******COMSATS University Islamabad (Lahore** **Campus)**

**Assignment <2> FALL 2022**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Course Title: | Computational Intelligence | | | Course Code: | | CSC | Credit Hours: | 3(2,1) |
| Course Instructor: | Dr. Atifa Athar | | | Programme Name: | | BS Computer Science | | |
| Semester: | 7th | Batch: | FA19 | Section: | **B** | Date: | 14-1-2022 | |
| **Due Date:** | **04-11-2022** | | | **Maximum Marks:** | | | **10** | |
| **Student Name:** | **Muhammad Awais**  **Abdullah**  **Muhammad Huzaifa Tariq** | | | **Registration No.** | | | **FA19-BCS-105**  **FA19-BCS-114**  **FA19-BCS-047** | |
| **Important Instructions / Guidelines:**   * **No late submissions will be accepted.** * **All assignments are required to be submitted using attached template only.** | | | | | | | | |

**Question No 1. Marks: 10**

***CLO: <3>; Bloom Taxonomy Level: <Applying>***

Select a textual dataset of your choice (for example IMDB movies review data). Design and implement RNN and LSTM Neural Network, Bert model (along with its variants) in Python using your selected dataset.

Submit a complete report including an introduction to the dataset. implementation, Results (accuracy tables and confusion matrix), and comparisons. Along with the Report, you are required to submit your code as well.

# Problem Statement:

Design, implement, evaluate and compare different NLP techniques for tweet classification such as RNN & LSTM, CNN, CNN+LSTM, and BERT in python on the COVID-19 vaccines dataset.

# Introduction to Dataset:

The dataset we have selected for this assignment is “Tweets Dataset on COVID-19 Vaccines”. It was collected by social media posts on COVID-19 vaccines using Twitter’s streaming API from December 9, 2020 - Feb 24, 2021, with keywords related to the vaccines ("Vaccine", "Pfizer", "BioNTech", "Moderna", "Janssen", "AstraZeneca", "Sinopharm"). This dataset includes CSV files that contain IDs, text, labels, and sixty other columns related to the COVID-19 tweets. On social media, content propagates through the network when accounts engage with posts by re-sharing (retweeting), replying to tweets, and quoting (quote tweets are retweets without a comment). There are 376K entries in the raw dataset with most of the tweets as retweeted, so there are unique 14.6k tweets, roughly 10,377 labeled as reliable and 4,267 labeled as unreliable/conspiracy.

## Data Info:

RangeIndex: 14644 entries, 0 to 14643

Data columns (total 69 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 cascade\_sno 14644 non-null int64

1 cascade\_tid 14644 non-null int64

2 tweetid 14644 non-null int64

3 userid 14644 non-null int64

4 screen\_name 14644 non-null object

5 date 14644 non-null object

6 lang 14644 non-null object

7 location 10398 non-null object

8 place\_id 52 non-null object

9 place\_url 52 non-null object

10 place\_type 52 non-null object

11 place\_name 52 non-null object

12 place\_full\_name 52 non-null object

13 place\_country\_code 52 non-null object

14 place\_country 52 non-null object

15 place\_bounding\_box 52 non-null object

16 text 14644 non-null object

17 extended 14644 non-null object

18 coord 0 non-null float64

19 reply\_userid 388 non-null float64

20 reply\_screen 388 non-null object

21 reply\_statusid 388 non-null float64

22 tweet\_type 14644 non-null object

23 friends\_count 14644 non-null int64

24 listed\_count 14644 non-null int64

25 followers\_count 14644 non-null int64

26 favourites\_count 14644 non-null int64

27 statuses\_count 14644 non-null int64

28 verified 14644 non-null bool

29 hashtag 14644 non-null object

30 urls\_list 14644 non-null object

31 profile\_pic\_url 12200 non-null object

32 profile\_banner\_url 12200 non-null object

33 display\_name 14643 non-null object

34 date\_first\_tweet 14644 non-null object

35 account\_creation\_date 14644 non-null object

36 rt\_urls\_list 14644 non-null object

37 mentionid 14644 non-null object

38 mentionsn 14644 non-null object

39 rt\_screen 8821 non-null object

40 rt\_userid 8821 non-null float64

41 rt\_text 8889 non-null object

42 rt\_hashtag 14644 non-null object

43 rt\_qtd\_count 14644 non-null int64

44 rt\_rt\_count 14644 non-null int64

45 rt\_reply\_count 14644 non-null int64

46 rt\_fav\_count 14644 non-null int64

47 rt\_tweetid 8889 non-null float64

48 rt\_location 7181 non-null object

49 qtd\_screen 129 non-null object

50 qtd\_userid 129 non-null float64

51 qtd\_text 132 non-null object

52 qtd\_hashtag 14644 non-null object

53 qtd\_qtd\_count 14644 non-null int64

54 qtd\_rt\_count 14644 non-null int64

55 qtd\_reply\_count 14644 non-null int64

56 qtd\_fav\_count 14644 non-null int64

57 qtd\_tweetid 132 non-null float64

58 qtd\_urls\_list 14644 non-null object

59 qtd\_location 97 non-null object

60 sent\_vader 14644 non-null float64

61 token 14641 non-null object

62 media\_urls 14644 non-null object

63 rt\_media\_urls 14644 non-null object

64 q\_media\_urls 14644 non-null object

65 corrected\_tweet\_type 14644 non-null object

66 ns\_label 14644 non-null object

67 ns\_url 14644 non-null object

68 timestamp 14644 non-null float64

# Data Preprocessing:

1. Clean all the tweets by converting them into lowercase and remove all unnecessary characters (and some unnecessarily repeated words).
2. Encode the string labels as “1” for unreliable/conspiracy and “0” for reliable.
3. Split the data into training and testing sets.
4. Generate tokens of the tweets data.
5. Generate sequences and add padding to the max length.

# NLP techniques for Text Classification:

There are several natural language processing (NLP) techniques that can be used for text classification:

1. **Bag-of-Words:** This technique represents each text as a bag of its words, disregarding grammar and word order but keeping track of occurrences. This can be used as input for a classifier such as Naive Bayes.
2. **Term Frequency-Inverse Document Frequency (TF-IDF):** This technique assigns weight to each word in the text, with more weight given to words that are more informative and less common words. This can also be used as input for a classifier.
3. **Word Embeddings:** This technique represents each word in a text as a dense vector, which captures semantic and syntactic information about the word. These vectors can be used as input for a classifier such as a feed-forward neural network or a convolutional neural network.
4. **Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM):** These are types of neural networks that process sequential data, such as text. They can be trained in text classification tasks and can output a probability of the text belonging to each class.
5. **Transformers:** These are deep learning models that are particularly well suited for NLP tasks, such as text classification. They have been shown to be very effective in achieving state-of-the-art performance on a wide range of NLP tasks.

# LSTM

LSTM stands for Long Short-Term Memory. It is a type of Recurrent Neural Network (RNN) that can capture long-term dependencies in sequential data. Unlike traditional RNNs, which tend to forget past information as the network unfolds over time, LSTMs have a built-in memory cell that can retain information for prolonged periods. This makes them well-suited for tasks such as language modeling, machine translation, and speech recognition.

**Implementation:** Implementing an LSTM network typically involves the following steps:

1. **Prepare the data:** The first step is to prepare the data by tokenizing and indexing the text, and then dividing it into training and testing sets.
2. **Choose the LSTM architecture:** The next step is to choose the LSTM architecture, which includes deciding on the number of layers, the number of units in each layer, and whether to use a bidirectional or unidirectional LSTM.
3. **Define the model:** The LSTM model can be defined using a deep learning library such as TensorFlow, Keras, or PyTorch. The model consists of an LSTM layer, followed by one or more fully connected layers.
4. **Compile the model:** Once the model is defined, it needs to be compiled by specifying the loss function, optimizer, and metrics.
5. **Train the model:** The model is then trained on the training data using the fit() function. The model will learn to associate the input sequences with the corresponding output labels.
6. **Evaluate the model:** After the model is trained, it can be evaluated on the test data using the evaluate() function, which will return the loss and any other metrics that were specified during compilation.

It's important to note that the specific implementation details of an LSTM will depend on the deep learning framework and the specific use case.

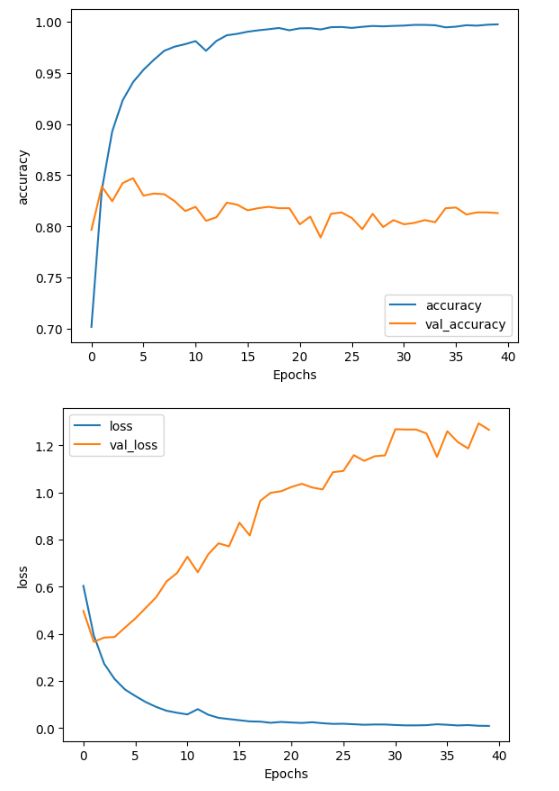
## LSTM Model Summary:

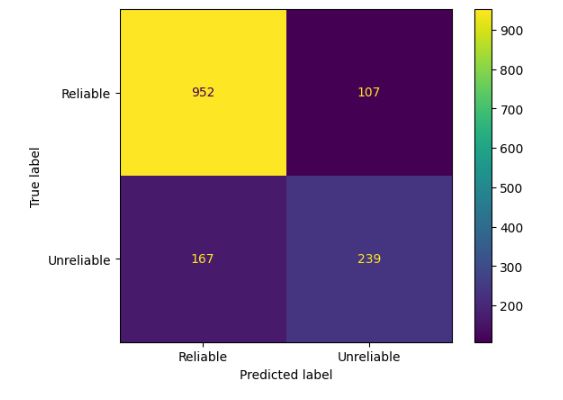
Table

Description automatically generated

Table

Description automatically generated with medium confidence





Graphical user interface, chart, scatter chart

Description automatically generated

# GRU

The Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) architecture that addresses some of the limitations of traditional RNNs, such as the vanishing gradient problem. The GRU architecture utilizes gating mechanisms to control the flow of information through the network, allowing for better preservation of long-term dependencies and improved performance on tasks such as language modeling and speech recognition. The GRU was introduced by Cho et al. in 2014 and has since become a popular choice in RNN architectures.

**Implementation:** Implementing a GRU in code involves several steps:

1. Initializing the GRU model. This typically includes defining the number of hidden units, the input dimension, and the number of layers in the model.
2. Defining the input and output layers. The input layer is typically an embedding layer, which converts the input words or sequences into a dense vector representation. The output layer is typically a linear layer that maps the hidden state of the GRU to the output of the model.
3. Defining the GRU layers. This typically involves creating instances of the GRU class, specifying the number of hidden units, and connecting the input and output layers to the GRU layers.
4. Training the model. This typically involves defining a loss function and an optimizer and then training the model on a dataset using a training loop.
5. Evaluating the model. This typically involves evaluating the model on a validation or test dataset and measuring its performance using metrics such as accuracy or perplexity.

## GRU Model Summary:

Table

Description automatically generated

Chart, histogram

Description automatically generatedChart, treemap chart

Description automatically generated

Table

Description automatically generated

# CNN

A Convolutional Neural Network (CNN) is a type of deep learning neural network that is designed to process and analyze data that has a grid-like structure, such as images, videos, and audio signals. CNNs are composed of multiple layers, including convolutional layers, pooling layers, and fully connected layers. The convolutional layers are responsible for analyzing the input data and extracting features, such as edges, textures, and patterns. The pooling layers are responsible for reducing the spatial dimensions of the data while maintaining the important features. The fully connected layers are responsible for classifying the data based on the features extracted by the convolutional and pooling layers.

CNNs are particularly useful for image classification, object detection, and image segmentation tasks, as they are able to learn and extract features from images that are not easily represented by traditional image processing techniques. They have been widely adopted in many computer vision applications such as self-driving cars, image search engines, and facial recognition. CNNs can also be used for text classification tasks, in which the goal is to classify a piece of text into one or more predefined categories. Text classification is a widely used technique in natural language processing (NLP) and is used in tasks such as sentiment analysis, topic classification, and spam detection. In text classification, a CNN is typically applied to the embedding of words in a sentence or a document, rather than to the raw input text. The embedding is a dense vector representation of each word, which captures the semantic meaning of the word. The embedding is typically obtained by training a word embedding model, such as word2vec or GloVe, on a large corpus of text.

The CNN architecture for text classification is similar to the CNN architecture for image classification, but with some modifications to adapt it to text data. The convolutional layers are typically 1-dimensional (1D) convolutional layers, which are applied to the embedding of the words in a sentence or a document. The pooling layers are typically max-pooling layers, which are applied to reduce the spatial dimensions of the data and to maintain the important features. The fully connected layers are responsible for classifying the data based on the features extracted by the convolutional and pooling layers.

In summary, CNNs are a powerful tool for analyzing grid-like data and are particularly useful for image and video processing tasks but they can be used for text classification tasks by applying 1D convolutional layers to the embedding of words in a sentence or a document and using max-pooling and fully connected layers to classify the data based on the features extracted by the convolutional and pooling layers. It is important to note that this type of architecture, while it can work well, is not the most common way to perform text classification and other architectures such as LSTM, GRU, or Transformer architectures are commonly used instead.

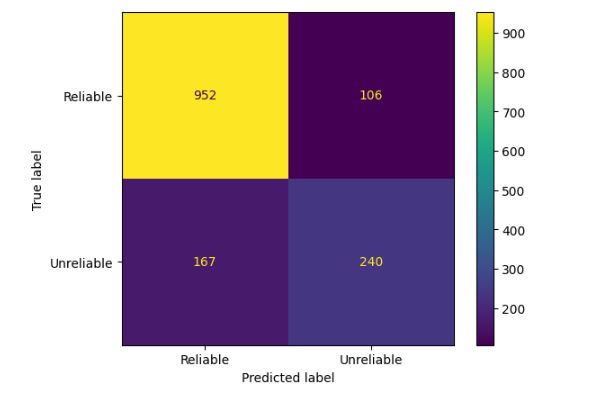
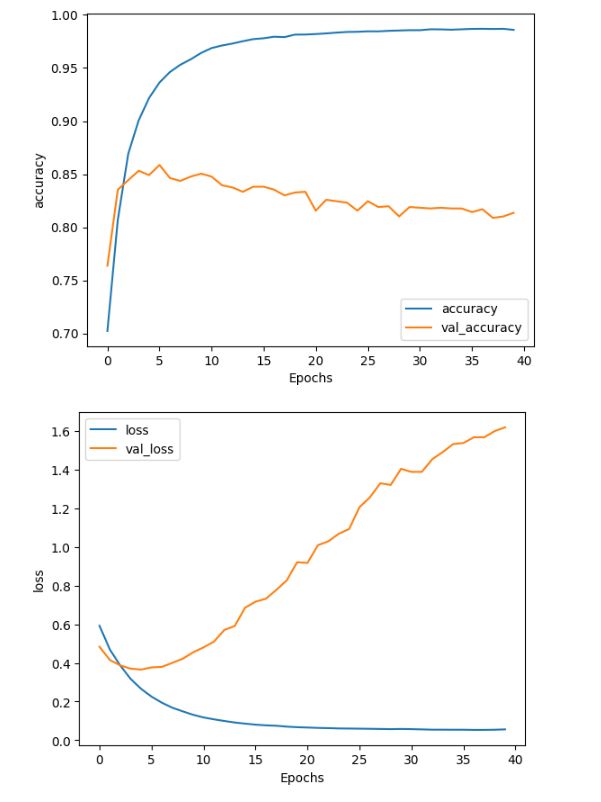
**Implementation:** Here are the general steps for implementing a CNN for text classification:

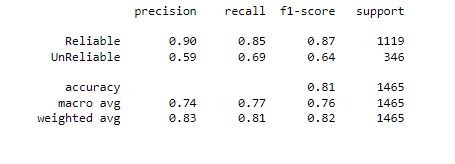
1. **Data preprocessing:** Clean and preprocess the text data, such as removing stop words, stemming, or lemmatizing the words, and tokenizing the text.
2. **Word embedding:** Train a word embedding model, such as word2vec or GloVe, on a large corpus of text, and use the trained model to obtain the embedding of the words in the text data.
3. **Define the CNN model:** Initialize the CNN model and define the architecture, including the number of convolutional layers, the number of filters, the filter size, the pooling layers, and the fully connected layers.
4. **Train the model:** Train the model on the text data and the corresponding labels, using a loss function such as cross-entropy loss and an optimizer such as Adam.
5. Evaluate the model: Evaluate the model on a validation or test dataset and measure its performance using metrics such as accuracy or F1-score.

## CNN Model Summary:

Table

Description automatically generated





# CNN + LSTM

A CNN-LSTM (Convolutional Neural Network-Long Short-Term Memory) network is a type of deep learning neural network that combines the strengths of CNNs and LSTMs. CNNs are particularly good at analyzing grid-like data, such as images and videos, and extracting features from the data, while LSTMs are particularly good at preserving long-term dependencies in sequential data, such as time series and natural language.

The idea behind a CNN-LSTM network is to use a CNN to extract features from the input data, and then pass the extracted features to an LSTM to preserve the long-term dependencies in the data. The CNN and LSTM are typically connected in a sequential manner, with the output of the CNN being passed as the input to the LSTM. CNN-LSTM networks are particularly useful for tasks that involve both spatial and temporal information, such as video classification and action recognition, and speech recognition.

**Implementation:** Here are the general steps for implementing a CNN-LSTM for text classification:

1. **Data preprocessing:** Clean and preprocess the text data, such as removing stop words, stemming, or lemmatizing the words, and tokenizing the text.
2. **Word embedding:** Train a word embedding model, such as word2vec or GloVe, on a large corpus of text, and use the trained model to obtain the embedding of the words in the text data.
3. **Define the CNN-LSTM model:** Initialize the CNN-LSTM model and define the architecture, including the number of convolutional layers, the number of filters, the filter size, the pooling layers, the LSTM layers, and the fully connected layers.
4. **Train the model:** Train the model on the text data and the corresponding labels, using a loss function such as cross-entropy loss and an optimizer such as Adam.
5. **Evaluate the model:** Evaluate the model on a validation or test dataset and measure its performance using metrics such as accuracy or F1-score.

## CNN Model Summary:

Table

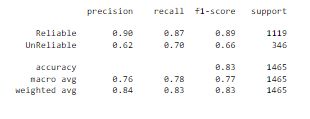
Description automatically generated with medium confidence

Chart, line chart, histogram

Description automatically generated

Chart, treemap chart

Description automatically generated



# BERT

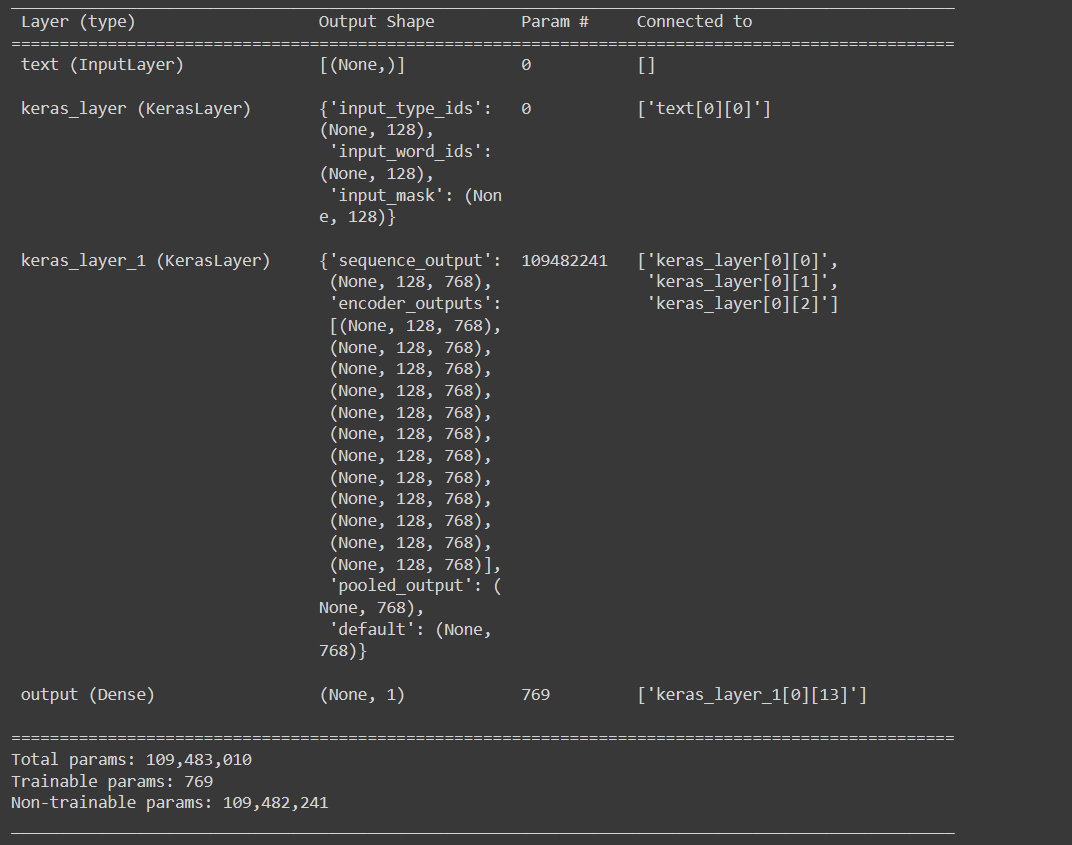
BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained transformer-based neural network model for natural language processing (NLP) tasks such as text classification, question answering, and named entity recognition. BERT is trained on a large corpus of text using a technique called unsupervised pre-training, which means it is trained on a massive amount of text data without any specific task in mind. This allows it to learn the general patterns and features of the language, which can then be fine-tuned for specific NLP tasks.

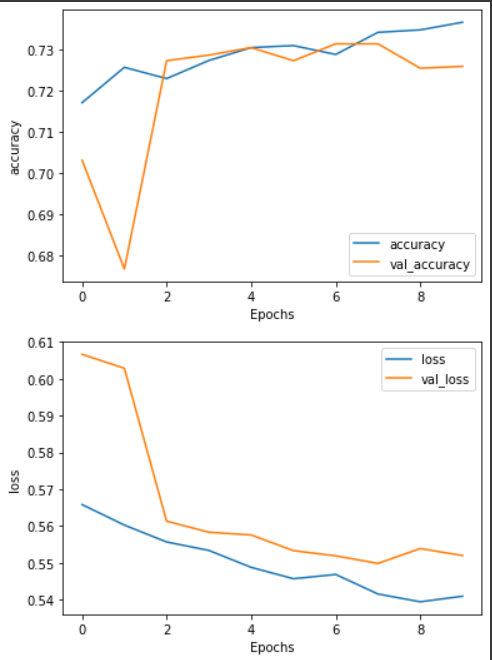
BERT is unique because it is bidirectional, meaning it considers the context of the words both to the left and to the right of the current word, rather than only the context to the left as in traditional models. This allows BERT to better understand the meaning of the words in the sentence and improve the performance on NLP tasks. BERT has achieved state-of-the-art results on a wide range of NLP tasks and has become a popular choice for natural language understanding and text classification tasks. It is available in several pre-trained versions, such as BERT-base and BERT-large, which can be fine-tuned to specific tasks and datasets.

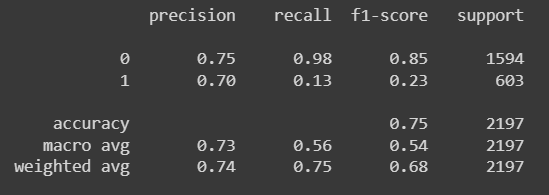
**Implementation:** Here are the general steps for implementing BERT for text classification:

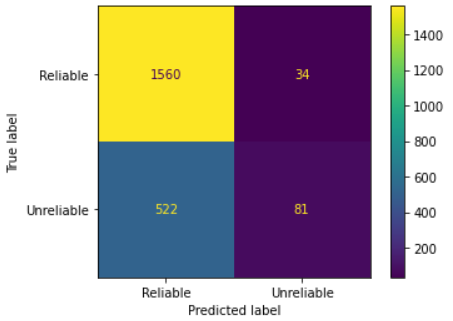
1. **Data preprocessing:** Clean and preprocess the text data, such as removing stop words, stemming or lemmatizing the words, and tokenizing the text.
2. **Download a pre-trained BERT model:** Download a pre-trained BERT model from the HuggingFace library or TensorFlow Hub, such as BERT-base or BERT-large.
3. **Tokenize the text data:** Use the BERT tokenizer to tokenize the text data, which will convert the text into a format that the BERT model can understand.
4. **Create a data loader:** Create a data loader that loads the tokenized text data and labels and creates batches for training and evaluation.
5. **Define the model architecture:** Define the architecture of the model, including the pre-trained BERT model and the additional layers for the classification task.
6. **Fine-tune the pre-trained model:** Fine-tune the pre-trained BERT model on the text classification task using a loss function such as cross-entropy loss and an optimizer such as Adam.
7. **Evaluate the model:** Evaluate the model on a validation or test dataset and measure its performance using metrics such as accuracy or F1-score.

## BERT Model Summary:









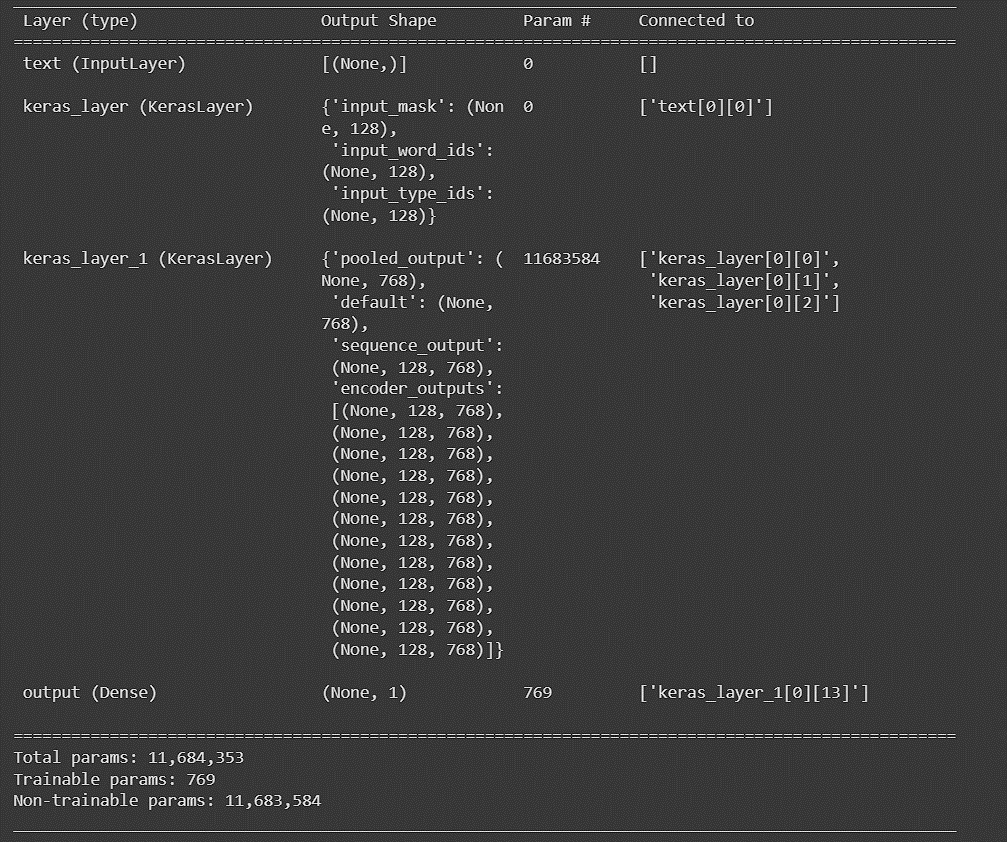
# ALBERT

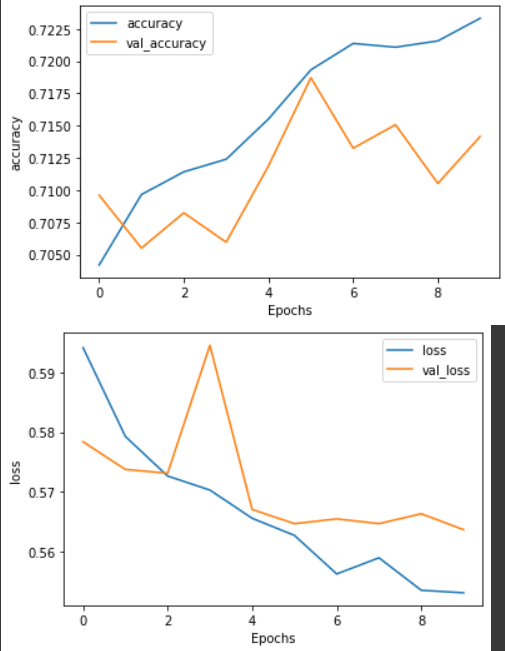
ALBERT (A Lite BERT) is a light version of BERT (Bidirectional Encoder Representations from Transformers), a pre-trained transformer-based neural network model for natural language processing (NLP) tasks such as text classification, question answering, and named entity recognition. ALBERT is similar to BERT but it is designed to be more lightweight and efficient, while still maintaining similar or even better performance on NLP tasks. It was achieved by applying two main techniques: factorization and cross-layer parameter sharing.

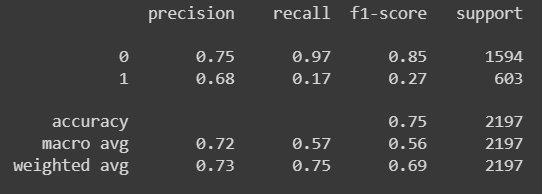
Factorization of the transformer-based architectures was used to reduce the number of parameters in the model, which makes the training and inference faster and more memory-efficient. Cross-layer parameter sharing was used to share the parameters across the layers, which also reduces the number of parameters. ALBERT is available in different versions such as ALBERT-base and ALBERT-xxlarge, which can be fine-tuned to specific tasks and datasets. It has been proven to have better performance on many NLP tasks compared to BERT and it is widely used in natural language understanding and text classification tasks.

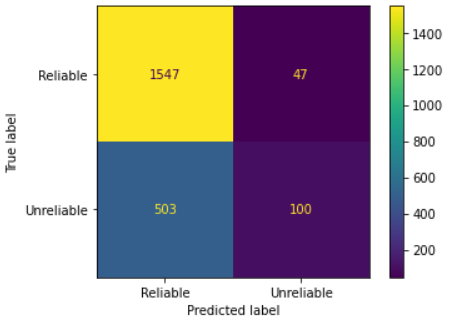
In summary, ALBERT is a light version of BERT, a pre-trained transformer-based neural network model for natural language processing tasks that is more lightweight and efficient while still maintaining similar or even better performance on NLP tasks. It was achieved by applying factorization and cross-layer parameter-sharing techniques. It is widely used in natural language understanding and text classification tasks.

## ALBERT Model Summary:









# Comparison Table

|  |  |  |
| --- | --- | --- |
| **Models** | **Accuracy**  **(Training Dataset)** | **Accuracy**  **(Testing Dataset)** |
| LSTM | 99.75% | 81.30% |
| GRU | 98.54% | 78.98% |
| CNN | 98.59% | 81.36% |
| CNN + LSTM | 98.59% | 82.80% |
| BERT | 73.67% | 75% |
| ALBERT | 72.33% | 75% |