SGD	Spam	0.501	0.753	0.602
	Not-Spam	0.504	0.254	0.338

Using Proposed Architecture (Our Implemented Model)

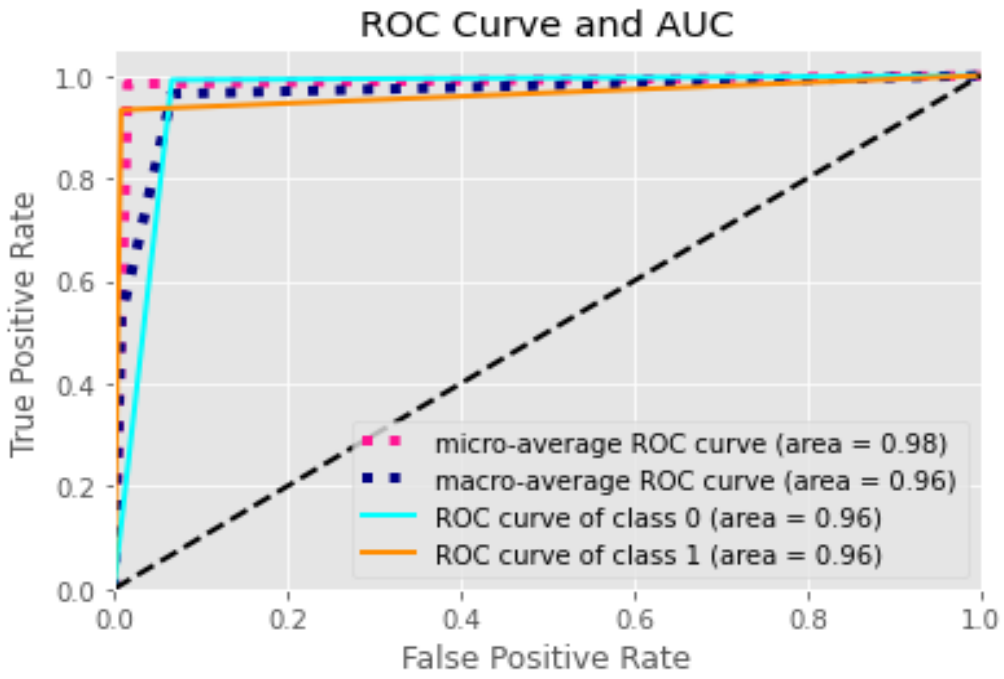
Classifier	Class	Precision (P)	Recall (R)	F1-Score (F1)
CNN	Spam	0.95	0.93	0.94
	Not-Spam	0.99	0.99	0.99
LSTM	Spam	0.92	0.95	0.94
	Not-Spam	0.99	0.99	0.99
BiLSTM+Attention	Spam	0.94	0.96	0.95
	Not-Spam	0.99	0.99	0.99

Source Paper Model (Not 10Fold CV)

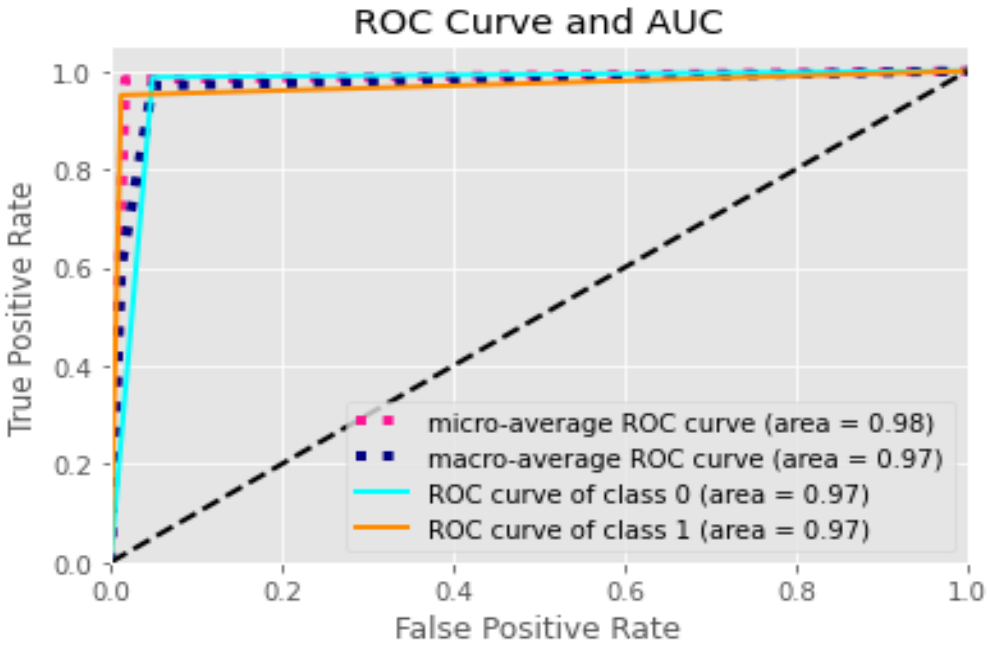
Classifier	Class	Precision (P)	Recall (R)	F1-Score (F1)
CNN	Spam	0.976	0.918	0.946
	Not-Spam	0.993	0.987	0.989
LSTM	Spam	0.896	0.842	0.868
	Not-Spam	0.977	0.989	0.982

ROC Curve And AUC

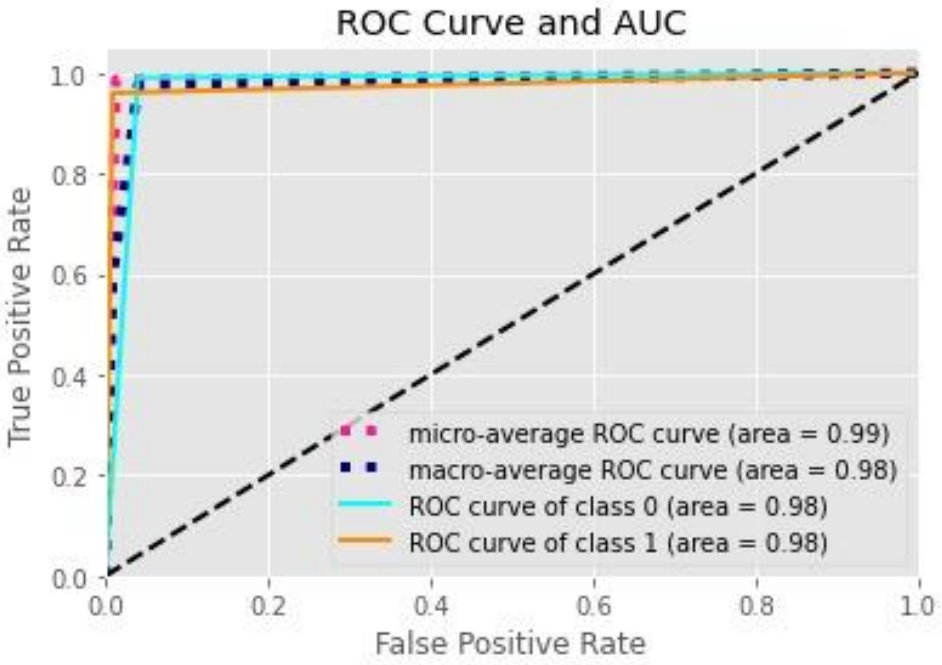
For CNN



For LSTM

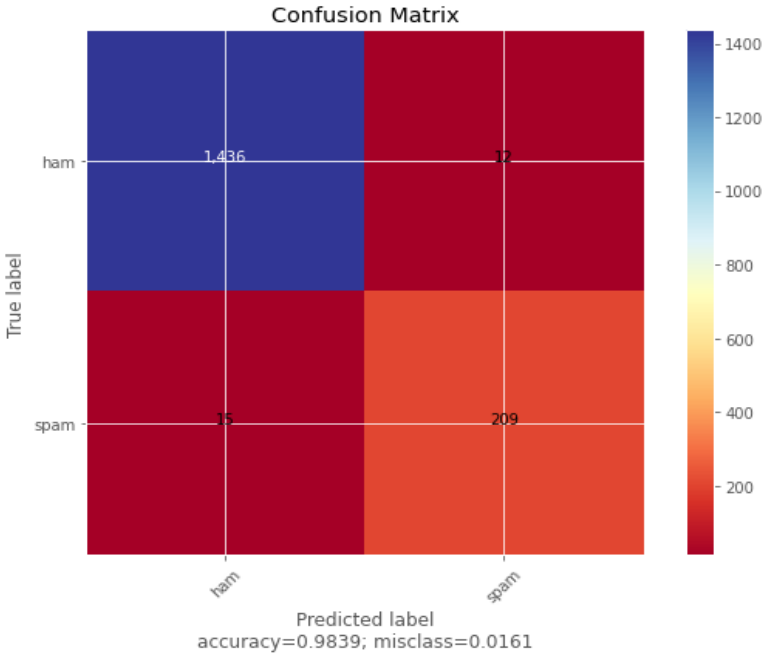


For BiLSTM+Attention

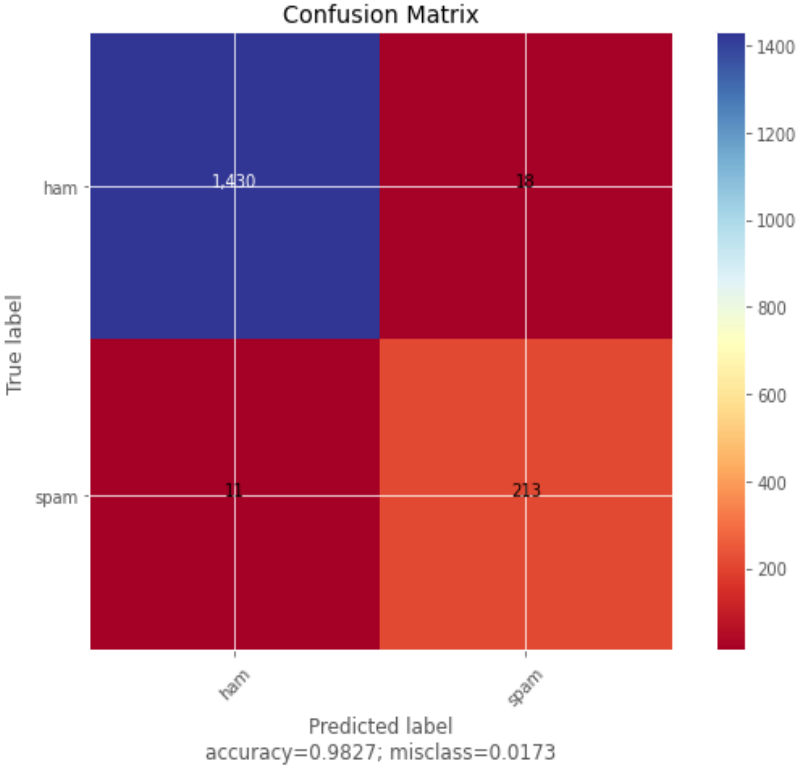


Confusion Matrix

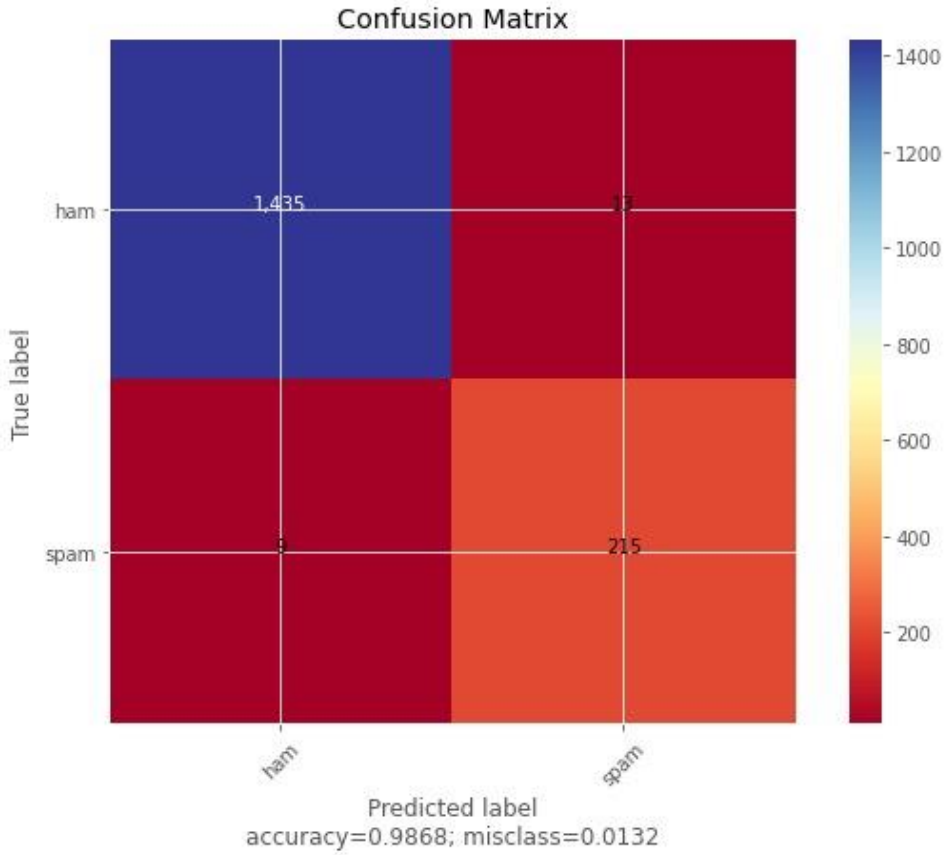
For CNN



For LSTM



For BiLSTM+Attention



Conclusion

- For the CNN Model, We managed to get close reported score and AUC value but can't reach completely with a AUC value of 96.23 instead of AUC score 96.8 and accuracy of 98.38% instead of 98.63% which is reported in paper.
- However out 10 fold Cross validation doesn't performed very well with an accuracy of 97.33% as compared to papers ground breaking 99.44%. Though I believe with increased no of epochs and some other minor tweaking we can get much close to the score if we have time.
- Also, We tried with both Glove's twitter text and wikipedia crawled text pretrained weights and found that twitter text pretrained weights are giving higher accuracy
- For the Simple LSTM model We got accuracy more than mentioned in paper with a value of 98.26% instead of 96.67% which is mentioned in paper
- Also we tried another Attention based implementation with BiLSTM network which is not mentioned in paper but found that It fetches better score than the Simple LSTM model with an accuracy of 98.68% and remarkable AUC value of 97.54 instead of what was with simple LSTM i.e. 98.26% accuracy and AUC of 96.92

References & Paper Used

- Paper Used : Deep learning to filter SMS Spam
- Attention? Attention
- Attention-based bidirectional LSTM for Classification Task (ICASSP) Paper
- Glove Vectors
- Bahdanau Attention AnalyticsVidhya

Note:

A large portion of this paper's theory has been copied/taken from the source Paper…..