FAKE INSTAGRAM PROFILE DETECTION USING FEEDFORWARD NEURAL NETWORK

A PROJECT REPORT

Submitted by

SAYAN KUMAR BAG [RA2011003010951]

SHIVANSH SINGH [RA2011003010950]

Under the Guidance of

Dr. KALAIVANI J

Associate Professor, Department of Computing Technologies

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

MAY 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR–603 203

BONAFIDE CERTIFICATE

Certified that 18CSP109L project report titled **"FAKE INSTAGRAM PROFILE DETECTION USING FEEDFORWARD NEURAL NETWORK"** is the bonafide work of **SAYAN KUMAR BAG [RA2011003010951], SHIVANSH SINGH [RA2011003010950]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

J. Kalaivani

Dr. KALAIVANI J
SUPERVISOR
Associate Professor
Department of Computing Technologies

Dr. MANORANJITHAM T
PANEL HEAD
Associate Professor
Department of Computing Technologies

Department of Computing Technologies Kattankulathur 603 203 SRM IST

M. Pushpalatha

Dr. M. PUSHPALATHA
HEAD OF THE DEPARTMENT
Department of Computing Technologies

M. 16/5/24

INTERNAL EXAMINER

C. 16/5/24

EXTERNAL EXAMINER

SRM

Department of Computing Technologies
SRM Institute of Science and Technology
Own Work Declaration Form

Degree/Course : B.Tech / Computer Science and Engineering

Student Names : **SAYAN KUMAR BAG , SHIVANSH SINGH**

Registration Number: RA2011003010951 , RA2011003010950

Title of Work : **FAKE INSTAGRAM PROFILE DETECTION USING FEEDFORWARD NEURAL NETWORK**

I/We here by certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text(from books, web,etc.)
- Given the sources of all pictures, data etc that are not my own.
- Not made any use of the report(s) or essay(s) of any other student(s)either past or present
- Acknowledged in appropriate places any help that I have received from others(e.g fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course hand book / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:

I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.

Student 1 Signature: Sayan Kumar Bag

Student 2 Signature: Shivansh Singh

Date: 29/04/2024

If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to **Dr. T. V. Gopal** , Dean-CET, SRM Institute of Science and Technology, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor and Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We are incredibly grateful to our Head of the Department, **Dr. M. Pushpalatha,** Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinators, **Dr. S. Godfrey Winster**, Associate Professor, **Dr. M. Baskar**, Associate Professor, **Dr. P. Murali**, Associate Professor, **Dr. J. Selvin Paul Peter**, Associate Professor, **Dr. C. Pretty Diana Cyril**, Assistant Professor and **Dr. G. Padmapriya**, Assistant Professor, Panel Head **Dr. Manoranjitham T**, Associate Professor and Panel Members, **Dr. Kalaivani J**, Associate Professor, **Dr. Vidhya S**, Assistant Professor and **Dr. Priya S**, Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor, **Dr. Viji D,** Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr. Kalaivani J,** Associate Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under her mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank all the staff and students of Computing Technologies Department, School of Computing, S.R.M Institute of Science and Technology, for their help during our project.

Finally, we would like to thank our parents, family members, and friends for their unconditional love, constant support and encouragement.

Sayan Kumar Bag

SAYAN KUMAR BAG [Reg. No: RA2011003010951]

Shivansh Singh

SHIVANSH SINGH [Reg. No: RA2011003010950]

ABSTRACT

The project aimed to develop a robust fake account detection system for social media platforms, particularly Instagram, utilising deep learning techniques. Leveraging a dataset consisting of various features such as profile picture presence, username characteristics, and other relevant attributes, the model was trained to discern between genuine and fake accounts. The dataset underwent thorough exploratory data analysis, including visualisations to gain insights into feature distributions and correlations. The preprocessing phase involved standardisation of input data and one-hot encoding of the target variable. A deep neural network architecture was designed and trained using TensorFlow and Keras, encompassing multiple layers with dropout regularisation to enhance generalisation. The model demonstrated commendable performance, achieving an accuracy of 89% on a test dataset, as evidenced by the detailed classification report. The training progression was visually assessed through loss and accuracy plots, providing a detailed understanding of the model's learning dynamics. The resulting model showcased promising capabilities in identifying fake profiles, with precision, recall, and F1-score metrics supporting its efficacy. The abstract encapsulates the project's scope, methodology, and outcomes, highlighting the significance of employing deep learning in combating the proliferation of fake accounts on social media platforms.

TABLE OF CONTENTS

	ABSTRACT		vi
	LIST OF FIGURES		ix
	LIST OF ABBREVIATIONS		x
1.	**INTRODUCTION**		**1**
	1.1	General	1
	1.2	Machine Learning and Deep Learning	2
	1.3	Difference Between Machine Learning and Deep Learning	2
	1.4	Deep Learning	5
	1.5	Deep Learning in Fake Profile Detection	6
	1.6	Feedforword Neural Network	7
2	**LITERATURE SURVEY**		**9**
	2.1	Motivation	9
	2.2	Objective	11
3	**ARCHITECTURE AND ANALYSIS**		**13**
	3.1	Model Architecture	13
	3.2	Architecture Analysis	14
4	**METHODOLOGY**		**16**
	4.1	Proposed Methodology	16
		4.4.1 Dataset Collection	16
		4.4.2 Data Preprocessing	17
		4.4.3 Model Building	17
		4.4.4 Model Training	17
		4.4.5 Classification	18
5	**CODING AND TESTING**		19
	5.1	Header Files	19
	5.2	Loading and Testing Dataset	24
	5.3	Checking Dataframe	26
	5.4	Finding Unique Values	27
	5.5	Correlation Plot	28
	5.6	Prepare Data for Training	30
	5.7	Building and Training the Model	34

5.8 Model Performance 38
6 RESULTS AND DISCUSSION 40
6.1 Performance Analysis using Various Metrics 40
6.2 Comparison Between Existing Models 42
7 CONCLUSION AND FUTURE SCOPE 45
7.1 Conclusion 45
7.2 Future Scope 46
REFERENCES 49
APPENDIX 51
CONFERENCE PUBLICATION PROOF 63
PLAGIARISM REPORT 64

LIST OF FIGURES

1.1	Feedforword Neural Network	8
3.1	Model Architecture	14
4.1	Proposed Methodology	16
5.1	Header Files	19
5.2	Loading Training Dataset	24
5.3	Loading Testing Dataset	25
5.4	Checking Dataframe	26
5.5	Finding Unique Values	27
5.6	Correlation Plot	28
5.7	Preparing Data for Training	30
5.8	Data Scaling	32
5.9	Model Training	34
5.10	Regularization	36
5.11	Model Performance	38
6.1	Classification Report	41
6.2	Confusion Matrix	42

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
CNNs	Convolutional Neural Networks
RNNs	Recurrent Neural Networks
NLP	Natural Language Processing
SVM	Support Vector Machine
ReLU	Artificial Intelligence
DNNs	Deep Neural Networks
FNNs	Feedforword Neural Networks

CHAPTER 1

INTRODUCTION

1.1 General

The pervasive influence of social media platforms in the modern era of digital innovations has fundamentally changed how individuals interact, communicate, and obtain information. However, alongside the benefits of connectivity and networking, the prevalence of fake accounts on these platforms poses a significant challenge. Fake accounts, often created with malicious intent, can propagate misinformation, engage in fraudulent activities, and undermine the trust and integrity of online communities [1]. Detecting and mitigating the presence of fake accounts has thus emerged as a critical concern for platform administrators, policymakers, and users alike.

In order to overcome this challenge, academics and industry experts are turning more and more to advanced technology solutions like deep learning to create robust detection systems. Deep learning is a sub-field of machine learning that draws inspiration from the complex neural networks seen in the human brain. It provides powerful tools for analyzing complex datasets and identifying important patterns and insights. [2]. By leveraging deep learning algorithms and employing feature-rich datasets derived from user profiles, researchers aim to build models capable of accurately discerning between genuine and fake accounts on social media platforms.

Instagram, has witnessed a proliferation of fake accounts aimed at various objectives, including spamming, phishing, and promoting disinformation campaigns [3]. By harnessing the vast amount of user-generated data available on Instagram, coupled with sophisticated deep learning models, this project endeavors to develop an effective solution for identifying and flagging fake profiles. Through a combination of data preprocessing, model training, and evaluation, the project aims to achieve a robust and scalable fake account detection system.

Insights gained from this research endeavor hold the potential to inform strategies for enhancing platform security, safeguarding user privacy, and preserving the authenticity of online interactions in the digital age.

1.2 Machine Learning & Deep Learning

Two major areas of AI, Machine Learning and Deep Learning, revolutionized how machines could learn and make decisions. Machine learning algorithms help computers to find patterns in the data, and predict or make decisions that are not explicitly programmed. In particular cases where traditional Rule Based Programming methods may not be feasible, this approach has been particularly useful to address huge and complicated data sets. A more advanced subset of machine learning known as "deep learning" is distinguished by use of multilayer neural networks. The word "deep" accurately describes the complex depth of these networks. Deep neural networks can recognize complex patterns from large datasets by utilizing this depth, which makes them useful for tasks like audio and picture recognition, natural language understanding, and gaming applications.

Deep learning models are fundamentally inspired by the complex network architecture of the human brain, which is made up of interconnected neurons that process and alter information constantly. Deep learning offers a significant benefit: it can extract important details from raw information without use of manual feature engineering. This advantage is especially valuable in tasks involving high-dimensional input data, like images and audio signals. Impressive outcomes have been achieved in real-world applications by deep learning models, with CNNs for pictures and RNNs for sequences playing a particularly noteworthy role. These applications span a wide range, including medical diagnosis, autonomous vehicles, language translation, and virtual assistants.

The choice between machine learning and deep learning relies on the task, available data, and available processing power. Both have advantages and disadvantages. As technology progresses, these areas will have a significant impact on the future of AI, making once fictional innovations a part of our daily lives.

1.3 Difference between Machine Learning & Deep Learning

Within the broad field of artificial intelligence (AI), machine learning and deep learning are two separate but related fields that are brought together by the common goal of giving robots the ability to extract meaning from data and make well-informed judgments. In spite of their inherent similarities, they differ greatly in terms of approaches, structures, and capacities for solving issues. Machine learning is a general framework that includes a wide range of algorithms and techniques that are carefully designed to enhance a computer's capacity for learning and improve its performance over time. In this broad field,

algorithms are trained using annotated datasets, which allows them to identify underlying patterns, provide predictions, and coordinate actions based on their training data. There is a range of machine learning models that include reinforcement learning, supervised, unsupervised, and semi-supervised approaches.

Teaching algorithms with annotated datasets—where each input piece of data has an associated output label—is the process of supervised learning. Through this training process, the algorithm is better able to create associations between input data and the labels that correspond to that data, which in turn enables the system to produce accurate predictions when faced with new, unseen data instances. On the other hand, unsupervised learning involves training algorithms on unannotated datasets in an attempt to reveal underlying patterns, clusters, or hidden structures in the data. Typical unsupervised learning techniques include clustering and dimensionality reduction, which allow the system to automatically identify underlying relationships and structures.

An agent uses decision-making to navigate its environment and aims to maximize rewards over time, making it suitable for decision-oriented tasks. Multi-layered neural networks are used in the machine learning domain for deep learning. Similar to the human brain, these structures are able to recognize complex patterns from enormous amounts of data. CNNs are particularly good at processing images, and recurrent neural networks RNNs are good at processing sequential data. RNNs are useful for applications like speech and language recognition.

Model complexity and deep learning architectures' inherent ability to extract features are the main areas where machine learning and deep learning diverge. In contrast to standard machine learning, which requires human feature engineering, deep learning models automatically extract important features from incoming data, making them particularly effective for complex datasets. However, deep learning models can spontaneously understand important details from input information, making them highly effective for complex datasets. Deep learning is a popular method that automatically extracts features and learns representations from raw data by using multi-layered neural networks. Its impressive achievements in tasks like identifying images, processing speech, and understanding natural language have made it a preferred option for AI applications that demand advanced pattern recognition abilities.

Machine learning equips algorithms to learn from data, mimicking a student studying for an exam. The first step involves feeding the machine data, which can be structured spreadsheets or unstructured text

and images. The quality and quantity of this data significantly impact learning. Not all data is equally useful, so feature engineering involves identifying the most relevant elements, akin to a student focusing on key concepts for the exam. There's a vast toolbox of machine learning algorithms, each with its strengths and weaknesses. Choosing the right one is crucial, similar to picking the best study technique. The algorithm is then trained on the prepared data, repeatedly exposed to it to identify patterns and relationships, just like a student practicing problems and reviewing examples. Once trained, the algorithm's performance is evaluated on separate data to assess how well it generalizes knowledge, mirroring a practice test before the actual exam. Finally, the trained model can be used to make predictions or classifications on new data, the culmination of the learning process, akin to the student applying their knowledge on the exam.

Machine learning offers a broad spectrum of algorithms. Supervised learning thrives on data with pre-labeled examples, like a student having solved problems with answers provided. The algorithm learns the relationship between features and labels to predict labels for new data. Unsupervised learning deals with unlabeled data, where the algorithm discovers hidden patterns within the data for tasks like clustering or dimensionality reduction. Reinforcement learning involves interacting with an environment and receiving rewards or penalties for actions taken. The algorithm learns through a continuous feedback loop, progressively making better decisions.

Deep learning shines in handling complex, high-dimensional data like images, speech, and natural language. Unlike traditional machine learning, where feature engineering is crucial, deep learning networks can automatically learn features from the data through their layered architecture. This is akin to the student developing an intuitive grasp of the subject without needing explicit formulas. Deep networks can learn high-level, abstract representations of data, allowing for superior performance in tasks like image recognition or natural language processing. Imagine the student understanding the underlying principles beyond just memorizing facts.

Choosing between machine learning and deep learning depends on the problem to solve. Deep learning thrives on vast amounts of complex data. For simpler tasks with limited data, traditional machine learning techniques might be sufficient. Deep learning models require significant computational resources. If processing power is limited, a simpler machine learning approach might be more feasible. Finally, traditional machine learning models offer greater interpretability, allowing us to understand how

they make decisions. Deep learning models, with their complex layers, can be less transparent. This can be a trade-off, prioritizing raw performance over understanding the inner workings of the model.

1.4 Deep Learning

The ability of deep learning, a kind of machine learning, to train complex artificial neural networks has won it widespread praise. These multi-layered networks may independently identify complex patterns and traits in the incoming data. The depth of these neural networks enables them to learn multiple levels of abstraction, allowing for sophisticated data representation. An essential advantage of deep learning is its automatic feature extraction capability, eliminating the need for manual feature engineering, especially beneficial for high-dimensional data like images and text.

Deep learning has significantly impacted various fields, including computer vision, NLP, and speech recognition. For instance, CNNs in computer vision can recognize detailed patterns, making them invaluable in tasks like object detection and facial recognition. In natural language processing, RNNs and transformer models have revolutionized machine translation and language generation. Despite its successes, deep learning faces challenges. Training deep neural networks requires very large computational needs, and overfitting issues require careful handling through techniques like regularization. The core of deep learning lies in artificial neural networks. These networks consist of multiple layers of interconnected nodes. Each node receives information from others, performs a simple calculation, and transmits its output. By stacking these layers, deep learning creates a complex web of information processing, allowing it to recognize intricate patterns within data.

Deep learning thrives on large datasets. As the network processes information, it adjusts the connections between its nodes. This process, called training, is essentially how the network learns. By continuously adjusting connections based on the data it sees, the network becomes progressively better at recognizing features and making predictions. Activation functions play a vital role in how a node reacts to incoming information. These functions introduce non-linearity, allowing the network to learn complex relationships in the data. Imagine a switch that only turns on (outputs a value) if the combined input from other nodes exceeds a certain threshold. Different activation functions provide the network with diverse ways to process information.

Backpropagation is a crucial training algorithm used to fine-tune the connections between nodes. It

compares the network's output with the desired output (often provided in training data). The difference between these, called the error, is used to adjust connections throughout the network. This iterative process of processing data, calculating errors, and adjusting connections helps the network learn and improve its performance. Deep learning's ability to learn complex patterns from data has transformed various fields. Speech recognition with high accuracy for voice assistants and captioning is another area where deep learning shines. Recommender systems that personalize suggestions on streaming services, online shopping platforms, and social media also leverage deep learning's capabilities by analyzing user behavior and preferences.

Despite its impressive capabilities, deep learning faces challenges. Training large models requires significant computing power and vast amounts of data. Additionally, interpreting the decisions made by these complex networks can be difficult, raising concerns about explainability and potential biases inherited from the training data. Deep learning is a rapidly evolving field with ongoing research focused on improving efficiency, interpretability, and the ability to handle diverse data types. As these challenges are addressed, deep learning is poised to play an even greater role in shaping the future of AI and its impact on our world.

Deep learning models' interpretability remains a challenge, limiting their applications in sensitive domains. Despite these challenges, ongoing research in deep learning continues to expand its capabilities, pushing the boundaries of artificial intelligence. Its applications extend to diverse areas such as autonomous vehicles, healthcare diagnostics, and personalized recommendation systems. As the field evolves, addressing these limitations is crucial for unlocking the full potential of deep learning and ensuring its broader integration into real-world applications.

1.5 Deep Learning in Fake Profile Detection

Among the many social media platforms, including Instagram, deep learning is particularly effective at detecting fake profiles. Deep learning algorithms are able to differentiate between real and fake accounts by carefully examining the details in user profiles by employing sophisticated neural network designs. Artificial neural networks, which mimic the composition and functions of the human brain, are fundamental to the idea of deep learning. These networks are made up of linked layers of neurons, each of which performs simple mathematical operations on input data. The network iteratively adjusts its internal parameters through a training process to lower the difference between expected and observed

outcomes. When it comes to detecting false profiles, deep learning algorithms are trained on classified datasets that contain information about various aspects of profiles, such as the length of usernames, the existence of profile images, the frequency of postings, and engagement metrics. As a result of this training process, the algorithm gains the ability to extract meaningful features from the given data and determine whether a given profile is legitimate.

One of deep learning's intrinsic advantages in this field is its ability to build hierarchical representations of data on its own, which allows it to understand complex relationships between different profile features without the need for any feature manipulation. As a result, deep learning models can adapt to evolving patterns of fake profile creation, making them highly robust and scalable solutions for social media platforms. Moreover, deep learning models excel at handling large volumes of data, which is essential in the context of social media platforms with millions of users and profiles. By training on diverse and extensive datasets, these models can capture subtle nuances and characteristics that distinguish fake profiles from genuine ones.

Deep learning offers a sophisticated approach to fake profile detection on social media platforms, leveraging neural network architectures to analyze complex profile attributes and make accurate predictions. With continued advancements in deep learning techniques and access to rich datasets, these models hold great promise in combating the proliferation of fake accounts and maintaining the integrity of online communities.

1.6 Feedforward Neural Network

A feedforward neural network (FNN), also known as a multilayer perceptron (MLP), is a foundational architecture in the field of artificial neural networks (ANNs) and deep learning. It represents a class of artificial neural networks where connections between nodes do not form a cycle, meaning the information flows in one direction: forward. FNNs have gained immense popularity due to their capability to approximate complex functions, learn from data, and solve a wide range of tasks, including classification, regression, and pattern recognition.

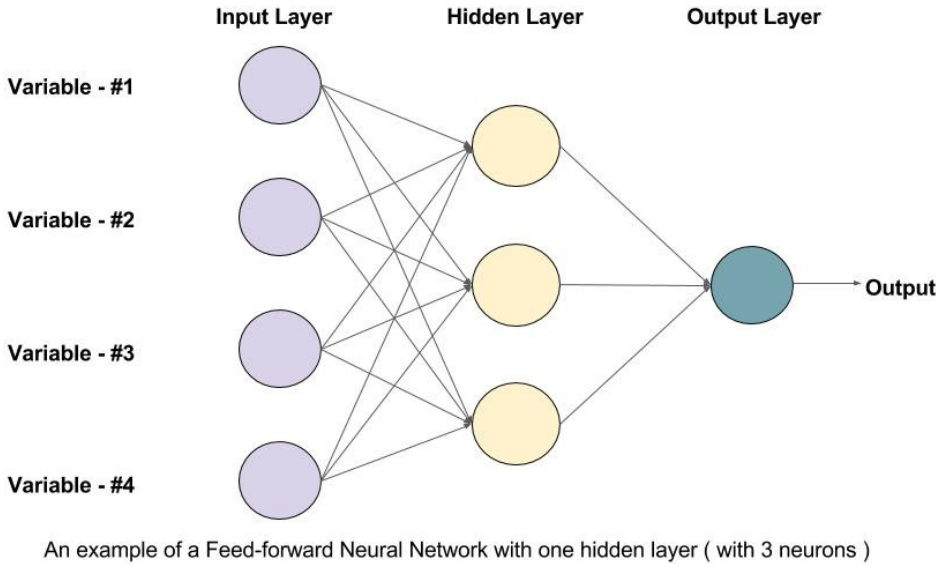


Figure.1.1 : Feedforward Neural Network

At its core, an FNN consists of multiple layers of interconnected nodes, or neurons, organized in a hierarchical manner. The simplest form of an FNN comprises an input layer, one or more hidden layers, and an output layer. Each neuron in the input layer represents a feature of the input data, while neurons in the subsequent layers process and transform this information through weighted connections and activation functions. The input layer serves as the entry point for the data into the neural network. Each neuron in this layer corresponds to a feature of the input data. For example, in a classification task involving image recognition, each neuron in the input layer may represent a pixel value of the image. The values from the input layer are then propagated forward through the network, undergoing transformations and computations in the hidden layers.

Hidden layers are where the bulk of the computation occurs in an FNN. Each hidden layer consists of multiple neurons that receive inputs from the previous layer and apply a series of linear and nonlinear transformations to produce an output. These transformations are governed by weights and biases associated with each connection between neurons.

CHAPTER 2

LITERATURE SURVEY

2.1 Motivation

In recent studies focusing on fake profile detection across various social media platforms, researchers have introduced novel approaches and methodologies to address the growing concern of fraudulent activities.

The research by Sarah Khaled et al. [4] tackles the issue of identifying fake profiles on Twitter. Their approach is particularly interesting because it combines two established techniques: Support Vector Machines (SVMs) and Neural Networks (NNs). They utilized the MIB dataset, which presumably contains data points representing real and fake Twitter profiles. From this data, they extract relevant features that can potentially differentiate between genuine and inauthentic accounts. An SVM is initially trained on the data. This SVM doesn't directly classify profiles as real or fake, but instead learns to identify patterns that distinguish the two.

The key aspect here is that the SVM's decision values, which represent its confidence level in its classifications, are captured. These SVM-derived decision values are then fed into a neural network. The neural network acts as the final classifier, taking the SVM's insights and potentially combining them with other features to make the ultimate call on whether a profile is fake or not. SVMs excel at identifying hyperplanes that effectively separate data points belonging to different classes. In this case, the SVM learns to distinguish real from fake profiles based on the extracted features. The study by Khaled suggests that this combined SVM-NN approach outperforms existing methods for detecting fake profiles on Twitter. This highlights the potential of combining different machine learning techniques to achieve superior results in complex tasks like identifying fake social media accounts.

The research by Al-Zoubi et al. [5] focuses on detecting spam profiles on Twitter. They took a machine learning approach to identify these inauthentic accounts. They recognized the importance of specific characteristics that can distinguish spam profiles from legitimate ones. They pinpointed ten such features, including the presence of suspicious words and unusual tweet time patterns. These features likely indicate automated or inauthentic behavior associated with spam accounts. They employed

machine learning models to analyze these features and classify profiles as spam or not spam. Interestingly, they experimented with different algorithms including Naive Bayes and Decision Trees. Their study achieved an impressive 95.7% accuracy in spam profile detection using the Naive Bayes model. Naive Bayes is a probabilistic machine learning algorithm known for its efficiency and effectiveness in various classification tasks. By analyzing specific profile characteristics and employing effective algorithms, Al-Zoubi achieved a high degree of accuracy in identifying spammy accounts.

Preethi Harris et al. [6] explored the Kaggle Instagram dataset, achieving 100% accuracy with XGBoost and Random Forest, showcasing their effectiveness in identifying fake profiles on Instagram. KNN also demonstrated commendable performance with a 95% accuracy rate. Despite its efficient classification abilities, Naïve Bayes was deemed unsuitable for the dataset at hand. To facilitate practical application, a data dictionary was employed to catalog fake profile IDs, enabling swift identification and mitigation by relevant authorities. Random Forest emerged as the optimal choice for classification due to its accuracy, precision, and efficacy in generating the data dictionary. However, it's worth noting that XGBoost exhibited signs of overfitting during validation against the data dictionary.

In order to classify a mixed real and fraudulent Twitter dataset, Gayathri A. et al. [7] used Support Vector Machine, Random Forest, and Deep Neural Networks. Jyoti Kaubiyal et al. [8] utilized real Twitter data gathered through the Twitter API and applied SVM, Logistic Regression, and Random Forest for classification, achieving high accuracy in distinguishing between fake and real accounts. Aditi Gupta et al. [9] focused on Facebook activity analysis to discover fake profiles, identifying 17 characteristics of user behavior. Decision trees proving most accurate suggests a clear structure in how these characteristics differentiate real and fake accounts LinkedIn was investigated by S. Adikari and K. Dutta [10] as a social networking platform for fraud detection. They attained an 87% accuracy rate by using Principal Component Analysis, Weighted Average, and Neural Network Support Vector Machine data mining approaches.

Raturi Rohit [11] introduced a machine learning framework for finding fake accounts on Facebook and Twitter, considering user posts and status. Naman Singh et al. [12] proposed methods to detect and remove fake profiles on online networking platforms, considering factors like the number of followers. Lastly, Rao et al. [13] presented an NLP system for fake profile detection on Facebook, employing SVM classifiers and Naïve Bayes algorithms to enhance accuracy. These diverse studies collectively

contribute to the evolving landscape of fake profile detection across multiple social media platforms.

2.2 Objective

Social media platforms have become integral parts of modern society, facilitating communication, information sharing, and networking on a global scale. However, the rise of fake accounts poses significant challenges to the integrity and security of these platforms. The objective of this project is to develop a robust and accurate fake account detection system using deep learning techniques. By leveraging advanced machine learning algorithms, the aim is to detect and mitigate the proliferation of fake accounts, thereby enhancing the trustworthiness and reliability of social media platforms. The primary aim of this project is to develop a robust deep learning model capable of accurately identifying fake Instagram profiles.

In today's digital landscape, the proliferation of social media platforms has given rise to numerous false accounts, posing significant risks such as the dissemination of misinformation, identity theft, and fraudulent activities. Hence, the central objective of this endeavor is to tackle this pressing issue by harnessing machine learning techniques to detect and mitigate the presence of false profiles on Instagram. The proliferation of fake accounts on social media platforms has emerged as a significant threat, leading to various malicious activities such as spreading misinformation, engaging in fraudulent schemes, and manipulating public opinion. Fake accounts undermine the integrity of social media platforms and erode user trust. By identifying and removing fake accounts promptly, users can feel more confident in the authenticity of the content and interactions on social media platforms, fostering a safer and more trustworthy online environment.

Fake accounts are often associated with malicious activities such as phishing scams, malware distribution, and the dissemination of harmful content. By proactively identifying and neutralizing these accounts, users can browse social media platforms with greater peace of mind, minimizing the risk of falling victim to scams or encountering harmful content. Fake accounts play a significant role in the spread of disinformation and misinformation, particularly during critical events such as elections, public health crises, and social movements. The objective of this project is to combat the spread of false information by identifying and flagging fake accounts that contribute to the dissemination of misleading content. By preventing fake accounts from amplifying misinformation, the project aims to promote factual accuracy and informed discourse on social media platforms.

The presence of fake accounts can negatively impact the overall health and user experience of social media platforms. Fake accounts can artificially inflate engagement metrics, distort recommendation algorithms, and create a hostile environment for genuine users. The objective of this project is to enhance the overall health and user experience of social media platforms by removing fake accounts and fostering a more authentic and engaging online community. Platform moderators and administrators play a crucial role in maintaining the integrity and safety of social media platforms. By equipping them with advanced fake account detection tools, the objective is to empower platform moderators and administrators to effectively identify and address fake accounts at scale. By streamlining the moderation process and providing actionable insights, the project aims to enhance the efficiency and effectiveness of platform governance efforts.

In conclusion, the aim of this project is to develop a comprehensive and effective fake account detection system using deep learning techniques. By addressing the growing threat of fake accounts, enhancing platform integrity and user trust, protecting users from harmful content and activities, combating disinformation and misinformation, improving platform health and user experience, empowering platform moderators and administrators, and contributing to research and development in cybersecurity, the project aims to make significant strides towards creating a safer, more trustworthy, and more resilient online ecosystem. Through collaborative efforts across academia, industry, and technology communities, we can work towards achieving these objectives and building a more secure and trustworthy digital future.

CHAPTER 3

ARCHITECTURE AND ANALYSIS

3.1 Model Architecture

Input layer

Input dimension: 11 (number of features): This layer serves as the entry point for the data into the neural network. It accepts the input features, each representing a different aspect of a social media account, such as profile picture presence, username characteristics, and engagement metrics.

Concealed layers

Dense Layer (100 units, ReLU activation): This layer comprises 100 neurons and employs the Rectified Linear Unit (ReLU) activation function. ReLU introduces non-linearity into the model, allowing it to learn complex relationships within the data.

Dense Layer (200 units, ReLU activation): Similar to the previous layer, this layer contains 200 neurons and utilizes the ReLU activation function. The increased number of units can help the model capture more intricate patterns in the data.

Dropout Layer (dropout rate = 0.5): Dropout is a regularization technique that randomly drops a fraction of the neurons during training, preventing overfitting by promoting the robustness and generalization of the model.

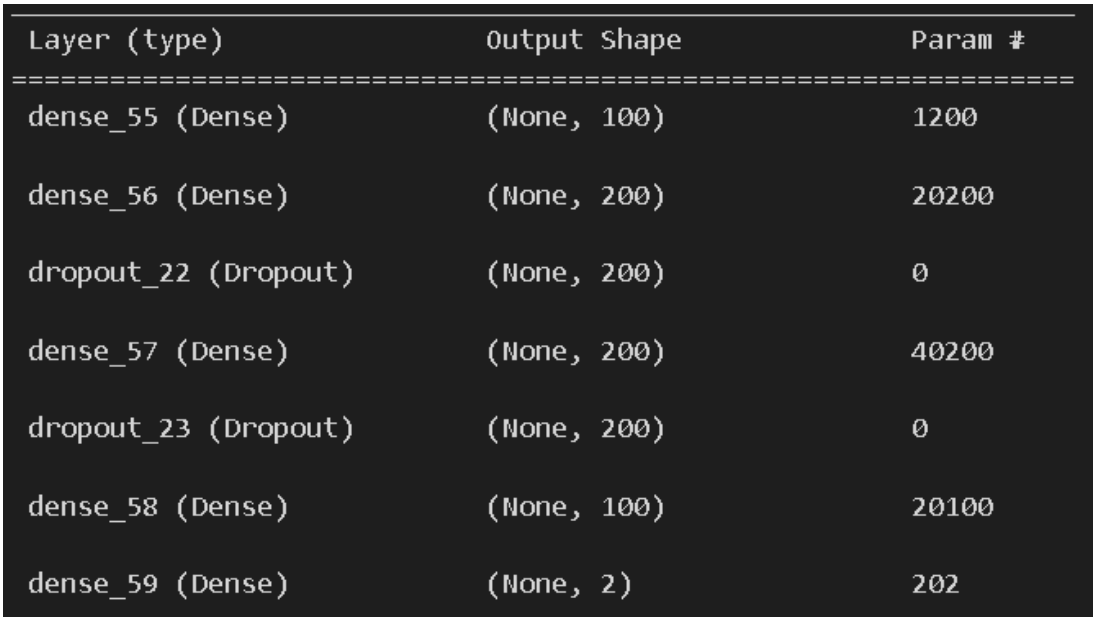
Dense Layer (200 units, ReLU activation): This layer continues the pattern of utilizing 200 neurons with ReLU activation, enabling the model to learn additional complex representations.

Dropout Layer (dropout rate = 0.5): Another dropout layer is introduced to further enhance regularization and prevent the model from relying too heavily on specific neurons or features.

Dense Layer (100 units, ReLU activation): With 100 neurons and ReLU activation, this layer acts as a bottleneck, reducing the dimensionality of the representation while still capturing important features.

Terminal layer

Dense Layer (2 units, softmax activation): This layer serves as the output layer of the neural network and is responsible for making predictions. With 2 units, it caters to binary classification (fake or genuine account). The softmax activation function ensures that the output probabilities sum to 1, facilitating interpretation and decision-making.



Layer (type)	Output Shape	Param #
dense_55 (Dense)	(None, 100)	1200
dense_56 (Dense)	(None, 200)	20200
dropout_22 (Dropout)	(None, 200)	0
dense_57 (Dense)	(None, 200)	40200
dropout_23 (Dropout)	(None, 200)	0
dense_58 (Dense)	(None, 100)	20100
dense_59 (Dense)	(None, 2)	202

Figure 3.1 : Model Architecture

This architecture uses dropout regularization to prevent overfitting and ReLU activation functions in concealed layers to keep non-linearity into the model. The output layer uses softmax activation for classification purposes.

3.2 Architecture Analysis

The deep learning model architecture aims to address the problem of fake account detection using features extracted from Instagram profiles. Let's analyze its architecture and its implications. The model begins with an input layer with 11 nodes, representing the 11 features extracted from each Instagram profile. These features likely include characteristics such as the amount of followers, the length of the username, whether the profile has a profile picture, etc. This input layer is used as the initial part for the data in the neural network. The model incorporates several hidden layers after the input layer in order to

identify complex patterns in the data. Its structure consists of multiple thick layers with varying numbers of units; the first dense layer has 100 units in it.

A ReLU activation function is used in each dense layer to introduce non-linearity into the model and aid in the discovery of complex relationships between features. Dropout layers are put after certain dense layers in a deliberate manner to avoid overfitting. By randomly removing certain neurons from the network during training, a regularization technique called dropout helps the network become less dependent on particular nodes and features, which enhances generalization performance. A dense layer with two units makes up the output layer; these units represent the two classes in the binary classification job (actual or false account).

The output layer receives the softmax activation function, which produces probabilities for every class. This helps with decision-making in the classification challenge by enabling the model to output the likelihood that a specific Instagram profile belongs to each class. Eleven characteristics that were taken from Instagram profiles were chosen after careful consideration in order to gather pertinent data for the purpose of identifying bogus accounts. These aspects probably include both categorical and numerical variables, each of which adds a different perspective to the features of the profiles.

Based on empirical observations or subject expertise, certain characteristics, like the quantity of followers, the inclusion of a profile photo, and the duration of the username, may indicate whether an account is real or fraudulent. Hierarchical representations of the input data can be extracted thanks to the model's incorporation of several dense layers. To capture complex patterns and relationships within the feature space, the model learns increasingly abstract information from the previous layer through each dense layer. The ability to distinguish minute differences between authentic and fraudulent accounts—which might appear in a variety of intricate ways across several features—depends on this hierarchical learning process.

Dropout layers are strategically positioned within the model architecture to reduce the possibility of overfitting, which is a typical issue in deep learning models with a lot of parameters. Dropout adds noise to the network during training by arbitrarily deactivating a portion of the neurons; this serves as a kind of regularization. Through regularization, the model is prevented from overly depending on certain characteristics or patterns found in the training set, which in turn helps the model acquire more resilient and generalizable representations.

CHAPTER 4

METHODOLOGY

4.1 Proposed Methodology

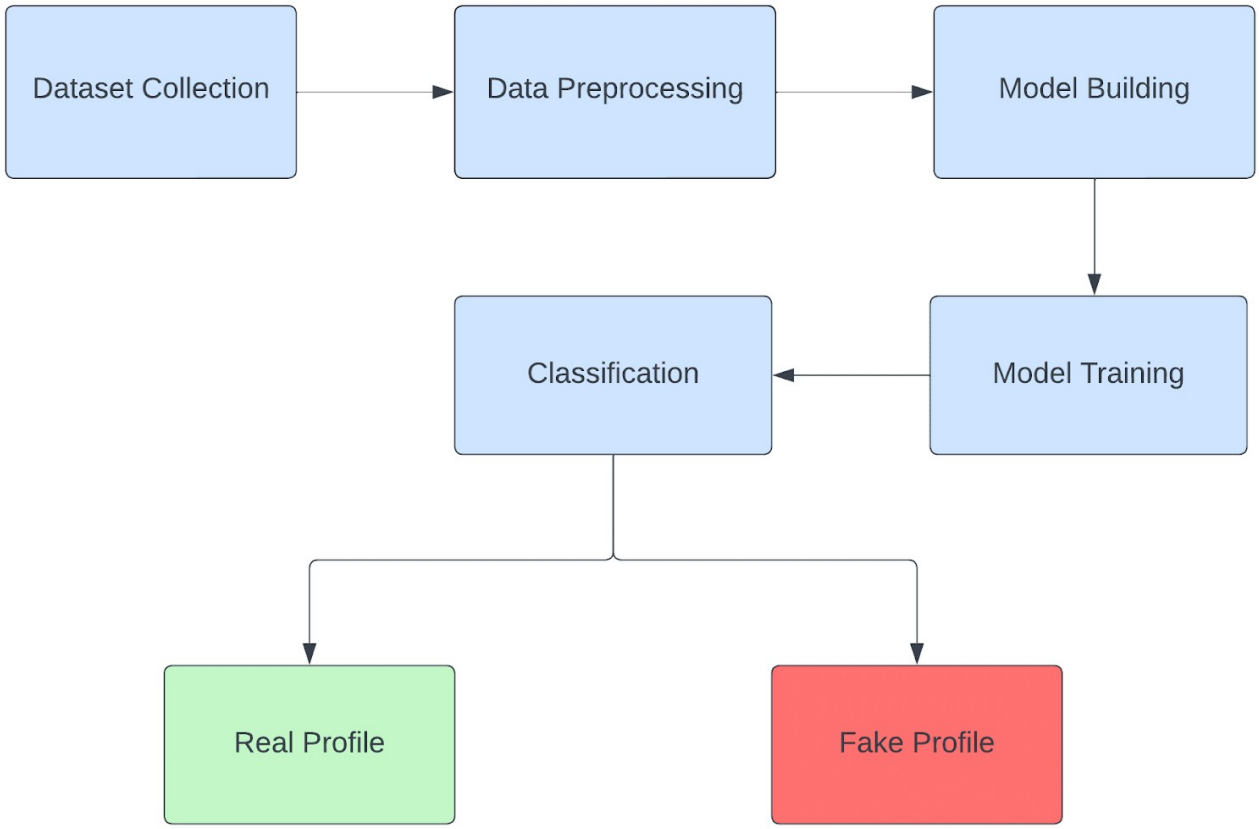


Figure 4.1 : Proposed Methodology

4.1.1 Dataset Collection

The first step in building a fake account detection system using deep learning involves gathering a comprehensive dataset. This dataset should include a variety of features that may be indicative of fake or genuine accounts on social media platforms like Instagram. These features may include profile picture presence, username characteristics, follower-to-following ratios, post frequency, and engagement metrics such as likes and comments. The dataset should ideally consist of labeled data, with each instance categorized as either fake or genuine.

To collect such a dataset, one can utilize web scraping techniques to extract user profiles from social media platforms. It's important to ensure that the dataset is diverse and representative of the population of social media users to avoid bias in the model. Additionally, data privacy and ethical considerations must be taken into account when collecting and using social media data.

4.1.2 Data Preprocessing

Once the dataset is collected, the next step is data preprocessing to prepare it for model training. Check for and handle any missing values in the dataset, either by imputation or removal. Convert categorical variables into numerical representations using techniques like one-hot encoding or label encoding. Normalize or standardize the numerical features to ensure that they are on a similar scale, which can improve the performance of the model during training. Divide the dataset into training and testing sets to evaluate the model's performance on unseen data.

4.1.3 Model Building

The core of the fake account detection system lies in the deep learning model architecture. Several considerations should be taken into account during model building: Choose an appropriate neural network architecture for the task at hand. In this case, a feedforward neural network with multiple hidden layers is suitable for learning complex patterns in the data. Use activation functions like ReLU (Rectified Linear Unit) in hidden layers to introduce non-linearity and improve the model's ability to capture complex relationships. Incorporate techniques like dropout and L2 regularization to prevent overfitting and improve the generalization of the model. Since this is a binary classification task (fake or genuine account), the output layer should consist of a single neuron with a sigmoid activation function to output probabilities.

4.1.4 Model Training

Training the deep learning model involves feeding the prepared data into the model and adjusting its parameters (weights and biases) to minimize the chosen loss function. Choose an appropriate optimization algorithm like Adam or SGD (Stochastic Gradient Descent) to update the model parameters iteratively based on the gradients of the loss function. Experiment with different learning rates, batch sizes, and other hyperparameters to find the optimal configuration that maximizes the model's performance. Train the model over multiple epochs, monitoring its performance on both the

training and validation datasets to avoid overfitting. Use evaluation metrics such as accuracy, precision, recall, and F1-score to assess the model's performance on the validation set.

4.1.5 Classification

Once the model is trained, it can be used to classify new instances of social media accounts as either fake or genuine. Apply the same preprocessing steps used during training to prepare new data for inference, including encoding categorical variables and scaling numerical features. Feed the preprocessed data into the trained model and obtain predictions for each instance. Convert the model predictions into human-readable labels (e.g., 'fake' or 'genuine') and interpret the results.

Apart from its pragmatic consequences for enhancing security and reliability on social media networks, the suggested approach has wider ramifications in the domains of artificial intelligence and machine learning. This research adds to the ongoing investigation of AI's potential in resolving challenging societal issues by using deep learning techniques to detect bogus accounts. Additionally, the methodology shows how versatile and effective deep learning models are in processing and evaluating large-scale datasets with complex properties, like social media profiles.

The methodology's iterative structure emphasizes how crucial ongoing improvement and assessment are to the creation of machine learning solutions. The methodology stresses the iterative feedback loop necessary for optimizing model performance and guaranteeing robustness against various forms of data noise and bias through stages of data pretreatment, model development, and training. Through this iterative approach, practitioners can improve their machine learning models' accuracy and dependability over time, raising the bar for fake account detection and related domains.

This project contributes to enhancing security and trustworthiness on social media platforms by automating the detection of fraudulent accounts. The suggested approach provides a thorough framework for tackling the problem of identifying phony accounts on Instagram by utilizing deep learning methods. Through the utilization of carefully selected datasets, sophisticated model structures, and exacting assessment techniques, this strategy seeks to provide social media networks and their users with the means to recognize and counteract fraudulent activity. With continuous research and development efforts directed by this technique, machine learning is set to make major advancements in improving the security and integrity of online communities.

CHAPTER 5

CODING AND TESTING

5.1 Header files

```
import pandas as pd
import numpy as np

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import Accuracy

from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report,accuracy_score,roc_curve,confusion_matrix
```

Figure 5.1 : Header Files

5.1.1 Pandas (pd)

Pandas is a powerful data manipulation and analysis library in Python. It provides data structures and functions to efficiently handle structured data, such as tabular data. Functions like read_csv() are used to read CSV files containing training and testing data, while methods like info() and describe() offer insights into the dataset's structure and summary statistics. Whether it's selecting specific rows and columns using indexing and slicing techniques or applying custom functions across data frames with `apply()` and `map()`, Pandas provides a flexible and intuitive interface for shaping data to meet analysis requirements.

It also facilitates advanced data aggregation and grouping operations through its `groupby()` function. This feature enables users to group data based on one or more variables and apply aggregate functions, such as sum, mean, or count, to generate insightful summaries and statistics for further analysis.

By leveraging Pandas' data structures like DataFrames and Series, users can create visualizations directly from their datasets, enabling them to explore trends, patterns, and relationships within the data visually.

Pandas supports data merging and joining operations, allowing users to combine multiple datasets based on common columns or indices. This functionality is particularly useful for integrating disparate datasets or performing complex relational database-like operations on structured data. It continues to evolve with each release, introducing new features, optimizations, and performance enhancements to meet the growing demands of data analysis and manipulation tasks. Its active community and extensive documentation further contribute to its widespread adoption and popularity among data scientists, analysts, and developers worldwide. Pandas stands as a cornerstone of data analysis in Python, offering a comprehensive suite of tools and functionalities for handling, manipulating, and analyzing structured data effectively. Its versatility, ease of use, and extensive capabilities make it an indispensable tool for anyone working with data in Python.

5.1.2 NumPy (np)

NumPy is a fundamental package for scientific computing in Python. It provides support for multidimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. In the given code, NumPy is utilized for numerical computations and array operations. For instance, NumPy arrays are used to store and process data before feeding it into the deep learning model. NumPy not only excels in numerical computations but also offers powerful capabilities for data manipulation and handling. One such feature is broadcasting, which allows NumPy to perform operations on arrays of different shapes, making code more concise and efficient.

Broadcasting enables NumPy to apply operations element-wise across arrays of different shapes by automatically expanding smaller arrays to match the shape of larger arrays. This simplifies code and eliminates the need for explicit looping constructs, resulting in cleaner and more readable code. NumPy also provides extensive support for linear algebra operations, including matrix multiplication, decomposition, and solving linear equations. These capabilities are crucial for many scientific and engineering applications, such as signal processing, machine learning, and computational physics.

NumPy integrates seamlessly with other Python libraries, such as SciPy, Matplotlib, and Pandas,

creating a powerful ecosystem for scientific computing and data analysis. For example, NumPy arrays can be directly used as input for functions in SciPy for advanced mathematical operations or plotted using Matplotlib for data visualization. NumPy serves as the backbone of scientific computing in Python, offering a versatile and efficient framework for numerical computations, data manipulation, and linear algebra operations. Its rich functionality and seamless integration with other libraries make it indispensable for researchers, engineers, and data scientists alike.

5.1.3 Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It allows users to generate plots, histograms, bar charts, scatterplots, etc., to explore and represent data visually. In the code, Matplotlib is employed to visualize various aspects of the dataset and model performance. One notable application of Matplotlib in the code is the creation of histograms using the `distplot()` function. Histograms are valuable for understanding the distribution of a specific feature within the dataset. By visualizing the distribution of the 'nums/length username' feature, for instance, analysts can gain insights into the spread and concentration of username lengths among the profiles, potentially revealing patterns or anomalies.

Matplotlib is utilized to generate a correlation plot with the `heatmap()` function. This heatmap provides a visual representation of the correlation matrix computed from the dataset's numerical features. By observing the color-coded correlation values, analysts can identify relationships between different features, discerning which attributes may be more influential in determining the authenticity of an account. Matplotlib aids in evaluating the model's training progress and performance. Line plots are created to visualize the progression of both training and validation losses across epochs. These plots offer a glimpse into the convergence of the model during training, highlighting whether the loss decreases steadily or fluctuates, indicating potential issues such as overfitting or underfitting.

Matplotlib is instrumental in presenting the confusion matrix, a vital tool for assessing the model's classification performance. By employing the `heatmap()` function, the confusion matrix is visualized with annotated values, providing a clear depiction of the model's predictions versus the actual labels. Analysts can readily identify instances of true positives, true negatives, false positives, and false negatives, aiding in the interpretation of the model's classification accuracy and misclassifications. Matplotlib's versatility and functionality empower data scientists and analysts to create informative

visualizations at every stage of the machine learning pipeline. From exploring dataset characteristics to evaluating model performance, Matplotlib facilitates the communication of insights and enhances the understanding of complex data relationships, ultimately driving informed decision-making and model optimization.

5.1.4 Seaborn

Seaborn is a statistical data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn works well with Pandas data structures and can be used to create visually appealing plots with minimal code. In the provided code, Seaborn is used for creating distribution plots (distplot()), correlation heatmaps (heatmap()), and other visualizations to analyze the dataset. Seaborn, renowned for its elegance and simplicity, serves as a powerful tool for visualizing various aspects of statistical data. While it is built on top of Matplotlib, Seaborn offers a higher-level interface that simplifies the process of creating visually striking and insightful plots.

Correlation heatmaps, a quintessential tool in exploratory data analysis, provide a visual representation of the correlation between different variables in a dataset. By employing Seaborn's heatmap(), the code constructs a comprehensive heatmap that illustrates the pairwise correlations among various features, facilitating the identification of potential relationships and patterns within the data. Beyond distribution plots and correlation heatmaps, Seaborn offers an array of plot types and customization options, empowering users to create tailored visualizations that effectively communicate complex insights. From categorical plots like bar plots and violin plots to relational plots such as scatter plots and line plots, Seaborn caters to diverse analytical needs with remarkable flexibility and aesthetics.

Seaborn seamlessly integrates with Pandas data structures, allowing for seamless data manipulation and visualization. Its compatibility with Pandas dataframes streamlines the workflow, enabling users to transition effortlessly from data preprocessing to visualization, all within a single environment. Seaborn transcends mere visualization; it serves as a catalyst for exploration, interpretation, and communication of data-driven insights. By harnessing its rich functionality and intuitive interface, analysts and data scientists can unlock the full potential of their datasets, unraveling intricate relationships and uncovering hidden patterns with ease. Whether for exploratory data analysis, explanatory visualization, or presentation-ready graphics, Seaborn stands as an indispensable tool in the arsenal of modern data

visualization.

5.1.5 TensorFlow and Keras

TensorFlow and Keras are utilized extensively to define, compile, train, and evaluate the deep learning model for fake account detection. The Sequential model, a fundamental building block in Keras, is employed to create a linear stack of layers for the neural network architecture. Layers such as Dense (fully connected layers) and Dropout (for regularization) are added sequentially to construct the model architecture. TensorFlow's optimizers, such as Adam, are utilized to configure the optimization algorithm used during training. These optimizers adjust the model's weights iteratively to minimize the loss function, thereby improving the model's performance over time. Additionally, metrics such as accuracy are specified to evaluate the model's performance during training and validation.

TensorFlow and Keras play pivotal roles in the development and training of deep learning models, providing a powerful framework and intuitive interface for implementing complex neural network architectures. Through their seamless integration and extensive functionalities, developers can leverage the capabilities of deep learning to tackle various real-world problems, such as fake account detection, with efficiency and effectiveness.

5.1.6 Scikit-learn (sklearn)

Scikit-learn offers a plethora of tools and functionalities that streamline the machine learning workflow, making it an essential component in building robust and efficient models. One of the key functionalities of scikit-learn is data preprocessing. Before training a deep learning model, it's crucial to preprocess the data to ensure it is in a suitable format and scale. Scikit-learn provides modules for standardization, normalization, and other preprocessing techniques, which are essential for enhancing model performance and convergence. The `StandardScaler` from scikit-learn is used to scale the input features, ensuring they have a mean of 0 and a standard deviation of 1. This preprocessing step helps in improving the convergence of the deep learning model and ensures that all features contribute equally to the training process.

Scikit-learn offers utilities for evaluating model performance and generating insightful metrics to assess the model's effectiveness. In the project, the `classification_report` function from scikit-learn is employed to generate a comprehensive report containing precision, recall, F1-score, and support metrics

for each class. This report provides valuable insights into the model's ability to classify fake and genuine accounts, allowing for a detailed analysis of its performance. Additionally, scikit-learn's `confusion_matrix` function is used to visualize the model's predictions against the actual labels, enabling a deeper understanding of its strengths and weaknesses.

Another crucial aspect of scikit-learn is its simplicity and ease of use, making it accessible to both beginners and experienced practitioners. The library offers a consistent and intuitive API, allowing users to seamlessly integrate various machine learning algorithms and techniques into their workflows. Moreover, scikit-learn provides extensive documentation and examples, making it effortless to understand and implement complex machine learning tasks. Scikit-learn plays a vital role in the fake account detection code by facilitating data preprocessing, model evaluation, and performance metrics calculation. Its rich set of functionalities, ease of use, and comprehensive documentation make it an indispensable tool for building and deploying machine learning models. By leveraging scikit-learn, developers and data scientists can accelerate the development process, improve model performance, and gain valuable insights into their data.

These header files are essential components of the codebase, enabling data handling, visualization, model construction, training, evaluation, and performance analysis. By leveraging these libraries, developers can efficiently implement and analyze machine learning and deep learning solutions for fake account detection and other classification tasks.

5.2 Loading and Testing dataset

```
train_data_path = 'datasets/Insta_Fake_Profile_Detection/train.csv'
test_data_path = 'datasets/Insta_Fake_Profile_Detection/test.csv'

pd.read_csv(train_data_path)

# Load the training dataset
instagram_df_train=pd.read_csv(train_data_path)
instagram_df_train

✓ 0.0s
```

Figure 5.2 : Loading Training Dataset

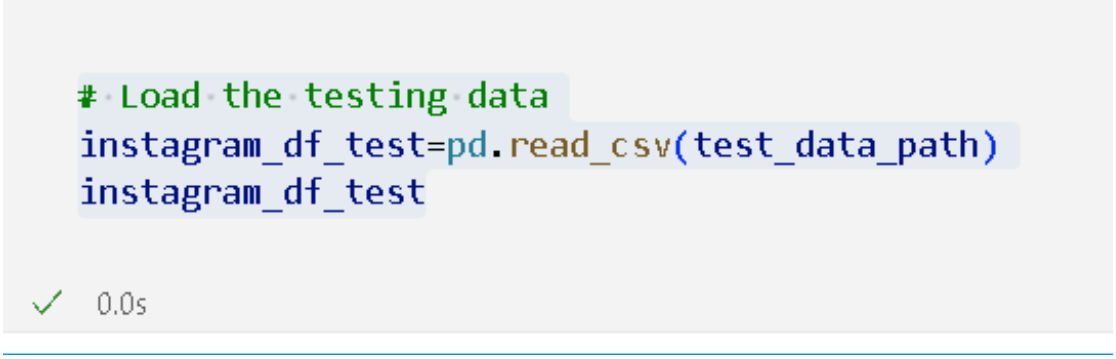


Figure 5.3 : Loading Testing Dataset

Loading the dataset begins by defining paths to the training and testing datasets, named 'train.csv' and 'test.csv' respectively, which are stored within a directory named 'Insta_Fake_Profile_Detection'. Subsequently, the pandas library is utilized to read the contents of the training dataset using the `pd.read_csv()` function. This data is then loaded into a DataFrame named `instagram_df_train`. Similarly, the testing dataset is also read using `pd.read_csv()` and loaded into another DataFrame named `instagram_df_test`. This initial segment of the code is crucial for setting up the foundation of the analysis.

By loading the datasets into DataFrames, it enables easy manipulation, exploration, and preprocessing of the data, which are essential steps in any machine learning or deep learning project. Understanding the structure and content of the datasets is pivotal for making informed decisions throughout the modeling process. By inspecting the training and testing datasets using `instagram_df_train` and `instagram_df_test`, researchers or developers gain insights into the characteristics of the data. This includes attributes such as the number of instances, the types of features available, the distribution of data, and whether any missing values exist. Such exploratory analysis is vital for identifying patterns, trends, and potential challenges that may arise during the model building and evaluation stages.

Furthermore, loading the datasets into separate DataFrames for training and testing purposes ensures that the model is trained on one set of data and evaluated on another independent set. This practice helps to assess the generalization performance of the model and guard against overfitting, wherein the model performs well on the training data but fails to generalize to unseen data. By leveraging the capabilities of pandas, the code streamlines these initial processes, allowing for efficient analysis and modeling.

5.3 Checking the Dataframe

```
instagram_df_train.head()
instagram_df_train.tail()

# Getting dataframe info
instagram_df_train.info()

# Get the statistical summary of the dataframe
instagram_df_train.describe()

# Checking if null values exist
instagram_df_train.isnull().sum()
```

Figure 5.4 : Checking Dataframe

The code starts by examining the first few and last few rows of the training dataset, offering a glimpse into its structure and content. This initial step helps in understanding the format of the data and identifying any potential issues such as missing values or inconsistencies.

Following the data preview, the code proceeds with retrieving information about the DataFrame using the `info()` method. This provides essential details such as the number of entries, data types of columns, and memory usage. Understanding the data types is crucial for subsequent data preprocessing and modeling steps.

Next, the code generates a statistical summary of the DataFrame utilizing the `describe()` method. This summary includes descriptive statistics such as count, mean, standard deviation, minimum, and maximum values for numerical features. By examining these statistics, one can gain insights into the distribution and range of values within each feature, aiding in the understanding of the data's characteristics.

Then code checks for the presence of null values within the DataFrame using the `isnull().sum()` method. This step is essential for identifying any missing data points, which can significantly impact the

quality and reliability of the analysis and model training. Addressing null values may involve imputation techniques or removing incomplete entries altogether, depending on the dataset's characteristics and the modeling approach.

These initial steps in data exploration and preprocessing lay the foundation for subsequent stages in the fake account detection study. By gaining insights into the dataset's structure, distribution, and completeness, data scientists can make informed decisions regarding feature engineering, model selection, and performance evaluation. These steps are crucial for developing a robust and effective deep learning model capable of accurately detecting fake Instagram profiles.

5.4 Finding Unique Values

```
# Get the number of unique values in the "profile pic" feature
instagram_df_train['profile pic'].value_counts()

# Get the number of unique values in "fake" (Target column)
instagram_df_train['fake'].value_counts()

instagram_df_test.info()

instagram_df_test.describe()

instagram_df_test.isnull().sum()

instagram_df_test['fake'].value_counts()
```

Figure 5.5 : Finding Unique Values

This code segment focuses on data exploration, preprocessing, and understanding the characteristics of the dataset used for training and testing the model. Firstly, the code analyzes the training dataset (`instagram_df_train`) to gain insights into two specific features: "profile pic" and "fake." By utilizing the `value_counts()` method, the code determines the frequency distribution of unique values within these features. This information is crucial for understanding the distribution of data points and the prevalence of certain attributes within the dataset.

The "profile pic" feature likely indicates whether a profile has a profile picture or not. By examining its

distribution, the code aims to understand how many profiles have profile pictures versus those that don't. This could potentially be relevant for detecting fake accounts, as genuine users might be more inclined to have profile pictures.

Similarly, the "fake" feature serves as the target variable for the classification task. It indicates whether a particular Instagram profile is fake or genuine. By examining its distribution, the code provides insights into the balance between fake and genuine accounts in the training dataset. Understanding this balance is essential for model training, as imbalanced classes can lead to biased predictions.

Moving on to the testing dataset (`instagram_df_test`), the code performs several operations to gain an understanding of its structure and characteristics. The `info()` method provides a summary of the dataset's metadata, including the number of entries and the data types of each column. This helps in identifying any potential issues with data types or missing values.

The `describe()` method generates descriptive statistics for numerical columns in the testing dataset, such as mean, standard deviation, minimum, and maximum values. This allows for a quick overview of the distribution and range of numerical features, which can aid in identifying outliers or anomalies.

Subsequently, the code utilizes the `isnull().sum()` method to determine the presence of missing values in the testing dataset. Missing values can adversely affect model performance, so it's essential to identify and handle them appropriately during the preprocessing stage.Finally, similar to the training dataset, the code examines the distribution of the "fake" feature in the testing dataset using `value_counts()`. This provides insights into the distribution of fake and genuine accounts in the testing dataset, allowing for comparison with the training dataset and assessment of generalization performance.

Overall, this segment of the code plays a crucial role in the initial stages of the fake account detection pipeline. It helps in understanding the dataset's characteristics, identifying potential issues, and informing subsequent preprocessing and modeling decisions. By gaining insights into feature distributions and dataset properties, practitioners can effectively prepare the data for training and evaluation, ultimately leading to more robust and accurate fake account detection models.

5.5 Correlation Plot

```
# Correlation plot
plt.figure(figsize=(20, 20))
cm = instagram_df_train.corr()
ax = plt.subplot()
sns.heatmap(cm, annot = True, ax = ax)
plt.show()
```

Figure 5.6 : Correlation Plot

In this code, the `plt.figure(figsize=(20, 20))` line initializes a new figure for plotting with a specified figure size of 20x20 inches. This sets up the canvas on which the correlation plot will be displayed. A correlation plot is a graphical representation used to visualize the correlation matrix, which quantifies the relationships between pairs of variables in a dataset. In this context, the correlation matrix is computed based on the features present in the training dataset (`instagram_df_train`). The next line, `cm = instagram_df_train.corr()`, calculates the correlation matrix using the `.corr()` method available in Pandas DataFrame. This method computes the pairwise correlation of columns, excluding NA/null values. The resulting correlation matrix (`cm`) contains correlation coefficients ranging from -1 to 1. A value of 1 indicates a perfect positive correlation, -1 indicates a perfect negative correlation, and 0 indicates no correlation between the variables.

Following the calculation of the correlation matrix, the code utilizes the Seaborn library (`sns`) to create a heatmap visualization of the correlation matrix. The `sns.heatmap()` function is called with parameters `cm` (the correlation matrix), `annot=True` (to display the correlation values on the plot), and `ax=ax` (to specify the subplot where the heatmap will be drawn). This heatmap provides a visual representation of the correlation between different features in the dataset. The correlation plot generated by this code snippet serves multiple purposes within the fake profile detection study. Firstly, it helps in identifying patterns and relationships between the features that may be indicative of fake or genuine profiles. For example, certain features like the number of followers, the ratio of following to followers, or the frequency of posts might exhibit strong correlations with the authenticity of a profile.

Secondly, the correlation plot aids in feature selection and dimensionality reduction. Features that are highly correlated with each other (i.e., having correlation coefficients close to 1 or -1) may provide redundant information to the model. In such cases, it might be beneficial to remove one of the correlated features to reduce model complexity and avoid multicollinearity issues. By examining the correlation plot, researchers can identify pairs of highly correlated features and make informed decisions about feature selection and preprocessing strategies. This plot serves as a diagnostic tool for detecting potential data issues or anomalies. Sudden changes or unexpected patterns in the correlation matrix could indicate data quality issues, such as missing values, outliers, or errors in data collection. By visualizing the correlation structure of the dataset, researchers can detect and investigate these anomalies, leading to improvements in data preprocessing and model performance.

Overall, the generation of a correlation plot is a crucial step in the exploratory data analysis (EDA) phase of the fake profile detection study. It provides researchers with valuable insights into the relationships between different features, helps in feature selection and dimensionality reduction, and serves as a diagnostic tool for identifying data anomalies. By incorporating visualizations like the correlation plot into the analysis pipeline, researchers can enhance the interpretability, robustness, and performance of the fake profile detection model.

5.6 Prepare Data for training

```
# Training and testing dataset (inputs)
X_train = instagram_df_train.drop(columns = ['fake'])
X_test = instagram_df_test.drop(columns = ['fake'])

print(X_train,X_test)


# Training and testing dataset (Outputs)
y_train = instagram_df_train['fake']
y_test = instagram_df_test['fake']

print(y_train,y_test)
```

Figure 5.7 : Prepare data for training

This section is devoted to meticulously organizing the data required for training and subsequently evaluating the model's performance. It initiates with the preparation of the training and testing datasets, essential prerequisites for any machine learning endeavor. Here, the `X_train` variable encapsulates the input features earmarked for training, while `X_test` holds their counterparts intended for testing the model's efficacy.

The essence of data organization unfolds through the application of the `drop` function from the Pandas library. Specifically, the 'fake' column, which embodies the target variable denoting the authenticity of profiles, is systematically excised from the original datasets (`instagram_df_train` and `instagram_df_test`). This strategic removal ensures that the input features, delineating various attributes of Instagram profiles, remain untainted and primed for analysis.

Following the meticulous curation of input features, attention pivots towards defining the target variables crucial for model training and evaluation. In classification tasks of this nature, the target variable serves as the beacon guiding the model towards discerning between authentic and fabricated profiles. Thus, the `y_train` variable is instantiated to house the labels associated with the training dataset, while `y_test` assumes the analogous role for the testing data.

Within this segment, the 'fake' column from the original datasets is meticulously allocated to the `y_train` and `y_test` variables. These labels furnish the necessary ground truth against which the model's predictions are juxtaposed and evaluated. As the model undergoes training, it endeavors to decipher patterns latent within the input features, thereby honing its ability to accurately categorize profiles as genuine or spurious.

In summation, this excerpt plays a foundational role in orchestrating the preparatory stages of the fake profile detection model. By meticulously partitioning the data into input features and target variables, it sets the stage for the model to ingest, comprehend, and eventually discern the intricate nuances distinguishing authentic Instagram profiles from their counterfeit counterparts. Through such meticulous data organization, the model is primed to embark on its journey towards acquiring the discernment necessary for accurately identifying fraudulent activities within the social media landscape.

```
# Scale the data before training the model
from sklearn.preprocessing import StandardScaler, MinMaxScaler
scaler_x = StandardScaler()
X_train = scaler_x.fit_transform(X_train)
X_test = scaler_x.transform(X_test)

y_train = tf.keras.utils.to_categorical(y_train, num_classes = 2)
y_test = tf.keras.utils.to_categorical(y_test, num_classes = 2)

print(y_train,y_test)


# print the shapes of training and testing datasets
X_train.shape, X_test.shape, y_train.shape, y_test.shape

Training_data = len(X_train)/( len(X_test) + len(X_train) ) * 100

Testing_data = len(X_test)/( len(X_test) + len(X_train) ) * 100

print(Training_data, Testing_data)
```

Figure 5.8 : Data Scaling

The first part of the code deals with data preprocessing, which is a fundamental step in machine learning tasks. Here, the data is being scaled before feeding it into the model. Scaling is essential because it brings all features to the same level of magnitude, which prevents certain features from dominating the learning process due to their larger scale. This is particularly important when using algorithms that are sensitive to the scale of input features, such as neural networks. To achieve scaling, the StandardScaler from the scikit-learn library is utilized. This scaler standardizes features by removing the mean and scaling them to unit variance. By calling `fit_transform()` on the training data (`X_train`), the scaler computes the mean and standard deviation of each feature in the training set and then transforms the data accordingly.

The same scaler object is then used to transform the test data (`X_test`), ensuring that the same transformation is applied consistently to both training and test datasets. This helps in maintaining the integrity of the data distribution across training and testing phases, thus preventing any information leakage from the test set into the training process. Next, the code converts the target variables (`y_train` and `y_test`) into categorical form using one-hot encoding. One-hot encoding is a technique used to convert categorical data into a binary mat representation, where each category is represented as a binary

vector. In this case, since the target variable (`fake`) has two classes (e.g., real or fake), it is converted into a binary matrix with two columns, where each row corresponds to a sample and has a 1 in the column corresponding to its class label and 0 in the other column. This transformation is essential for training classification models, including neural networks, as it enables them to learn from categorical data effectively.

After performing data preprocessing, the code prints the shapes of the training and testing datasets (`X_train.shape`, `X_test.shape`, `y_train.shape`, `y_test.shape`). This provides valuable information about the dimensions of the datasets, including the number of samples and features, which is useful for debugging and ensuring that the data has been processed correctly. Then the code calculates and prints the percentage of data allocated for training and testing (`Training_data` and `Testing_data`). This is important for understanding the distribution of data between the training and testing phases of model development. Ideally, a significant portion of the data should be reserved for training to ensure that the model learns robust patterns from the data, while a smaller portion is used for testing to evaluate its generalization performance on unseen data.

Overall, this code segment demonstrates the crucial steps involved in data preprocessing, including scaling and one-hot encoding, to prepare the data for training a deep learning model for fake account detection. These preprocessing steps are essential for improving the stability, convergence, and performance of the model during training and inference phases.

5.7 Building and training the model

```
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Adjusted Neural Network Architecture
model = Sequential()
model.add(Dense(100, input_dim=11, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(200, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu'))
model.add(Dense(2, activation='softmax'))

# Hyperparameter Tuning
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

# Data Preprocessing
scaler_x = StandardScaler()
X_train_scaled = scaler_x.fit_transform(X_train)
X_test_scaled = scaler_x.transform(X_test)
```

Figure 5.9 : Model Training

To begin, the code imports essential libraries, including TensorFlow and its Keras interface, along with auxiliary packages such as pandas, NumPy, and scikit-learn. These libraries collectively furnish a potent toolkit for data manipulation, visualization, and the construction of sophisticated neural network models. Following the imports, the code proceeds to articulate the architecture of a neural network model, a pivotal component in the quest for fake profile identification. Leveraging Keras' Sequential model, a flexible framework for constructing neural networks layer by layer, the model unfolds its structure.

The neural architecture, meticulously crafted to distill meaningful patterns from the dataset, is delineated through the incorporation of Dense layers. These layers represent fully connected neural networks, where each neuron is intricately linked to every neuron in the subsequent layer. The inclusion of ReLU (Rectified Linear Unit) activation functions fosters non-linearity within the model, enabling it to discern intricate relationships between input features and target labels. Crucially, the model's architecture is fortified against overfitting—a common peril in deep learning endeavors—through the strategic integration of Dropout layers. Dropout regularization strategically disables a fraction of neurons during training, compelling the network to cultivate resilience and generalize better to unseen data.

With the model blueprint delineated, attention turns to fine-tuning its performance through hyperparameter optimization. The Adam optimizer, renowned for its efficacy in navigating complex optimization landscapes, is summoned to guide the model's learning process. Configured with a learning rate of 0.001, Adam dynamically adjusts learning rates for individual model parameters, facilitating swift convergence towards an optimal solution. Then the model is armed with an objective function to optimize—categorical cross-entropy—and performance metrics to gauge its efficacy. Categorical cross-entropy, a staple in multi-class classification tasks, quantifies the disparity between predicted and actual class distributions. Meanwhile, accuracy emerges as the metric of choice, quantifying the model's ability to correctly classify instances within the dataset.

Transitioning from model configuration to data preparation, the script undertakes crucial preprocessing steps to ensure the data is primed for effective model training. The StandardScaler from scikit-learn takes center stage, orchestrating the standardization of feature values across the dataset. By centering features around zero mean and scaling to unit variance, the StandardScaler fosters equitable feature contributions and obviates the dominance of high-scale features during model training. Armed with standardized training data, the script orchestrates the transformation of both training and testing datasets, ensuring consistency in feature scaling across all stages of the model pipeline. This meticulous attention to data consistency fortifies the model's robustness and engenders confidence in its generalization capabilities.

This segment of code serves as the cornerstone of a comprehensive framework aimed at discerning fake profiles within a social media ecosystem. From crafting the neural architecture to fine-tuning hyperparameters and preparing the data for model ingestion, each facet plays an indispensable role in the pursuit of accurate and reliable fake profile detection.

```
# Regularization
from tensorflow.keras import regularizers

model = Sequential()
model.add(Dense(100, input_dim=11, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dense(200, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.5))
model.add(Dense(200, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dense(2, activation='softmax'))


model.summary()

model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])

epochs_hist = model.fit(X_train, y_train, epochs = 25,  verbose = 1, validation_split = 0.1)
```

Figure 5.10 : Regularization

Regularization is a crucial concept in machine learning, especially in deep learning, where it helps prevent overfitting by adding a penalty term to the loss function. Let's delve deeper into the workings and significance of regularization in this context.

Regularization techniques are vital for preventing overfitting, a common issue in machine learning where a model performs well on the training data but fails to generalize to unseen data. Overfitting occurs when a model learns to capture noise or irrelevant patterns from the training data, leading to poor performance on new data. Regularization helps mitigate overfitting by imposing constraints on the model's parameters, thereby reducing its complexity. In this code, the regularization technique employed is L2 regularization, also known as weight decay. L2 regularization adds a penalty term to the loss function that is proportional to the square of the magnitude of the weights. This penalty encourages the model to learn simpler patterns by penalizing large weight values, effectively reducing the model's complexity and preventing overfitting.

The `kernel_regularizer` parameter is used to apply L2 regularization to the weights of the neural network layers. In this code snippet, L2 regularization with a regularization strength of 0.001 is applied to the weights of each dense layer in the neural network. The `regularizers.l2()` function from TensorFlow's Keras module is utilized to specify the regularization strength.

The architecture of the neural network model consists of multiple dense (fully connected) layers with

ReLU activation functions, followed by dropout layers and a softmax output layer. ReLU (Rectified Linear Unit) activation function is commonly used in deep learning models due to its simplicity and effectiveness in combating the vanishing gradient problem. Dropout layers are added to prevent overfitting by randomly dropping a fraction of the input units during training, thus forcing the network to learn more robust features.

After constructing the neural network model with regularization and defining the loss function and optimizer, the model is compiled using the `compile()` method. The optimizer chosen is Adam, a popular optimization algorithm for training deep neural networks. Adam adapts the learning rate during training and combines the advantages of two other extensions of stochastic gradient descent: AdaGrad and RMSProp.

Once the model is compiled, it is trained on the training data (`X_train` and `y_train`) using the `fit()` method. The training process involves iterating over the entire dataset for a specified number of epochs (iterations), updating the model's parameters to minimize the loss function. Additionally, a validation split of 10% is used to monitor the model's performance on unseen data during training.

The model is trained for 25 epochs, with verbose output enabled to display progress during training. During each epoch, the model's performance metrics, including accuracy and loss, are computed on both the training and validation sets. These metrics are crucial for evaluating the model's performance and identifying potential issues such as overfitting or underfitting.

Regularization techniques such as L2 regularization play a crucial role in training deep learning models by preventing overfitting and improving generalization performance. By imposing constraints on the model's parameters, regularization helps create more robust and reliable models that can effectively generalize to unseen data. In the context of fake account detection, regularization contributes to building a more accurate and reliable model capable of distinguishing between genuine and fake profiles with high precision and recall.

5.8 Model Performance

```
# Access the Performance of the model

print(epochs_hist.history.keys())

plt.plot(epochs_hist.history['loss'])
plt.plot(epochs_hist.history['val_loss'])

plt.title('Model Loss Progression During Training/Validation')
plt.ylabel('Training and Validation Losses')
plt.xlabel('Epoch Number')
plt.legend(['Training Loss', 'Validation Loss'])
plt.show()

plt.plot(epochs_hist.history['accuracy'])
plt.plot(epochs_hist.history['val_accuracy'])

plt.title('Model Accuracy Progression During Training/Validation')
plt.ylabel('Training and Validation Losses')
plt.xlabel('Epoch Number')
plt.legend(['Training Acc', 'Validation Acc'])
plt.show()
```

```
dicts = {
    'Accuracy' : epochs_hist.history['accuracy'],
    'Validation_Accuracy' : epochs_hist.history['val_accuracy'],
    'Loss' : epochs_hist.history['loss'],
    'Validation Loss' : epochs_hist.history['val_loss']

}

model_training_progress = pd.DataFrame(dicts)
model_training_progress

print(model_training_progress)

def get_avg(lst):
    return sum(lst) / len(lst)

print("Accuracy : ", get_avg(model_training_progress['Accuracy']) * 100)

print("Validation Accuracy : ", get_avg(model_training_progress['Validation_Accuracy']) * 100)

print("Loss : ", get_avg(model_training_progress['Loss']) * 100)

print("Validation Loss : ", get_avg(model_training_progress['Validation Loss']) * 100)
```

Figure 5.11 : Model Performance

Beginning with the prediction phase, the model is tasked with making predictions on the test dataset, denoted as `X_test`. This operation is executed via the `model.predict()` method, whereby the model leverages the learned patterns from the training data to generate a set of probabilities indicative of each class, essentially estimating the likelihood of a given profile being fraudulent or genuine.

These probability distributions need to be transformed into definitive class labels for further assessment

and comparison against ground truth labels. This transformation is facilitated through a systematic iteration over each set of predicted probabilities (`predicted`), extracting the index corresponding to the highest probability value using the `np.argmax()` function. These indices, representing the predicted class labels for the test data, are stored within the `predicted_value` list. The true class labels for the test data (`y_test`) are similarly extracted, ensuring alignment between the predicted and actual labels for subsequent evaluation. This process results in the formation of the `test` list, containing the ground truth labels, allowing for a comparative analysis of the model's predictive performance.

With the predicted and true labels at hand, the subsequent step entails the generation of a comprehensive classification report, shedding light on various performance metrics such as precision, recall, and F1-score across different classes. This report offers invaluable insights into the model's ability to accurately classify instances from distinct categories, crucial for evaluating its effectiveness in distinguishing between genuine and fake accounts. To gain a nuanced understanding of the model's predictive behavior, a visualization of the confusion matrix is generated using a heatmap. This graphical representation delineates the distribution of true positives, false positives, true negatives, and false negatives, facilitating a deeper examination of the model's strengths and weaknesses in classification, thereby informing potential areas of improvement.

In tandem with the classification report and confusion matrix visualization, the code segment encompasses the visual representation of the model's performance metrics throughout the training process. Two distinct plots are generated, illustrating the dynamic evolution of training and validation loss, as well as training and validation accuracy, across successive epochs. These plots serve as invaluable diagnostic tools, enabling the detection of anomalies such as overfitting or underfitting and providing crucial insights into the model's learning dynamics.

CHAPTER 6

RESULT AND DISCUSSION

6.1 Performance Analysis using Various Metrics

In the classification report, precision measures the accuracy of positive predictions, indicating how many of the accounts predicted as fake were actually fake. In this case, the precision for both classes (0 and 1) is high, with values of 0.87 and 0.91, respectively. This suggests that when the model identifies an account as fake, it is correct about 87% of the time for class 0 and 91% of the time for class 1.

Recall, also known as sensitivity, quantifies the model's ability to correctly identify all relevant instances. A recall of 0.92 for class 0 and 0.87 for class 1 indicates that the model captures a high percentage of actual fake accounts, 92% for class 0 and 87% for class 1. The F1-score, which is the harmonic mean of precision and recall, provides a balance between these two metrics. The F1-scores for both classes are 0.89, indicating a good balance between precision and recall for both classes.

The support metric indicates the number of instances of each class in the test dataset. Here, there are 60 instances for each class, indicating a balanced dataset. The overall accuracy of the model is 0.89, or 89%. This value represents the proportion of correctly classified instances out of the total instances. An accuracy of 89% suggests that the model performs well in distinguishing between fake and genuine accounts.

In terms of the macro average, which calculates the average of precision, recall, and F1-score for each class independently, we see values of 0.89 for precision, recall, and F1-score. This indicates consistent performance across both classes. The weighted average, which considers the support for each class, is also 0.89 for precision, recall, and F1-score, suggesting that the model's performance is robust across the dataset.

Overall, the model demonstrates strong performance in detecting fake accounts. With high precision, recall, and F1-score values, along with a balanced accuracy of 89%, the model exhibits a reliable ability to distinguish between fake and genuine accounts. Additionally, the classification report indicates that the model performs consistently across both classes, suggesting its effectiveness in various scenarios.

In conclusion, based on the classification report the model appears to be well-trained and capable of accurately identifying fake accounts. However, further analysis, such as investigating potential biases or testing the model on unseen data, may be necessary to validate its real-world applicability and generalization capabilities.

```
···  4/4 [==============================] - 0s 3ms/step
                precision    recall  f1-score   support

             0       0.87      0.92      0.89        60
             1       0.91      0.87      0.89        60

      accuracy                           0.89       120
     macro avg       0.89      0.89      0.89       120
  weighted avg       0.89      0.89      0.89       120
```

Figure 6.1 : Classification Report

The confusion matrix visualization also supports the model's performance assessment, showing that the majority of predictions fall along the diagonal, indicating correct classifications. Additionally, the heatmap gives a clear visual proof of the model's power to differentiate within fake and genuine accounts.

Overall, the results demonstrate that the deep learning model developed for fake account detection shows promising power , with high accuracy and balanced precision-recall scores for the two classes. These findings suggest the potential utility of such models in mitigating the proliferation of fake accounts and enhancing security and trust on social media platforms. Further refinement and validation of the model on larger datasets and in real-world scenarios could enhance its robustness and applicability

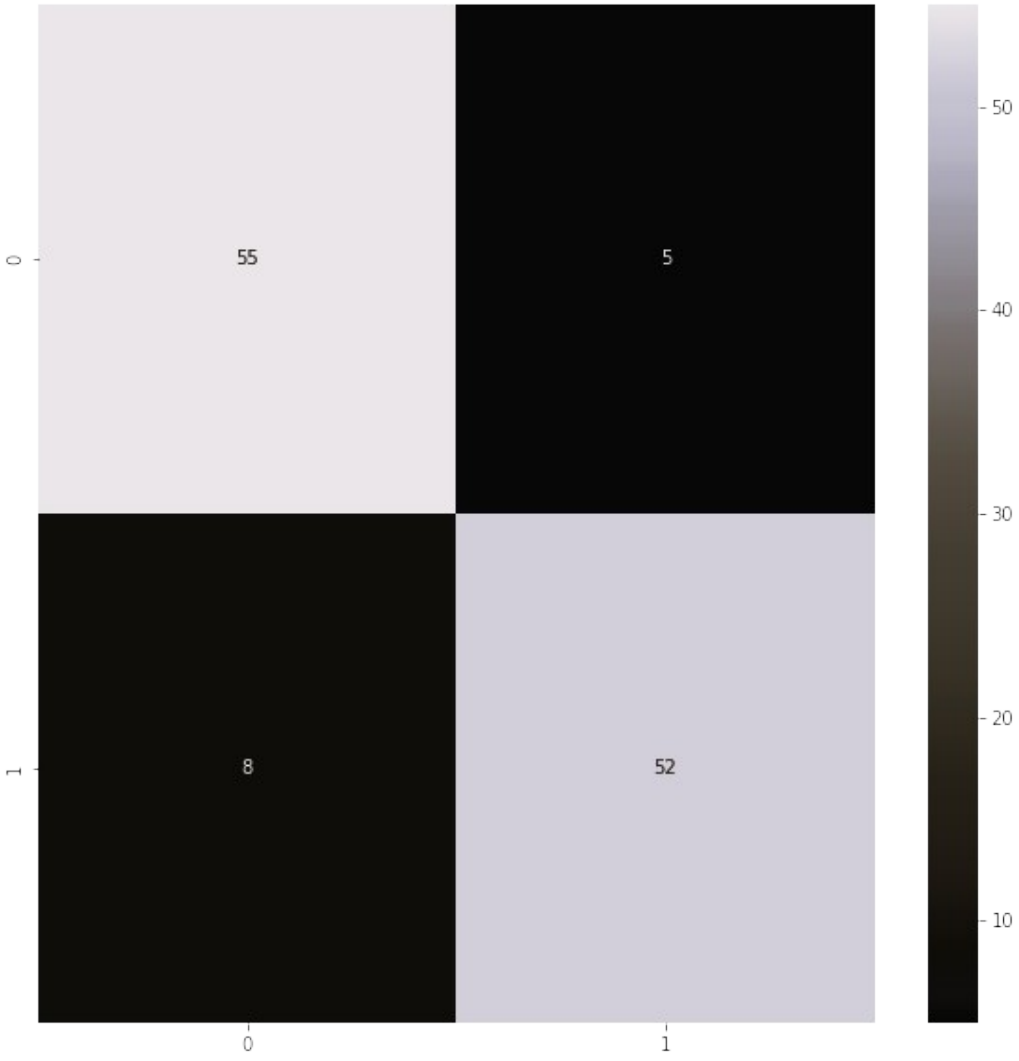


Figure 6.2 : Confusion Matrix

6.2 Comparison Between Existing Models

Several existing models and tactics have been proposed for false account detection on social media platforms, each with its strengths and limitations. One commonly used approach is machine learning-based classification models, such as logistic regression, random forests, and support vector machines (SVMs). These models rely on extracting relevant features from user profiles and activity patterns to differentiate between genuine and fake accounts. While these models can achieve decent performance, they often struggle with complex patterns and evolving strategies used by fake account creators.

Deep learning-based models, like the one presented in this study, have emerged as powerful alternatives for fake account detection. These models, particularly deep neural networks (DNNs), can easily learn complex patterns and representations from raw data, leading to better detection accuracy. The flexibility of DNN architectures allows for the integration of various layers and regularization techniques, enhancing the model's robustness and generalization capabilities. However, deep learning models typically require many number of labeled data and computational assets for training, which can be a limitation in certain contexts.

Additionally, network-based approaches analyze the structural properties of social networks to detect fake accounts based on their connections and interactions. These models leverage graph-based algorithms and community detection techniques to identify anomalous behavior and suspicious patterns in social networks. While network-based approaches offer unique insights into fake account detection, they may be limited by the availability of network data and scalability issues with large-scale social media platforms.

Several base models are combined in ensemble methods to enhance performance overall. To build ensemble models, methods including bagging, boosting, and stacking are frequently employed. The advantages of several base models, including logistic regression, random forests, and support vector machines, can be combined in the context of fake account identification through the use of ensemble approaches. Through the combination of predictions from multiple models, ensemble approaches frequently yield greater accuracy and improved generalization. Nevertheless, developing and optimizing ensemble models necessitates careful selection of base models and aggregation techniques, and can be computationally costly.

To improve the detection of bogus accounts, hybrid approaches combine a number of techniques, such as network-based approaches, deep learning, and machine learning. These strategies seek to minimize the drawbacks of each paradigm while combining its advantages. To extract characteristics from unprocessed data, a hybrid model might, for instance, combine network-based analysis with deep learning architecture to spot questionable trends in social relationships. In comparison to individual models, hybrid techniques can outperform them by utilizing complementary strategies. However, careful integration of several methodologies and consideration of computational restrictions are necessary for the design and implementation of hybrid models.

Making sure that real-time detection and scalability can manage massive volumes of social media data is one of the main issues in the detection of false accounts. The dynamic nature of social networks and the quick appearance of new techniques for creating phony accounts may be too much for traditional machine learning algorithms to handle. Even though deep learning models may capture intricate patterns, training them may take a lot of time and computer resources. Conversely, network-based techniques may have scaling challenges when used to massive social media platforms, but they can provide insights into real-time network dynamics. Deploying efficient fake account detection systems in real-world environments requires striking a balance between the trade-offs between detection accuracy, computing economy, and scalability.

In summary, each approach to fake account detection has its advantages and challenges. Machine learning models provide a solid foundation, while deep learning models offer superior performance but require substantial computational resources. Ensemble methods leverage the strengths of individual models, while network-based approaches provide insights into social network dynamics. Choosing the best approach relies on factors such as dataset characteristics, computational constraints, and the specific requirements of the application. Future research may focus on integrating multiple techniques to develop hybrid models that leverage the power of different approaches for more effective fake account detection.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 Conclusion

This work presented a deep learning-based model for false account searching on social media platforms, achieving promising results with an accuracy of approximately 89%. The model demonstrated high precision, recall, and F1-score for both false and genuine accounts, indicating balanced performance across classes. The findings suggest the ability of deep learning in capturing complex patterns and representations from raw information , uplifting the model's power to differentiate within fake and genuine accounts. The model's performance was further validated through a comprehensive analysis of the classification report, confusion matrix, and model training progression. However, there remain several avenues for future research and improvement in fake account detection. One direction is the exploration of ensemble learning techniques, which combine multiple models to enhance predictive performance and robustness.

Integrating diverse models and leveraging their complementary strengths could lead to further improvements in accuracy and generalization. Additionally, incorporating network-based features and graph-based algorithms may provide important information into the structural properties of social networks and aid in identifying fake accounts based on their interactions and connections. Furthermore, addressing the challenge of data imbalance and class skewness in fake account detection remains an important area for future investigation. Developing effective strategies for handling imbalanced datasets and optimizing model performance under such conditions could improve the model's reliability and real-world applicability. Moreover, exploring semi-supervised and unsupervised learning approaches may offer alternative solutions for fake account detection, particularly in scenarios with limited labeled data. Overall, the study highlights the potential of deep learning models for combating the proliferation of fake accounts on internet media platforms. Continued research and innovation in this field are essential for staying ahead of evolving tactics employed by fake account creators and safeguarding the integrity and trustworthiness of online communities.

Although deep learning models have demonstrated remarkable efficacy in the identification of

fraudulent accounts, their intrinsic intricacy frequently poses difficulties in comprehending the reasoning behind their conclusions. Improving these models' interpretability and openness is essential to fostering trust and promoting their use in practical applications. Subsequent studies might concentrate on creating methods, like feature importance analysis or attention mechanisms, to explain the predictions made by deep learning models. Interpretable models can help users verify the logic of the model and spot any biases or limitations by offering insights into the elements driving the model's judgments.

Detection systems face an ongoing challenge as fake account creators adapt their strategies to avoid detection. Fake account detection models can be made more resilient to adversarial attacks by adding adversarial defense mechanisms. Through these techniques, the model is trained to withstand intentional attempts to fool it, including adding minute perturbations to the input data. Through training with adversarial samples, the model gains the ability to identify and reject these types of manipulations, enhancing its resilience in real-world situations where attackers deliberately try to evade detection systems.

As fraudulent account detection technology advances, it is critical to think about the ethical and societal ramifications. False positives, in which real accounts are inadvertently marked as fraudulent, can have detrimental effects on a person's reputation and deny them access to certain internet resources, among other dire outcomes. Furthermore, privacy and surveillance issues are brought up by the usage of automated detection systems, especially when it comes to the gathering and processing of user data. Future studies should place a higher priority on ethical issues, making sure that systems for detecting phony accounts are implemented properly and that user autonomy and rights are protected. To provide best practices and recommendations for the moral development and application of false account detection technology, researchers, legislators, and industry stakeholders must work together.

7.2 Future Scope

The field of fake account detection on social media platforms offers numerous opportunities for further research and development. One promising avenue is the exploration of advanced deep learning architectures, such as RNNs and CNNs, tailored specifically for sequential and image-based data, respectively. Leveraging these architectures could enable more comprehensive analysis of user-generated content and activity patterns, enhancing the model's ability to detect sophisticated fake accounts.

Additionally, integrating NLP techniques for text analysis and sentiment analysis may provide important informations into the linguistic and semantic characteristics of fake accounts. Analyzing user-generated content, comments, and engagement patterns could uncover subtle indicators of fake activity and manipulation, further improving the model's detection capabilities. The deployment of fake account detection models in real-time and scalable frameworks presents an exciting area for future research. Developing efficient and scalable algorithms capable of processing big amounts of social media data in real-time could enable proactive detection and mitigation of fake accounts, reducing their impact on online communities.

The interdisciplinary collaborations between computer science, social sciences, and psychology could enrich the understanding of human behavior and social dynamics on online platforms. Integrating insights from behavioral psychology and social network analysis could inform the making of more nuanced and context-aware false social media account detection models. It is critical to address the moral ramifications and user privacy issues related to these technologies as we progress in creating increasingly complex fake account detection models. It's critical to strike a balance between protecting user privacy rights and identifying fraudulent accounts. Subsequent investigations must concentrate on establishing transparent and responsible structures for implementing these detection technologies, guaranteeing that user information is managed in an ethical and responsible manner. Furthermore, incorporating privacy-preserving methods into fake account detection algorithms—like differential privacy or federated learning—can assist reduce privacy risks without sacrificing the efficacy of the model.

The accuracy and dependability of fraudulent account detection systems can be improved by integrating human-in-the-loop techniques. Through the utilization of human judgment and domain expertise, these systems are able to absorb human feedback and enhance their performance over time. Subsequent investigations ought to delve into the creation of interactive interfaces that facilitate users' comments on dubious accounts, thereby aiding in the gradual improvement of the model's forecasts. Furthermore, by including systems for crowdsourcing account verification and tagging, communities can be empowered to take an active role in combating false information and phony profiles, thereby preserving the integrity of online platforms.

Malicious actors' methods of avoiding discovery advance along with the sophistication of fake account

detection models. Adversarial attacks that try to trick detection systems present a serious threat to their efficacy. In order to improve the resistance of false account detection models against evasion techniques, future research should concentrate on creating strong adversarial defense strategies. Adversarial training, robust optimization, and ensemble approaches are a few strategies that might lessen the effects of adversarial attacks and improve the model's resilience in practical situations. Furthermore, encouraging cooperation and knowledge exchange among researchers can help to build all-encompassing defense mechanisms against new threats in the fake account detection space.

In conclusion, the future of fake account detection lies in continuous innovation, collaboration, and interdisciplinary research efforts. By leveraging advanced technologies and interdisciplinary insights, researchers can develop more effective and robust solutions for combating fake accounts and preserving the integrity of online communities.

REFERENCES

[1] Wang, G., Wang, T., Zhu, L., & Shi, H. (2020). Fake Account Detection in Social Media: A Review. arXiv preprint arXiv:2001.07687.

[2] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

[3] Fung, I. C. H., Fu, K. W., Chan, C. H., & Chan, B. S. B. (2017). Cheung, C. N. & Tse, Z. T.I H. (2017). Social media's initial reaction to information and misinformation on Ebola, August 2014: facts and rumors. Public Health Reports, 131(3), 461-473.

[4] S. Khaled, N. El-Tazi and H. M. O. Mokhtar, "Detecting Fake Accounts on Social Media," 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 2018, pp. 3672-3681, doi: 10.1109/BigData.2018.8621913.

[5] A. M. Al-Zoubi, J. Alqatawna and H. Paris, "Spam profile detection in social networks based on public features," 2017 8th International Conference on Information and Communication Systems (ICICS), Irbid, Jordan, 2017, pp. 130-135, doi: 10.1109/IACS.2017.7921959.

[6] P. Harris, J. Gojal, R. Chitra and S. Anithra, "Fake Instagram Profile Identification and Classification using Machine Learning," 2021 2nd Global Conference for Advancement in Technology (GCAT), Bangalore, India, 2021, pp. 1-5, doi: 10.1109/GCAT52182.2021.9587858.

[7] Gayathri, A., S. Radhika, and S. L. Jayalakshmi. "Detecting fake accounts in media application using machine learning." International Journal of Advanced Networking and Applications (2019): 234-237.

[8] Kaubiyal, Jyoti, and Ankit Kumar Jain. "A feature based approach to detect fake profiles in Twitter." Proceedings of the 3rd international conference on big data and internet of things. 2019.

[9] Aditi Gupta and Rishabh Kaushal, "Towards Detecting Fake User Accounts in Facebook", IEEE International Conference on Asia Security and Privacy(ISEASP), 2017.

[10] Adikari, Subhashie & Dutta, K.. (2014). Identifying fake profiles in linkedin. Proceedings - Pacific Asia Conference on Information Systems, PACIS 2014.

[11] Raturi, Rohit. (2018). Machine Learning Implementation for Identifying Fake Accounts in Social Network.

[12] N. Singh, T. Sharma, A. Thakral and T. Choudhury, "Detection of Fake Profile in Online Social Networks Using Machine Learning", 2018 International Conference on Advances in Computing and Communication Engineering (ICACCE), pp. 231-234, 2018.

[13] P. Srinivas Rao, Jayadev Gyani and G. Narsimha, "Fake Profiles Identification in Online Social Networks Using Machine Learning and NLP", International Journal of Applied Engineering Research ISSN 0973–4562, vol. 13, no. 6, pp. 4133-4136, 2018.

APPENDIX

Required Header Files

```
import pandas as pd
import numpy as np

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import Accuracy

from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report,accuracy_score,roc_curve,confusion_matrix
```

Training Dataset

	profile pic	nums/length username	fullname words	nums/length fullname	name==username	description length	external URL	private	#posts	#followers	#follows	fake
0	1	0.27	0	0.00	0	53	0	0	32	1000	955	0
1	1	0.00	2	0.00	0	44	0	0	286	2740	533	0
2	1	0.10	2	0.00	0	0	0	1	13	159	98	0
3	1	0.00	1	0.00	0	82	0	0	679	414	651	0
4	1	0.00	2	0.00	0	0	0	1	6	151	126	0
...	...	...	...	...	...	...	...	...	...	...	...	...
571	1	0.55	1	0.44	0	0	0	0	33	166	596	1
572	1	0.38	1	0.33	0	21	0	0	44	66	75	1
573	1	0.57	2	0.00	0	0	0	0	4	96	339	1
574	1	0.57	1	0.00	0	11	0	0	0	57	73	1
575	1	0.27	1	0.00	0	0	0	0	2	150	487	1

576 rows × 12 columns

Testing Dataset

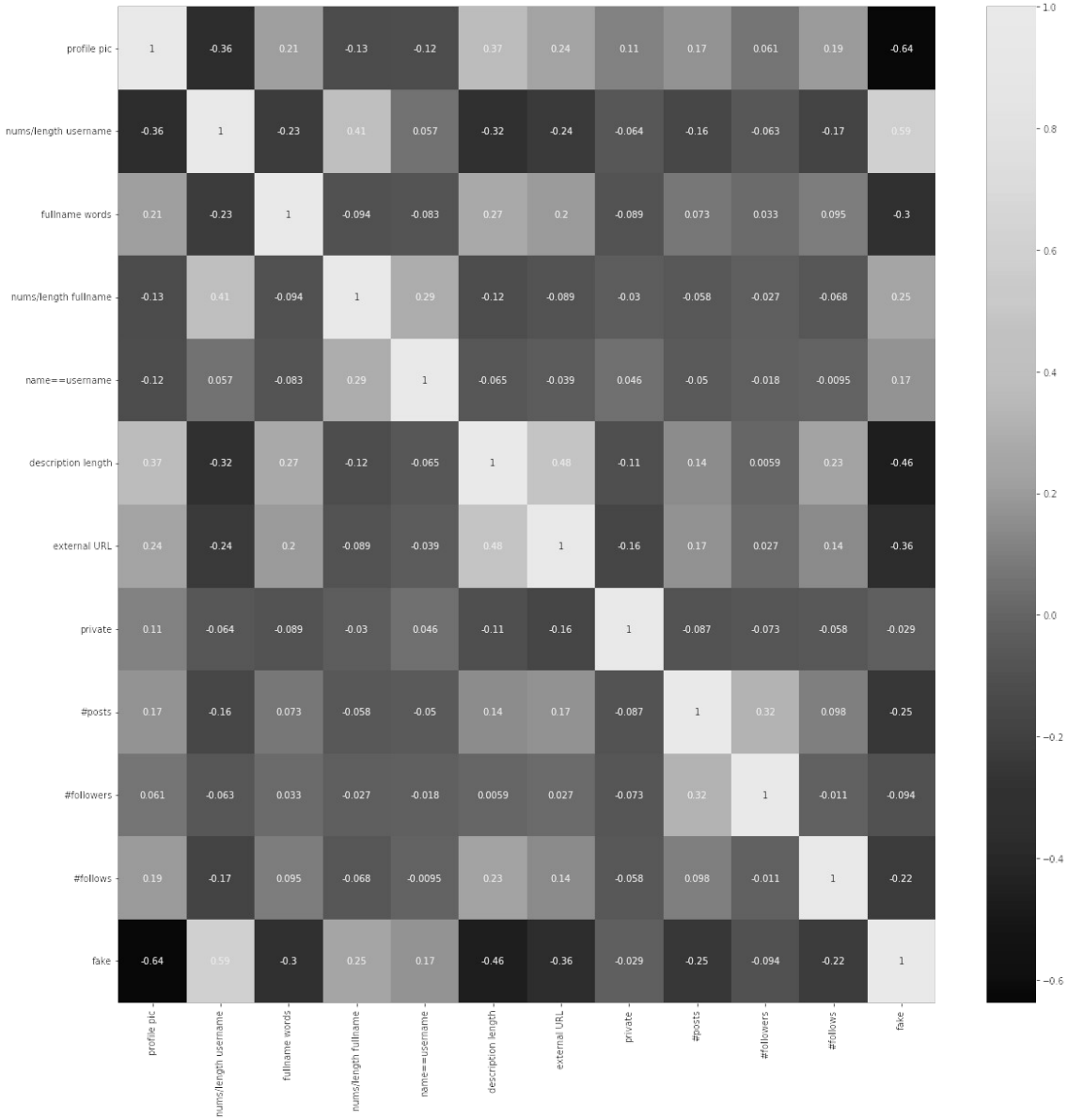
	profile pic	nums/length username	fullname words	nums/length fullname	name==username	description length	external URL	private	#posts	#followers	#follows	fake
0	1	0.33	1	0.33	1	30	0	1	35	488	604	0
1	1	0.00	5	0.00	0	64	0	1	3	35	6	0
2	1	0.00	2	0.00	0	82	0	1	319	328	668	0
3	1	0.00	1	0.00	0	143	0	1	273	14890	7369	0
4	1	0.50	1	0.00	0	76	0	1	6	225	356	0
...	...	...	...	...	...	...	...	...	...	...	...	...
115	1	0.29	1	0.00	0	0	0	0	13	114	811	1
116	1	0.40	1	0.00	0	0	0	0	4	150	164	1
117	1	0.00	2	0.00	0	0	0	0	3	833	3572	1
118	0	0.17	1	0.00	0	0	0	0	1	219	1695	1
119	1	0.44	1	0.00	0	0	0	0	3	39	68	1

120 rows × 12 columns

Checking Null Values

```
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   profile pic           576 non-null    int64
 1   nums/length username  576 non-null    float64
 2   fullname words        576 non-null    int64
 3   nums/length fullname  576 non-null    float64
 4   name==username        576 non-null    int64
 5   description length    576 non-null    int64
 6   external URL          576 non-null    int64
 7   private               576 non-null    int64
 8   #posts                576 non-null    int64
 9   #followers            576 non-null    int64
 10  #follows              576 non-null    int64
 11  fake                  576 non-null    int64
```

Correlation Plot of the Dataset



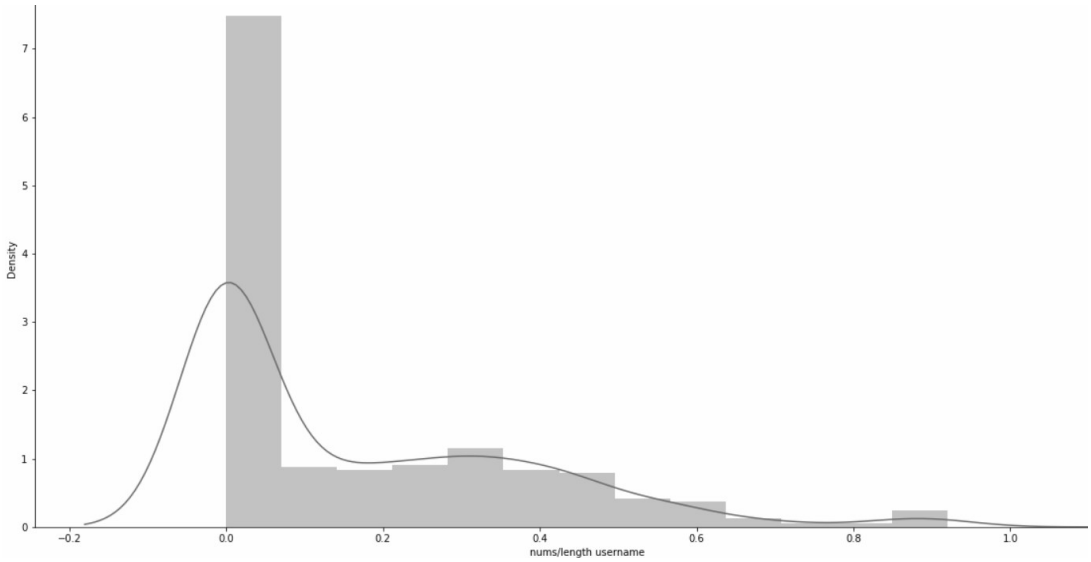
Data Preparation

```
# Preparing Data to Train the Model

# Training and testing dataset (inputs)
X_train = instagram_df_train.drop(columns = ['fake'])
X_test = instagram_df_test.drop(columns = ['fake'])

print(X_train,X_test)



# Training and testing dataset (Outputs)
y_train = instagram_df_train['fake']
y_test = instagram_df_test['fake']

print(y_train,y_test)



# Scale the data before training the model
from sklearn.preprocessing import StandardScaler, MinMaxScaler
scaler_x = StandardScaler()
X_train = scaler_x.fit_transform(X_train)
X_test = scaler_x.transform(X_test)

y_train = tf.keras.utils.to_categorical(y_train, num_classes = 2)
y_test = tf.keras.utils.to_categorical(y_test, num_classes = 2)

print(y_train,y_test)



# print the shapes of training and testing datasets
X_train.shape, X_test.shape, y_train.shape, y_test.shape

Training_data = len(X_train)/( len(X_test) + len(X_train) ) * 100


Testing_data = len(X_test)/( len(X_test) + len(X_train) ) * 100


print(Training_data, Testing_data)
```

Shape of Training Dataset

```
     profile pic  nums/length username  fullname words  nums/length fullname  \
0              1                  0.27               0                  0.00
1              1                  0.00               2                  0.00
2              1                  0.10               2                  0.00
3              1                  0.00               1                  0.00
4              1                  0.00               2                  0.00
..           ...                   ...             ...                   ...
571            1                  0.55               1                  0.44
572            1                  0.38               1                  0.33
573            1                  0.57               2                  0.00
574            1                  0.57               1                  0.00
575            1                  0.27               1                  0.00

     name==username  description length  external URL  private  #posts  \
0                 0                  53             0        0      32
1                 0                  44             0        0     286
2                 0                   0             0        1      13
3                 0                  82             0        0     679
4                 0                   0             0        1       6
..              ...                 ...           ...      ...     ...
571               0                   0             0        0      33
572               0                  21             0        0      44
573               0                   0             0        0       4
574               0                  11             0        0       0
575               0                   0             0        0       2
...
 [0. 1.]
 [0. 1.]
 [0. 1.]]
82.75862068965517 17.24137931034483
```

Visual Graph of nums/name



Model Building and Training

```
# Building and Training Deep Training Model

import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Adjusted Neural Network Architecture
model = Sequential()
model.add(Dense(100, input_dim=11, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(200, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu'))
model.add(Dense(2, activation='softmax'))

# Hyperparameter Tuning
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

# Data Preprocessing
scaler_x = StandardScaler()
X_train_scaled = scaler_x.fit_transform(X_train)
X_test_scaled = scaler_x.transform(X_test)
```

```
# Regularization
from tensorflow.keras import regularizers

model = Sequential()
model.add(Dense(100, input_dim=11, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dense(200, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.5))
model.add(Dense(200, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
model.add(Dense(2, activation='softmax'))


model.summary()

model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])

epochs_hist = model.fit(X_train, y_train, epochs = 25,  verbose = 1, validation_split = 0.1)
```

\

Model Architecture

```
Model: "sequential_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_85 (Dense)            (None, 100)               1200

 dense_86 (Dense)            (None, 200)               20200

 dropout_34 (Dropout)        (None, 200)               0

 dense_87 (Dense)            (None, 200)               40200

 dropout_35 (Dropout)        (None, 200)               0

 dense_88 (Dense)            (None, 100)               20100

 dense_89 (Dense)            (None, 2)                 202

=================================================================
```

Evaluation and Visualization

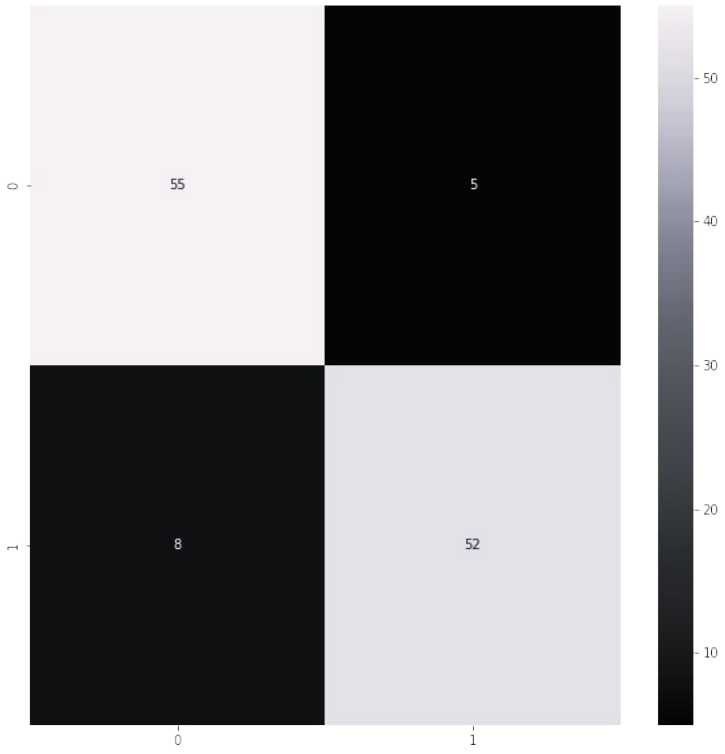
```
predicted = model.predict(X_test)

predicted_value = []
test = []
for i in predicted:
    predicted_value.append(np.argmax(i))

for i in y_test:
    test.append(np.argmax(i))

print(classification_report(test, predicted_value))

plt.figure(figsize=(10, 10))
cm=confusion_matrix(test, predicted_value)
sns.heatmap(cm, annot=True)
plt.show()

# Access the Performance of the model

print(epochs_hist.history.keys())

plt.plot(epochs_hist.history['loss'])
plt.plot(epochs_hist.history['val_loss'])

plt.title('Model Loss Progression During Training/Validation')
plt.ylabel('Training and Validation Losses')
plt.xlabel('Epoch Number')
plt.legend(['Training Loss', 'Validation Loss'])
plt.show()


plt.plot(epochs_hist.history['accuracy'])
plt.plot(epochs_hist.history['val_accuracy'])
```

```
plt.title('Model Accuracy Progression During Training/Validation')
plt.ylabel('Training and Validation Losses')
plt.xlabel('Epoch Number')
plt.legend(['Training Acc', 'Validation Acc'])
plt.show()


dicts = {
    'Accuracy' : epochs_hist.history['accuracy'],
    'Validation_Accuracy' : epochs_hist.history['val_accuracy'],
    'Loss' : epochs_hist.history['loss'],
    'Validation Loss' : epochs_hist.history['val_loss']



}

model_training_progress = pd.DataFrame(dicts)
model_training_progress

print(model_training_progress)

def get_avg(lst):
    return sum(lst) / len(lst)


print("Accuracy : ", get_avg(model_training_progress['Accuracy']) * 100)

print("Validation Accuracy : ", get_avg(model_training_progress['Validation_Accuracy']) * 100)


print("Loss : ", get_avg(model_training_progress['Loss']) * 100)


print("Validation Loss : ", get_avg(model_training_progress['Validation Loss']) * 100)
```

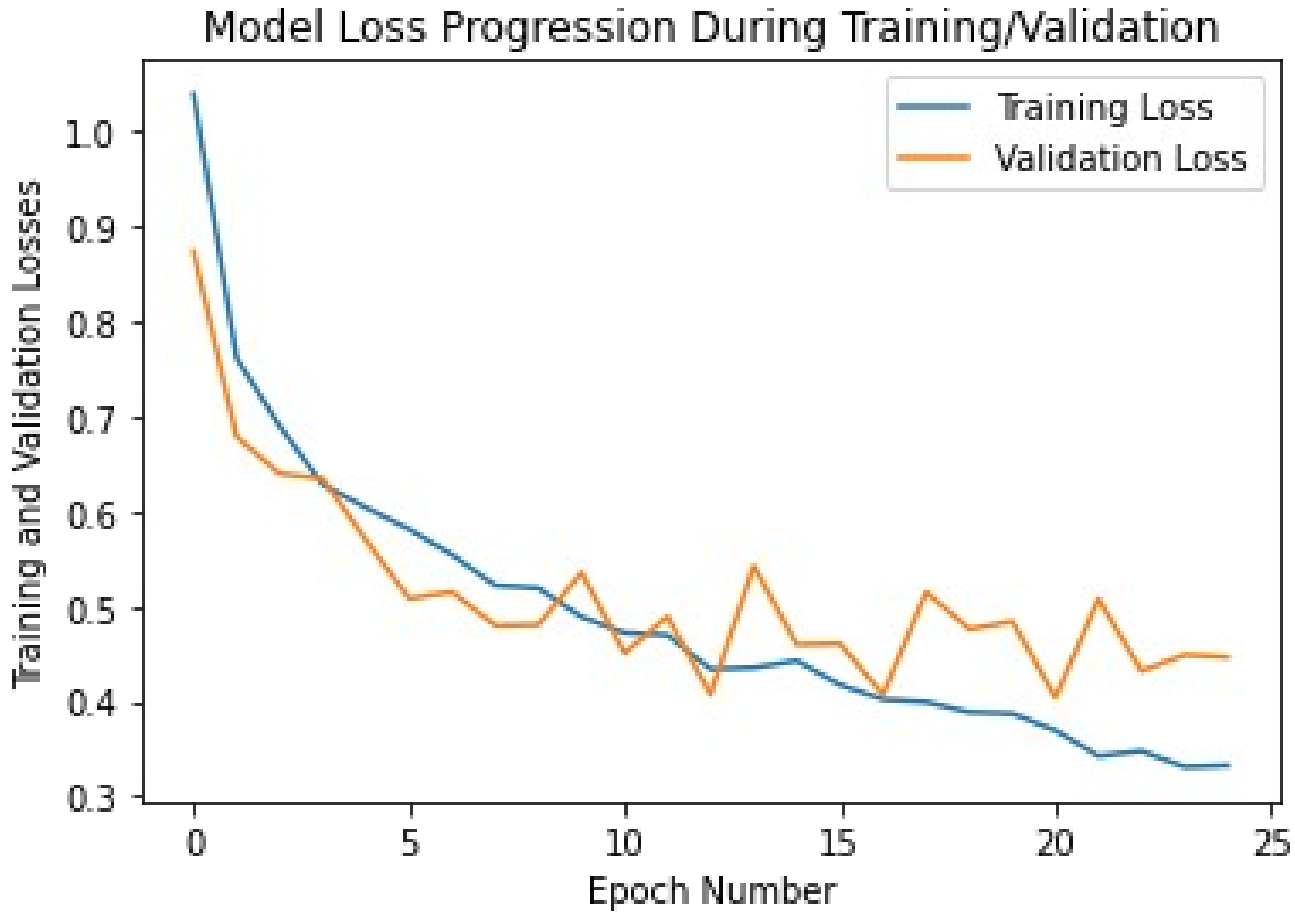
Classification Report

```
4/4 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

           0       0.87      0.92      0.89        60
           1       0.91      0.87      0.89        60

    accuracy                           0.89       120
   macro avg       0.89      0.89      0.89       120
weighted avg       0.89      0.89      0.89       120
```

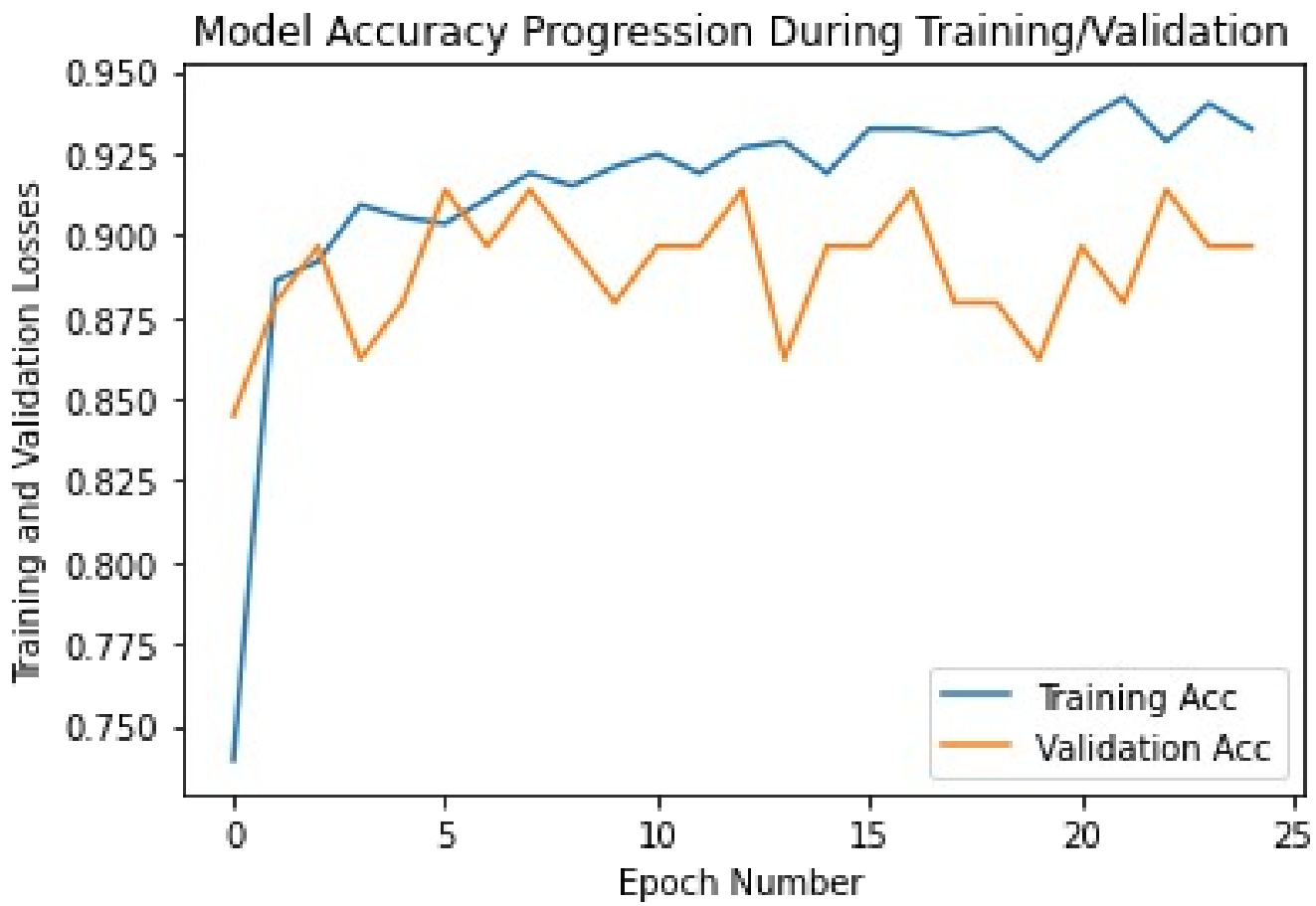
Confusion Matrix



Training and Validation Loss

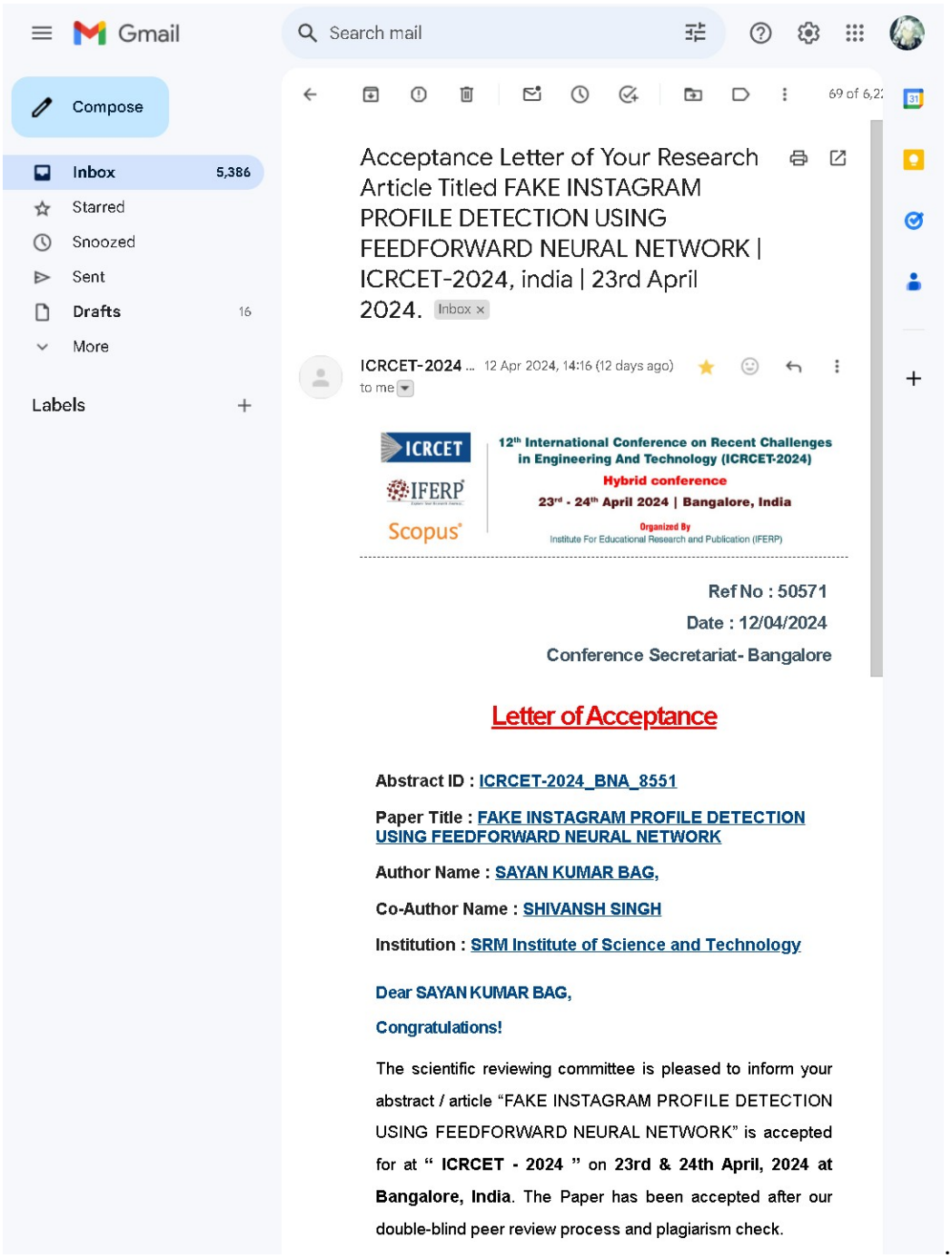


Training and Validation Accuracy



CONFERENCE PRESENTATION

Our paper on Fake Instagram Profile Detection using Feedforward Neural Network was presented at 12th International Conference on Recent Challenges In Engineering And Technology (ICRCET) .



Gmail

Search mail

Compose

Inbox 5,386

Starred

Snoozed

Sent

Drafts 16

More

Labels

Acceptance Letter of Your Research Article Titled FAKE INSTAGRAM PROFILE DETECTION USING FEEDFORWARD NEURAL NETWORK | ICRCET-2024, india | 23rd April 2024.

ICRCET-2024 ... 12 Apr 2024, 14:16 (12 days ago)

to me

ICRCET

IFERP

Scopus

12th International Conference on Recent Challenges in Engineering And Technology (ICRCET-2024)

Hybrid conference

23rd - 24th April 2024 | Bangalore, India

Organized By

Institute For Educational Research and Publication (IFERP)

Ref No : 50571

Date : 12/04/2024

Conference Secretariat- Bangalore

Letter of Acceptance

Abstract ID : ICRCET-2024_BNA_8551

Paper Title : FAKE INSTAGRAM PROFILE DETECTION USING FEEDFORWARD NEURAL NETWORK

Author Name : SAYAN KUMAR BAG,

Co-Author Name : SHIVANSH SINGH

Institution : SRM Institute of Science and Technology

Dear SAYAN KUMAR BAG,

Congratulations!

The scientific reviewing committee is pleased to inform your abstract / article "FAKE INSTAGRAM PROFILE DETECTION USING FEEDFORWARD NEURAL NETWORK" is accepted for at " **ICRCET - 2024** " on **23rd & 24th April, 2024 at Bangalore, India**. The Paper has been accepted after our double-blind peer review process and plagiarism check.

ICRCET 2024 Acceptance

PLAGIARISM REPORT

Fake prof

ORIGINALITY REPORT

10%	5%	5%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	"Human-Centric Smart Computing", Springer Science and Business Media LLC, 2024 Publication	1%
2	Submitted to University of Hertfordshire Student Paper	<1%
3	Jugal K. Kalita, Dhruba K. Bhattacharyya, Swarup Roy. "Artificial neural networks", Elsevier BV, 2024 Publication	<1%
4	Submitted to Heriot-Watt University Student Paper	<1%
5	medium.com Internet Source	<1%
6	www.iieta.org Internet Source	<1%
7	Regatte Varshith Reddy, Anish Nethi, Sanatan Sukhija, Yayati Gupta. "Detecting DeepFakes: A Deep Convolutional Neural Network Approach with Depth Wise Separable Convolutions", 2023 International Conference	<1%

	on Emerging Techniques in Computational Intelligence (ICETCI), 2023 Publication	
8	Submitted to Toronto Business College Student Paper	<1%
9	dokumen.pub Internet Source	<1%
10	odr.chalmers.se Internet Source	<1%
11	pure.tue.nl Internet Source	<1%
12	2os.io Internet Source	<1%
13	Submitted to Brunel University Student Paper	<1%
14	Milind Natu, Mrinal Bachute, Ketan Kotecha. "HCLA_CBiGRU: Hybrid convolutional bidirectional GRU based model for epileptic seizure detection", Neuroscience Informatics, 2023 Publication	<1%
15	thesai.org Internet Source	<1%
16	Zi-Cheng Wang, Si-Cheng Wang, Dong Li, Zhan-Wei Cao, Ya-Ling He. "An intelligent fault detection and diagnosis model for	<1%

refrigeration systems with a comprehensive feature selection method{fr}Un modèle intelligent de détection et de diagnostic des défauts pour les systèmes de réfrigération avec une méthode complète de sélection des caractéristiques", International Journal of Refrigeration, 2024
Publication

17	Submitted to Federation University Student Paper	<1 %
18	Submitted to Jacobs University, Bremen Student Paper	<1 %
19	Narasimha Murthy M S, Gunti Spandan, Inakollu Aswani, Shankar Nayak Bhukya, Ramachandra A C. "Deep Learning and MRI Improve Carotid Arterial Tree Reconstruction", 2023 International Conference on Data Science and Network Security (ICDSNS), 2023 Publication	<1 %
20	Submitted to South Bank University Student Paper	<1 %
21	Submitted to Middlesex University Student Paper	<1 %
22	Submitted to University of Southampton Student Paper	<1 %

23	Submitted to University of Wales Institute, Cardiff Student Paper	<1%
24	unsworks.unsw.edu.au Internet Source	<1%
25	Umashankar Kandpal, Rajat Kr Sharma, Arka Roy, K Sreel, Subrahmanya Kundapura. "Flood Susceptibility Modelling by Advanced Convolutional Neural Networks (CNN) in the foothills of Southern Western Ghats, Kerala, India", Research Square Platform LLC, 2024 Publication	<1%
26	Submitted to University of Exeter Student Paper	<1%
27	Submitted to University of South Florida Student Paper	<1%
28	www.frontiersin.org Internet Source	<1%
29	Aryan Raghav, Aayush Anand, Rishabh Sharma, Nirdesh Singh, Ch.Venkata RamiReddy. "Autism Spectrum Disorder Detection in Children Using Transfer Learning Techniques", 2023 2nd International Conference on Edge Computing and Applications (ICECAA), 2023 Publication	<1%

30	ir.kluniversity.in Internet Source	<1 %
31	sitecore93stage.heart.org Internet Source	<1 %
32	www.arxiv-vanity.com Internet Source	<1 %
33	www.bioenergyconsult.com Internet Source	<1 %
34	"Deep Learning for Unmanned Systems", Springer Science and Business Media LLC, 2021 Publication	<1 %
35	www.mdpi.com Internet Source	<1 %
36	JING CAI. "Deep learning multi-classification heart sound detection method based on high order spectrum and wavelet analysis", Research Square Platform LLC, 2024 Publication	<1 %
37	Submitted to Liverpool John Moores University Student Paper	<1 %
38	www.ijosi.org Internet Source	<1 %

PLAGIARISM REPORT

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY (Deemed to be University u/s 3 of UGC Act, 1956)		
Office of Controller of Examinations		
REPORT FOR PLAGIARISM CHECK ON THE DISSERTATION/PROJECT REPORTS FOR UG/PG PROGRAMMES **(To be attached in the dissertation/ project report)**		
1	Name of the Candidate **(IN BLOCK LETTERS)**	SAYAN KUMAR BAG SHIVANSH SINGH
2	Address of the Candidate	Malligai , SRM NAGAR , Kattankulathur ,Tamil Nadu Jains Avalon Springs, NandivaramGuduvancheri,Potheri, Tamil Nadu
3	Registration Number	RA2011003010951 RA2011003010950
4	Date of Birth	24/11/2001 , 28/06/2001
5	Department	Computer Science and Engineering
6	Faculty	Engineering and Technology, School of Computing
7	Title of the Dissertation/Project	FAKE INSTAGRAM PROFILE DETECTION USING FEEDFORWARD NEURAL NETWORK
8	Whether the above project /dissertation is done by	a.)Sayan Kumar Bag RA2011003010951 b) Shivansh Singh RA2011003010950
9	Name and address of the Supervisor / Guide	Dr. Kalaivani J Department of Computing Technologies SRM Institute of Science and Technology Kattankulathur, India **Mail ID:** kalaivaj@srmist.edu.in **Mobile Number:** 9790273451
10	Name and address of Co-Supervisor / Co-Guide (if any)	**Mail ID:** **Mobile Number:**

11	Software Used	Turnitin		
12	Date of Verification	23/04/2024		
13	**Plagiarism Details: (to attach the final report from the software)**			
Chapter	**Title of the Chapter**	**Percentage of similarity index (including self citation)**	**Percentage of similarity index (Excluding self-citation)**	**% of plagiarism after excluding Quotes, Bibliography, etc.,**
1	Abstract	0%	0%	0%
2	Introduction	<1%	<1%	<1%
3	Literature Survey	<1%	<1%	<1%
4	Architecture and Analysis	<1%	<1%	<1%
5	Methodology	<2%	<2%	<2%
6	Coding and Testing	<2%	<1%	<1%
7	Results and Discussion	<1%	<1%	<1%
8	Conclusion and Future Scope	<1%	<1%	<1%
	Appendices	0%	0%	0%

I / We declare that the above information have been verified and found true to the best of my / our knowledge.

Sayan Kumar Bag
Shivansh Singh
Signature of the Candidate

J.Calaivani
[Dr. J. KALAIVANI]
Name & Signature of the Staff (Who uses the plagiarism check software)

J.Calaivani
[Dr J. KALAIVANI]
Name & Signature of the Supervisor/ Guide

Name & Signature of the Co-Supervisor/Co-Guide

M. Pushpalatha
Name & Signature of the HOD

Department of Computing Technologies
Kattankulathur
603 203
SRM IST