SE 4050 – Deep Learning

4th Year second semester

Assignment

M.R.A.Fadhil

IT20784720

B.Sc. (Hons) in Information Technology Specializing in Software Engineering

Submitted to

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

Table of Content

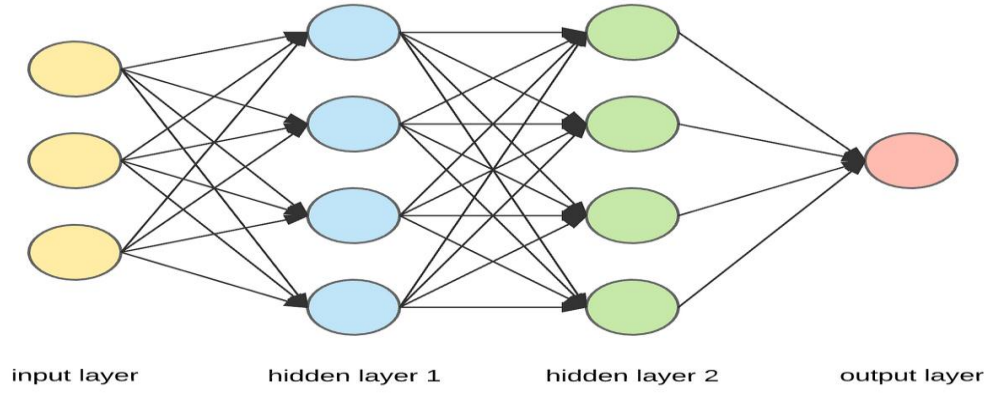
1. Problem introduction

Short Message Service (SMS) has become one of the most prevalent forms of personal and business communication. However, with its widespread use, SMS has also become a fertile ground for unwanted and often malicious messages, commonly use known as spam messages, spam messages can be widely used by cyber attackers to get particular user information by sending a spam message that will include some malicious link, once user unknowingly click the link cyber attackers have the ability to get the access of some user information. Sometimes some span messages comes with a congrats message and it says to contact a particular phone number and its calls as spam calls, so it's important that there is a way to identify these malicious attack through sms messages, so by collecting a various amount of data of sample spam messages and also ham messages, ham messages are messages are normal messages that users receives day to day life. By collecting ham messages as well as spam messages and by feeding into a model, so model can identify the format of 'spam' and 'ham' messages.so it can help to reduce the scams that occurs through spam messages in sms.

2. Algorithm Introduction

This is a classification problem, so when selecting an algorithm, our initial consideration revolves around choosing a model suitable for handling classification tasks. In this particular instance, we are working with an SMS 'spam' and 'ham' classification dataset aimed at distinguishing between spam and legitimate messages. Additionally, it's essential to note that this problem falls under binary classification, meaning it only predicts whether a message is spam or ham. Consequently, we explored various deep learning algorithms to construct models for this task, including Convolutional Neural Networks (CNNs), Long Short-Term Memory networks (LSTMs), Bidirectional LSTMs (BiLSTMs), and Artificial Neural Networks (ANNs). It's worth mentioning that LSTMs and BiLSTMs are part of the Recurrent Neural Network (RNN) family, which can be particularly useful for sequential data analysis.

2.1 ANN (Artificial Neural network)



Artificial Neural Networks are inspired by the structure and functioning of the human brain. They consist of interconnected nodes that are organized into layers, including an input layer, one or more hidden layers, and an output layer.

ANNs can be highly effective for text classification tasks due to their ability to capture complex patterns and relationships in textual data.

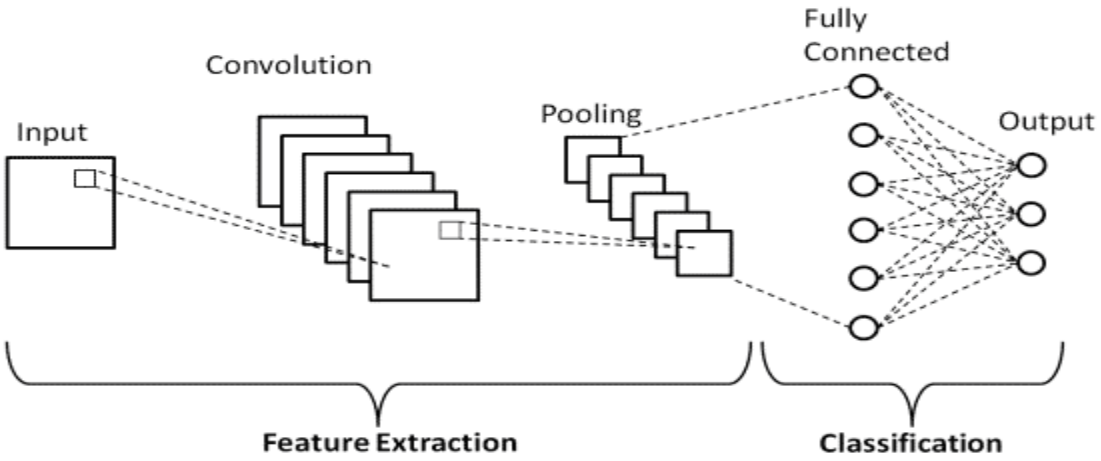
ANN was developed as follows, first the text are included to preprocessing steps and tokenization was applied to training dataset, then when developing the model, the model contains an input layer with 96 input neuros and identifies as input shape= (96,), then there are three hidden layers which each with 64, 32 and 16 units and all three layers uses ReLU (Rectified Linear Unit) activation function. It helps the network learn complex patterns in the data. And the output layer consists of a 1 unit with a sigmoid activation function. This architecture is typical for binary classification tasks. The sigmoid activation function maps the network's output to a probability value between 0 and 1, making it suitable for binary classification to predict one of two classes.

And also model compiles with some additional configurations by using ‘binary cross entropy’ as loss function which is common for binary classification problems, then the optimizer was ‘Adam’ optimizer with a learning rate of 0.01. Adam is an optimization algorithm that helps adjust the model's weights during training to minimize the loss function. And the model was evaluated using the 'accuracy' metric.

```
#model summary
ann.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                6208

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 16)                528

 dense_3 (Dense)             (None, 1)                 17

=================================================================
Total params: 8833 (34.50 KB)
Trainable params: 8833 (34.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

2.2 CNN (Convolutional Neural Network)



Convolutional Neural Networks (CNNs) are a type of deep learning architecture commonly used for text classification tasks, leveraging their ability to automatically extract meaningful features from text data through convolutional layers to improve classification accuracy.

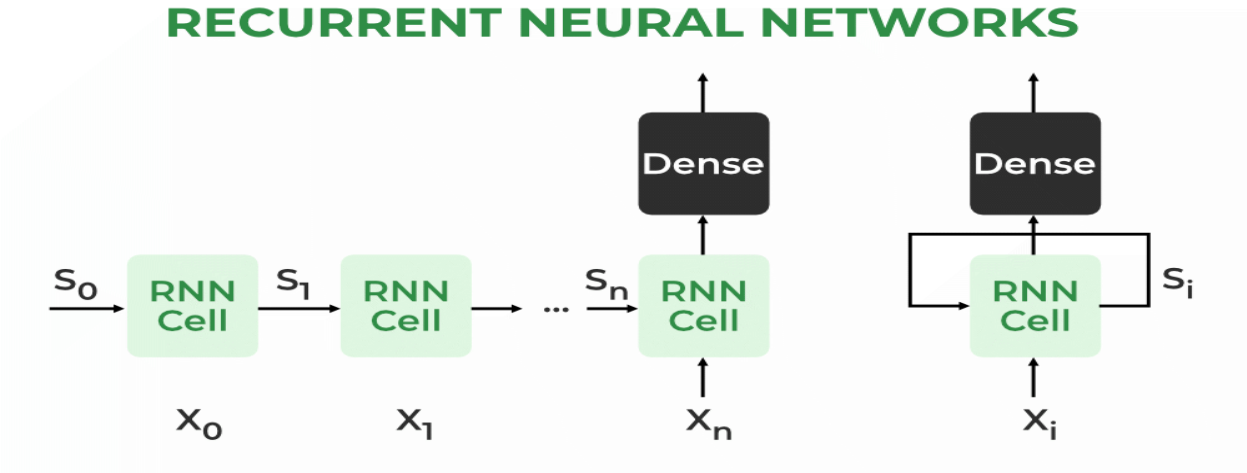
CNN was developed as follows, The Input layer defines the input shape of the model. It expects sequences of a certain length (96,), then an embedding layer was used to convert categorical data into continuous vector representation, then two Conv1D layer with 128 filters, a kernel size of 3, and uses the ReLU activation function was used. It is used for feature extraction from the input and after each Conv1D layer, a MaxPooling1D layer with a pool size of 3 was added. This reduces the spatial dimensions of the output, which can help the model focus on the most important features. And then GlobalMaxPooling1D was added it extracts the most important features from the entire sequence. Then a Dense layer with 128 units and ReLU activation added after the global max-pooling layer. And the output layer consists of a 1 unit with a sigmoid activation function. This architecture is typical for binary classification tasks. The sigmoid activation function maps the network's output to a probability value between 0 and 1, making it suitable for binary classification to predict one of two classes.

And also model compiles with some additional configurations by using ‘binary cross entropy’ as loss function which is common for binary classification problems, then the optimizer was RMSprop optimizer was used. And the model was evaluated using the 'accuracy' metric.

```
#model summary
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 96)]              0

 embedding (Embedding)       (None, 96, 96)            794496

 conv1d (Conv1D)             (None, 94, 128)           36992

 max_pooling1d (MaxPooling1  (None, 31, 128)           0
 D)

 conv1d_1 (Conv1D)           (None, 29, 128)           49280

 max_pooling1d_1 (MaxPoolin  (None, 9, 128)            0
 g1D)

 conv1d_2 (Conv1D)           (None, 7, 128)            49280

 global_max_pooling1d (Glob  (None, 128)               0
 alMaxPooling1D)

 dense (Dense)               (None, 128)               16512

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 946689 (3.61 MB)
Trainable params: 152193 (594.50 KB)
Non-trainable params: 794496 (3.03 MB)
_________________________________________________________________
```

2.3 RNN (Recurrent Neural Network)



Recurrent Neural Networks (RNNs) are particularly well-suited for tasks such as natural language processing, speech recognition, and time series analysis. They excel at modeling dependencies and relationships within sequential data. However, traditional RNNs have

limitations in capturing long-term dependencies due to the vanishing gradient problem. To overcome this challenge, more advanced RNN variants like Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Bidirectional Long Short-Term Memory (BiLSTM) have been developed. These variants are proven to be more effective at learning and retaining information over longer sequences.

2.3.1 LSTM and BiLSTM

Long Short-Term Memory (LSTM) and Bidirectional Long Short-Term Memory (BiLSTM) are specialized recurrent neural network (RNN) architectures designed to efficiently handle sequential data. These models are particularly useful in text classification tasks due to their ability to capture dependencies and relationships within text data over long sequences.

LSTMs have the ability to effectively model long-range dependencies in text data. BiLSTMs, on the other hand, take the concept of LSTMs a step further by processing the input sequence in both forward and backward directions simultaneously. In text classification, the need for LSTM and BiLSTM models arises from the complexity and variability of natural language. Textual data often contains nuanced relationships, long-range dependencies, and context-specific meanings that traditional models struggle to capture effectively.

Bilstm and lstm both developed as follows, The Input layer defines the input shape of the model. It expects sequences of a certain length (96,), then an embedding layer was used to convert categorical data into continuous vector representation, then for the Bilstm an Bidirectional LSTM layer with 32 hidden units and sets return sequences=True to ensure that it returns the sequences instead of just the final output. The Bidirectional layer processes the input sequence in both forward and backward directions, which can capture bidirectional dependencies in the data. This layer takes the output from the embedding layer as input. then for the lstm an Bidirectional LSTM layer with 32 hidden units and sets return sequences=True to ensure that it returns the sequences instead of just the final output, which means it will produce output sequences of the same length as the input sequences. Then uses an global max-pooling over the sequence data. It extracts the maximum value from each feature across the entire sequence, reducing the sequence length to a fixed size. And the output layer consists of a 1 unit with a sigmoid activation function. This architecture is typical for binary classification tasks. The sigmoid activation function maps the network's output to a probability value between 0 and 1, making it suitable for binary classification to predict one of two classes.

And also model compiles with some additional configurations by using ‘binary cross entropy’ as loss function which is common for binary classification problems, then the optimizer was ‘Adam’ optimizer with a learning rate of 0.01. Adam is an optimization algorithm that helps adjust the model's weights during training to minimize the loss function. And the model was evaluated using the 'accuracy' metric.

```
#model summary
model.summary()
```

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 96)]              0

 embedding_2 (Embedding)     (None, 96, 96)            794496

 bidirectional_3 (Bidirecti  (None, 96, 64)            33024
 onal)

 global_max_pooling1d_3 (Gl  (None, 64)                0
 obalMaxPooling1D)

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 827585 (3.16 MB)
Trainable params: 33089 (129.25 KB)
Non-trainable params: 794496 (3.03 MB)
_________________________________________________________________
```

```
#model summary
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 96)]              0

 embedding (Embedding)       (None, 96, 96)            794496

 lstm (LSTM)                 (None, 96, 32)            16512

 global_max_pooling1d (Glob  (None, 32)                0
 alMaxPooling1D)

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 811041 (3.09 MB)
Trainable params: 16545 (64.63 KB)
Non-trainable params: 794496 (3.03 MB)
_________________________________________________________________
```

3. About Dataset

The SMS spam collection data set is a kaggle dataset, the spam messages are collected from Grumble text website and the ham messages are collected through a collection of dataset collected for research at the department of computer science at the national university of Singapore.

The dataset mainly contains two main columns one classify whether the message is ‘spam’ or ‘ham’ and other column is contain the spam or ham message.

Resource URL:

https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset

Download url:

https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset/download?datasetVersionNumber=1

The dataset contains collection of 5574 datasets, with classified as ham or spam.

4. Data analysis and visualization

At the beginning we need to load the dataset as csv and display the dataset, for that we displayed top 20 head datasets. Then if there are unwanted columns available we can remove those columns.

```
# Import the dataset as csv using pandas and remove unwanted columns
dataset= pd.read_csv("./data/sms_spam_ham.csv",encoding='latin-1')
dataset.drop(columns=[
    'Unnamed: 2',
    'Unnamed: 3',
    'Unnamed: 4',
] , inplace=True)
dataset.head(5)
```

	label	review
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...

There are mainly 2 columns , first one is the Label column which classified as spam and ham , if the message spam the column value will be span and if the message is ham the column value will be ham. The other column will display the spam or ham message.

```
dataset.describe()
```

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

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   label   5572 non-null   object
 1   review  5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

When grouping the dataset according to the 'Label', it shows there are 4825 ham messages and 747 spam messages, and also we can observe that there are 4516 unique ham messages and 653 unique spam messages that need to consider when doing the preprocessing to identify the duplicate values and remove from the dataset.

```
dataset.groupby('label').describe()
```

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

To identify the message lengths of each message, we also added a new column containing the length of each message, and we can observe that the top message contains length of 910.

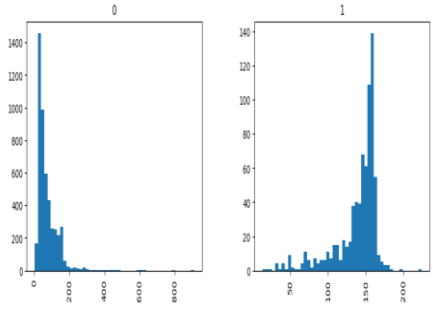
```
dataset['length'] = dataset['review'].apply(len) # add a length column for get the length of each review texts
```

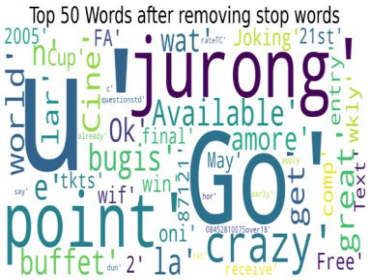
```
dataset.head()
```

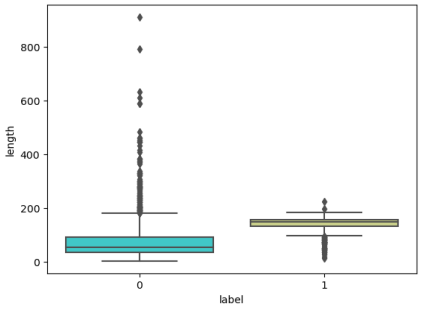
	label	review	length
0	ham	Go until jurong point, crazy.. Available only ...	111
1	ham	Ok lar... Joking wif u oni...	29
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	ham	U dun say so early hor... U c already then say...	49
4	ham	Nah I don't think he goes to usf, he lives aro...	61

We use the visualization according to the message length base off the label type (spam or ham), for the visualization we use count plot of number of occurrence for each type of label, boxplot of text length for each label type.

One of the main goal is to, tokenize each message and then apply preprocessing to remove unwanted punctuations and stop words and then apply lemmatization. and then apply all the changes to training data , so those steps will be handled in the preprocessing steps , once all stop words removed we can display the common words that are used in the datasets.







5. Data preprocessing

The data preprocessing is really important to get a great accuracy, in nlp there are many steps need to follow in order to complete the preprocessing, when it comes to our sms spam and ham dataset, the following preprocessing methods are performed,

1. Identify the null values from the dataset and remove from the dataset
2. Identify the duplicate values or entries from the dataset and remove from the dataset
3. Apply label encoder on the target column to convert the values to 0 and 1
4. Break in to tokens each row of message and identify if there ant word of punctuations.
5. Check each words single letter to identify if there are any punctuations was attached with the word.
6. Stop words are frequently occurred words since we can remove stop words because it doesn't contribute in our problem
7. After removing the stop words again join each words as string
8. Apply lemmatization , lemmatization convert a word to its root form
9. Tokenizing to convert the words to integer

```
# Checking for NaN values
dataset.isnull().sum()
```

```
label     0
review    0
length    0
dtype: int64
```

```
# Checking for empty whitespace strings and assign for an array
empty_list = []

for i,label,review,length in dataset.itertuples():  # iterate over the DataFrame
    if type(review)==str:
        if review.isspace():
            empty_list.append(i)

len(empty_list)
```

```
0
```

```
#droping NaN values
dataset.dropna(inplace=True)
```

```
#drop empty space values in review
dataset.drop(empty_list,inplace=True)
```

```
# Checking for duplicate values
dataset.duplicated().sum()
```

```
403
```

```
#remove duplicate entries
dataset.drop_duplicates(keep='first',inplace=True)
```

```
#Create a function to identify the punctuations in each words
def separate_puncuation(doc_text):
    return [token for token in nlp_load(doc_text) if token.text not in '\n\n \n\n\n!"-#$%&()--.*+,-/:;<=>?@[\\]^_`{|}~\t\n ']
```

```
#check wheter there are any single puncuations and convert the row to string
def remove_punctuation_and_convert_to_string(token):
    return ''.join(char for char in str(token) if char not in string.punctuation)
```

```
#remove stopwords from each row
def remove_stop(token):
    return [word for word in token.split() if word.lower() not in stopwords.words('english')]
```

```
lemmatize = WordNetLemmatizer()
# lemmatize word
def lemmatize_word(doc_text):
    lemmatized_word = [lemmatize.lemmatize(word, pos ='v') for word in doc_text]
    return lemmatized_word
```

```
#convert each tokens to string after removing stopwords
def join_list_to_string(lst):
    return ' '.join(lst)
```

```
dataset['review'] = dataset['review'].apply(separate_puncuation)
```

```
dataset['review'] = dataset['review'].apply(remove_punctuation_and_convert_to_string)
```

```
dataset['review'] = dataset['review'].apply(remove_stop)
```

```
dataset['review'] = dataset['review'].apply(lemmatize_word)
```

```
dataset['review'] = dataset['review'].apply(lambda x: join_list_to_string(x))
```

When cleaning the data removing the stop words and then applying lemmatization was are important steps need to be done in order to clean the text in the dataset.

We use Tokenizer method to break the string into word and then convert each word in to integer value that can be used for word embedding. Once each word is convert to integer by using ‘tokenizer.word_index’ method we can map each integer to the appropriate word.

```
#convert the sentence to integer
tokenizer = Tokenizer(num_words=vocabulary_size)
tokenizer.fit_on_texts(train)
sequences = tokenizer.texts_to_sequences(train)
```

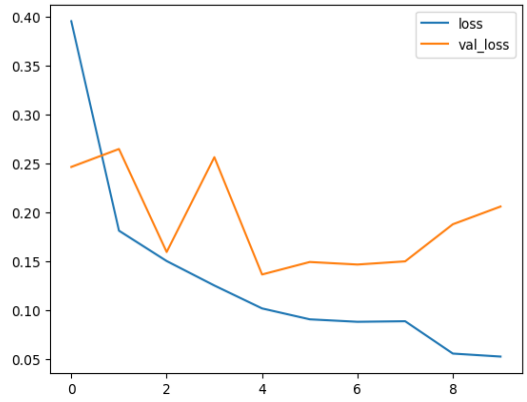
```
#word integer mapping
word_index = tokenizer.word_index
```

6. Test result and analysis

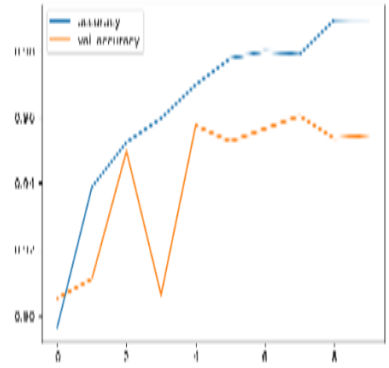
6.1 CNN (Convolutional Neural Network)

Loss

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



Accuracy




```
#AUC Score
roc =roc_auc_score(target,prediction)
print("ROC_AUC_SCORE:", roc)
```

```
ROC_AUC_SCORE: 0.9652684618379164
```

```
#Root Mean Squared Error
from math import sqrt
rmse = sqrt(mean_squared_error(target, prediction_summary))
print("RMSE:", rmse)
```

```
RMSE: 0.12973476223291455
```

```
#Mean Squared Error
mse = mean_squared_error(target,prediction_summary)
print("MSE:", mse)
```

```
MSE: 0.016831108531630876
```

```
# Create Precision and Recall metrics
precision = Precision()
recall = Recall()
accuracy = Accuracy()

precision.update_state(target, prediction_summary)
recall.update_state(target, prediction_summary)
accuracy .update_state(target, prediction_summary)


precision_result = precision.result().numpy()
recall_result = recall.result().numpy()
accuracy_result = accuracy.result().numpy()
```

```
# display precision and recall values
print(f"Precision: {precision_result}")
print(f"Recall: {recall_result}")
print(f"Accuracy: {accuracy_result}")
```

```
Precision: 0.9564515948295593
Recall: 0.9081164002418518
Accuracy: 0.9831689001297
```

```
print(classification_report(target, prediction_summary))
```

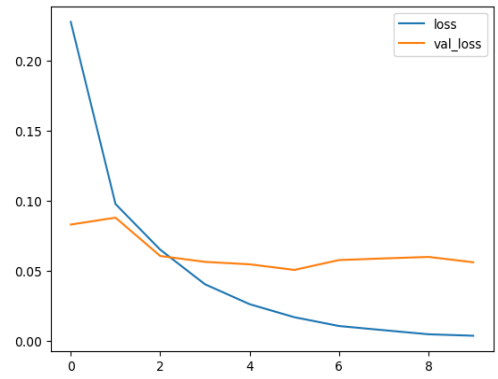
	precision	recall	f1-score	support
0	0.99	0.99	0.99	4516
1	0.96	0.91	0.93	653
accuracy			0.98	5169
macro avg	0.97	0.95	0.96	5169
weighted avg	0.98	0.98	0.98	5169

The CNN model having an ROC AUC (Receiver Operating Characteristic Area Under the Curve) score of 0.9652, it demonstrates a high ability to distinguish between positive and negative classes, indicating strong discriminatory power. The low RMSE (Root Mean Square Error) value of 0.1297 suggests that the model's predictions closely align with the actual values in a regression context, implying good predictive accuracy. Additionally, the precision score of 0.95645 indicates that the model has a high proportion of true positive predictions among its positive class predictions, minimizing false positives. The recall of 0.9081 showcases the model's ability to correctly identify a substantial portion of actual positive cases. Lastly, the high accuracy of 0.9831 underscores the model's overall correctness in classifying instances, making it a robust and reliable tool for the given task.

6.2 BiLstm (Bidirectional Long Short Term Memory)

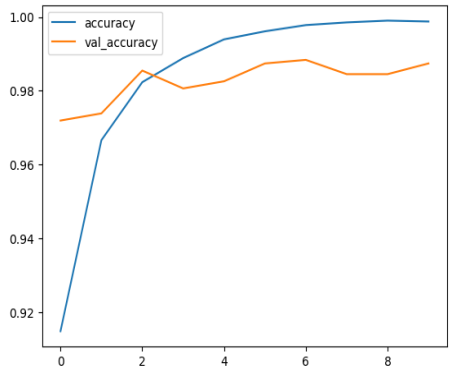
Loss

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



Accuracy

```
plt.plot(model_history.history['accuracy'],label='accuracy')
plt.plot(model_history.history['val_accuracy'],label='val_accuracy')
plt.legend()
plt.show()
```




```
#AUC Score
roc =roc_auc_score(target,prediction)
print("ROC_AUC_SCORE:", roc)
```

```
ROC_AUC_SCORE: 0.9979436734727095
```

```
#Root Mean Squared Error
from sklearn.metrics import mean_squared_error
from math import sqrt
rmse = sqrt(mean_squared_error(target, prediction_summary))
print("RMSE:", rmse)
```

```
RMSE: 0.05014970816330224
```

```
#Mean Squared Error
mse = mean_squared_error(target, prediction_summary)
print("MSE:", mse)
```

```
MSE: 0.0025149932288643837
```

```
# Create Precision and Recall metrics
precision = Precision()
recall = Recall()
accuracy = Accuracy()

precision.update_state(target, prediction_summary)
recall.update_state(target, prediction_summary)
accuracy .update_state(target, prediction_summary)


precision_result = precision.result().numpy()
recall_result = recall.result().numpy()
accuracy_result = accuracy.result().numpy()
```

```
# display precision and recall values
print(f"Precision: {precision_result}")
print(f"Recall: {recall_result}")
print(f"Accuracy: {accuracy_result}")
```

```
Precision: 0.9923076629638672
Recall: 0.9877488613128662
Accuracy: 0.9974849820137024
```

```
print(classification_report(target, prediction_summary))
```

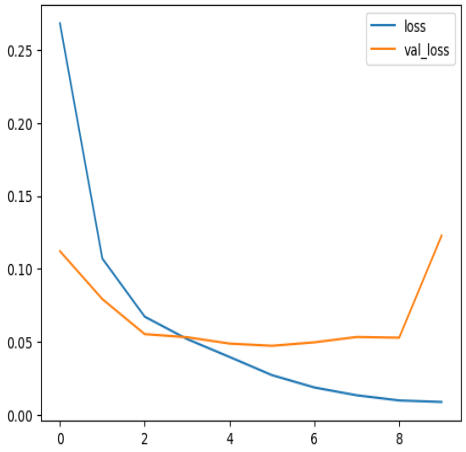
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4516
           1       0.99      0.99      0.99       653

    accuracy                           1.00      5169
   macro avg       1.00      0.99      0.99      5169
weighted avg       1.00      1.00      1.00      5169
```

The Bilstm model having an ROC AUC (Receiver Operating Characteristic Area Under the Curve) score of 0.997, it demonstrates outstanding discriminative power in distinguishing between positive and negative cases, suggesting that it's highly effective at classification tasks. The low RMSE (Root Mean Square Error) of 0.050 indicates that the model's predictions are very close to the actual values, highlighting its accuracy in regression tasks. Moreover, the model achieves a precision of 0.992307 and recall of 0.9877488, demonstrating its ability to not only make accurate positive predictions but also effectively identify a large proportion of actual positives. This combination of high precision and recall suggests a well-balanced trade-off between minimizing false positives and false negatives. Finally, the accuracy of 0.997 indicates that the model correctly classifies the majority of the instances in the dataset, further underscoring its overall excellence in predictive performance. Overall, this model showcases exceptional performance across various metrics.

6.3 LSTM (Long Short Term Memory)

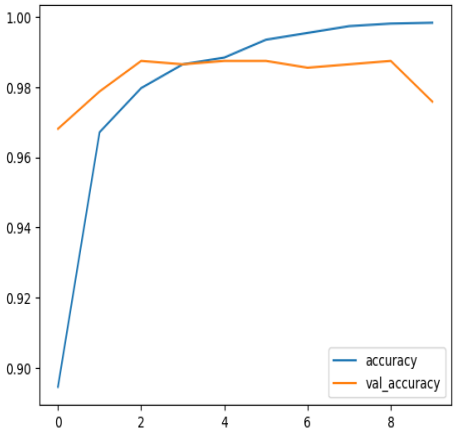
Loss

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



Accuracy

```
plt.plot(model_history.history['accuracy'],label='accuracy')
plt.plot(model_history.history['val_accuracy'],label='val_accuracy')
plt.legend()
plt.show()
```




```
#AUC Score
roc =roc_auc_score(target,prediction)
print("ROC_AUC_SCORE:", roc)
```

```
ROC_AUC_SCORE: 0.9968266649666254
```

```
#Root Mean Squared Error
from sklearn.metrics import mean_squared_error
from math import sqrt
rmse = sqrt(mean_squared_error(target, prediction_summary))
print("RMSE:", rmse)
```

```
RMSE: 0.11802200331953609
```

```
#Mean Squared Error
mse = mean_squared_error(target, prediction_summary)
print("MSE:", mse)
```

```
MSE: 0.013929193267556587
```

```
# Create Precision and Recall metrics
precision = Precision()
recall = Recall()
accuracy = Accuracy()

precision.update_state(target, prediction_summary)
recall.update_state(target, prediction_summary)
accuracy .update_state(target, prediction_summary)


precision_result = precision.result().numpy()
recall_result = recall.result().numpy()
accuracy_result = accuracy.result().numpy()
```

```
# display precision and recall values
print(f"Precision: {precision_result}")
print(f"Recall: {recall_result}")
print(f"Accuracy: {accuracy_result}")
```

```
Precision: 0.9144079685211182
Recall: 0.9816232919692993
Accuracy: 0.9860708117485046
```

```
print(classification_report(target, prediction_summary))
```

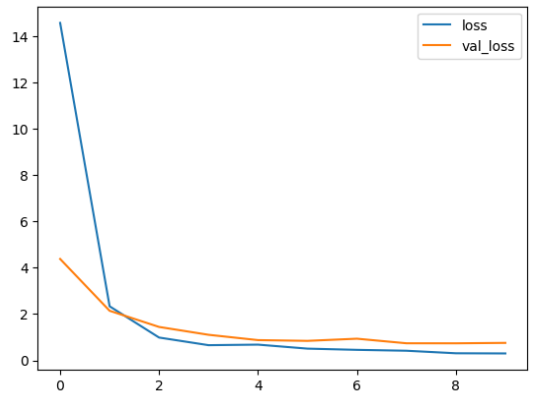
```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      4516
           1       0.91      0.98      0.95       653

    accuracy                           0.99      5169
   macro avg       0.96      0.98      0.97      5169
weighted avg       0.99      0.99      0.99      5169
```

The Lstm model having an ROC AUC score of 0.996 suggests that the model has a high ability to distinguish between positive and negative classes, making it well-suited for binary classification tasks. The low RMSE (Root Mean Square Error) value of 0.1180 indicates that the model's predictions are very close to the actual values, which is particularly relevant for regression tasks. Furthermore, the precision of 0.914407 and recall of 0.981623 signify that the model strikes a strong balance between minimizing false positives and false negatives, essential for scenarios where precision and recall are critical. Lastly, an accuracy of 0.9860 demonstrates that the model correctly classifies a large majority of the samples, further highlighting its overall robustness and competence in handling the given task.

6.4 ANN (Artificial Neural Network)

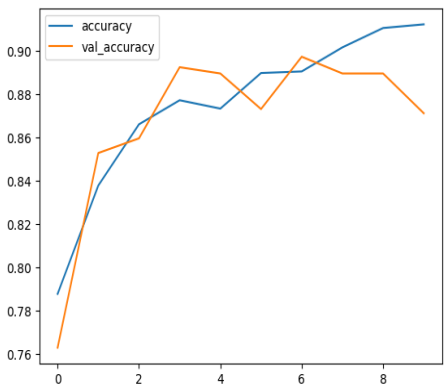
Loss

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



Accuracy

```
plt.plot(model_history.history['accuracy'],label='accuracy')
plt.plot(model_history.history['val_accuracy'],label='val_accuracy')
plt.legend()
plt.show()
```



```
#AUC Score
roc =roc_auc_score(target,prediction)
print("ROC_AUC_SCORE:", roc)
```

```
ROC_AUC_SCORE: 0.8804158296450124
```

```
#Root Mean Squared Error
from sklearn.metrics import mean_squared_error
from math import sqrt
rmse = sqrt(mean_squared_error(target, prediction_summary))
print("RMSE:", rmse)
```

```
RMSE: 0.29928657028188393
```

```
#Mean Squared Error
mse = mean_squared_error(target, prediction_summary)
print("MSE:", mse)
```

```
MSE: 0.08957245115109305
```

```
# Create Precision and Recall metrics
precision = Precision()
recall = Recall()
accuracy = Accuracy()

precision.update_state(target, prediction_summary)
recall.update_state(target, prediction_summary)
accuracy .update_state(target, prediction_summary)


precision_result = precision.result().numpy()
recall_result = recall.result().numpy()
accuracy_result = accuracy.result().numpy()
```

```
# display precision and recall values
print(f"Precision: {precision_result}")
print(f"Recall: {recall_result}")
print(f"Accuracy: {accuracy_result}")
```

```
Precision: 0.627345860004425
Recall: 0.7166922092437744
Accuracy: 0.9104275703430176
```

```
print(classification_report(target, prediction_summary))
```

```
              precision    recall  f1-score   support

           0       0.96      0.94      0.95      4516
           1       0.63      0.72      0.67       653

    accuracy                           0.91      5169
   macro avg       0.79      0.83      0.81      5169
weighted avg       0.92      0.91      0.91      5169
```

The ANN model having an ROC AUC (Receiver Operating Characteristic Area Under the Curve) score of 0.8804, it demonstrates a strong ability to discriminate between positive and negative classes in binary classification tasks. A lower RMSE (Root Mean Square Error) value of 0.2992 suggests that the model is accurate in predicting continuous numerical values, indicating good regression performance. Additionally, a precision of 0.6273 indicates that the model has a relatively low rate of false positives, which is important in applications where minimizing Type I errors is critical. A recall of 0.7166 shows that the model captures a substantial portion of the true positive cases, and an accuracy of 0.9104 indicates that it is overall quite accurate in its predictions.

7. Conclusion

Based on the evaluation of four different models, The BiLSTM model stands out with an outstanding ROC AUC score of 0.997, indicating exceptional discriminative power in binary classification tasks. Its low RMSE of 0.050 demonstrates high accuracy in regression tasks, and it achieves an impressive precision of 0.992307 and recall of 0.9877488, striking a well-balanced trade-off between false positives and false negatives. The high accuracy of 0.997 further solidifies its overall excellence. In conclusion, the BiLSTM model appears to be the top-performing model in overall.