Spam Email Filter

Sakchi Shrestha
Department of Computer Science and Engineering
University of Texas at Arlington
Arlington, USA
Email: sakchi.shrestha@gmail.com

Shaina Ayer
Department of Computer Science and Engineering
University of Texas at Arlington
Arlington, USA
Email: ayershaina567@gmail.com

***Abstract*—The growing use of the internet has led to issues like email phishing, spreading viruses and malwares, and many more. Many people receive internet spams on a daily basis, that consists of irrelevant texts sent in the form of messages or emails. To detect these email spams, we use Natural Language Processing to train the email spam detector to recognize and classify emails into spam and no-spam. With Natural Language Processing, our machine can make sense of written text and perform tasks including speech recognition, sentiment analysis, and automatic text summarization. Binary classification model is used to detect whether an email message is spam. Machine learning algorithms such as Long-Short-Term Memory, Logical Regression, Dense Classifier Sequential Neural Networks were used as Deep Learning models. Other approaches ranging from white/black list, Bayesian analysis, content scanning can also be used as a spam filter. The Long-Short-Term-Memory model possess a training and validation score with the overall accuracy of 94 percent. On the other hand, Dense Classifier Sequential Neural Network shows very high training and validation score with overall accuracy of 97 percent. More research on other parameters with other types of deep learning models can also be used in order to improve the accuracy of email phishing detection. The results of the models are discussed below.**

***Index Terms*—Email Spam, Dense Classifier Sequential Neural Network, Long-Short-Term Memory, Natural Language Processing, Deep Learning, Phishing, Logical Regression**

I. Introduction

In this era of technology, Internet has been one of the prominent aspect of our daily life. From checking important emails, or connecting with your friends on social medias, to writing reviews in Amazon, all of these would not be possible without the Internet. This source of gathering information worldwide also led to Internet Spam. [1] Internet Spam is simply a collection of irrelevant texts or links sent in the form of emails or text messages with the motive of spreading malwares or false advertisement of vulnerable websites. If we unintentionally click those spam links, then virus will be automatically transferred to the system, and thus, leaking all of our critical information. Once transmitted, the virus can also send the same spam links/emails to our contacts. So, spam message is a threat to both individual and society. That's why spam email filter is necessary to filter out all the spam messages sent to the user's inbox. [2]

Cyber criminals mostly conduct social-engineering-based-attacks that allow the attackers to perform identity theft known as Phishing Attack. According to the study conducted by the State of the Phish 2022, Phishing attacks have surged globally with the increase of 12 percent per year. [2] So, text processing techniques like Natural Language Processing play a vital role in Email Phishing Detection. Most researchers use supervised ML algorithm for spam detection, such as Naive Baiyes, SVM, Decision Tree. Incorporating two or more supervised ML algorithms help to improve overall training and validation accuracy. [3] Here, we demonstrate two of the spam detection techniques using Dense Classifier Sequential Neural Network and Long-Short-Term-Memory (LSTM). The model to implement in this ML techniques will be "Classifier" which gives binary outputs- either spam or ham. [4]

The objective of this research is to implement and build a Deep Learning Model for email spam detection. We incorporate Dense Sequential Classifier Neural Network, Long-Short-Term-Memory Networks (LSTM).Next, we compare the overall accuracies and select the best ML technique. Our main contributions towards email spam detection in this research are illustrated as follows:

- We perform statistical analysis and visualization of the data through tokenization, sequential and padding techniques.
- We use Keras Sequential model which consists of Embedding Layer, Pooling Layer and Dense Layer.
- We note that the model performs better on the training set and takes relatively less time in the neural network compared to the LSTM network.
- We use UCI Datasets to evaluate the performance of this spam detection project. We split this data into train and test sub-datasets.
- Our Dense Spam Detection Model outperforms other ML techniques based on loss and accuracy.

The rest of this paper is pictured as follows: Section II illustrates literature review, Section III illustrates the implementation of the datasets and utilized tools, Section IV visualizes the output and compares the results, Section V summarizes the outcomes of this experiment.

II. Literature Review

The highlighted features which are in the email header will be used to identify and classify spam messages efficiently. The features are shortlisted on the basis of their performance in detecting spam messages which will be a generic spam messages since it will communalize the features in Yahoo mail. Gmail and Hotmail. The new approach is based on the

strategy of how frequently used words are repeated. Firstly the key sentences from the incoming emails should be tagged and then the grammar of the entire word in the sentence. After that, they are arranged together in a vector order. K - Mean algorithm is used to classify the received email through vector determination. The main function of vector determination is used to determine the category to which the email belongs too. Algorithms like support vector machines, logistic regression, regression trees, and random forests are considered for the classification. The phishing Corpus dataset is used where the extraction approach is through the help of a Bag of words.

It is known that cyber attacks like phishers and malicious attackers use email for sending false information to the targeted users where they can potentially lose money and their social reputations. Ultimately, it can result in gaining credentials and other confidential data. Therefore, the Bayesian classifiers are used to consider every single letter and word in the mail. The Bayesian classifier is used as the model to classify emails as spam or not spam on the basis of their context. To use the Bayesian classifier, the classifier should be trained on a set of labeled emails and after that, it calculates the probability that a new email is a spam. if the probability is above some of the thresholds then the email is classified as spam. The main advantage of the Bayesian classifier easily updates as new emails are received also they are even effective if the spammers change their tactics.

Machine learning techniques are used to detect a pattern of repetitive keywords through a proposed system that is classified as spam. Various parameters are used for the classification of email in the structure such as header, domain, and Cc/Bcc and each parameter is considered a feature when it is applied to the Machine learning algorithm. The feedback mechanism is used through the machine learning model of a pre-trained model to distinguish between a proper and ambiguous output.

The string matching algorithms namely Longest Common Subsequence(LCS), Levenshtein Distance(LD), jaro, jaro-winkler, Bi-gram and TFIDF on the various dataset. These are mostly Enron corpus and CSDMC2010 spam dataset and which is mostly observed that a Bi-gram algorithm performs the best in the spam detection.

III. Working Methodology

Natural Language Processing(NLP) plays a vital role in filtering out spam emails by analyzing the contents of email messages. It mostly focuses on the interaction between computers and humans through natural language.Spam email detection using NLP involves several steps that are illustrated below.

- **Text Preprocessing:** Raw email messages are preprocessed to remove any unwanted characters such as punctuation, HTML tags and special characters. Then, those messages are tokenized into words or phrases.
- **Feature Extraction:** Frequency of occurence of certain words or the length of the messages and many more are extracted from the messages. These data would help to train a machine learning model to differentiate between spam or non-spam emails.
- **Text Classification:** The machine learning model is trained on a spam emails datasets which classifies them as spam or non-spam based on the frequencies and length calculated.
- **Spam Filtering:** Once the model is fully trained, it can filter incoming emails and distinguish them. Spam emails detected would be automatically deleted or moved to separate folder named ”Spam/Junk”.

Overall, NLP plays a crucial role in spam email detection by improving email experiences for the user.

Dense Classifier Sequential Neural Network: A Dense Classifier SNN is a neural network that comprises of multiple layers of densely connected neurons, which can also filter out spam email messages. This neural network comprises of various layers that are illustrated below.

- **Input Layer:** Here, preprocessed email messages are taken as an input.
- **Embedding layer:** After the preprocessed emails are retrieved as input, Embedding layer converts those into a vector of a fixed length by mapping each word to a high-dimensional vector space.
- **Dense Layer:** The densely connected neurons performs computations on the inputs. The output of the first dense layer is passed through non-linear activation function to introduce non-linearity into the network. The received output is again passed to the next layer, where same process is repeated. The number of neurons in each dense layer can also optimize the network performance for a specific task. It plays a crucial tole in extracting essential features from a given datasets, and transform them into classification decision.
- **Output Layer:** The output layer produces a final binary output which indicates whether an email is spam or not.

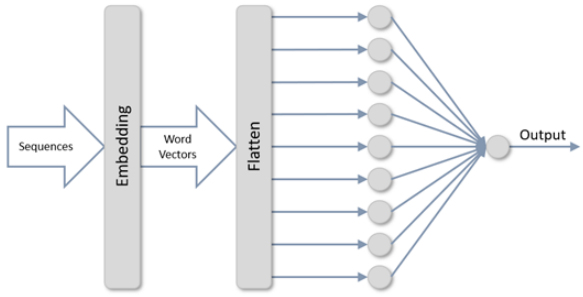

Fig. 1. Architecture of Dense Classifier Sequential Neural Network

Long-Short-Term Memory: LSTM is a recurrent neural network (RNN) that models sequential data by maintaining past memory inputs. It consists many memory cells that maintains the flow of information into and out of the cells. Overall, LSTM networks can be a powerful tool for spam email detection and can provide high accuracy in filtering out unwanted messages.The architecture of an LSTM network for spam email detection consists of following layers.

- **Input Layer:** This layer takes preprocessed emails as input.
- **Embedded Layer:** This layer converts the inputs into a vector of a fixed length by mapping each word to a high-dimensional vector space.
- **LSTM Layer:** LSTM layer maintains the information flow into and out of the cells. It also maintains a memory of past inputs and identifies patterns in input.
- **Dense Layer:** This layer consist of densely connected neurons that perform computations on the output of the LSTM layers.
- **Output Layer:** The output layer produces a final binary output which indicates whether an email is spam or not.

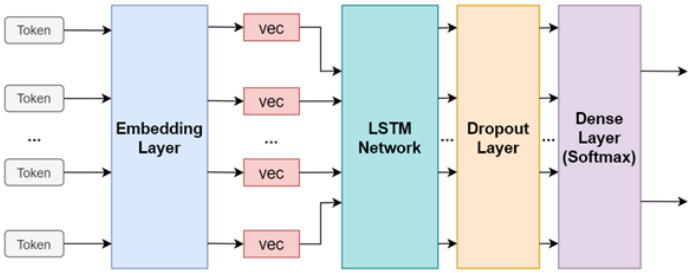

Fig. 2. LSTM Networks for Spam Email Detection

IV. Dataset and Experimental Settings

We will use the text data from UCI Datasets for this spam email detection project. The data contains 5,572 spam messages labelled spam or not spam(ham). We split the dataset as training and testing. We use latest version of Python as a platform to develop the project. Next, we use libraries available in Python such as pandas, numpy and many more for the preprocessing tasks. We also employ tensorflow (version greater than 2.0), keras, scikit-learn, wordcloud, matplotlib libraries to evaluate the performance of our project. We use spam email message classification dataset. The description of dataset is given in the table below.

Dataset	Total Messages	Total Spam Message (%)
Spam SMS	5,572	13.4

TABLE I
Description of the Dataset

V. Spam Detection Using Machine Learning

1) First, we would use Matplotlib to plot histogram and graphs, pandas for manipulation and analysis of data, Word Cloud to present the text data and NumPy for any mathematical operations. We import all the required packages which are shown below.

```
In [2]: import numpy as np
        import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
        from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

        %matplotlib inline

        # deep learning libraries for text pre-processing
        import tensorflow as tf
        from tensorflow.keras.preprocessing.text import Tokenizer
        from tensorflow.keras.preprocessing.sequence import pad_sequences
        from tensorflow.keras.callbacks import EarlyStopping
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Embedding, GlobalAveragePooling1D, Dense, Dropout, LSTM, Bidirectional
```

Fig. 3. Required packages

2) Next, we load and explore the data. We will only get three rows of the dataset to make sure that the data has been loaded properly.

```
In [5]: file = 'C:\\Users\\sakch\\Downloads\\spam'
        messages = pd.read_csv(file, sep ='\t',names=["label", "message"])
        messages[:3]
```

Out[5]:

	label	message
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...

Fig. 4. Loaded dataset

3) To get a summary details of the data, we use .describe function. The 'messages.describe()' gives us the message count, two unique labels which are "spam" and "ham", and unique message count. The output is shown below.

```
In [6]: messages.describe()
```

Out[6]:

	label	message
count	5572	5572
unique	2	5169
top	ham	Sorry, I'll call later
freq	4825	30

Fig. 5. Summary of the loaded spam messages

4) Since we have number of duplicate messages, we store them on a separate variable 'duplicatedRow'.

```
In [7]: duplicatedRow = messages[messages.duplicated()]
        print(duplicatedRow[:5])
```

	label	message
103	ham	As per your request 'Melle Melle (Oru Minnamin...
154	ham	As per your request 'Melle Melle (Oru Minnamin...
207	ham	As I entered my cabin my PA said, '' Happy B'd...
223	ham	Sorry, I'll call later
326	ham	No calls..messages..missed calls

Fig. 6. Summary of duplicate messages

5) Next, we group the data by labels. The below figure indicates that the data of spam and ham messages are imbalanced that we need to fix.

```
In [8]: messages.groupby('label').describe().T
```

Out[8]:

	label	ham	spam
message	count	4825	747
	unique	4516	653
	top	Sorry, I'll call later	Please call our customer service representativ...
	freq	30	4

Fig. 7. Labelling of messages

6) We need to find out the most repeated words in ham and spam messages. So, we use WordCloud library.

```
In [10]: # Get all the ham and spam messages
ham_msg = messages[messages.label =='ham']
spam_msg = messages[messages.label=='spam']

# Create numpy list to visualize using wordcloud
ham_msg_txt = " ".join(ham_msg.message.to_numpy().tolist())
spam_msg_txt = " ".join(spam_msg.message.to_numpy().tolist())

# wordcloud of ham messages
ham_msg_wcloud = WordCloud(width =520, height =260, stopwords=STOPWORDS,max_font_size=50, background_color ="red",
                           colormap='Blues').generate(ham_msg_txt)
plt.figure(figsize=(16,10))
plt.imshow(ham_msg_wcloud, interpolation='bilinear')
plt.axis('off') # turn off axis
plt.show()
```

Fig. 8. Creating WordCloud for ham messages

Fig. 9. Wordcloud for ham messages

```
In [12]: # wordcloud of spam messages
spam_msg_wcloud = WordCloud(width =520, height =260, stopwords=STOPWORDS,max_font_size=50, background_color ="black",
                            colormap='Spectral_r').generate(spam_msg_txt)
plt.figure(figsize=(16,10))
plt.imshow(spam_msg_wcloud, interpolation='bilinear')
plt.axis('off') # turn off axis
plt.show()
```

Fig. 10. Creating Wordcloud for spam messages

Fig. 11. Wordcloud for spam messages

7) We need to plot a bar graph to estimate the percentage of spam and ham ratio to visualize the imbalanced data.

Out[14]: 15.481865284974095

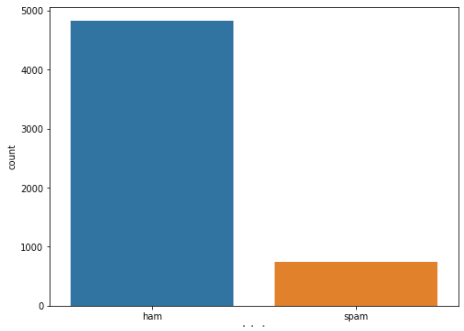

Fig. 12. Bar graph showing the percentage of spam and ham ratio

8) We use downsampling method to deal with the imbalanced data issue. One way to solve is to downsample the ham message count to the spam message count.

Out[16]: Text(0.5, 0, 'Message labels')

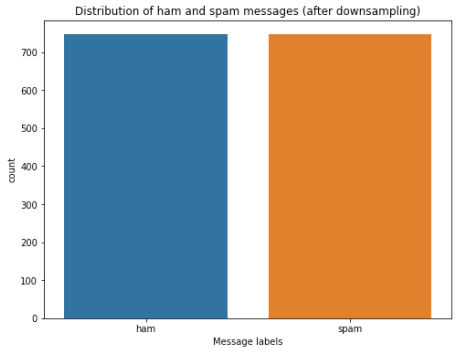

Fig. 13. Bar graph showing the balanced percentage of spam and ham ratio

9) We split the data into training and testing sets.

```
In [25]: # Map ham label as 0 and spam as 1
msg_df['msg_type']= msg_df['label'].map({'ham': 0, 'spam': 1})
msg_label = msg_df['msg_type'].values

# Split data into train and test
train_msg, test_msg, train_labels, test_labels = train_test_split(msg_df['message'], msg_label,
                                                                  test_size=0.2, random_state=434)
```

Fig. 14. Splitting the datasets into training and testing

10) We also need to preprocess the input messages and use tokenizer() to tokenize the words into numerical representation.

```
In [30]: # Defining pre-processing hyperparameters
max_len = 50
trunc_type = "post"
padding_type = "post"
oov_tok = "<OOV>"
vocab_size = 500

tokenizer = Tokenizer(num_words = vocab_size, char_level=False, oov_token = oov_tok)
tokenizer.fit_on_texts(train_msg)

# Get the word_index
word_index = tokenizer.word_index
tot_words = len(word_index)
print('There are %s unique tokens in training data. ' % tot_words)

There are 4169 unique tokens in training data.
```

Fig. 15. Preprocessing the input

11) The next preprocessing task are: Sequencing which represent each sentence by sequences of numbers and Padding which ensures each sequence will have same length.

```
In [33]:
training_sequences = tokenizer.texts_to_sequences(train_msg)
training_padded = pad_sequences (training_sequences, maxlen = max_len, padding = padding_type, truncating = trunc_type )

#test
testing_sequences = tokenizer.texts_to_sequences(test_msg)
testing_padded = pad_sequences(testing_sequences, maxlen = max_len,padding = padding_type, truncating = trunc_type)
```

Fig. 16. Sequencing and padding

VI. Results

- **Dense Classifier Sequential Neural Network:**

1) We design the architecture of Dense Sequential model by tuning the hyper parameters first.The Dense layer in this architecture outputs the probabilities between ham(0) and spam(1). We then look at the summary of the model.

```
In [36]:
vocab_size = 500
embeding_dim = 16
drop_value = 0.2
n_dense = 24

#Dense Sequential model architecture
model = Sequential()
model.add(Embedding(vocab_size, embeding_dim, input_length=max_len))
model.add(GlobalAveragePooling1D())
model.add(Dense(24, activation='relu'))
model.add(Dropout(drop_value))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

Fig. 17. Tuning the hyperparameters

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 50, 16)            8000

global_average_pooling1d (G  (None, 16)                0
lobalAveragePooling1D)

dense (Dense)                (None, 24)                408

dropout (Dropout)            (None, 24)                0

dense_1 (Dense)              (None, 1)                 25

=================================================================
Total params: 8,433
Trainable params: 8,433
Non-trainable params: 0
_________________________________________________________________
```

Fig. 18. Summary of Dense Sequential Model

2) Our model has been compiled and ready to fit in the training data.

```
In [38]:
model.compile(loss='binary_crossentropy',optimizer='adam' ,metrics=['accuracy'])
```

Fig. 19. Compiled Model

3) The results we got after 30th epoch are: training loss: 0.04, training accuracy: 98 percent , validation loss: 0.1, validation accuracy: 96 percent.

```
In [40]:
num_epochs = 30
early_stop = EarlyStopping(monitor='val_loss', patience=3)
history = model.fit(training_padded, train_labels, epochs=num_epochs, validation_data=(testing_padded, test_labels),
                    callbacks =[early_stop], verbose=2)
```

Fig. 20. Getting the results after epoch

```
Epoch 1/30
38/38 - 1s - loss: 0.6868 - accuracy: 0.5456 - val_loss: 0.6785 - val_accuracy: 0.6020 - 1s/epoch - 33ms/step
Epoch 2/30
38/38 - 0s - loss: 0.6681 - accuracy: 0.7556 - val_loss: 0.6520 - val_accuracy: 0.7559 - 90ms/epoch - 2ms/step
Epoch 3/30
38/38 - 0s - loss: 0.6269 - accuracy: 0.8167 - val_loss: 0.5934 - val_accuracy: 0.8261 - 90ms/epoch - 2ms/step
Epoch 4/30
38/38 - 0s - loss: 0.5484 - accuracy: 0.8594 - val_loss: 0.5063 - val_accuracy: 0.8528 - 86ms/epoch - 2ms/step
Epoch 5/30
38/38 - 0s - loss: 0.4562 - accuracy: 0.8887 - val_loss: 0.4247 - val_accuracy: 0.8729 - 95ms/epoch - 2ms/step
Epoch 6/30
38/38 - 0s - loss: 0.3712 - accuracy: 0.8979 - val_loss: 0.3560 - val_accuracy: 0.8829 - 110ms/epoch - 3ms/step
Epoch 7/30
38/38 - 0s - loss: 0.3085 - accuracy: 0.9088 - val_loss: 0.3065 - val_accuracy: 0.8896 - 75ms/epoch - 2ms/step
Epoch 8/30
38/38 - 0s - loss: 0.2568 - accuracy: 0.9255 - val_loss: 0.2654 - val_accuracy: 0.8963 - 83ms/epoch - 2ms/step
Epoch 9/30
38/38 - 0s - loss: 0.2241 - accuracy: 0.9297 - val_loss: 0.2300 - val_accuracy: 0.9030 - 114ms/epoch - 3ms/step
Epoch 10/30
38/38 - 0s - loss: 0.1895 - accuracy: 0.9414 - val_loss: 0.1997 - val_accuracy: 0.9130 - 106ms/epoch - 3ms/step
Epoch 11/30
38/38 - 0s - loss: 0.1653 - accuracy: 0.9481 - val_loss: 0.1819 - val_accuracy: 0.9331 - 83ms/epoch - 2ms/step
Epoch 12/30
38/38 - 0s - loss: 0.1496 - accuracy: 0.9515 - val_loss: 0.1641 - val_accuracy: 0.9398 - 99ms/epoch - 3ms/step
Epoch 13/30
38/38 - 0s - loss: 0.1362 - accuracy: 0.9607 - val_loss: 0.1525 - val_accuracy: 0.9398 - 93ms/epoch - 2ms/step
Epoch 14/30
38/38 - 0s - loss: 0.1281 - accuracy: 0.9615 - val_loss: 0.1422 - val_accuracy: 0.9465 - 84ms/epoch - 2ms/step
Epoch 15/30
38/38 - 0s - loss: 0.1120 - accuracy: 0.9657 - val_loss: 0.1397 - val_accuracy: 0.9431 - 86ms/epoch - 2ms/step
Epoch 16/30
38/38 - 0s - loss: 0.1115 - accuracy: 0.9674 - val_loss: 0.1308 - val_accuracy: 0.9532 - 76ms/epoch - 2ms/step
Epoch 17/30
38/38 - 0s - loss: 0.1006 - accuracy: 0.9682 - val_loss: 0.1300 - val_accuracy: 0.9465 - 73ms/epoch - 2ms/step
```

Fig. 21. Results after 30th epoch

4) Checking how our model predicts unknown labels with the results shown below. The results are: loss = 0.11 and accuracy = 95 percent.

```
In [25]:
model.evaluate(testing_padded, test_labels)

10/10 [==============================] - 0s 1ms/step - loss: 0.1151 - accuracy: 0.9532

Out[25]:
[0.11506739258766174, 0.953177273273468]
```

Fig. 22. Results when checking unknown labels

5) Lastly, we visualize the results by plotting loss and accuracy vs number of epochs.

```
In [45]:
metrics = pd.DataFrame(history.history)
# Rename column
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])
plot_graphs1('Training_Loss', 'Validation_Loss', 'loss')
```

Fig. 23. Plotting the results

6) Below plot shows that accuracy is increasing with increase of epochs as well as model performs better in training set than validation set.

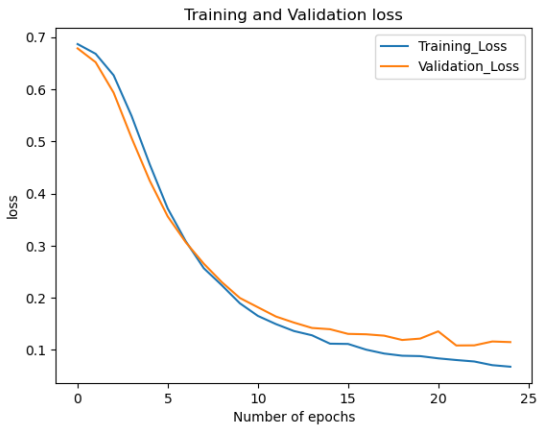

Fig. 24. Plotting the results

Lastly, we compare the results with Long-Short-Term Memory (LSTM).

- **Long-Short-Term Memory (LSTM):**

1) First, we tune out hyperparameters and code the architecture of our model.

```
In [20]:
#LSTM hyperparameters
n_lstm = 20 # the number of nodes in the hidden layers within the LSTM cell
drop_lstm =0.2 #dropout that prevents overfitting

#LSTM Spam detection architecture
model1 = Sequential()
model1.add(Embedding(vocab_size, embeding_dim, input_length=max_len))
model1.add(LSTM(n_lstm, dropout=drop_lstm, return_sequences=True))
model1.add(LSTM(n_lstm, dropout=drop_lstm, return_sequences=True))
model1.add(Dense(1, activation='sigmoid'))
```

Fig. 25. LSTM Model Architecture

2) Next, we will compile our model.

```
In [22]:
model1.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics=['accuracy'])
```

Fig. 26. Compiled model

3) After the model has been compiled, we train our model. The results we got after 30th epoch are: training loss: 0.12, training accuracy: 96 percent , validation loss: 0.3, validation accuracy: 92.2 percent.

```
In [26]:
num_epochs = 30
early_stop = EarlyStopping(monitor='val_loss', patience=2)
history = model1.fit(training_padded, train_labels, epochs=num_epochs, validation_data=(testing_padded, test_labels),
                     callbacks =[early_stop], verbose=2)
```

Fig. 27. Training the model with results

4) Lastly, we visualize the results by plotting training vs validation loss.

```
In [20]:
metrics = pd.DataFrame(history.history)
# Rename column
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('LSTM Model: Training and Validation ' + string)
    plt.xlabel ('Number of epochs')
    plt.ylabel(string)
    plt.legend([var1, var2])
plot_graphs1('Training_Loss', 'Validation_Loss', 'loss')
```

Fig. 28. Visualizing the results

5) Below plot shows that the results we got are somewhat positive. We can see that somewhere between the training, the loss is increasing which led to lesser accuracy than the Dense Sequential Model.

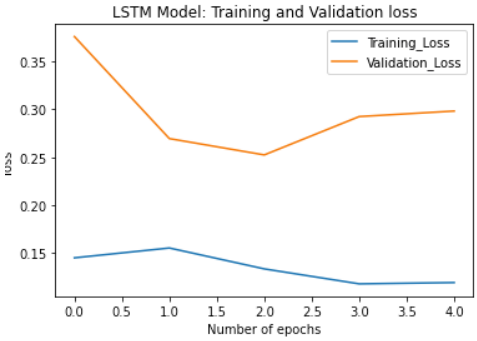

Fig. 29. Training vs validation loss plot

VII. Conclusion

We conclude that among the two of the machine learning models, Dense Sequential Model outperformed LSTM. So, we select Dense Sequential Model as our final model for filtering out spam emails based on loss, accuracy and plots. We check this dense model if the message is spam or ham from our original message below.

```
In [68]:
predict_msg = ["Go until jurong point, crazy.. Available only in bugis n great world la e buffetwat",
               "Ok lar... Joking wif u oni...",
               "Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question"]
# Defining prediction function
def predict_spam(predict_msg):
    new_seq = tokenizer.texts_to_sequences(predict_msg)
    padded = pad_sequences(new_seq, maxlen =max_len,
                           padding = padding_type,
                           truncating=trunc_type)
    return (model.predict(padded))
predict_spam(predict_msg)
```

Fig. 30. Testing the text with dense model

The below results shows that there is 99 percent chance for our third message to be spam which is correct.

```
1/1 [==============================] - 0s 14ms/step

Out[68]: array([[0.06104595],
                [0.0125065 ],
                [0.9959111 ]], dtype=float32)
```

Fig. 31. Prediction Results

This dense model can also be implemented in the future by using different algorithms and also adding more features to the existing system. Lastly, we will be also comparing the above ML models with Bi-LSTM networks in the near future to check which model have higher chances for predicting spam email messages.

References

[1] Aakash Atul Alurkar, Sourabh Bharat Ranade, Shreeya Vijay Joshi, Siddhesh Sanjay Ranade, Piyush A. Sonewa, Parikshit N. Mahalle, Arvind V. Deshpande "A Proposed Data Science Approach for Email Spam Classification using Machine Learning Techniques", 2017.

[2] "2022 State of the Phish," PFP UK TR State of the Phish-2022, Jan 2022. [Online]. Available: https://www.proofpoint.com/sites/default/files/threat-reports/pfpt-uk-tr-state-of-the-phish-2022.pdf.

[3] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2015, pp. 4580–4584.

[4] bibitemb3 T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2015, pp. 4580–4584

[5] Rathod, Sunil B., and Tareek M. Pattewar. "Content based spam detection in email using Bayesian classifier." International Conference on. IEEE, 2015.