Multimodal Fake News Detection using Text and Image

by

Trisha Biswas
20101628
Tasmim Afroj Lamia
19301190
Tarikul Islam Shykat
19301008
Md. Arifin Ahmed Rafi
19301009

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
May 2023

Declaration

It is hereby declared that

1. /we completed the thesis as part of our degree program at Brac University.
2. There is nothing in the thesis that has been published or authored by a a third source, unless properly cited through complete and precise reference.
3. The thesis doesn't contain any information that has been approved or submitted for any other University or Institution's diploma or degree
4. All sources that were significant for support have been mentioned

Student's Full Name & Signature:

Trisha Biswas	Tasmim Afroj Lamia
Trisha Biswas 20101628	Tasmim Afroj Lamia 19301190
Tarikul Islam Shykat	Arifin Ahmed Rafi
Tarikul Islam Shykat 19301008	Md. Arifin Ahmed Rafi 19301009

Approval

The thesis/project titled "“Multimodal Fake News Detection using Text and Image” submitted by

1. Trisha Biswas (20101628)
2. Tasmim Afroj Lamia (19301190)
3. Tarikul Islam Shykat (19301008)
4. Md. Arifin Ahmed Rafi (19301009)

Of Spring, 2023 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on May 22, 2023.

Examining Committee:

Supervisor:
(Member)

Farig Yousuf Sadeque

Dr. Farig Yousuf Sadeque
Assistant Professor
Department of Computer Science and Enginneering
BRAC University

Program Coordinator:
(Member)

Name of Program Coordinator
Designation
Department
BRAC University

Head of Department:
(Chair)

Sadia Hamid Kazi, PhD
Chairperson and Associate Professor
Department of Computer Science and Engineering
BRAC University

Abstract

Development in information and technology has made the communication easier in the recent decades. Easy access of social media is creating restraints amid of differentiating fake and real news. In the recent period the problem has increased drastically and use of image is making the news more impactful. Even though news websites are publishing the news and provide the source of authentication still there are other portals and platform which intentionally spread fake news to exploit an event. In this paper we proposed a hybrid system where we are combining CNN and RNN to detect fake news . We applied two techniques to reduce the model complexity and increase accuracy based on text data and image.With this system, detecting fake news it'll stop misleading people and creating an unstable situation as well as taking benefits of the situation

Keywords: Natural Language Processing, Fake News, Real News, Classification, Support Vector Machine, Multinomial Naive Bayes, BERT, Bi-GRU, Bi-LSTM, Word-Embedding.

Acknowledgement

First of all, glory be to the Great Allah, with whose help we were able to finish writing our thesis without too many setbacks.Second, we thank our advisor Farig Yousuf Sadeque sir for his thoughtful guidance and help during our task. He came to our aid anytime we needed it.
And last but not least, our parents, without whose continuous support it could not be possible. We are currently preparing to graduate thanks to their kind prayers and support.

Table of Contents

List of Figures

List of Tables

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional NeuralNetwork
EDA	Exploratory Data Ananlysis
GRU	Gated Recurrent Unit
HAN	Hierarchical Attention Network
IDF	Inverse Document Frequency
LSTM	Long Short Term Memory
ML	Machine Learning
RNN	Recurrent Neural Network
SVM	Support Vector Machine
TF	Term Frequency

Chapter 1

Introduction

1.1 Thoughts behind the Fake or Real News detection model

Today, "Fake News" is a common occurrence in our daily lives. But we frequently give little thought to what it implies or how it might erect our lives. False information is disseminated and presented as actual news in an eect to influence people. Additionally, the employment of many photo types has increased the issue's sensitivity and plausibility. In earlier decades, the issue could not have attracted as much attention due to a lack of internet access and current technology. Today's advancements in the information and communication fields have greatly accelerated the dissemination of false information and other forms of images. The usage of images makes it harder for the locals to distinguish between true and false news. Social media is an extremely efficient method for quickly getting news out to a big audience. A group of people are using websites like Facebook, Twitter, Instagram, and others to distribute this false information. The general public is terrified by this fake news, which also causes them to form misconceptions about society that are not supported by facts. The motivation behind spreading fake news diers depending on the subject. In the past, fake news about political matters has had a disastrous influence on people's daily lives. A million people were misled by this information in under a minute. The purpose of disseminating these false reports is primarily to incite unrest among communities and disturb the peace. The graphics are used to call attention to the news. On social media, spreading false information occasionally serves the purpose of gaining popularity. Even though popularity tends to fade quickly, its influence on people's lives endures. The way people think can also have an impact on a society's framework. Social media users are currently most likely to disseminate news without verifying the appropriate credentials. The scenes underlying the growth of fake news concerning sports, aside from politics, are most likely to incite the supporters of one group against the members of other groups and cause anarchy. The regular people, whose daily lives are impeded, are the victims of these social and community disruptions. In order to recognize fake news and prevent it from spreading, we are attempting to oer a model in this article. By doing so, we can expect to have a better society with greater peace and harmony. By separating bogus from real, our approach aims to aid the populace. We utilized an RNN model to analyze the text, and a CNN model to analyze the images.In order to classify the news into two categories—Fake and Real—this hybrid model will first aggregate the data from the news and compare it to the dataset. In order to simplify the process and increase accuracy, two strategies will be integrated. Basically, regarding the task of predicting fake news, we declare news headlines, societal situations, and related ancillary data as input and output will label either as ”fake” or ”real” depending on the input. The goal of fake news identification is to evaluate whether a news item is authentic or fraudulent given a multi-source news dataset and the social circumstances of news consumers. Our model must be able to handle the number of co inputs since fake news uses multimedia data to deceive readers and proliferate. As far as we are aware, we suggested fake news identification for brand-new and time-sensitive events, which can recognize fake news based on multiple aspects and learn portable aspects by deleting the occurrence aspects. We suggest an edge event adaptive perceptron to do this. To lessen the negative consequences fake news can have, the spread of false information across various venues, particularly online ones, has not yet been totally stopped or reduced back. The cause is that there is currently no mechanism in place that can manage fake news with minimal to no human intervention. With a starting collection of instances to be built on, finding demonstrated that machine and learning algorithms could be able to recognize fake news. The task of detecting bogus news has considerable potential for deep learning approaches. But little research indicates the value of neural networks throughout this field. The structure of this essay is as follows: A overview of earlier investigations is included in Section 2. The description of the dataset, the pre-processing steps in section 3, and the classification models as well as the workflow are included in Section 4. Explanation of the suggested technique. Results of the models as well as result analysis are described in section 5. Finally, the summary of this paper is found in Section 6.

1.2 Problem Statement

The challenge of determining if a news story is fabricated or not utilizing both conventional methods and cutting-edge machine learning techniques.

1.3 Description of the Problem

For a very long time, fake news has been an issue in the mainstream media, and any news organization can disseminate fake news pieces. As a problem of authenticity score prediction, the fake news identification problem is defined. Online news consumers are generally the targets of fake news because the majority of social media platforms do not have rigid standards governing overall information flow on their websites. Additionally, news in online platforms is easily accessible and relatively inexpensive, allowing ”Fake News” to be widely disseminated. Even major social media platforms like Facebook, Twitter, and Google are finding it difficult to combat the rise of fake news on a worldwide scale. Fake news is written with a intention to mislead readers into believing false information, trying to make it challenging and complicated to identify based solely on news content. Hence, we must consider assistant information like the news source, comments on the article, photos used in the article, headline etc. to make an accurate decision whether the news article is fake or not. Using this supplementary data fake news detection is troublesome due to the fake news article's massive, imperfect, unstructured, and noisy data. In this research, we tried to create a solution that may be utilized to identify and filter out the articles that include fake news in order to assist consumers in avoiding being attracted in by click baits. Disguised information can also have major detrimental effects on a person as well as the entire society by skewing people's perceptions of the world and causing social chaos. As a result, it's critical to identify a method for detecting fake information that will be beneficial for both readers and legitimate news organizations. Here, we have suggested a method that can effectively identify fake news using several machine learning techniques. We intended to concurrently detect fake news from the websites relying on many sorts of diverse sources of information, including both linguistic documents and the copyright and linkages among them. We constructed this collaborative filtering problem as a reliability reasoning problem, in which the validity of accurate news will be higher whereas that of fake news would be lower. An entire concept and formulation of the issue are essential and important before exploring the new research challenge of fake news identification which is called problem formulation. From the web-based media, a collection of text documents on the profiles, titles, and substance of the news pieces, authors, and topics can be obtained. It will be necessary to use a powerful extraction of features and training models to gather data demonstrating their authenticity. The author and section linkages between them also serve as indicators of the reliability labeling of news headlines, writers, and topics, which have very significant associations. For more accurate results in determining the legitimacy of fake news, it will be helpful to effectively include these connections into the structure of learning.

1.4 Research Objective

By offering an automatic fake news detection approach for both long sequence textual data, like news articles, and short sequence textual data, like tweets, this research aims to supplement existing research. We will try to create our model by tuning different ML models. The objectives of this research are:

1. To comprehend the detrimental effects of fake news in this modern world.
2. To develop a model that can classify a given piece of news is whether true or false.
3. To implement the proposed model and evaluate it.
4. To overcome some of the drawbacks of other existing fake news detection models.
5. To make suggestions for upgrading the other models.

This research will help people:

1. Distinguishing fake or real contents

2. By informing others about the rumors

3. In taking political or business decisions

4. To avoid clickbait

5. Preventing people from being manipulated by fabricated news

Chapter 2

Related Work

Despite being a relatively new concept, fake news detection has gained a lot of interest due to how quickly it is propagating. In current history, researchers have made multiple attempts to build models that can classify and filter fake news using multiple techniques. The majority of analyses and research have concentrated on various sorts of fake news on numerous platforms. This section contains a review of the literature on fake news from several categories, including existing and related works. We have divided the issue of detecting fake news up into different subgroups:

Social Media :

Misinformation frequently spreads due to a user's online status on social media, which is the fastest venue for doing so. Users are more likely to have their postings accepted and disseminated as fact if a group of people trusts them. There are millions of robots, or bots, living within social media in addition to the billions of people that use it. Bots aid in the spread of false information and increase its perceived popularity on social media. Moreover Trolls are persons who use their social media accounts for one main purpose: to quarrel with others, to attack and denigrate other users and celebrities, to try to discredit views they disagree with, and to frighten others who publish those beliefs. Additionally, they promote and support false news that aligns with their ideologies. Some research work has been conducted on this regard.
By utilizing the CNN and RNN models from the social media platform "Twitter," Ajao et al.,[1] attempted to develop a system that can distinguish between real and fraudulent news. The system recognized significant traits related to the news without any prior domain knowledge. Zubiaga's contribution to the PHEME dataset, which only had the two categories of rumors and non- rumors, was used in the research to compile a collection of roughly 5,800 authentic tweets related to five rumors. False news was broken down into two categories in order to be controlled by the system: a. characteristics identification without prior information, and b. determining and classifying the false news. There are three deep neural network models has been used for text and image analysis which are LSTM, LSTM with dropout regularization, and LSTM with CNN. Different RNN methods, including truncated propagation, penalties, gradient clipping echo state networks, and others were employed to analyze the texts, but the major focus of the article is on LSTM because it can keep track of previous phase's memories. After all was said and done, the system's LSTM approach produced an accuracy of 82%. Additionally, when

comparing LSTMDrop and LSTM-CNN, LSTM performed best in terms of precision, recall, and f-measure (80% accuracy). LSTMDrop had the lowest accuracy (74%) as a result of underfitting.
In another similar research work done by Nasir et al.,[6] attempted to identify the research gap on pertinently fresh datasets and offered a new model to address the holes. This new model is the innovative hybrid deep learning model integrating CNN and RNN.. In an effort to identify bogus news, they also aimed to provide future research with a direction. They used Elhadad's "FA- KES" dataset (804 articles), which is a publicly accessible dataset, to carry out their investigation. In addition, they used the "ISOT" dataset (45000 articles) provided by Ahmed as well as another dataset to help guide future studies. There are seven supervised machine learning approaches employed. Word2Vec and GloVe have been used to manage word embedding. Additionally, they employed tokenization and LSTM. For the one dimensional CNN layer, they employed the Rectified Linear unit (ReLu) function and for the LSTM layer, they used the Sigmoid activation function. Additionally, decision trees, stochastic gradient descent, naive bayes, and logistic regression were employed. For the ISOT dataset, the system produced results with almost 100accuracy, whereas for the FA-KES dataset, it produced results with 60% accuracy. For both datasets, the hybrid CNN and RNN results were noticeably superior to the non-hybrid ones.
Furthermore, in another research paper which has been attempted by Abbas et al.,[9] have proposed a deep learning-based system that incorporates CNN and RNN is used in a fake news detection model that intends to investigate the identification of false news in online social networks using particular news articles, their authors, and their subject. They used the feasible dataset called LIAR which consists of 12,800 remarks of varying length that were gathered from POLITIFACT.COM more than ten years ago. The dataset includes an external link for every case that leads users to an external source for in-depth analysis. The RNN, LSTM, and CNN algorithms serve as the baselines for learning techniques based on this dataset. 70% of the data taken into consideration will be used for model training and 30% for testing in order to increase the precision and accuracy of the proposed fake news detection model. This model will generate a conclusion based on the inference of credibility, where genuine news will be given a higher credibility rating than false news. In this model, the result of the preprocessed data was taken as the input of RNN models while the output was considered as input for the CNN models. The CNN section produces a vector output that, taken as a digital representation of the sentence's properties, describes the sentence's characteristics. Due to their efficiency in determining if a news item is legitimate (V) or false (F) using the output from the CNN layer, LSTM was chosen as the final step. In comparison to individual CNN and RNN models, the suggested model's (a hybrid LSTM-CNN-RNN architecture) accuracy will be about 5% higher.

Epidemic :

SARS-CoV-2, a coronavirus that first surfaced in December 2019, is the cause of COVID-19, the illness it causes. COVID-19 can be deadly, and it has resulted in millions of deaths worldwide as well as long-lasting health issues in some survivors. However, there were numerous news stories on various platforms that led to the erroneous information spreading, which made people quite confused and anxious.Many

people even refused to receive vaccinations due to erroneous information. Numerous deaths were also brought on by false information about COVID-19. Because of this, researchers have worked to develop a technique that can determine whether or not news about COVID-19 is real.
In order to stop the spread of widespread false information regarding the COVID-19 pandemic, Gundapu and Mamidi[3] looked for a technique to evaluate the accuracy of the information in their work from 2021. They used a dataset made available by the Constraint AI–2021 shared task organizers, which contained the COVID-19 dataset for fake news in English, which collected 10,700 data points from various online social networks, such as Facebook, Instagram, and Twitter. In order to discover faults in the errors in the dataset that was provided, they frst constructed machine learning (ML) methods employing Term Frequency and Inverse Document Frequency (TF-IDF) feature vectors. LSTM, BiLSTM with Attention, CNN, and CNN + BiL- STM were among the deep learning models used. Then, using three transformer models (BERT, XL- Net, and ALBERT), they developed an effective ensemble model for detecting bogus news on social media sites. The findings revealed that for COVID-19 misinformation detection tasks, Transformer-based models outperform other machine and deep learning models by a signifcant margin. The transformer model approximation is pretty comparable to the f1-score bidirectional LSTM with attention technique. The BERT, XLNet, and ALBERT outperform deep learning models. The ensemble of the transformer-based model generates the fnest Result on the testing set, 0.9855.
In an additional related study on the epidemic fake news in 2021, Kaliyar, Goswami, and Narang [4] suggested a hybrid model for fake news detection that combined convolutional layers with various kernel sizes and LSTM layers followed by three dense layers. The FN-COV dataset was utilized. Around 69,976 news articles totalling 44.84% bogus were gathered for the dataset creation from the GDELT project 1, which was funded by Google. Additionally, they used the PHEME dataset, which is a corpus of tweets pulled from Twitter that were posted during five different events related to breaking news. Using both real-world fake news datasets and their proposed C-LSTM network, experiments have been done in this study. In their neural network, three thick layers have been taken into account. To avoid overfitting, they chose a value for dropout and utilized binary cross-entropy as the loss function. Their suggested models, which combine CNN and LSTM layers and are deep and hybrid in nature, have done very well, achieving accuracy levels of 98.62% with the FN-COV dataset and over 90% with the PHEME dataset, which is 5% better than state-of-the- art techniques. They attained the F1-score with FN-COV at 99.40% and PHEME at 90.30%.
Another similar research was conducted by Wani et al.,[7] where they have evaluated various supervised text classification methods in 2021 using the Contraint@AAAI 2021 Covid-19 Fake news detection dataset. For the purpose of detecting fake news, they assessed current developments in deep learning-based text classification algorithms.They used the Contraint@AAAI 2021 Covid-19 Fake news detection dataset, which contained tweets and the labels that went along with them. 10,700 media articles and posts were gathered from multiple platforms, including 6420 samples from the train data, 2140 samples from the test data, and 2140 samples from the validation data. They used long short-term memory (LSTM), bi-LSTM + attention, hierarchical attention networks (HAN), transformer-based designs like BERT and

DistilBERT, convolutional neural networks, and LSTM. TensorFlow 2.0 was used to train all of the models.Each model was trained for a total of 10 epochs, and the optimal epoch was chosen using validation loss. Additionally, BERT and DistilBert models were manually pre-trained on a Covid tweets dataset using the huggingface library. After reviewing the outcome, The best accuracy recorded when employing the SVM model is referred to as the baseline accuracy. The Covid-19 tweets corpus pretrained BERT and DistilBERT models outperform the ones that were just fine tuned on the dataset. The non-transformer versions do the best overall, with an accuracy of 94.25%, according to HAN. With an absolute accuracy difference of 3–4%, transformer-based models perform better than other fundamental models. We improved accuracy over the baseline accuracy of 93.32% to a maximum of 98.41% using the sung language model pre-training on BERT.

Politics :

Political news has a greater impact on a country as well as the world. We can see from previous experiences, during an election, war or different national or international crisis, fake news like rumors, propaganda spread a lot. Many international and national social media based newspapers spread clickbait news articles for more people engagement to their sites. However, it distracts people from the main problem and creates political and economic unrest to a country.
To work on the fake news related to political issues, this study by Mr. Nishant Rai et al.,[10] demonstrates a content-based approach to classification that uses a BERT model with output coupled to an LSTM layer. Based on the news title, a classification is made. A feed-forward network with 768 hidden sizes has been employed with the model. To train and evaluate the model, researchers used the Fake News Net dataset. Politifact and GossipCop are the two subsets that make up the Fakenews Net dataset. Politics and entertainment news are included in the subsets, though. Accuracy, precision, recall and F1 score have been calculated to evaluate the data. The accuracy of the BERT and BERT-LSTM based model is followed by 86.25 percent and 87.75 percent on the Politifact subset. On the other hand, for GossipCop subset the accuracy of BERT and BERT-LSTM based models is 83.00 percent and 84.10 percent respectively. Here, the BERT-LSTM based model performs better. The precision, accuracy and recall for BERT and BERT-LSTM based models are followed by 0.90, 0.87, 0.88 and 0.91, 0.90, 0.90. The training was done on 80 percent of the data for training and 20 percent of data for testing which have been selected randomly. The proposed model can achieve greater performance by fine tuning both the BERT and other following layers.
In continuation of the work this paper by Kaliyar et al.,[4] suggested a deep learning method based on BERT called ”FakeBERT” that combines BERT with numerous CNN layers. From different social media networks, including Facebook, Twitter, Instagram, and others, they have gathered real-world news data from the 2016 U.S.A. General Presidential Election. CNN, LSTM, and the proposed model FakeBERT are deep learning models that were used in the studies. The suggested model, however, obtained validation accuracy of 98.90%, whereas CNN and LSTM based model reached 92.70% and 97.55% correspondingly with 10 epochs. This was done by using multiple layers of CNN with dropout and activation function RELU. The FakeBERT model has the lowest cross entropy loss among these three. In this study, Mr. Kaliyar et al. have shown the efectiveness of their suggested model (FakeBERT), which

combines BERT with three parallel blocks of 1d CNN that have varied kernel sized convolutional layers to get a higher level of accuracy.
In the research conducted by Khan et al.,[5] tried to show a benchmark study of diferent models to identify fake news for online platform. In this research they performed several models to compare the accuracy. In this research Khan et al., used total three datasets. First, they choosed the "LIAR" dataset which is publicly available dataset and it includes short statements from POLITIFACT.COM which involves 56% true data and 44% fake data. Another dataset they used is "Fake or Real News " by George Mclntire. It involves around 6300 data regarding 2016 USA election cycle where half were true data and half were false data. The third dataset they used was "Combined Corpus" dataset which contains about 80000 news regarding economy,politics, health,sports etc. where 51% were real data and 49To identify the fake and real news tgey used lexical and sentiment features to identify positive and negative sentiments, n-gram features (both bi-gram and uni-gram), empath generated features to fnd violence, crime, pride etc. They used GloVe algorithm for unsupervised data. For frst three model they used SVM, LR (logistics regression), Decision tree etc. Then they used Naive bayes classifer, k-NN classifer. They further evaluated six deep learning models CNN, LSTM, Bi-LSTM, C-LSTM, HAN and convolutional HAN. They also evaluated the dataset using BERT, RoBERTa, DistiBERT, ELECTRA, ELMo. In the study they found that BERT-based models did really well with small datasets in which RoBERTa achieved over 90% accuracy. They also observed that LSTM based models had gradually improved when dataset increased from smaller to larger. The models of HAN showed best performance on LIAR dataset (accuracy 75%). C-LSTM also showed the best performance on combined corpus dataset. But comparing overall performance they found that RoBERTa showed the best among all.

Financial :

Now, investors rely heavily on financial news as a basis for research and making investment choices. On the other hand, fake financial news is overflowing into daily lives of individuals. Such types of fake news may have an influence on public perception and offer some crooks an opportunity to manipulate the financial market. In order to resolve the issues introduced by financial fake news, Zhi et al.,[8] have proposed a multi fact CNN-LSTM model to outperform the existing machine learning models that work manually by extracting features to identify the financial fake news so that investors can use the valid source of news when making investment decisions. They obtained the dataset for this model from a number of financial websites, including East Money Information, Sina, Headline, and others. A collection of 8000 labelled samples was created by using the 100 events from the previous five years to label other samples and associate them as labelled samples. Every sample contained titles, texts, sources, as well as responses of the data was set aside for model tuning and holding, while the train dataset, validation dataset, and test dataset were distributed in a 3:1:1 ratio. The input of this model was divided into two sections in order to perform better than the other models: the combination of news, sources and market data; and the comments. Character-level features were extracted using character convolutional layers, and when the character features sequence was encoded to a vector, an LSTM layer was implied. Prior to decoding, an attention mechanism was used to determine the relationship between the contents

and the comments, and two LSTM layers were then used as decoders to predict the output characters. Their evaluation shows that CNN-LSTM model achieved 92.1% accuracy whereas LSTM, SVM, Tree LSTM models achieved 73.1%, 77.2%, 80.8% accuracy respectively. This study demonstrated that the fundamental LSTM and char CNN can successfully learn the differential features.
We observed another comparable research paper by Zhang et al.,[2] where on platforms that crowd-source content for the financial system, they have examined fake news. They used a special dataset of clear-cut fake news articles that were investigated by the Securities Exchange Board, dissemination statistics of these kind of news on other online platforms, and financial performance data of the focal organization to develop a well-justified and clear machine-learning structure to anticipate fake financial news on social networks. To improve its consistency and comprehensibility, they accomplished the Truth-Default Theory (TDT), a cutting-edge deceit theory. Based on TDT and earlier case study, researchers have offered a comprehensive set of quantifiable indications of financial news's background and purpose, author mindset, 3rd parties' response, material consistency, and data coherence. Their in-depth research and comparative evaluation demonstrate that this approach is more effective at identifying bogus financial news than previous parameter allows and sensory information models. They chose Seeking Alpha as the resource for financial news due to its substantial influence on the stock market and since the SEC examined and fled money laundering charges involving people and companies in April 2017 for propagating FFN, the bulk of which had been released on this website. Their sample comprised of 381 financial fake news and 6866 true financial news. They employed various methods of text mining (word2vec, LSA, LDA, LIWC, etc.) to extricate benchmarks of information cooperation and communications from unstructured information. For every news item in the survey, they used the latest details: headline, date of publication, URL, primary contents, writer profile, list of stocks, remark document discussed in the article. They started by randomly splitting their dataset together into training sample and a test sample set with something like a 4:1 ratio.
The reliable news in the training phase was then under sampled, and the data in the test dataset was kept in its initial, innately state. This allowed them to create a symmetrical training set. They do sample t-test on F1 scores contrasting TDT based systems and the benchmark model to ascertain whether the higher level is statistically relevant. After reviewing the outcome, they discovered that Random Forest, when using all the characteristics, linguistic inquiry and word count, has the highest F-1 score of 94.1%, which is pretty close to Gradient Boosting's score of 92.7%. However, when employing the chosen features, gradient boost outperformed Random Forest, scoring 93.6% instead of 90% on the F-1 score. They split the financial news articles into a training set for the classifiers to learn from, and a test set to assess how well the classifiers performed during the previous time period. They used to 2013-publish financial news items as a test set to gauge how well the classifiers performed over time. Models containing all TDT features outperform others in both earlier and later times, they discovered. Additionally, when applying all TDT characteristics, the deprecation of F1 scores is substantially slower for ensemble learning methods (RF and GB).

Chapter 3

Dataset

3.1 For Recurrent Neural Network :

3.1.1 Description of the Data :

A dataset from Kaggle called WELFake dataset was used for the study. (WELFake) is a dataset comprising 72,134 news stories, 35,028 of which are true and 37,106 of which are fraudulent. To avoid classifier overfitting and to give more text data for better ML training, authors combined four well-known news datasets (Kaggle, McIntire, Reuters, and BuzzFeed Political) for this purpose. Four columns make up the dataset: Serial number (beginning at 0), Title (pertaining to the text news heading), Text (pertaining to the news content), and Label (0 = fake and 1 = real). Out of the 78098 data entries in the csv fle, only 72,134 are accessed in accordance with the data frame.

3.1.2 Data Exploration :

Data exploration helps provide a more comprehensive picture of particular topics to be researched.Users evaluate information using manual and automated approaches to choose the optimal model for the succeeding stages of data analysis. ML algorithms or exploration software are frequently used to rapidly identify relationships between different data parameters and data frameworks and provide data values that can indicate patterns or exciting areas. We evaluated several dataset using manual and automated approaches to choose the optimal model for the succeeding stages of data analysis. Given that we are working with massive amounts of data, data exploration was necessary for our solution.

3.1.3 Data Pre-processing :

The most important phase in the machine learning workflow is preparing data before using it with a model. It aids in increasing the model's precision, decreasing the time and resources needed to train it, avoiding overfitting, and making it easier to understand.
We pre-processed the dataset after analyzing. The steps that were followed to pre-process the data are given below:

1. Removing special characters: The most common places to find these characters are in remarks, references, currency figures, etc. These characters introduce noise into algorithms and don't improve text comprehension.

2. Lower casing: All data should be converted to lowercase as this will aid in preprocessing and later parsing stages of the NLP application.

3. Stopwords removal: Stop words include a, an, the, is, has, of, are, and others. Most of the time, they amplify the features. In order to create a cleaner dataset with better features for a machine learning model, stop words should be removed. To improve search performance, all stop words are eliminated from multiple word queries, including frequent words like a and the.

4. Lemmatization: Lemmatization reduces any word to its fundamental root mode. Lemmatization allows end users to search for any variation of a basic term and get relevant results. Lemmatization reduces a term to its logical base form, or lemma, while taking context into consideration. Sometimes a word will have numerous distinct lemmas.

5. Clean sentence: By removing meaningless words from the dataset, such as zzzz and aaaa, using the NLTK package clean sentence, it was possible to obtain meaningful terms for the model's training.

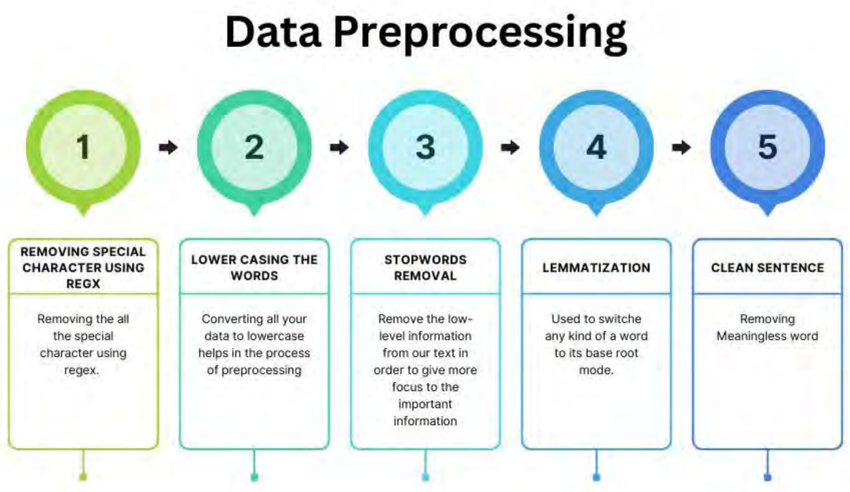

Data Preprocessing
1
2
3
4
5
REMOVING SPECIAL CHARACTER USING REGX
Removing the all the special character using regex.
LOWER CASING THE WORDS
Converting all your data to lowercase helps in the process of preprocessing
STOPWORDS REMOVAL
Remove the low-level information from our text in order to give more focus to the important information
LEMMATIZATION
Used to switche any kind of a word to its base root mode.
CLEAN SENTENCE
Removing Meaningless word

Figure 3.1: Data Pre-processing Steps

Following that, we ran several searches to see if there was any correlation between the data. We discovered that, on average, fake news titles contained seven words, whereas real news titles contained fewer words. However, the text of the fake news contained fewer words than the text of the actual news, which contained 228 and 261, respectively.

3.1.4 Pre-processing Techniques :

Count Vectorization :

Count Data preprocessing for natural language processing (NLP) tasks frequently uses a technique called vectorization. It works to transform a group of text files into a matrix representation, with each document being represented as a vector of term frequencies.

Following are the steps involved in count vectorization:

1. Tokenization : Each document is initially divided into separate words or tokens. Whitespace splitting and more sophisticated approaches like word tokenization libraries can also be used for tokenization.
2. Vocabulary Creation : The process of developing a vocabulary involves building a list of unique words—also referred to as terms or features—that exist throughout the entire collection of materials. The vocabulary contains words that are each given a different index.
3. Count calculation : Each term's count in the vocabulary is determined for each document. This is accomplished by calculating the number of times each term appears in the text.
4. Vector Representation : Each document is represented as a vector, with the length of the vector being equal to the vocabulary's size. The count of the matching term in the document is represented by the value at each index. A "term frequency" vector is a common name for the vector.

Text data can be easily and effectively converted into a numerical format that can be utilized as the input for machine learning algorithms using count vectorization. The classification of documents, sentiment analysis, topic modeling, and information retrieval are only a few NLP activities that frequently use it. It offers a straightforward yet useful model of text data that may show how words are distributed throughout various manuscripts. The resulting matrix is also known as a term-document matrix or a count matrix. The matrix's rows stand in for documents, and the columns for terms.

Term Frequency-Inverse Term Frequency (TF-IDF):

Information retrieval and text mining use the TF-IDF (Term Frequency-Inverse Document Frequency) statistic to quantify the importance of a term inside a document or a collection of documents. It is commonly employed in tasks like document classification, information retrieval, and text analysis.

By taking into account two parameters, the TF-IDF score assesses the significance of a term in a document.

1. Term Frequency (TF) : Term Frequency (TF) is a measure of how often a term appears in a document. It displays the frequency of a term in relation to the total number of terms in a document. A term's TF score increases with how frequently it appears in a document. The formula for TF(t, d) is (Number of

times term t appears in document d) / (Total number of terms in document d).

2. Inverse Document Frequency (IDF) : A term's rarity over the whole document collection is evaluated using the Inverse Document Frequency (IDF) method. IDF downplays the significance of terms that are frequently used in documents because they are probably less specific and ubiquitous. IDF scores are typically higher for uncommon phrases that appear in fewer documents.

   IDF(t, D) = log_e*(Total documents in the collection / Total documents with term t)*

   The TF and IDF numbers are multiplied to produce the TF-IDF score:

   IDF(t, d) * TF(t, d) = TF-IDF(t, d, D)

   A phrase may include significant information that is unique to a certain document if it is frequent in that document and relatively uncommon overall, as shown by a higher TF-IDF score.

Before using machine learning methods, text input is frequently preprocessed using TF-IDF. Unstructured text is converted to a numeric representation that can be utilized as input for a number of natural language processing tasks, including text classification, clustering, and information retrieval. The performance of these tasks can be enhanced by TF-IDF by giving larger weights to phrases that are more discriminative and informative.

Max-pooling :

The max pooling operation separates the input feature map into rectangular, non-overlapping ”pooling windows” or ”kernels,” and outputs the highest value found within each window. The spatial dimensions are decreased while the number of channels is left unchanged since it functions independently on each feature map channel.

1. Pooling Window : A fixed-size pooling window is established, usually with a stride that is equal to the window size. A typical pooling window is 2x2 or 3x3.

2. Non-overlapping areas: Based on the size and stride of the pooling window, the input feature map is separated into non-overlapping areas.

3. Maximum Value: The highest value is chosen for each region. The trait that dominates or is most significant in that area is represented by this value.

4. Output Feature Map: A feature map that has smaller spatial dimensions than the input feature map is created using the highest values from each region.

Several benefits of max pooling in CNNs include:

1. Translation Invariance: By choosing the highest value possible inside a pooling window, max pooling can accurately capture the most prominent feature in a region, regardless of its precise location. As a result, the model is more resistant to minor spatial translations in the input and achieves a certain amount of translation invariance.

2. Reduced Dimensions: Maxpooling shrinks the feature map's spatial dimensions, which reduces the number of parameters and layer-upon-layer computational complexity. By doing so, efficiency is increased and overfitting is reduced.

3. Feature Robustness: Maxpooling keeps the most noticeable features while removing unimportant or noisy information. To make the network more resistant to changes and noise in the input data, it helps to highlight the most discriminative characteristics.

Dropout :

Neural networks, particularly those used in natural language processing (NLP) applications, commonly use a regularization technique known as dropout. Dropout improves the models' ability to generalize while preventing overfitting.
In NLP, dropout is widely used in the neural network layers of recurrent neural networks (RNNs) or transformer-based models like the Transformer or BERT.
The dropout approach works during the training phase by randomly setting some of the output values or neural activations to 0. This dropout chance typically ranges from 0.1 to 0.5. If the dropout probability is set at 0.1, for example, 10% of the outputs will be zeroed out. Dropout has several benefits in NLP tasks, including:

1. Regularization : Dropout is a regularization technique that reduces the model's reliance on certain features or hidden units, hence preventing overfitting. As a result, the network is less likely to memorize noise or oddities in the training data.

2. Ensemble effect : Dropout can be explained by the ensemble effect, when a number of models are trained simultaneously utilizing different subsets of neurons. Dropout is often inhibited at test time, and predictions are made by the entire network. The ensemble effect of training dropouts helps the model perform better.

3. Reducing Co-Adaptation : Co-adaptation can be lessened by dropout, which helps network neurons. Co-adaptation occurs when neurons depend on one another so heavily that it is difficult for the model to generalize to novel inputs. Dropout destroys these connections, forcing the network to produce more independent and revealing representations.

A popular NLP technique called dropout can be used to a number of network levels, including the input embeddings, hidden layers, and output layers. Where to employ dropout depends on the architecture and details of the current NLP task. It is often determined by experimentation and performance analysis of validation sets.

Early Stopping :

Early stopping is a machine learning approach that is used, especially while training neural networks, to decrease overfitting and increase generalization. Early stopping is founded on the idea that training should be terminated as soon as a model's performance on a validation set starts to deteriorate.

The performance of the model is evaluated using a different validation set or a subset of the training data that is not used for training. The validation set is used as a substitute for the model's performance on unseen data. It is common practice to evaluate the performance of the model using an appropriate assessment statistic, such as accuracy, loss, or validation error.

The training procedure is carried out until the model's performance on the validation set begins to decline rather than continue to improve. When this deterioration is noticed, an early stop is made. The training phase is then completed when the model's parameters are set to the point where it performs best on the validation set.

The likelihood that a model will start to overfit the training set when it becomes too complex or begins to detect noise in the training data justifies early stopping. Overfitting is the term used to describe when a model performs well on training data but struggles to generalize to new, untested data. Monitoring the model's performance on the validation set helps prevent overfitting since early stopping blocks training before the model starts to memorize the training data too closely.

Early stopping enables balancing the performance of generalization and model complexity. Selecting a model that performs well on unobserved data without making too many compromises by overfitting the training data is beneficial.

The best time to cease training might vary depending on the dataset, the model architecture, and other variables. It is crucial to remember that early stopping is not always the best option. The optimal stopping criteria for a given work must be determined through experimentation and validation set performance analysis.

GloVe Word Embeddings :

The terms "Word GloVe embedding" seem to combine GloVe and Word Embeddings, two related ideas.
Word embeddings: These are dense vector representations of words in a continuous vector space that capture the semantic and syntactic relationships between words. Algorithms for machine learning can understand and interpret natural language because of these numerical representations, or embeddings.

GloVe : Using unsupervised learning, the word embedding generation algorithm GloVe (Global Vectors for Word Representation) creates new word embeddings. Its name, "Global Vectors," emphasizes the fact that it takes into account a corpus's statistics of word co-occurrence around the globe. GloVe creates word embeddings, which represent the semantic relationships between words, by analyzing the statistics of word co-occurrence in a significant text collection.

GloVe embeddings have gained popularity in the NLP community because of their efficiency and capacity to capture a variety of language features. They are frequently employed in activities like sentiment analysis, machine translation, and word similarity searches.

A significant collection of text is utilized as input to produce GloVe embeddings. In a sliding window across the collections the method determines the co-occurrence statistics of terms. A word co-occurrence matrix is then built, and dimensionality reduction methods like Singular Value Decomposition (SVD) are used to create lower-dimensional word embeddings.

Because they record both local and global context, GloVe embeddings are advantageous. The embeddings serve a variety of NLP tasks by representing the overall word relationships in the text.

Padding :

Padding is a technique used in data pre-processing to ensure that each data point is the same length or dimensionality by adding extra elements or values to a dataset. When working with sequential data, such text or time series, where the inputs could have varied durations, padding is typically used.

The fundamental purpose of padding is to provide the data a consistent shape or size, which is typically required by machine learning algorithms or neural networks that demand constant input dimensions. By adding extra elements to shorter sequences to make them equal in length, shorter sequences can be processed and computed more quickly.

It is usual practice to include different tokens or values in order to represent padding. For instance, using a specific padding token to stretch shorter phrases to the length of the largest sentence in the dataset is a common method in natural language processing applications.

The data can then be handled in a batch mode, which is more efficient computationally, once the padding has been applied. The masking or attention mechanisms in neural networks often do not support learning during the training phase because they ignore the padding bits.

Tokenization :

When a textual sequence is broken up into tokens, such as a sentence or document, it is said to have been tokenized. These tokens could be single words, subwords, or even characters, depending on the specific tokenization method used.

Tokenization is an essential step in natural language processing (NLP) and text mining tasks because it enables more detailed analysis and interpretation of textual data. Text can be broken up into tokens in order to extract valuable information. Tokens can then be further analyzed using a variety of techniques, such as word frequency analysis, word sequence analysis, or machine learning models.

The process for tokenization that is used will depend on the unique requirements of the project at hand. Tokenization can be done in many different ways.

1. Word Tokenization: This technique separates the text into discrete words depending on punctuation or whitespace. For instance, the phrase ”Tokenization is important!” would be tokenized into the following tokens: ”Tokenization”, ”is”, ”important”, ”!”
2. Subword Tokenization: This method divides text into smaller units, like subwords or character n-grams, rather than treating each word as a separate token. This method is effective for dealing with words that are not commonly used or languages with complicated morphology. Byte-Pair Encoding (BPE) and SentencePiece are two common subword tokenization techniques.
3. Character tokenization: In this technique, every character in the text is turned into a distinct token. When working with languages where words are not clearly separated or when doing tasks requiring fine-grained analysis, character tokenization is helpful.

Dense Layer :

A crucial sort of layer in the context of neural networks and deep learning is a dense layer, also known as a completely linked layer, which links every neuron in the layer below to every neuron in the layer above. The subsequent layers' neurons are intricately connected to one another.
Every neuron in the layer beneath it contributes information to the neurons in the layer above it. After calculating a weighted total of the inputs to determine each neuron's output, an activation function is used. Since these weights are acquired during training, the dense layer can detect complex connections and patterns in the data. An illustration of the result of the dense layer is as follows:

output = activation(W*input+b)

Here,
input = the previous layer's input vector.
W = The weights connecting the neurons from the previous layer to the current layer are contained in the weight matrix. The weights connected to each row of the weight matrix correspond to a specific neuron in the current layer.
b = The weighted total is increased by the bias vector.
activation = In order to incorporate non-linearity into the network, the activation function is element-wise applied to the weighted sum. Rectified Linear Unit (ReLU), sigmoid, and tanh are frequently used activation functions.

Dense layers play a key role in deep neural networks because they may replicate intricate relationships between input data and objective variables. The ability to learn hierarchical representations of the data is provided to the network as input flows through successive layers, which enables it to extract high-level characteristics and generate predictions while learning these features.
In a typical deep learning model, a number of dense layers are stacked together to form an architecture known as a multi-layer perceptron (MLP). The output of each

thick layer is fed into the input of the layer below it, allowing the network to learn increasingly abstract and complex representations of the data.

BERT Tokenization :

Understanding the nuances of human language is a challenging task in the field of natural language processing. However, advances in deep learning have produced ground-breaking models that excel in language understanding tasks, such as BERT (Bidirectional Encoder Representations from Transformers). The advanced tokenization method used by BERT is the key to its success. Language modeling's core is tokenization, the division of text into smaller pieces. In the case of BERT, tokenization extends beyond just words and includes subword units, allowing it to collect fine-grained data and address the problems presented by words that are not commonly used. The fundamental mechanisms that enable this model to interpret and handle natural language at a remarkable level of sophistication are revealed by probing the depths of BERT tokenization.

BERT tokenization is a sophisticated method that divides text into manageable chunks, enabling the model to successfully comprehend and analyze natural language. By breaking down words into subwords via WordPiece tokenization, BERT is able to capture morphological variants and handle words that are not commonly used. BERT is able to understand linguistic nuance thanks to the fine-grained information provided by these subwords. Text is mapped to the model's vocabulary, which is made up of distinct tokens with individual IDs, during the tokenization process. The model can extract richer meaning from text thanks to BERT's tokenization method, which fills the gap between simple language understanding and raw text.

BERT adds unique tokens that improve the model's capacity for language comprehension. The [CLS] token, which denotes the start of the input sequence, aids BERT in comprehending the overall context. To distinguish various sentences inside the input, utilize the [SEP] token. BERT is able to recognize the relationships between sentences, carry out tasks like text classification and question answering, and keep a cohesive understanding of the input text thanks to the use of these specific tokens and positional embeddings.

To distinguish between distinct sentences in the input, BERT makes use of segment embeddings. A segment ID identifying each token's involvement in a phrase is assigned to it. By identifying the connections and dependencies between sentences and recording contextual information throughout the text, BERT is able to comprehend the relationships and dependencies between sentences. BERT can now perform tasks requiring many sentences, including document categorization or inferring natural language, by incorporating segment embeddings.

In order for BERT to maintain positional information inside the input sequence, position embeddings are essential. BERT recognizes the text's sequential structure by taking the order of the tokens into account. Each token is given a distinct position ID thanks to position embeddings, which also preserve the context of the input and enable BERT to capture the sequential dependencies between words. For BERT to accurately represent the input text and provide predictions based on the token sequence, it has to be aware of its position.

For effective processing, BERT requires fixed-length input sequences. A sequence needs to be trimmed if it exceeds the maximum length allowed in order to fit. Tokens towards the end of the sequence are removed during truncation, but the most important data is kept. On the other hand, if a sequence is too short, it needs to be padded with unique [PAD] tokens to make it the desired length. By ensuring homogeneity in the input size, padding enables BERT to process batches of sequences at once and retain accurate calculations for various inputs.

BERT uses masked language modeling during pre-training to learn contextual representations. A certain proportion of the input tokens are in this procedure randomly masked, or replaced, with a [MASK] token. Then, based on the context, the model is trained to forecast what these masked tokens' original values will be. By using the context to the left and right of the masked token, this method enables BERT to comprehend the bidirectional links between words. By using masking, BERT is better able to interpret words and capture contextual information.

BERT Encoder :

The core element of the BERT model that is in charge of encoding the contextual representations of input text is the BERT (Bidirectional Encoder Representations from Transformers) encoder. It is intended to alleviate the drawbacks of conventional language models that only partially or unidirectionally parse text. The BERT encoder provides bidirectional learning by utilizing a transformer-based design, allowing the model to access both preceding and subsequent words while creating representations for each token.
Multiple layers make up the BERT encoder, which are frequently referred to as transformer encoder layers. A position-wise feed-forward neural network and a multi-head self-attention mechanism are the two sub-layers that make up each layer. Together, these sub-layers capture the relationships and contextual information included in the incoming text.
The BERT encoder can evaluate the significance of various tokens based on their context thanks to the self-attention mechanism, a crucial component of the transformer architecture. By giving relevant tokens larger attention weights and less important ones lower weights, it enables the model to understand the relationships between words. The self-attention mechanism captures long-range dependencies and the overall context of the input text by taking into account the full sequence of tokens.
The BERT encoder includes position-wise feed-forward neural networks in addition to self-attention. The output of the self-attention layer is processed by these networks, which then use non-linear transformations to improve the representations of specific tokens. The BERT encoder can recognize intricate patterns and connections within the input text because to the feed-forward networks, which are made up of numerous completely connected layers with a nonlinear activation function in between.
The BERT encoder hierarchically improves the representations of each token across the levels of the transformer encoder. Each token's meaning is captured within the

context of the full input sequence as the first word embeddings gradually develop into rich, contextualized representations. As they capture the semantic and syntactic links required for precise predictions, these contextualized representations are essential for downstream tasks like text classification.
The BERT model has accomplished exceptional success in a variety of natural language processing applications by utilizing the strength of the BERT encoder. In tasks including sentiment analysis, question-answering, and named entity identification, it has shown cutting-edge performance. The BERT encoder's contextual representations give the model the ability to comprehend linguistic complexity, accurately capture word dependencies, and produce context-based predictions.
In conclusion, to extract the contextual representations of input text, the BERT encoder employs a transformer-based architecture with self-attention and position-wise feed-forward neural networks. Bidirectional learning is made possible by it, enabling the model to comprehend the connections between tokens and produce detailed representations that produce precise predictions for a variety of language understanding tasks.

3.1.5 Data Analysis and Visualization

Analyzing the dataset is needed because we are implementing a big dataset for our research. Our dataset had around 72 thousand data consisting of both fake and real news. The data set had 4 columns and out of these 4 columns the data type of the 'Label' and 'Unnamed: 0' column was int64 and 'text', 'title' was object data datatype. The proportion of fake and real news was 48.56% and 51.44% respectively in the data-set. Here 0 means fake and 1 means real. When cleaning

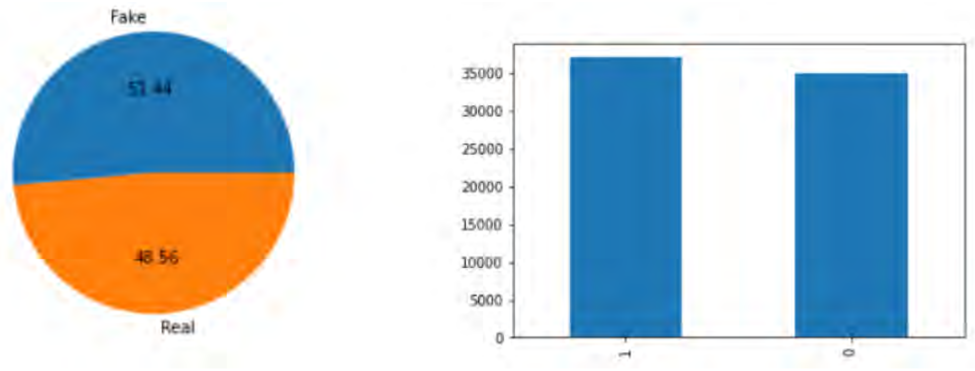


Figure 3.2: Proportion of Real and Fake news

the data, we looked to see if the dataset had any null data of any type. We removed 558 null data records during the data cleaning process. In the dataset, we also looked for duplicate information. The dataset did not contain any duplicate data. We also removed redundant columns from the dataset, such as" Unnamed: 0".

For a clearer understanding of the dataset alongside the advantages of pre-processing, we further divided this data evaluation process into two segments: before pre-processing visualization and after pre-processing visualization.

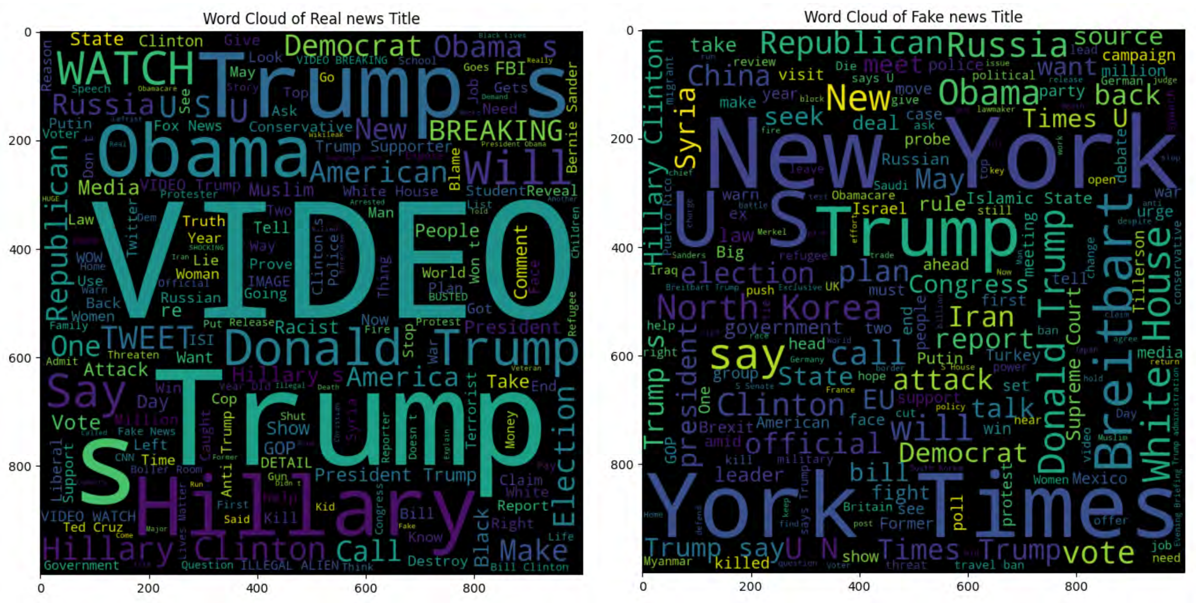


Figure 3.3: Word cloud of Real and Fake news Title Before Cleaning the data

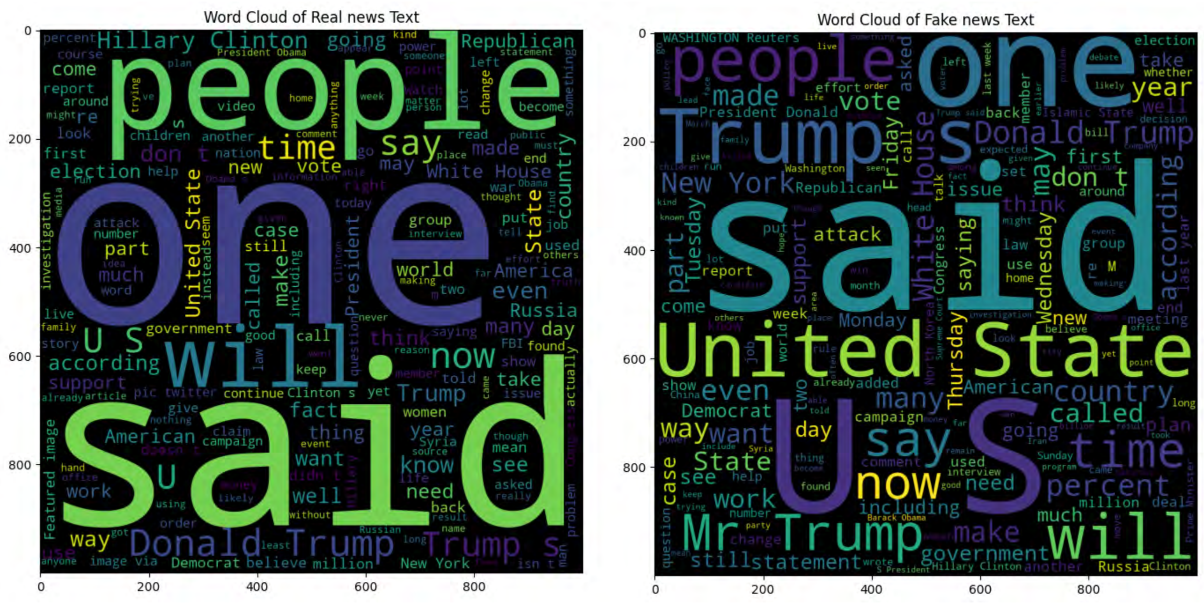


Figure 3.4: Word cloud of Real and Fake news Text Before Cleaning the data

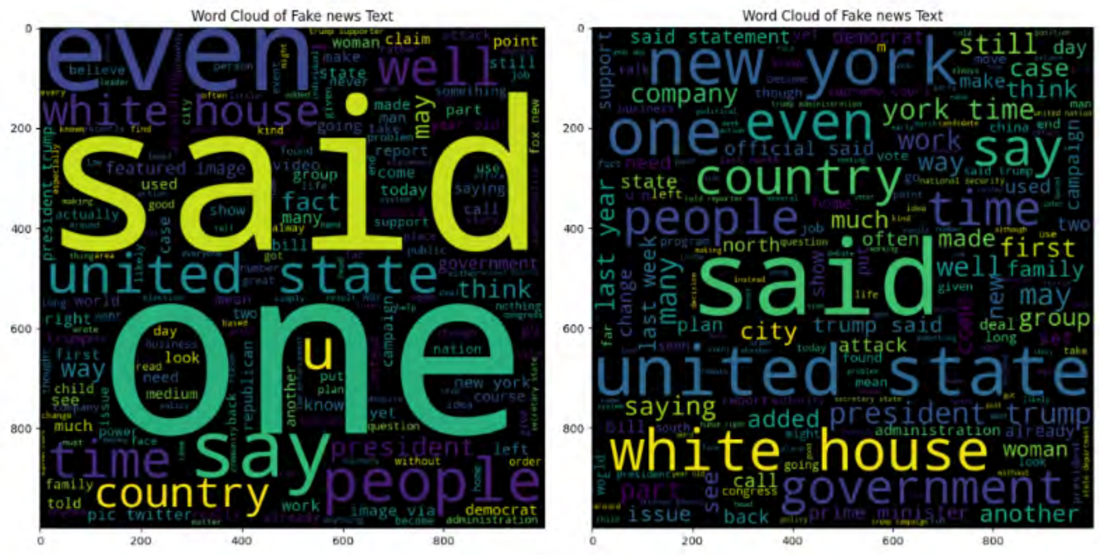


Figure 3.5: Word cloud of Real and Fake news Text Before Cleaning the data

After merging the news's title and body content, we cleaned and processed them to create this word cloud.

Frequent words in the dataset :

Additionally, we utilized some data visualization to determine the words or phrases that appeared the most frequently in the fake news and actual news datasets both for the title and the text. The two results are provided below:
Afterwards, using TF-IDF, we attempted to determine the top 20-word frequencies in both the fake and real news texts in the dataset. We know, significant words or phrases in any article can be found using TF-IDF based on term frequency. This method assists in separating common words from precise and meaningful ones, helping in locating the most pertinent and important words within a paper. A word's term frequency (TF) within a document is defined as the number of times it appears in that document. It is considered that a word with higher term frequency is more important or relevant to the text document as it appears more times in the text document. The inverse document frequency (IDF) takes consideration of a word's significance throughout the whole document. Based on their relevance to the text itself and the lack of them in the dataset we tried to identify significant words or keywords inside the dataset.

From the fake news texts, we initialized the TF-IDF Vectorizer and then the texts were fitted and transformed. Afterwards, we get the feature names to calculate TF-IDF score for each word. Finally, we retrieved the top 20 words by sorting them according to their TF-IDF score in descending order. The following two outcomes are given:

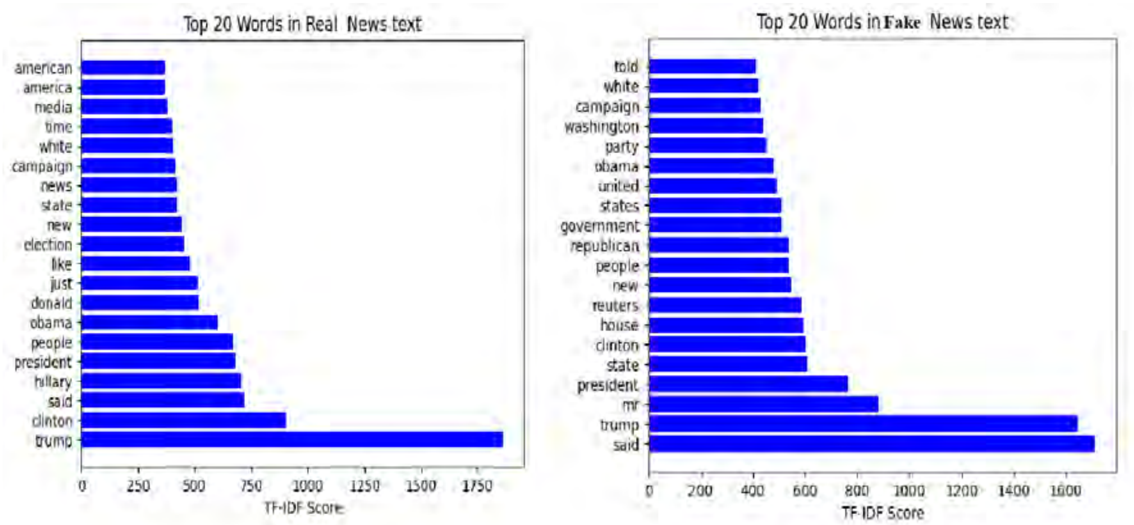


Figure 3.6: Top 20 Words of Fake and Real news Text (Before pre-processing)

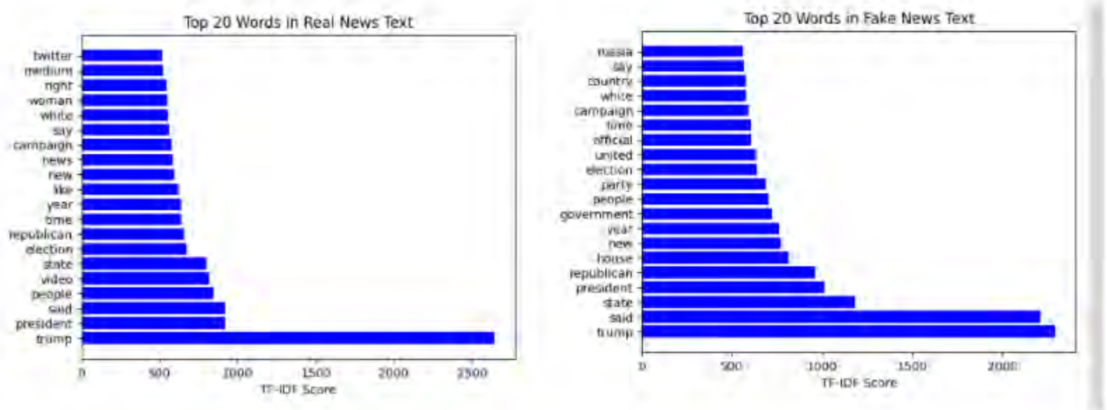


Figure 3.7: Top 20 Words of Fake and Real news Text (After pre-processing)

In the end, to figure out the sequence of words that frequently showed up together in the dataset, we implemented TF-IDF on bigrams and trigrams of words for better visualization. This is necessary so that we can differentiate between fake and appropriate news as we are aware that word sequences convey meaning more accurately than a single word itself does.

The terms "bigram" and "trigram" represent word sequences of two and three respectively. In this instance, we create the TF-IDF vectorizer of length two for bigrams and length three for trigrams. After that, we implemented TF-IDF as earlier of this analysis and obtained the top 20 bunch of word sequences for bigrams along with trigrams. The outcomes:

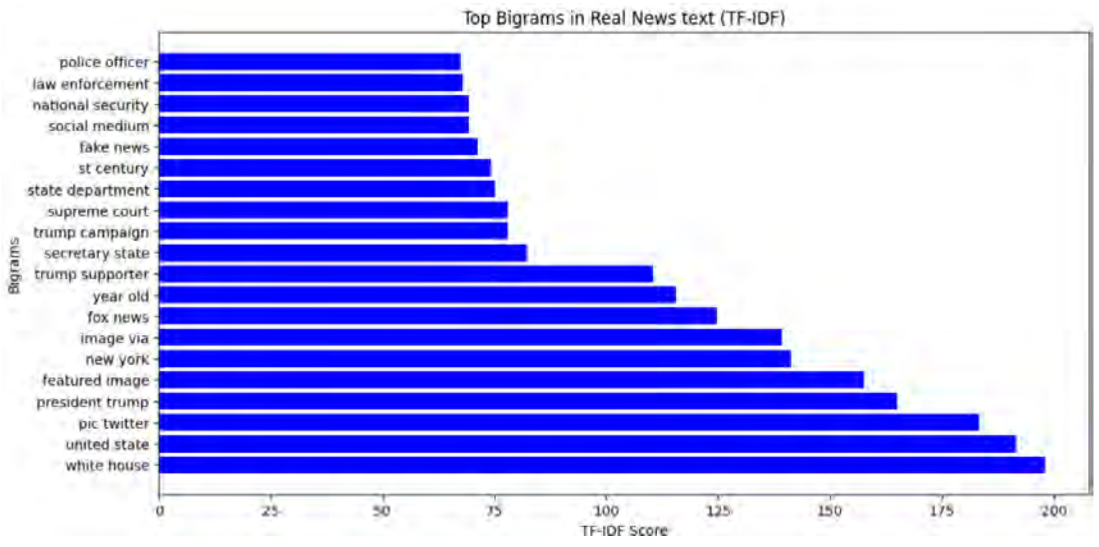


Figure 3.8: Top 20 Bigrams of Real news Text

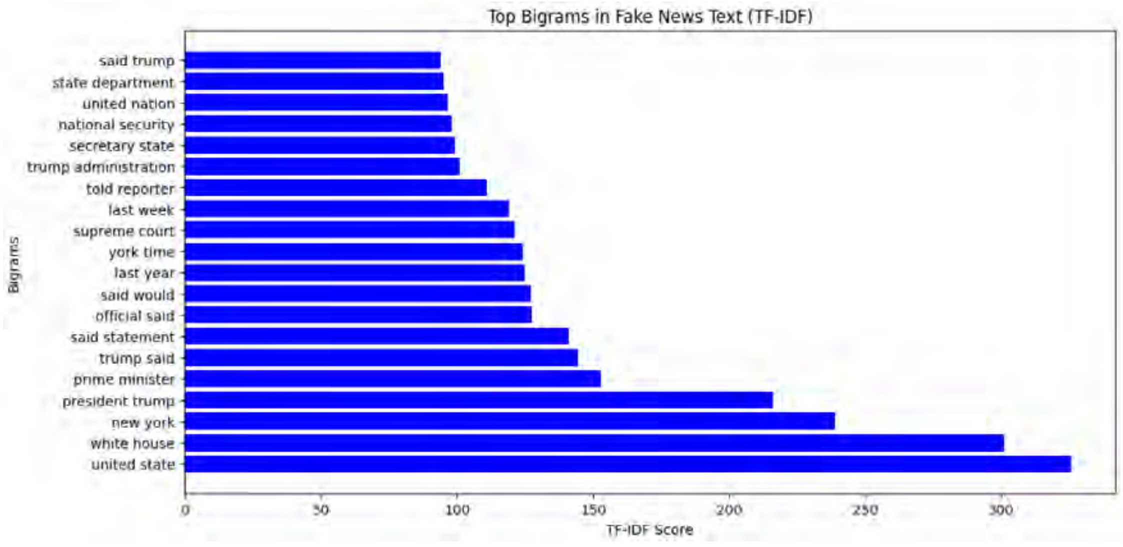


Figure 3.9: Top 20 Bigrams of Fake news Text

3.2 For Convolutional Neural Network :

3.2.1 Dataset Description :

The model has been trained using the well-known and frequently used CASIA dataset. It is a collection of photographs that was especially selected for the purpose of countering spoofing, which seeks to differentiate between authentic images and those produced using various spoofing methods.

The CASIA collection consists of up of a significant amount of photos, both actual (authentic) and spoof (false) samples. The authentic examples are taken from real people in a variety of stances, expressions, and lighting situations to produce a wide and accurate collection of photographs from everyday life.

Several spoofing methods and approaches, including printed pictures, video replays, and 3D masks, are used to produce the spoofed samples. These expertly constructed spoof examples test the model's ability to distinguish between actual and phony images by simulating real-world situations.

The CASIA dataset offers a complete and realistic collection of photos, making it appropriate for anti-spoofing model training and evaluation. The model can be trained on this dataset to identify and generalize features that distinguish actual photographs from fake images, improving the model's capacity to identify and categorize unfamiliar images in practical applications.

We got a total number of 12616 images where 80 percent of these will be used to train the model.

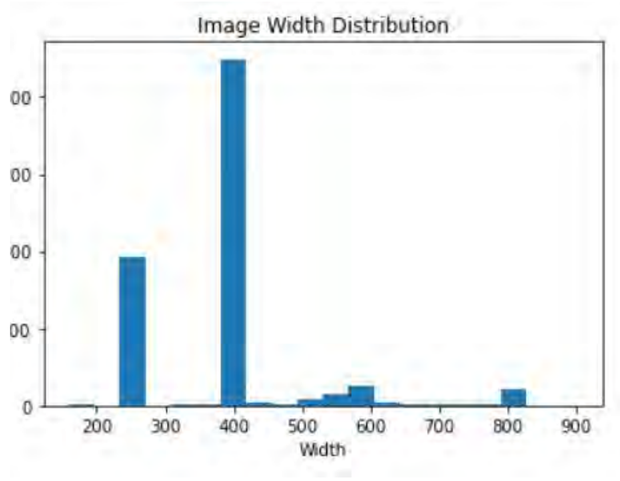


Image Width:

-Minimum: 160

-Maximum:900

-Average: 379.5425618710906

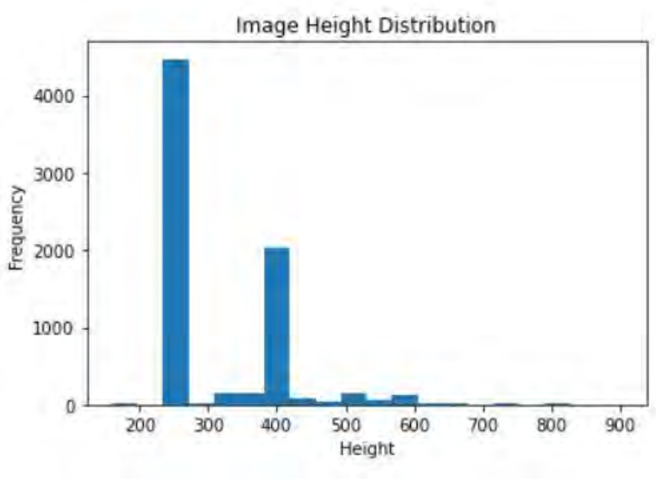


Image Height:

-Minimum: 160

-Maximum: 901

-Average: 315.2483002447648

Image Channels:

- Most common: 3

- Count: 1

3.2.2 Data Pre -processing :

Before training the model, the data was preprocessed using the following steps:

1. Normalization : The pixel values of the input images were normalized to a range between 0 and 1. This ensures that all features have a similar scale and prevents any particular feature from dominating the learning process.

2. Resizing : The images in the CASIA dataset were resized to a uniform size of 128x128 pixels. Resizing the images to a consistent size allows for efficient processing and ensures that all images have the same dimensions, which is required by the model architecture.

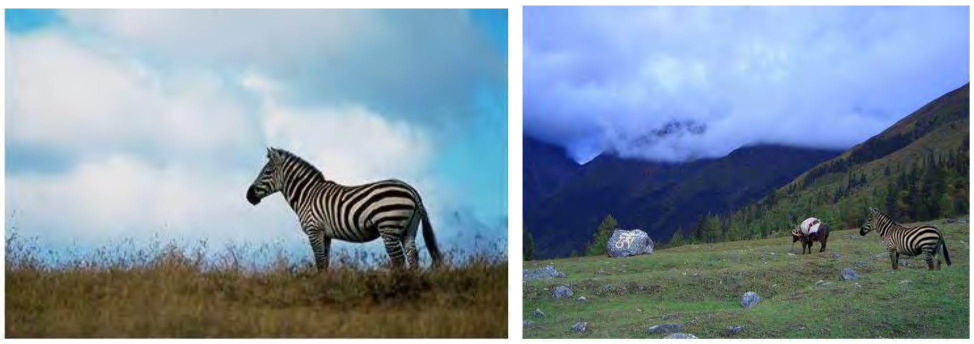
3. Data Split : The dataset was split into training and validation sets. The training set was used to train the model, while the validation set was used to evaluate the model's performance and make adjustments if needed.

4. Label Encoding : The target labels, indicating whether an image is genuine or spoofed, were encoded into numerical format. For example, the labels may be encoded as 0 for genuine and 1 for spoofed. This encoding allows the model to understand and learn from the target variable during training.

These preprocessing steps help to prepare the data for effective training of the model. By normalizing the pixel values, resizing the images, splitting the dataset, and encoding the labels, the data is made suitable for input into the model and facilitates the learning process.

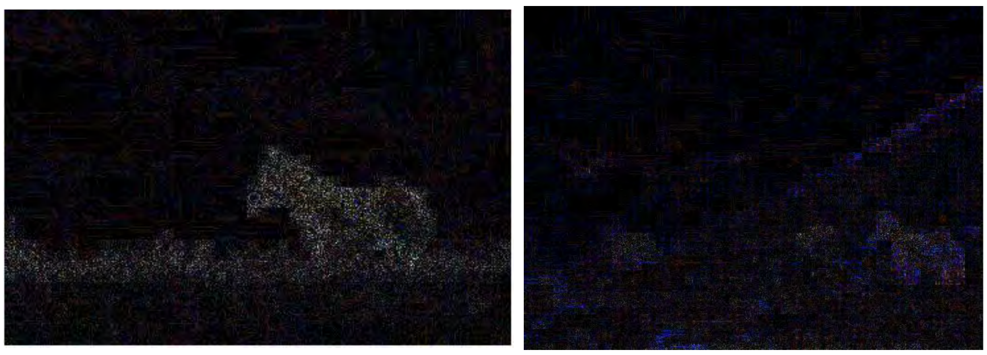
3.2.3 Model Description :

1. Convolutional Layers: The model includes two convolutional layers that perform feature extraction on the input images. Each layer applies a set of 32 filters to the input, with each filter learning to detect different patterns or features in the image. The filters slide over the input image, performing element-wise multiplications and summing up the results to create feature maps. The use of convolutional layers allows the model to capture spatial relationships and local patterns in the images.



Real Image Fake/Edited Images

Figure 3.10: Before converting to ELA images



Real Images Fake/Edited Images

Figure 3.11: After converting to ELA images

2. Max Pooling Layer: Following the convolutional layers, a max pooling layer is introduced to reduce the spatial dimensions of the feature maps. Max pooling divides the feature maps into non-overlapping regions and outputs the maximum value within each region. This downsampling operation helps to retain the most important features while reducing the computational complexity and extracting the dominant features from the previous layers.

3. Dropout Layer: To prevent overfitting, a dropout layer is added after the max pooling layer. Dropout randomly sets a fraction of the input units (in this case, 25%) to zero during each training iteration. By doing so, dropout helps to reduce the interdependency between neurons, forcing the model to learn more robust and generalized features. It acts as a form of regularization, making the model less sensitive to specific patterns and reducing the chances of overfitting.

4. Flattening Layer: The output from the previous layers is then flattened into a 1-dimensional vector. This flattening operation reshapes the multi-dimensional feature maps into a single vector, preserving the learned features while discarding spatial information. Flattening prepares the data for the fully connected layers, which require input in the form of a vector rather than a matrix.

5. Dense Layers: Following the flattening layer, two dense layers are added to perform classification based on the learned features. The first dense layer consists of 256 units and applies the rectified linear unit (ReLU) activation function.

ReLU introduces non-linearity to the model, allowing it to learn complex relationships between features. The second dense layer, with 2 units, serves as the output layer for binary classification. It applies the sigmoid activation function, producing probabilities for each class (fake and real) independently.

Each layer in the model plays a crucial role in the overall architecture. The convolutional layers extract relevant features from the input images, the max pooling layer reduces spatial dimensions, the dropout layer prevents overfitting, the flattening layer reshapes the data, and the dense layers perform the final classification.

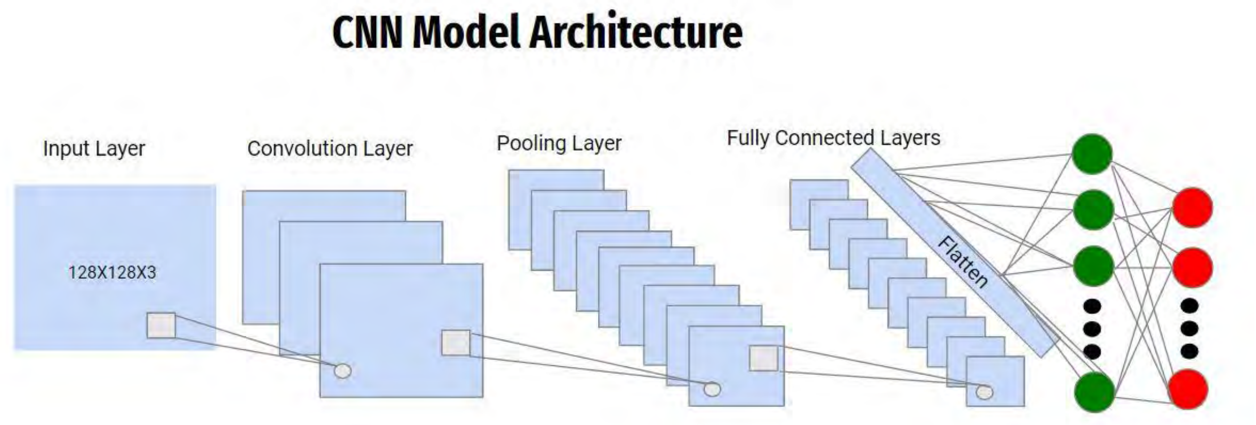


Figure 3.12: CNN Model

Chapter 4

Description of the Models

To build the whole model we have applied algorithms from three different techniques. These subgroups include Machine learning algorithms, RNN algorithms, CNN algorithm. These are described below :

4.1 Machine Learning Models :

Total four number of machine learning models will be described here that includes Support Vector Machine (SVM), Multinomial Naive Bayes, Logistic Regression, Random Forest.

4.1.1 Support Vector Machine (SVM):

Support Vector Machine (SVM) classification is the technique we frst employed for training. An example of a machine learning algorithm is SVM, which was inspired by statistical learning theory. Although Support Vector Machines are often thought of as a classification method, they can be used to solve both classification and regression issues. It can deal with a variety of categorical and continuous variables with ease. In order to distinguish between several classes, SVM creates a hyperplane in multidimensional space. Error minimization is accomplished through SVM, which iteratively constructs the best hyperplane. Finding a maximum marginal hyperplane (MMH) that most effectively categorizes the dataset into classes is the central goal of SVM. These methods are advantageous over Naive Bayes algorithm in that they predict outcomes more quickly and with higher accuracy. As a result of only using a portion of the training points during the choice phase, they also consume less memory. When the margin of separation is distinct and the space is large, SVM performs effectively. Here is a detailed description of how SVM functions:

Working Principle :

1. Data Preparation: To start, you must format your data appropriately. A feature vector should be used to represent each data point, with each feature standing for a different property or feature of the data.

2. Class Labeling: Give each data point a class label. The most typical situation in SVM is binary classification, where each data point is given either a positive (+1) or negative (-1) label. SVM can be expanded to handle multi-class problems utilizing strategies like one-vs-one or one-vs-all.

3. Hyperplane Selection: The primary objective of SVM is to choose the appropriate hyperplane for separating the data points of various classes. In the feature space, the hyperplane serves as a decision boundary. The hyperplane for a binary classification problem is either a line in two dimensions or a hyperplane in higher dimensions.

4. Maximizing the Margin : The margin, or the distance between the hyperplane and the nearest data points from each class, also known as the support vectors, is what SVM seeks to maximize. For accurate generalization of unknown data, the margin is essential. The most ideal hyperplane is that which maximizes the margin.

5. Margin Calculation : The margin is calculated by locating the nearest data points from each class and figuring out how far away from the hyperplane they are. Data points that are outside of the margin or that otherwise defy the margin constraint are known as support vectors.

6. Solving the Optimization Problem : Using SVM, the issue is framed as an optimization task that needs to be solved. The goal is to reduce misclassification error while increasing margin. Solving a convex optimization problem that requires minimizing a cost function while meeting certain constraints allows for this to be accomplished.

7. Kernel Trick: The kernel trick is a technique that SVM can use when the data cannot be separated linearly. The original data is changed by the kernel function into a higher-dimensional feature space, where it might be linearly separable. After that, an ideal hyperplane is discovered using the altered data. The following kernel functions are frequently used: linear, polynomial, sigmoid, and radial basis function (RBF).

8. Training and Testing: After being trained on the labeled data, the SVM model may be used to forecast the class labels of omitted data points. The trained model determines the position of fresh data points in relation to the hyperplane and labels them with the appropriate class.

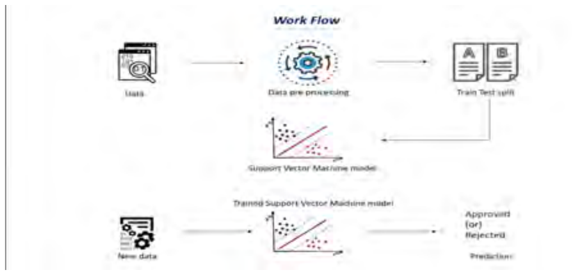


Figure 4.1: SVM workflow

4.1.2 Multinomial Naive Bayes :

Multinomial naive Bayes was the second technique we employed. Depending on the Bayes rule, the Multinomial Naive Bayes approach predicts the label of a text, like an e - mail or newspaper article. For a specific sample, it calculates the likelihood of each tag, then outputs the tag with the highest likelihood. The advantage of using this method is that it simply necessitates the calculation of probability, making it easy to put into practice. This method works for both continuous and discrete data. P(A—B) is calculated as P(A) * P(B—A)/P. (B) In this case, we are determining probability A given that B is known. Prior Probability of B = P(B) Prior probability of class A is P(A). P(B—A) = likelihood that predictor B will occur given class A probability. The steps listed below can be used to summarize the Multinomial Naive Bayes' basic operation:

Working Principle :

1. Data Preparation: To prepare the training data, each document is represented as a feature vector. In text categorization, the features often match the word counts or frequencies in the text. The document classes are represented by the target variable.

2. Feature Probability Calculation: The algorithm determines the probability distribution of each feature (word) conditioned on each class in the training set. This is accomplished by calculating the number of times each word appears in the documents belonging to a given class, then dividing that number by the total number of words in that class.

3. Class Prior Probability: The prior probability of each class is calculated by adding up all the documents in the class and dividing that sum by the number of documents in the class.

4. Conditional Probability Calculation : When a new document needs to be classified, the method determines the conditional probability that the document belongs to each class. The probabilities of the characteristics (word probabilities) for each class are multiplied in this manner, assuming independence between the features (words). Numerical underflow is frequently prevented by using the logarithm of probability.

5. Class Prediction: For the new document, the predicted class is chosen to be the one with the highest conditional probability. The algorithm places the document in the class with the highest likelihood based on the observed feature values.

In applications like spam detection, emotion analysis, and topic classification where word frequencies are important, multinomial Naive Bayes is frequently utilized. When working with feature spaces that have a lot of dimensions and training datasets that aren't very big, it is particularly appropriate. Multinomial Naive Bayes' Theorem, which is probabilistic in nature, is used to: Based on word frequencies and the likelihood of each class given the observed attributes, Bayes offers a quick and efficient method for classifying text texts.

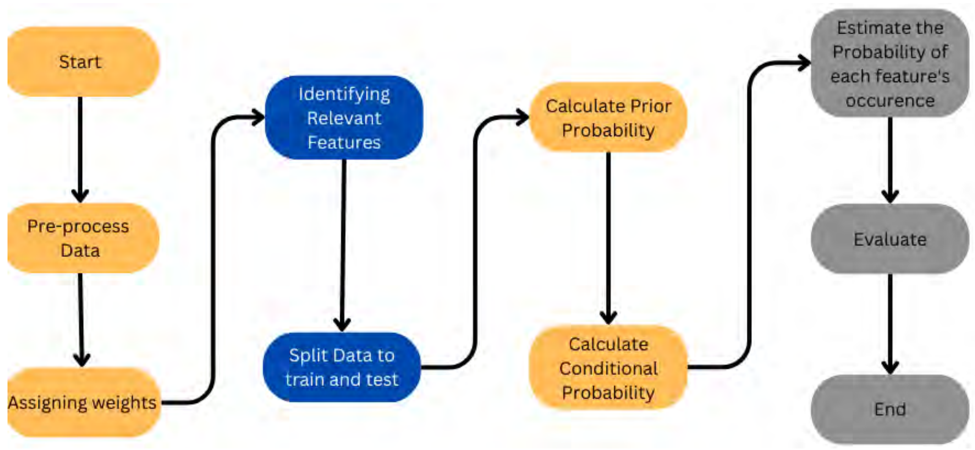

Start
Pre-process Data
Assigning weights
Identifying Relevant Features
Split Data to train and test
Calculate Prior Probability
Calculate Conditional Probability
Estimate the Probability of each feature's occurence
Evaluate
End

Figure 4.2: Multinomial Naive Bayes workflow

4.1.3 Logistic Regression :

Logistic regression is a statistical algorithm that is used to classify data. It predicts the outcome as binary depending on an independent variable set. A flexible and popular approach for binary classification applications is logistic regression. It handles both numerical and category information, provides findings that are easy to understand, and calculates class probabilities.

A logistic regression model makes predictions about a dependent data variable by investigating the correlation between one or more independent variables that are already present.

1. Data Preparation : To prepare the training data, feature values (input variables) and categorical target labels or similar binary values are collected. A feature vector and an associated target label serve as representations for each instance or observation in the collection.

2. Logistic Function (Sigmoid Activation) : The logistic function, also called the sigmoid function, is used in logistic regression to translate the input features into a probability score. The logistic function transforms the input characteristics into a linear combination before compressing the values between 0 and 1.The logistic function is defined as :

$$f(x) = \frac{1}{1 + e^{-x}}$$

3. Hypothesis Function : The logistic regression model specifies a hypothesis function that converts the properties of the input data into the projected probabilities of the target variable. Logistic regression's hypothesis suggests that the cost function should be constrained to a range between 0 and 1. In light of the fact that it can have a value larger than 1 or less than 0, which is not feasible according to the logistic regression hypothesis, linear functions are therefore unable to accurately describe it.The hypothesis function can be defined as :

$$h_\theta(x) = \sigma(\theta^T x)$$
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$h_\Theta(x)$ = the predicted output

x, Θ are the model parameters (coefficients or weights)

$z = \Theta^T x$

4. Model Training (Parameter Estimation): A procedure known as maximum likelihood estimation is used to estimate the model parameters . Finding the parameters that maximize the likelihood of the observed target labels given the input features is the objective. Usually, a cost function, such as the cross-entropy loss or the negative log-likelihood, is minimized by using an optimization algorithm like gradient descent, which iteratively changes the parameters.

5. Decision Boundary: Following the learning of the model parameters, a decision boundary is established using the predicted probabilities. The decision boundary divides the data points into various classes according to a given threshold probability (for example, 0.5). Instances with predicted probabilities over the threshold are categorized into one class, and those below the threshold are categorized into the opposite class.

6. Model Evaluation: A separate validation or test dataset is used to assess the trained logistic regression model using evaluation metrics including accuracy, precision, recall, and F1. These metrics measure how well the model performs in correctly predicting the class labels for the target variable.

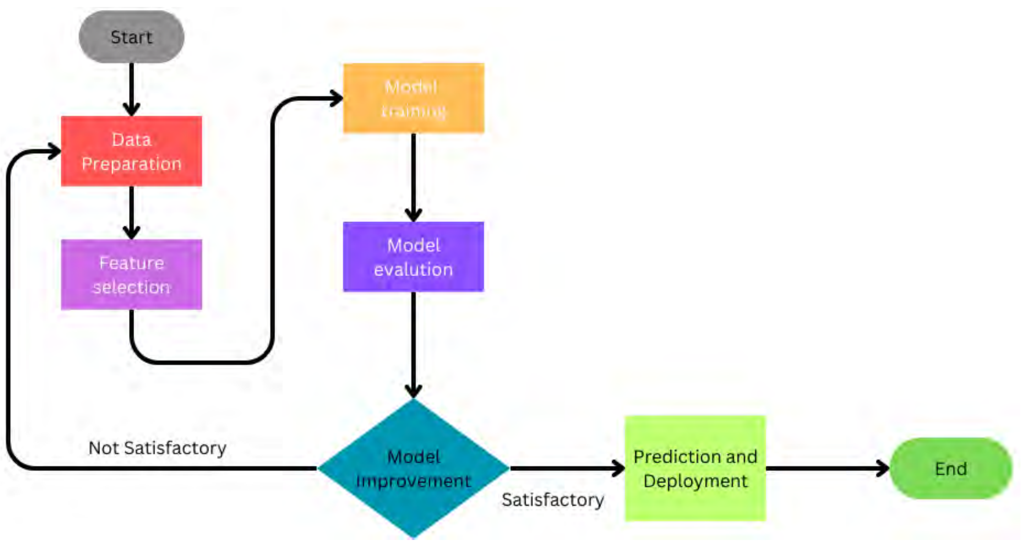


Figure 4.3: Logistic Regression workflow

4.1.4 Random Forest :

A popular machine learning approach called random forest which produces the output combining various decision trees together. Its widespread use is motivated by its adaptability and usability because it can solve classification and regression issues. Key characteristics and working principles of Random Forest :

1. Ensemble of Decision Trees: Random Forest is composed of a combination of decision trees. Each decision tree is individually constructed, and it based its forecasts on the categorization or regression of the average of each of its individual tree forecasts.

2. Random Sampling: Random Forest employs a technique called bootstrapping, which comprises randomly sampling the training dataset with replacement. Each decision tree in the ensemble is trained using a different bootstrapped sample of the data to add diversity to the training process.

3. Random selection features : Random Forest selects a subset of features at random for each split point in a decision tree from the available features. This ensures that different trees focus on different feature subsets and reduces the link between trees. The number of features to consider at each split point is typically the square root or a user-specified value.

4. Construction of decision trees : Using a recursive process, each decision tree in the Random Forest is constructed by selecting the best split at each node in accordance with a predetermined criterion, such as information gain or Gini impurity. The splitting procedure goes on until a preventing condition is met, such as when the maximum tree depth or the minimum number of samples are achieved at a node.

5. Prediction aggregation: Once all the decision trees have been trained, predictions are made by merging the individual forecasts of each tree. The class that earned the most votes among the decision trees is the final prediction in classification problems. In regression problems, the average of the individual tree forecasts is used to get the final prediction.

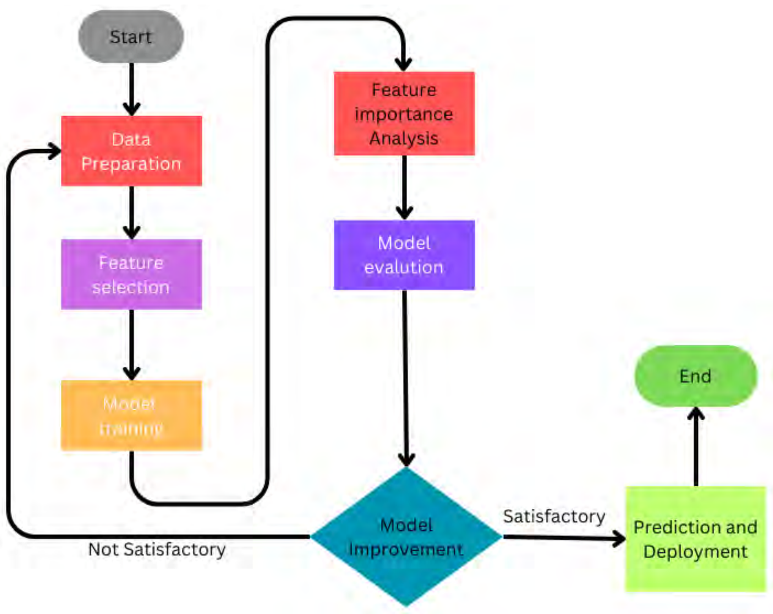


Figure 4.4: Random Forest workflow

4.2 Recurrent Neural Network :

All total three different algorithms has been used for Recurrent Neural Network. These are Bi_LSTM ,Bi_GRU, BERT (state of the art).

4.2.1 Bidirectional Long Short Term Memory (Bi_LSTM)

By accurately preserving and deleting data over a lengthy period of time, recurrent neural networks (RNNs) with Long Short-Term Memory (LSTM) can handle long-term dependencies. The LSTM design combines the three basic types of gates known as the input gate, forget gate, and output gate. The input gate is active for the whole data stream. While the input gate selects how much of the current input should be added to the cell state, the forget gate selects which fraction of the previous state should be maintained. The output gate will select how much of the cell state will be used for output since it makes the final forecast. A sigmoid function is used by the input gate, forget gate, and output gate to generate values between 0 and 1.

Input gate, $i_t = (W_i[x_t, h_{t-1}] + b_i)$
Forget gate, $f_t = (W_f[x_t, h_{t-1}] + b_f)$
Output gate, $o_t = (W_o[x_t, h_{t-1}] + b_o)$

Here, Weight matrices W_i, W_f and W_o bias vectors b_i, b_f and b_o the input at time t, the hidden state from the previous time step h_{t-1}.

The hyperbolic tangent (tanh) function is used in the equation for the candidate cell state to provide a vector of potential values that can be added to or removed from the cell state. Candidate cell state, $g_t = \tanh(W_c[x_t, h_{t-1}] + b_c)$

The equations for the current hidden state(h_t)and the current cell state(C_t) update the current cell state and hidden state using values from the gates and the candidate cell state. Current cell state: $C_t = f_t * C_{t-1} + i_t * g_t$
Current hidden state: $h_t = o_t * \tanh(C_t)$
LSTMs were developed to address the vanishing gradient problem that could arise when training standard RNNs

A version of LSTM called Bidirectional Long Short-Term Memory (Bi-LSTM) is frequently employed for the processing of sequential input. It is an update on the traditional LSTM architecture, which works for processing sequences with long-term dependencies. A Bi-LSTM's architecture includes two independent LSTM layers, one of which analyzes input sequences in a forward direction and the other in a backward direction. The final output of the Bi-LSTM is created by concatenating the output of these two layers at each time step. The input sequence of input vectors x = [x1, x2, ...xn], where xi is the value of the input vector at time i, is provided to the forward LSTM layer. The forward LSTM layer uses the input vector and the prior hidden state and cell state to derive a hidden state (ht) and a cell state (Ct) at each time step t. The transmission of information through the cell state is managed by the input gate, forget gate and output gate.

The same series of input vectors x = [x1, x2,..., xn], but in the opposite direction, i.e., [xn, xn-1,.....x1], are used as input by the backward LSTM layer. With the input vector and the prior hidden state and cell state, the backward LSTM layer determines a hidden state (ht') and a cell state (Ct') at each time step t. The input gate, forget gate, and output gate all control how information moves inside the cell state. The outcome of the Bi-LSTM is the sum of the forward and backward hidden states at each time step. Utilizing a Bi-LSTM over a regular LSTM has the benefit of being able to detect both forward and backward dependency in the order of inputs, which can be advantageous for ML projects.

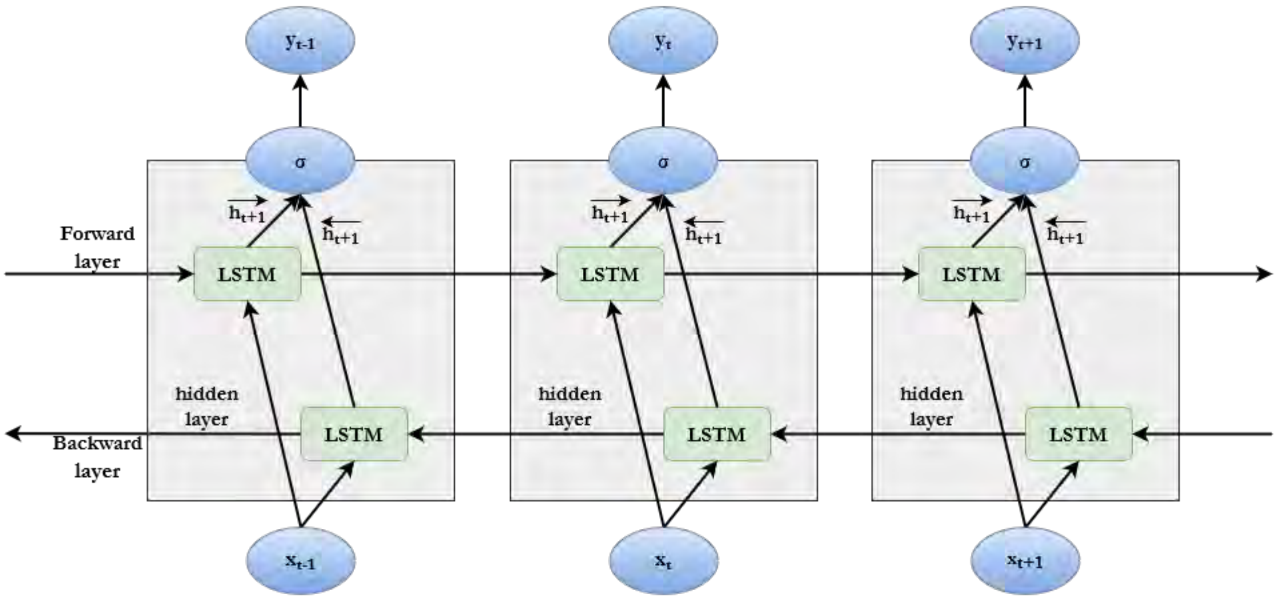


Figure 4.5: Workflow diagram of Bi_LSTM

4.2.2 Bidirectional Gated Recurent Unit (Bi_GRU):

Gated recurrent units (GRU), a type of neural network model, are widely used to process sequential input, much like LSTM. The architecture of a GRU is made up of just one layer of GRU units, which process the input sequences sequentially. Each GRU unit has an update gate and a reset gate that control the flow of data through the unit. The update gate decides the amount of the new input should be absorbed into the present hidden state, while the reset gate defines how part of the prior hidden state should be forgotten. A hidden state vector is the product from every GRU unit, and it is transferred to the following one in the chain. A GRU unit calculates reset gate and the update gate using the prior hidden state and the input vector
. zt = sigmoid(Wz * xt + Uz * ht-1 + bz)
rt = sigmoid(Wr * xt + Ur * ht-1 + br) ;
here Wz, Uz, Wr, Ur, bz, and br are learnable weight matrices and bias vectors and xt is the input vector at time t, ht-1 is the prior hidden state.
The candidate hidden state of the GRU unit, h t = tanh(W * xt + U * (rt * ht-1) + b). Here, where b is a bias vector and W and U are learnable weight matrices.

The candidate hidden state and the previous hidden state are combined linearly to get the final hidden state of the GRU unit, ht = (1 - zt) * ht-1 + zt * h t.

The hidden state sequence that is generated by every GRU unit in the sequence is the output of the GRU layer. Because a GRU has fewer parameters than an LSTM, it can be trained more quickly and with less memory usage. However, it has been demonstrated that LSTMs performed better in some situations, particularly when dealing with longer sequences or intricate interactions among sequence parts.

The Bidirectional Gated Recurrent Unit (Bi-GRU), a variant of the GRU architecture, is designed to process sequential data in both forward and backward directions. Bi-GRU is a type of bidirectional recurrent neural network that is frequently used in sequential data applications, such as NLP and speech recognition.
The Bi-GRU's architecture consists of two GRU layers, one of which processes the input sequence forward and the other of which processes it backward. At each time step, the hidden state vectors of the two layers are concatenated to produce the output vector of the Bi-GRU unit. Every layer of a Bi-GRU unit has the update gate and the reset gate, which control the flow of data across the unit. The internal memory of the unit is represented by the hidden state vector of the layers.

Bi_GRU is ideal for applications like sentiment analysis and sequence labeling because it can record both the previous and prospective context relating to the input sequence. On several comparison datasets, it has been demonstrated that it exceeds other varieties of bidirectional RNNs.

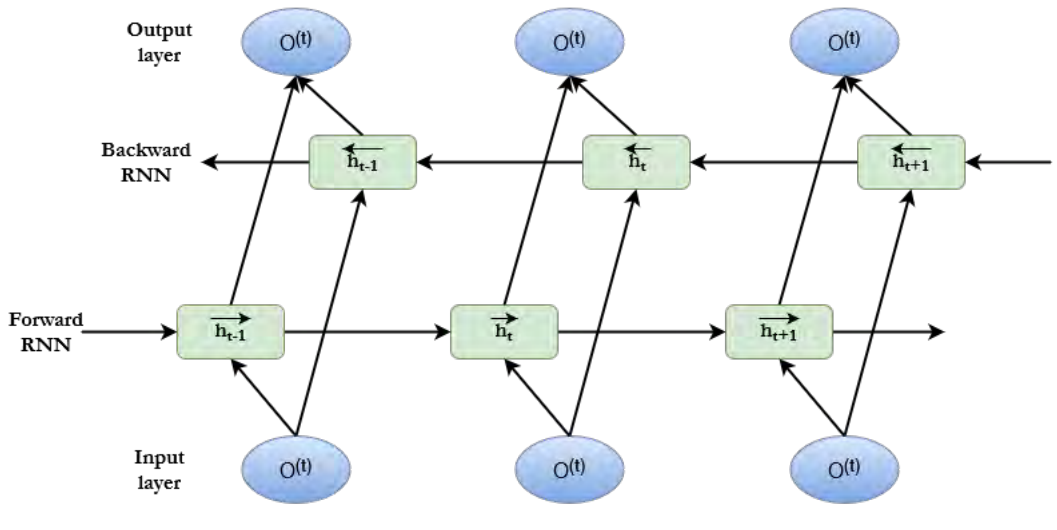


Figure 4.6: Workflow diagram of Bi_GRU

4.2.3 Bidirectional Encoder Representations from Transformers (BERT) :

The Bidirectional Encoder Representations from Transformers (BERT) pre-trained deep learning model is employed in natural language processing applications. The transform structure handles sequential data, including text written in natural language, by using self-attention techniques. With the use of the self-attention mechanism, the model can assess each token while focusing on diverse elements of the input sequence.

Each transformer encoder layer in the BERT system has several heads of self-attention and position-wise fully interconnected feed-forward networks. Before moving on to the following layer, each layer's output is subjected to a layer normalization procedure. A sequence of tokens that are initially transformed into embeddings using an embedding matrix form the input to BERT. An embedding vector in the d-dimensional range, where d is the dimension of the embedding space, is used to represent each token. To represent the meaning of each token in the input sequence, the embedding matrix is learned during pre-training.
Calculating the self-attention scores between all token pairs in the input sequence is the initial step in each transformer encoder layer. The self-attention mechanism calculates a weighted total of the embeddings of all tokens in the input sequence, where the weight given to each token is based on how similar it is to the other tokens in the sequence. The self-attention score, Here, d_k is the dimensionality of

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

the key matrix, Q, K, and V are the query, key, and value matrices, respectively, and softmax is the softmax function that normalizes the attention scores.

The model conducts a weighted sum of the embeddings of all tokens in the input sequence after computing the self-attention scores, with the weights being based on the self-attention scores. The output of the self-attention process is subsequently transmitted via a positionally fully linked feed-forward network, which is made up of two linear transformations separated by a ReLU activation function. Before moving on to the following layer, the output from every transformer encoder layer is normalized. The output from each layer is normalized using the layer normalization approach to have a unit variance and zero mean. This aids in enhancing the model's performance and stabilizing the training process.

Two unsupervised learning tasks: masked language modeling and next sentence prediction—are used to train Bert on vast volumes of text data during pre-training. The model is trained to predict the original tokens based on the masked tokens and the surrounding context in masked language modeling, which randomly masks a particular percentage of the input sequence's tokens. Due to the need to infer the proper token from the surrounding context, this drives the model to learn framed representations of each token in the input sequence.

When predicting the next sentence, the model is trained to determine whether or not two sentences in the input sequence are consecutive. This exercise improves the model's comprehension of the links between sentences. During fine-tuning, a task-specific output layer is placed on top of the previously trained BERT model, and the entire model is trained on the appropriate labeled dataset. During the fine-tuning stage, the parameters of this model's pre-trained version are changed to better suit the task at hand. BERT's cutting-edge performance on a number of NLP tasks has led to its acceptance by various academics and business executives. Numerous further transformer-based variations have been produced as a result of its popularity.

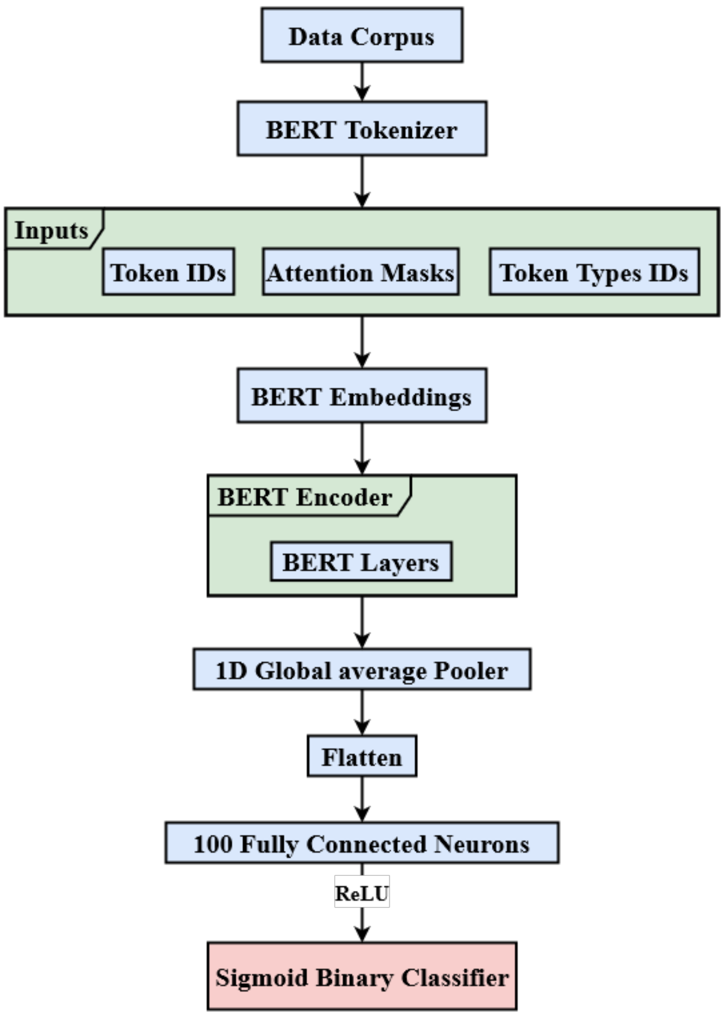


Figure 4.7: Workflow of BERT

4.3 Convolutional Neural Network (CNN) :

For the analysis of visual data a convolutional neural network (CNN) is a popular deep learning approach as CNNs are composed of several layers such as Convolution layer, Activation layer, Pooling layer, Fully connected layer that have the ability to learn and retrieve pertinent characteristics from the input data.

The convolutional layer is the most important component of a CNN. Convolutional kernels or a set of learnable filters are added to the input data in this layer. In order to create a feature map, each filter performs element-wise additions and multiplication with the input. This method hierarchically learns more sophisticated representations while capturing local spatial patterns. In order to introduce non-linearities, an activation function is applied elementwise following each convolutional layer. The sigmoid, tanh and ReLU activation functions are frequently used and among them ReLU is more effective and simpler.

A pooling layer comes after convolution. Utilizing methods like average pooling or maximum pooling, pooling decreases the spatial dimensions of the feature maps. Although average pooling determines a median value within a pooling zone, maximum pooling chooses the maximum value. Downsampling helps to keep coordinates while reducing the amount of computation required and extracting key features.

To improve the network's architecture and enable the model to learn more and more abstract and excellent information, many convolutional and pooling layers can be stacked. Fully connected layers are used to process the learned features in more detail. A vector representation of the feature maps is created, and they are then joined to one or more fully connected layers. Based on the retrieved features, these layers carry out decision-making and high-level analysis. The fully connected layer's neurons individually calculate the weighted total of their inputs, employ an activation function, and generate an output.

Due to the global receptive fields and combined weights utilized by convolutional layers, one of the advantages of CNNs is their capacity to automatically learn pertinent features from unprocessed input data. The local receptive fields make sure that each neuron concentrates on a specific area of the input, while the shared weights enable the network to detect identical patterns in multiple points. These characteristics enable CNNs to handle grid-like patterns and be resilient to changes in input data.

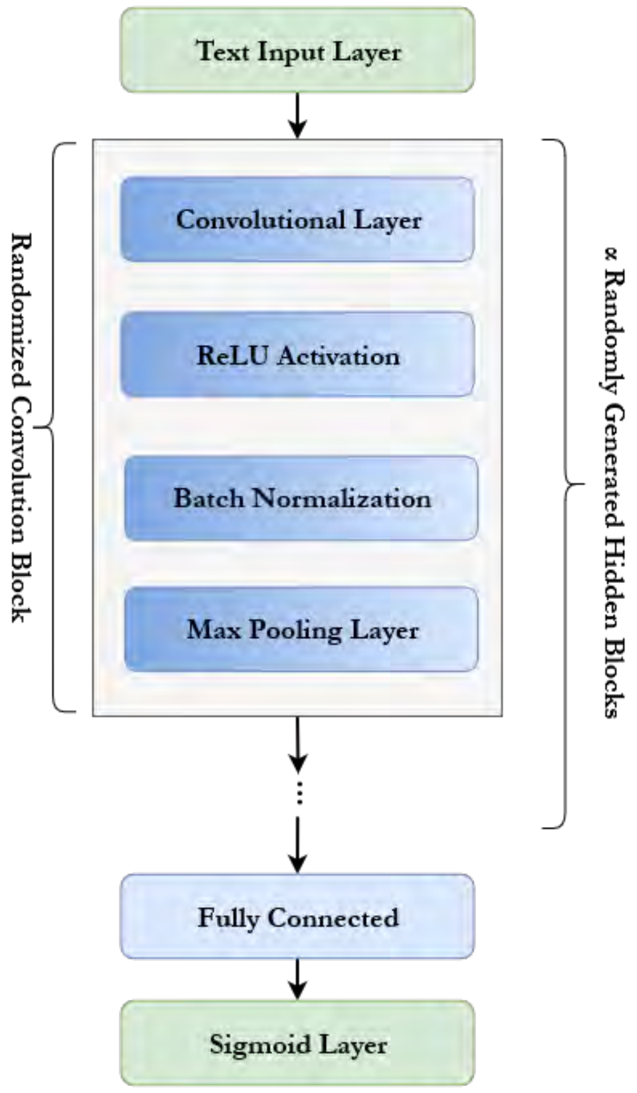


Figure 4.8: Workflow of CNN

Chapter 5

Result Analysis

5.1 Precision, Recall and F1 score

Precision, recall, f1 score, and confusion matrix are just a few of the ways that the models' outputs might be shown. The precision, recall, and F1 score of a classification model are important factors to take into account. They provide informative information on various aspects of the model's success and aid in evaluating the model's performance in certain situations.

True Positives (TP): Occurrences that the model accurately identified as positive. Instances where the model accurately classifies positive outcomes as positive occur in these situations.
True Negatives (TN): Occurrences that the model accurately identified as negative. In these situations, the model accurately classifies negative events as such.
False Positives (FP): Occurrences that the model mistakenly predicts as positive. In some circumstances, the model misclassified negative events as positive ones.
False Negatives (FN): Situations that the model mistakenly predicts as negative. In some circumstances, the model misclassified positive instances as negative ones.
Precision : Out of every single instance that the model predicted to be positive, precision is a measure of the model's ability to properly identify positive instances. The goal is to reduce false positives. A high precision means that there is little chance that the model would mistakenly categorize negative instances as positive. For instance, in a scenario involving a medical diagnosis, precision is the percentage of instances that are accurately diagnosed as positive out of all cases that were projected to be positive.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.1}$$

Recall: The model's capacity to accurately identify all of the positive instances is measured by recall, sometimes referred to as sensitivity or true positive rate. It emphasizes reducing false negatives. With a high recall, the model is less likely to misclassify positive cases as negative. It assesses the model's capacity to correctly detect positive cases and is also known as sensitivity or true positive rate. It is the proportion of accurate positive predictions to all instances of actual positive outcomes (TP + FN). The model's recall tells us how many of the real positive events it was able to identify.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.2}$$

F1 Score: The harmonic mean of recall and precision is the F1 score. It offers a single metric that balances recall and precision into a single measurement. If either precision or recall is much lower than the other, the harmonic mean will give lower values more weight, lowering the F1 score. In other words, rather than favoring one over the other, the F1 score rewards models that have high precision and high recall. The harmonic mean of recall and precision is the F1 score. When there is a trade-off between these two measurements, it offers a balanced measure that takes both precision and recall into account. The F1 score has a range of 0 to 1, with 1 being the highest attainable result.

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.3}$$

5.2 Result of Machine Learning Models

A machine learning model is an algorithmic or mathematical representation that uses input data to identify patterns and generate predictions or judgments. We used machine learning models including SVM, Naive Bayes, Logistic Regression, and Random Forest for our fake and true news detection task. We preprocessed the data before dividing it into train, test, and validation subsets for the model training: 80:10:10. Then, we converted the text data into numerical characteristics that could be utilized as input for machine learning models by using the TF-IDF (Term Frequency-Inverse Document Frequency) algorithm.

We used the linear linear kernel for the support vector machine. For data that can be separated linearly, the linear kernel is frequently employed and is useful in a variety of classification applications. The decision boundary is described as a hyperplane dividing the classes. The validation accuracy for the SVM model was 93.26%, and the test accuracy was 92.58%. Precision was 93.04%, Recall was 93.56%, and F1 score was 93.3% on the validation set. Precision: 93.26%, Recall: 91.76%, and F1 score of 92.68% on the test data.

A classification problem is classified using Multinomial Naive Bayes, which calculates the probability of each class given the feature vector (input). The validation accuracy and test accuracy for the Multinomial Naive Bayes model, respectively, were 84.16% and 83.49%. Precision: 83.91%, Recall: 84.68%, and F1 score of 84.26% were recorded on the validation set. Precision: 83.98%, Recall: 83.73%, and F1 score of 83.85% on the test data.

Additionally, logistic regression classifies a classification issue by simulating the likelihood that a single instance will belong to a certain class. It converts the input features into a probability score using a logistic function, also referred to as the sigmoid function. The test accuracy and validation accuracy for the logistic regression model, respectively, were 91.70% and 92.63%. Precision: 92.35%, Recall: 93.03%, and F1 score of 92.69% were obtained on the validation set. Precision: 92.44%, Recall: 91.25%, and F1 score of 91.84% on the test data.

Last but not least, we employed Random Forest classification, an ensemble learning technique that combines the categorization predictions of various decision trees. The validation accuracy for the Random Forest model was 92.28%, and the test accuracy was 91.30%. Precision, Recall, and F1 scores on the validation set were 92.31%, 92.31%, and 92.31%, respectively. Precision: 91.57%, Recall: 91.41%, and F1 score of 91.49% on the test data.

Model Name	Test	Validation	Precision	Recall	F-1 score
SVM	92.58	93.26	93.26	91.76	92.68
Multinomial Naive Bayes	83.49	84.16	83.98	83.73	83.85
Logistic Regression	91.70	92.63	92.44	91.25	91.84
Random Forest	91.30	92.28	91.57	91.41	91.49

Table 5.1: Machiene Learing models result

Both validation accuracy and test accuracy are essential in determining how well a model performs, but test accuracy is frequently regarded as being more significant because it shows how well a model can generalize to new data. from every model that has been examined. Following logistic regression and random forest classifier, we discover that the Support vector machine model has the highest test accuracy.We also extracted the precision, recall, and f1 score for test and validation. Since it represents the model's genuine ability to generalize to unobserved data, we are only concentrating on the test data. The support vector machine also has the highest f1 score, recall, and precision.

5.3 Result of Recurrent Neural Networks Model

Recurrent neural networks (RNNs) are a subclass of artificial neural networks created with the purpose of processing time-series and sequential data efficiently. For problem-solving, we employed models like Bi-LSTM and Bi-Gru.

We utilized the glove word embedding "glove.6B.300d" for all embedding layers.GloVe (Global Vectors for Word Representation) is a particular pre-trained word embedding model that has been trained on a substantial corpus of text data, and is the model referenced by the glove word embedding "GloVe.6B.300d". The word "6B" in "GloVe.6B.300d" stands for the vocabulary's about 6 billion tokens or words. Each word in the lexicon is represented as a dense vector of 300 dimensions, as indicated by the "300d" designation.

Basic ML models and RNNs are different in the following ways: RNNs deal with sequential data and take temporal dependencies into account. They have a memory inside of them that can store context over time. RNNs are able to generalize to sequences of various lengths because they share weights across time steps. Long-term dependencies and complicated non-linear patterns can be captured by them. Gradient-based optimization approaches are used to train RNNs. Basic ML models lack the ability to model time and assume independence.

Bi LSTM: Recurrent neural networks of the Bi-LSTM variety are employed in NLP tasks. It has two LSTM layers, one of which runs the sequence forward and the other backward. This enables the model to grasp sequential data better by capturing information from both past and future context.
The Bi LSTM layer employed the pre-trained GloVe embeddings. The embedding layer in the model architecture has a 300 embedding dimension and is initialized with pre-trained GloVe embeddings. The embedding layer is configured to be non-trainable, and the input sequences have a maximum length of max_sequence_length = 300.There are two bidirectional LSTM layers in the model. While the second LSTM layer only has 128 units, the first LSTM layer has 256 units and returns sequences. Following both LSTM layers are dropout layers with a 0.2 dropout rate. The dense layer with a sigmoid activation function that makes up the last layer produces a single value for binary classification. The layers and their corresponding configurations, including the quantity of parameters, are displayed in the model summary. A binary cross-entropy loss function and the Adam optimizer are used in the model's construction.

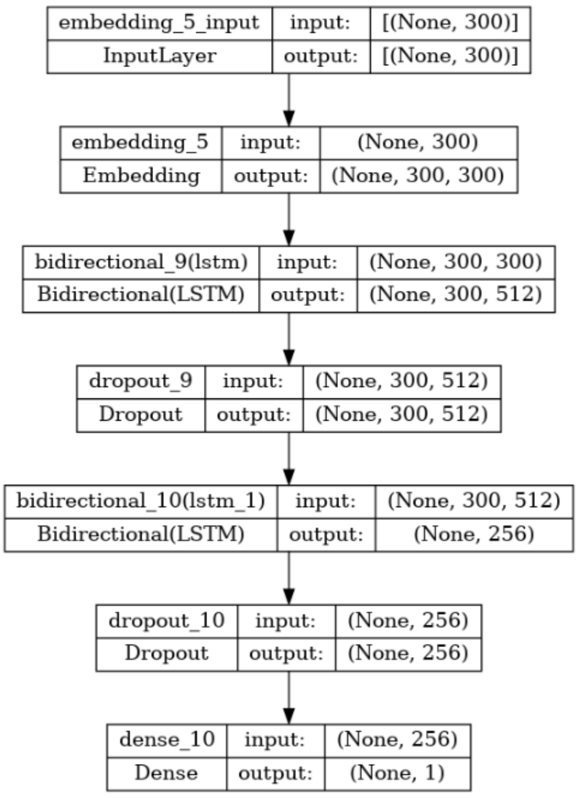


Figure 5.1: Architecture of Bi - LSTM

The input and output shapes of the embedding layer are (None, 300) and (None, 300, 300), respectively. There are 10,453,200 non-trainable parameters in all.The output form of the first bidirectional LSTM layer, which receives the output from the embedding layer, is (None, 300, 512). 1,140,736 parameters make up this.Following the initial LSTM layer is a dropout layer that has no trainable parameters and has no impact on the output's form.The output shape of the second bidirectional LSTM layer, which accepts the output from the first LSTM layer, is (None, 256). There are 656,384 parameters in it.Following the second LSTM layer is a dropout layer that has no trainable parameters and has no impact on the output's form.The dense layer has 257 parameters and an output shape of (None, 1).The model includes 12,250,577 parameters in total. 10,453,200 of these parameters cannot be trained, leaving 1,797,377 trainable parameters.

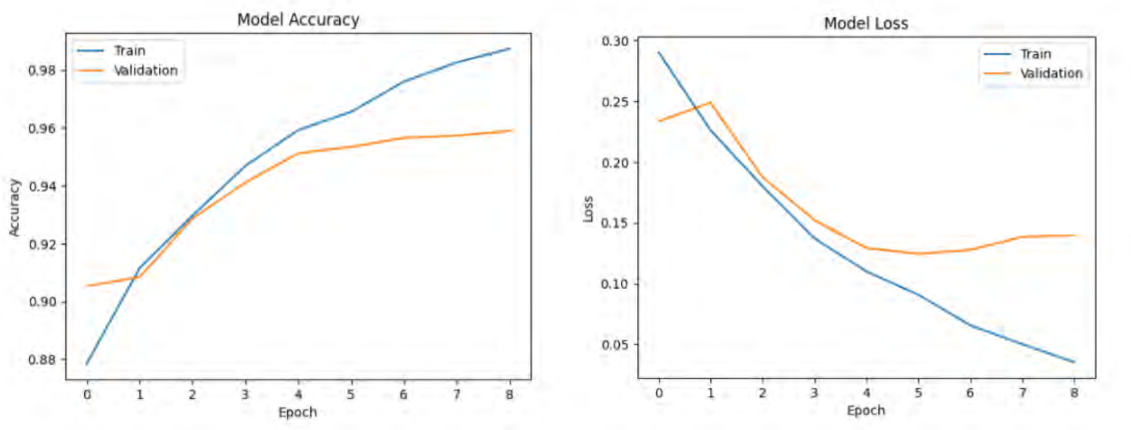


Figure 5.2: Validation accuracy and loss compared to training OF Bi-LSTM

20 epochs and a batch size of 128 are employed during training. The optimal weights of the model based on validation loss are restored by using the early halting mechanism with a patience of 3. The fit function is used to complete the training process, and training data, validation data, and callbacks (including early terminating) are all supplied. The training metrics, such as loss and accuracy, are stored in the history variable for further analysis and performance evaluation of the model.

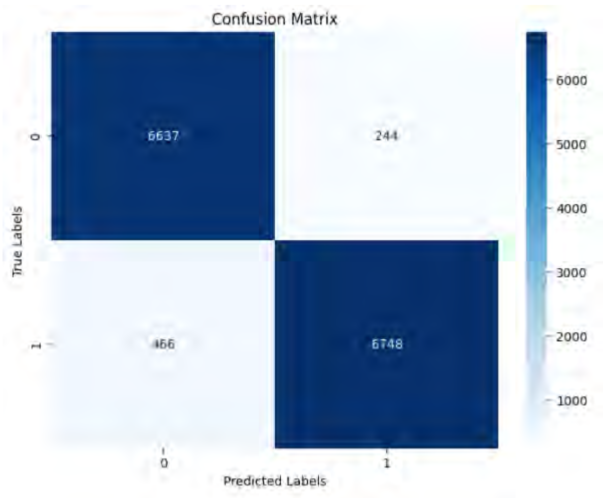
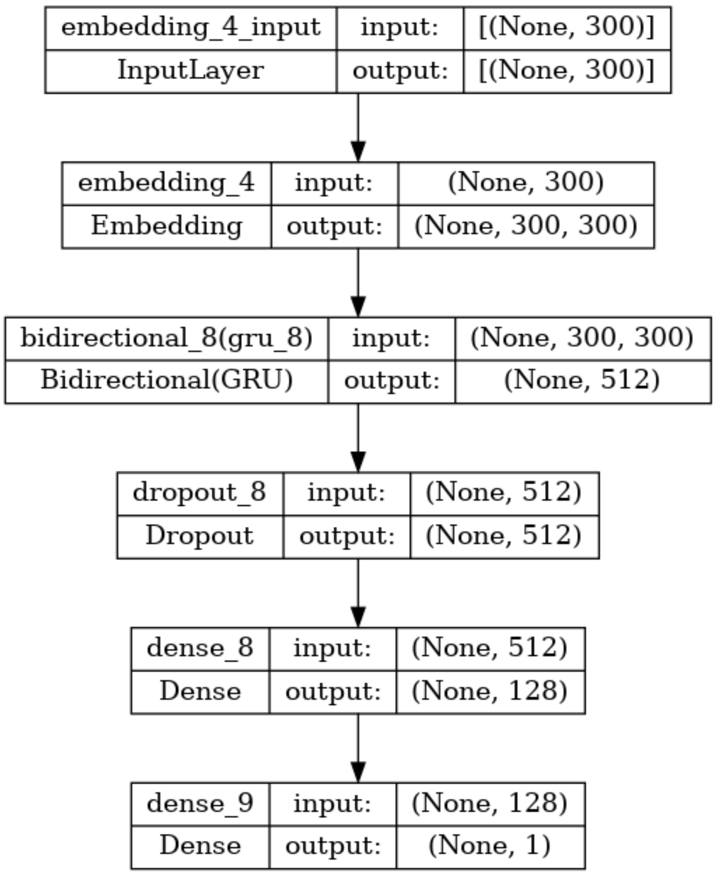


Figure 5.3: Bi-LSTM confusion matrix

The early stop was implemented as before. At the early halt, the weights of the model were also reset to their best training-era values while we maintained an eye on the validation loss. The batch size was increased to 128 with an epoch of 20. The test loss was 13.64 percent, and the test accuracy we received was 94.96 percent. The scores for precision, recall, and f1 are 96.51 percent, 93.54 percent, and 95.0%, respectively.

Bi GRU: The Gated Recurrent Unit that is bidirectional. Due to its ability to collect data from both the past and the future, the bi-GRU (bidirectional gated recurrent unit) is frequently regarded as being preferable than a unidirectional GRU in some situations. A pre-trained GloVe word embedding layer (glove.6B.300d) with a vocabulary size of 10,453,200 words and an embedding dimension of 300 is the first layer in the model architecture. The pre-trained embedding matrix is used to set the weights of the embedding layer. The longest input sequences can be 300 tokens long. Bidirectional GRU (Gated Recurrent Unit) layer with 256 units is the following layer. The layer's bidirectional nature enables the model to gather contextual data moving both ahead and backward. This layer aids in the text data's ability to capture enduring dependencies. In order to avoid overfitting, a Dropout layer with a rate of 0.2 is added after the GRU layer. This layer randomly removes 20% of the neuron activations during training. The generalization of the model is enhanced by this regularization method.


embedding_4_input | input: | [(None, 300)]
InputLayer | output: | [(None, 300)]
embedding_4 | input: | (None, 300)
Embedding | output: | (None, 300, 300)
bidirectional_8(gru_8) | input: | (None, 300, 300)
Bidirectional(GRU) | output: | (None, 512)
dropout_8 | input: | (None, 512)
Dropout | output: | (None, 512)
dense_8 | input: | (None, 512)
Dense | output: | (None, 128)
dense_9 | input: | (None, 128)
Dense | output: | (None, 1)
Figure 5.4: Architecture of Bi - GRU

The layer beyond that has a sigmoid activation function and 128 units of density. It gives the model more nonlinearity and aids in identifying intricate patterns in the data. For binary classification, a Dense output layer with one unit and a sigmoid activation function is added. The likelihood that the input belongs to the positive class is represented by the probability value between 0 and 1 that the sigmoid activation function outputs.
The binary_crossentropy loss function, which is frequently employed for binary classification tasks, and the Adam optimizer, an effective optimization tool, are used in the model's construction. The total number of parameters in each layer and the overall model are displayed in the model summary. There are 11,376,081 total parameters in the model, 922,881 of which may be trained, and 10,453,200 of which cannot be learned because of pre-trained embeddings.

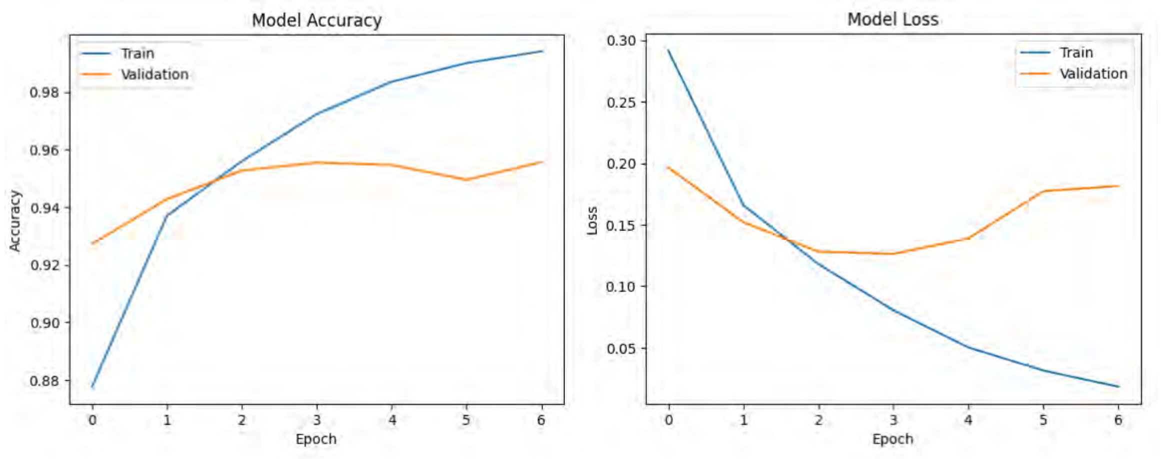


Figure 5.5: Validation accuracy and loss compared to training Of Bi-GRU

This model's test loss is 12.75% and test accuracy is 95.13%. The precision, recall and F1-scores are 95.67%, 94.67% and 95.21% respectively.

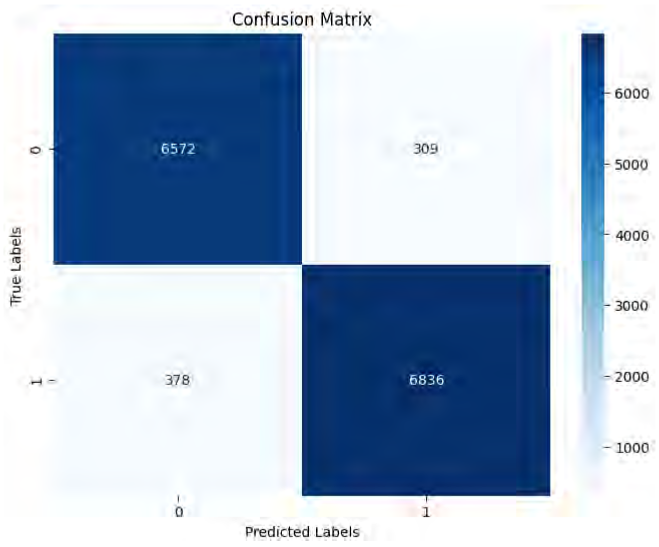


Figure 5.6: Bi-GRU confusion matrix

The model based on Bi-GRU architecture outperformed the model based on Bi-LSTM architecture in the job of classifying bogus news. In comparison to the Bi-LSTM model's test accuracy of 94.96%, the Bi-GRU model attained a higher test accuracy of 95.13%.

Model Name	Test accuracy	Test Loss	Precision	Recall	F-1 score
Bi-LSTM	94.96	13.64	93.54	95.56	95.-0
Bi-GRU	95.13	12.75	95.67	94.70	95.21

Table 5.2: RNN models result

The model based on Bi-GRU architecture outperformed the model based on Bi-LSTM architecture in the job of classifying bogus news. In comparison to the Bi-LSTM model's test accuracy of 94.96%, the Bi-GRU model attained a higher test accuracy of 95.13%. The Bi-GRU model also performed better when precision, recall, and F1 scores were considered. The Bi-GRU model beat other machine learning models including SVM, Multinomial Naive Bayes, Logistic Regression, and Random Forest in the classification of fake news. Bi-GRU has a number of advantages, such as the ability to gather sequential data, control long-term dependencies, learn complicated representations, and efficiently use trained word embeddings. These factors contribute to its enhanced comprehension of the context and patterns included in text data, which leads to better accuracy, precision, recall, and F1 score. The model's deep learning architecture, non-linear transformations, and usage of word embeddings enable it to effectively distinguish between fake and real news by collecting relevant linguistic clues and nuances.

5.4 Result of BERT Model:

Language comprehension has been revolutionized by the groundbreaking natural language processing technique known as BERT (Bidirectional Encoder Representations from Transformers). BERT captures the context-based meaning of words and phrases thanks to its transformer-based architecture and bidirectional learning strategy. BERT creates educational representations by taking into account the words around it, which enables it to comprehend intricate linguistic patterns. It has attained cutting-edge performance across a variety of language tasks, making it a crucial tool for NLP practitioners and researchers. The effects of BERT on language processing have created new opportunities for precise and complex language comprehension.

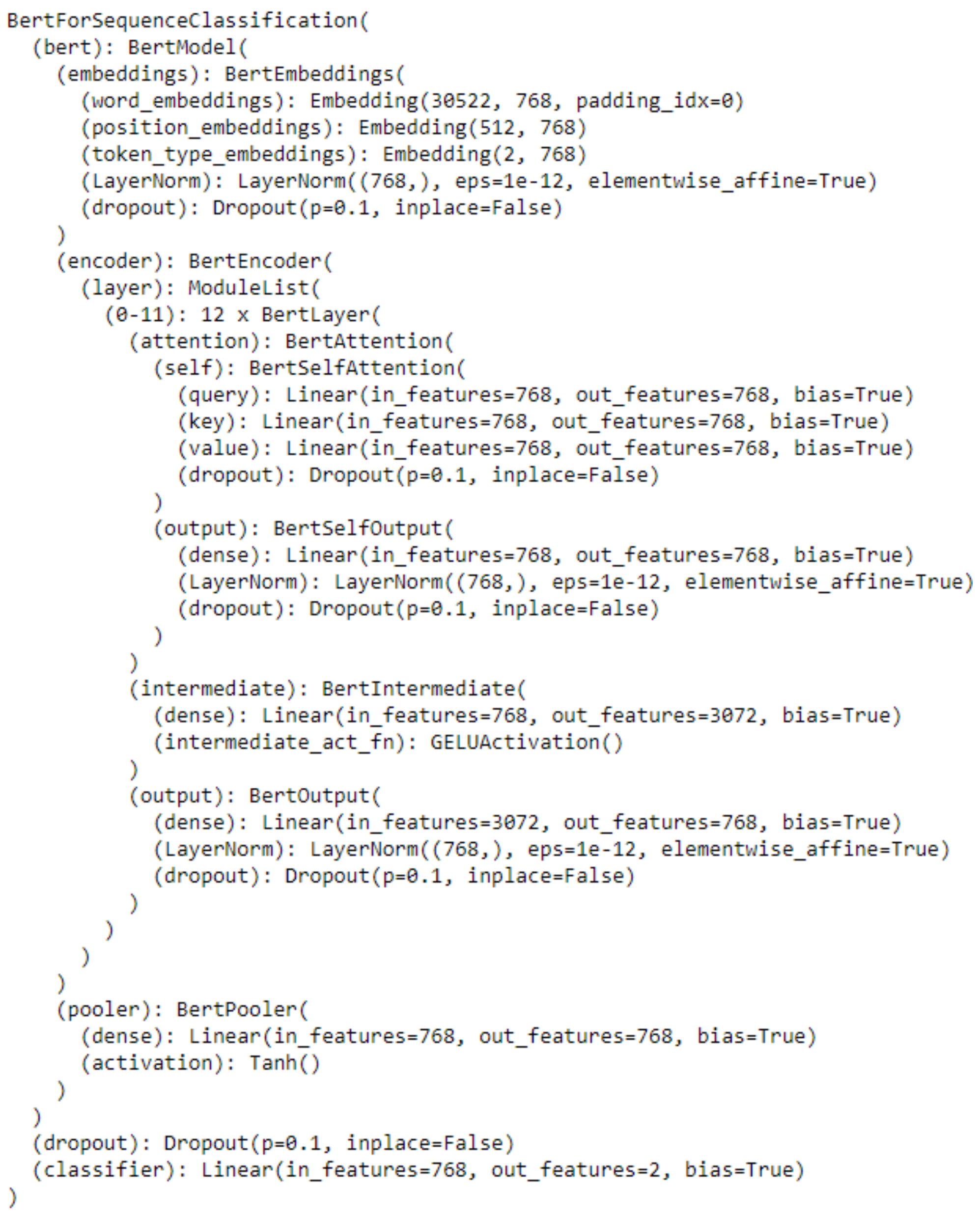
The text and label columns from a DataFrame were first removed. Then, using the 'bert-base-uncased' model, which changes all text to lowercase, we initialized the BERT tokenizer.The definition of a preprocessing function follows. By tokenizing the text, adding unique tokens, and making attention masks, this function encodes the input text using the BERT tokenizer. The text is broken up into tokens or subwords, special tokens such [CLS] and [SEP] are added, and the sequence is padded or cut off at a maximum length of 128 tokens. The encoded inputs are returned as PyTorch tensors by the function. Each sample in the text data is subsequently subjected to the preprocessing function by the code. Both the attention masks and the decoded input IDs are kept in separate lists. In order to build tensors for the input IDs and attention masks, the lists are finally concatenated along the relevant dimension. Additionally, the labels are transformed into a tensor.The overall preprocessing function transforms the input characteristics (token IDs and attention masks) and labels into PyTorch tensors in preparation for additional classification tasks with BERT. The text data is processed by encoding it with the BERT tokenizer.

The data was then divided into training, validation, and test sets, and data loaders were developed for effective data processing. It initially divides the data into training and remaining sets based on the ratios provided, and then further divides the remaining set into a validation and test set. To maintain label distribution in each set, the data splitting is stratified based on the labels. The token IDs, attention masks, and labels are then combined to generate TensorDataset objects for each set. In order to facilitate effective batch-wise data loading during training, validation, and testing, DataLoader objects are lastly constructed for each set. Sequential sampling is used by the validation and test data loaders, whereas random sampling is used by the training data loader. This code snippet offers an organized method for getting the data ready for BERT-based classification tasks, making the training and evaluation procedures that follow easier.

Then we initialize an optimizer for training as well as a BERT model for classifying sequences. Pre-trained weights from the 'bert-base-uncased' model, which was trained on uncased English text, are fed into the BERT model. It is set up for a two-label binary classification task. The model is configured to not output hidden states or attention weights. The model's parameters are updated throughout training by the optimizer, AdamW. For numerical stability, the epsilon value is set to 1e-08 and the learning rate to 1e-5. With the help of this code, the BERT model and optimizer are ready to be trained on the input features and labels. This paves the way for further training and fine-tuning of the model for tasks requiring sequence classification.

Word, position, and token type embeddings are all used as the model's initial input embeddings. These embeddings record the spatial and semantic details of the incoming text. The BERT encoder, which comprises of numerous stacked layers known as BertLayers, receives the input embeddings. For the purpose of capturing contextual information and associations between words in the input sequence, each BertLayer engages in self-attention and feed-forward operations. The top BertLayer is used to extract the special [CLS] token's last hidden state, which is then passed through a pooler. The hyperbolic tangent activation function (Tanh) is applied by the pooler after a linear translation. A fixed-size representation of the full input sequence is produced in this stage. The output of the BERT model is then subjected to a dropout layer. Through the arbitrary removal of some units during training, dropout helps avoid overfitting. The BERT model's output is mapped to the appropriate number of output classes using a linear classifier, which receives the dropout layer's output as input. In this instance, the classifier has two binary output units.

The embedding layer used by the model has a vocabulary size of 30,522 words and an embedding dimension of 768. An embedding layer with a maximum sequence



```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
```

Figure 5.7: Architecture of BERT

length of 512 and an embedding dimension of 768 is used to encode positional data. Token type information is encoded using an embedding layer with two types and an embedding size of 768 to discriminate between various segments or sentences. The embeddings are subjected to layer normalization using an epsilon value of 1e-12 and a vector size of 768.

There are 12 stacked BertLayers in the model. There are many sublayers within each layer. Self-attention method with a dropout rate of 0.1 and 768-byte linear transformations (query, key, and value). output layer following self-attention with a 768-by-768-pixel linear transformation, layer normalization, and a 0.1 dropout rate. Intermediate layer with an activation function (GELU) and a linear translation from 768 to 3072. After the intermediate layer, there is an output layer with a dropout rate of 0.1, layer normalization, and a linear transformation from 3072 to 768. The hyperbolic tangent activation function (Tanh) is used after a pooler layer with a linear transformation from 768 to 768. To regularize the output, a dropout layer with a dropout probability of 0.1 is added after the BERT model. The model includes a linear classifier with an output size of 2 for binary classification and an input size of 768 (the output size of the BERT model). There is also a prejudice word. The training is configured for 15 epochs, and variables for tracking the best validation loss, patience for early stopping, and counter for epochs without improvement are initialised. It ran for 4 epochs.

After running the model we got an Testing accuracy of 95.96% Test Precision: 94.72% Test Recall: 97.52 %.

Model Name	Test accuracy	Precision	Recall
SVM	92.58	93.26	91.76
Multinomial Naive Bayes	83.49	83.98	83.73
Logistic Regression	91.70	92.44	91.25
Random Forest	91.30	91.57	91.41
Bi-LSTM	94.96	93.54	95.56
Bi-GRU	95.13	95.67	94.70
BERT	95.96	94.72	97.52

Table 5.3: Result of all the models

5.5 Result of Covolutional Neural Network (CNN):

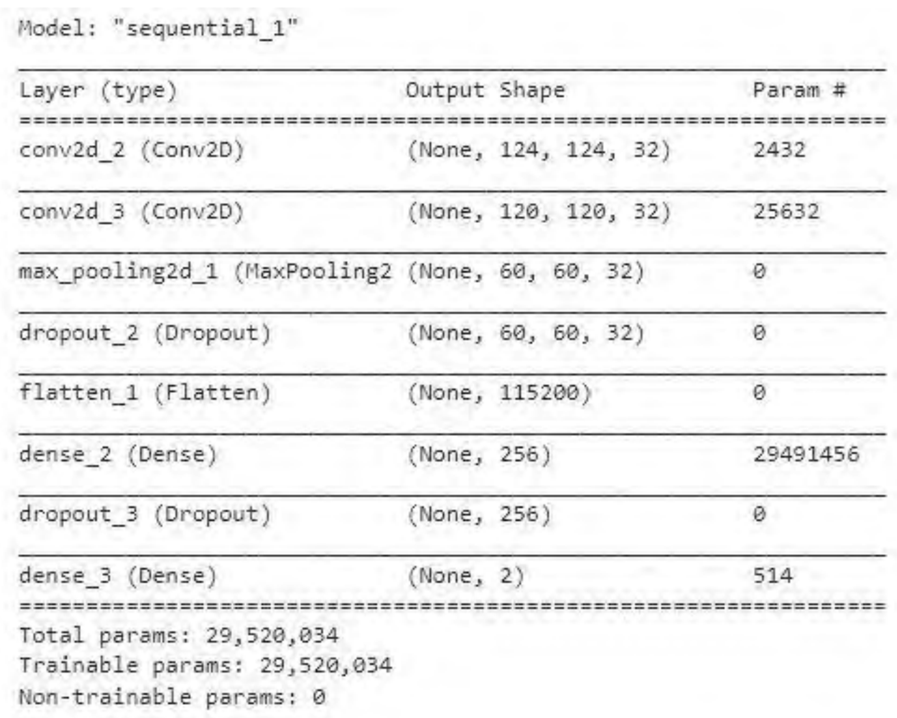
The given code snippet represents the training process and the resulting metrics of a model. Here is a description of the key components:

1. Epochs : The model is trained for 18 epochs, where each epoch represents a complete pass through the entire training dataset.
2. Batch Size : During training, the data is divided into batches, and each batch contains 64 samples. The model's weights are updated after processing each batch.
3. Learning Rate : The initial learning rate is set to 0.0001 (1e-4). The learning rate controls the step size at which the model's parameters are updated during training.
4. Optimizer : The Adam optimizer is used for updating the model's parameters. It utilizes adaptive learning rates and momentum to efficiently optimize the model's performance.
5. Model Compilation : The model is compiled with the Adam optimizer, binary cross-entropy loss function, and accuracy as the evaluation metric.
6. Early Stopping : An early stopping callback is defined to monitor the validation accuracy. If the accuracy does not improve for two consecutive epochs, training is stopped early.
7. Training Process : The model is trained using the 'fit' method. The training data ('X_train' and 'Y_train') and validation data ('X_val' and 'Y_val') are provided. The training progress is displayed for each epoch, showing the loss and accuracy values for both the training and validation sets.
8. Model Saving : The trained model is saved to a file named 'model_casia_run1.h5' for future use.
9. Plotting Loss and Accuracy : After training, the loss and accuracy curves for the training and validation sets are plotted using Matplotlib. The first subplot displays the loss values, while the second subplot shows the accuracy values. The curves provide visual insights into the model's learning progress and performance.

Overall, the code represents the training of a model on the CASIA dataset for face anti-spoofing, utilizing a specific configuration of epochs, batch size, learning rate, and optimizer. The training progress and resulting metrics are visualized to evaluate the model's performance.

In the final result, the model achieved the following performance metrics:

1. Final Accuracy : The model achieved an accuracy of 95.08% on the training data, indicating that it correctly classified 95.08% of the training samples.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 124, 124, 32)      2432
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 120, 120, 32)      25632
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 60, 60, 32)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 60, 60, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 115200)            0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               29491456
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 514
=================================================================
Total params: 29,520,034
Trainable params: 29,520,034
Non-trainable params: 0
```

Figure 5.8: Number of parameter and metrices of CNN

2. Validation Accuracy : The validation accuracy of the model reached 92.92%, demonstrating its ability to generalize well to unseen data.

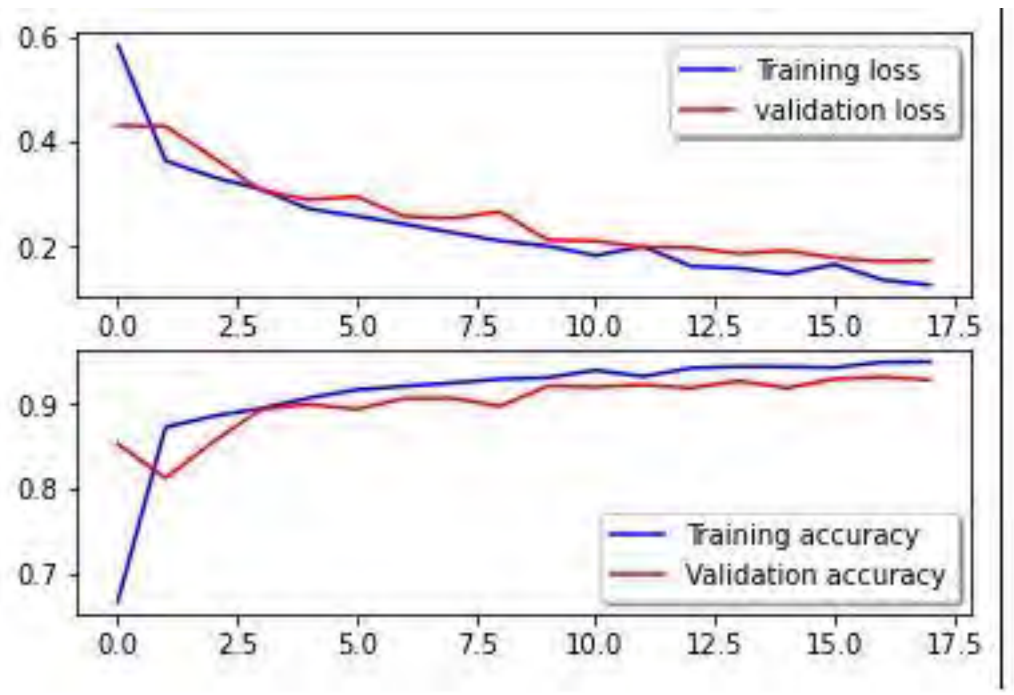
3. Final Loss : The final loss value on the training data was 0.1273, indicating the model's ability to minimize the discrepancy between predicted and actual values during training.

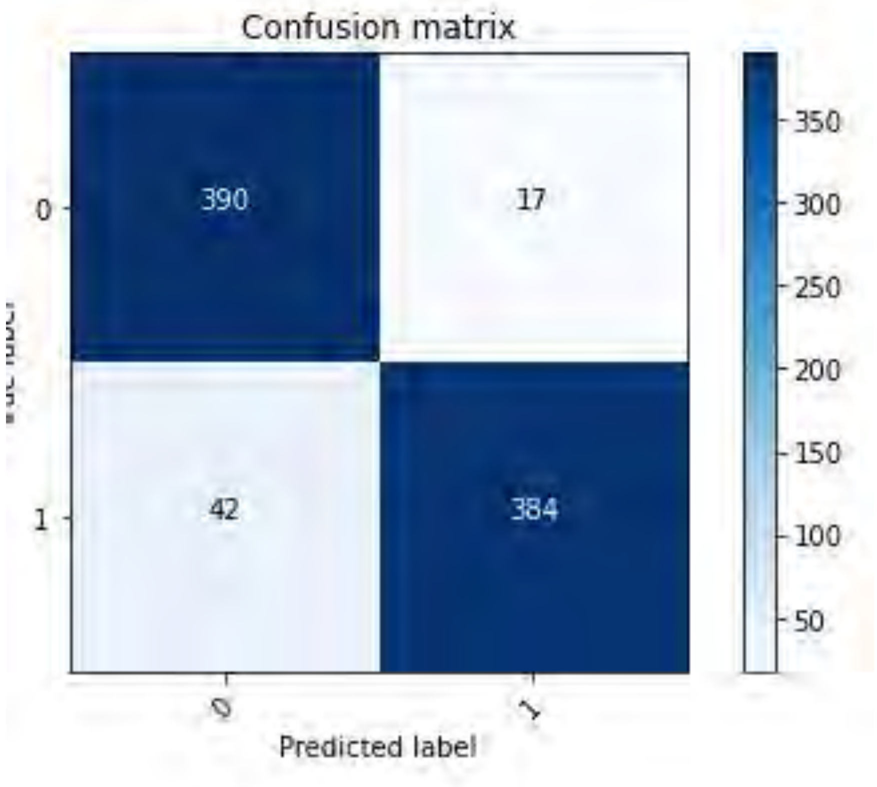
4. Validation Loss : The validation loss at the end of training was 0.1740. This metric quantifies the model's performance on unseen data, with lower values indicating better generalization.

The training and validation loss curves plotted during training show a decreasing trend, suggesting that the model is learning to make more accurate predictions and reducing its training error over time. Similarly, the accuracy curves show an increasing trend, indicating improved classification performance on both training and validation sets.

These results indicate that the model was able to effectively learn patterns and features in the CASIA dataset, resulting in a high level of accuracy in classifying face images as genuine or spoofed.
In the final evaluation, the model was tested on both real and fake/edited images. Out of 2064 fake images, the model correctly identified 2029 images as fake, achieving an accuracy rate of 98.30%. Similarly, out of a total of 7354 real images, the model accurately detected 6643 images, resulting in an accuracy rate of 90.331%.

Please note that these results indicate the model's performance in distinguishing between real and fake/edited images, with a high accuracy rate observed for fake images and a slightly lower accuracy rate for real images.


Training loss
validation loss
Training accuracy
Validation accuracy
0.6
0.4
0.2
0.9
0.8
0.7
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Confusion matrix
390
17
42
384
0
1
Predicted label
350
300
250
200
150
100
50

Chapter 6

Conclusion :

Even however, a lot of research has been done in the last 10 years to identify strategies for halting the dissemination of misleading information. It still merits considerable attention, nevertheless, as a major problem. ICT growth is accelerating the 17 dissemination of misleading information, but we can also use technology to alleviate the issue. In the past, the majority of research efforts focused mostly on text when attempting to identify bogus news. It has been accelerated though after the adding of the pictures. In this analysis, we have looked at a number of ways that have been utilized in the past to discern between authentic news and fake news. We suggest the hybrid system in an effort to make the work more productive and efficient.In order to solve the problem more successfully, we expect to have both improved accuracy and a better solution. In the coming years, we intend to build a more peaceful society by utilizing technology advancement.

Bibliography

[1] O. Ajao, D. Bhowmik, and S. Zargari, "Fake news identification on twitter with hybrid cnn and rnn models," in *Proceedings of the 9th international conference on social media and society*, 2018, pp. 226–230.

[2] X. Zhang, Q. Du, and Z. Zhang, "An explainable machine learning framework for fake financial news detection," in *2020 International Conference on Information Systems-Making Digital Inclusive: Blending the Local and the Global, ICIS 2020*, Association for Information Systems, 2020.

[3] S. Gundapu and R. Mamidi, "Transformer based automatic covid-19 fake news detection system," vol. abs/2101.00180, 2021.

[4] R. K. Kaliyar, A. Goswami, and P. Narang, "Fakebert: Fake news detection in social media with a bert-based deep learning approach," *Multimedia Tools and Applications*, vol. 80, no. 8, pp. 11 765–11 788, Mar. 2021, ISSN: 1573-7721. DOI: 10.1007/s11042-020-10183-2. [Online]. Available: https://doi.org/10.1007/s11042-020-10183-2.

[5] J. Y. Khan, M. T. I. Khondaker, S. Afroz, G. Uddin, and A. Iqbal, "A benchmark study of machine learning models for online fake news detection," *Machine Learning with Applications*, vol. 4, p. 100 032, 2021, ISSN: 2666-8270. DOI: https://doi.org/10.1016/j.mlwa.2021.100032. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S266682702100013X.

[6] J. Nasir, O. Khan, and I. Varlamis, "Fake news detection: A hybrid cnn-rnn based deep learning approach," vol. 1, Apr. 2021, p. 100 007. DOI: 10.1016/j.jjimei.2020.100007.

[7] A. Wani, I. Joshi, S. Khandve, V. Wagh, and R. Joshi, "Evaluating deep learning approaches for covid19 fake news detection," in *Combating Online Hostile Posts in Regional Languages during Emergency Situation*, T. Chakraborty, K. Shu, H. R. Bernard, H. Liu, and M. S. Akhtar, Eds., Cham: Springer International Publishing, 2021, pp. 153–163, ISBN: 978-3-030-73696-5.

[8] X. Zhi, L. Xue, W. Zhi, *et al.*, "Financial fake news detection with multi fact cnn-lstm model," in *2021 IEEE 4th International Conference on Electronics Technology (ICET)*, IEEE, 2021, pp. 1338–1341.

[9] Q. Abbas, M. U. Zeshan, and M. Asif, "A cnn-rnn based fake news detection model using deep learning," in *2022 International Seminar on Computer Science and Engineering Technology (SCSET)*, IEEE, 2022, pp. 40–45.

[10] N. Rai, D. Kumar, N. Kaushik, C. Raj, and A. Ali, “Fake news classification using transformer based enhanced lstm and bert,” *International Journal of Cognitive Computing in Engineering*, vol. 3, pp. 98–105, 2022, ISSN: 2666-3074. DOI: https://doi.org/10.1016/j.ijcce.2022.03.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666307422000092.