UNIVERSITY OF BAHRAIN

College of Engineering

Department of Electrical & Electronics

Identification of Fake Profile Accounts using Dynamic CNN Topology

A Thesis Submitted in Partial Fulfillment of the Requirements for the M.Sc. Degree in Artificial Intelligence Systems

Submitted by:
Fatema Abduljabbar Jawad
20163743

Supervised by:
Prof. Ebrahim Mattar
جامعة البحرين

Kingdom of Bahrain

May 2023

UNIVERSITY OF BAHRAIN

College of Engineering

Department of Electrical & Electronics

جامعة البحرين
المكتبة
LIBRARY
UNIVERSITY OF BAHRAIN

Identification of Fake Profile Accounts using Dynamic CNN Topology

A Thesis Submitted in Partial Fulfillment of the Requirements for the **M.Sc.** Degree in **Artificial Intelligence Systems**

Submitted by

Fatema Abduljabbar Jawad

20163743

Supervised by

Prof. Ebrahim Mattar

جامعة البحرين

Kingdom of Bahrain

May 2023

Defense Committee's Approval

University of Bahrain
College of Engineering
Department of Electrical and Electronics Engineering
MSc in Artificial Intelligence Systems

Student Fatema Abduljabbar Jawad ID number 20163743 has defended her thesis by which she has fulfilled the requirement for the degree of MSc in Artificial Intelligence Systems entitled (Identification of Fake Profile Accounts using Dynamic CNN Topology) on Tuesday, May 23, 2023.

The committee recommends awarding of MSc. Degree with the grade of: Pass

(Defense Committee)

Name		Signature
Professor Ebrahim A Mattar	Supervisor	Ebrahim Mattar
Associate Professor Belal Ayyoub	External examiner	Belal Ayyoub
Professor Wael Al Madani	Internal examiner	

ELECTRICAL & ELECTRONIC ENGINEERING
UNIVERSITY OF BAHRAIN

Academic department's stamp

Abstract

People have become addicted to social media; everyone cannot pass the day without browsing social media. People are sharing their lives now on social media; getting personal information about others has become easy. With this phenomenon, social attacks are increasing, and producing an attack is becoming easier with time. Creating fake accounts and spotting real users for malicious purposes is smooth and effortless. Researchers have put huge efforts into trying to build the perfect model to detect fake accounts accurately, and this is quite hard because of the unexpectable behavior of the fake user. This research aims to build a model that detects fake accounts on Instagram with high accuracy, which would significantly help in keeping our virtual world safe. In our approach, we have used 3 datasets: the 1st dataset and the 2nd dataset were found from an open-access source, and we collected the 3rd dataset using web scraper tools like the IG Exporter Extension from Google Chrome and APIFY, the web scraping and automation platform. Features were extracted and analyzed, and k-mean clustering was applied to end up with five classification classes for the third dataset. We divided our work into two stages. In stage 1, we applied Artificial Neural Network (ANN), K-Nearest Neighbor (KNN), Support Vector Machine (SVM), and Random Forest (RF) machine learning algorithms to the three datasets having a binary classification of real and fake. In stage 2, we have applied a convolutional neural network (CNN) to the 3rd dataset, which has a 5-classification of real, fake1 (fake celebrity account), fake2 (spam account), fake3 (attacker account), and fake4 (bot account). We have applied down sampling and oversampling techniques. A comparison was discussed with previously done work by other researchers.

Publications

Fatema, A. S., and Mattar, E. A. (2022). Recent advances in artificial intelligence techniques for identifying fake accounts: A Review. *6th IET Smart Cities Symposium (SCS 2022)*. doi:10.1049/icp.2023.0651

Sarhan, F. A., and Mattar, E. A. (2023). Fake accounts detection in online social networks using Hybrid Machine Learning Models. *UKSim-AMSS 25th International Conference on Modelling & Simulation*. doi:10.5013/ijssst.a.24.02.02

Sarhan, F. A., and Mattar, E. A. (2023). Fake accounts detection in online social networks using Hybrid Machine Learning Models. *International Journal of Simulation: Systems, Science & Technology*. doi:10.5013/ijssst.a.24.02.02

Sarhan, F. A., & Mattar, E. A. (2023). Deep Learning Approaches for Detecting Fake Accounts in OSNs: Submitted to IET Information Security.

Dedication

I dedicate this thesis to my God, my creator and source of all inspiration, wisdom, and understanding. Throughout this program, he has been my source of strength, and I could never have achieved without his mercy and blessings. This work is also a tribute to my parents, who have always loved me unconditionally and supported my efforts to pursue my goals. To my husband, who has consistently provided me with motivation and support during the master's program. I truly appreciate having you in my life.

Acknowledgments

First and foremost, I would want to express my gratitude to Allah, who has granted me good health and the strength to effectively complete my thesis work. Second, I want to express my gratitude to Professor Ebrahim Mattar for agreeing to be my supervisor and for his ongoing support, kindness, and encouragement during the entire process. He willingly shared his knowledge and expertise.

I'm grateful that the University of Bahrain accepted me and provided me with all the resources I needed to earn my master's degree. The last thing I would like to do is to sincerely thank all my family and friends for supporting and encouraging me as I wrote my thesis. Without their support, I could not have finished my master's degree.

Table of Contents

Defense committee's approval ..... B
Abstract C
Publications ..... D
Dedication ..... E
Acknowledgments ..... F
Table of Contents ..... G
List of Tables ..... J
List of Figures ..... K
List of Abbreviations ..... M

Chapter One
Introduction to the study

1.1. Overview 14
1.2. Background of the study 15
1.3. Problem statement 15
1.4. Objectives of the study 16
1.5. Research questions 16
1.6. The model of the study 17
1.7. Significance of the study 17
1.8. Limitations of the study 18
1.9. Structure of the thesis 19

Chapter Two
Literature Review

2.1. Overview 20
2.2. Types of malicious accounts 21
2.3. Threats and risks 23
2.3.1. Classic threats: 23
2.3.2. Modern threats: 24
2.3.3. Combination threats: 25
2.3.4. Threats targeting children: 25
2.4. Related work 27

2.5. Summary 52

Chapter Three
Research Methodology

3.1. Introduction 53
3.2. Datasets 53
3.2.1. Open-access datasets (1st and 2nd): 54
3.2.2. Dataset preparation (3rd): 56
3.2.3. Features preparation: 60
3.2.4. K-mean clustering: 69
3.2.5. Features analyzing: 73
3.3. First stage 78
3.3.1. System architecture: 78
3.3.2. System implementation: 79
3.4. Second Stage (CNN) 100
3.4.1. System architecture: 101
3.4.2. System implementation: 102
3.5. Summary 129

Chapter Four
Experimental Results

4.1. Overview 130
4.2. Datasets Analysis 130
4.3. Implementation parameters 132
4.4. Evaluation metrics 136
4.5. Stage 1 results 137
4.6. Stage 2 results 143
4.6.1. Before resampling: 143
4.6.2. Down sampling results: 147
4.6.3. Oversampling (SMOTE) results: 152
4.7. Comparative study 156

Chapter Five
Conclusion and Future work

5.1. Overview 162

5.2. Conclusion 162
5.3. Future work 164
References 165

List of Tables

2.1 SUMMARY OF RELATED WORK. 48
3.1 REAL AND FAKE ACCOUNTS DISTRIBUTION 69
3.2 DATA POINTS FOR ROW NUMBER 374. 72
3.3 CENTROIDS. 73
3.4 IS PRIVATE COUNT FOR EACH LABEL. 76
3.5 IS VERIFIED COUNT FOR EACH LABEL 76
3.6 IS BUSINESS COUNT FOR EACH LABEL. 77
3.7 HAS WEBSITE COUNT FOR EACH LABEL. 77
3.8 #POSTS AVERAGE FOR EACH LABEL. 77
3.9 CATEGORICAL FORM FOR THE LABEL 114
4.1 FEATURES OF DATASETS (1ST AND 2ND) 131
4.2 FEATURES OF 3RD DATASET. 132
4.3 PARAMETER DESCRIPTION FOR ANN MODEL. 133
4.4 PARAMETER DESCRIPTION FOR KNN MODEL. 133
4.5 PARAMETER DESCRIPTION FOR RF MODEL 134
4.6 PARAMETER DESCRIPTION FOR SVM MODEL. 134
4.7 PARAMETERS DESCRIPTION FOR CNN MODEL. 135
4.8 CONFUSION MATRIX. 136
4.9 ACCURACIES FOR STAGE 1. 139
4.10 CONFUSION MATRIX AND CLASSIFICATION REPORT FOR STAGE 1. 141
4.11 CONFUSION MATRIX AND CLASSIFICATION REPORT FOR STAGE 1, CONTINUED. 142
4.12 TRAINING AND VALIDATION ACC-LOSS, NO RESAMPLING. 143
4.13 CONFUSION MATRIX AND CLASSIFICATION REPORT, NO RESAMPLING. 144
4.14 TRAINING AND VALIDATION ACC-LOSS, DOWN SAMPLING. 148
4.15 CONFUSION MATRIX AND CLASSIFICATION REPORT, DOWN SAMPLING. 148
4.16 TRAINING AND VALIDATION ACC-LOSS, OVERSAMPLING. 152
4.17 CONFUSION MATRIX AND CLASSIFICATION REPORT, OVERSAMPLING. 152
4.18 PREVIOUS STUDIES COMPARISON WITH PROPOSED STUDY. 161

List of Figures

2.1 RESEARCH METHODOLOGY, (PURBA ET AL., 2020). 33
2.2 USERS INTERACTION WITH FAKE POSTS OVER TIME, (UPPADA ET AL., 2022). 35
2.3 USERS INTERACTION WITH REAL POSTS OVER TIME, (UPPADA ET AL., 2022). 35
3.1 1ST DATASET FAKE AND REAL USERS' DISTRIBUTION. 55
3.2 2ND DATASET FAKE AND REAL USERS' DISTRIBUTION. 55
3.3 IG EXPORTER. 56
3.4 IG EXPORTER VIEW. 57
3.5 APIFY, INSTAGRAM PROFILE SCRAPER. 58
3.6 APIFY, INSTAGRAM PROFILE SCRAPER VIEW. 58
3.7 APIFY, INSTAGRAM PROFILE SCRAPER 1ST RUN. 59
3.8. APIFY, INSTAGRAM PROFILE SCRAPER 2ND RUN. 59
3.9 INSTAGRAM USER DETAILS. 60
3.10 INSTAGRAM NEW USER 61
3.11 INSTAGRAM PRIVATE USER 62
3.12 INSTAGRAM VERIFIED USER. 62
3.13 DESCRIPTION LENGTH FUNCTION. 63
3.14 HAS WEBSITE FUNCTION. 63
3.15 USERNAME==FULL NAME FUNCTION. 64
3.16 #DIGITS IN USERNAME FUNCTION. 65
3.17 #DIGITS IN FULL NAME FUNCTION. 65
3.18 USERNAME HAS DIGITS FUNCTION. 66
3.19 FULL NAME HAS DIGITS FUNCTION 67
3.20 LENGTH OF FULL NAME FUNCTION 67
3.21 LENGTH OF USERNAME FUNCTION 68
3.22 ALL FEATURES OF 3RD DATASET 68
3.23 LABEL VS FOLLOWERS. 74
3.24 LABELS VS FOLLOWING. 75
3.25 LABELS VS HIGHLIGHT REEL COUNT. 75
3.26 LABEL VS #POSTS. 76
3.27 SYSTEM ARCHITECTURE. 79
3.28 POSITIVE CORRELATION BETWEEN VARIABLES, (KHAN ACADEMY, N.D.). 84
3.29 NEGATIVE CORRELATION BETWEEN VARIABLES, (KHAN ACADEMY, N.D.). 84
3.30 NO CORRELATION BETWEEN VARIABLES, (KHAN ACADEMY, N.D.). 84
3.31 CORRELATION MAP FOR 1ST DATASET 85
3.32 CORRELATION MAP FOR 2ND DATASET. 86
3.33 GAUSSIAN DISTRIBUTION, (UPADHYAY, 2020). 89
3.34 3RD DATASET + STAGE 2 (CNN) ARCHITECTURE. 102

3.35 DISTRIBUTION OF 3RD DATASET 107
3.36 CORRELATION MAP FOR 3RD DATASET. 109
3.37 BALANCED DISTRIBUTION FOR 3RD DATASET. 112
3.38 CNN LAYERS VISUALIZATION. 117
3.39 STEPS OF CONVOLUTION LAYER 118
3.40 STEPS OF MAXIMUM POOLING LAYER. 121
3.41 FLATTEN AND FULLY CONNECTED LAYERS, (JEONG, 2019). 122
3.42 SIGMOID AND RELU FUNCTIONS, (SHARMA, 2022). 124
3.43 PLOT OF EXP, (PRIYA, N.D.) 125
3.44 CNN MODEL LAYERS. 128
4.1 ACCURACY DEPENDING ON K VALUE FOR 1ST DATASET. 138
4.2 ACCURACY DEPENDING ON K VALUE FOR 2ND DATASET 138
4.3 ACCURACY DEPENDING ON K VALUE FOR 3RD DATASET 139
4.4 ACCURACY FOR 70 BATCH WITH NO RESAMPLING. 144
4.5 LOSS FOR 70 BATCH WITH NO RESAMPLING. 145
4.6 ACCURACY FOR 32 BATCH WITH NO RESAMPLING. 145
4.7 LOSS FOR 32 BATCH WITH NO RESAMPLING. 146
4.8 ACCURACY FOR 10 BATCH WITH NO RESAMPLING. 146
4.9 LOSS FOR 10 BATCH WITH NO RESAMPLING. 147
4.10 ACCURACY FOR 70 BATCH WITH DOWN SAMPLING. 149
4.11 LOSS FOR 70 BATCH WITH DOWN SAMPLING. 149
4.12 ACCURACY FOR 32 BATCH WITH DOWN SAMPLING. 150
4.13 LOSS FOR 32 BATCH WITH DOWN SAMPLING. 150
4.14 ACCURACY FOR 10 BATCH WITH DOWN SAMPLING. 151
4.15 LOSS FOR 10 BATCH WITH DOWN SAMPLING. 151
4.16 ACCURACY FOR 70 BATCH WITH OVERSAMPLING. 153
4.17 LOSS FOR 70 BATCH WITH OVERSAMPLING. 154
4.18 ACCURACY FOR 32 BATCH WITH OVERSAMPLING. 154
4.19 LOSS FOR 32 BATCH WITH OVERSAMPLING. 155
4.20 ACCURACY FOR 10 BATCH WITH OVERSAMPLING. 155
4.21 LOSS FOR 10 BATCH WITH OVERSAMPLING. 156
4.22 PERFORMANCE OF ALGORITHMS. 157

List of Abbreviations

OSNs	Online Social Networks
SMOTE	Synthetic Minority Oversampling Technique
ReLU	Rectified Linear Unit
SVM	Support Vector Machine
ANN	Artificial Neural Network
RF	Random Forest
NN	Neural Network
AUC	Area Under the Curve
IG	Instagram
ML	Machine Learning
DL	Deep Learning
SVC	Support Vector Classifier
FC	Fully Connected
Conv Net	Convolutional Neural Network
TP	True Positives
FP	False Positives
TN	True Negatives
FN	False Negatives

Chapter One

Introduction to the study

1.1. Overview

By using the Internet, finding any information from any source anywhere in the world has become very unpretentious. Users can get a lot of information and data about other users because of the rising demand for social media sites. Wrong people are drawn to these websites because of the abundance of info that is offered there (Erçahin as cited in Masood & Ud Din, 2019).

On open OSNs, there are millions of connected user accounts that share a variety of material, including videos, photos, and messages. On the other side, fake accounts pose a serious threat to OSN security. In numerous contemporary threats and other illicit activities, counterfeit profiles are utilized. Data analysis and individual accounts are two distinct methods for identifying false profiles in the OSN. Most people agree that the creation of fake profiles does more harm than any other cybercrime (Wanda, 2022).

The OSN environment maintains a vast amount of confidential information. However, it has drawn attackers who steal confidential information, disseminate false information, and spread malicious software. The hacker can commit fraud to acquire private data, including credit card numbers, PINs, and passwords. Other dangerous actions on the OSN mostly rely on fake profiles (Wanda, 2022).

There are many problems associated with fake accounts, including vertical assaults on OSN providers and horizontal attacks on OSN members. The server claims that fake accounts can imitate authentic users by using a fake

name and publishing misleading information on their profiles. Finding fake profiles in a big volume of unstructured data is one of the most difficult tasks. To find abnormalities, it must thoroughly identify user traits (Bindu and Thilagam, as cited in Wanda, 2022).

1.2. Background of the study

Numerous social studies that highlight the impact and risk of fake accounts have drawn the attention of many academics in recent years to the detection of fake accounts.

Recently, researchers have become interested in the discovery of spam on social networking sites. The difficult task of spam identification is essential to social network security (Masood, 2019).

To protect users from all types of dangerous assaults and to maintain their security and privacy, it is crucial to identify spam on OSN sites. Spammers use dangerous techniques that significantly harm the community in the real world (Masood, 2019).

1.3. Problem statement

The considered problem in this research is to apply machine/deep learning algorithms that detects fake accounts in social media and accurately predict fake user behavior, since there are a massive number of users in different social media applications it is difficult to catch each fake user, for this reason we need a very large dataset, and collecting huge amount of data needs time and effort.

1.4. Objectives of the study

The main objective of the thesis is to identify fake accounts using dynamic CNN topology and to evaluate it against other techniques. We chose to work on Instagram platform only because, according to the literature analysis, studies were conducted on one platform either Twitter, Instagram, or Facebook. But we discovered that, in contrast to Facebook and Twitter, there aren't as many datasets as possible available for Instagram. For this reason, we made the difficult choice and put in the effort to collect our own dataset. For this thesis to achieve its goals, we must optimize both our dataset and our module.

The following are the various objectives of this thesis work:

1. To explore literature for related research.
2. To create an AI-based analyzing system with control that can be used to detect abnormal personal behaviors in social media. The system will evaluate anomaly detection analysis methods accurately.
3. To gain access to and review the datasets of available Instagram accounts. Pictures, posts, and other personal information can be included in these datasets.
4. To develop different models for detecting fake accounts using open access datasets and compare it with our main module.

1.5. Research questions

1. What is the effectiveness of utilizing convolutional neural networks (CNN) in detecting fake Instagram accounts?

2. In detecting fake accounts, do convolutional neural networks (CNN) outperform other typical machine learning models?
3. Does SMOTE oversampling improve CNN performance?

1.6. The model of the study

This research paper describes the issue of fraudulent users engaging in destructive behaviors and goes into detail about the technology that is used to identify fake accounts on social media. The curriculum was divided into two stages:

The first entailed gathering two datasets found from the internet and training them with our dataset for comparison using different machine learning algorithms.

For the second stage, which is the focus of our thesis, we assembled and collected our own dataset, cleaned, and processed it, and then, once it was ready, we created a CNN module and fed it with our dataset. We then ran the module several times to assess its performance.

1.7. Significance of the study

The current study provides significant contributions by trying to fill several gaps.

First, the study adds to the limited body of knowledge regarding the actions of fake accounts and their effects on the lives of the public.

Second, in previous work using various machine learning or deep learning algorithms, a binary classification with fake and real was typically used. In our proposed work, we have provided five classifications: real, fake1,

fake2, fake3, and fake4. A clustering algorithm is used to categorize fake users based on their behavior.

Third, because our dataset is of a numerical type, our model offers an evaluation of other machine learning algorithms besides CNN. Offering an evaluation of machine learning models supports our assumption that deep learning approaches can outperform machine learning models with enough size of data.

Finally, to the author's knowledge, no prior study has identified the ideal dataset that always operates or the module that provides the highest accuracy for various datasets. This information was obtained through research in peer-reviewed databases. Nevertheless, the lack of datasets and unexpected actions of fictitious users make this extremely difficult.

1.8. Limitations of the study

This study has two limitations. The first is that we only analyzed numerical data; while this can produce excellent results, using Natural Language Processing to analyze linguistic data, such as usernames, biographies, and post captions, can produce more accurate outcomes. Secondly, after processing and cleaning the data, we were left with 14 features in our dataset. By adding more features and image visualization for profile pictures and image posts, we may create a more precise model. Although working with images and posts of Instagram as a dataset would make it difficult due to privacy concerns.

1.9. Structure of the thesis

Our thesis is divided into 5 chapters.

Chapter 1: This chapter provides an introduction that consists of a background study, research objectives, Methodology, contribution, and thesis outline.

Chapter 2: This chapter provides a literature review about published papers, including types of malicious activities, threats, and risks of fake accounts.

Chapter 3: This chapter describes our proposed work, it starts with dataset preparation, features selection, k-mean clustering, then stage 1 work, and stage 2 work.

Chapter 4: This chapter provides results and a comparative study.

Chapter 5: This chapter includes a conclusion and future aspects.

Chapter Two
Literature Review

2.1. Overview

In the field of artificial intelligence, fake identification can be used for network intrusion malicious activities, fake news, and fake accounts. As the quality of the news is perceived to be lesser than that of traditional news sources, there is a lot of fake news. Spotting fake news is becoming more important and well-liked due to the detrimental effects it has on people and society. The efficacy of detecting fake news only from content is often subpar; hence, it is advised to add user social actions as auxiliary information to enhance fake news identification. Thus, a thorough understanding of the connection between fake news and user profiles on social media is necessary (Shu et al., 2018).

Fake news has detrimental effects on both individuals and society. First, intentionally false information is disseminated to induce people to believe it. Second, spreading false information affects how people respond to real news. Since it might be possible to leverage social media user engagements to help in the detection of fake news, Shu et al., (2018) investigated the relationship between user profiles and either real or fake news. According to experimental results on real-world datasets, some users are more likely to believe false news than accurate news, and these users have different characteristics from those who are more likely to believe factual news. These results make it easier to create profile traits for the detection of fake news (Shu et al., 2018).

As the number of OSNs users grows, fake accounts are created daily, which is considered a major issue that has a negative impact on society, either socially or economically.

Several research has been carried out to discover these artificial entities in online social networks (Rostami et al., 2020). The research is outpacing the rise of these fake accounts/bots. Despite their efforts to construct an exact model for detecting fraudulent accounts, false and automated accounts vary and are widely spread; thus, a huge dataset is required to have a nearly flawless model that works perfectly.

2.2. Types of malicious accounts

User accounts created in social media are specified according to the target aimed when creating them. Ferrara (2022) identified various types of spam on Twitter and categorized them as:

- Promotional accounts: This category includes any account that is set up on the platform with the express purpose of spamming—for example, promoting goods or services for a fee to attract the attention of regular users. This kind of account can be managed manually, automatically, or by a combination of both (Ferrara, 2022).
- False accounts: are those that have been created on the website with the intention of enhancing the visibility or perceived impact of other accounts. As fake followers, false accounts are regularly supplied. Users who utilize pay-per-follower services on Twitter can increase their apparent visibility and popularity by purchasing followers (Ferrara, 2022).

- Bot Accounts: Accounts that are operated by bots (or automated software) are distinguished from regular user accounts in this category. Bot accounts are entirely or primarily controlled by software, i.e., complex computer scripts. The purpose of a bot account is to mimic, faithfully, how a human user would behave on the platform. Due to the automated nature of their activity, some bots are disproportionally busier than most human users. Simple bots have limited capabilities, such as automatically retweeting or liking specific information, whereas more complex artificial intelligence-based bots are able to produce content that is frequently impossible to discern from human-generated content (Ferrara, 2022).

- Malicious accounts are separated into this category based on their intended usage. When malicious users intentionally start or follow an agenda, such as political propaganda, conspiratorial campaigns, disinformation, influence operations, toxic attitudes, etc., with the intention of artificially steering or distorting a Twitter conversation and/or deceiving legitimate users therein involved, it is common practice in the literature to consider them to be engaged in inauthentic activities (Ferrara, 2022).

- Compromised accounts: Accounts that were initially created by reliable parties but afterwards compromised and transferred to another party. The original owner of such accounts may or may not have access to them, making them potentially complicit in the misuse of their accounts without their knowledge or consent. The owner of these accounts might not know they are regularly dormant (Ferrara, 2022).

2.3. Threats and risks

There are various vulnerabilities to social networks; first, we'll highlight the broad dangers before delving more into the problems posed by fake accounts.

Prabhu Kavin et al., (2022) has classified threats of fake accounts into 4 main categories: Classic threats, Modern Threats, Combination threats, and Threats targeting children (Al-Zoubi, as cited in Kavin, 2022).

- The first category includes traditional threats to privacy and security that affect not only OSN members but also web users who do not utilize social networking sites (Al-Zoubi, Faris, as cited in Kavin, 2022).
- The second category discusses growing concerns, such as assaults that are relatively new to the OSN ecosystem and employing OSN technology to threaten users' security and anonymity.
- The third category is combined risks, which explain how hackers nowadays may, and occasionally do, combine numerous types of threats to create complicated and fatal attacks (Reddy, as cited in Kavin, 2022).
- The fourth and final categories of dangers involve kids directly accessing social networks.

2.3.1. Classic threats:

Simply by changing the threat to the user's credit guarantee data, classic threats may exploit the user's personal information posted on a social

network to target both the user and his friends (Zubiaga, as cited in Kavin, 2022).

- Malware: Malware is software created to obstruct a device's operation to gather user credentials and access user's personal data (Kavin, 2022).
- Phishing Attacks: A phishing assault is a sort of social engineering in which a reliable third party gets victim's sensitive and private information (Kavin, 2022).
- Cross-Site Scripting (XSS): A web-based attack is known as an XSS incursion. The intruder who uses XSS benefits from the website user's confidence by allowing malware to run on the user's computer to collect personal data (Kavin, 2022).

2.3.2. Modern threats:

Online social networking is usually linked to these risks. User wants to learn more about other users in contrast to their connections. Attackers attack a user's confidentiality on social networking sites like Twitter because it is important to them. An attacker will be able to obtain this data if secret information is made public (Hemeida, Kasliwal, as cited in Kavin, 2022); otherwise, an attacker can send a friend's request to certain users who have a customized configuration. User's private information will then be shared when the specific attack confirms the request for friendship. Here is a list of the most advanced dangers.

- Clickjacking: Clickjacking is a malware technique that fools people into clicking on something else (Kavin, 2022).

- Deanonymization Attacks: Users will employ pseudonyms to safeguard their secrecy and privacy on a variety of OSNs, including Twitter and Myspace (Kavin, 2022).
- Fake Profiles: Active or semiautomatic profiles in OSNs replicate human actions. Fake accounts can also be used to collect personal information from members of social networking sites (Kavin, 2022).
- Inference attacks: Inference attacks are used in OSNs to predict users' secrets, private details, such as religious affiliation or sexual identity, that they do not want to divulge (Kavin, 2022).

2.3.3. Combination threats:

To create a more complicated threat, attackers can still mix traditional and modern threats (Kavin, 2022).

2.3.4. Threats targeting children:

Children of all ages are likely to be exposed to classic and modern attack features, although some risks are explicitly aimed at younger OSN users (Reddy, Caldeira, cited in Kavin, 2022).

- Online predators: Online predators, often known as online child molesters, pose the greatest risk to the confidentiality of children's personal information (Reddy, cited in Kavin, 2022).
- Risky behavior: This includes "open online interactions with strangers", "using discussion forums to meet strangers", and "sharing personal information and photos with strangers" (Kavin, 2022).

- Occurrence of Cyberbullying: In a technical network, such as email, chat, mobile chat, OSN, cyberbullying occurs when a hacker uses the Internet to harass the victim by sending hurtful text messages, comments, or vulgar messages. on many occasions, posting embarrassing pictures or videos of the victim, or engaging in other abusive behaviors (Boshmaf, Kshirsagar cited in Kavin, 2022).

Spam and fake accounts have a severe impact on the platform's overall health as well as the quality of the user experience. Thorough, peer-reviewed research investigations have identified a wide range of harmful effects caused by spam accounts. Ferrara (2022) has provided a detailed explanation of the implications of spam accounts:

1. Scams and frauds: Twitter rife with this type of abuse. Spam accounts are frequently linked to many scams and frauds, most notably cryptocurrency Ponzi schemes, the promotion of counterfeit goods, links to websites that offer illegal or unlicensed pharmaceuticals, advance-fee scam advertising, phishing website promotion, links to malware downloads, etc. (Nizzoli, as cited in Ferrara, 2022).
2. Public health misinformation: Researchers found that spam and fake accounts are key contributors to the transmission of incorrect information about public health. Numerous studies, both before and after the COVID-19 epidemic, showed that spam accounts' activity was partially or mostly responsible for the success of rumors, unsupported or unconfirmed medical information, and public health disinformation on Twitter. (Ferrara, 2022)
3. Crisis Response Disruption: In times of emergency or natural disaster, such as earthquakes, fires, storms, etc., researchers have demonstrated that Twitter serves as a responsive social sensor, reflecting the

information-seeking activities of those concerned (Sasaki, as cited in Ferrara, 2022). Numerous studies have investigated the impact of spam and fake accounts on conversations that are taking place during an emergency. Gupta (as cited in Ferrara, 2022) examined Twitter media for content created during the April 15, 2013, Boston Marathon Blasts occurrence. The Twitter network served as the source of a lot of harmful profiles and fraudulent information during this time. According to their findings, 29% of the Twitter posts that went viral during the Boston crisis were rumors and fake content, while 51% were general opinions and comments, and the remaining content was actual news.

2.4. Related work

Researchers do not get the same results for machine learning techniques because of the different algorithms used in the model and, most importantly, the different datasets used, which also have different various features, as demonstrated by (Prabhu Kavin et al., 2022) and (Kaur et al., 2021), who used the same machine learning techniques but obtained different results, they used Support vector Machine (SVM), Artificial Neural Network (ANN), and Random forest (RF), SVM has the greatest precision, recall, and F-measure in (Prabhu Kavin et al., 2022). But Kaur et al., (2021) found that RF and NN brings higher AUC than SVM. Prabhu Kavin et al., (2022) used twitter dataset and Kaur et al., (2021) used a public dataset from GitHub website.

Suganya et al., (2021) claimed that Random Forest outperforms Decision Tree and that the bigger the number of attributes, the greater the accuracy. Suganya et al., (2021) also claimed that SVM, naïve bayes, random forest,

and decision tree algorithms are inefficient for detecting fake identities in social media platforms, and they have used a developed algorithm in their work which is an Adaptive SVM classifier (Ada-boosting algorithm with SVM (Ada-boost SVM)), Formula (2.1).

$$\boldsymbol{H(x)} = sign(\sum_{t=1}^{T} \alpha_t h_t(x)) \quad \textbf{(2.1)}$$

whereas,

H(x), is the result of the weak classifier t for the input x.

αt, is the classifier's weight.

$\alpha_t = 0.5 \times \ln(\frac{1-E}{E})$, the classifier's weight is determined by the error rate E.

It can identify the accounts and classify them into bots and non bots with high accuracy, they have used twitter dataset, they first chose the features, then used SVM to train the classifier, then Ada-boost to improve the classifier (weights are reassigned to each instance), last they visualized the results.

Suganya et al., (2021) employed data cleaning, data imputations, oversampling, data integration, and data normalization procedures. Suganya et al., (2021) compared the accuracy of Multinomial naive bayes, Decision tree, Random Forest, SVM, and Ada-boost SVM to be 69.76, 87.85, 91.49, 92.84, and 97.16 respectively, proving that Ada-boost SVM accuracy was the best and outperformed other techniques.

According to Rostami et al., (2020), the feature selection process in classification-based techniques is critical for detecting fake accounts in online social networks since the classifier system must be applied on a large volume of data in real time. Feature Selection is used to improve the

accuracy of the procedure. It also improves the algorithms' prediction capacity by picking the most important variables and removing the redundant and unimportant ones. Furthermore, researchers frequently study and investigate detected features to better understand the behavior of fraudulent accounts. It implies that another priority that should be considered is the stability of the feature selection process. Additionally, stability is not considered a sufficient measure for evaluating the selected features and is frequently paired with classification performance.

Rostami et al. (2020) use a multi-objective hybrid approach to identify the best feature set for spotting fake profiles on the Twitter social network.

In their suggested work, the minimum-Redundancy-Maximum-Relevance (mRMR) algorithm selects a compact collection of outstanding features to decrease search space (Ding & Peng as cited in Rostami, 2020). Assume F denotes a feature set with P features; the mRMR method picks the feature with the strongest correlation to the target class (y), as shown in Formula (2.2).

$$\boldsymbol{F1} = \arg\ max_{x_i}\big(cor(x_i, y)\big) \quad x_i \in F \quad i = 1, \dots, p \qquad (2.2)$$

The mRMR technique employs two criteria to compute the correlation of characteristics to the target class.

1. Mutual information, which is a standard criterion for identifying variable dependency and is utilized for discrete / nominal characteristics, Formula (2.3).

$$\boldsymbol{I(x, y)} = \int\int p(x, y) \log \frac{p(x,y)}{p(x)p(y)} dxdy \qquad (2.3)$$

2. The F-test measure for continuous variables, which examines the rate of variation between the class and the mean of variance within the class, Formula (2.4).

$$F(x,y) = \frac{(n-C)\sum_c n_{c(\overline{x_C}-\bar{x})^2}}{c-1\sum_c(n_c-1)\sigma_c{}^2} \quad (2.4)$$

The Pearson's correlation coefficient, Formula (2.5) is used to calculate the redundancy of continuous characteristics.

$$R(x_i x_j) = \frac{Cov(x_i,x_j)}{\sqrt{Var(x_i)Var(x_j)}} \qquad x_i, x_j \epsilon F \quad (2.5)$$

The stability of the feature selection process is computed using the mean similarity of all pairs of feature sets formed on K subsampling's of the dataset, as shown in Formula (2.6).

F feature sets chosen from the subsampling i and F demonstrate the similarity requirement of the pairs of features F and F.

$$S_{tot} = \frac{2\times\sum_{i=1}^{k}\sum_{j=i+1}^{k} S(F_i,F_j)}{k\times(k-1)} \quad (2.6)$$

The Jaccard Index criteria is used to calculate the similarity of pairs of feature sets constructed via K subsampling of the dataset, Formula (2.7) (Saeys as cited in Rostami, 2020).

$$S(F_i,F_j) = \frac{|F_i \cap F_j|}{|F_i \cup F_j|} \quad (2.7)$$

Rostami et al., (2020) used two Twitter social network datasets provided by Cresci as cited in Rostami et al., (2020). The first dataset (Social Spam Bot 1) contains false accounts that were used by one of the Rome municipal candidates to post its policies in 2014, while the second dataset (Social Spam Bot 3) contains bots that are used to advertise and publish Amazon

firm products. Genuine accounts in the databases also refer to human users. Experiments on two Twitter datasets revealed that the proposed strategy might provide more optimal and balanced performance when compared to other existing methods (Rostami et al., 2020).

Fake accounts can also be purchased online from a variety of websites that sell them. Purba et al., (2020) purchased fake followers from a variety of places and gathered real users from private university followers on Instagram.

Fake accounts have a broad concept that has a lot of meanings, for example Suganya et al., (2021) has classified fake accounts into two types: duplicate accounts, which are created by users for additional malicious use, and false/unwanted accounts, which are created by businesses or organizations and usually violate the social media terms and services. Hashtags on Twitter are sometimes generated by bots to increase product sales through paid promotions.

The primary distinction between automated and fake profiles, according to Akyon et al. (2020), is that the first enhances its own metrics, whilst the second enhances those of other users and creates an undesirable social media atmosphere. Akyon et al., (2020) has implemented fake account detection and automated account detection for two datasets.

SMOTE-NC was utilized for oversampling in fake account detection (Akyon et al., 2020). The macro average approach is used to calculate F1 scores. The reason for using macro average is because the distribution in the data does not match the true distribution on Instagram. It is desirable to value both false and actual users equally. Oversampling has improved the performance of all approaches. The neural network has the best

performance without oversampling, whereas SVM and neural network perform equally well with oversampling (94%) (Akyon et al., 2020).

Neural networks and SVM have the highest overall F-1 scores in automated account detection. It is expected since neural networks can learn complex mappings with enough training data, and SVMs are widely recognized for optimizing the margin better than most other binary methods. Because the characteristics are not clearly independent, poor performance from Naive Bayes and logistic regression was unsurprising (considering these methods highly rely on the independence of the features). Furthermore, the low precision of the Naive Bayes with Gaussian distribution indicates that the true distribution of the labels does not conform to this probability assumption (Akyon et al., 2020).

As a result, SVM and neural network-based approaches got the most promising F1 score for automated account detection (86%), with neural network achieving the best F1 score performance (95%) (Akyon et al., 2020).

Purba et al., (2020) Research methodology is shown in Figure (2.1). They conducted two experiments: one with two classes (fake or authentic user) and one with four classes (Active fake users, Inactive fake users, Spammers, and authentic users). The 17 classification features have been shown to be sufficient for distinguishing between authentic and fake users. The maximum classification accuracies for two and four classes are 90.09% and 91.76%, respectively, when Random Forest is used. Furthermore, the features importance result suggests that all metadata aspects, i.e., number of posts, number of followers, link availability, number of followings, and biography length, are the most important predictors in the classification (Purba et al., 2020).

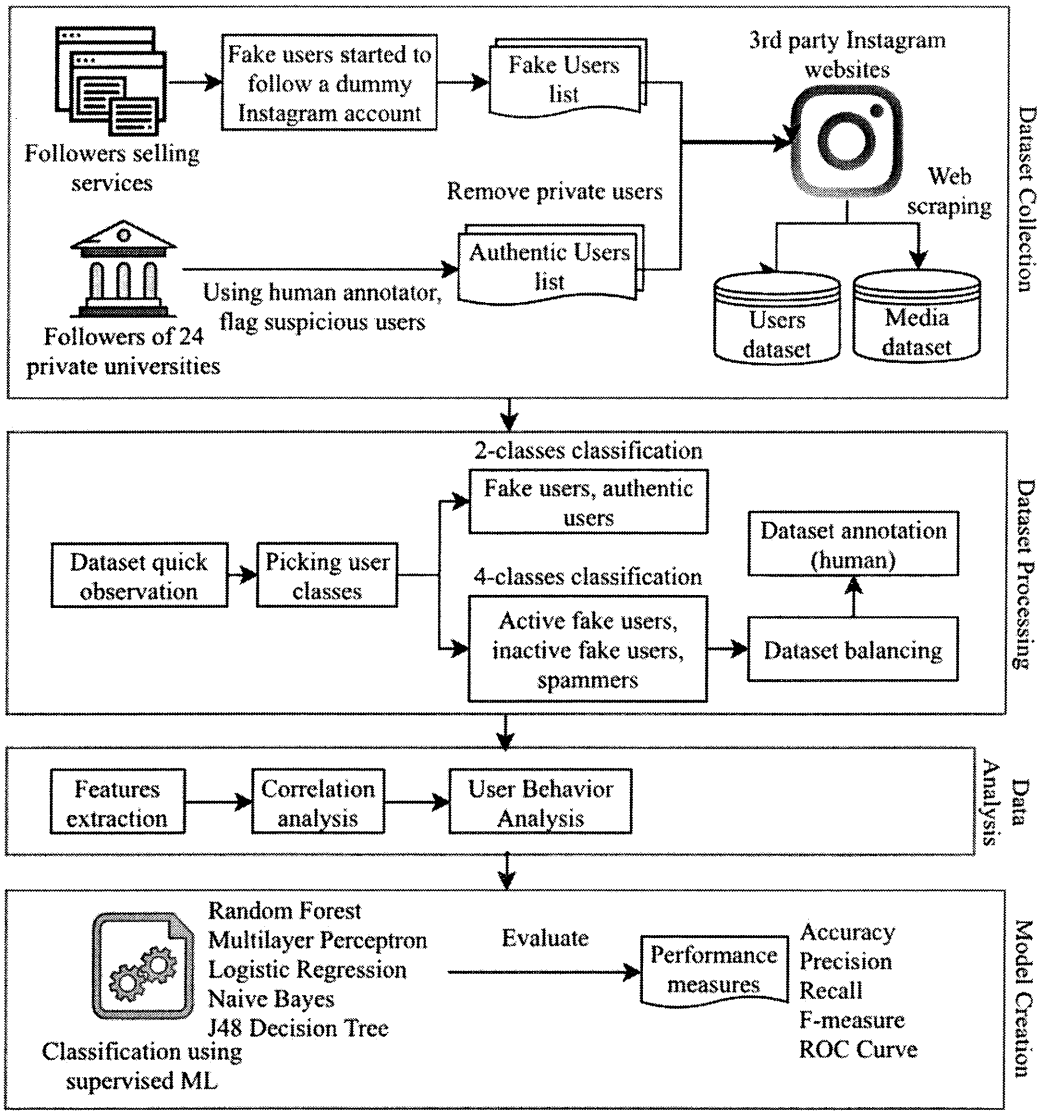


Figure (2.1): Research methodology, (Purba et al., 2020).

Because metadata features are so important, another experiment is being conducted: categorization using solely metadata features. With 79.66% accuracy, this classification was the most accurate. The advantage of utilizing metadata-only classification is that the metadata is easily obtained, even from private users. This is a significant benefit because 49.11% of the users in the raw dataset are private users. As a result, while using all the features improves classification accuracy, classification using metadata-only is still adequate considering that many Instagram users are private users (Purba et al., 2020).

Deep learning algorithms have a lot of potential for spotting fake news. There haven't been many studies that show the value of neural networks in this field (Nasir et al., 2021). Nasir et al., (2021) developed a novel hybrid deep learning model for false news classification [CNN-RNN] that mixes convolutional and recurrent neural networks. Because of their capacity to capture both local and sequential properties of input data, the CNN-RNN combination has proven beneficial in various classification and regression applications.

The model was proven successfully on two fake news datasets (ISOT and FA-KES). Because it distinguishes between fake and genuine news, this challenge is framed as a binary classification problem. Overall, the hybrid CNN-RNN model suggested here surpasses all existing supervised classification algorithms. Random Forests outperformed the other approaches in terms of accuracy for both datasets. The maximum accuracy performance was achieved by KNNs for FA-KES and Decision Tree for ISOT (Nasir et al., 2021).

Users try to spread fake news via image-based articles since visuals spread faster and have strong engagement patterns. The visual element of such fake news contributes greatly to its spread. The fake news image motivates the viewer to share the news by providing a perception of realism, resulting in the spread of the fake news over the online social network, Figure (2.2) and (2.3) illustrates the user's interaction with fake and real posts in time, users are more likely to engage with fake posts (Uppada et al., 2022).

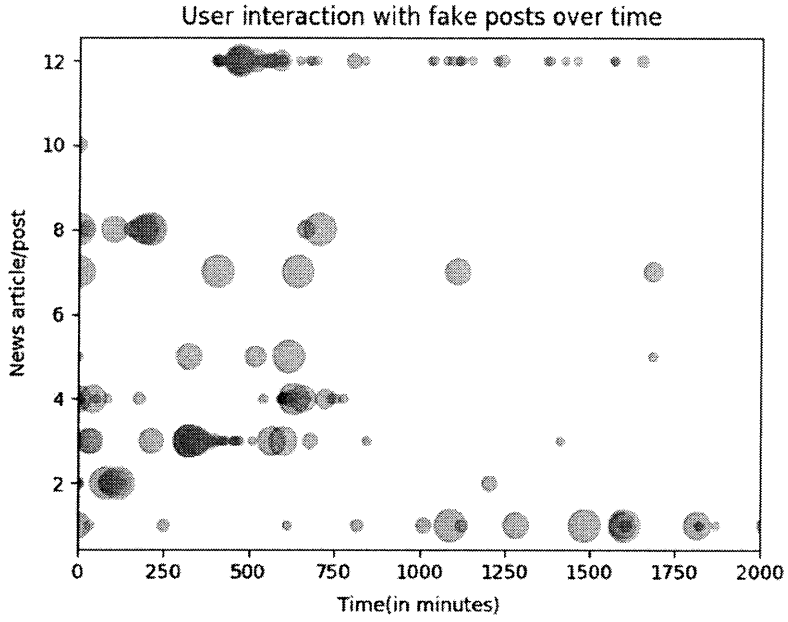


Figure (2.2): Users interaction with fake posts over time, (Uppada et al., 2022).

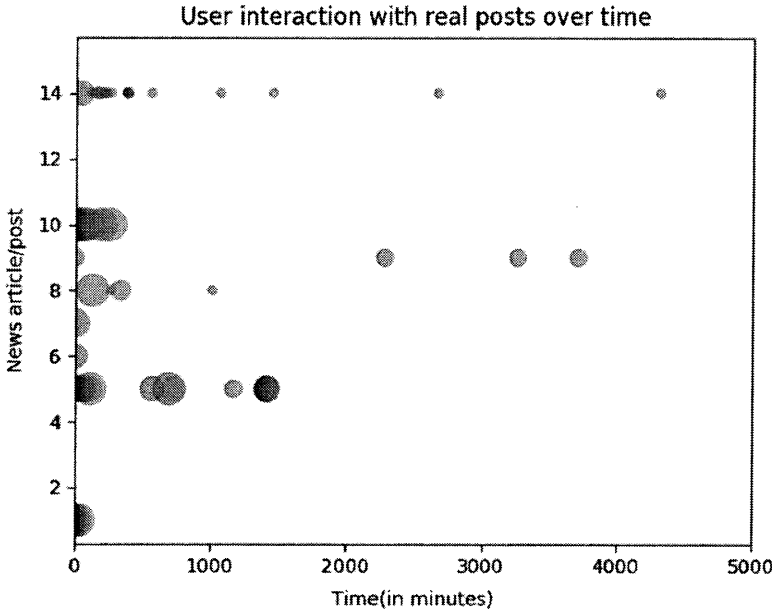


Figure (2.3): Users interaction with real posts over time, (Uppada et al., 2022).

Rumors are detected by Twitter by validating news accounts, and any news that is tweeted without a confirmed source is classified as a rumor. The verified status on Twitter aids in determining the validity of an account.

Because a protected account only permits a restricted number of users to view the published messages, the likelihood of spreading fake news is reduced. Twitter's authorized TV network accounts would have reliable proof in comparison to the innocent unverified user accounts. News organizations verify the material before it is released. Twitter confirms their identities and safeguards the news channel from fraudulent profiles. There is also trust in the verified media outlet account's facts (Prabhu Kavin et al., 2022), (Uppada et al., 2022).

Fake and News can also be linked to the creation of fake or anonymous accounts (usually bots) for the faster distribution of news without revealing the identity of the news spreader. Fake news identification is a collection of content-based, social context-based, and propagation-based approaches, as demonstrated by Uppada et al., (2022). Prabhu Kavin et al., (2022) also divided features into content-based features and user-based features.

Content-based techniques to fake news detection rely on linguistic (lexical and syntactical) elements that can detect false cues or writing styles (Uppada et al., 2022).

Prabhu Kavin et al., (2022) demonstrated that content-based features on Twitter include: characteristics linked to user tweets, fraudsters posting duplicate tweets, the number of tweets, the frequency of tweets, spam words, hashtag ratio, Formula (2.8), URL ratio, Formula (2.9), and mention ratio, Formula (2.10).

$$\boldsymbol{hashtag\ ratio} = \frac{duplicate\ hashtag}{unique\ hashtag * tweet\ count} \quad (2.8)$$

$$\boldsymbol{url's\ ratio} = \frac{hashtag\ duplicate\ urls}{hashtag\ unique\ urls * tweet\ count} \quad (2.9)$$

$$\boldsymbol{mention\ ratio;\ @tweets} = \frac{tweets\ containing\ @}{total\ number\ of\ tweets} \quad (2.10)$$

Social context-based elements, namely user-based traits identified by Prabhu Kavin et al., (2022), aid in the investigation of account authenticity: Accountancy age, Follower-to-Follower Ratio (lower for normal users and higher for fraudsters), Formula (2.11), Verified (whether or not Twitter verifies the account), Protected (whether or not the user account is protected), screen name length, has description (whether or not the user account has a description), name length, favorites count, statuses count (Uppada et al., 2022).

$$\boldsymbol{ffratio} = \frac{number\ of\ following}{number\ of\ followers} \quad (2.11)$$

$$\boldsymbol{reputation} = \frac{followers}{followers+following} \quad (2.12)$$

Ali & Abdullah, (2022) utilized The Management Information Base "MIB" dataset, which is made up of five Twitter datasets—two of which represent genuine accounts and three of which represent fake accounts. 5,301 total accounts with 29 features make up the total. During the data gathering procedure, they ran into a scenario where some mistakes happened, and some data was lost. Due to this issue, the quality of the data is reduced, which results in poor outcomes when the data are analyzed and explored. Many of the fields in their data are empty, which has a negative impact on classification and yields inaccurate findings. So, they made the choice to focus solely on important features during the feature extraction step, avoiding irrelevant or redundant features to create the best possible subset of features for the model.

They employed the Spearman correlation coefficient, one of the feature selection filtering techniques that assesses the strength and direction of the monotonic link between two quantitative variables. To indicate the degree of correlation, they have values ranging from (-1) to (+1). Each correlation

metric is 0 when the two variables are not correlated. They also employed the chi-square test, a statistical technique for assessing the independence of two events.

They applied the ensemble system by stacking features that had been retrieved from the datasets and normalized. To increase predictive power, the ensemble learning technique of stacking combines several classifiers or regression models using a meta-classifier or meta-regressor. Comparing the stack technique to employing each algorithm independently, the phase of detecting fake accounts yielded a high evaluation rate in terms of accuracy, precision, and F1 score. The stack approach resulted in an accuracy of 99%, a precision of 99%, and an F1 score of 99.2%.

Munga & Mohandas, (2022) used a dataset containing 1481 real profiles and 1337 fake profiles. 20% of the accounts are used for data testing, and the remaining 80% are used for data training. 25 features make up the collected dataset. For each round, a different dataset is created. The dataset with all features is used in the first round. Label encoding is used to fill features with null values at random. The second round consists of a dataset with few features and no null values. Following the completion of the data augmentation, the processed data is now ready to be trained for the model. The dataset is subjected to the use of three different classification methods: RF, SVM, and NN, to produce three distinct models. Both rounds 1 and 2 of each algorithm are conducted independently. Because the features selected in round 2 are the best feature set, accuracy has increased from round 1 to round 2.

Shevtsov et al., (2022) develop a dataset where the most popular hashtags connected to the US 2020 elections are initially gathered to capture the Twitter dynamics of the US 2020 elections just before election day. From

September 1st, 2020, to November 3rd, 2020, all tweets using these hashtags were retrieved using the Twitter API, yielding a dataset of 15.6 million tweets and 3.2 million users. Shevtsov et al., (2022) introduced a novel approach for differentiating between bots and authentic Twitter users by extracting a variety of attributes using a supervised machine learning (ML) framework. The ground truth labels are calculated by combining the output from two online bot detection technologies, and the suggested system specifically includes the extraction and labeling of several features.

Shevtsov et al., (2022) determined the Extreme Gradient Boosting (XGBoost) technique to be the best ML model together with a particular set of features after doing an extensive ML study that involved train/validation/test split, feature selection, oversampling, and hyper-parameter tuning. The chosen XGBoost model achieves the highest bot detection accuracy when trained on a wide range of combination features, ranging from profile and context features to time-based and interaction characteristics.

S B, (2022) made the case that community detection is essential for comprehending the architecture of complex networks and ultimately deriving informational value from them. Each profile on a social network offers a wealth of information, including gender, friends, comments, education, employment, and more. This data has both private and public portions.

S B, (2022) have only used the information that is public to identify the fake profiles in the social network because private information is not available. The dataset is split into training and testing data, and S B, (2022) utilized Random Forest to analyze it. A training dataset is used to train

classification algorithms, and a testing dataset is used to assess the algorithm's effectiveness.

The results are related to determining whether the record is real or fake using designed features and materials created using AI models like brain organizations. The forecasts show that the calculating brain network produced predictions with 93% accuracy. Later, it is anticipated that new aspects would make it easier to identify and detect things, such as executing skin location should be achievable by using more accurate regular language handling algorithms (S B, 2022).

Less than half of the 4000+ Facebook profiles that Awan et al., (2022) collected from publicly accessible data were fake profiles. Using a Spark-based project that is trained on 70% of the total uploaded profile data and then attempts to forecast the remaining 30% of the data using a random forest model, Awan et al., (2022) suggest a method for preventing fraudulent profiles. Their data's classification accuracy is calculated using the X-test and Y-prediction accuracy scores. This results in a 94% accuracy rate.

The work by Ghafoor et al., (2022) makes use of three datasets. The first dataset, which has three columns—title, content, and label—was compiled from "Tencent's Fact Platform" and "People Daily Online." 7796 entries with four attributes are included in the second dataset. These three are label, title text, and id. A whole dataset makes up the third one. These dataset's material is compiled from a variety of other sources. Additionally, it has the same column and the same number of features as dataset 2.

Passive aggressive, decision tree, and naive Bayes classifiers are utilized in the work of Ghafoor et al., (2022). These algorithms were all developed using machine learning techniques.

In the three trials, three classifications were applied to the datasets (Ghafoor et al., 2022). The dataset was divided into several ratios so that each classifier's robustness could be tested and assessed (30-70, 40-60, 50-50, 60-40, and 70-30).

According to the first experimental findings, a passive-aggressive algorithm performed the best across all data ratios. The dataset was divided in half, with 70% of the data utilized for training and 30% for testing, yielding the greatest performance (93.70%). Additionally, the decision tree and naive Bayes also obtained 81% and 83%, classification accuracy, respectively. Due to the limited training data, all three classification algorithms also had the lowest accuracy when data was divided by a 30–70% ratio.

The results of the second and third trials showed that a passive-aggressive algorithm outperformed all other algorithms for all data ratios. The best results were 93.90% accuracy for the second trial, and 99.90% for the third trial. The samples were divided by a 70–30% ratio (Ghafoor et al., 2022).

Bordbar et al., (2022) used the GitHub website to gather Twitter information. The dataset has 5,384,162 users and 16,011,445 connections between them. The data, which contains 10,000 fake nodes, is divided into 1,000,000 nodes, and then mapped into a graph. The network is then transformed into an adjacency matrix to examine the relationships and activities between each node (Jouili, as cited in Bordbar et al., 2022).

Three similarity criteria were added at this step-in order to have strong features and improved separation in fake classes. Ten similarity measures are applied to each matrix because of new similarity criteria termed Adamic Adar, Pearson Correlation, and Euclidean similarity. All matrices underwent PCA due to their numerous and dispersed features, and ten crucial aspects were accepted; as a result, they have a matrix of 1,000,000 * 100 dimensions. The Min-Max normalization algorithm (Gajera, as cited in Bordbar et al., 2022) was then used. The ratio of 0.3 to 0.7 between the train and test portions of the data—which were divided into two parts—was also determined through testing. The GAN overcomes the challenge of the minority class of training data by fabricating fake nodes using a generator model. The discriminator learns to distinguish between fake nodes and real ones.

This methodology (Bordbar et al., 2022) tried to address the problems with huge data and unequal classes. In generative deep learning, the discriminator learns nodes while the generator creates fake data. Their system was superior since it was adaptable to managing various types of data as well as minority classes. According to the findings of their trial, the approach can achieve excellent accuracy with an AUC score of 98.1%. They asserted that using GAN could be a viable way to manage data and minority classes.

Varuna et al., (2022) applied LSTM configuration. They achieved training accuracy of 98% and validation accuracy of 97.9%. The user-generated content and profile metadata of users are the data stored in social networks. The crucial public profile elements, such as the username, screen name, etc., give personal information about the user's identity. The ties in the social network graph that reflect the user's varied interests are used to define the connectivity using this information. In total, the amount of

personal data that social network users directly contribute makes up various kinds of publicly accessible data. They compared their proposed work with two existing methods: Sequential Minimal Optimization (SMO) and Markov Clustering Algorithm (MCA). They stated that their work is quite successful, and that LSTM exceeds the two existing methods, SMO and MCL.

Swathi et al., (2021) developed a method to distinguish between authentic and fake Facebook profiles using machine learning techniques. The same framework had also been applied to the prediction of stalking. By using publicly accessible datasets of fake and real user to train the neural network model, they were able to detect fake profiles. 1469 real users and 1329 fake users make up their data set. They chose and retrieved certain features from the dataset, and then they applied a classification algorithm to it. The list of user ids that was readily available in the "Initial Chat Friends List" area of the page source was used for stalking data.

Swathi et al., (2021) noted that the user IDs on this list correspond to those that engage with a profile the most frequently in terms of chats, likes, comments, timeline views, etc. Through the Facebook Graph API, they obtained the user's information using this id. By expressing the objects in the graph and the connections between them, this API provides a consistent representation of the social network. The stalking data is identified as user data. To determine if the person is stalking from a fake profile or not, they used this data as the test data for the prediction. When the detection model has been adequately trained and evaluated, they can use it for stalking prediction. They carried out training using the back-propagation approach. They viewed the profiles on the list of people stalking as the ones that needed to be judged as fake or real. They used a sample data set of 20 users

for stalking to conduct tests, and they achieved a 93% accuracy rate with a 6% error rate.

A dynamic CNN was suggested by Wanda, (2022) to train a classification model. They added a new component to the last layer of the neural network, termed "Run Max", which is a general activation function. By employing the suggested activation layer, they increased accuracy during the training and testing process. Their method uses user profile data as features and achieves Precision=94.00, Recall=93.21, F1-Score=93.42, and a better area under curve (AUC) score=0.9547. In this challenge, they were aiming for a result with improved accuracy and minimal loss.

Rather than employing a standard method like SoftMax, which has a bottleneck in a language modeling problem while training in extended features (Yang as mentioned in Wanda, 2022). Wanda, (2022) presented "Run Max" as an alternative activation function to improve the existing SoftMax to cope with the SoftMax bottleneck from the output set (range) and design a function without extra parameters. "Run Max" has ideal qualities for output activation functions since its gradient computation is composed of an exponential function and a Gaussian function.

Wanda, (2022) proposed RunMax nonlinear activation as a possible solution to the SoftMax bottleneck in a 1D dataset. Wanda, (2022) created RunMax linear classifier f(z)i in the following manner:

$$\boldsymbol{f(z)_i} = \frac{\exp(z_i)\varphi(z_i)}{\sum_{m=1}^{M}\exp(z_m)\varphi(z_m)}, \qquad i = 1, \ldots, M \qquad \textbf{(2.13)}$$

To create the fully connected layer, Wanda, (2022) suggested a unique linear classifier with $\varphi(z_i)$ and $\varphi(z_m)$ as a Gaussian function (φ) for the z_i and z_m samples, rather than a normal activation function that generates logits' probability values.

In the training phase, Wanda (2022) employed gradient descent algorithms to denote the stride size and achieve a minimum. Based on the demonstration, they were able to outperform the typical activation function in terms of accuracy throughout training and testing. They also pointed out that, to assess the neural network's performance in the challenge of classifying fake accounts, picking an appropriate Gaussian factor becomes crucial.

Wanda, et al. (2021) also suggests, in another study, using traditional machine learning to solve the problem. However, it takes a lot of money and time. In order to improve the performance of the graph, Wanda, et al. (2021) uses a WalkPool function to illustrate the dynamic deep learning classifier. The neural network with a dynamic optimizer can achieve improved accuracy and a smaller loss by utilizing the suggested model with a generic pooling function. If the Deep Friend model is trained with a larger epoch and a high momentum score in SGD, it can also give a competitive outcome. As a result, a model with a dynamic optimizer may be a viable choice to train using powerful hardware. According to their experiment findings, DeepFriend outperforms traditional learning algorithms in terms of AUC score, producing the highest AUC score of 0.9970. Furthermore, not only does the graph operate more efficiently, but it also produces excellent accuracy. In comparison to other evaluated algorithms, their model may achieve the greatest TP and TN scores. It may show the best detection rate for identifying benign and abnormal nodes.

Wanda et al. (2020) suggested DeepProfile, a model with a novel CNN with a generic pooling function to cope with fake profile categorization, as an alternative to employing common learning and expensive feature engineering and computational resources. By utilizing the WalkPool pooling layer to optimize CNN computation and boost accuracy, the

approach creates a dynamic CNN architecture. Because DeepProfile produces AUC = 0.9547 and performs better than other traditional learning strategies, it can make accurate level predictions. A combination of DeepProfile with the WalkPool pooling function outperform traditional learning algorithms in terms of accuracy and loss.

Sahoo & Gupta (2021) suggested an autonomous fake news detection system for the Chrome environment that can recognize misleading material on Facebook. To be more specific, they used several news item attributes together with several Facebook account-related variables to analyze the behavior of the account using deep learning. They said that their intended way of spotting fake news was more accurate than the latest cutting-edge techniques. They proposed a method that combined several elements related to the user's profile with machine learning and deep learning-based analysis to detect false news in things like advertisements, messages, images, and other stuff. They claim that their proposed technique actively seeks for fake news.

Additionally, their method gathers numerous news articles shared by Facebook users as well as various attributes connected to various profiles to analyze fictitious activity. Additionally, the user's homepage in the Chrome environment displays the outcome in the form of a pop-up box. According to Sahoo & Gupta (2021), choosing attributes based on a user's profile and shared material offers better input for their approach to outcome prediction. The Facebook API and their crawler extract several new features. Additionally, based on a few chosen and already-existing features for processing, they assessed the effectiveness of several postings retrieved from various Facebook profiles.

To identify fake news, they have separately assessed user content and news content aspects. SVM (Support vector machine) provides superior accuracy in a machine learning environment compared to KNN, Logistic Regression, Decision Tree, and Naive Bayes classifiers. Due to the lengthy storing of existing records, they have seen improved accuracy in LSTM-based deep learning analyses. When user profile and news content variables are considered, LSTM in machine learning provides greater accuracy of 99.42% as compared to SVM. When using both user profiles and fake news features, their approach predicts better outcomes in a deep learning environment compared to a machine learning environment. This achieves their goal of determining how accurately their approach based on multiple features separates fake news from real news in the online environment.

Bindu et al., (2022) requested MIB projects for the dataset. There are four sections to the data. First, for academic research on spotting fake accounts, real accounts with the codes E13 and TFP have been made available. Genuine accounts are mostly made by US and European researchers, social scientists, and journalists. The fake accounts were purchased from numerous Twitter internet marketplaces. The collection contains 34 features, some of which are primarily unfilled. There are altogether 3964 rows in the data. There are fields with null values following the removal of some fields and the cleansing of the data. Fields containing null values were filled with 0. There is a description box with text in it that is not categorical. Therefore, the profile is filled with 1 if it has a description and 0 otherwise. In their method, random forest provided better outcomes. While KNN also produces good results, Radial SVM falls short due to its higher rate of False Negatives.

Table (2.1): Summary of Related work.

No.	Title	Year	Method	Sample	Evaluation
1	Machine Learning-Based Secure Data Acquisition for Fake Accounts Detection in Future Mobile Communication Networks	2022	Detecting fake account and Comparing between SVM, ANN and RF.	From a twitter account, real accounts are manually collected. Twitter REST API is used to collect user profile data. Another data set of the "The Fake Project" was combined. The 7,973 account records in the final data set were split into two groups: model training (75%), and model testing (25%).	SVM algorithm has the highest scores utilizing the user-based features, and content-based features.
2	FAKE SOCIAL PROFILE DETECTION USING MACHINE LEARNING	2021	Detecting fake accounts using SVM, NN and RF.	Open accessed dataset from GitHub with 564 datapoints.	RF and NN resulted in a higher AUC score than SVM.
3	Retraction: Detect fake identities using improved machine learning algorithm (J. Phys.: Conf. ser. 1916 012056).	2021	Define the identity of account whether it is real or fake using (Ada-boost SVM).	Twitter dataset.	Ada-boost SVM is the efficient classifier for Identifying Fake Accounts.
4	Detecting Fake Accounts on Twitter Social Network Using Multi-Objective Hybrid Feature Selection Approach	2020	Detecting fake accounts on Twitter using RF, NB, and SVM	Two twitter datasets. The Social Spam Bot 1, and the Social Spam Bot 3	Support Vector Machine algorithm was the best classifier.
5	Classification of Instagram fake users using supervised machine learning algorithms	2020	Detecting fake users using RF, Multilayer perceptron, LR, NB, and J48 DT	Web scraping was used to get dataset from various Instagram websites. The fake users were purchased from Indonesian sellers, whereas the real users gathered from private universities.	RF resulted with better accuracy using metadata and media data, DT resulted with the highest accuracy using only meta data.
6	Instagram Fake and Automated Account Detection	2020	Fake account detection and automated account detection using SVM, NB, LR, and NN.	1002 real account and 201 fake accounts has been gathered after manual labeling for the first dataset, the second dataset consists of 700 real account and 700 automated accounts.	In fake account detection, oversampling has improved the performance. NN was the best. In automated account detection, SVM and NN had the highest overall F-1 score.

7	Fake news detection: A hybrid CNN-RNN based deep learning approach	2021	Fake news detection using a combination of CNN and RNN	Two open access datasets are used for the assessment studies. There are 804 news articles in the FA-KES dataset. The 45,000 news articles that make up the ISOT dataset.	In the ISOT dataset. Accuracy rises, and loss fall, showing that the model improves at classifying the articles. The accuracy is not increasing smoothly for the FA-KES dataset. And the loss is almost constant over all 10 epochs.
8	Novel approaches to fake news and fake account detection in OSNs: user social engagement and visual content centric model	2022	SENAD model identifies the dissemination of fake news and helps in locating accounts that distribute fake news. CredNN, CNN to detect fake images.	Fake Newsnet and Twitter dataset, fake image dataset (Weibo dataset)	It is time-consuming to identify the deliberately altered photos that are compressed, downloaded, and uploaded several times because ELA operates with compromised images. It can be difficult to locate an image dataset with visual sentiment data associated with it.
9	Fake accounts detection on social media using stack ensemble system	2022	Detecting fake accounts in twitter using stack ensemble system, RF, SVM, NB, LR	Management Information Base "MIB" dataset is used.	When compared to applying each algorithm independently, the stack technique had a higher evaluation rate in terms of accuracy, precision, and F1 score.
10	Feature Selection for Identification of Fake Profiles on Facebook	2022	Using RF, SVM, and NN, to determine whether a Facebook profile is fake or authentic	A dataset of 1337 fake and 1481 real profiles is collected with numerous attributes, including friends, followers, status, and other metrics.	Each algorithm runs its first and second iterations individually. The accuracy has improved from round 1 to round 2 because the features chosen in round 2 are better.
11	Identification of Twitter Bots Based on an Explainable Machine Learning Framework: The US 2020 Elections Case Study	2022	Employing RF, SVM, XGBoost to identify Twitter bots based on labeled Twitter data	The most well-liked hashtags associated with the US 2020 elections are initially gathered by the author and added to a dataset. All the tweets that contain these hashtags are retrieved using the Twitter API.	The selected XGBoost model provides the highest bot identification accuracy.

12	Use of ANN to Identify Fake Profiles	2022	Detect fake profiles using RF	As no informal community dataset is openly available for the investigation purpose of identifying the fake accounts, the author turned to scraping profiles.	The projections demonstrate that the analytical brain network produced predictions with an accuracy of 93%.
13	Fake profile recognition using big data analytics in social media platforms	2022	Detect fake profiles using Spark ML, and RF	4000+ Facebook profiles were collected from publicly available data by the author.	Accuracy reached 94%.
14	Fake News Identification on social media Using Machine Learning Techniques	2022	Detect fake news using passive aggressive, J48, and NB.	Author used 3 datasets collected from Kaggle and GitHub.	The findings indicate that on all three datasets, the passive-aggressive machine learning algorithm performed best with the proposed features that had a training to testing ratio of 70% to 30%.
15	Detecting Fake Accounts Trough Generative Adversarial Network in Online social media	2022	Detecting fake accounts using GAN, LR, and SVM.	The Twitter information used to support the study's findings are gathered from a GitHub repository. In the dataset, there are 5,384,162 users and 16,011,445 links between them.	This method excelled in handling minority classes as well as being adaptable in handling all types of data. GAN is a potentially effective method for managing minority classes and data.
16	An Efficient Framework for Fake Profile Identification Using Metaheuristic and Deep Learning Techniques	2022	Detecting fake profiles using LSTM	The user-generated content and profile metadata of users are the information stored in Social Media networks. The crucial public profile details and particular communication details were considered as features.	The proposed method had a validation accuracy of 97.9% and training accuracy of 98%.
17	Fake Profile Detection and Stalking Prediction on Facebook	2021	Detecting fake profile using NN.	The model is trained using open access dataset, author used users' id to get the user details through Facebook Graph API.	The model achieved 93% accuracy, and 6% error.
18	RunMax: fake profile classification using novel nonlinear activation in CNN	2022	Detecting fake profile using CNN.	Author gathered and derived from the corpus several diverse features. By breaking out specific user profile data, the author was able to compile a sample dataset of over	By utilizing the generic function network, the suggested model outperforms previous activation techniques in terms of accuracy,

				3000 unique users from OSN profiles.	enhancing the CNN graph's efficiency.
19	Multiple features-based approach for automatic fake news detection on social networks using deep learning	2021	The suggested method employs KNN, SVM, LR, DT, NB, and LSTM in a reactive manner to identify fake news.	The shared content features of several profiles have been gathered by the author. Then, to categorize fake news on Facebook, he examined features using his Chrome extension.	In comparison to machine learning algorithms, the deep learning algorithm displayed remarkable performance with a 99.4% accuracy rate. The trial findings showed that the method successfully detected fake news in real time.
20	Detection of Fake Accounts in Twitter Using Data Science	2022	Fake accounts detection using KNN, SVM, and RF.	Data for this Project was given by MIB projects. Authentic accounts are mostly made by American and European researchers, social scientists, and journalists. The fake accounts were bought from different Twitter online marketplaces that deal in fake accounts.	For predicting, random forest produces superior outcomes. While KNN also produces good results, Radial SVM falls short due to its higher rate of False Negatives.
21	DeepFriend: Finding abnormal nodes in online social networks using Dynamic Deep Learning.	2021	Abnormal node detection in OSNs using DeepFriend Method	Data were collected from Facebook and Flicker.	Proposed method achieved better the greatest accuracy with 98%.
22	Finding fake profile in online social network using dynamic CNN.	2020	Fake profile detection using DeepProfile method with dynamic CNN.	More than 3000 users are collected from OSN.	Proposed model achieved better accuracy with small epoch.

2.5. Summary

The study of fake account detection is important, particularly in the context of social media and cybersecurity. The widespread usage of fake accounts on social media platforms has emerged as a significant issue, and their detection is now essential to preventing a variety of dangerous behaviors. Machine learning algorithms is one of the areas of study for methods and instruments for identifying fake accounts. Researchers must research the characteristics and behavior of social media accounts to develop a successful model.

Chapter Three

Research Methodology

3.1. Introduction

Detecting fake accounts on social media has many prospects and to design the model properly, we need to figure out what are the best algorithms that gives the better accuracies, and this is achieved by collecting the data and comparing between different AI algorithms. Many researchers have proposed similar work for detecting fake accounts using different machine learning and deep learning algorithms. There are various social media applications, but the most used applications are Facebook, Instagram, and twitter. In this chapter, we will describe our model. We are going to use 3 Instagram datasets, in two stages, the 1st stage describes ANN, KNN, RF, and SVM. The 2nd stage describes CNN topology.

3.2. Datasets

From our searching's we have notice that there are many work have been done for Facebook and twitter, either for detecting fake profiles or detecting fake news, also there are many open access datasets for Facebook and Twitter, we found two open-access small Instagram datasets to work on for the first stage, besides that, we have collected the 3rd dataset by ourselves for a total of 83,862 rows and 17 features and 5 classification labels: real users, fake1, fake2, fake3, and fake4. Fake users are numbered according to their behavior.

3.2.1. Open-access datasets (1st and 2nd):

We have found two publicly accessible datasets. The first dataset was gathered by Jafari, (2021) who made an Instagram account and subsequently purchased 700 fake followers. To stop new users from following the page, the account privacy was adjusted to private after that. He utilized a C# web scraper to get data from all 700 accounts. From his private Instagram account, real users were taken. This will be the 1st dataset. The 2nd dataset was compiled by Bakhshandeh, (2019). The dataset has a high level of accuracy as Bakhshandeh, (2019) claims, he personally recognized the spammer/fake accounts present in this dataset after carefully evaluating each occurrence, even though Bakhshandeh, (2019) acknowledged that there may be a few accounts in the spammers list that were incorrectly identified. A crawler was used to gather the data between March 15 and March 19, 2019.

The 1st dataset, which is not balanced as depicted in Figure (3.1), comprises 12 features with 692 fake users and 93 real users. Since none of the accounts in the dataset have a channel, the "has channel" attribute has no impact on the dataset and therefore has no impact on the results, and we removed it. The 2nd dataset contains 11 features with 348 actual accounts and 348 fake accounts, which are in this instance balanced as illustrated in Figure (3.2). 20% of the dataset was used for testing, while 80% was used for training, for both datasets.

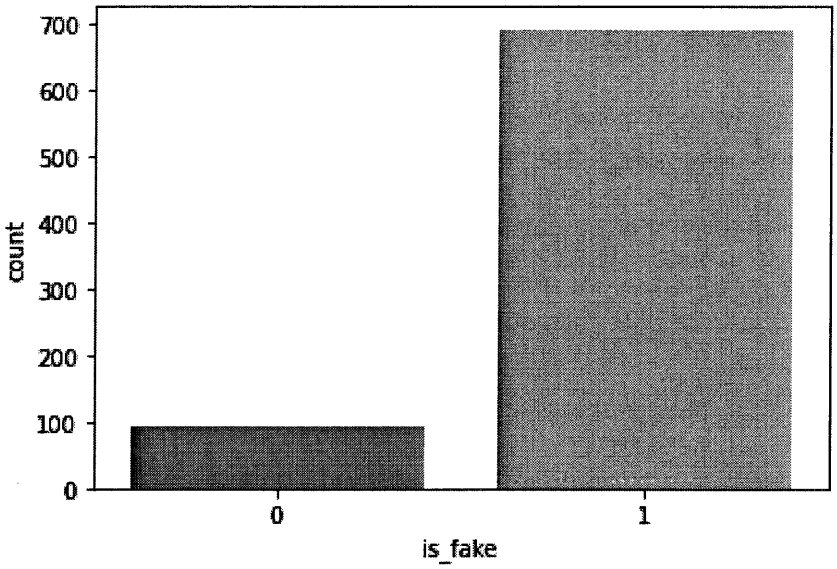


Figure (3.1): 1st Dataset fake and real users' distribution.

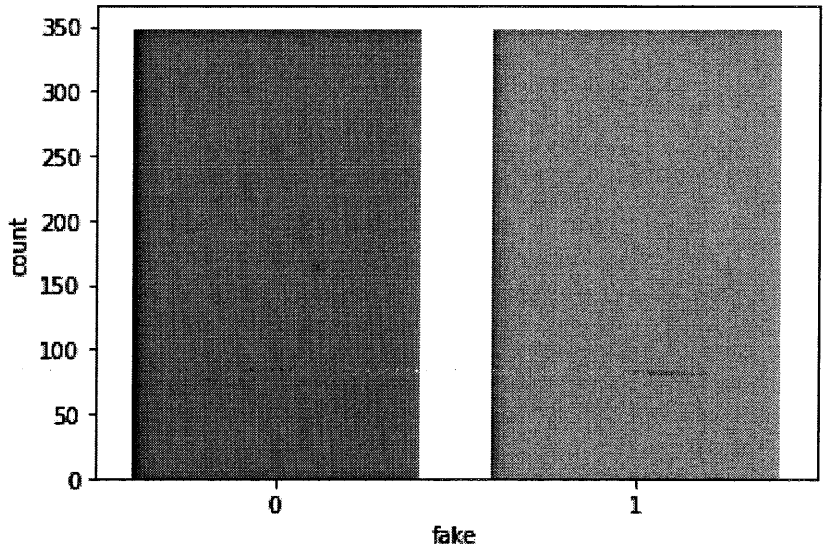


Figure (3.2): 2nd Dataset fake and real users' distribution.

3.2.2. Dataset preparation (3rd):

Collecting and gathering the dataset was a challenging part due to privacy reasons, we have used IG Exporter provided by chrome web store, it is a tool that exports Instagram user's followers to an Excel or CSV file, Figure (3.3). For fake accounts we have bought them from an online market, this market sells followers, likes, and comments, we have bought only followers. Real accounts are taken from a university account.



Figure (3.3): IG Exporter.

Exporter for IG extension has a delay option which is the time between scraping different users, and this delay can be increased if we faced any problem like blocking our Instagram account, if so, the parsing will freeze, and no more scraping happens, Figure (3.4).

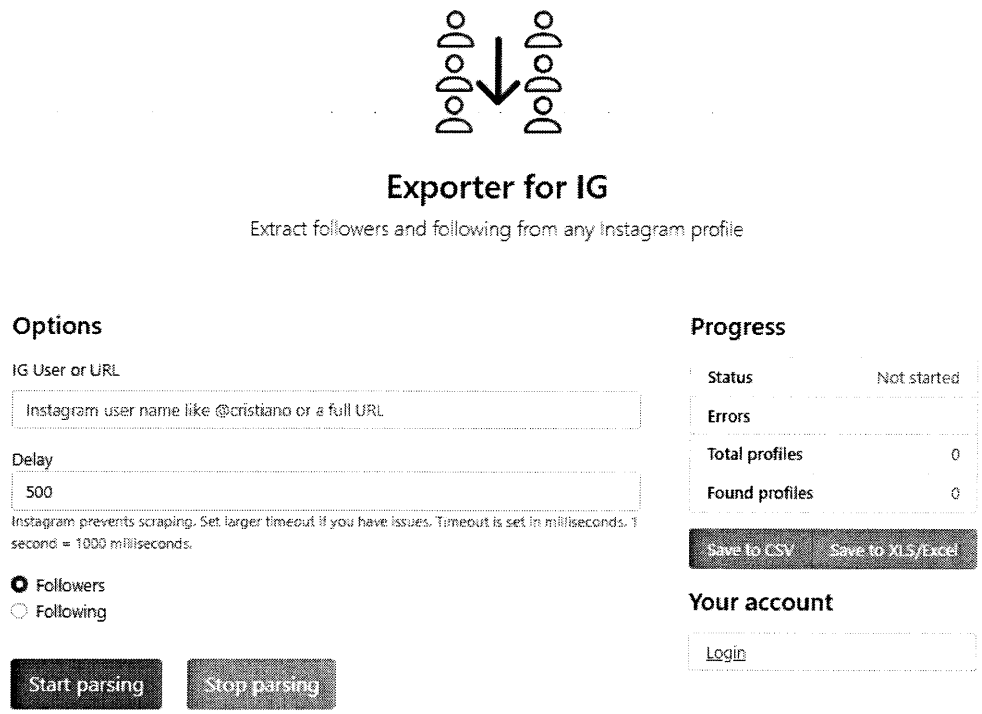


Figure (3.4): IG Exporter View.

Exporter for IG extension gives only five attributes which are: ID (numeric Instagram ID), Username, Full name, Profile picture URL, and Is IG profile verified. These attributes are not enough to be fed to the model, so we needed to extract more features. So we have tried at an early stage to use a python code for extracting more data, the code was built in Visual Studio Code, we have used Instagramy package that was offered by pypi.org, the code worked successfully and we only needed to provide the usernames of the accounts and the code will collect all features about them, but the code wasn't useful at all, because we have a huge amount of data to be collected, and it was freezing at every run and it stops at a range of 24-40 accounts collected only, and to rerun the code we had to reset the IP address each time, sometime it works and sometime we need to wait at least 1 hour before running it again. And we were struggling due to this issue. Then we found a useful good tool on APIFY that was beneficial to our situation.

APIFY is the platform where developers build, deploy, and monitor web scraping and browser automation tools, Figure (3.5). We have used Instagram Profile Scraper by Zuzka Pelechová, this great tool scrap 1,000 users at once and it takes less than five minutes for one run, all we needed is just usernames of the accounts, and we already have it thanks to Exporter for IG extension.

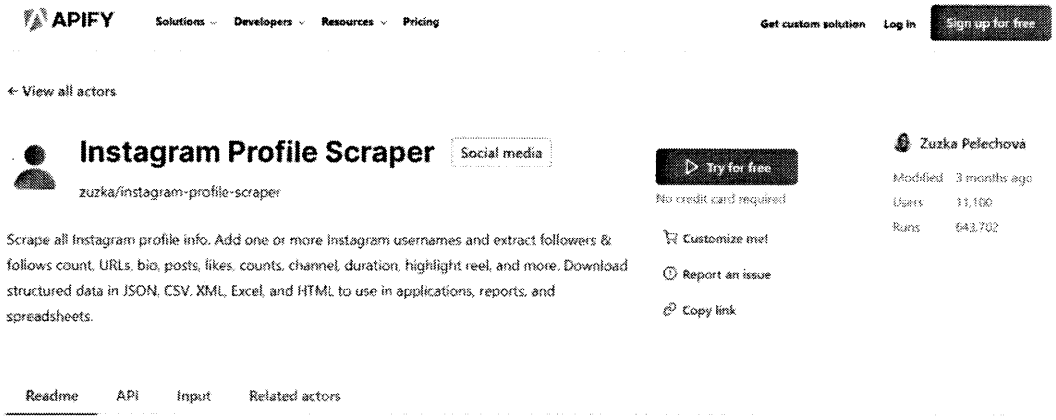


Figure (3.5): APIFY, Instagram Profile Scraper.

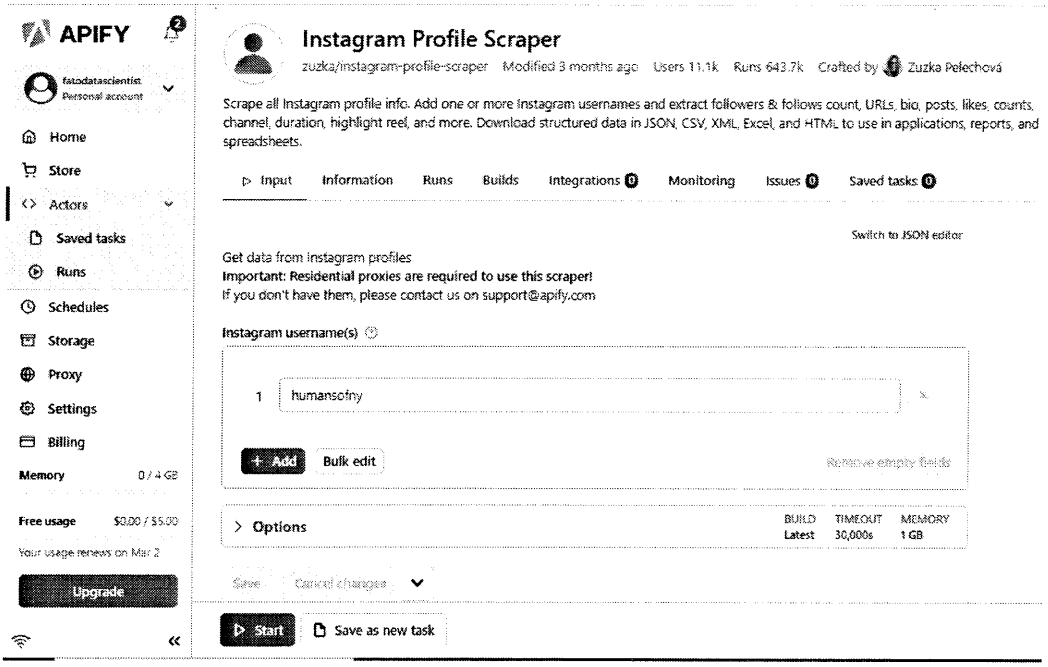


Figure (3.6): APIFY, Instagram Profile Scraper View.

After clicking Start Instagram Profile Scraper will start to run, Figure (3.6) and (3.7), when it completes we can Export the data gathered to an Excel file, Figure (3.8)

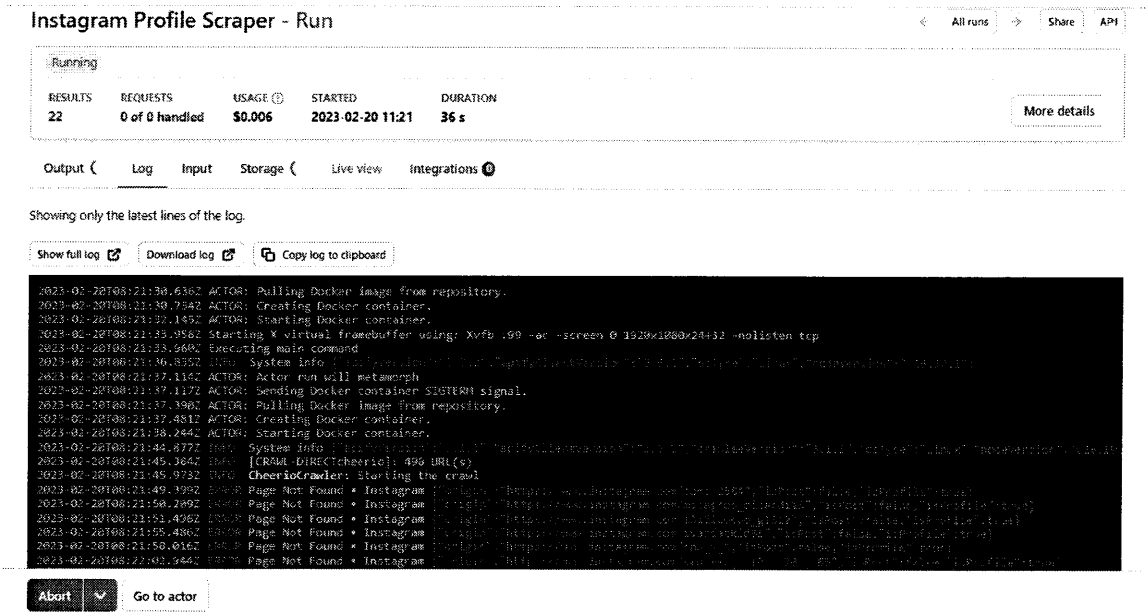


Figure (3.7): APIFY, Instagram Profile Scraper 1st run.

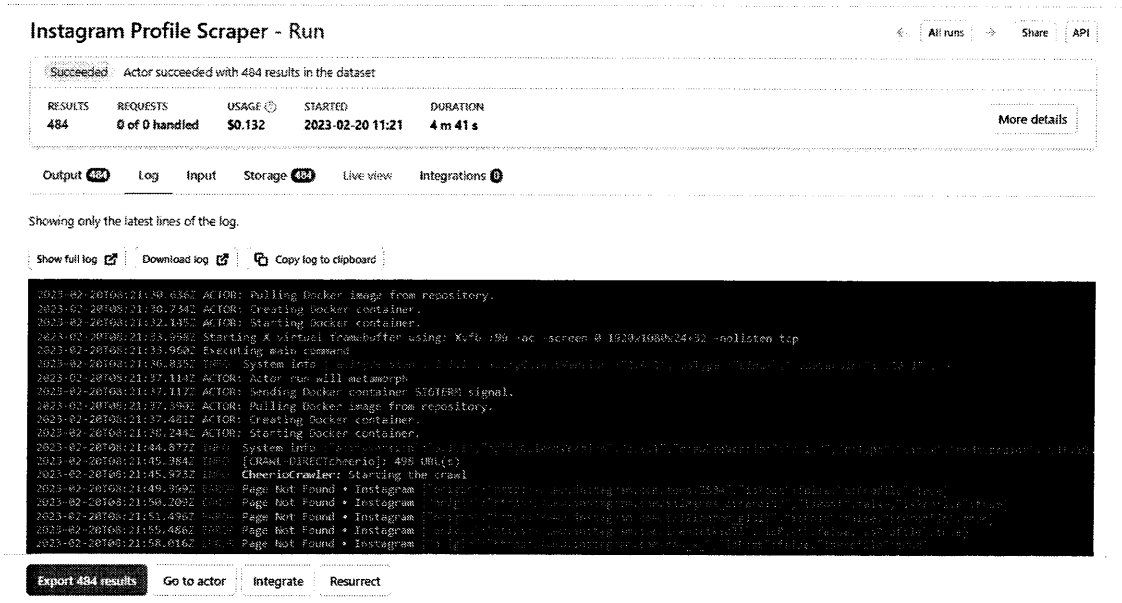


Figure (3.8): APIFY, Instagram Profile Scraper 2nd run.

After we have extracted all the data, we have chosen specific attributes and combined all data in one Excel file. The features chosen were biography (description bio), external URL (account's website), followers Count (number of followers), follows Count (number of followings), full name, highlight reel count, is business account, joined recently, posts count, private, username, and verified. After that we created our features from these attributes in Excel.

3.2.3. Features preparation:

We have 17 features total described below:

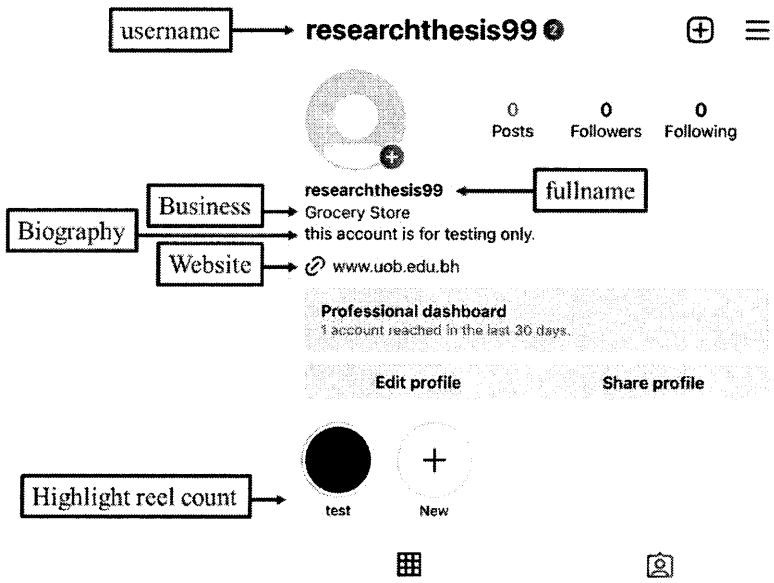


Figure (3.9): Instagram User details.

1. Followers.

   This feature indicates the number of followers, Figure (3.9).

2. Following.

   This feature indicates the number of followings, Figure (3.9).

3. Highlight reel count.

This feature indicates the number of highlight reel count, Figure (3.9).

4. is business?

   This feature indicates if this account run a business or not, Figure (3.9).

5. is joined recently?

   This feature indicates if the account user is new or not, Figure (3.10).

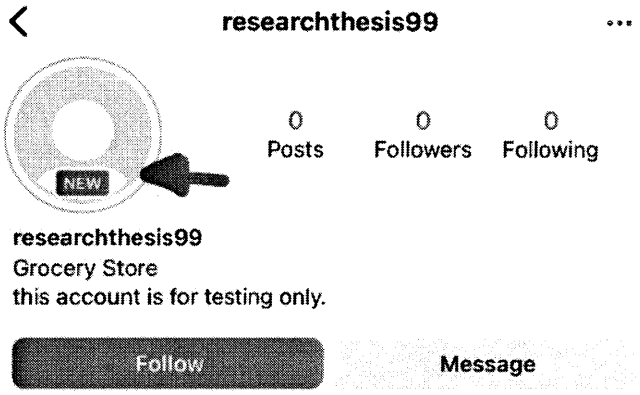


Figure (3.10): Instagram new user.

6. #posts.

   This feature indicates number of posts that user account has shared, Figure (3.9).

7. Is private?

   This feature indicates whether account is private or not, Figure (3.11).

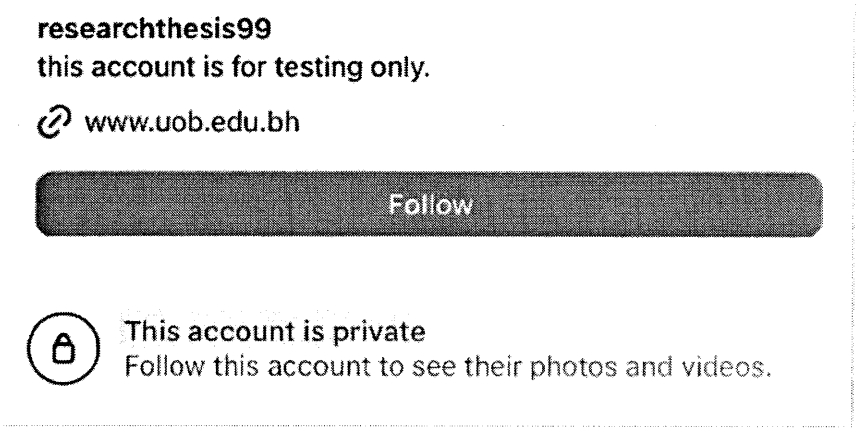


Figure (3.11): Instagram private user

8. Is verified?

   This feature indicates whether account is verified or not, Figure (3.12).



Figure (3.12): Instagram verified user.

9. Description length.

   Description length is the total length of the biography, Figure (3.9). We have applied the function **=LEN (*column#*)**, Figure (3.13). The function indicates the length of column A# which is the biography.

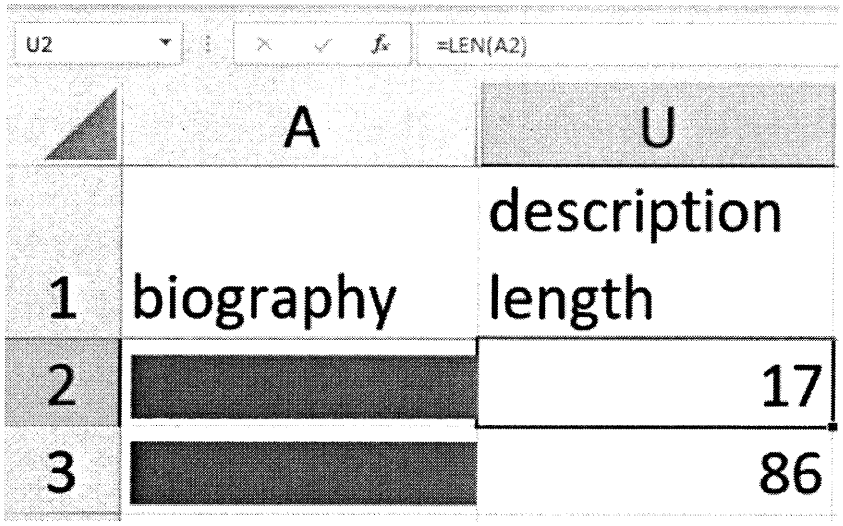


Figure (3.13): Description length function.

10. Has website.

"Has website" feature determine whether the account have website in their bio or not, it is binary, Figure (3.9). we have used the function **=IF (LEN (*column#*) = 0,0,1)**, this function is an if statement indicates that if the length of column B# is equal to 0 then the value of "has website" feature will be 0 which means it does not have website link, otherwise it will be 1, Figure (3.14). Column B is externalUrl.

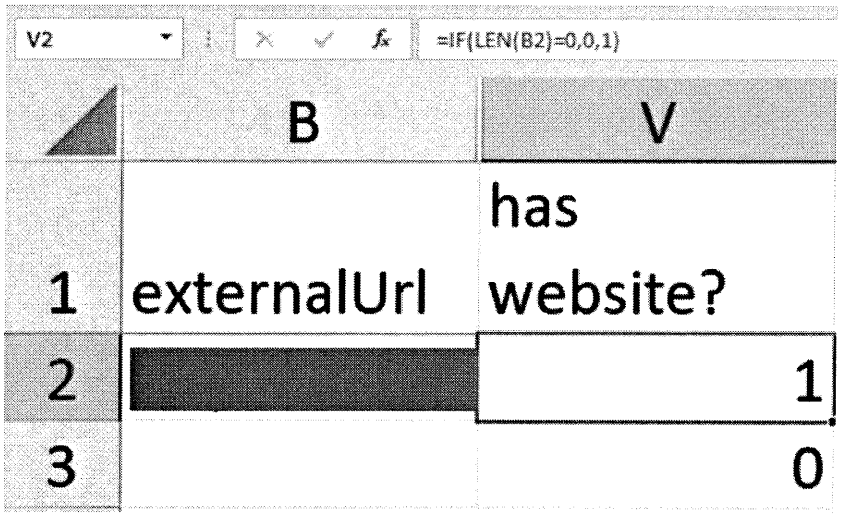


Figure (3.14): has website function.

11. Username==full name?

This feature determines if the username is equal to the full name or not, Figure (3.9). It is binary, we have used the function **=IF (*column#=column#*,1,0)**, it indicates that if username is equal to full name, then the value is 1, otherwise it will be 0, Figure (3.15). Column K# is username and column E# is the full name.

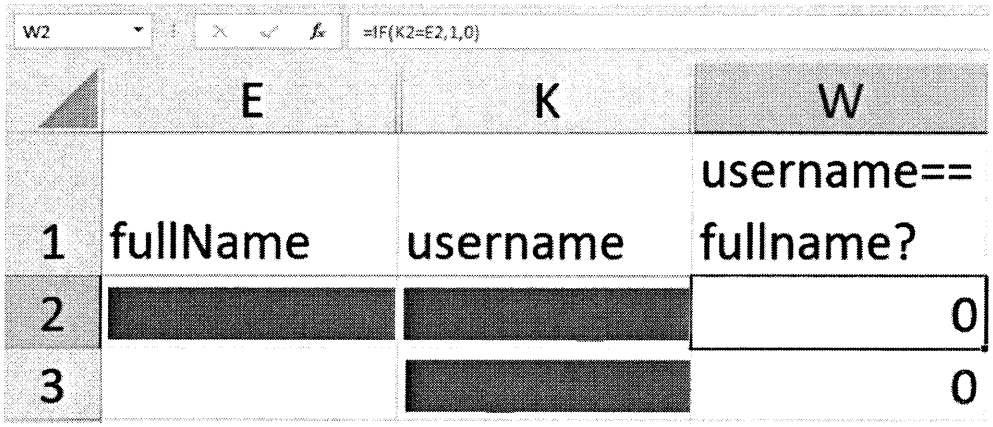


Figure (3.15): username==full name function.

12. #digits in username.

This feature gives us the number of digits in the username, Figure (3.9). We have used the function **=SUM(LEN(*column#*)-LEN(SUBSTITUTE(*column#*,{1,2,3,4,5,6,7,8,9,0},)))**, we will subtract the length of the username (without numbers) from the length of the entire username , and that will give us the number of digits in the username, column K# is username, Figure (3.16).

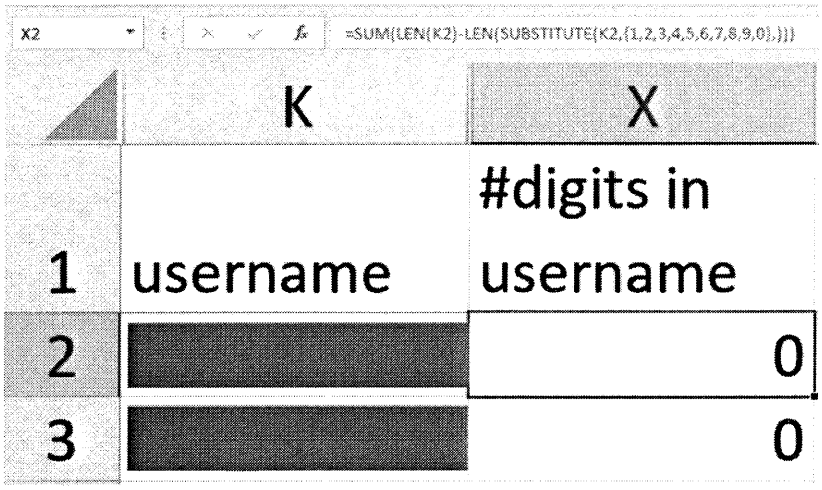


Figure (3.16): #digits in username function.

13. #digits in full name.

This feature gives us the number of digits in the full name, Figure (3.9). We have used the function **=SUM(LEN(*column#*)-LEN(SUBSTITUTE(*column#*,{1,2,3,4,5,6,7,8,9,0},)))**, we will subtract the length of the full name (without numbers) from the length of the entire full name , and that will give us the number of digits in the full name, column E# is full name, Figure (3.17).

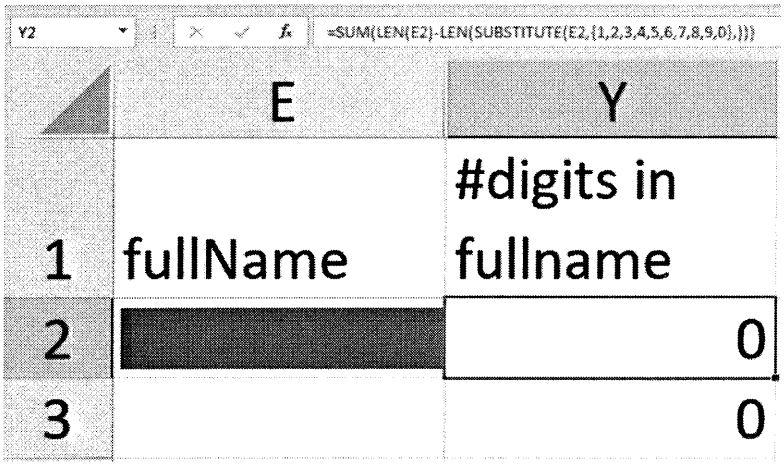


Figure (3.17): #digits in full name function.

14. Username has digit.

This feature determines if username has digits or not, it is binary, Figure (3.9). We have used the function **=IF (*column#*=0,0,1)**, it is an if statement indicates that if #digits in username is equal to 0 then value will be 0 otherwise it will be 1, column X# is #digits in username, Figure (3.18).

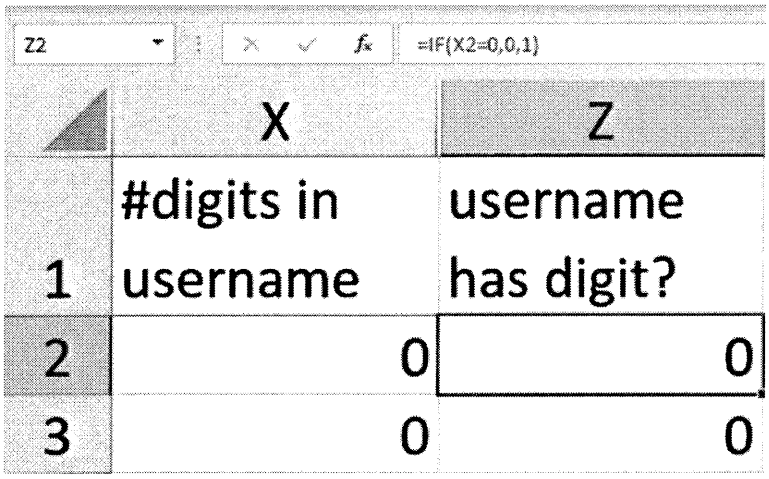


Figure (3.18): username has digits function.

15. Full name has digit?

This feature determines if full name has digits or not, it is binary, Figure (3.9). We have used the function **=IF (*column#*=0,0,1)**, it is an if statement indicates that if #digits in full name is equal to 0 then value will be 0 otherwise it will be 1, column Y# is #digits in full name, Figure (3.19).

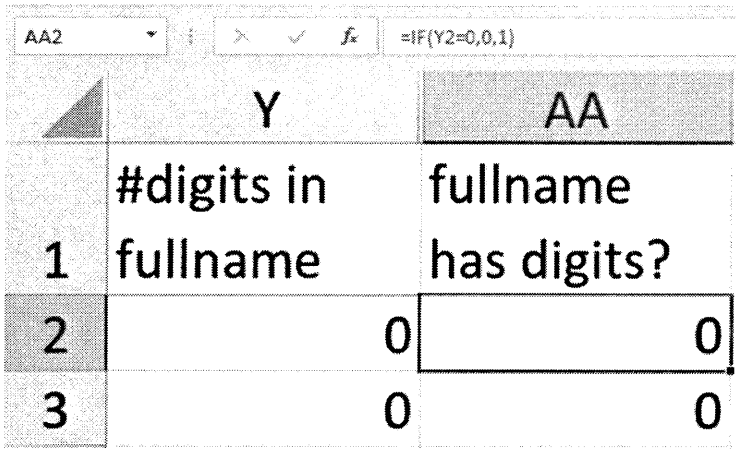


Figure (3.19): Full name has digits function.

16. Length of full name.

Length of full name is the total length of the full name, Figure (3.9). we have applied the function **=LEN (*column#*)**. The function indicates the length of column E# which is the full name, Figure (3.20).

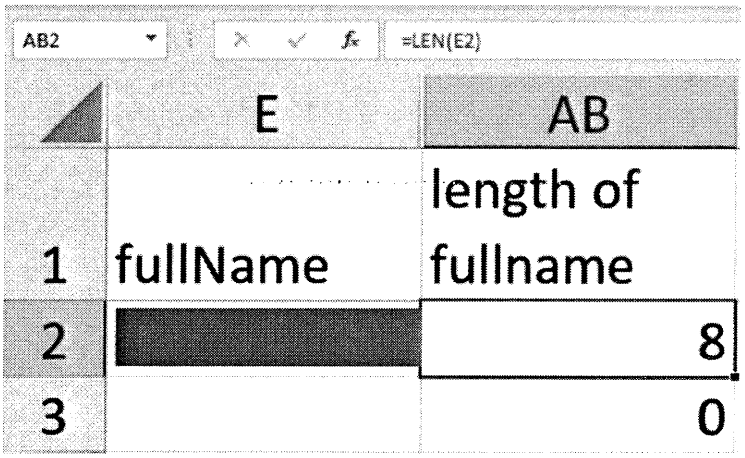


Figure (3.20): length of full name function.

17. Length of username:

Length of username is the total length of the username, Figure (3.9). we have applied the function **=LEN (*column#*)**. The function indicates the length of column K# which is the username, Figure (3.21).

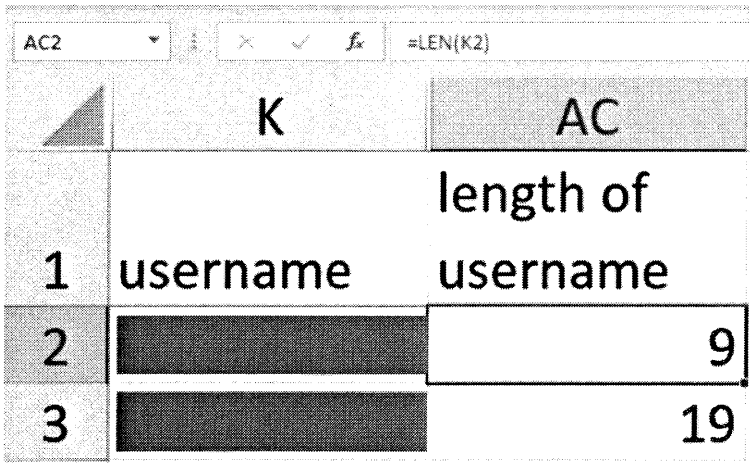


Figure (3.21): length of username function

18. Is Fake?

    This is the label in which it classifies the category of the user account whether it is real or fake.

We have a total of 17 features now with label, Figure (3.22).

S18

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	followers	following	highlight r	is buisnes	is joined r	#posts	is private	is verified	descriptio	has websi	username	#digits in	#digits in	username	fullname l	length of	length of	isFake
2	1611	457	0	1	0	52	0	0	17	1	0	0	0	0	0	8	9	3
3	1170	7135	0	0	0	9	1	0	86	0	0	0	0	0	0	0	19	1
4	260	319	0	0	0	78	0	0	108	1	0	0	0	0	0	12	12	3
5	862	1875	0	0	0	165	1	0	168	0	0	5	0	1	0	10	27	3
6	531	2320	0	0	0	1	0	0	0	0	0	2	0	1	0	0	11	3
7	0	2276	0	0	0	0	1	0	0	0	0	4	4	1	1	10	11	3
8	431	4183	0	0	0	0	0	0	0	0	0	0	0	0	0	19	18	2
9	209	2260	0	0	0	3	1	0	52	0	0	0	0	0	0	6	9	3

Figure (3.22): All Features of 3rd dataset.

3.2.4. K-mean clustering:

We only had the real and fake classifications after gathering and collecting the features, but our goal goes beyond binary classification, and as shown in Table (3.1), the data is not balanced and we need to reclassify the fake data, so we had to use k-means clustering.

Table (3.1): Real and fake accounts distribution.

Labels (is fake?)	
Real accounts	Fake accounts
15,942	67,919

K-means clustering aims to convert unlabeled data into groups and categories, and we have used the functions as described below:

1. First, we must define number of clusters we need, and we want to have 4 fake labels, so k=4.
2. After that we have to set 4 centroids, which are chosen randomly from the data and repeatedly chosen after calculations until convergence, but in our case, we have a numerous amount of dataset so it would take long time to figure out the best centroids, so we have decided to get the help from python library to find out the best centroid for us, code is described below:

We have used Jupiter notebook to run python code, we have used Pandas library to help us reading our dataset, also we have used "sklearn.cluster.KMeans" library to help us to do the clustering, the fake dataset contains 67918 columns and 17 features.

```
import pandas as pd
from sklearn.cluster import KMeans

data = r'C:\Users\LENOVO\Documents\College\Master\Thesis\dataset\final da-
tasets\fake data .csv'
df = pd.read_csv(data)

df.shape
(67918, 17)
```

We make sure of the dataset information; all datasets are of type integer.

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 67918 entries, 0 to 67917
Data columns (total 17 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   followers             67918 non-null  int64
 1   following             67918 non-null  int64
 2   highlight reel count  67918 non-null  int64
 3   is buisness?          67918 non-null  int64
 4   is joined recently?   67918 non-null  int64
 5   #posts                67918 non-null  int64
 6   is private?           67918 non-null  int64
 7   is verified?          67918 non-null  int64
 8   description length    67918 non-null  int64
 9   has website?          67918 non-null  int64
 10  username==fullname?   67918 non-null  int64
 11  #digits in username   67918 non-null  int64
 12  #digits in fullname   67918 non-null  int64
 13  username has digit?   67918 non-null  int64
 14  fullname has digits?  67918 non-null  int64
 15  length of fullname    67918 non-null  int64
 16  length of username    67918 non-null  int64
dtypes: int64(17)
memory usage: 8.8 MB
```

We make sure there is no null values in our dataset.

```
df.isnull().sum()
followers              0
following              0
highlight reel count   0
is buisness?           0
is joined recently?    0
#posts                 0
is private?            0
```

```
is verified?            0
description length      0
has website?            0
username==fullname?     0
#digits in username     0
#digits in fullname     0
username has digit?     0
fullname has digits?    0
length of fullname      0
length of username      0
dtype: int64
```

We run the k-mean choosing 4 clusters, k=4, and we have taken the centroid values as shown below.

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4, random_state=0)
kmeans.fit(df)
KMeans(n_clusters=4, random_state=0)

kmeans.cluster_centers_
array([[3.41902773e+01, 7.26661239e+03, 1.66870724e-01, 5.68959309e-03,
        0.00000000e+00, 1.72629456e+01, 2.01656464e-03, 0.00000000e+00,
        9.58343536e+00, 2.52070580e-03, 7.25963270e-02, 3.32999640e+00,
        4.01008282e-01, 9.00540151e-01, 1.12495499e-01, 5.04796543e+00,
        1.15292762e+01],
       [2.06537713e+01, 4.00741796e+03, 8.41303476e-02, 2.66307772e-03,
        0.00000000e+00, 1.63074566e+01, 1.11677453e-03, 0.00000000e+00,
        6.95925205e+00, 1.77538514e-03, 6.90682092e-02, 3.82435141e+00,
        4.15984193e-01, 9.30072734e-01, 1.03230056e-01, 4.26728137e+00,
        1.18551629e+01],
       [2.06561495e+01, 2.76465702e+03, 1.05666914e-01, 3.34075724e-03,
        0.00000000e+00, 1.84565702e+01, 4.57807473e-03, 0.00000000e+00,
        4.79052215e+00, 2.10343974e-03, 4.02128186e-02, 3.29175947e+00,
        2.34471665e-01, 8.64761198e-01, 5.98861668e-02, 3.84793368e+00,
        1.29309577e+01],
       [1.97590897e+01, 5.37354892e+03, 5.94795539e-02, 1.54139088e-03,
        0.00000000e+00, 1.94560704e+01, 1.54139088e-03, 0.00000000e+00,
        1.74993200e+00, 7.25360413e-04, 1.25124671e-02, 2.21117055e+00,
        7.07226403e-02, 6.63614108e-01, 1.91313809e-02, 5.51754466e+00,
        1.37064104e+01]])
```

We have added a new column to our dataset which is considered the labeling column, and we have stored the dataset in a file.

```
kmeans.labels_
array([2, 0, 2, ..., 1, 1, 1])

df["clusters"] = kmeans.labels_
df.head()

df.to_csv(r'C:\Users\LENOVO\Documents\College\Master\Thesis\clustered data.csv',
index=False)
```

To elaborate more in this step, we will explain the calculation of k means clustering, before starting we must determine the k value which in our case is 4, then after that we must determine 4 center values, and as mentioned earlier we have got our centers from python, Table (3.3). After that we must calculate the distance between our data points and the center values, there are various distance functions used with k-means, the most famous one is Euclidean distance, but we are going to use Manhattan distance, Formula (3.1).

$$|X - Y| = \sum_{i=1}^{k} |x_i - y_i| \quad (3.1)$$

We are going to apply the equation on a random data point from our dataset, so we are going to choose the row number 374 data points, Table (3.2).

Table (3.2): Data points for row number 374.

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
xi	30	3990	0	0	0	17	0	0	0	0	0	4	0	1	0	0	11	2

Now our data points are labeled as 2 which means fake2 labeling, when we apply the Manhattan distance, we are going to have 4 centers, and we need to choose only one to determine the label, our Manhattan Formula (3.2).

$$L = |X - C| = \sum_{i=1}^{k} |x_i - c_i| \quad (3.2)$$

$$L1 = |30 - 34.19| + |3990 - 7266.61| + |0 - 0.17| + |0 - 0.01| + |0 - 0| + |17 - 17.26| + |0 - 0| + |0 - 0| + |0 - 9.58| + |0 - 0| + |0 - 0.07| + |4 - 3.33| + |0 - 0.4| + |1 - 0.9| + |0 - 0.11| + |0 - 5.05| + |11 - 11.53| = 3297.75$$

Applying Manhattan distance for all centroids give us L1=3297.75, L2=40.47, L3=1248, and L4=1408.52, in this case the lower center is center 2 so the label will be fake2. The rest of datapoints are labeled the same way.

Table (3.3): Centroids.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
C1	34.19	7266.61	0.17	0.01	0	17.26	0	0	9.58	0	0.07	3.33	0.4	0.9	0.11	5.05	11.53
C2	20.65	4007.42	0.08	0	0	16.31	0	0	6.96	0	0.07	3.82	0.42	0.93	0.1	4.27	11.86
C3	20.66	2764.66	0.11	0	0	18.46	0	0	4.79	0	0.04	3.29	0.23	0.86	0.06	3.85	12.93
C4	19.76	5373.55	0.06	0	0	19.46	0	0	1.75	0	0.01	2.21	0.07	0.66	0.02	5.52	13.71

3.2.5. Features analyzing:

In this section, we'll show how specific features relate to labels to comprehend the behavior of fake accounts. The first characteristic is the number of followers; as the Figure (3.23) shows, real accounts are more likely to have a significant number of followers than do fake accounts. Based on how many individuals the account is following, fake accounts were categorized, Figure (3.24).

Genuine accounts can have up to 500 highlight reel counts; real users are more likely to have those numbers, while fake accounts typically have less than 100, Figure (3.25). While fake users often have fewer than 5,000 posts, real users can have up to 35,000 posts, Figure (3.26).

Compared to fake users, real users are more likely to have private accounts, Table (3.4). Only legitimate users may have verified accounts, Table (3.5), fake users are not permitted to do so. Real users are nonetheless more likely to have a business account than fake users, Table (3.6). Although fake users might include their website in their profile, real users are more likely to have a website, Table (3.7).

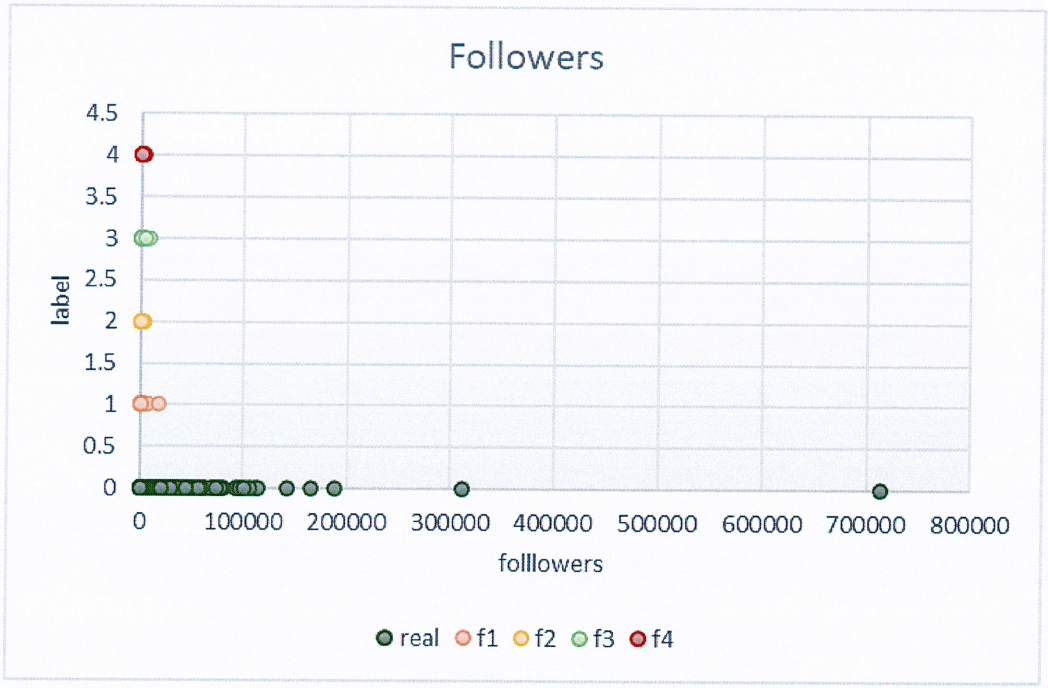


Figure (3.23): label vs Followers.

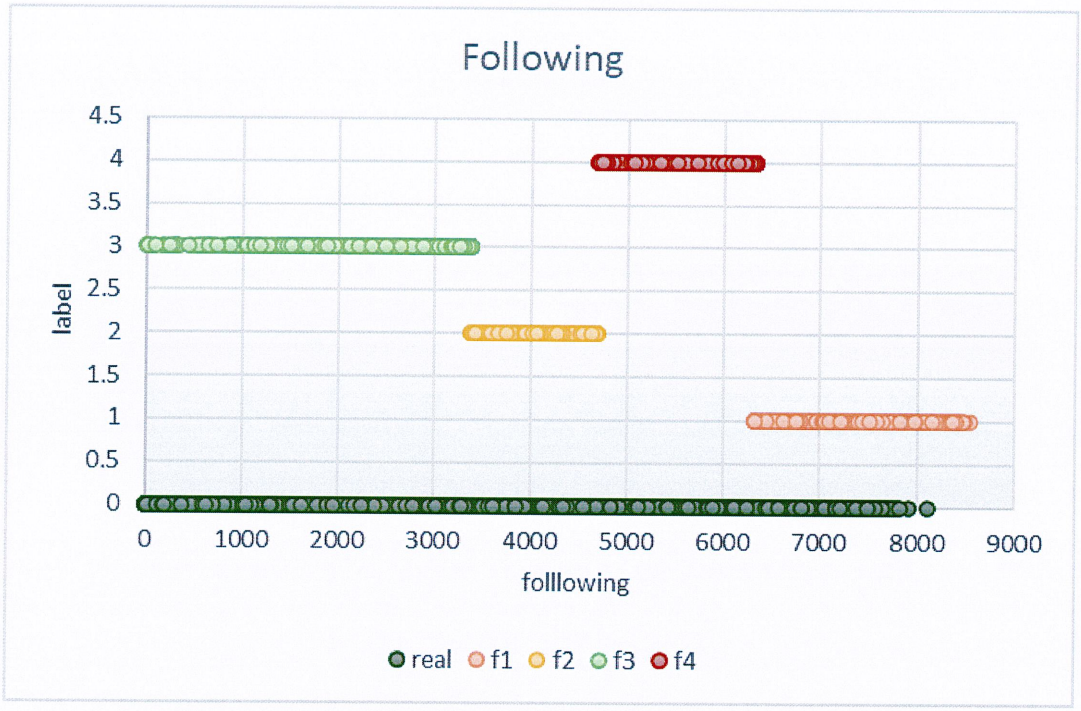


Figure (3.24): labels vs Following.

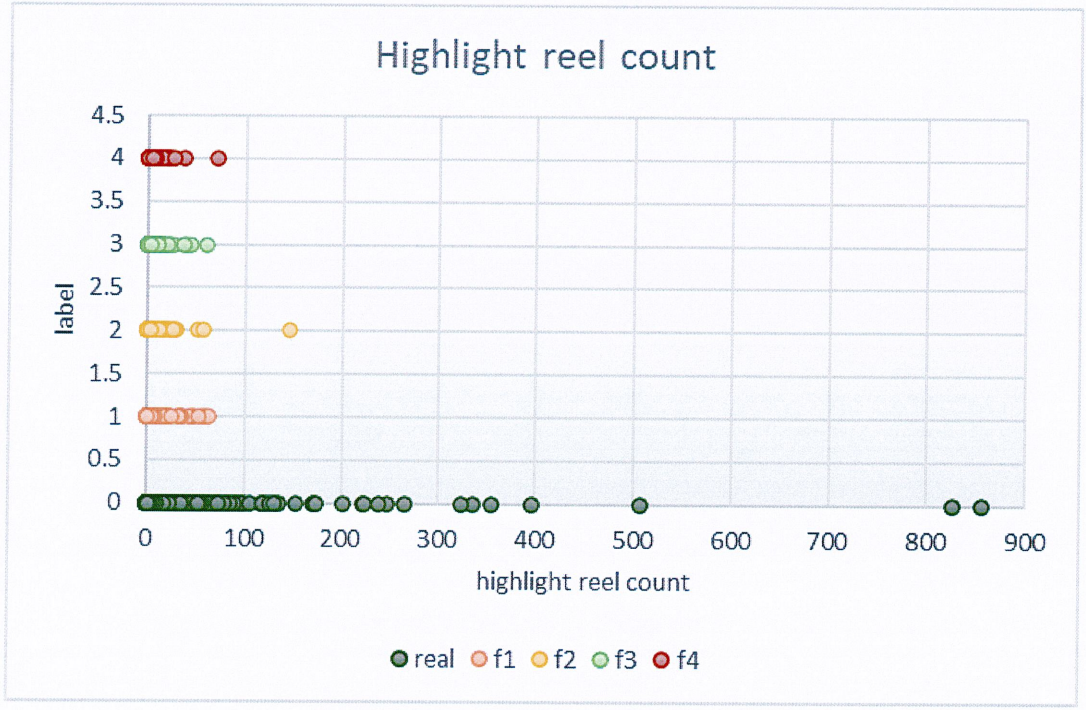


Figure (3.25): labels vs Highlight reel count.

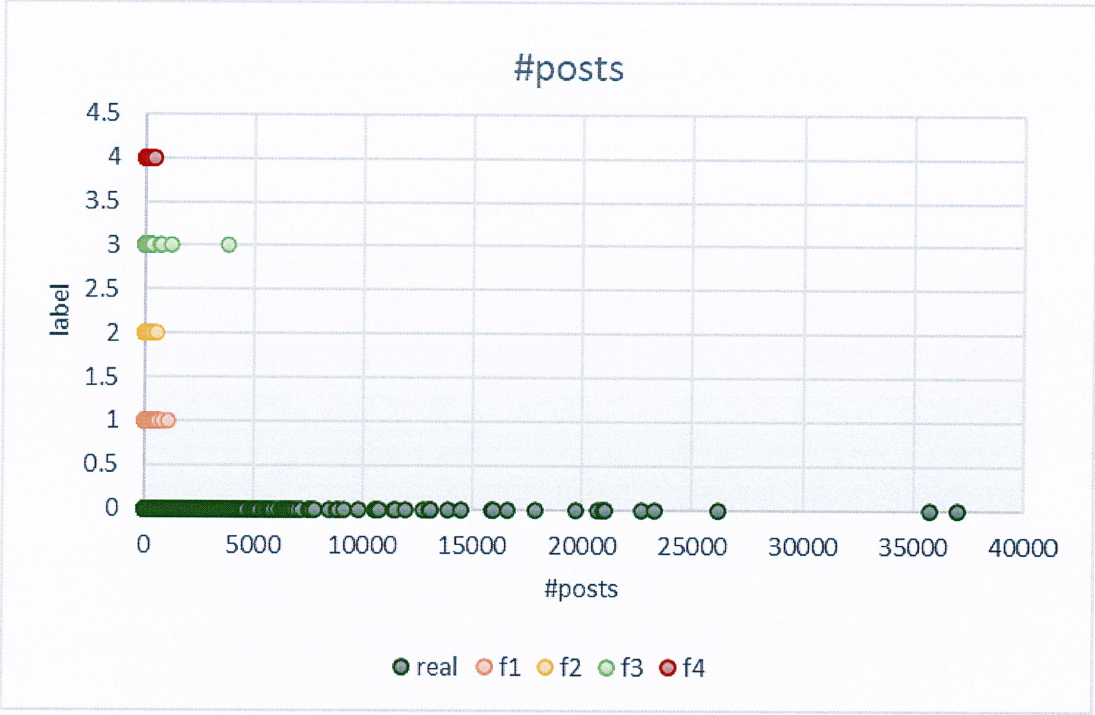


Figure (3.26): label vs #posts.

Table (3.4): is private count for each label.

is private?	0	1	2	3	4	Grand Total
0	7456	13857	34881	8047	11012	75253
1	8487	28	39	37	17	8608
Grand Total	**15943**	**13885**	**34920**	**8084**	**11029**	**83861**

Table (3.5): is verified count for each label.

is verified?	0	1	2	3	4	Grand Total
0	15571	13885	34920	8084	11029	83489
1	372					372
Grand Total	**15943**	**13885**	**34920**	**8084**	**11029**	**83861**

Table (3.6): is business count for each label.

is business?	0	1	2	3	4	Grand Total
0	14338	13806	34827	8057	11012	82040
1	1605	79	93	27	17	1821
Grand Total	**15943**	**13885**	**34920**	**8084**	**11029**	**83861**

Table (3.7): has website count for each label.

has website?	0	1	2	3	4	Grand Total
0	14334	13850	34858	8067	11021	82130
1	1609	35	62	17	8	1731
Grand Total	**15943**	**13885**	**34920**	**8084**	**11029**	**83861**

Table (3.8): #posts average for each label.

Is Fake	0	1	2	3	4	Grand Total
Average of #posts	171.72	17.26	16.30	18.45	19.45	**46.63**

From the feature analyzing of classes, we have noted the behavior of fake accounts. The fake1 class shows that this kind of accounts tend to have large number of followings, Figure (3.24), and the purpose from following many accounts, is to get the followers to reach the fame, so fake1 is the fake celebrity account. The fake2 class shows that it usually uses link in the bio, Table (3.7), the link can spread malwares or fool innocent people to share personal information, so fake2 is the spam account. The fake3 class shows that it has less followings, Figure (3.24) , this account's purpose is to threaten people, cyberbullying, and target individuals in direct messages, so fake3 is the attacker account. The fake4 class shows that it has more average number of posts than other fake classes, Table (3.8), so it can promote specific idea, spread misinformation of data or to manipulate public opinion on a specific topic, so it is the bot account.

3.3. First stage

For classifying real from fake accounts, this proposed work employs techniques such as artificial neural networks (ANN), K nearest neighbor (KNN), Random Forest (RF), and support vector machine (SVM). The feature set that affects Instagram's ability to identify fake accounts will be applied. This work is anticipated to produce the higher accuracy values needed for the detection of fake Instagram accounts. In this stage, machine learning techniques' accuracies will be compared.

3.3.1. System architecture:

Three different Instagram datasets were used in this stage. All datasets were fed into the system for pre-processing, data splitting into training and test sets, and then a machine learning model was applied. In this stage, we are using four machine learning models: SVM, KNN, ANN, and RF. We will compare and demonstrate the results. Figure (3.27) illustrate the System Architecture.

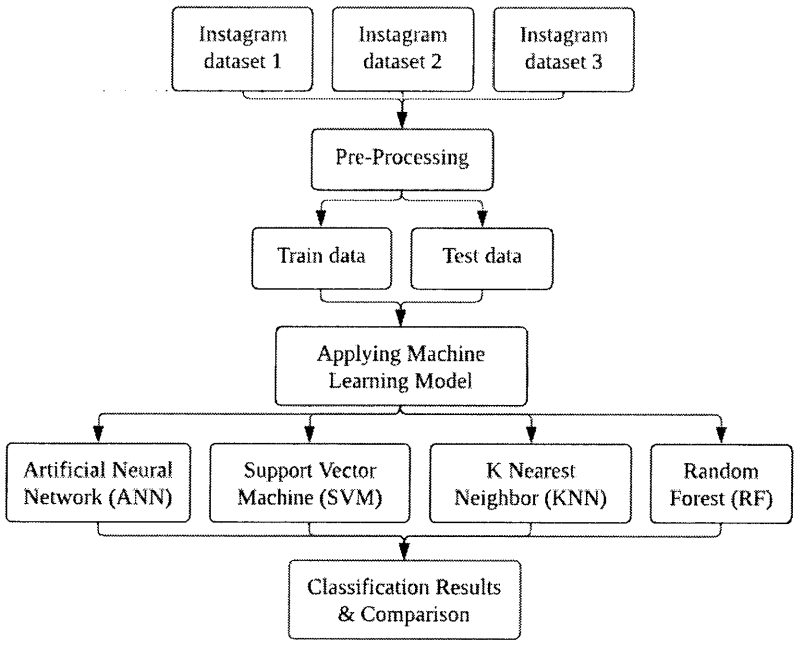


Figure (3.27): System Architecture.

3.3.2. System implementation:

In this section the code will be explained using the 1st dataset. The same code is applied for the 2nd and 3rd datasets, and results are discussed in chapter four. For the machine learning models, we have imported four main libraries that we will need in the implementation, one of the most frequently used Python scientific computing packages is **NumPy**. In addition to modifications like masks and matrices, it offers a multidimensional array object that may be utilized for a variety of mathematical operations (M, 2022). To make data intelligible, **Matplotlib** offers users the ability to visualize data using a range of various types of plots. It employs a variety of plot formats, including scatterplots, histograms, bar charts, error charts, and boxplots, among others (Kumar, 2022a).

Pandas is built on two critical Python libraries: NumPy for mathematical operations and Matplotlib for data visualization. Pandas facilitates access to various matplotlib and NumPy methods by acting as a wrapper for these libraries (Mode, 2016). Seaborn is a matplotlib-based open-source Python library. It is used to explore and visualize data. Seaborn operates with ease owing to data frames and the Pandas library. Furthermore, the generated graphs may be effortlessly modified (Banerjee, 2021).

```
# linear algebra
import numpy as np
# data processing, CSV file I/O
import pandas as pd
# for data visualization
import matplotlib.pyplot as plt
# for statistical data visualization
import seaborn as sns
```

Certainly, on building or running the code, we would encounter problems and warnings. Hence, in Python, we must disable warnings. There are times when a Python programmer must utilize a specific filter function of the warnings module to suppress or ignore warnings. When it's required to notify a user of a condition in their code, a programmer will deliver a warning message in Python. But ordinarily this circumstance does not result in the application being terminated and an exception being raised (Patel, 2022).

```
import warnings
warnings.filterwarnings('ignore')
```

The 1st dataset has 13 features including the label, and 785 datapoints, the data we have here is not balanced as mentioned earlier, in which we have 93 (12%) real users and 692 (88%) fake users.

```
data1 = 'final-v1.csv'
FirstData = pd.read_csv(data1)

# view dimensions of dataset
FirstData.shape
(785, 13)

# preview first 5 datapoints
FirstData.head()

# view the column names of the dataframe
col_names = FirstData.columns
print(col_names)
Index(['edge_followed_by', 'edge_follow', 'username_length',
'username_has_number', 'full_name_has_number', 'full_name_length', 'is_private',
'is_joined_recently', 'has_channel', 'is_business_account', 'has_guides',
'has_external_url', 'is_fake'], dtype='object')

# check distribution of is_fake column
FirstData['is_fake'].value_counts()
1       692
0        93
Name: is_fake, dtype: int64

# view the percentage distribution of is_fake column
FirstData['is_fake'].value_counts()/np.float(len(FirstData))
1       0.881529
0       0.118471
Name: is_fake, dtype: float64
```

All the datapoints are numerical values, all features are of type integer/float.

```
# view summary of dataset
FirstData.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 785 entries, 0 to 784
Data columns (total 13 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   edge_followed_by     785 non-null    float64
 1   edge_follow          785 non-null    float64
 2   username_length      785 non-null    int64
 3   username_has_number  785 non-null    int64
 4   full_name_has_number 785 non-null    int64
 5   full_name_length     785 non-null    int64
 6   is_private           785 non-null    int64
 7   is_joined_recently   785 non-null    int64
 8   has_channel          785 non-null    int64
 9   is_business_account  785 non-null    int64
 10  has_guides           785 non-null    int64
```

```
 11  has_external_url      785 non-null    int64
 12  is_fake               785 non-null    int64
dtypes: float64(2), int64(11)
memory usage: 79.9 KB
```

One of the important steps is checking if there are null values. Absence of observations in columns is referred to as missing data. Values like "0," "NA," "Nan," "NULL," "Not Applicable," and "None" are examples of its values. The reason of it might be due to data corruption, failure to capture data, a lack of information, incomplete results, the fact that a person may not have supplied the data knowingly, a system or piece of equipment failure, etc. Any number of factors might cause the dataset to include missing values. One of the major effects of missing data is that it can impair the model's accuracy or skew the outcomes of machine learning algorithms. Thus, handling missing values is crucial (Nawale, 2022).

```
# check for missing values in variables
FirstData.isnull().sum()
edge_followed_by        0
edge_follow             0
username_length         0
username_has_number     0
full_name_has_number    0
full_name_length        0
is_private              0
is_joined_recently      0
has_channel             0
is_business_account     0
has_guides              0
has_external_url        0
is_fake                 0
dtype: int64
```

To continue we had to determine the value of X and y, X will be the features, and y is the labeling, a correlation map for the features is maintained to study the relationship between them. The strength of links between numerical variables can be seen using a particular form of graphic called a correlation heatmap. Understanding which variables are related to one another and the strength of this relationship requires knowledge of

correlation graphs. Formula (3.3) demonstrates how the correlation is calculated between features.

$$r_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}} \quad (3.3)$$

whereas,

r_{xy}, the correlation coefficient of the linear link among x and y.

x_i, the x-feature values in a sample.

$\bar{x}$, the mean of the x-feature's values.

y_i, the y-feature values in a sample.

$\bar{y}$, the mean of the y-feature's values.

In a correlation plot, each numerical variable is represented by a column and includes several numerical variables. Each pair of relationships between the variables is shown as a row. The values in the cells represent the strength of the association; positive values represent a positive relationship, while negative values represent a negative relationship. It is possible to identify potential links between variables and gauge their strength by using correlation heatmaps (Kumar, 2022b).

Figure (3.28), (3.29), (3.30) explains the scatter plot for positive correlation, negative correlation, and no correlation between features (Khan Academy, n.d.).

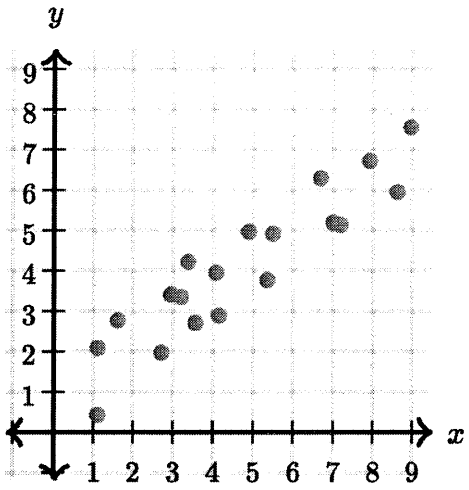


Figure (3.28): Positive correlation between variables, (Khan Academy, n.d.).

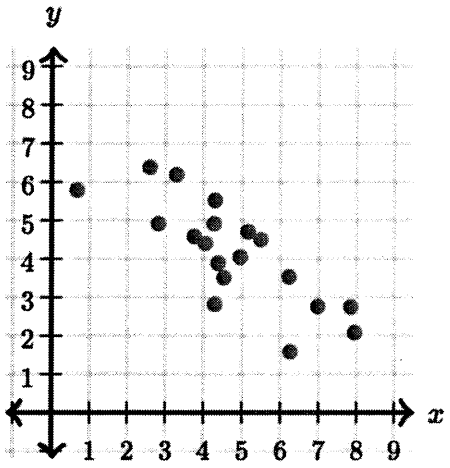


Figure (3.29): Negative correlation between variables, (Khan Academy, n.d.).

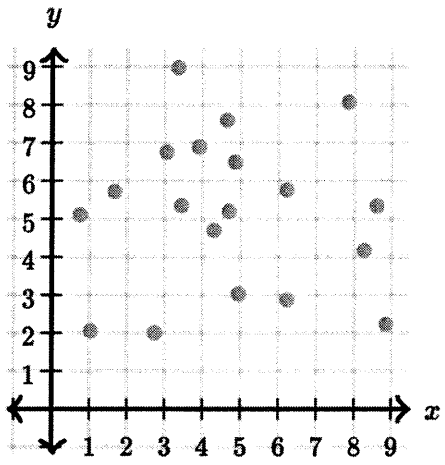


Figure (3.30): No correlation between variables, (Khan Academy, n.d.).

```
X = FirstData.drop(['is_fake'], axis=1)
y = FirstData['is_fake']
# view feature distribution
ax = sns.countplot(y,label="Count")
F, R = y.value_counts()
print('Number of Fake: ',F)
print('Number of Real : ',R)
Number of Fake:  692
Number of Real :  93
# describtion of data
X.describe()
#correlation map
f,ax = plt.subplots(figsize=(18, 18))
sns.heatmap(X.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
```

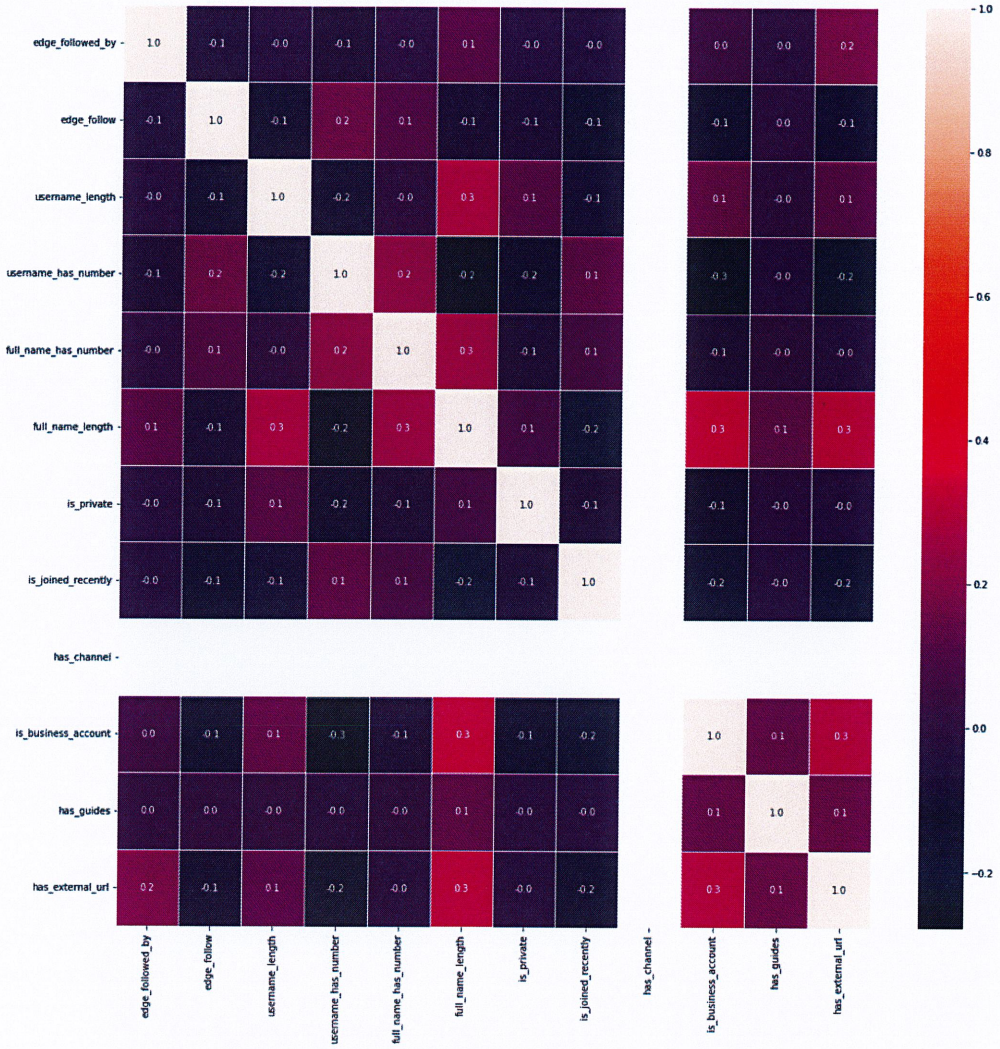


Figure (3.31): Correlation map for 1st dataset.

Feature "has channel" row/column is empty, this happens when the column has equal values, that will not benefit in training, so we must remove this feature.

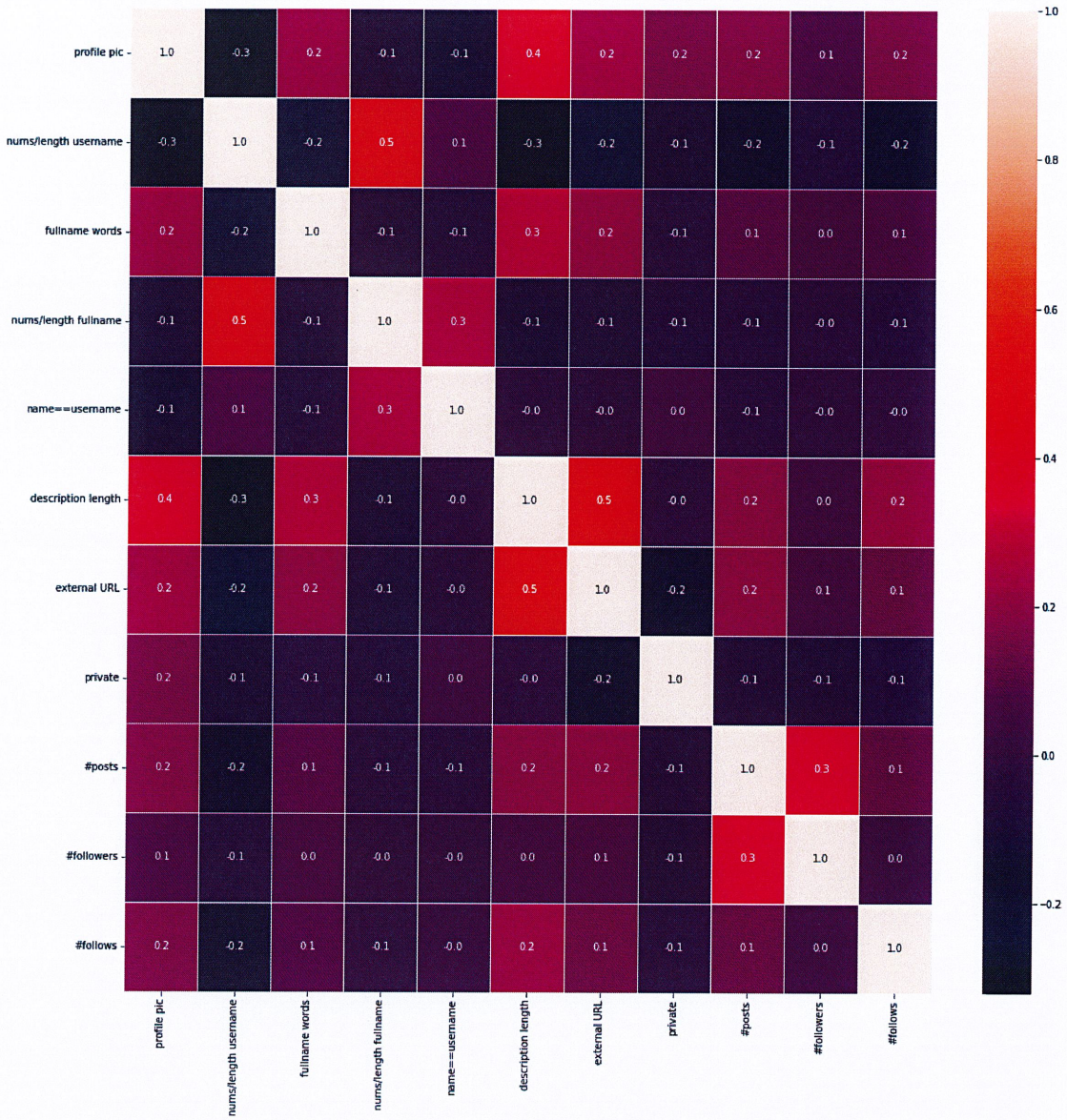


Figure (3.32): Correlation map for 2nd dataset.

Feature "nums/length username" has a correlation value of 0.5 with the feature "nums/length full name", and feature "description length" also has a correlation of 0.5 with "externalUrl". We kept them and no features were removed, since the correlation is considered weak to moderate.

From the correlation map, Figure (3.31), it is observed that "has channel" feature row/column is empty, and this happens when the column has equal values, that will not benefit in training, so removing this feature is the best decision we take in this step. For the second dataset, Figure (3.32) we didn't removed any feature.

Next step is splitting the dataset into training and testing sets, so we chose to set 20% of the dataset as testing data and the 80% will be assigned for training. Next, we applied feature scaling on the data, when features are on a similar scale and/or nearly normally distributed, several machine learning methods work better or converge more quickly.

Applying feature scaling aims to ensure that features are almost on the same scale, making each feature equally relevant and making it simpler for most ML algorithms to handle. Standard Scaler scales a feature to unit variance after subtracting the mean to standardize it. By the standard deviation, we may calculate the unit variance of all the values. When the data is distributed according to a Gaussian curve, standardization can be useful (Upadhyay, 2020).

```
X['has_channel'].value_counts()
0    785
Name: has_channel, dtype: int64

drop_list1 = ['has_channel']
x1 = X.drop(drop_list1,axis=1)
x1.head()

# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x1, y, test_size = 0.2,
random_state = 0)
```

```
# shape of X_train and X_test
X_train.shape, X_test.shape

cols = X_train.columns

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_train = pd.DataFrame(X_train, columns=[cols])

X_test = pd.DataFrame(X_test, columns=[cols])

X_train.describe()
```

Standard scores, also known as (z) scores, of the samples is computed as follows:

$$\boldsymbol{z} = \frac{x-\mu}{\sigma} \quad (3.4)$$

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i \quad (3.5)$$

$$\boldsymbol{\sigma} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \quad (3.6)$$

whereas,

μ, the mean (average).

σ, the standard deviation.

x, the value.

N, the total number of values.

The training data is scaled, and its scaling parameters are learned by applying fit_transform () to the training data. The model will discover the training set's feature mean and variance. The test data is then scaled using these newly learned parameters.

Figure (3.33) demonstrates the normal distribution in red line in which we have a standard deviation of 1 and a mean of 0.

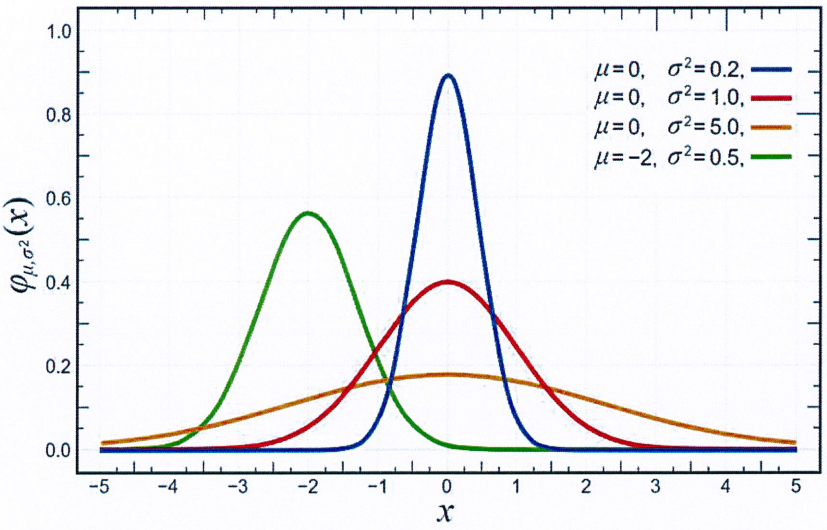


Figure (3.33): Gaussian Distribution, (Upadhyay, 2020).

The mean and variance derived from the training data can be used to transform the test data using the transform () method. We may therefore convert the test data using the parameters that the model learned from the training set (Khanna, 2020).

If the fit method is utilized to the test data as well, we will compute a new mean and variance, which is a new scale for each feature, and we will allow the model to pick up on the information from the test data as well. As a result, the model will no longer be unaware of what we want to maintain a surprise, and we will not be able to determine how well the model is performing on test data, which is the overarching goal of developing a model using a machine learning method (Khanna, 2020).

Support Vector Machine (SVM):

Support vector machines (SVMs) are a type of supervised learning algorithms used to solve classification, and regression problems. This method maps the data in a massively dimensional input space and then creates a perfect disjointed hyperplane within it. The most common

software problem is quadratic, however for neural network designs, an inclination tutoring technique degrades from the authenticity of most native minima. An SVM model is just a hyperplane in multidimensional space that represents several classes. SVM will generate the hyperplane in an iterative manner to reduce error. SVM aims to classify datasets to find a maximum marginal hyperplane (MMH) (Biyani, 2021).

SVMs is expressed in the following formula (Marius, 2021):

$$\min \frac{1}{2} w^T w + c \sum_{i=1}^{n} \zeta_i \quad (3.7)$$

$$y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i \quad (3.8)$$

whereas,

ζ_i is the distance to the correct margin.

c is the regularization parameter.

$w^T w$ is the normal vector.

$\phi(x_i)$ is the transformed space vector.

b is the bias parameter.

y_i is the i-th target value.

By maximizing the margin from the Support Vectors to the hyperplane and decreasing the term $w^T w$, the goal is to correctly categorize as many data points as feasible. In other words, the goal can alternatively be defined as determining the best w and b such that most samples are accurately predicted. Most of the time, not all data points can be correctly distributed so that ζ_i represents the distance to the correct margin. A line drawn by the normal vector passes through the coordinate origin. This orthogonal straight line is sliced by the hyperplanes at a distance of $\frac{b}{\|w\|_2}$ from the origin. The regularization parameter c determines the severity of the penalty for the

number of data points that were incorrectly assigned with a total distance of $\sum_{i=1}^{n} \zeta_i$ (Marius, 2021).

The optimization work can be thought of as a dual problem because it seeks to maximize the margin while minimizing the parameters. Lagrange multipliers are used to solve the dual issue (Marius, 2021).

$$\boldsymbol{L(w,b,n)} = \frac{1}{2}\|w\|^2 - \sum_{n=1}^{N} a_n\{y_n(w^T\phi(x_n) + b) - 1\} \quad \textbf{(3.9)}$$

a represents the Lagrange multipliers, b is the bias parameter, w is the normal vector.

By utilizing the following two conditions:

$$\boldsymbol{w} = \sum_{i=1}^{N} a_i y_i \phi(x_i) \quad \textbf{(3.10)}$$

$$\boldsymbol{0} = \sum_{i=1}^{N} a_i y_i \quad \textbf{(3.11)}$$

We can remove w and b from L (w, b, a).

$$\boldsymbol{L(a)} = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N} a_n a_m y_n y_m k(x_n, x_m) \quad \textbf{(3.12)}$$

With the constraints $a_n \geq 0$ and $\sum_{n=1}^{N} a_n t_n = 0$ with the kernel function $k(x_n, x_m)$. To solve the optimization problem, new data points can be classified by using:

$$\sum_{n=1}^{N} a_n y_n k(x_n, x_m) \quad \textbf{(3.13)}$$

The kernel function can be filled with one of the following:

1. Linear function:

$$k(x_i, x_j) = x_i * x_j \quad (3.14)$$

2. Polynomial Function

$$k(x_i, x_j) = (1 + x_i * x_j)^d \quad (3.15)$$

3. Radial Basis Function (RBF)

$$k(x_i, x_j) = \exp(-\gamma \|X_i - X_j\|^2) \quad (3.16)$$

4. Sigmoid Function

$$k(x_i, x_j) = \tanh(\alpha x^T y + c) \quad (3.17)$$

A component of Scikit-Learn is the SVM library, which has built-in classes for several SVM algorithms. For carrying out a classification job, the support vector classifier class, also known as SVC in the Scikit-Learn SVM package, is utilized. All input for this class is of the kernel type. The parameter for a basic SVM will be "linear," as this type of SVM can only categorize data that can be separated along linear axes. The SVC class's fit method is called to supply the training data as a parameter to the fit function, which is used to train the algorithm on (Malik, 2022).

Therefore, a straight line cannot be employed as a decision boundary when dealing with non-linearly separable data. In essence, the kernel SVM assigns data points from various classes to various dimensions by projecting non-linearly separable data from lower dimensions to linearly separable data in higher dimensions. To implement and use the kernel SVM, we used the Scikit-Learn package for Python (Malik, 2022).

SVM with all types of kernels was implemented for all datasets, and a comparison of results was discussed in Chapter 4.

```
# Apply SVM Model
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
svc=SVC()
svc.fit(X_train, y_train)
y_pred=svc.predict(X_test)
svmscore=accuracy_score(y_test, y_pred)
svmc=classification_report(y_test, y_pred)
print('Model accuracy : {0:0.3f}'. format(svmscore))
Model accuracy : 0.911

# Apply SVM model using Linear Kernel function
linear_classifier=SVC(kernel='linear').fit(X_train,y_train)
y_pred = linear_classifier.predict(X_test)
lsvmscore=accuracy_score(y_test, y_pred)
lsvmc=classification_report(y_test, y_pred)
print('Model accuracy with linear kernel : {0:0.3f}'. format(lsvmscore))
Model accuracy with linear kernel : 0.924

# Apply SVM model using Gaussian RBF kernel function
rbf_svc=SVC(kernel='rbf').fit(X_train,y_train)
y_pred = rbf_svc.predict(X_test)
rbfsvmscore=accuracy_score(y_test, y_pred)
rbfsvmc=classification_report(y_test, y_pred)
print('Model accuracy with rbf kernel : {0:0.3f}'. format(rbfsvmscore))
Model accuracy with rbf kernel : 0.911


# Apply SVM model using Polynomial Kernel function
Poly_svc=SVC(kernel='poly', C=1).fit(X_train,y_train)
y_pred = Poly_svc.predict(X_test)
psvmscore=accuracy_score(y_test, y_pred)
psvmc=classification_report(y_test, y_pred)
print('Model accuracy with poly kernel : {0:0.3f}'. format(psvmscore))
Model accuracy with poly kernel : 0.924

# Apply SVM model using Sigmoid Kernel function
Poly_svc=SVC(kernel='sigmoid', C=1).fit(X_train,y_train)
y_pred = Poly_svc.predict(X_test)
ssvmscore=accuracy_score(y_test, y_pred)
ssvmc=classification_report(y_test, y_pred)
print('Model accuracy with sigmoid kernel : {0:0.3f}'. format(ssvmscore))
Model accuracy with sigmoid kernel : 0.892
```

Random Forest:

One of the most well-liked and frequently employed algorithms among data scientists is random forest. Random forest is a supervised machine learning algorithm that is frequently employed in Classification and Regression issues. It creates decision trees from several samples, using the majority vote for classification and the average in the case of regression (R, 2023).

The Random Forest Algorithm's ability to handle data sets with both continuous variables, as in the case of regression, and categorical variables, as in the case of classification, is one of its most significant features. For tasks requiring classification and regression, it performs better (R, 2023).

One of the techniques used in decision tree algorithms to select the best split from a root node and subsequent splits is the Gini impurity measure (Loaiza, 2020).

Gini Impurity of the node is the probability that a randomly selected sample within a node would be erroneously labeled if it were labeled by the distribution of samples within the node (Koehrsen, 2018). The Gini Impurity of a node n is 1 minus the square root of the fraction of examples in each class p_i, which is 2 for a binary classification assignment.

$$I_G(n) = 1 - \sum_{i=1}^{J} (P_i)^2 \qquad (3.18)$$

The decision tree looks through the features at each node to determine which value to split on to get the biggest reduction in Gini impurity. The splitting process is then repeated until either a maximum depth is reached or only samples from one class are present in each node of the tree. Each node's Gini impurity is weighted according to the percentage of points from the parent node that are present there. The weighted total Gini impurity of

the final layer eventually decreases to 0, indicating that each node is entirely pure and that there is no possibility that a point chosen at random from that node will be incorrectly categorized (Koehrsen, 2018).

All of datasets were fed into this model and results are discussed in Chapter 4.

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

from sklearn.metrics import accuracy_score
rfscore=accuracy_score(y_test, y_pred)
print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
Model accuracy score: 0.9236
```

K Nearest Neighbor:

K-nearest Neighbor (KNN) is a supervised classification technique based on discovering similarities between predicted data and the underlying data. Although it can also be used to address regression issues, KNN is most frequently employed to handle classification challenges. The initial presumption was that the data were clustered or located nearby. Since KNN is a non-parametric method, it does not attempt to make assumptions about the data, such as whether the data is normally distributed (Al-Serw, 2021).

KNN implementation steps (Al-Serw, 2021):

1. Set an initial value for K, which will represent all the nearby data points to which we will attempt to compare the new prediction.
2. Determine how similar all the data points are to the fresh data points. The Euclidean distance is employed in this.

$$\boldsymbol{d(p,q)} = \boldsymbol{d}(q,p) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \qquad \textbf{(3.19)}$$

The distance between points p and q, which is also the distance from "q" to "p", is shown by Formula (3.19). To calculate the distance, we multiply the size of each coordinate by one another, add them all up, use the power of 2, and then square the result (Al-Serw, 2021).

3. Using the previously indicated Euclidean distance, choose the K-nearest number of observations in the training data.
4. Predicting the next data point based on the most common response.

The KNN method is now repeated multiple times using different values, for example on a range between 2 and 11, so that the procedure is run a total of 10 times, to determine which K is appropriate for the dataset (Al-Serw, 2021). In our example the best value of K was 8.

```
import sklearn
from sklearn import neighbors
def sklearn_classify(train_data, label_data, test_data, k_num):
    knn = neighbors.KNeighborsClassifier(n_neighbors=k_num)
    # Train
    knn.fit(train_data, label_data)
    # Predict
    predict_label = knn.predict(test_data)
    # Return
    return predict_label
y_predict=sklearn_classify(X_train, y_train, X_test, 8)
```

```
def get_accuracy(test_labels, pred_labels):
    correct = np.sum(test_labels == pred_labels)
    n = len(test_labels)
    accur=correct/n
    return accur
knnscore=get_accuracy(y_test, y_predict)
print(knnscore)
0.9171974522292994

normal_accuracy = []
k_value = range(2, 11)
for k in k_value:
    y_predict = sklearn_classify(X_train, y_train, X_test, k)
    accuracy = get_accuracy(y_test, y_predict)
    normal_accuracy.append(accuracy)
plt.xlabel("k")
plt.ylabel("accuracy")
new_ticks = np.linspace(0.6, 0.9, 10)
plt.yticks(new_ticks)
plt.plot(k_value, normal_accuracy, c='r')
plt.grid(True)
```

All of datasets were fed into this model and results are discussed in Chapter 4.

Artificial Neural Network:

A neural network learns to predict outcomes by utilizing data as input, making a prediction, comparing the anticipated result with the intended result, and then altering its internal state to create a more accurate prediction the next time. Vectors are used to store the data, and arrays are used to store these vectors in Python. Each layer transforms the data from the layer before it. When using neural networks, we begin with a set of random weights and bias vectors, make a prediction, compare it to the intended result, and then change the vectors to make a more accurate prediction the following time. Up until there is little variation between the prediction and the actual targets, the process is repeated (Real Python, 2021).

Multi-layer fully connected neural nets are what artificial neural networks (ANN) are made of. An input layer, several hidden layers, and an output layer make them up. Each node in one layer is linked to each node in the layer above it. By adding more hidden layers, we make the network deeper (Dertat, 2017).

This is the equation for a certain node. A nonlinear activation function was used to process its inputs. It may be represented as a vector dot product where n is the number of inputs for the node (Dertat, 2017).

$$z = f(x \cdot w) = f(\sum_{i=1}^{n} x_i w_i) \quad (3.20)$$

A bias can be added to the equation, Bias is an input that each node receives and is always set to 1. It enables shifting the activation function's result to the left or right. When all the input features are 0, it aids the model's training process (Dertat, 2017).

$$z = f(b + x \cdot w) = f(b + \sum_{i=1}^{n} x_i w_i) \quad (3.21)$$

In Keras, we only create layers for hidden and output nodes; we don't add layers for input nodes. The input nodes are connected to the hidden layers of our current model's two hidden layers, and finally there is an output node (Dertat, 2017).

The fully connected neural network layer is built using Keras' Dense function, with the weights initialized as biases automatically (Dertat, 2017).

```
import keras
from keras.models import Sequential
from keras.layers import Dense

classifier = Sequential()
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation =
'relu', input_dim = 11))
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation =
'relu'))
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation =
'sigmoid'))
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =
['accuracy'])
```

Units: The number of nodes in this layer is represented by the first parameter (Dertat, 2017). Each hidden layer has a value of 6 in our example.

Kernel initializer: The first random weights of Keras layers are defined by initializers. The initializer uniform produces tensors with a uniform distribution (Team, n.d.).

Activation: 'ReLU' is the rectified linear unit activation function. This yields the typical ReLU activation, $max(x, 0)$, together with the input tensor and the element-wise maximum of 0. The usage of non-zero thresholds, changing the maximum value of the activation, and using a non-zero multiple of the input for values below the threshold are all possible by modifying the default parameters (Team, n.d.). Sigmoid activation function, $sigmoid(x) = \frac{1}{1 + exp(-x)}$. Logistic function, also known as the sigmoid, serves as the logistic regression model's activation function. Sigmoid returns a value that is almost equal to zero for small values (-5), and nearly one for high values (>-5) (Team, n.d.) (Dertat, 2017).

Optimizer: The goal of optimization is to reduce the loss function. Many optimizers exist, the majority of which are based on gradient descent

(Dertat, 2017). The default optimizer that people prefer to use is the Adam one, which we employed in our study.

Loss: Minimize the loss function. As we're creating a binary 0/1 classifier, binary cross entropy is the loss function to minimize (Dertat, 2017).

Metrics: Which measure to report statistics on; accuracy is chosen for classification issues (Dertat, 2017).

```
classifier.fit(X_train, y_train, batch_size = 10, epochs = 100)

y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

from sklearn.metrics import accuracy_score
annscore=accuracy_score(y_test, y_pred)
annscore
0.9299363057324841

from sklearn.metrics import confusion_matrix
cmann = confusion_matrix(y_test, y_pred)
cmann
array([[ 13,  11],
       [  5, 128]], dtype=int64)
```

3.4. Second Stage (CNN)

CNN is the most well-known and widely used DL algorithm (Alzubaidi & Farhan, 2021). CNN has a substantial benefit over its predecessors in that it detects key characteristics automatically and without the need for human intervention (Wang as cited in Alzubaidi & Farhan, 2021). CNNs have been widely employed in several disciplines, including computer vision, audio processing, face recognition, and so on. CNNs, like classic neural networks, have a structure influenced by neurons found in human and animal brains. More precisely, the visual cortex in an animal's brain is made up of a complicated arrangement of cells, and the CNN mimics this structure (Hubel as cited in Alzubaidi & Farhan, 2021).

Equivalent representations, sparse interactions, and parameter sharing are three of the CNN's main advantages, according to Goodfellow as cited in Alzubaidi & Farhan, (2021). In contrast to typical fully connected (FC) networks, the CNN uses shared weights and local connections to fully exploit 2D input-data structures, such as visual signals. This technique uses a remarkably minimal number of parameters, which speeds up the network while also making training easier.

3.4.1. System architecture:

As mentioned earlier, the 3rd dataset was collected from Instagram Platform, Fake users were purchased from an online market, real users were collected from university account. IG Exporter tool was used with APIFY to gather users accounts and features, Features were prepared, and clustering was applied to the fake users, feature were extracted, and dataset is fed to CNN at the end. Figure (3.34) demonstrates the process behind the 3rd dataset preparation until feeding it to the CNN network.

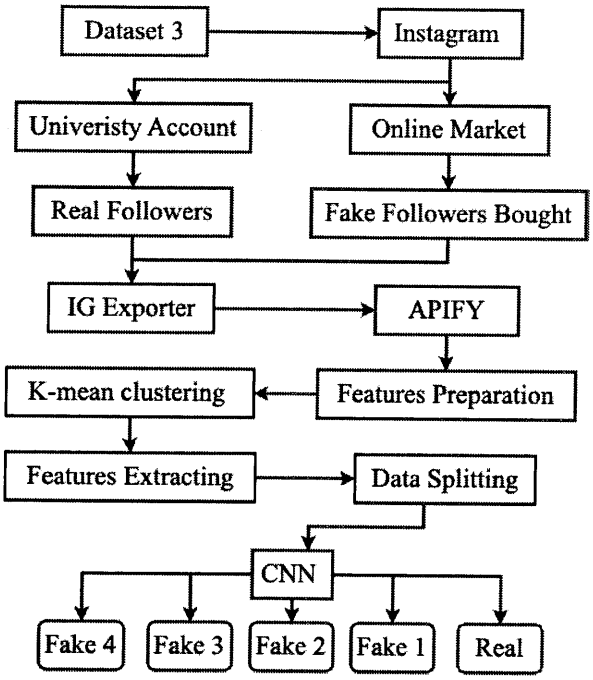


Figure (3.34): 3rd Dataset + Stage 2 (CNN) Architecture.

3.4.2. System implementation:

We have applied CNN topology in Jupyter Notebook using python libraries, steps as discussed below:

Step 1: importing libraries.

```
# import libraries
import numpy as np
import pandas as pd
import os, re, time, math, tqdm, itertools
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.offline as pyo
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.neural_network import MLPClassifier
import keras
from keras.layers import Conv2D, Conv1D, MaxPooling2D, MaxPooling1D, Flatten,
BatchNormalization, Dense, Dropout
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.callbacks import CSVLogger, ModelCheckpoint
```

- NumPy: The acronym NumPy, or "Numerical Python," has long been a pillar of Python's support for numerical computing. It provides the Python library glue, data structures, and algorithms required for most scientific applications requiring numerical data. One of NumPy's main purposes in data analysis, aside from the quick array processing capabilities it gives Python, is as a container for data that needs to be transmitted between algorithms and libraries. NumPy arrays are more effective than the other built-in Python data structures at storing and manipulating numerical data (McKinney, 2012).

- Pandas: When working with structured or tabular data, pandas offer high-level data structures and functions that are intended to be quick, simple, and expressive. Since its debut in 2010, it has contributed to making Python a potent and effective data analysis environment (McKinney, 2012).

- Matplotlib: The most widely used Python library for creating plots and other two-dimensional data visualizations is matplotlib. It was initially developed by John D. Hunter, and a large group of developers now looks after it. It is made to help writers craft stories that are fit for publication (McKinney, 2012).

- Seaborn: A graphic visualization package called Seaborn is based on the core settings of Matplotlib. It allows users access to some of the most widely used data visualization techniques, such as assigning a color to a variable or applying global faceting standards. It offers a more comprehensive seaborn for utilizing Pandas Data Frames (Odegua, as cited in Sial & Khan, 2021).

- scikit-learn: The best all-purpose machine learning toolbox for Python programmers is called scikit-learn. Scikit-learn has been essential for enabling Python to be a successful data science programming language, along with pandas, stats models, and I Python. Models including classification, regression, clustering, dimension reduction, model selection, and preprocessing are included (McKinney, 2012).

- Keras: The Python deep learning framework Keras offers an easy way to define and train practically any deep learning model. In the beginning, Keras was created for researchers with the intention of facilitating quick experimentation. Over 200,000 people use Keras, including academic researchers, engineers working for small and large businesses, graduate students, and hobbyists (Chollet, 2021).

Step 2: loading data.

```
%%time
# load the data into memory
instagram_data = pd.read_csv("clustered data shuffeled.csv")
CPU times: total: 234 ms
Wall time: 275 ms

# to display all column of datapoints
pd.set_option('display.max_columns', None)
```

We have used here *pandas.read_csv* which read the csv file and load the dataset into memory. *pandas.set_option* was also used to set *display.max_columns* which means when maximum columns are reached, objects are either centrally truncated or printed as a summary view depending on *large_repr*, and "None" means unlimited.

Step 3: Reading and processing the data.

```
# check the shape of data
instagram_data.shape

# check the number of rows and columns
print('Number of Rows (Samples): %s' % str((instagram_data.shape[0])))
print('Number of Columns (Features): %s' % str((instagram_data.shape[1])))
Number of Rows (Samples): 83861
Number of Columns (Features): 18

# check the columns in data
instagram_data.columns
Index(['followers', 'following', 'highlight reel count', 'is buisness?',
       'is joined recently?', '#posts', 'is private?', 'is verified?',
       'description length', 'has website?', 'username==fullname?',
       '#digits in username', '#digits in fullname', 'username has digit?',
       'fullname has digits?', 'length of fullname', 'length of username',
       'isFake'],
      dtype='object')

# check the number of columns
print('Total columns in our data: %s' % str(len(instagram_data.columns)))
Total columns in our data: 18
```

The data is consisting of 83861 rows and 18 columns (17 features and 1 label) as shown in code.

```
instagram_data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 83861 entries, 0 to 83860
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   followers             83861 non-null  int64
 1   following             82864 non-null  float64
 2   highlight reel count  83861 non-null  int64
 3   is buisness?          83861 non-null  int64
 4   is joined recently?   83861 non-null  int64
 5   #posts                83859 non-null  float64
 6   is private?           83861 non-null  int64
 7   is verified?          83861 non-null  int64
 8   description length    83861 non-null  int64
 9   has website?          83861 non-null  int64

 10  username==fullname?   83861 non-null  int64
 11  #digits in username   83861 non-null  int64
 12  #digits in fullname   83861 non-null  int64
 13  username has digit?   83861 non-null  int64
 14  fullname has digits?  83861 non-null  int64
 15  length of fullname    83861 non-null  int64
 16  length of username    83861 non-null  int64
 17  isFake                83861 non-null  int64
dtypes: float64(2), int64(16)
memory usage: 11.5 MB
```

We can view that all features are of type integer, only 'following' and '#posts' are of type float.

```
# check the number of values for fake and real
instagram_data['isFake'].value_counts()
2 34920
0 15943
1 13885
4 11029
3 8084
Name: isFake, dtype: int64
```

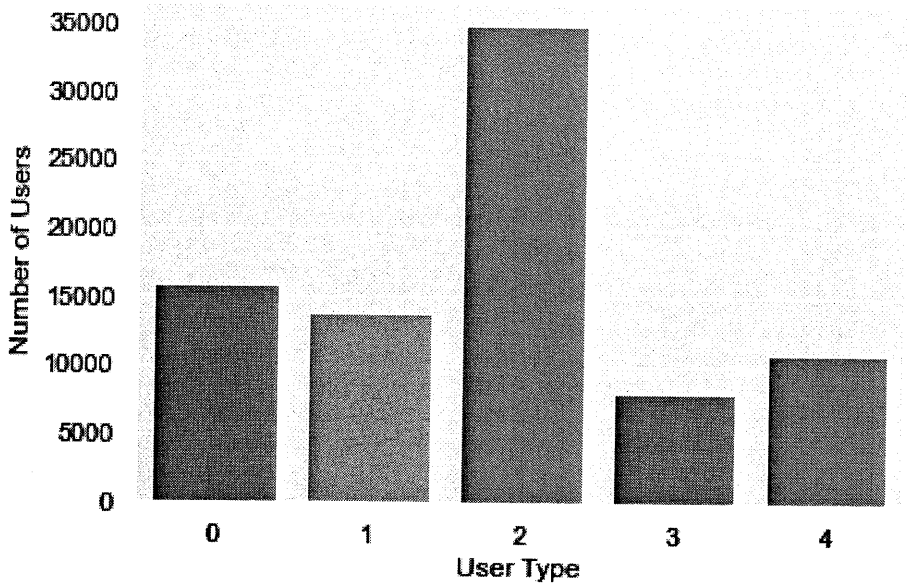


Figure (3.35): Distribution of 3rd dataset.

The label 'is Fake' has 5 classifications, their distribution as follows: '0' real user is 15942, '1' fake1 users are 13856, '2' fake2 users are 34920 which is highest, '3' fake3 users are 8084 which is lowest, '4' fake4 users are 11059, Figure (3.35). As mentioned earlier that fake users are reclassified into 4 classes based on their behavior, it is obvious that the distribution of the label is not balanced, resampling is to be used later.

Step 4: cleaning and pre-processing dataset

```
# check for some null or missing values in our dataset
instagram_data.isna().sum().to_numpy()
array([ 0, 997, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0], dtype=int64)

# drop null or missing columns
cleaned_data = instagram_data.dropna()
cleaned_data.isna().sum().to_numpy()
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

We have checked if there is any null values and it appeared that there are some null values, these datapoint will not be useful in training and we must

remove them, so we have used the function dropna() that will do the purpose and remove rows (datapoints) that contains null.

```
fetaures_name =  ["followers","following","highlight reel count","is
buisness?","is joined recently?","#posts","is private?",
                 "is verified?","description length","has web-
site?","username==fullname?","#digits in username",
                 "#digits in fullname","username has digit?","fullname has dig-
its?","length of fullname",
                 "length of username","isFake"]
cleaned_data = cleaned_data.drop_duplicates(subset = fetaures_name , keep
='first' , inplace = False)
```

We have also used the function drop duplicates to remove the duplicated rows, because having them will consume storage and time without any gain.

After that we had to check the relationship between features, and as illustrated in Figure (3.36), the feature "username==full name?" is highly correlated with "#digits in full name" and "full name has digits?", also "full name has digits?" is highly related to "#digits in full name", another one is "username has digit?" is highly related to "#digits in username".

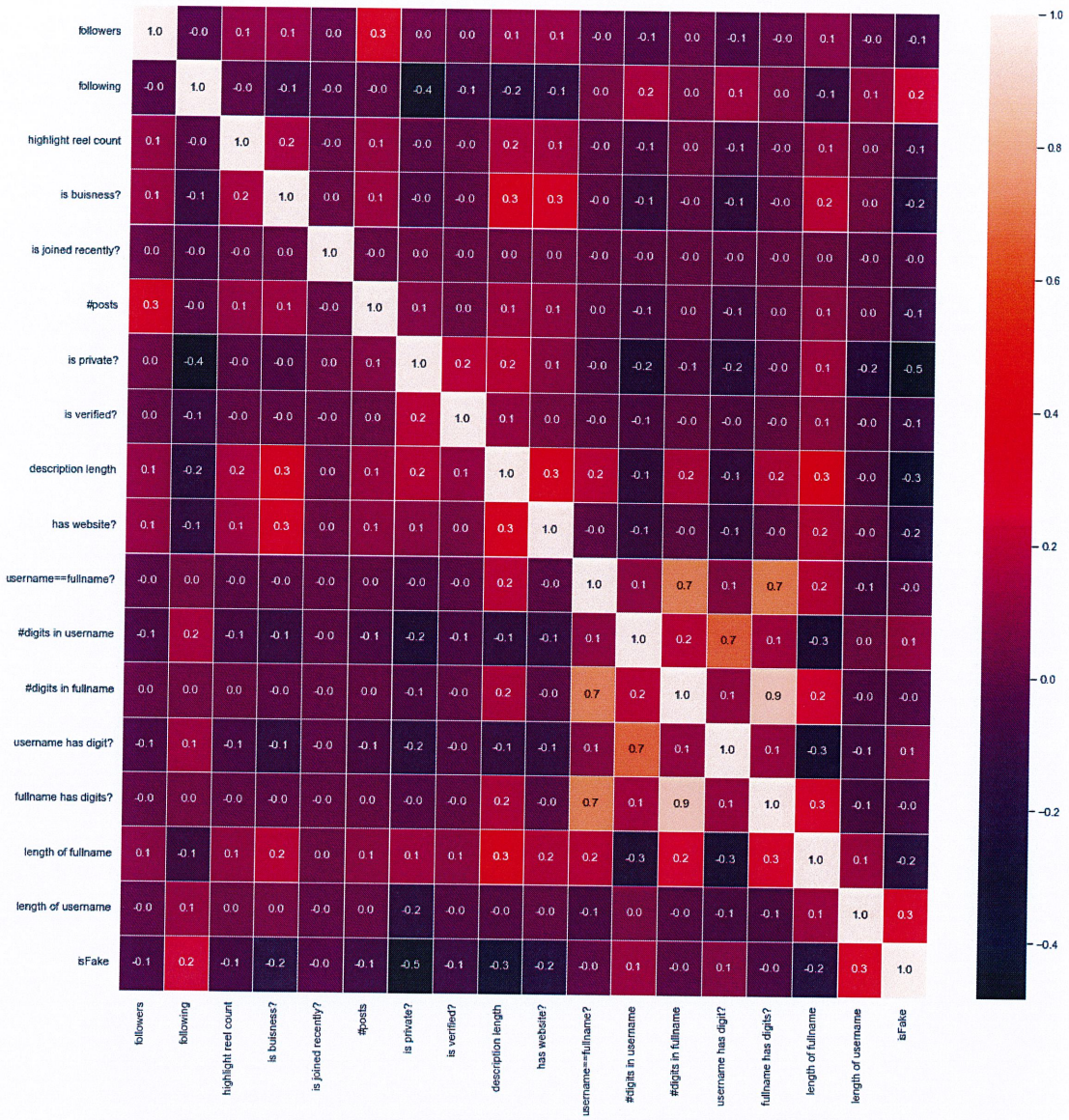


Figure (3.36): Correlation map for 3rd dataset.

The feature "username==full name?" is highly correlated with "#digits in full name" and "full name has digits?", also "full name has digits?" is highly related to "#digits in full name", and "username has digit?" is highly related to "#digits in username". As result, we removed "username==full name?", "full name has digits", and "username has digits?".

As a result, we removed the columns "username==full name?", "full name has digits", and "username has digits?".

```
drop_list1 = ['username==fullname?', 'username has digit?','fullname has dig-
its?']
cleaned_data = cleaned_data.drop(drop_list1,axis = 1 )
cleaned_data.head()
```

Step 5: resampling the data.

We have separated the data to have each label stored alone, and then we have combined them as X and y.

```
# make 5 seperate datasets for 5 feature labels
data_1 = cleaned_data[cleaned_data['isFake'] == 0]
data_2 = cleaned_data[cleaned_data['isFake'] == 1]
data_3 = cleaned_data[cleaned_data['isFake'] == 2]
data_4 = cleaned_data[cleaned_data['isFake'] == 3]
data_5 = cleaned_data[cleaned_data['isFake'] == 4]

# make real feature
y_1 = data_1['isFake']

# make fake1 feature
y_2 = data_2['isFake']

# make fake2 feature
y_3 = data_3['isFake']

# make fake3 feature
y_4 = data_4['isFake']

# make fake4 feature
y_5 = data_5['isFake']

# merging the original dataframe
X = pd.concat([data_1, data_2, data_3, data_4, data_5], sort=True)

y = pd.concat([y_1, y_2, y_3, y_4, y_5], sort=True)
```

- Down sampling:

```
from sklearn.utils import resample

data_1_resample= resample(data_1, n_samples=7000, random_state=12, replace=False)
data_2_resample= resample(data_2, n_samples=7000, random_state=12, replace=False)
data_3_resample= resample(data_3, n_samples=7000, random_state=12, replace=False)
data_4_resample= resample(data_4, n_samples=7000, random_state=12, replace=False)
data_5_resample= resample(data_5, n_samples=7000, random_state=12, replace=False)

dataf = pd.concat([data_1_resample, data_2_resample, data_3_resample,
data_4_resample, data_5_resample])

dataf.shape
(35000, 15)

y=dataf['isFake']
X = dataf.iloc[:, :-1].values
X.shape,y.shape
((35000, 14), (35000,))
```

We have used *pandas.DataFrame.resample*, this will consistently resample the data, we have chosen to set n_samples=7000 samples for each label classification so we can get a balanced dataset. Replace is set to False so no replacement of values will happen, since we don't want duplicated rows, this is a must because the default is set to True. n_sample is only set in the case of down sampling; in the case of oversampling, we remove this hyperparameter. We have dropped the last column for X to get it ready for training and testing. The dataset now is balanced, Figure (3.37).

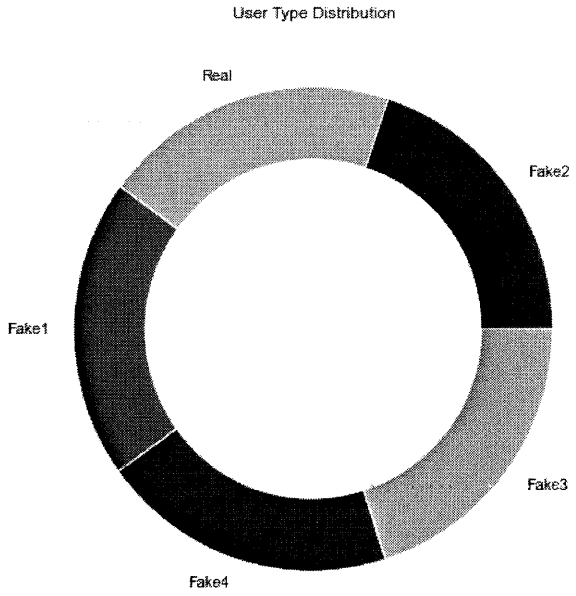


Figure (3.37): Balanced distribution for 3rd dataset.

- Oversampling:

We have used SMOTE technique to rise and balance our datapoints, it is a machine learning method that can be used to generate new samples that have the same distribution as the existing dataset. The fundamental idea behind this approach is to oversample minority classes to balance out their representation in the data collection. This will lessen the risk of overfitting and increase the model's precision when projecting future observations or drawing conclusions about hypothetical examples from the same population (Satpathy, 2021).

In our code, we have dropped the last column for X to get it ready for training and testing. The label with the highest datapoints is fake2 having 27472 datapoints, and when applying oversampling with SMOTE, the rest of labels will increase their datapoint to reach 27472 datapoints, and balanced dataset will be achieved.

```
X = X.iloc[:, :-1].values
print(X.shape)
print(y.shape)
print(y_train.unique())
print(y_test.unique())

print("Number transactions X_train dataset: ", X_train.shape)
print("Number transactions y_train dataset: ", y_train.shape)
print("Number transactions X_test dataset: ", X_test.shape)
print("Number transactions y_test dataset: ", y_test.shape)
```

(82228, 14)

(82228,)

[2 0 3 4 1]

[2 1 0 3 4]

Number transactions X_train dataset: (65782, 14)

Number transactions y_train dataset: (65782,)

Number transactions X_test dataset: (16446, 14)

Number transactions y_test dataset: (16446,)

```
from imblearn.over_sampling import SMOTE

print("Before OverSampling, counts of label '4': {}".format(sum(y_train==4)))
print("Before OverSampling, counts of label '3': {}".format(sum(y_train==3)))
print("Before OverSampling, counts of label '2': {}".format(sum(y_train==2)))
print("Before OverSampling, counts of label '1': {}".format(sum(y_train==1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train==0)))

sm = SMOTE(random_state=2)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train.ravel())

print('After OverSampling, the shape of train_X: {}'.format(X_train_res.shape))
print('After OverSampling, the shape of train_y: {}
\n'.format(y_train_res.shape))

print("After OverSampling, counts of label '4': {}".format(sum(y_train_res==4)))
print("After OverSampling, counts of label '3': {}".format(sum(y_train_res==3)))
print("After OverSampling, counts of label '2': {}".format(sum(y_train_res==2)))
print("After OverSampling, counts of label '1': {}".format(sum(y_train_res==1)))
print("After OverSampling, counts of label '0': {}".format(sum(y_train_res==0)))
```

Before OverSampling, counts of label '4': 8704

Before OverSampling, counts of label '3': 6430

Before OverSampling, counts of label '2': 27472

Before OverSampling, counts of label '1': 11151

Before OverSampling, counts of label '0': 12025

After OverSampling, the shape of train_X: (137360, 14)

After OverSampling, the shape of train_y: (137360,)

After OverSampling, counts of label '4': 27472

After OverSampling, counts of label '3': 27472

After OverSampling, counts of label '2': 27472

After OverSampling, counts of label '1': 27472

After OverSampling, counts of label '0': 27472

Step 6: splitting the dataset.

We have split the dataset to have 20% as test data and 80% as training data, moreover, we must convert the label into categorical form to be ready to be fed in CNN, the label data will be converted as shown in Table (3.9), random values were chosen to show the form in the code.

Table (3.9): categorical form for the label.

Label "is Fake"	Categorical form
0	[1., 0., 0., 0., 0.]
1	[0., 1., 0., 0., 0.]
2	[0., 0., 1., 0., 0.]
3	[0., 0., 0., 1., 0.]
4	[0., 0., 0., 0., 1.]

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran-
dom_state=0)


y_train = to_categorical(y_train_res, num_classes=5)
y_test = to_categorical(y_test, num_classes=5)

y_train.shape
(28000, 5)


y_train[3000],y_train[11120],y_train[13700],y_train[15120],y_train[21120]
(array([0., 0., 0., 0., 1.], dtype=float32),
 array([0., 0., 0., 1., 0.], dtype=float32),
 array([0., 0., 1., 0., 0.], dtype=float32),
 array([0., 1., 0., 0., 0.], dtype=float32),
 array([1., 0., 0., 0., 0.], dtype=float32))

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

(28000, 14)
(7000, 14)
(28000, 5)
(7000, 5)


# reshape the data for CNN
X_train = X_train.reshape(len(X_train), X_train.shape[1], 1)
X_test = X_test.reshape(len(X_test), X_test.shape[1], 1)
X_train.shape, X_test.shape

((28000, 14, 1), (7000, 14, 1))
```

Step 7: training the model:

In this step we are now ready to train the model. A neural network type called a convolutional neural network, or CNN or Conv Net, is particularly adept at processing input with a grid-like architecture, like an image. A binary representation of visual data is a digital image. It is made up of a

grid-like arrangement of pixels, each of which has a pixel value to indicate how bright and what color it should be (Mishra, 2020).

The moment we perceive an image, the human brain begins processing a massive amount of data. Every neuron has a distinct receptive field and is coupled to other neurons so that they collectively cover the whole visual field. Each neuron in a CNN processes data only in its receptive field, like how each neuron in the biological vision system responds to stimuli only in the constrained area of the visual field known as the receptive field. The layers are set up so that simpler patterns are detected early on and more complicated patterns later. One can enable sight to computers by employing a CNN (Mishra, 2020).

Convolution is a mathematical process that involves multiplying two functions to create a third function that expresses how the form of one function is altered by the other. The term "convolution" is used in CNN to refer to this mathematical activity. To extract features from a picture, two images that can be represented as matrices are multiplied to produce an output (Gurucharan, 2022).

```
# making the deep learning function
def model():
    model = Sequential()
    model.add(Conv1D(filters=64, kernel_size=6, activation='relu',
                     padding='same', input_shape=(14, 1)))
    model.add(BatchNormalization())
    model.add(MaxPooling1D(pool_size=(3), strides=2, padding='same'))
    model.add(Flatten())
    model.add(Dropout(0.1))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(5, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', met-
rics=['accuracy'])
    return model
```

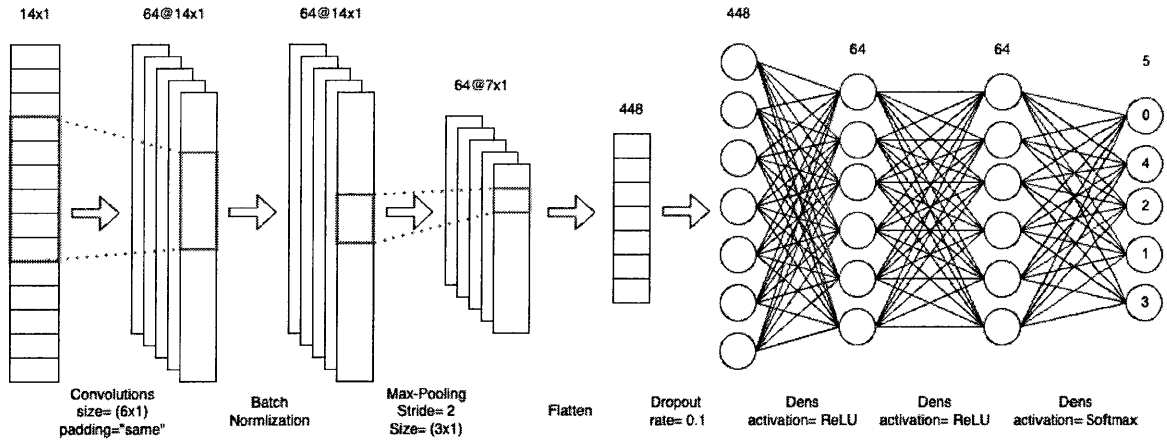


Figure (3.38): CNN layers visualization.

The CNN is made up of three different kinds of layers: convolutional, pooling, and fully connected (FC). A CNN architecture is created when these layers are stacked (Gurucharan, 2022). The used layers in our model are visualized in Figure (3.38). Layers are discussed below with steps:

1. 1D Convolutional layer:

   The foundational component of the CNN is the convolution layer. It carries most of the computational load on the network (Mishra, 2020).

   The first layer utilized to extract the different features from the input photos is this one. The input image and a filter of a specific size (MxM) are convolution mathematically in this layer; in our scenario, the input image will be of size (14x1). The dot product between the filter and the portions of the input image regarding the size of the filter (MxM), which is in our example (6x1), is taken by swiping the filter over the input image (Gurucharan, 2022).

$$\boldsymbol{Wout} = \frac{W-F+P}{S} + 1 \qquad (3.22)$$

From Formula (3.22), we can calculate the size of the output volume if we have an input of size W x W x D, D_{out} number of kernels, spatial/kernel size F (a single integer or set of single integers specifying the 1D convolution window's length), stride S (the kernel's movable size), and amount of padding P (When you pad, the output size is equal to the input size because of the same convolution) (Nkomo, 2022; Mishra, 2020).

In our case the input shape is (None,14,1) so W=14, and the kernel size is 6 so F=6, and padding is 'same' that indicates we want the output to be same size of our input, so $W=W_{out}=14$, and S is 1 by default.

$$output = (None, 14,64)$$

Figure (3.39) illustrates the steps of Convolution layer.

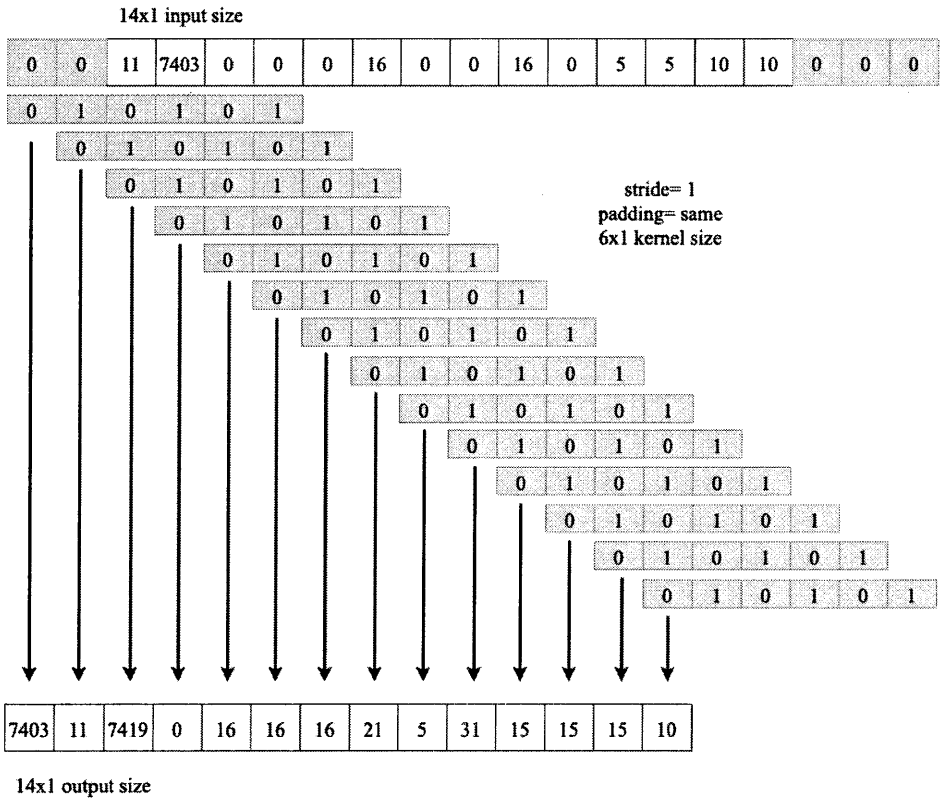


Figure (3.39): Steps of Convolution Layer.

2. Batch Normalization Layer

Instead of conducting the normalization in the raw data, batch normalization is done between neural network layers. Instead of utilizing the complete data set, mini batches are used. It accelerates training and uses faster learning rates to promote learning (Riva, 2022). The output dimension is the same as the input dimension.

$$output = (None, 14{,}64)$$

The theory behind the normalization layer is demonstrated in the proceeding equations, (Peccia, 2019). The batch mean is calculated with the batch variance, Formula (3.23), and Formula (3.24).

$$\boldsymbol{\mu_B} = \frac{1}{m}\sum_{i=1}^{m} x_i \quad \textbf{(3.23)}$$

$$\boldsymbol{\sigma_B^2} = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \quad \textbf{(3.24)}$$

Next step is applying normalization with Formula (3.25),

$$\boldsymbol{\bar{x}_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad \textbf{(3.25)}$$

Last step is applying scale and shift to get the output of the layer with Formula (3.26).

$$\boldsymbol{y_i} = \gamma \bar{x}_i + \beta \quad \textbf{(3.26)}$$

3. 1D Max Pooling Layer

By obtaining a summary statistic of the surrounding outputs, the pooling layer substitutes for the network's output at specific locations. The representation's spatial size is lowered as a result, which lowers the calculation and weighting requirements. Every

slice of the representation that the pooling operation touches is treated separately (Mishra, 2020).

A weighted average based on the distance from the central pixel is one of the pooling functions, along with the average of the rectangular neighborhood and the L2 norm of the rectangle neighborhood. However, max pooling, which reflects the highest output from the neighborhood, is the most well-liked method (Mishra, 2020).

The greatest component in Max Pooling is derived from a feature map. The elements inside a predetermined sized Picture slice are averaged using the average pooling technique. Sum Pooling calculates the aggregate sum of the components in the predefined section. Convolutional Layer and Fully Connected Layer are typically connected by the Pooling Layer (Gurucharan, 2022).

$$\boldsymbol{Wout} = \frac{W-F}{S} + 1 \qquad (3.27)$$

The output volume can be calculated using Formula (3.27) if we have an activation map of size W x W x D, a pooling kernel of spatial size F, and stride S (Mishra, 2020), in our case here the output of the previous layer is to be used as the input in this layer, so W=14, F=3, and S=2. However, using the "same" padding option produces the following output shape: $\boldsymbol{output\ shape} = \frac{\boldsymbol{input\ shape}}{\boldsymbol{strides}}$

$$\boldsymbol{Wout} = \frac{14}{2} = 7 \qquad (3.28)$$

$$output = (None, 7, 64)$$

Figure (3.40) illustrates steps of Max-Pooling layer.

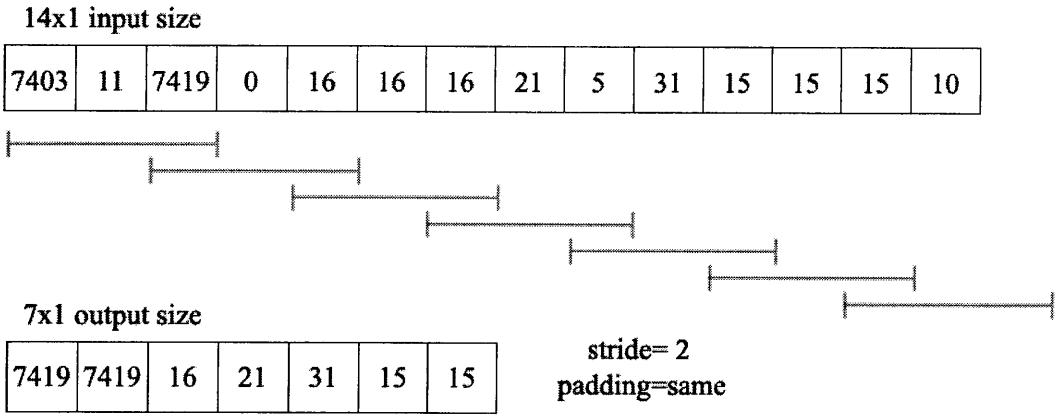


Figure (3.40): Steps of Maximum Pooling layer.

4. Flatten Layer

We have flattened and completely linked layers at the end of CNN. It must take the form of a one-dimensional linear vector. Direct inputs are not allowed to be rectangular or cubical in form. As a result, we need layers that are fully connected and flattened. When data is converted into a 1-dimensional array for input into the next layer, it is flattened. We flatten the output of the convolutional layer to create a feature vector. It is also linked to the previous classification model, which is known as a fully connected layer, as shown in Figure (3.41). In other words, we connect the last layer to a single line that contains all the pixel data (Jeong, 2019).

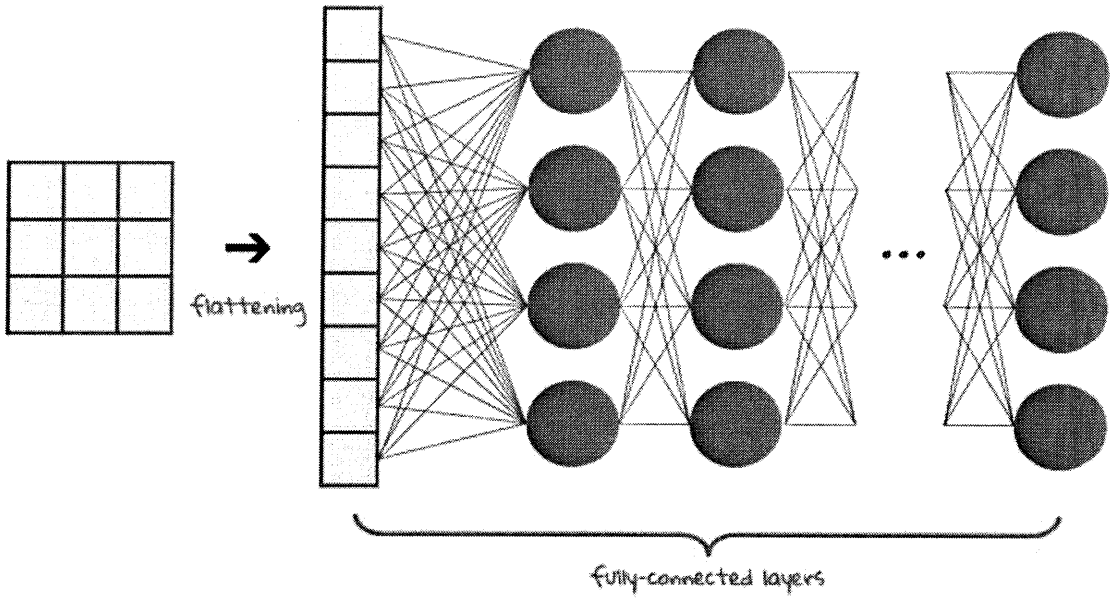


Figure (3.41): Flatten and fully connected layers, (Jeong, 2019).

By taking the previous output as an input for this layer, we got the following output:

$$output = (None, 7x64) = (None, 448)$$

5. Dropout layer

A dropout layer is a particular kind of regularization method that is used to avoid overfitting and enhance generalization error in machine learning and neural network models. A dropout layer randomly removes a part of its input units during each training cycle. As a result, because it cannot rely on a single input unit to always be available, the model is compelled to acquire more reliable and generalizable properties. Any layer in a neural network model can have dropout layers added, and a hyperparameter can be used to adjust how many units will really drop out of the network (Brownlee, 2019).

$$output = (None, 448)$$

6. Dens Layer

Each neuron in a dense layer that is connected to the layer above it in a neural network signifies that layer is connected to the layer above it. This layer is the one that is most typically used in artificial neural network networks. Each neuron in the layer above delivers signals to the neurons in the dense layer, which multiply matrices and vectors, in a model. During matrix vector multiplication, the row vector of the output from the preceding layers is identical to the column vector of the dense layer. To multiply matrices using vectors, the row vector must have the same number of columns as the column vector (Verma, 2023).

One of the most fundamental and important parameters of the Keras dense layer, which determines the size of the dense layer's output, is units. Since it reflects the output vector's dimensions, it must be a positive integer; in this layer, the units are 64. The activation function in neural networks is a function that transforms the input values of the neurons. It adds nonlinearity to neural network networks so that the networks can figure out how input and output values relate to one another. In this layer, we are use the ReLU activation function (Verma, 2023).

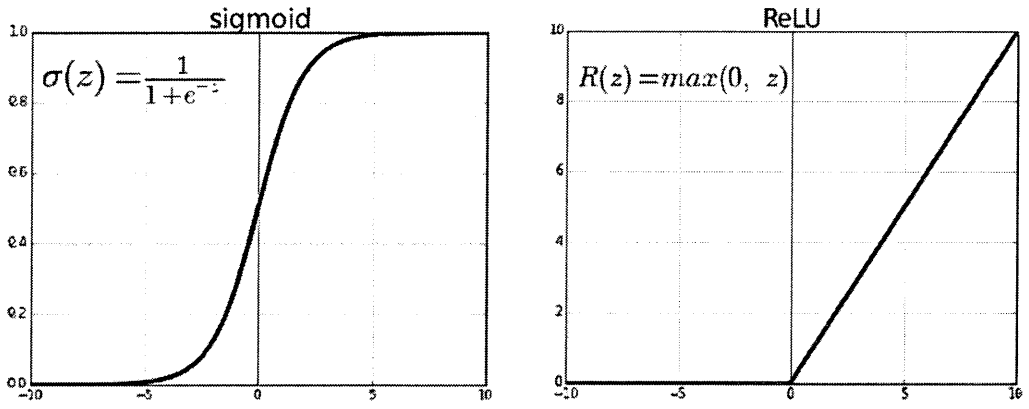


Figure (3.42): Sigmoid and ReLU functions, (Sharma, 2022).

Nowadays, the ReLU is the activation function that is employed the most globally. Figure (3.42) illustrates how the ReLU is only partially rectified (from bottom). When z is less than zero, f(z) equals zero, and when z is more than or equal to zero, f(z) equals z. Hence, the Range is [0 to infinite) (Sharma, 2022).

$$output = (None, 64)$$

7. Dens Layer

$$output = (None, 64)$$

8. Dens Layer

The neural network's raw outputs are input into the SoftMax activation function, which then produces a vector of probability scores (Priya, n.d.).

- The neural network's raw outputs are represented by the vector z.
- 2.718 is the value of e.

- The expected probability that the test input will belong to class i is represented by the i-th entry in the SoftMax output vector SoftMax(z).

The function of SoftMax is represented below:

$$\boldsymbol{softmax}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \quad (3.29)$$

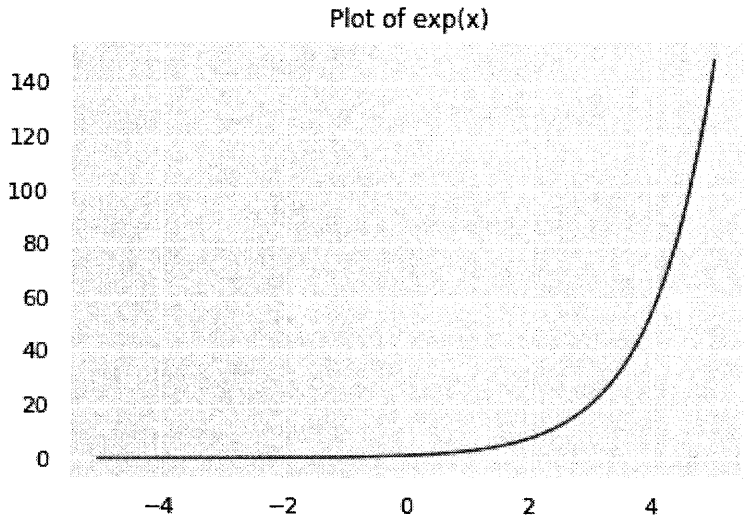


Figure (3.43): Plot of exp, (Priya, n.d.).

It's clear from the plot of e^x that e^x is always a positive value, regardless of whether the input x is positive, negative, or zero, Figure (3.43), (Priya, n.d.).

As we have five classes—real, fake1, fake2, fake3, fake4—and the valid indices are 0, 1, 2, 3, and 4, N = 5 in our example. By assuming the given the neural network's raw outputs, represented by the vector z = [0.3, 1.15, -0.7, 5.2, 1.4].

To elaborate more on how the calculation process is done, the following calculations are done with referral to (Priya, n.d.):

1. Determine each entry's exponent.

2. Subtract the outcome of step 1 from the total exponents of all the entries.

$$\frac{e^{0.3}}{e^{0.3} + e^{1.15} + e^{-0.7} + e^{5.2} + e^{1.4}} = 0.00709$$

$$\frac{e^{1.15}}{e^{0.3} + e^{1.15} + e^{-0.7} + e^{5.2} + e^{1.4}} = 0.01659$$

$$\frac{e^{-0.7}}{e^{0.3} + e^{1.15} + e^{-0.7} + e^{5.2} + e^{1.4}} = 0.00260$$

$$\frac{e^{5.2}}{e^{0.3} + e^{1.15} + e^{-0.7} + e^{5.2} + e^{1.4}} = 0.95239$$

$$\frac{e^{1.4}}{e^{0.3} + e^{1.15} + e^{-0.7} + e^{5.2} + e^{1.4}} = 0.02130$$

$$\begin{bmatrix} 0.3 \\ 1.15 \\ -0.7 \\ 5.2 \\ 1.4 \end{bmatrix} \rightarrow Softmax \rightarrow \begin{bmatrix} 0.00709 \\ 0.01659 \\ 0.00260 \\ 0.95239 \\ 0.02130 \end{bmatrix}$$

The following conclusions can be drawn from the SoftMax output above: The greatest element in the SoftMax output vector, 0.952, is represented by the maximum value of 5.2 in the vector z of raw outputs. SoftMax's application thus maintains the relative ranking of scores. The SoftMax output vector's entries are all between 0 and 1. See how the entries of the SoftMax output aggregate to 1 in a multiclass classification problem where the classes are mutually

exclusive (Priya, n.d.): $0.0071 + 0.0166 + 0.00260 + 0.9524 + 0.02130 = 1$.

By checking the SoftMax output for our case as a following step.

The greatest value in the vector $softmax(z) = [0.0071, 0.0166, 0.00260, 0.9524, 0.02130]$ is 0.9524. This indicates that there is a 95.24% likelihood that the provided input is a member of class 4, or the fake3 class.

The output dimension of this layer is:

$$output = (None, 5)$$

Figure (3.44) demonstrates the layers of CNN model.

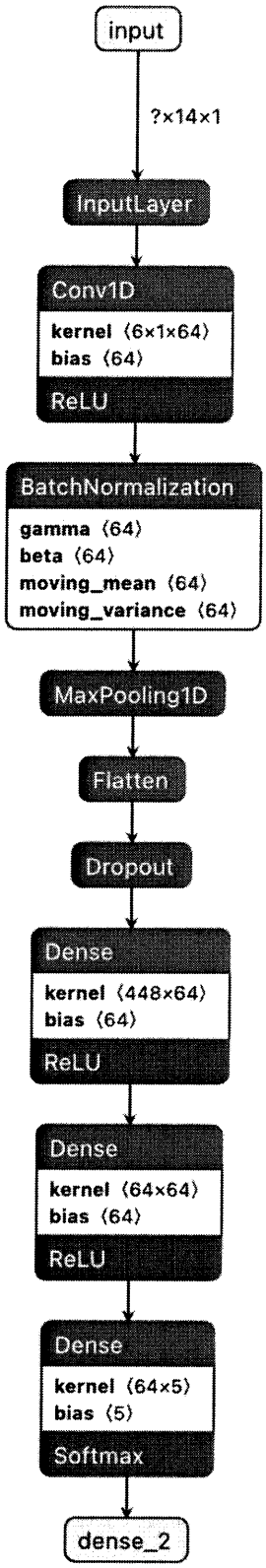


Figure (3.44): CNN Model Layers.

3.5. Summary

This chapter discusses our work in detail, it includes dataset preparation, stage 1, and stage 2 entire work. We have used two open-access datasets, and we have collected and processed the third dataset by ourselves, we have applied k-mean clustering to our dataset. In stage 1 we used all datasets in four machine learning models: ANN, KNN, RF, and SVM. In stage 2 we have applied CNN topology for the third dataset, with resampling.

Chapter Four

Experimental Results

4.1. Overview

The research findings and outcomes are highlighted and addressed in this chapter in relation to the study's goal, which was to anticipate and categorize real from fake user accounts and examine their online behavior. The datasets used in this study and the specifics of their gathering are also described. It includes analysis, a discussion of the suggested models, comparisons, and validation.

Also, this chapter provides various tables and graphs showing the conclusions of the investigation. The created models were coded using the Python programming language (Anaconda software using Jupyter Notebook 6.0.3). The investigations were carried out utilizing (Lenovo Laptop with Windows 10 operating system, Intel Core TM i7-5500U CPU, 2.40GHz, 8 GB RAM)

4.2. Datasets Analysis

The first dataset that was collected by Jafari, (2021) who bought fake followers, gathered real users from his private account, and used C# web scraper to export accounts information. This dataset consists of 12 features, 785 rows, and 1 binary classification label, Table (4.1). The data was not balanced, it had 88% fake users and around 12% real ones, this data did not have null values, but it had a column "has channel" that was pointless, because it had 785 datapoints as 0 which means all data has no channel, so

we removed this column and the dataset had 11 features, 785 rows, and 1 label.

The second dataset was gathered by Bakhshandeh, (2019), who classified the accounts into fake or real users manually by himself. This dataset consists of 11 features, 696 rows, and 1 binary classification label, Table (4.1). The data was balanced, it had 50% fake users and 50% real ones, this data did not have null values.

Table (4.1): Features of datasets (1st and 2nd).

Features	
1st dataset	*2nd dataset*
edge_followed_by	#followers
username_has_number	nums/length full name
full_name_has_number	name=username
is private	private
has_channel	description length
has guides	profile pic
edge follow	#follows
username length	nums/length username
full_name_length	full name words
is_joined_recently	#posts
has_external_url	external URL
is_business_account	

The third dataset was collected by us, Table (4.2). We used IG Exporter and APIFY to gather information, then we selected features from the data we obtained. The fake followers we collected were more than the real ones, and this was done with the intention of reclassifying the fake followers once more to have more than just binary classification. To do this, we used k-mean clustering with Python and Microsoft Excel. We ended up with five

classification models. The dataset has 83861 rows, 17 characteristics, and 1 label with 5 classifications. We eliminated rows from the features "#posts" and "Following," which together had 997 empty rows and 2 empty rows, respectively. Also, the data contained duplicate values, therefore we eliminated them to leave 82228 rows. We employed resampling to obtain balanced data, once with down sampling and once with SMOTE over sampling. Since we had 5 categories, we restructured the label column to contain 5 columns.

Table (4.2): Features of 3rd dataset.

Third dataset features					
follower	Following	highlight reel count	is business?	is joined recently?	#posts
is private?	is verified?	description length	has website?	username==full name?	#digits in username
#digits in full name	username has digit?	full name has digits?	length of full name	length of username	is Fake

4.3. Implementation parameters

The proposed feature selection and classification algorithms in this work have been applied using a set of parameters. The input parameters were assigned to the corresponding values shown in Tables (4.3), (4.4), (4.5), and Table (4.6) to train the models. Chapter 3 provides a detailed explanation of all the parameters used in the models' implementation.

The values of the ANN model parameters are as follows: units were set to 6 for all datasets, input dim was 11 for the first and second datasets, and 14 for the third dataset. Kernel initializer is set to uniform for all datasets, the activation was set to ReLU for the first two layers, and sigmoid for the output layer, x and y values are set to the X_train and y_train to represent the training and testing sets, respectively. Batch size is set to 10 for all

datasets, and epochs is set to 100 for first and second datasets, and 20 for the third dataset, Table (4.3).

Table (4.3): Parameter description for ANN model.

Parameter	Value	Description
Units	6	Number of neurons in the layer.
Input dim	11, 14	Number of predictors in the input data, set for the first layer only.
Kernel initializer	uniform	The algorithm determines the value for each weight before the Neurons begin their computation.
activation	ReLU/Sigmoid	This describes the calculation's activation function inside each neuron.
X	X_train	Features columns.
y	y_train	Target column.
batch size	10	Processing of several samples before updating the model.
epochs	100, 20	The dataset is run through the training process a fixed number of times.
Optimizer	Adam	This parameter assists in determining the optimal weight values for each node in the neural network.
loss	Binary_crossentropy	A function that assesses how effectively the neural network mimics the training data by comparing the target and estimated output values.

The values of the KNN model parameters are as follows: x and y values are set to the X_train and y_train to represent the training and testing sets, respectively. n_neighbors is set to 8 for the first and second datasets, and 4 for the third one, Table (4.4).

Table (4.4): Parameter description for KNN model.

Parameter	Value	Description
X	X_train	Features columns.
Y	y_train	Target column.
n_neighbors	8, 4	Required number of neighbors for each sample.

The values of the RF model parameters are as follows: x and y values are set to the X_train and y_train to represent the training and testing sets, respectively. n_estimator is set to 100 for all datasets by default, Table (4.5).

Table (4.5): Parameter description for RF model.

Parameter	Value	Description
X	X_train	Features columns.
y	y_train	Target column.
n_estimators	100	The number of trees in the forest.

The values of the SVM model parameters are as follows: x and y values are set to the X_train and y_train to represent the training and testing sets, respectively. kernel is set to 4 types for all datasets, each type has a run, table (4.6).

Table (4.6): Parameter description for SVM model.

Parameter	Value	Description
X	X_train	Features columns.
y	y_train	Target column.
kernel	Linear / poly / default=rbf / sigmoid	Defines the type of kernel to be used in the algorithm.

CNN was applied only for the third dataset; the values of the CNN model parameters are described in table (4.7).

Table (4.7): Parameters description for CNN model.

Parameter	Layer	Value	Description
filters	Conv1D	64	The layer's output dimension.
Kernel size	Conv1D	6	Defines the 1D convolution window's length.
Activation	Conv1D, Dense	ReLU, SoftMax	This describes the calculation's activation function inside each neuron.
Padding	Conv1D, MaxPooling1D	same	The quantity of pixels added to a picture during CNN's kernel processing.
Input shape	Conv1D	(14,1)	Shape of width and height.
Pool size	MaxPooling1D	3	Maximum pooling window size
Strides	MaxPooling1D	2	Indicates the amount that the pooling window shifts with each pooling step.
units	Dense	64, 5	Number of neurons in the layer.
optimizer	-	Adam	This parameter assists in determining the optimal weight values for each node in the neural network.
X	-	X_train	Features columns.
Y	-	y_train	Target column.
Batch size	-	10, 32, 70	Processing of several samples before updating the model.
epochs	-	50	The dataset is run through the training process a fixed number of times.
Validation data	-	X_test, y_test	A portion of data withheld from model training that is used to assess the model's performance while tuning its hyperparameters.
loss	-	categorical_crossentropy	A function that assesses how effectively the neural network fits the training data by comparing the target and predict output values.

4.4. Evaluation metrics

The performance of the classification method is defined using the findings of the confusion matrix. The performance of the models was measured using accuracy, precision, recall, and F1 measures. Formulas (4.1), (4.2), (4.3), and (4.4), respectively, show the formulas for calculating the categorization metrics.

$$\boldsymbol{Accuracy} = \frac{TP+TN}{TP+FP+TN+FN} \quad (4.1)$$

TP refers to True Positives, it means predicting fake user and its indeed fake user.

FP refers to False Positives, it means predicting real user and its indeed real user.

TN refers to True Negatives, it means predicting fake user and it is in fact real user.

FN refers to True Negatives, it means predicting real user and it is in fact fake user.

$$\boldsymbol{precision} = \frac{TP}{TP+FP} \quad (4.2)$$

$$\boldsymbol{recall} = \frac{TP}{TP+FN} \quad (4.3)$$

$$\boldsymbol{F1} = 2 * \frac{precision*recall}{precision+recall} \quad (4.4)$$

Table (4.8) shows the confusion matrix values.

Table (4.8): Confusion matrix.

	Predicted		
Actual		Negative	Positive
	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

The model's precision measures how accurate the model is, or how many of the positive results that were predicted are indeed positive. Recall analyzes how many true positives the model catches by classifying them as positive (true positive). The F1 Score is function of precision and recall, and it balance between both.

4.5. Stage 1 results

All of datasets were processed using ANN, RF, KNN, and SVM at this stage. The first dataset (Jafari, 2021) achieved the highest accuracy of 92.99% when processed using ANN. The second dataset (Bakhshandeh, 2019) achieved the highest accuracy of 92.85% when processed using KNN. The third dataset achieved the highest accuracy of 99.57% when processed using RF. The performance reached its lowest when SVM with a sigmoid kernel was used, Table (4.9). To determine the best value for k while using KNN, we applied it with a range from 2 to 11 and obtained accuracy results on a graph. For the first dataset, 8 was the best value, 8-9-10 for the second dataset and 4-6 for the third.

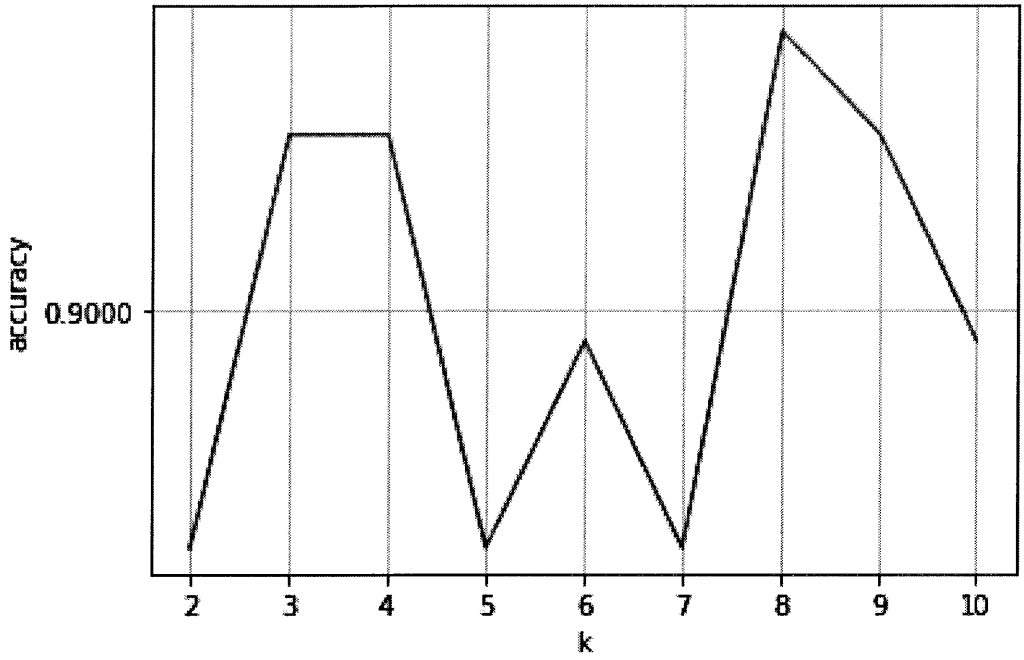


Figure (4.1): Accuracy depending on k value for 1st dataset.

The accuracy reaches its highest when k value is 8.

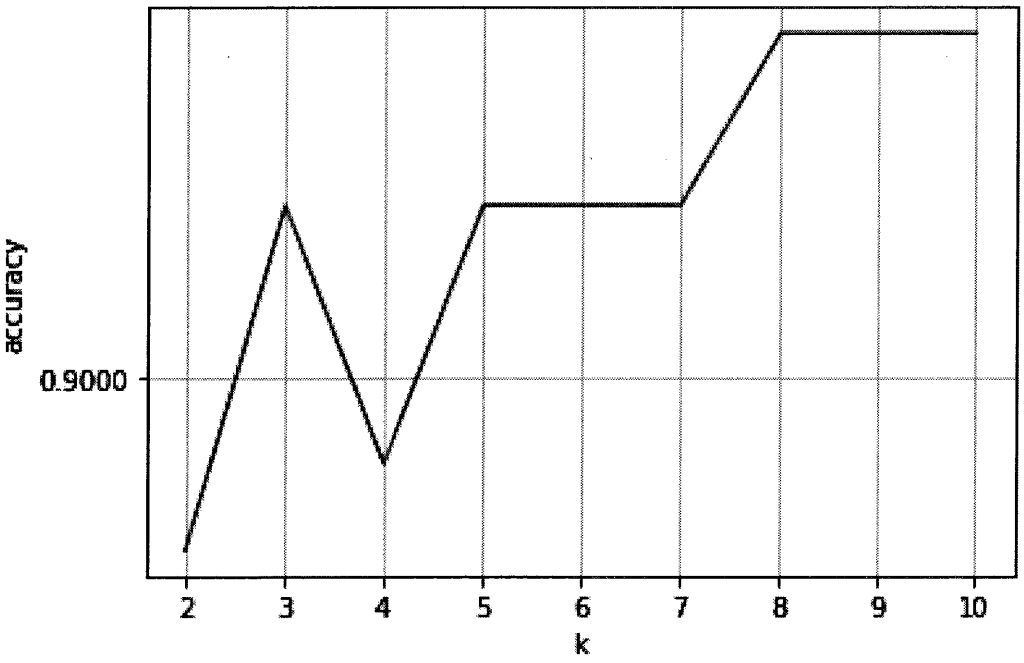


Figure (4.2): Accuracy depending on k value for 2nd dataset.

The accuracy reaches its highest when k value is 8, 9 and 10.

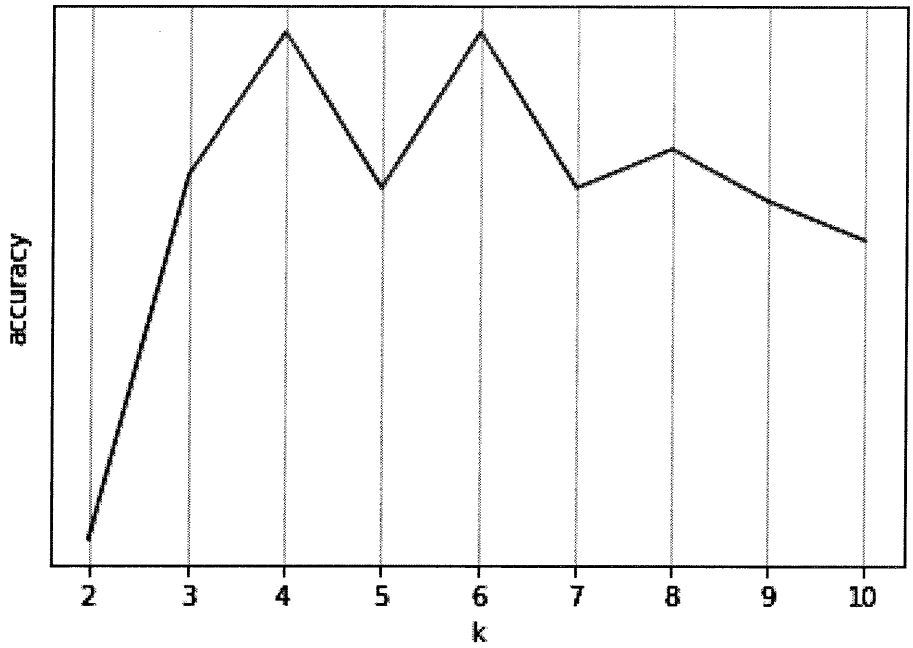


Figure (4.3): Accuracy depending on k value for 3rd dataset.

The accuracy reaches its highest when k value is 4 and 6.

Table (4.9): Accuracies for stage 1.

Dataset	ANN	RF	KNN	SVM (linear)	SVM (rbf)	SVM (poly)	SVM (sigmoid)
1	92.99%	92.35%	91.71%	92.35%	91.08%	92.35%	89.17%
2	88.57%	90.00%	92.85%	88.57%	91.42%	85.00%	88.57%
3	99.37%	99.57%	98.05%	99.30%	98.95%	98.26%	93.23%

As illustrated in the Table (4.10), the ANN model predicted 129 real samples out of 133 and 17 fake samples out of 24 in the first dataset, with 0.88 average precision, 0.84 average recall, and 0.86 average F1-score. It also predicted 62 real samples out of 73 and 62 fake samples out of 67 in the second dataset, with 0.89 average precision, 0.89 average recall, and 0.89 average F1-score. For the third dataset, ANN predicted 13574 real

samples out of 13655 and 2896 fake samples out of 2918, with 0.99 average precision, 0.99 average recall, and 0.99 average F1-score.

The KNN model predicted 127 real samples out of 133 and 17 fake samples out of 24 in the first dataset, with 0.84 average precision, 0.83 average recall, and 0.84 average F1-score. It also predicted 66 real samples out of 73 and 64 fake samples out of 67 in the second dataset, with 0.93 average precision, 0.93 average recall, and 0.93 average F1-score. For the third dataset, KNN predicted 13561 real samples out of 13655 and 2674 fake samples out of 2918, with 0.97 average precision, 0.95 average recall, and 0.96 average F1-score.

The RF model predicted 129 real samples out of 133 and 16 fake samples out of 24 in the first dataset, with 0.87 average precision, 0.82 average recall, and 0.84 average F1-score. It also predicted 67 real samples out of 73 and 59 fake samples out of 67 in the second dataset, with 0.90 average precision, 0.90 average recall, and 0.90 average F1-score. For the third dataset, KNN predicted 13593 real samples out of 13655 and 2909 fake samples out of 2918, with 0.99 average precision, 1.00 average recall, and 0.99 average F1-score.

For the SVM model, results are shown in Table (4.10) and Table (4.11), for all the kernels.

Table (4.10): Confusion matrix and classification report for stage 1.

ANN confusion matrix and classification report									
Dataset	TP	TN	FP	FN	Label	Precision	Recall	F1	Support
1	129	17	7	4	Real	0.81	0.71	0.76	24
					Fake	0.95	0.97	0.96	133
2	62	62	5	11	Real	0.85	0.93	0.89	67
					Fake	0.93	0.85	0.89	73
3	13574	2896	22	81	Real	0.97	0.99	0.98	2918
					Fake	1.00	0.99	1.00	13655
KNN confusion matrix and classification report									
Dataset	TP	TN	FP	FN	Label	Precision	Recall	F1	Support
1	127	17	7	6	Real	0.74	0.71	0.72	24
					Fake	0.95	0.95	0.95	133
2	66	64	3	7	Real	0.90	0.96	0.93	67
					Fake	0.96	0.90	0.93	73
3	13561	2674	244	94	Real	0.97	0.92	0.94	2918
					Fake	0.98	0.99	0.99	13655
RF confusion matrix and classification report									
Dataset	TP	TN	FP	FN	Label	Precision	Recall	F1	Support
1	129	16	8	4	Real	0.80	0.67	0.73	24
					Fake	0.94	0.97	0.96	133
2	67	59	8	6	Real	0.91	0.88	0.89	67
					Fake	0.98	0.92	0.91	73
3	13593	2909	9	62	Real	0.98	1.00	0.99	2918
					Fake	1.00	1.00	1.00	13655
SVM (linear) confusion matrix and classification report									
Dataset	TP	TN	FP	FN	Label	Precision	Recall	F1	Support
1	127	18	6	6	Real	0.75	0.75	0.75	24
					Fake	0.95	0.95	0.95	133
2	62	62	5	11	Real	0.85	0.93	0.89	67
					Fake	0.93	0.85	0.89	73
3	13590	2867	51	65	Real	0.98	0.98	0.98	2918
					Fake	1.00	1.00	1.00	13655

Table (4.11): Confusion matrix and classification report for stage 1, continued.

SVM (rbf) confusion matrix and classification report									
Dataset	TP	TN	FP	FN	Label	Precision	Recall	F1	Support
1	131	12	12	2	Real	0.86	0.50	0.63	24
					Fake	0.92	0.98	0.95	133
2	65	63	4	8	Real	0.89	0.94	0.91	67
					Fake	0.94	0.89	0.92	73
3	13582	2817	101	73	Real	0.97	0.97	0.97	2918
					Fake	0.99	0.99	0.99	13655
SVM (poly) confusion matrix and classification report									
Dataset	TP	TN	FP	FN	Label	Precision	Recall	F1	Support
1	129	16	8	4	Real	0.80	0.67	0.73	24
					Fake	0.94	0.97	0.96	133
2	54	65	2	19	Real	0.77	0.97	0.86	67
					Fake	0.96	0.74	0.84	73
3	13600	2685	233	55	Real	0.98	0.92	0.95	2918
					Fake	0.98	1.00	0.99	13655
SVM (sigmoid) confusion matrix and classification report									
Dataset	TP	TN	FP	FN	Label	Precision	Recall	F1	Support
1	125	15	9	8	Real	0.65	0.62	0.64	24
					Fake	0.93	0.94	0.94	133
2	62	62	5	11	Real	0.85	0.93	0.89	67
					Fake	0.93	0.85	0.89	73
3	13071	2381	537	584	Real	0.80	0.82	0.81	2918
					Fake	0.96	0.96	0.96	13655

4.6. Stage 2 results

This stage was applied to the third dataset only using CNN algorithm, this stage has 3 substages, in the first substage we have applied CNN on the dataset without resampling the data (unbalanced dataset), in the second stage we have reduced the dataset (balanced dataset), in the third stage we have increased the data with SMOTE (balanced dataset).

4.6.1. Before resampling:

With the original unbalanced dataset, we applied the CNN algorithm in this substage. The validation accuracy was 99.99% with 0.09% validation loss when using 10 as the batch size. With 32 batches, the validation accuracy was 90.81% and the validation loss was 27.30%. Applying 70 batches resulted in acceptable validation accuracy of 99.96% and validation loss of 0.21%. The model predicted 2918 real samples out of 2918 samples when applying a batch size of 10. It also predicted 2721 fake1 samples out of 2722 samples, 6976 fake2 samples out of 6976 samples, 1611 fake3 samples out of 1612 samples, and 2218 fake4 samples out of 2218 samples. With 1.00 F1 scores, 1.00 recall, and 1.00 precision for all labels.

Table (4.12): Training and Validation Acc-Loss, No resampling.

Before Resampling 14 features					
Epoch	Batch	Accuracy	loss	Val accuracy	Val loss
50	70	99.73%	0.82%	99.96%	0.21%
50	32	93.88%	16.66%	90.81%	27.30%
50	10	99.65%	1.20%	99.99%	0.09%

Table (4.13): Confusion Matrix and classification report, No resampling.

CNN confusion matrix and classification report -10 batch										
		Predicted					Label	Precision	Recall	F1	Support
		Real	Fake1	Fake2	Fake3	Fake4					
Actual	Real	2918	0	0	0	0	Real	1.00	1.00	1.00	2918
	Fake1	1	2721	0	0	0	Fake1	1.00	1.00	1.00	2722
	Fake2	0	0	6976	0	0	Fake2	1.00	1.00	1.00	6976
	Fake3	1	0	0	1611	0	Fake3	1.00	1.00	1.00	1612
	Fake4	0	0	0	0	2218	Fake4	1.00	1.00	1.00	2218

According to the accuracy and loss graphs for 70 batch sizes, there were some fluctuations in the validation accuracy and loss up until epoch 40, after which everything went smoothly. When using a 32-batch size, validation accuracy and loss varied; starting at epoch 37, the fluctuation grew and became obvious. When using batch sizes of 10, validation accuracy and loss fluctuated. After that, everything went smoothly until epoch 41, when a significant drop was noticed.

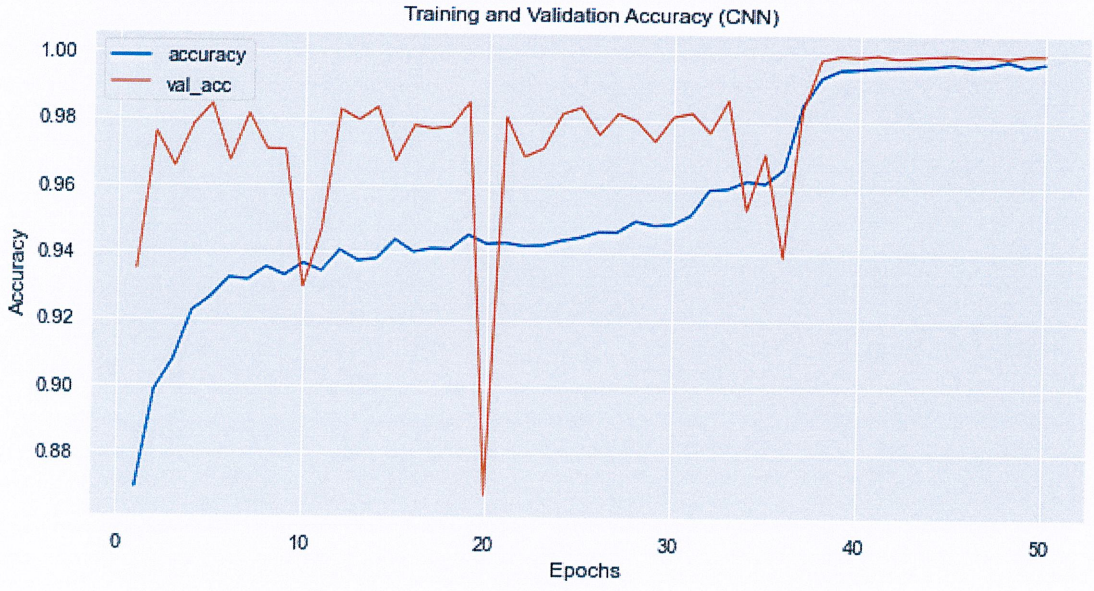


Figure (4.4): Accuracy for 70 batch with no resampling.

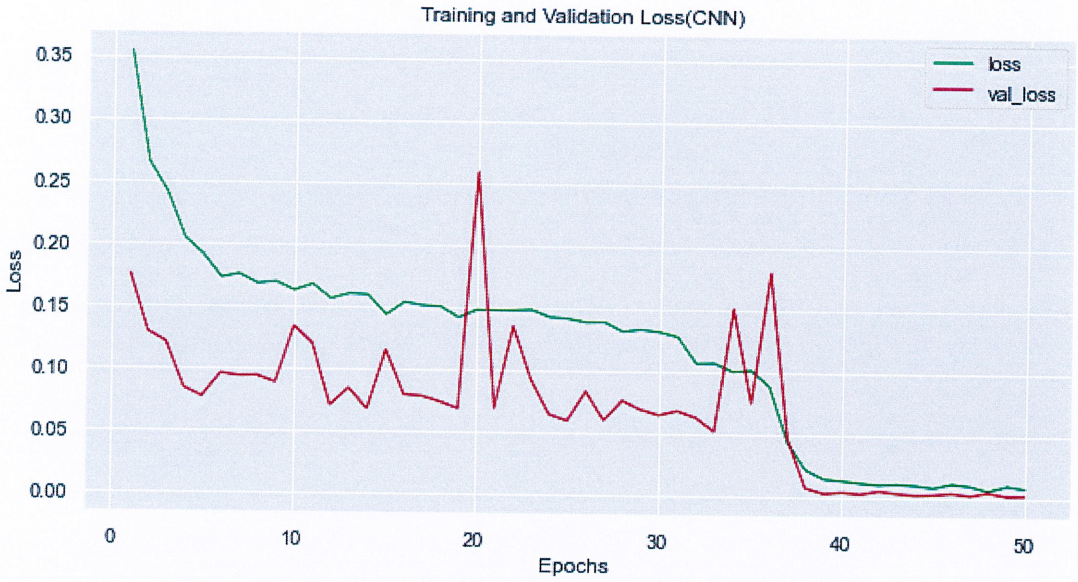


Figure (4.5): Loss for 70 batch with no resampling.

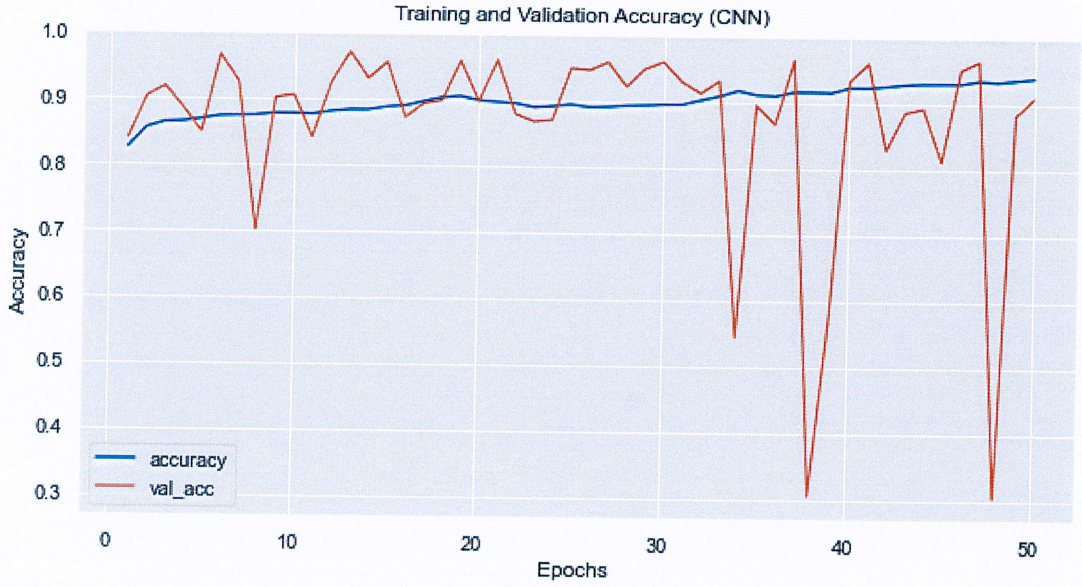


Figure (4.6): Accuracy for 32 batch with no resampling.

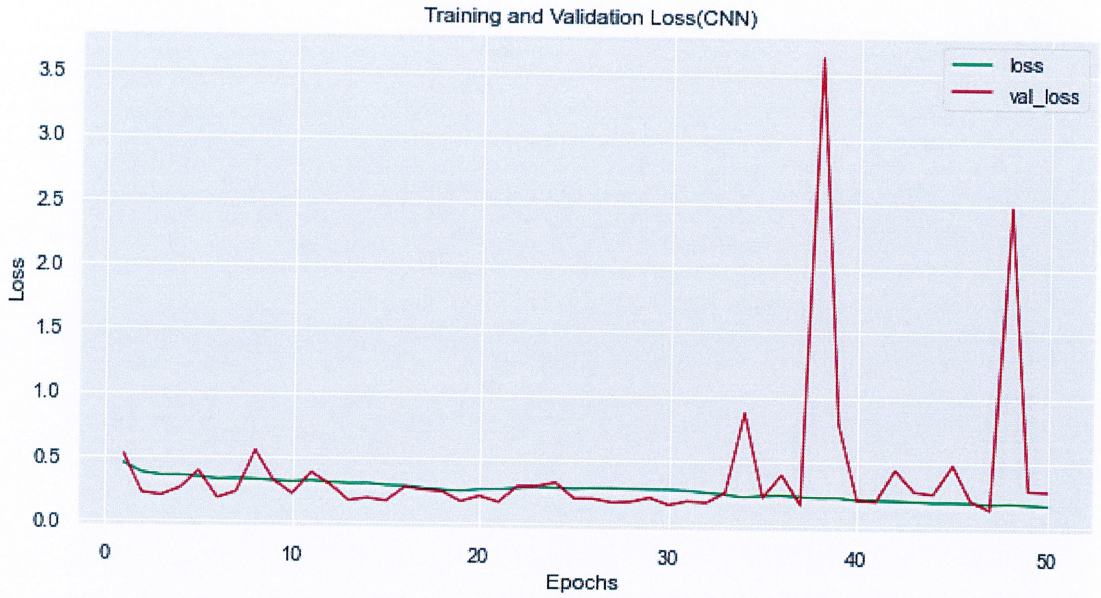


Figure (4.7): Loss for 32 batch with no resampling.

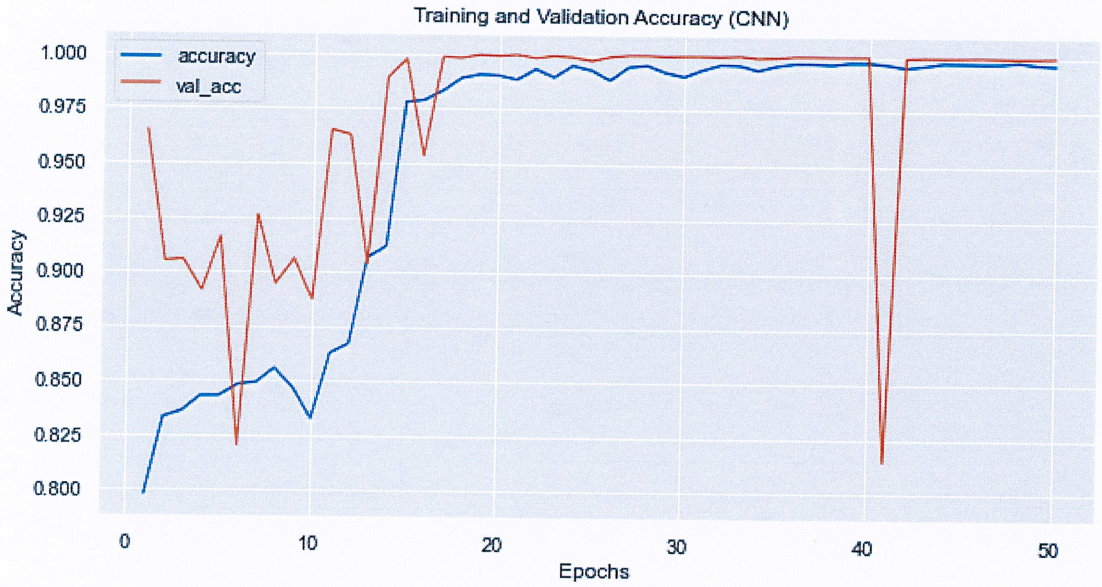


Figure (4.8): Accuracy for 10 batch with no resampling.

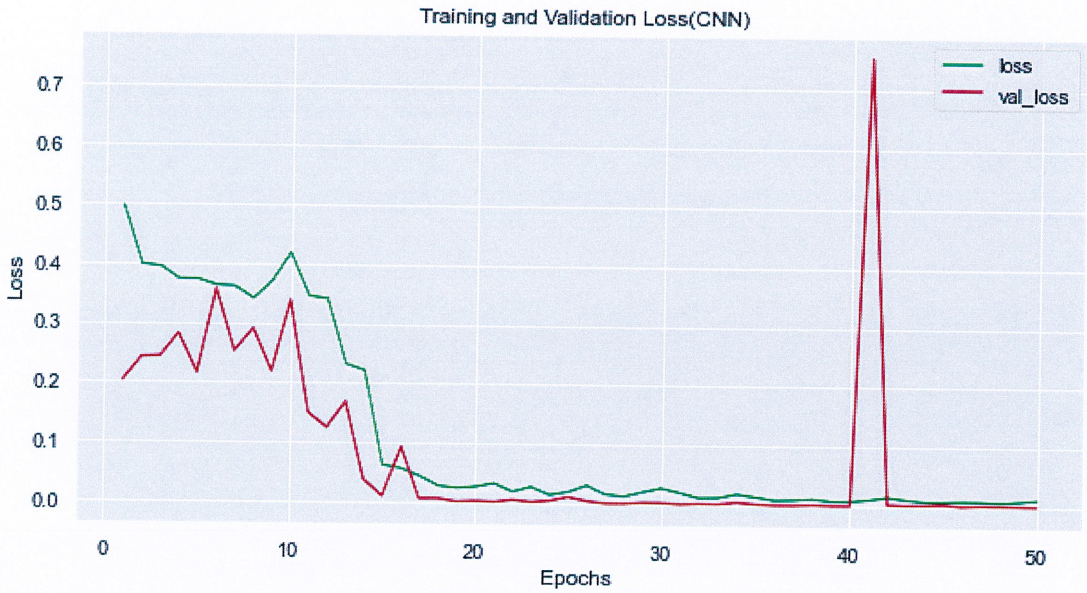


Figure (4.9): Loss for 10 batch with no resampling.

4.6.2. Down sampling results:

The dataset was reduced from 82228 to 35000, with balanced datapoints for all labels and 7000 rows per label. The validation accuracy in batch 70 was, in the best case, 96.47% with a 10.62% validation loss. Validation accuracy after applying batch 32 was 94.43%, while validation loss was 19.04%. Applying 10 batches produced a validation accuracy of 96.67% and a validation loss of 13.13%. With a batch size of 70, the model correctly predicted 1371 real samples out of 1392, with a precision, recall, and F1 score of 0.98. With 0.91 precision, 0.99 recall, and 0.95 F1 score, it also correctly predicted 1360 fake1 samples out of 1370. With 0.98 precision, 0.97 recall, and a 0.98 F1 score, the model correctly predicted 1331 of the 1366 fake2 samples. With 0.98 precision, 0.99 recall, and a 0.98 F1 score, 1400 fake3 samples out of 1421 were predicted. With 0.99 precision, 0.89 recall, and a 0.94 F1 score, 1289 fake4 samples out of 1451 were predicted.

Table (4.14): Training and Validation Acc-Loss, Down sampling.

Resampling (down sampling)					
Epoch	Batch	Accuracy	loss	Val accuracy	Val loss
50	70	94.00%	16.22%	96.47%	10.62%
50	32	93.05%	18.63%	94.43%	19.04%
50	10	93.95%	16.66%	96.67%	13.13%

Table (4.15): Confusion Matrix and classification report, Down sampling.

CNN confusion matrix and classification report - 70 batch										
		Predicted					Label	Precision	Recall	F1	Support
		Real	Fake1	Fake2	Fake3	Fake4					
Actual	Real	1371	14	3	2	2	Real	0.98	0.98	0.98	1392
	Fake1	4	1360	0	0	6	Fake1	0.91	0.99	0.95	1370
	Fake2	6	0	1331	20	9	Fake2	0.98	0.97	0.98	1366
	Fake3	21	0	0	1400	0	Fake3	0.98	0.99	0.98	1421
	Fake4	7	125	30	0	1289	Fake4	0.99	0.89	0.94	1451

According to the accuracy and loss graph when applying 70 batch size there was a clear fluctuation that was much clearer on epoch 26 with 72.07% validation accuracy and 57.11% validation loss. When applying batch 32 and 10 there was a strong fluctuation that didn't stop until epoch 50.

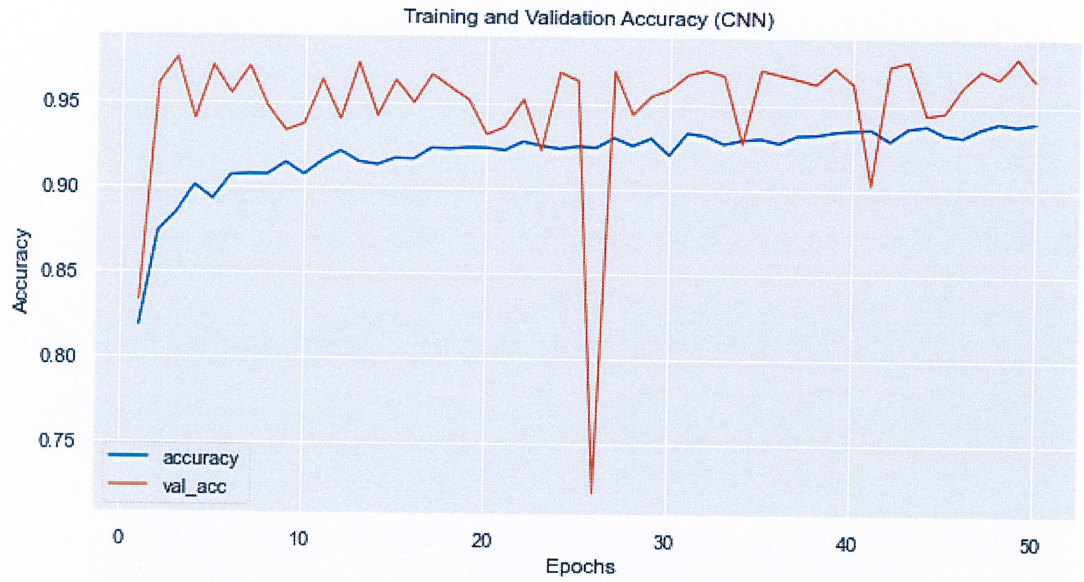


Figure (4.10): Accuracy for 70 batch with down sampling.

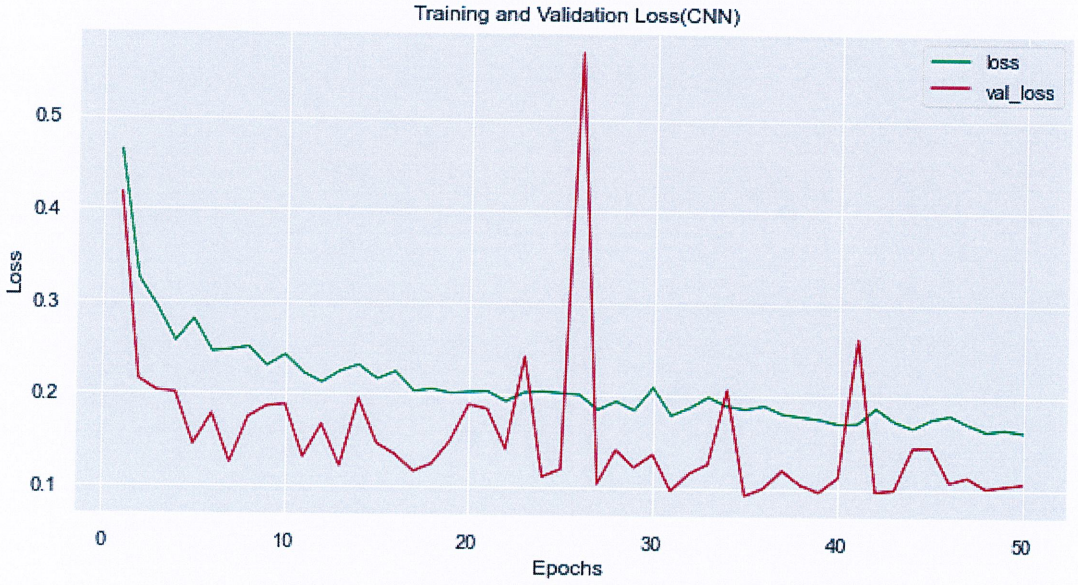


Figure (4.11): Loss for 70 batch with down sampling.

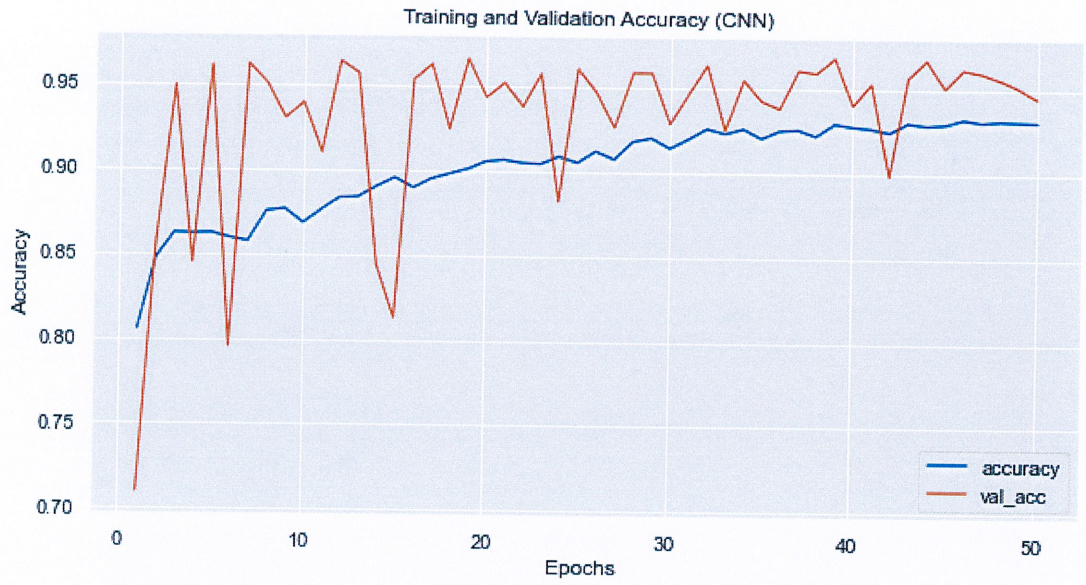


Figure (4.12): Accuracy for 32 batch with down sampling.

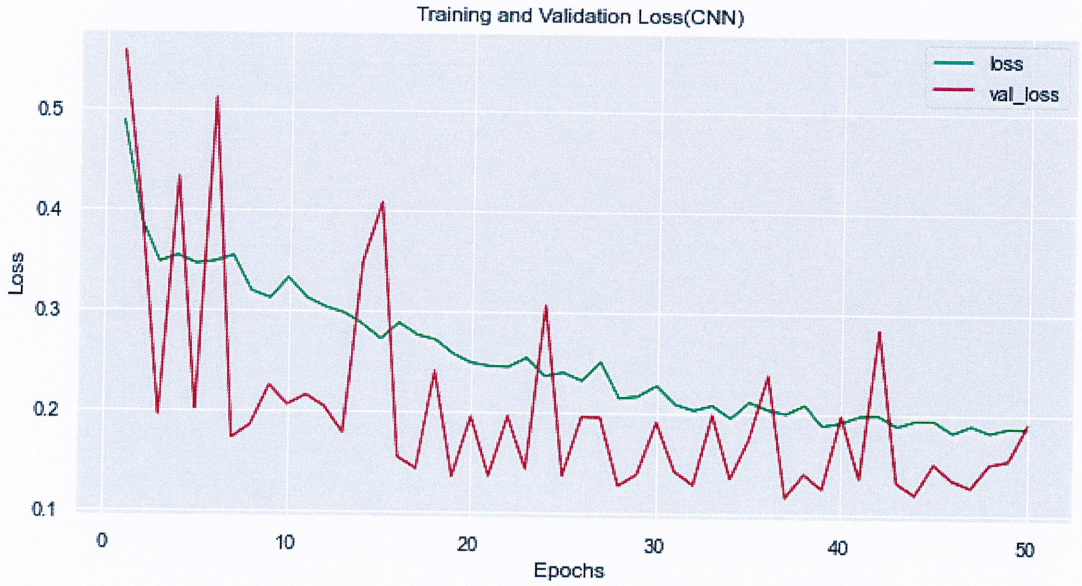


Figure (4.13): Loss for 32 batch with down sampling.

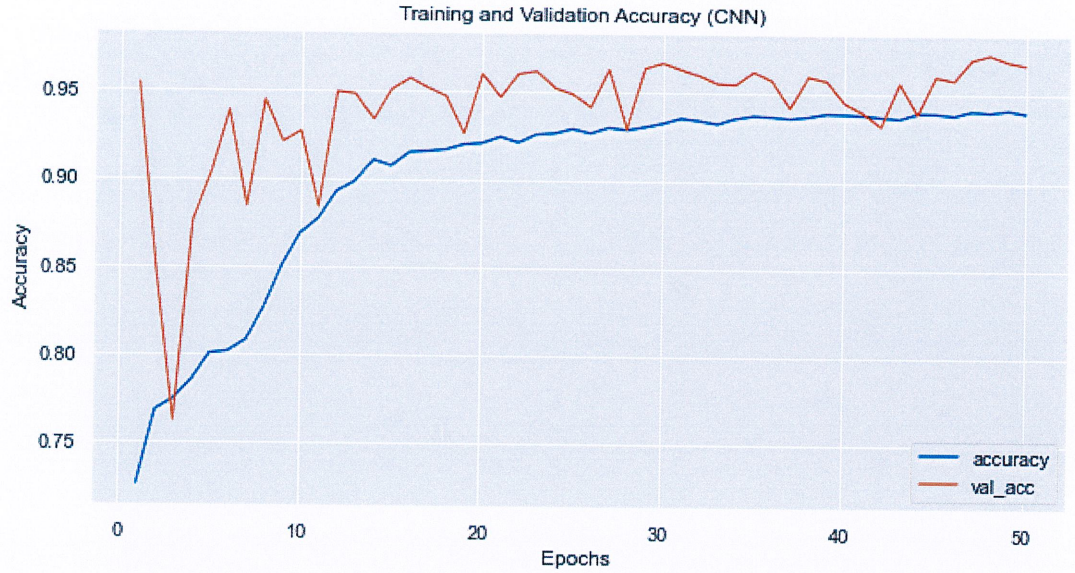


Figure (4.14): Accuracy for 10 batch with down sampling.

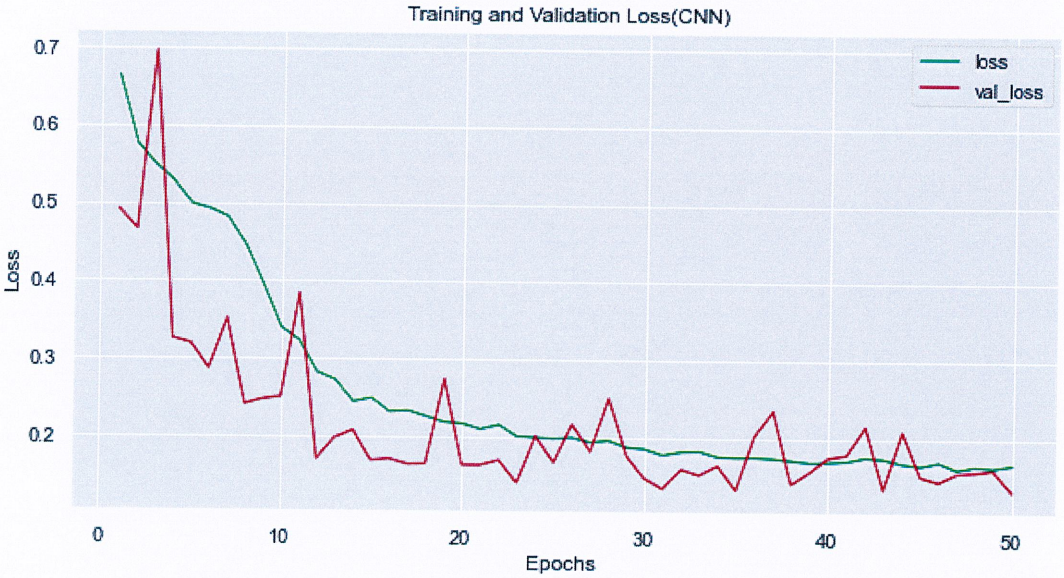


Figure (4.15): Loss for 10 batch with down sampling.

4.6.3. Oversampling (SMOTE) results:

We divided the 82228-row original dataset into training and testing sets before oversampling. Training data increased from 65782 to 137360 rows with balanced datapoints for all labels and 27472 rows per label, while the testing set had 16446 rows. Batch 70 had a 99.98% validation accuracy and a 0.31% validation loss. When a batch size of 32 was used, validation accuracy was 99.98% with 0.14% validation loss. When using a batch size of 10, validation accuracy was 99.99% with a validation loss of 0.70%. The model predicted 2918 real samples out of 2918 and 2720 fake1 samples out of 2722. Out of 6976 fake2 samples, the model predicted 6976. The model predicted 1610 fake3 samples out of 1612 and 2218 fake4 samples out of 2218. The F1 score, recall, and precision were all 1.00.

Table (4.16): Training and Validation Acc-Loss, Oversampling.

Oversampling					
Epoch	Batch	Accuracy	loss	Val accuracy	Val loss
50	70	99.46%	1.50%	99.98%	0.31%
50	32	99.40%	1.82%	99.98%	0.14%
50	10	99.82%	0.43%	99.99%	0.70%

Table (4.17): Confusion Matrix and classification report, Oversampling.

CNN confusion matrix and classification report- 70 batch										
		Predicted					Label	Precision	Recall	F1	Support
		Real	Fake1	Fake2	Fake3	Fake4					
Actual	Real	2918	0	0	0	0	Real	1.00	1.00	1.00	2918
	Fake1	2	2720	0	0	0	Fake1	1.00	1.00	1.00	2722
	Fake2	0	0	6976	0	0	Fake2	1.00	1.00	1.00	6976
	Fake3	1	0	0	1610	1	Fake3	1.00	1.00	1.00	1612
	Fake4	0	0	0	0	2218	Fake4	1.00	1.00	1.00	2218

The validation accuracy reached 99% at epoch 7, and the validation loss was almost 0%, going smoothly and without fluctuations, according to the accuracy and loss graphs when 70 batch sizes were applied. Applying batch 32 resulted in 99% validation accuracy from epoch 10 and almost 0% validation loss. With a batch size of 10, there was a noticeable fluctuation up until epoch 23, at which point the validation accuracy reached 99% and the validation loss was almost 0%. From that point on, everything went well until Epoch 50.

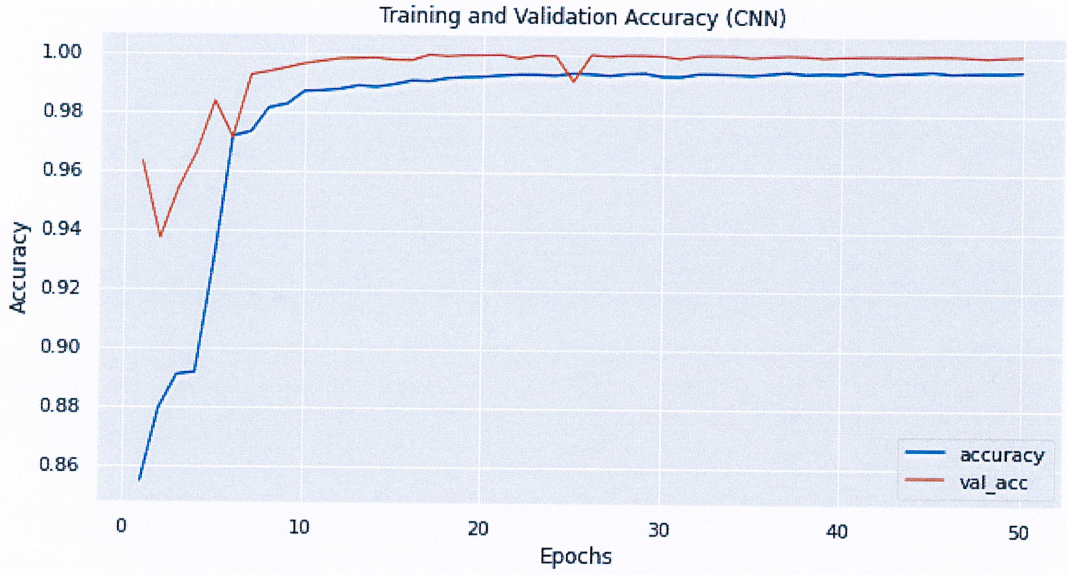


Figure (4.16): Accuracy for 70 batch with oversampling.

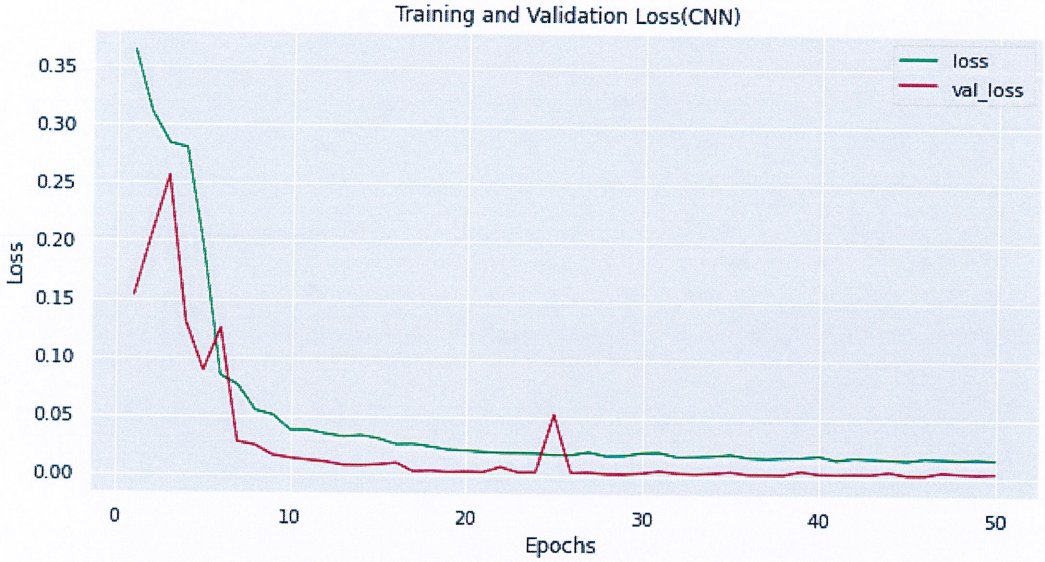


Figure (4.17): Loss for 70 batch with oversampling.

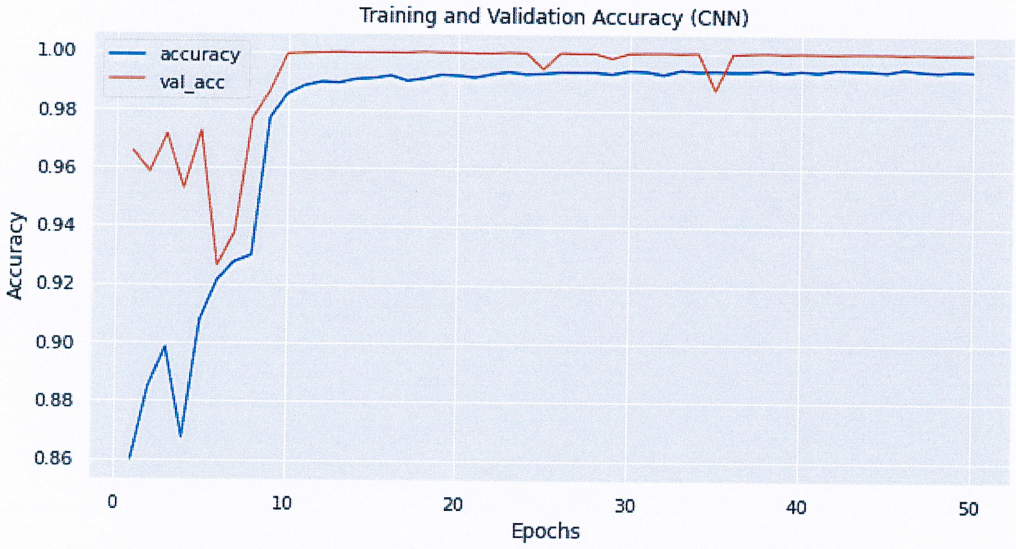


Figure (4.18): Accuracy for 32 batch with oversampling.

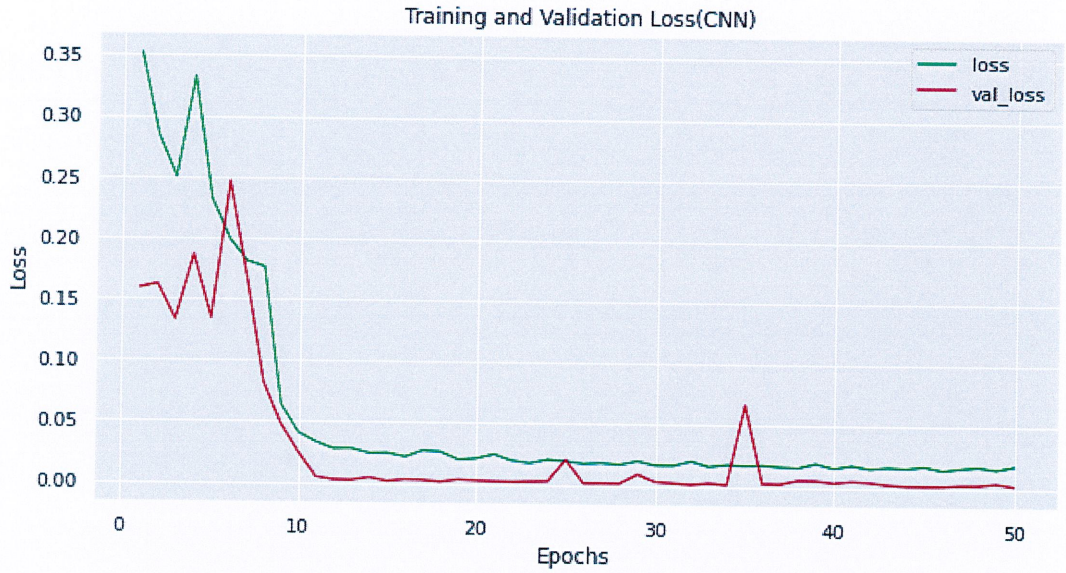


Figure (4.19): Loss for 32 batch with oversampling.

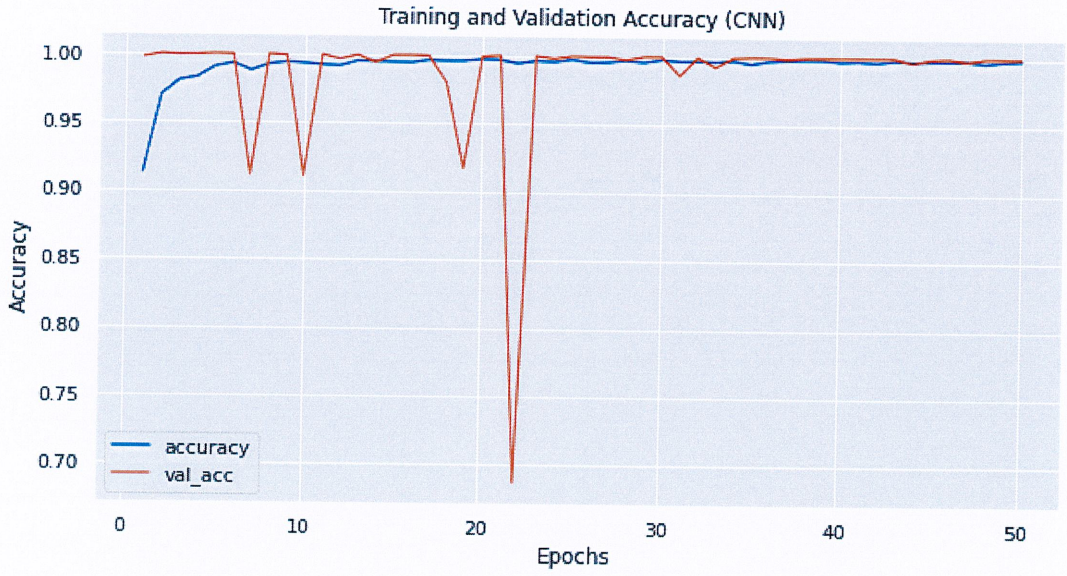


Figure (4.20): Accuracy for 10 batch with oversampling.

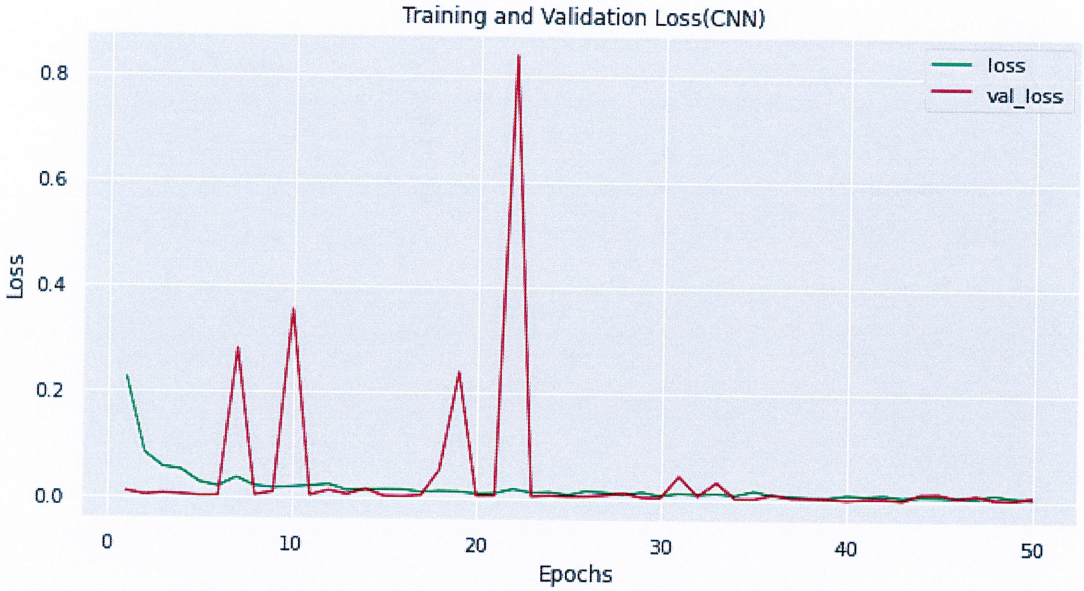


Figure (4.21): Loss for 10 batch with oversampling.

4.7. Comparative study

A comparison of another related research is shown in Table (4.18). Most of them employed machine learning algorithms and produced good results, but deep learning algorithms, such as CNN, outperformed the typical machine learning model. In our proposed work, the 3rd dataset was unbalanced and contained two labels (real and fake) in stage 1, but it still outperformed datasets 1 and 2 because of its larger size. In stage 2, it was evident that resampling altered the outcomes and that the quantity of datasets affected how well the deep learning algorithm operated.

According to several studies, the performance of convolutional neural network (CNN) models for classification tasks is significantly impacted by the quantity of the dataset. Bigger datasets often produce better results since they give the model more training data and let it develop more strong features. The batch size employed during training can also have an influence on how well a CNN performs, with bigger batches often producing better accuracy. The model in stage 2 with no resampling

worked well, with a little bit of fluctuation in validation accuracy and loss, when we applied down sampling the fluctuation increased obviously, and the best results we've got when we applied oversampling with a batch size of 70 with a validation accuracy of 99.98% and a validation loss of 0.31%.

From the graph below, the 3rd dataset performed very well in each model, but it reached its best performance with CNN when oversampling is applied.

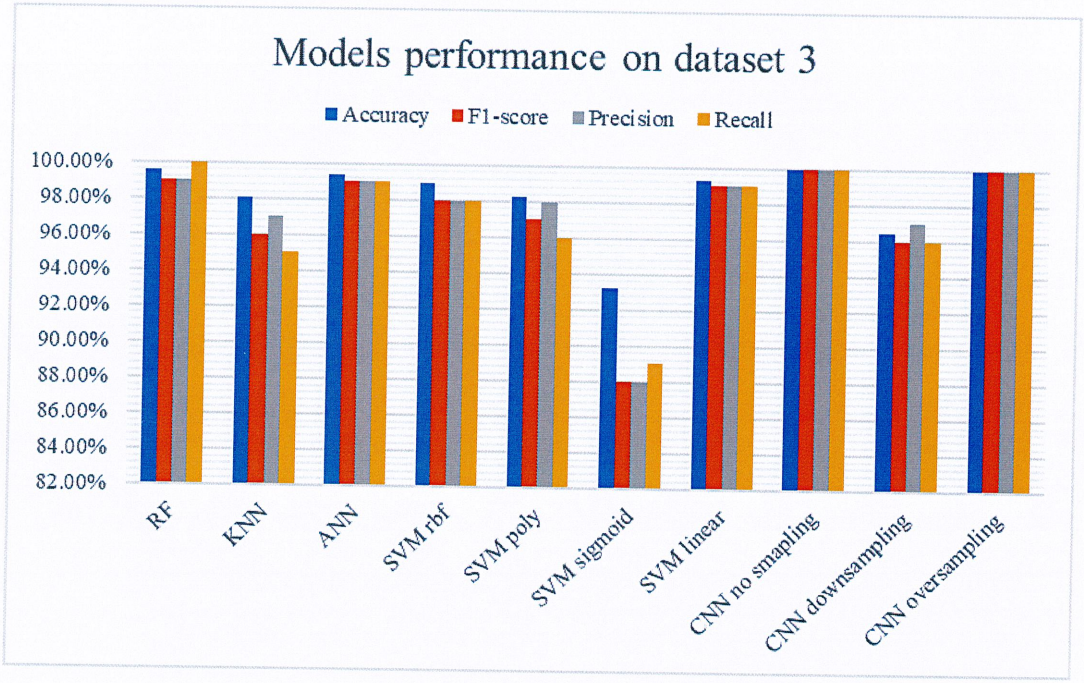


Figure (4.22): Performance of algorithms.

Datasets were gathered by Rostami et al., (2020), Nasir et al., (2021), Suganya, et al. (2021), and Awan et al., (2022) from public domain websites like GitHub and Kaggle. On the other hand, Purba et al., (2020) independently gathered datasets, purchased fake users from online websites, and gathered real users from private universities. The dataset was manually assembled and annotated by Akyon et al., (2020). S B, (2022) also used scraping profiles because there was no publicly accessible informal community dataset for the investigation's purpose of identifying

the fake accounts. Since there aren't enough datasets for training and social media has a huge user base, researchers need to gather as many as they can, especially for Instagram datasets because of the lack of Instagram datasets.

Suganya, et al. (2021), and Akyon et al., (2020) agreed that oversampling is necessary to improve the performance of the model, and Akyon et al., (2020) used SMOTE oversampling with an equal number of training samples of 1002 for each class to train all classifiers. Suganya, et al. (2021) used methods like bootstrapping and repetition to generate additional observations and samples, which she then connected to the underrepresented groups. To balance the two distributions (bot vs. normal) by averaging the bot class distribution, Shevtsov et al., (2022) applied the synthetic minority oversampling technique while performing the 5-fold cross-validation process on the training folds.

Bordbar et al., (2022) used PCA and GAN for the unbalanced dataset, while Purba et al., (2020) used down sampling and removed some users to keep the dataset balanced. The biased features in datasets can be balanced by finding the appropriate real users, according to Akyon et al., (2020), who explained that time series user data can be used for better detection of fake accounts.

Data normalization is necessary to prevent bias in data because when data is not normalized, the model will give larger weights for larger scales of data. Some researchers, such as Suganya, et al., (2021) and Akyon et al., (2020), who applied machine learning algorithms, have also applied normalization for datasets. To overcome these unjustified biases and choose the most useful features, a cost-sensitive genetic feature selection algorithm was developed. Akyon et al., (2020) encountered some negative bias in some of the features. They started by normalizing the continuous

features while leaving the binary features alone. The genetic algorithm is then given the normalized data to select a cost-sensitive feature set.

For ANN, KNN, RF, and SVM, standard scalers have been used in our work, and the batch normalization layer is used to apply normalization between layers in the deep learning algorithm of CNN.

Rostami et al., (2020) concentrated on the feature selection process by identifying the most useful feature set using a multi-hybrid approach. The number of features may increase the computational cost, but by reducing the features, the computational cost will be reduced, and the performance of the model will increase, so feature reduction is a crucial step to pay attention to if it's possible. To extract significant features, Shevtsov et al., (2022) used the Shapley values based on game theory. Additionally, S B, (2022) used features reduction. For the feature extraction process, Ali & Abdullah, (2022) used the Chi-square test and the Spearman rank correlation.

Munga & Mohandas, (2022) showed that removing unnecessary features will improve the model's accuracy and save time. They used all 25 features in the first round, and the second round's features were reduced to just 5 features based on the information gained from the features. Different algorithms were used to further demonstrate how the smallest set of features with the highest information gain results in the best accuracy. Using the correlation map, we have applied feature reduction to datasets 1 and 3. Rostami et al., (2020) showed how the model could be used in the future with a variety of different features.

The publicly accessible Instagram developer API only makes the user's publicly accessible data available. Most authors also agreed that it is not possible to access some users' actions or login information when they have

already turned their accounts to private mode. It is thought that this issue makes gathering data a bit harder.

Sheikhi, (2020) has gathered users from Instagram's explore function using a crawler to identify regular users and add them to the dataset's list of regular users. It is simpler to gather public accounts from Instagram's Explore because it only shares media data and posts from public users and highlights recently posted images and videos that have attracted other users' attention. This shows that the accounts being used to post are generally authentic and reliable.

And if a researcher wants to collect a dataset that includes any kind of user, whether they are public or private, they should only acquire metadata data. Even if it were possible to obtain media data from individual users, doing so would require immoral methods like hacking accounts, which is not advised. Bordbar et al., (2022), Varuna et al., (2022), and Swathi et al., (2021) addressed the problem of insufficient data as well as the moral concerns with data collection.

Because using techniques other than binary classification makes it possible to extract more relevant insights from large datasets, delivering better results and more accurate forecasts, we employed clustering in stage 2 for the third dataset to acquire 5-class classifications. However, Purba et al., (2020) applied binary and 4-class classifications and found that they produced better results than users who simply used binary classification to distinguish between fake and real users.

Whether a user's account is private or not, we solely gathered publicly available data about them for this study. If given enough datasets, deep learning algorithms can yield models with remarkable outcomes. The outcomes of our model were better than we were expecting, and it was

developed and run in an appropriate amount of time. Our model's accuracy performance was the best when compared to previous studies, as shown in Table (4.18).

Table (4.18): Previous studies comparison with proposed study.

Study	Method	Fake accounts detection	Fake news detection	Accuracy
Proposed work	KNN, ANN, SVM, RF, CNN	√		98.05%, 99.37%, 99.30%, 99.57%, 99.98%
(Suganya, et al. 2021)	Ada-boost SVM	√		97.16%
(Rostami et al., 2020)	RF, NB, SVM	√		98.00%, 97.6%, 98%
(Purba et al., 2020).	RF, Multilayer perceptron, LR, NB, J48 DT	√		91.76%, 73.7%, 68.1% 54.2%, 88.3%
(Akyon et al., 2020).	SVM, NB, LR, and NN	√		94.00%, 88.2%, 90.8%, 94.0%
(Sheikhi, 2020)	Bagged decision Tree	√		98.45%
(Nasir et al., 2021).	CNN, RNN		√	99.00%
(Uppada et al., 2022).	SENAD, CredNN, CNN	√	√	93.50%
(Ali & Abdullah, 2022)	Stack ensemble system, RF, SVM, NB, LR	√		99.01%, 98.86%, 91.32% 76.39%, 67.49%
(Munga & Mohandas, 2022)	RF, SVM, NN	√		99.46%, 99.64%, 94.13%
(Shevtsov et al., 2022)	RF, SVM, XGBoost	√		97.30%, 96.4%, 97.9%
(S B, 2022)	RF	√		93.00%
(Awan et al., 2022)	Spark ML, RF	√		94.00%
(Bordbar et al., 2022)	GAN, LR, SVM.	√		98.00%, 66.3%, 79.5%
(Varuna et al., 2022)	LSTM	√		97.90%
(Swathi et al., 2021)	NN	√		93.00%
(Wanda, 2022)	CNN	√		95.47%
(Sahoo & Gupta, 2021)	KNN, SVM, LR, DT, NB, LSTM		√	99.3%, 99.3%, 99.0%, 99.1% 98.6%, 99.40%

Chapter Five

Conclusion and Future work

5.1. Overview

The final chapter of this thesis study is chapter five. An overall summary of the thesis study is provided in the chapter. The study's contributions are discussed, and the key conclusions of the suggested models are highlighted. It also contains suggestions for further research and the conclusion.

5.2. Conclusion

The detection of fake accounts is essential for preserving the integrity of social media platforms since they have grown to be a major problem. An individual who creates a fake account does so with the goal of concealing their genuine identity or running a scam. Fake accounts can be used for a variety of nefarious activities, including spamming, phishing, and the dissemination of disinformation.

Algorithms based on AI can be trained to spot patterns in user behavior and alert users to suspect accounts. To spot fake accounts, these algorithms can examine a variety of variables, including user activity, content, and network structure. A machine learning algorithm, for instance, can highlight accounts that have many followers but few interactions or accounts that have shared the same content repeatedly. According to our literature review, several researchers attempted to collect datasets and extract valuable features to develop a successful model that could distinguish between fraudulent and legitimate accounts. This topic has drawn the attention of many academics.

Working with Instagram platform to detect fake accounts is recent, and there are few studies that implemented Fake accounts detection on Instagram. CNN would not benefit from our discovery of two small Instagram datasets, as CNN require large amount of data, so we created our own dataset using IG Exporter and APIFY. We then applied k-mean clustering to the dataset and obtained 5 classification classes.

Our work has been separated into two stages. In the first stage, we used ANN, KNN, RF, and SVM on three datasets. After preprocessing, cleaning the data, and running the models, the accuracy of the 3rd dataset while using RF was the greatest, coming in at 99%. In the second step, we applied CNN to our dataset. After numerous tries without resampling, down sampling, we encountered significant fluctuation in the results. This problem was resolved with oversampling using SMOTE, which produced the greatest accuracy of 99.98%.

We found that CNN is a strong method for dealing with vast numerical data in one dimension. A CNN may assist to extract important patterns and features from data and achieve advanced performance on a wide range of applications by using local feature extraction, hierarchical feature learning, regularization using dropout layer to prevent overfitting, so it can outperform typical machine learning algorithms.

Also, when the dataset is small and unbalanced, SMOTE can improve CNN performance since CNN may not have enough examples of the minority class to learn it's features. In these instances, SMOTE may be used to oversample the minority class and produce new examples, allowing CNN to learn a more accurate features of the minority class.

5.3. Future work

Building new linguistic features as opposed to numerical ones would be a new step in our research to see how well AI can identify fake accounts. Collecting additional data and extracting new features would also help the model perform better. Although there are numerous methods and tools for spotting fake accounts, it's crucial to note that no single strategy is infallible. For the best outcomes in the future, a variety of strategies and instruments should be utilized.

References

Prabhu Kavin, B., Karki, S., Hemalatha, S., Singh, D., Vijayalakshmi, R., & Thangamani, M. et al. (2022). Machine Learning-Based Secure Data Acquisition for Fake Accounts Detection in Future Mobile Communication Networks. *Wireless Communications And Mobile Computing, 2022*, 1-10. doi: 10.1155/2022/6356152

Purba, K., Asirvatham, D., & Murugesan, R. (2020). Classification of instagram fake users using supervised machine learning algorithms. *International Journal Of Electrical And Computer Engineering (IJECE), 10*(3), 2763. doi: 10.11591/ijece.v10i3.pp2763-2772

Masood, F., Ammad, G., Almogren, A., Abbas, A., Khattak, H., & Ud Din, I. et al. (2019). Spammer Detection and Fake User Identification on Social Networks. *IEEE Access, 7*, 68140-68152. doi: 10.1109/access.2019.2918196

Nasir, J., Khan, O., & Varlamis, I. (2021). Fake news detection: A hybrid CNN-RNN based deep learning approach. *International Journal Of Information Management Data Insights, 1*(1), 100007. doi: 10.1016/j.jjimei.2020.100007

Rostami, R. R., & Karbasi, S. (2020). Detecting Fake Accounts on Twitter Social Network Using Multi-Objective Hybrid Feature Selection Approach. *Webology, 17*(1).

Akyon, F. C., & Kalfaoglu, M. E. (2019, October). Instagram fake and automated account detection. In *2019 Innovations in intelligent systems and applications conference (ASYU)* (pp. 1-7). IEEE.

Kaur, E., & Bhandari, D. (2021). FAKE SOCIAL PROFILE DETECTION USING MACHINE LEARNING. *Science, Technology And Development.*

Suganya, R., Muthulakshmi, S., Venmuhilan, B., Kumar, K. V., & Vignesh, G. (2021). Retraction: Detect fake identities using improved machine learning algorithm (J. Phys.: Conf. ser. 1916 012056). *Journal of Physics: Conference Series, 1916*(1), 012296. doi:10.1088/1742-6596/1916/1/012296

Uppada, S., Manasa, K., Vidhathri, B., Harini, R., & Sivaselvan, B. (2022). Novel approaches to fake news and fake account detection in OSNs: user social engagement

and visual content centric model. *Social Network Analysis And Mining, 12*(1). doi: 10.1007/s13278-022-00878-9

Ferrara, E. (2022). Twitter Spam and False Accounts Prevalence, Detection and Characterization: A Survey. arXiv preprint arXiv:2211.05913.

Ali, A. K., & Abdullah, A. M. (2022). Fake accounts detection on social media using Stack Ensemble System. *International Journal of Electrical and Computer Engineering (IJECE), 12*(3), 3013. doi:10.11591/ijece.v12i3.pp3013-3022

Munga, J. B., & Mohandas, P. (2022, April). Feature Selection for Identification of Fake Profiles on Facebook. In *6th Kuala Lumpur International Conference on Biomedical Engineering 2021: Proceedings of BioMed2021, July 28-29, 2021* (pp. 489-497). Cham: Springer International Publishing.

Shevtsov, A., Tzagkarakis, C., Antonakaki, D., & Ioannidis, S. (2022, May). Identification of Twitter Bots Based on an Explainable Machine Learning Framework: The US 2020 Elections Case Study. In Proceedings of the International AAAI Conference on Web and Social Media (Vol. 16, pp. 956-967)

S B, S. (2022). Use of ann to identify fake profiles. *International Journal for Research in Applied Science and Engineering Technology, 10*(4), 2689-2694. doi:10.22214/ijraset.2022.41867

Awan, M. J., Khan, M. A., Ansari, Z. K., Yasin, A., & Shehzad, H. M. (2022). Fake profile recognition using big data analytics in social media platforms. *International Journal of Computer Applications in Technology, 68*(3), 215. doi:10.1504/ijcat.2022.124942

Ghafoor, H. Y., Jaffar, A., Jahangir, R., Iqbal, M. W., & Abbas, M. Z. (2022). Fake news identification on social media using Machine Learning Techniques. *Lecture Notes in Networks and Systems,* 87-98. doi:10.1007/978-981-16-7618-5_8

Bordbar, J., Mohammadrezaie, M., Ardalan, S., & Shiri, M. E. (2022). Detecting fake accounts through Generative Adversarial Network in online social media. *arXiv preprint arXiv:2210.15657.*

Varuna, W. R., Shalini, K., & Roy, M. E. A. (2022). An Efficient Framework for Fake Profile Identification Using Metaheuristic and Deep Learning Techniques. Journal of Positive School Psychology, 3741-3750.

Swathi, M., Anoop, A., & Rudra, B. (2021). Fake profile detection and stalking prediction on Facebook. *Advances in Intelligent Systems and Computing,* 13-21. doi:10.1007/978-981-16-1740-9_2

Wanda, P. (2022). Runmax: Fake profile classification using novel nonlinear activation in CNN. *Social Network Analysis and Mining, 12*(1). doi:10.1007/s13278-022-00983-9

Wanda, P., & Jie, H. J. (2021). DeepFriend: Finding abnormal nodes in online social networks using Dynamic Deep Learning. Social Network Analysis and Mining, 11(1). doi:10.1007/s13278-021-00742-2

Wanda, P., & Jie, H. J. (2020). DeepProfile: Finding fake profile in online social network using dynamic CNN. Journal of Information Security and Applications, 52, 102465. doi:10.1016/j.jisa.2020.102465

Sahoo, S. R., & Gupta, B. (2021). Multiple features based approach for automatic fake news detection on social networks using Deep Learning. *Applied Soft Computing, 100*, 106983. doi:10.1016/j.asoc.2020.106983

Bindu, K., Rishith, B. P., Sathish, D., Subhash, V., Harika, B., & Swathi, N. (2022). Detection of fake accounts in Twitter using data science. *Int Res J Mod Eng Technol Sci, 4*(5), 3552-3556.

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., ... & Farhan, L. (2021). Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *Journal of big Data, 8*, 1-74.

McKinney, W. (2012). *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython.* " O'Reilly Media, Inc.".

Sial, A. H., Rashdi, S. Y. S., & Khan, A. H. (2021). Comparative analysis of data visualization libraries Matplotlib and Seaborn in Python. *International Journal, 10*(1).

Chollet, F. (2021). *Deep learning with Python*. Simon and Schuster.

Mishra, M. (2020, September 02). Convolutional Neural Networks, explained. Retrieved March 1, 2023, from https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

Gurucharan, M. (2022, October 03). Basic CNN architecture: Explaining 5 layers of Convolutional Neural Network. Retrieved March 1, 2023, from https://www.upgrad.com/blog/basic-cnn-architecture/

Nkomo, B. (2022, June 01). Convolutional Neural Networks-part 2: Padding and strided convolutions. Retrieved March 1, 2023, from https://medium.com/swlh/convolutional-neural-networks-part-2-padding-and-strided-convolutions-c63c25026eaa

Riva, W. (2022, November 11). Batch normalization in Convolutional Neural Networks. Retrieved March 1, 2023, from https://www.baeldung.com/cs/batch-normalization-cnn

Jafari, R. (2021, January 07). Instagram fake and real accounts dataset. Retrieved March 4, 2023, from https://www.kaggle.com/datasets/rezaunderfit/instagram-fake-and-real-accounts-dataset

Bakhshandeh, B. (2019, March 22). Instagram fake spammer genuine accounts. Retrieved March 4, 2023, from https://www.kaggle.com/datasets/free4ever1/instagram-fake-spammer-genuine-accounts

Biyani, Y. V. (2021). SPAM detection in social media using machine learning algorithm. *International Journal for Research in Applied Science and Engineering Technology, 9*(1), 432-439. doi:10.22214/ijraset.2021.32832

Jeong, J. (2019, July 17). The most intuitive and easiest guide for CNN. Retrieved March 6, 2023, from https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480

Verma, Y. (2023, January 14). A complete understanding of dense layers in neural networks. Retrieved March 6, 2023, from https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks/

Sharma, S. (2022, November 20). Activation functions in neural networks. Retrieved March 6, 2023, from https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Priya C, B. (n.d.). Softmax activation function: Everything you need to know. Retrieved March 6, 2023, from https://www.pinecone.io/learn/softmax-activation/

M, R. (2022, July 11). What is Numpy used for in python? Retrieved March 12, 2023, from https://www.activestate.com/resources/quick-reads/what-is-numpy-used-for-in-python/

Mode. (2016, May 23). Pandas: Python library - mode. Retrieved March 12, 2023, from https://mode.com/python-tutorial/libraries/pandas/

Kumar, B. (2022a, November 29). What is matplotlib and how to use it in Python. Retrieved March 12, 2023, from https://pythonguides.com/what-is-matplotlib/

Banerjee, R. (2021). How to use Seaborn for Data Visualization. Retrieved March 12, 2023, from https://www.section.io/engineering-education/seaborn-tutorial/

Patel, Y. (2022, August 19). Suppress warnings in Python: All you need to know. Retrieved March 12, 2023, from https://www.pythonpool.com/suppress-warnings-in-python/

Nawale, T. (2022, May 25). How to deal with missing values in machine learning. Retrieved March 12, 2023, from https://medium.com/geekculture/how-to-deal-with-missing-values-in-machine-learning-98e47f025b9c

Kumar, A. (2022b, April 16). Correlation Concepts, Matrix & Heatmap using Seaborn. Retrieved March 12, 2023, from https://vitalflux.com/correlation-heatmap-with-seaborn-pandas/

Upadhyay, A. (2020, July 23). Standardscaler and normalization with code and graph. Retrieved March 12, 2023, from https://medium.com/analytics-vidhya/standardscaler-and-normalization-with-code-and-graph-ba220025c054

Khanna, C. (2020, December 25). What and why behind fit_transform() vs transform() in scikit-learn ! Retrieved March 12, 2023, from https://towardsdatascience.com/what-and-why-behind-fit-transform-vs-transform-in-scikit-learn-78f915cf96fe

Marius, H. (2021, December 10). Multiclass classification with support vector machines (SVM), Kernel Trick & Kernel functions. Retrieved March 13, 2023, from https://towardsdatascience.com/multiclass-classification-with-support-vector-machines-svm-kernel-trick-kernel-functions-f9d5377d6f02

Malik, U. (2022, July 21). Implementing SVM and kernel SVM with Python's Scikit-Learn. Retrieved March 13, 2023, from https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/

R, S. (2023, February 15). Random Forest Algorithms - comprehensive guide with examples. Retrieved March 13, 2023, from https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

Loaiza, S. (2020, March 23). Gini impurity measure. Retrieved March 13, 2023, from https://towardsdatascience.com/gini-impurity-measure-dbd3878ead33

Koehrsen, W. (2018, August 31). An implementation and explanation of the Random Forest in python. Retrieved March 13, 2023, from https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76

Al-Serw, N. (2021, May 17). K-nearest neighbor: The maths behind it, how it works and an example. Retrieved March 14, 2023, from https://medium.com/analytics-vidhya/k-nearest-neighbor-the-maths-behind-it-how-it-works-and-an-example-f1de1208546c

Real Python. (2021, February 27). Python ai: How to build A neural network & make predictions. Retrieved March 14, 2023, from https://realpython.com/python-ai-neural-network/

Dertat, A. (2017, October 09). Applied deep learning - part 1: Artificial Neural Networks. Retrieved March 14, 2023, from https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

Team, K. (n.d.). Keras documentation: Layer weight initializers. Retrieved March 14, 2023, from https://keras.io/api/layers/initializers/#randomnormal

Satpathy, S. (2021, January 06). Smote: Overcoming class imbalance problem using smote. Retrieved March 14, 2023, from

https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/

Shu, K., Wang, S., & Liu, H. (2018). Understanding user profiles on social media for fake news detection. Retrieved March 30, 2023, from https://ieeexplore.ieee.org/document/8397048

Sheikhi, S. (2020). An efficient method for detection of fake accounts on the Instagram platform. *Revue D'Intelligence Artificielle, 34*(4), 429-436. doi:10.18280/ria.340407

Brownlee, J. (2019, August 06). A gentle introduction to dropout for Regularizing Deep Neural Networks. Retrieved April 20, 2023, from https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

Khan Academy. (n.d.). Scatterplots and Correlation Review (article). Retrieved April 27, 2023, from https://www.khanacademy.org/math/statistics-probability/describing-relationships-quantitative-data/introduction-to-scatterplots/a/scatterplots-and-correlation-review

Peccia, F. (2019, January 09). Batch normalization: Theory and how to use it with tensorflow. Retrieved April 29, 2023, from https://towardsdatascience.com/batch-normalization-theory-and-how-to-use-it-with-tensorflow-1892ca0173ad

الملخص

الناس في هذا العصر مدمنين على استخدام وسائل التواصل الاجتماعي، حيث أصبح كل شخص لا يستطيع قضاء يومه دون تصفح تطبيقات التواصل الاجتماعي. إن مشاركة الناس لحياتهم اليومية على هذه المنصات، أدت إلى سهولة الحصول على معلوماتهم الشخصية. مع ازياد هذه الظاهرة تتزايد الهجمات الالكترونية، حيث يعد إنشاء حسابات مزيفة للكشف عن معلومات المستخدمين للأغراض الضارة أمرًا سهلاً للغاية. بذل الباحثون جهودًا كبيرة لمحاولة بناء نموذج مثالي لاكتشاف الحسابات المزيفة بدقة جيدة، وهذا صعب للغاية بسبب السلوك غير المتوقع للمستخدم المزيف. يهدف هذا البحث إلى بناء نموذج يكتشف الحسابات المزيفة على منصة انستغرام بدقة عالية، مما سيساعد بشكل كبير في الحفاظ على عالمنا الافتراضي آمنًا. في هذا البحث، استخدمنا ثلاث مجموعات بيانات: تم العثور على مجموعة البيانات الأولى والثانية من مصدر مفتوح الوصول، وقمنا بتجميع مجموعة البيانات الثالثة باستخدام أدوات استخلاص البيانات من الويب مثل برنامج مصدر انستغرام من متصفح جوجل كروم وموقع أبيفي، مستخلص معلومات الويب. تم استخراج الميزات وتحليلها، وتم تطبيق الخوارزمية التصنيفية لنحصل في النهاية على خمس فئات تصنيف لمجموعة البيانات الثالثة. قسمنا عملنا إلى مرحلتين. في المرحلة الأولى، قمنا بتطبيق خوارزميات الشبكة العصبونية الاصطناعية، وخوارزمية الجار القريب من العدد ك، وخوارزمية شبكة المتجهات الداعمة، وخوارزمية الغابات العشوائية على مجموعات البيانات الثلاث التي تحتوي على تصنيف ثنائي للحقيقية والمزيفة. في المرحلة الثانية، طبقنا خوارزمية الشبكات العصبونية الالتفافية على مجموعة البيانات الثالثة، والتي تحتوي على خمس تصنيفات وهي الحقيقي، والمزيف الأول (حساب المشاهير المزيف)، والمزيف الثاني (حساب البريد العشوائي)، والمزيف الثالث (حساب المهاجم)، والمزيف الرابع (حساب بوت). لقد طبقنا تقنيات اختزال العينات والإفراط في أخذ العينات. تمت المناقشة المقارنة مع الاعمال السابقة التي قام بها باحثون آخرون.

جامعة البحرين
المكتبة
LIBRARY
UNIVERSITY OF BAHRAIN

جامعة البحرين

كلية الهندسة

قسم الهندسة الكهربائية والالكترونية

تحديد حسابات الملفات الشخصية المزيفة باستخدام خوارزمية الشبكة العصبية الالتفافية الديناميكية

أطروحة مقدمة كجزء من متطلبات الحصول على درجة الماجستير في برنامج أنظمة الذكاء الاصطناعي

اعداد

فاطمة عبدالجبار جواد

٢٠١٦٣٧٤٣

اشراف

بروفيسور إبراهيم مطر

جامعة البحرين

مملكة البحرين

مايو ٢٠٢٣