“INTRUSION DETECTION USING DEEP LEARNING”

MAJOR PROJECT

SUBMITTED in PARTIAL FULFILLMENT of

THE REQUIREMENT FOR THE AWARD OF THE DEGREE of

BACHELOR OF TECHNOLOGY

(COMPUTER SCIENCE AND ENGINEERING)

SUBMITTED BY

AMRIT SINGH – 00696302715

AYUSH SHARMA – 01296302715

VARDAAN ARORA - 40596302715

Under the guidance of

Mr. Adeel Hashmi

Assistant Professor, Dept of CSE

Department of Computer Science and Engineering

MAHARAJA SURAJMAL INSTITUTE OF TECHNOLOGY,

JANAKPURI DELHI-58

GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY

DELHI, INDIA

MAY-2019

ACKNOWLEDGEMENT

We truly acknowledge the cooperation and help made by Mr Adeel Hashmi, Assistant Professor, Department of Computer Science and Engineering, Maharaja Surajmal Institute of Technology, C-4, Janakpuri , Delhi, Guru Gobind Singh Indraprastha University, Delhi. He has been a constant source of guidance throughout the course of this project. We would also like to thank him for his help and guidance in understanding intrusion detection using deep learning. We are also thankful to our friends and family whose silent support led us to complete our project.

(Signature)
Amrit Singh (00696302715)
Ayush Sharma (01296302715)
Vardaan Arora (40596302715)

CERTIFICATE

This is to certify that the project entitled “Intrusion detection using deep learning” is a bonafide work carried out by Mr. Amrit Singh, Ayush Sharma and Vardaan Arora under my guidance and supervision and submitted in partial fulfillment of B.Tech degree in Computer Science and Engineering of Guru Gobind Singh Indraprastha University, Delhi. The work embodied in this project has not been submitted for any other degree or diploma.

Mr. Adeel Hashmi
Assistant Professor
Dept. of Computer Science and Engineering
MSIT, Delhi

Dr. Naveen Dahiya
Head, Department of Computer Science and Engineering, 2nd shift
MSIT, Delhi.

LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Introduction

An Intrusion Detection System is an application used for monitoring the network and protecting it from the intruder. With the rapid progress in the internet based technology new application areas for computer network have emerged. In instances, the fields like business, financial, industry, security and healthcare sectors the LAN and WAN applications have progressed. All of these application areas made the network an attractive target for the abuse and a big vulnerability for the community. Malicious users or hackers use the organization's internal systems to collect information and cause vulnerabilities like software bugs, Lapse in administration, leaving systems to default configuration.

As network behaviors and patterns change and intrusions evolve, it has very much become necessary to move away from static and one-time datasets toward more dynamically generated datasets which not only reflect the traffic compositions and intrusions of that time, but are also modifiable, extensible, and reproducible.

So to do this, we are going to train a deep learning model to identify anomaly from a given data set.

Due to the application of machine learning within the system, anomaly-based detection is rendered the most effective among the intrusion detection systems as they have no need to search for any specific pattern of anomaly, but they rather just treat anything that does not match the profile as "Anomalous".

1.1.1 Purpose

Imbalanced class distribution is a scenario where the number of observations belonging to one class is significantly lower than those belonging to the other classes. This problem is predominant in scenarios where anomaly detection is crucial like electricity pilferage, fraudulent transactions in banks etc. This happens because machine learning algorithms are usually designed to improve accuracy by reducing the error. Thus, they do not consider the class distribution / proportion or balance of classes. This leads to false classification of many frauds as genuine transactions, which leads to losses to organizations and individuals. To combat this, we explore different resampling techniques to balance the two classes, i.e. majority and minority by using techniques like undersampling and oversampling. We also explore deep learning methods and transfer learning approaches.

1.1.2 Project Scope

The scope of anomaly detection is not just limited to intrusion detection. Outlier detection is a major problem in many notable fields like electricity pilferation, detecting oil spills, detection of diseases. Intrusion detection was chosen as to demonstrate the capabilities of the discussed techniques. This can be used by major security organizations of the world like banks and networking companies, which can make network much secure and sensitive to intrusion detection.

1.1.3 Operating environment

Hardware:

- Central Processing Unit (CPU)—Intel Core i5 6th Generation processor or higher. An AMD equivalent processor will also be optimal.
- Operating System—Microsoft Windows 10 (64-bit recommended) Pro or Home.
- Graphics Processing Unit (GPU)—NVIDIA GeForce GTX 940 or higher. AMD GPUs are not able to perform deep learning regardless. NVIDIA CUDA-enabled GPUs for deep learning
- RAM—8 GB minimum, 16 GB or higher is recommended.

Software:

- Text editor: PyCharm, Jupyter notebook
- Libraries: Matplotlib, Numpy, Scikit-learn, pandas, keras, tensorflow
- Language: Python

1.1.4 Methodology

At ISCX, a systematic approach to generate the required datasets is introduced to address this need. The underlying notion is based on the concept of profiles which contain detailed descriptions of intrusions and abstract distribution models for applications, protocols, or lower level network entities. Real traces are analyzed to create profiles for agents that generate real traffic for HTTP, SMTP, SSH, IMAP, POP3, and FTP. The UNB ISCX IDS 2012 dataset consists of labeled network traces, including full packet payloads in pcap format, which along with the relevant profiles are publicly available for researchers.

Since the dataset is in pcap format, which is unusable in machine learning models, we need to convert the dataset files into machine readable formats like csv or xml. XML format is

chosen because it is easily parsable and required tags can be extracted by using code. Two tags are focussed here : *destinationPayloadAsUTF* and *Tag*. The value of these tags is extracted using ETree library in Java with a program called ISCXFlowMeter. The 2 features extracted from the XML file are used for further processing. The <destinationPayloadAsUTF> tag is first checked if it is empty or not . If it is not empty then the <Tag> tag is processed . A new array called data_array is made which contains binary values and the the value of those tags is appended into the array. Now the <Tag> tag is found out and is checked if the text contained within it is Normal or not. If the text is normal then the binary value 0 is written in the array in a new row. If the text is not normal then the binary value 1 is written in the array in a new row which signifies that the given Tag is anomalous. New numpy file is created and saved in the memory for further processing .

This processed data is fed into a deep learning model based on transfer learning technique on the VGG-19 Keras pre-trained model. We did not utilize the pre-trained weights of VGG-19 since our images were custom images we therefore opted to train the model on our image set using the conventional technique of anomaly-detection using training and test phases.

1.1.5 Design and implementation constraints

- The machine must have all the latest version of libraries installed on the system to be able to run the machine learning model.
- The machine must have at least 8-16 GB of RAM for processing of dataset as the dataset is collected over 7 days and is very large in size.
- As python 3 does not have backwards compatibility with python 2.7 no computers using older version of python can run the script.

CHAPTER 2

LITERATURE SURVEY

2.1 Abstract of the papers referred

2.1.1 Intrusion detection system – A study[1]

Intrusion Detection System (IDS) is meant to be a software application which monitors the network or system activities and finds if any malicious operations occur. Tremendous growth and usage of internet raises concerns about how to protect and communicate the digital information in a safe manner.[1]

Nowadays, hackers use different types of attacks for getting the valuable information. Many intrusion detection techniques, methods and algorithms help to detect these attacks. This main objective of this paper is to provide a complete study about the definition of intrusion detection, history, life cycle, types of intrusion detection methods, types of attacks, different tools and techniques, research needs, challenges and applications.[1]

2.1.2 Intrusion Detection Using Machine Learning: A Comparison Study[3]

With the advancement of internet over years, the number of attacks over internet has also increased. A powerful Intrusion Detection System (IDS) is required to ensure the security of a network. The aim of IDS is to monitor the processes prevailing in a network and to analyze them for signs of any possible deviations. Some studies have been done in this field but a deep and exhaustive work has still not been done.[3]

This paper proposes an IDS using machine learning for network with a good union of feature selection technique and classifier by studying the combinations of most of the popular feature selection techniques and classifiers. A set of significant features is selected from the original set of features using feature selection techniques and then the set of significant features is used to train different types of classifiers to make the IDS. Five folds cross validation is done on NSL-KDD dataset to find results. It is finally observed that K-NN classifier produces better performance than others and, among the feature selection methods, information gain ratio based feature selection method is better.[3]

2.1.3 A comprehensive survey of data mining-based intrusion detection research.[7]

This survey paper categorises, compares, and summarises from almost all published technical and review articles in automated fraud detection within the last 10 years. It defines the professional fraudster, formalises the main types and subtypes of known fraud, and presents the nature of data evidence collected within affected industries.[7] Within the business context of mining the data to achieve higher cost savings, this research presents methods and techniques together with their problems. Compared to all related reviews on fraud detection, this survey covers much more technical articles and is the only one, to the best of our knowledge, which proposes alternative data and solutions from related domains.[7]

2.1.4 Toward developing a systematic approach to generate benchmark datasets for intrusion detection[8]

In network intrusion detection, anomaly-based approaches in particular suffer from accurate evaluation, comparison, and deployment which originates from the scarcity of adequate datasets. Many such datasets are internal and cannot be shared due to privacy issues, others are heavily anonymized and do not reflect current trends, or they lack certain statistical characteristics.[8] These deficiencies are primarily the reasons why a perfect dataset is yet to exist. Thus, researchers must resort to datasets that are often suboptimal. As network behaviors and patterns change and intrusions evolve, it has very much become necessary to move away from static and one-time datasets toward more dynamically generated datasets which not only reflect the traffic compositions and intrusions of that time, but are also modifiable, extensible, and reproducible. In this paper, a systematic approach to generate the required datasets is introduced to address this need. The underlying notion is based on the concept of profiles which contain detailed descriptions of intrusions and abstract distribution models for applications, protocols, or lower level network entities.[8] Real traces are analyzed to create profiles for agents that generate real traffic for HTTP, SMTP, SSH, IMAP, POP3, and FTP. In this regard, a set of guidelines is established to outline valid datasets, which set the basis for generating profiles. These guidelines are vital for the effectiveness of the dataset in terms of realism, evaluation

capabilities, total capture, completeness, and malicious activity. The profiles are then employed in an experiment to generate the desirable dataset in a testbed environment. Various multi-stage attacks scenarios were subsequently carried out to supply the anomalous portion of the dataset. The intent for this dataset is to assist various researchers in acquiring datasets of this kind for testing, evaluation, and comparison purposes, through sharing the generated datasets and profiles.[8]

2.1.5 An analysis of deep neural network problems for practical applications[5]

Since the emergence of Deep Neural Networks (DNNs) as a prominent technique in the field of computer vision, the ImageNet classification challenge has played a major role in advancing the state-of-the-art.[5] While accuracy figures have steadily increased, the resource utilisation of winning models has not been properly taken into account. In this work, we present a comprehensive analysis of important metrics in practical applications: accuracy, memory footprint, parameters, operations count, inference time and power consumption. Key findings are: (1) power consumption is independent of batch size and architecture; (2) accuracy and inference time are in a hyperbolic relationship; (3) energy constraint is an upper bound on the maximum achievable accuracy and model complexity; (4) the number of operations is a reliable estimate of the inference time. We believe our analysis provides a compelling set of information that helps design and engineer efficient DNNs.[5]

2.1.6 A Deep Learning Approach for Network Intrusion Detection System[6]

A Network Intrusion Detection System (NIDS) helps system administrators to detect network security breaches in their organizations.[6] However, many challenges arise while developing a flexible and efficient NIDS for unforeseen and unpredictable attacks. We propose a deep learning based approach for developing such an efficient and flexible NIDS. We use Self-taught Learning (STL), a deep learning based technique, on NSL-KDD - a benchmark dataset for network intrusion.[6] We present the performance of our approach and compare it with a few previous work. Compared metrics include accuracy, precision, recall, and f-measure values.[6]

2.2 Findings from the research papers

2.2.1 Intrusion detection system – A study[1]

From "Intrusion detection system – A study" by Dr. S.Vijayarani1 and Ms. Maria Sylviaa.S , intrusion Detection System (IDS) is meant to be a software application which monitors the network or system activities and finds if any malicious operations occur. Tremendous growth and usage of internet raises concerns about how to protect and communicate the digital information in a safe manner.[1]

Nowadays, hackers use different types of attacks for getting the valuable information. Many intrusion detection techniques, methods and algorithms help to detect these attacks. This main objective of this paper is to provide a complete study about the definition of intrusion detection, history, life cycle, types of intrusion detection methods, types of attacks, different tools and techniques, research needs, challenges and applications.[1]

2.2.2 A Multiple Resampling Method for Learning from Imbalanced Data Sets[4]

From "A Multiple Resampling Method for Learning from Imbalanced Data Sets" by Andrew Estabrooks it is found out that in a concept- learning problem, the data set is said to present a class imbalance if it contains many more examples of one class than the other.[4] Such a situation poses challenges for typical classifiers such as Decision Tree Induction Systems or Multi-Layer Perceptrons that are designed to optimize overall accuracy without taking into account the relative distribution of each class (Japkowicz & Stephen 2002; Estabrooks 2000). As a result, these classifiers tend to ignore small classes while concentrating on classifying the large ones accurately.[4]

Evaluation of a classification algorithm performance is measured by the Confusion Matrix which contains information about the actual and the predicted class.[4]

Random Undersampling aims to balance class distribution by randomly eliminating majority class examples. This is done until the majority and minority class instances are balanced out.

Advantage is that it can help improve run time and storage problems by reducing the number of training data samples when the training data set is huge.[4]

2.2.3 Intrusion Detection Using Machine Learning: A Comparison Study[3]

From Paper of "*Intrusion Detection Using Machine Learning: A Comparison Study*" by Saroj Kr. Biswas it is found that the paper uses different mix of feature selection algorithms and classifiers because each of the classifiers and the feature selection algorithms has advantage as well as disadvantage.[3] It is difficult to choose one over another to implement an intrusion detection system. Further, the experimental results show that machine learning can be used in intrusion detection because all the combinations produce significant accuracy. k-NN classifier produces better performance than others and, among the feature selection methods,IGR feature selection method is better than others and CFS is inferior to others.[3]

2.2.4 An analysis of deep neural network problems for practical applications[5]

*"*An analysis of deep neural network problems for practical applications*"* states the analysis of multiple state-of-the-art deep neural networks submitted to the ImageNet challenge, in terms of accuracy, memory footprint, parameters, operations count, inference time and power consumption.[5] The goal is to provide insights into the design choices that can lead to efficient neural networks for practical application, and optimisation of the often-limited resources in actual deployments, which lead to the creation of ENet — or Efficient-Network — for ImageNet.[5] It is shown that accuracy and inference time are in a hyperbolic relationship: a little increment in accuracy costs a lot of computational time. It is shown that number of operations in a network model can effectively estimate inference time. It is shown that an energy constraint will set a specific upper bound on the maximum achievable accuracy and model complexity, in terms of operations counts.[5]

CHAPTER 3

TECHNOLOGIES IN DOMAIN

3.1 Machine Learning

3.1.1 Introduction

- Machine learning (ML) is a field of artificial intelligence that uses statistical techniques to give computer systems the ability to "learn" (e.g., progressively improve performance on a specific task) from data, without being explicitly programmed.

- The name machine learning was coined in 1959 by Arthur Samuel. Machine learning explores the study and construction of algorithms that can learn from and make predictions on data – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions, through building a model from sample inputs. Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or infeasible; example applications include email filtering, detection of network intruders, and computer vision.

- Machine learning is closely related to (and often overlaps with) computational statistics, which also focuses on prediction-making through the use of computers. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field. Machine learning is sometimes conflated with data mining, where the latter subfield focuses more on exploratory data analysis and is known as unsupervised learning.

- Within the field of data analytics, machine learning is a method used to devise complex models and algorithms that lend themselves to prediction; in commercial use, this is known as predictive analytics. These analytical models allow researchers, data scientists, engineers, and analysts to "produce reliable, repeatable decisions and results" and uncover "hidden insights" through learning from historical relationships and trends in the data.

From figure 3.1, we infer the steps that take place in building a machine learning model.

Feature extraction is performed first, and only contributing features make it to the machine learning algorithm, where the model is formed which is used to perform a multitude of tasks, ranging from prediction, classification and clustering.

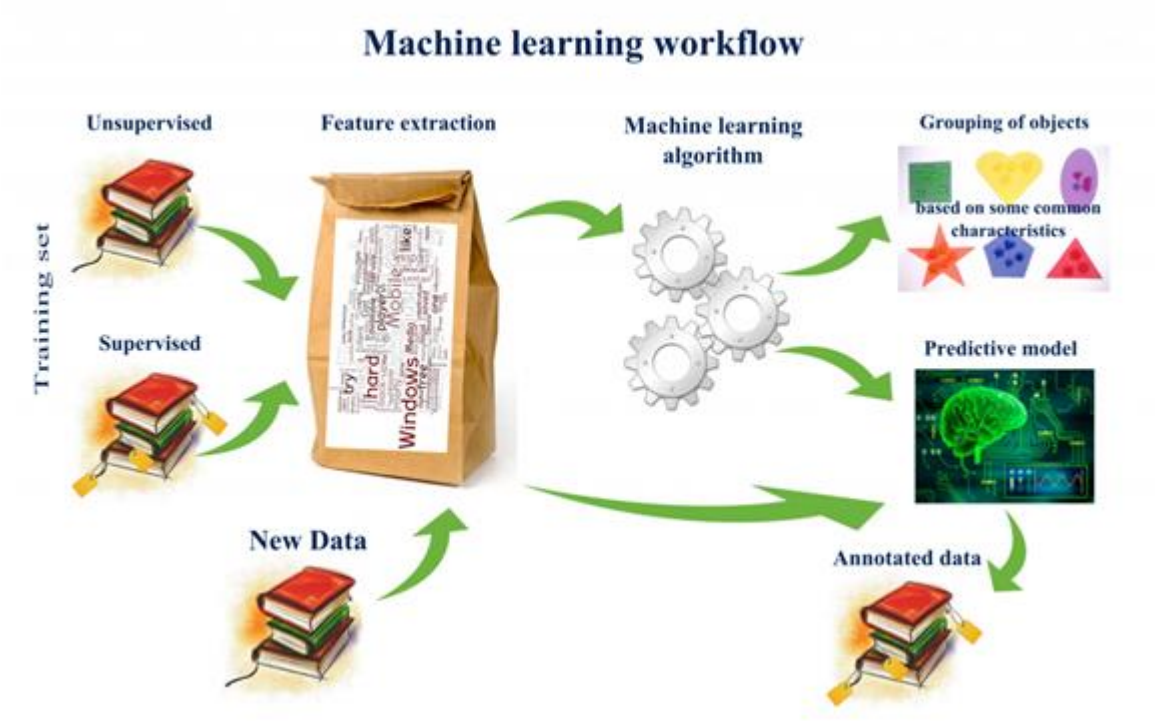

Figure 3.1: Machine Learning Workflow

3.1.2 Need for Machine Learning

- Machine Learning is a field which is raised out of Artificial Intelligence(AI). Applying AI, we wanted to build better and intelligent machines. But except for few mere tasks such as finding the shortest path between point A and B, we were unable to program more complex and constantly evolving challenges. There was a realisation that the only way to be able to achieve this task was to let machine learn from itself. This sounds similar to a child learning from its self. So machine learning was developed as a new capability for computers. And now machine learning is present in so many segments of technology, that we don't even realise it while using it.

- Finding patterns in data on planet earth is possible only for human brains. The data being very massive, the time taken to compute is increased, and this is where Machine Learning comes into action, to help people with large data in minimum time.

- If big data and cloud computing are gaining importance for their contributions, machine learning as technology helps analyse those big chunks of data, easing the task of data scientists in an automated process and gaining equal importance and recognition.

- The techniques we use for data mining have been around for many years, but they were not effective as they did not have the competitive power to run the algorithms. If you run deep learning with access to better data, the output we get will lead to dramatic breakthroughs which is machine learning.

Actual	Predicted	
	Positive Class	Negative Class
Positive Class	True Positive(TP)	False Negative (FN)
Negative Class	False Positive (FP)	True Negative (TN)

Fig 3.2 Accuracy of a model = (TP+TN) / (TP+FN+FP+TN)

Fig 3.2 shows the metrics to measure the performance of a machine learning model.

3.1.3 Supervised Learning

Supervised learning is a type of machine learning algorithm. A majority of practical machine learning uses supervised learning.

In supervised learning, the system tries to learn from the previous examples that are given. (On the other hand, in unsupervised learning, the system attempts to find the patterns directly from the example given.)Speaking mathematically, supervised learning is where you have both input variables (x) and output variables(Y) and can use an algorithm to derive the mapping function from the input to the output.

The mapping function is expressed as Y = f(X).

In order to solve a given problem of supervised learning, one has to perform the following steps:

1. Determine the type of training examples. Before doing anything else, the user should decide what kind of data is to be used as a training set. In case of handwriting analysis, for example, this might be a single handwritten character, an entire handwritten word, or an entire line of handwriting.
2. Gather a training set. The training set needs to be representative of the real-world use of the function. Thus, a set of input objects is gathered and corresponding outputs are also gathered, either from human experts or from measurements.
3. Determine the input feature representation of the learned function. The accuracy of the learned function depends strongly on how the input object is represented. Typically, the input object is transformed into a feature vector, which contains a number of features that are descriptive of the object. The number of features should not be too large, because of the curse of dimensionality; but should contain enough information to accurately predict the output.
4. Determine the structure of the learned function and corresponding learning algorithm. For example, the engineer may choose to use support vector machines or decision trees.
5. Complete the design. Run the learning algorithm on the gathered training set. Some supervised learning algorithms require the user to determine certain control parameters. These parameters may be adjusted by optimizing performance on a subset (called a validation set) of the training set, or via cross-validation.
6. Evaluate the accuracy of the learned function. After parameter adjustment and learning, the performance of the resulting function should be measured on a test set that is separate from the training set.

3.1.4 Classification

In machine learning and statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. Examples are assigning a given email to the "spam" or "non-spam" class, and assigning a diagnosis to a given patient based on observed characteristics of the patient (sex, blood

pressure, presence or absence of certain symptoms, etc.). Classification is an example of pattern recognition.

In the terminology of machine learning, classification is considered an instance of supervised learning, i.e., learning where a training set of correctly identified observations is available. The corresponding unsupervised procedure is known as clustering, and involves grouping data into categories based on some measure of inherent similarity or distance.

Often, the individual observations are analysed into a set of quantifiable properties, known variously as explanatory variables or features. These properties may variously be categorical (e.g. "A", "B", "AB" or "O", for blood type), ordinal (e.g. "large", "medium" or "small"), integer-valued (e.g. the number of occurrences of a particular word in an email) or real-valued (e.g. a measurement of blood pressure). Other classifiers work by comparing observations to previous observations by means of a similarity or distance function.

An algorithm that implements classification, especially in a concrete implementation, is known as a classifier. The term "classifier" sometimes also refers to the mathematical function, implemented by a classification algorithm, that maps input data to a category.

Terminology across fields is quite varied. In statistics, where classification is often done with logistic regression or a similar procedure, the properties of observations are termed explanatory variables (or independent variables, regressors, etc.), and the categories to be predicted are known as outcomes, which are considered to be possible values of the dependent variable. In machine learning, the observations are often known as instances, the explanatory variables are termed features (grouped into a feature vector), and the possible categories to be predicted are classes. Other fields may use different terminology: e.g. in community ecology, the term "classification" normally refers to cluster analysis, i.e., a type of unsupervised learning, rather than the supervised learning described in this article.

From fig 3.3, we see the difference between classification and regression. Regression and classification are both related to prediction, where regression predicts a value from a continuous set, whereas classification predicts the 'belonging' to the class.

For example, the price of a house depending on the 'size' (in some unit) and say 'location' of the house, can be some 'numerical value' (which can be continuous): this relates to regression.

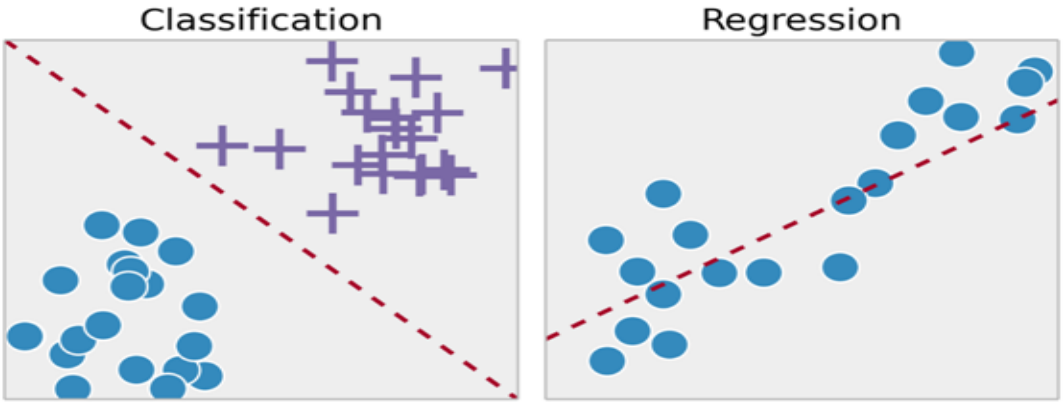

Figure 3.3: Classification vs Regression

Similarly, the prediction of price can be in words, viz., 'very costly', 'costly', 'affordable', 'cheap', and 'very cheap': this relates to classification. Each class may correspond to some range of values.

3.1.5 Deep Learning

Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on the layers used in artificial neural networks. Learning can be supervised, semi-supervised or unsupervised. Deep learning architectures such as deep neural networks, deep belief networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases superior to human experts. Neural networks were originally inspired by information processing and distributed communication nodes in biological systems synaptic structures yet have various differences from the structural and functional properties of biological brains, which make them incompatible with the neurological evidence. Specifically, Neural networks tend to be static and symbolic, while the biological brain of most living organisms is dynamic (plasticity) and analog.

CHAPTER 4

RESEARCH METHODOLOGY

4.1 Problem Statement

1. Imbalanced class distribution is a scenario where the number of observations belonging to one class is significantly lower than those belonging to the other classes.
2. This problem is predominant in scenarios where anomaly detection is crucial like electricity pilferage, fraudulent transactions in banks etc.
3. This happens because Machine Learning Algorithms are usually designed to improve accuracy by reducing the error. Thus, they do not consider the class distribution / proportion or balance of classes.
4. Standard classifier algorithms like Decision Tree have a bias towards classes which have number of instances. They tend to only predict the majority class data. The features of the minority class are treated as noise and are often ignored. Thus, there is a high probability of misclassification of the minority class as compared to the majority class.

4.2 Objectives

- Solving imbalanced dataset classification problem in detecting intrusions by using deep learning.
- Extracting features from ISCX 2012 Dataset collected by the Canadian Institute for Cybersecurity.
- Using deep learning methods (keras) and transfer learning techniques in models like VGG-19.

4.3 Solution proposed

4.3.1. Exploratory Data Analysis

To judge which type of attacks are intrusive, and to check how the value of attribute *Tag* is set. Queries are made on the dataset and relations are found. As it turns out, a few of the features are unclear on how the value is set to determine the corresponding *Tag* value, and hence need to be dropped.

4.3.2. Data Cleaning

Since the dataset is in pcap format, which is unusable in machine learning models, we need to convert the dataset files into machine readable formats like .csv or .xml. They are chosen because they are easily parsable and required tags can be extracted by using code. Two

attributes are extracted here : *destinationPayloadAsUTF* and *Tag.* The value of these tags is extracted using ETree library in python and a program called ISCXFlowMeter.

ISCXFlowMeter is a network traffic flow generator. It can be used to generate bidirectional flows, where the first packet determines the forward (source to destination) and backward (destination to source) directions, hence the statistical time-related features can be calculated separately in the forward and backward directions. Additional functionalities include, selecting features from the list of existing features, adding new features, and controlling the duration of flow timeout. This flowmeter takes in the .pcap files and converts them into a readable .xml file format that stacks each data part as flows where the first packet determines the forward (source to destination) and backward (destination to source) directions.
The .pcap file extension is mainly used for analyzing networks. pcap files are data files created using a program and they contain the packet data of a network. These files are mainly used in analyzing the network characteristics of a certain data.

4.3.3. Feature Engineering

Process the XML data , extract and convert two XML tags into Numpy Arrays.
The XML file is processed and converted to numpy array file . The XML tree is traversed and the 2 features are extracted and used for data preprocessing.

- destinationPayloadAsUTF
- Tag

Multiple xml files are read and are preprocessed consecutively. If there is an error in reading a file then the error is generated else the counter is incremented by one.

The 2 features extracted from the XML file are used for further processing . The <destinationPayloadAsUTF> tag is first checked if it is empty or not . If it is not empty then the <Tag> tag is processed . A new array called data_array is made which contains binary values. Now the <Tag> tag is found out and is checked if the text contained within it is Normal or not.
If the text is normal then the binary value 0 is written in the array in a new row with the help of vstack() python function.

If the text is not normal then the binary value 1 is written in the array in a new row which signifies that the given Tag is anomalous.
New numpy file is created and saved in the memory for further processing .

4.3.4. Machine Learning to Detect intrusion in highly skewed data.

Finally, numpy array is fed as input to the deep learning model using transfer learning on models like VGG-19.

4.4 Methodology and technologies used

4.4.1 Transfer learning

- Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.
- It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems.
- It is common to perform transfer learning with predictive modeling problems that use image data as input.
- This may be a prediction task that takes photographs or video data as input.
- For these types of problems, it is common to use a deep learning model pre-trained for a large and challenging image classification task such as the ImageNet 1000-class photograph classification competition.
- The research organizations that develop models for this competition and do well often release their final model under a permissive license for reuse. These models can take days or weeks to train on modern hardware.
- These models can be downloaded and incorporated directly into new models that expect image data as input.
- This approach is effective because the images were trained on a large corpus of photographs and require the model to make predictions on a relatively large number of classes, in turn, requiring that the model efficiently learn to extract features from photographs in order to perform well on the problem.

4.4.2 VGG-19 model

- VGG-19 is a convolutional neural network that is trained on more than a million images from the ImageNet database. The network is 19 layers deep and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224.
- It scored first place on the image localization task and second place on the image classification task.
- The VGG() class takes a few arguments that may only interest you if you are looking to use the model in your own project, e.g. for transfer learning.

For example:

- include_top (True): Whether or not to include the output layers for the model. You don't need these if you are fitting the model on your own problem.
- weights ('imagenet'): What weights to load. You can specify None to not load pre-trained weights if you are interested in training the model yourself from scratch.
- input_tensor (None): A new input layer if you intend to fit the model on new data of a different size.
- input_shape (None): The size of images that the model is expected to take if you change the input layer.
- pooling (None): The type of pooling to use when you are training a new set of output layers.
- classes (1000): The number of classes (e.g. size of output vector) for the model.

Syntax

net = vgg19

Description

net = vgg19 returns a pretrained VGG-19 network.

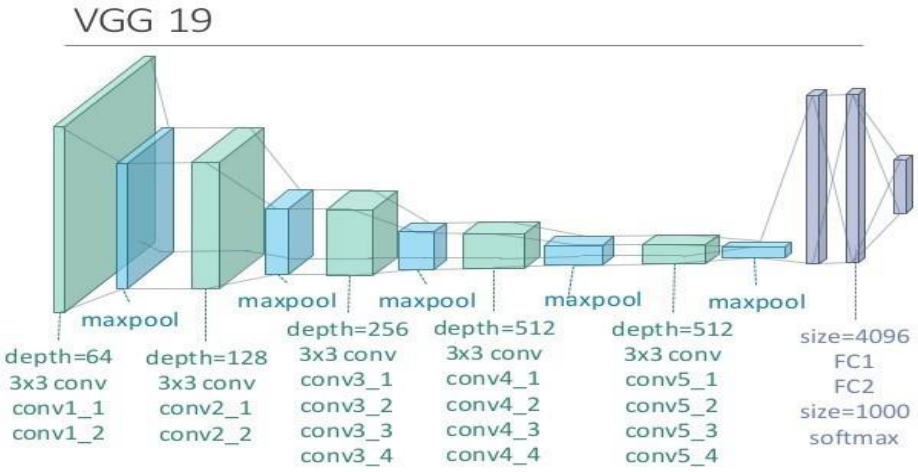

Fig 4.1: VGG19 model layout

Fig 4.1 shows the layout of VGG19 model and information about its layers. The network has 47 layers. There are 19 layers with learnable weights: 16 convolutional layers, and 3 fully connected layers.

CHAPTER – 5

IMPLEMENTATION AND RESULTS

5.1. Synthetic dataset generated by Information Security Centre of Excellence (ISCX), Faculty of Computer Science, University of New Brunswick

In network intrusion detection (IDS), anomaly-based approaches in particular suffer from accurate evaluation, comparison, and deployment which originates from the scarcity of adequate datasets. Many such datasets are internal and cannot be shared due to privacy issues, others are heavily anonymized and do not reflect current trends, or they lack certain statistical characteristics.

Privacy concerns related to sharing real network traces has been one of the major obstacles for network security researchers as data providers are often reluctant to share such information. Consequently, most such traces are either used internally, which limits other researchers from accurately evaluating and comparing their systems, or are heavily anonymized with the payload entirely removed resulting in decreased utility to researchers.

Real traces are analyzed to create profiles for agents that generate real traffic for HTTP, SMTP, SSH, IMAP, POP3, and FTP. Various multi-stage attacks scenarios were subsequently carried out to supply the anomalous portion of the dataset.

The UNB ISCX IDS 2012 dataset consists of labeled network traces, including full packet payloads in pcap format. The ISCX-IDS-2012 intrusion detection evaluation dataset consists of the following 7 days of network activity (normal and malicious):

Day, Date, Description, Size (GB)

- Friday, 11/6/2010, Normal Activity. No malicious activity, 16.1
- Saturday, 12/6/2010, Normal Activity. No malicious activity, 4.22
- Sunday, 13/6/2010, Infiltrating the network from inside + Normal Activity, 3.95
- Monday, 14/6/2010, HTTP Denial of Service + Normal Activity, 6.85
- Tuesday, 15/6/2010, Distributed Denial of Service using an IRC Botnet, 23.4
- Wednesday, 16/6/2010, Normal Activity. No malicious activity, 17.6
- Thursday, 17/6/2010, Brute Force SSH + Normal Activity, 12.3

5.2 Platforms

Following are the platforms which were used for the completion of our project.

5.2.1 Anaconda

Anaconda is a free and open-source distribution of the Python and R programming languages for data science and machine learning applications (large-scale data processing, predictive analytics, scientific computing), that aims to simplify package management and deployment. Package versions are managed by the package management system conda. The Anaconda distribution is used by over 6 million users and includes more than 250 popular data-science packages suitable for Windows, Linux, and MacOS.

5.2.2 Anaconda Navigator

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux. Navigator is automatically included with Anaconda version 4.0.0 or higher (See Fig.5.1).

The following applications are available by default in Navigator:

- JupyterLab
- Jupyter Notebook
- QtConsole
- Spyder
- Glueviz
- Orange
- Rstudio
- Visual Studio Code

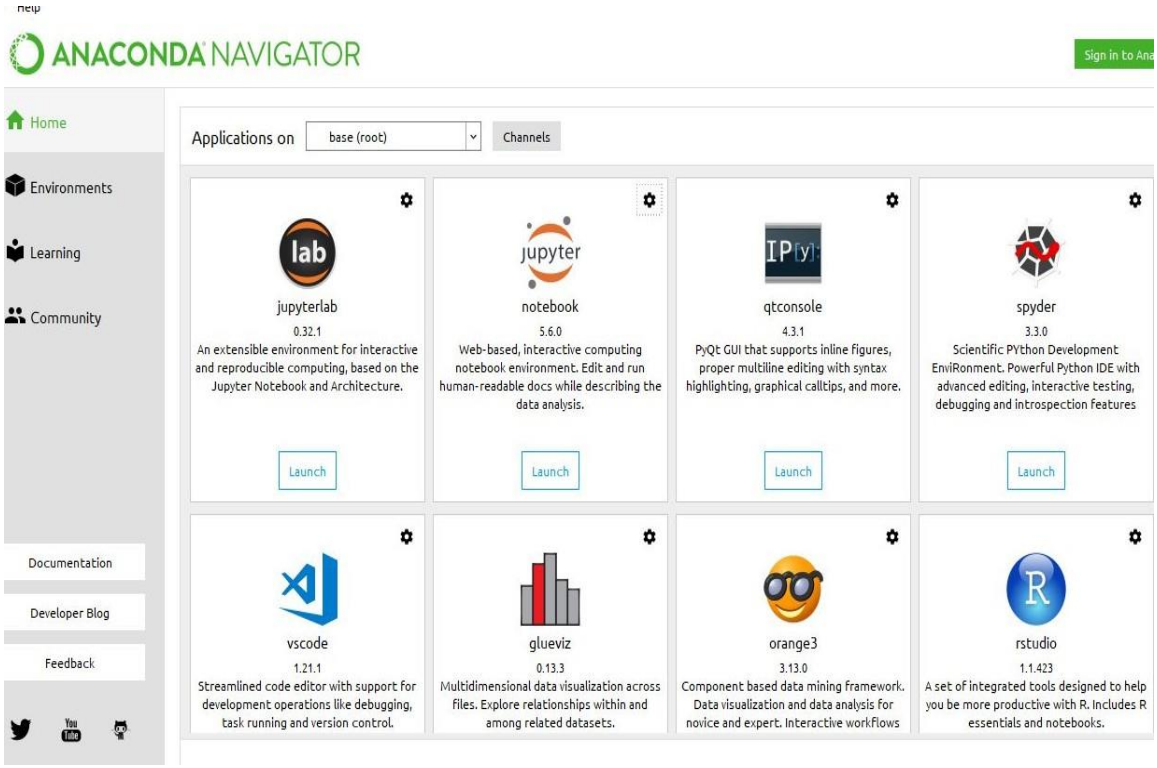

Fig.5.1: Anaconda Navigator

5.3 Libraries and packages used in project

Following are the libraries and packages used in our project.

5.3.1 NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities
- Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

5.3.2 Pandas

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labelled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal.

Pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
- Any other form of observational / statistical data sets. The data actually need not be labelled at all to be placed into a pandas data structure.

Advantages of Pandas

- Easy handling of missing data (represented as NaN) in floating point as well as non-floating-point data
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
- Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let Series, DataFrame, etc. automatically align the data for you in computations
- Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data
- Make it easy to convert ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects • Intelligent label-based slicing, fancy indexing, and subsetting of large data sets
- Intuitive merging and joining data sets
- Flexible reshaping and pivoting of data sets
- Hierarchical labeling of axes (possible to have multiple labels per tick)
- Robust IO tools for loading data from flat files (CSV and delimited), Excel files,

databases, and saving / loading data from the ultrafast HDF5 format.

- Time series-specific functionality: date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging, etc.

5.3.3 Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hard copy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits. Pyplot is a matplotlib module which provides a MATLAB-like interface. Matplotlib is designed to be as usable as MATLAB, with the ability to use Python, and the advantage of being free and open-source. Fig.5.2 shows few examples of graph from Matplotlib.

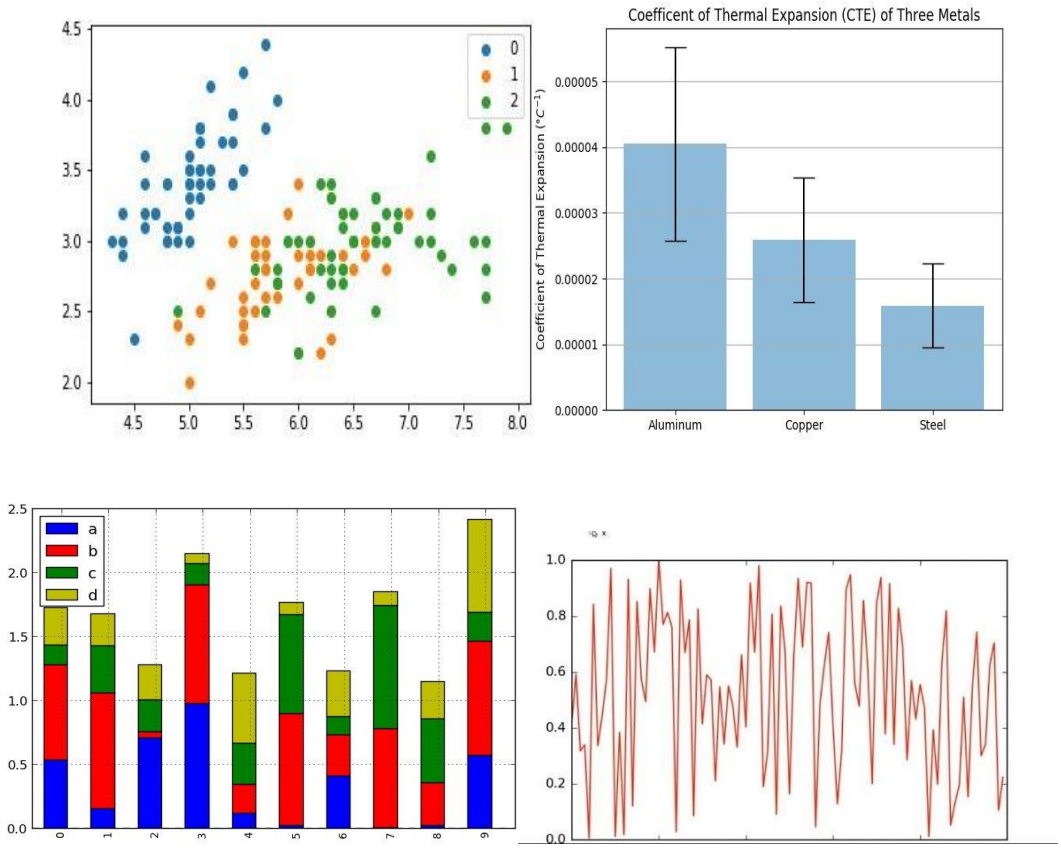

Fig.5.2: Few Graphs of Matplotlib

5.3.4 Scikit-Learn

Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Scikit-learn is largely written in Python, with some core algorithms written in Cython to achieve performance. Support vector machines are implemented by a Cython wrapper around LIBSVM; logistic regression and linear support vector machines by a similar wrapper around LIBLINEAR.

5.4 Implementation steps

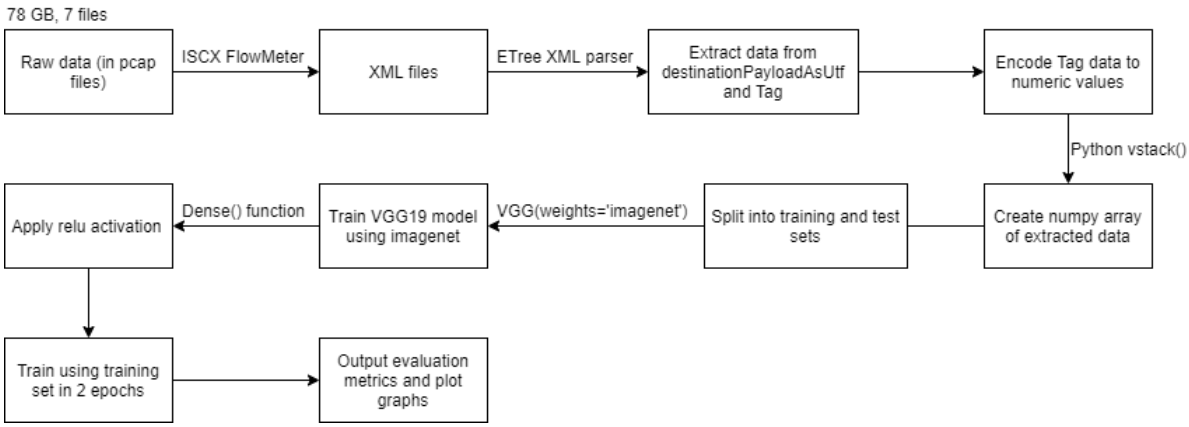

Fig 5.3: Flowchart for the implementation process

Fig 5.3 shows the steps taken to implement a solution for the stated problem in a flowchart manner.

5.4.1 Converting PCAP files to XML

ISCXFlowMeter is a network traffic flow generator available from here . It can be used to generate bidirectional flows, where the first packet determines the forward (source to destination) and backward (destination to source) directions, hence the statistical time-related features can be calculated separately in the forward and backward directions. Additional functionalities include, selecting features from the list of existing features, adding new features, and controlling the duration of flow timeout.

5.4.2 Converting XML files to numpy arrays

The .pcap file extension is mainly used for analyzing networks. pcap files are data files created using a program and they contain the packet data of a network. These files are mainly used in analyzing the network characteristics of a certain data.

5.4.3 Reading all XML files

We import the directory where all the xml files are stored, as shown in figure 5.4. Then we make an array called the data array which will contain the binary numpy arrays of the xml files. We convert the xml files to numpy arrays to make the data understandable for all machine learning models.

```
import_directory = '/home/ayush/intrusionDet/labeled_flows_xml/'

files = os.listdir(import_directory)

errors = []

start_time = time.time()
i = -1
data_array = np.empty((0, 2))
counter = 0
actual = (50**2) * 3
for file in files:
    print(file)
    try:
        tree = ET.parse(import_directory + file)
        print('Reading File ', file)
        root = tree.getroot()
    except:
        errors += file
        continue
```

Fig 5.4: Data pre-processing (Reading the xml file)

5.4.4 Taking the two important tags from xml file and converting it into numpy arrays (framing the data set)

Then we parse the xml files and pick up the two important tags that are destinationPayloadAsUTF and Tag ‘tag’. If the destinationPayloadAsUTF then the program checks if the tag is normal or not. If the tag is normal then 0 is assigned to that

instance otherwise if it is an anomaly then 1 is assigned to the instance and the result is stored in the numpy arrays and hence array data is binary.

```
for child in root:
    for next_child in child:
        if next_child.tag == 'destinationPayloadAsUTF':
            if next_child.text is not None:
                x = next_child.text
                if len(x) > actual:
                    x = x[: actual]
                else:
                    while len(x) < actual:
                        x += x
                    x = x[:actual]
                if child.find('Tag').text == 'Normal':
                    data_array = np.vstack((data_array, np.array([np.fromstring(x, dtype=np.uint8), 0]))
                else:
                    data_array = np.vstack((data_array, np.array([np.fromstring(x, dtype=np.uint8), 1]))
                counter += 1
print('Time taken: {}'.format(time.time() - start_time))
start_time = time.time()
np.save('Database2\destinationPayload_' + file, np.array(data_array))
data_array = np.empty((0, 2))
```

Fig 5.5: Extracting the required elements

As we can see in Figure 5.5, the ETree library makes a tree structure for the tags, which can be parsed using child nodes and the required data can be extracted. The vstack() function is sued to generate data array.

5.4.5 VGG19 model implementation

```
import numpy as np
from keras.applications.vgg19 import VGG19
from keras.applications.vgg16 import VGG16
import keras
from keras.layers import Input, Flatten, Dense
from keras.models import Model
import sys
import os

import matplotlib.pyplot as plt
import matplotlib as mpl


def escape():
    sys.exit()
```

```
C:\Users\vardaan\Anaconda3\envs\tensorflow-gpu-keras\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of the second
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

Fig 5.6: Importing necessary libraries

As shown in figure 5.6, Activating the environment where tensorflow has been setup and importing the necessary libraries.

```
3  #np.random.shuffle(separated)
   np.random.shuffle(data_array)
   print("ok")

   ok

4  img_row = 50
   img_col = 50

   N = np.shape(data_array)[0]
   train_test_split_percentage = 0.75

   X_train = data_array[:int(N * train_test_split_percentage), 0]
   X_test = data_array[int(N * train_test_split_percentage):, 0]

   X_train = np.array([x.reshape(img_row, img_col, 3) for x in X_train])
   X_test = np.array([x.reshape(img_row, img_col, 3) for x in X_test])

   No output
```

Fig 5.7: Train - Test split

As shown in figure 5.7, Using 3/4th of the data for creating the training set and using the remaining 1/4th for testing.

5.4.6 Deep Learning to detect intrusion in highly skewed data

```
5  y_train = data_array[:int(N * train_test_split_percentage), 1]
   y_test = data_array[int(N * train_test_split_percentage):, 1]

   y_train = np.array([[x] for x in y_train])
   y_test = np.array([[x] for x in y_test])

   No output
```

Fig 5.8: Training the model

In figure 5.8, train and test sets are generated.

```
6  model_vgg19_conv = VGG19(include_top=False, weights='imagenet', input_shape=(img_col, img_row, 3))

   WARNING:tensorflow:From C:\Users\vardaan\Anaconda3\envs\tensorflow-gpu-keras\lib\site-packages\ten
   Instructions for updating:
   Colocations handled automatically by placer.
```

Fig 5.9: Giving parameters to model

Using VGG19 with weights obtained after training the algorithm on ‘imagenet’ dataset (for transfer learning on IDS dataset) as shown in Figure 5.9.

```
for layer in model_vgg19_conv.layers:
    layer.trainable = False
    print(layer.name)
```

```
input_1
block1_conv1
block1_conv2
block1_pool
block2_conv1
block2_conv2
block2_pool
block3_conv1
block3_conv2
block3_conv3
block3_conv4
block3_pool
block4_conv1
block4_conv2
block4_conv3
block4_conv4
block4_pool
block5_conv1
block5_conv2
block5_conv3
block5_conv4
block5_pool
```

Fig 5.10: The structure of VGG19.

Figure 5.10 shows the layers of VGG 19 model. The network is 19 layers deep and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224.

```
x = model_vgg19_conv.output
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
x = Dense(1, activation='sigmoid', name='predictions')(x)

my_model = Model(input=model_vgg19_conv.input, output=x)
my_model.summary()
```

Fig 5.11: Adding layers for output

- Flatten - Flattens the input. Does not affect the batch size. To convert a multidimensional tensor into a 1-D tensor, we use Flatten.

- Dense - Dense implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bias is True).

- Activation - Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron. Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases. A neural network without an activation function is essentially just a linear regression model.

- The rectified linear activation function is a piecewise linear function that will output the input directly if is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance. The sigmoid and hyperbolic tangent activation functions cannot be used in networks with many layers due to the vanishing gradient problem. The rectified linear activation function overcomes the vanishing gradient problem, allowing models to learn faster and perform better. The rectified linear activation is the default activation when developing multilayer Perceptron and convolutional neural networks.

The types of layers in VGG19 model are Conv2D and MaxPooling2D. The network is 19 layers deep and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224.

The VGG() class takes a few arguments that may only interest you if you are looking to use the model in your own project, e.g. for transfer learning.

Figure 5.12 shows details for the trainable 19 layers of VGG19 model. It shows the number of params for each layer and the output shape of the corresponding layers.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 50, 50, 3)	0
block1_conv1 (Conv2D)	(None, 50, 50, 64)	1792
block1_conv2 (Conv2D)	(None, 50, 50, 64)	36928
block1_pool (MaxPooling2D)	(None, 25, 25, 64)	0
block2_conv1 (Conv2D)	(None, 25, 25, 128)	73856
block2_conv2 (Conv2D)	(None, 25, 25, 128)	147584
block2_pool (MaxPooling2D)	(None, 12, 12, 128)	0
block3_conv1 (Conv2D)	(None, 12, 12, 256)	295168
block3_conv2 (Conv2D)	(None, 12, 12, 256)	590080
block3_conv3 (Conv2D)	(None, 12, 12, 256)	590080
block3_conv4 (Conv2D)	(None, 12, 12, 256)	590080
block3_pool (MaxPooling2D)	(None, 6, 6, 256)	0
block4_conv1 (Conv2D)	(None, 6, 6, 512)	1180160
block4_conv2 (Conv2D)	(None, 6, 6, 512)	2359808
block4_conv3 (Conv2D)	(None, 6, 6, 512)	2359808
block4_conv4 (Conv2D)	(None, 6, 6, 512)	2359808
block4_pool (MaxPooling2D)	(None, 3, 3, 512)	0
block5_conv1 (Conv2D)	(None, 3, 3, 512)	2359808
block5_conv2 (Conv2D)	(None, 3, 3, 512)	2359808
block5_conv3 (Conv2D)	(None, 3, 3, 512)	2359808
block5_conv4 (Conv2D)	(None, 3, 3, 512)	2359808

Fig 5.12: VGG-19 (Layer details)

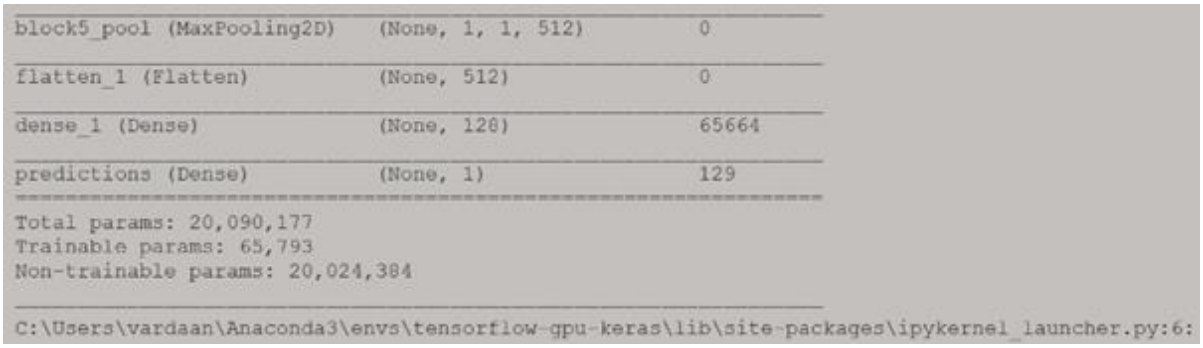

Fig 5.13: Model summary

Figure 5.13 gives the model summary: total params, trainable params and non-trainable params.

```
10 my_model.compile(loss=keras.losses.binary_crossentropy,
                    optimizer=keras.optimizers.RMSprop(),
                    metrics=['accuracy'])

   batch_size = 32
   epochs = 2

   No output
```

Fig 5.14: Compiling the model

- Optimizer - Every time a neural network finishes passing a batch through the network and generating prediction results, it must decide how to use the difference between the results it got and the values it knows to be true to adjust the weights on the nodes so that the network steps towards a solution. The algorithm that determines that step is known as the optimization algorithm

- Loss - A loss function (or objective function, or optimization score function) is one of the two parameters required to compile a model.
    - binary_crossentropy: binary_cross_entropy is also stated as negative_log_loss function. Binary cross entropy is just a special case of categorical cross entropy. With binary cross entropy, you can only classify two classes (in the aforementioned case - 0 and 1).

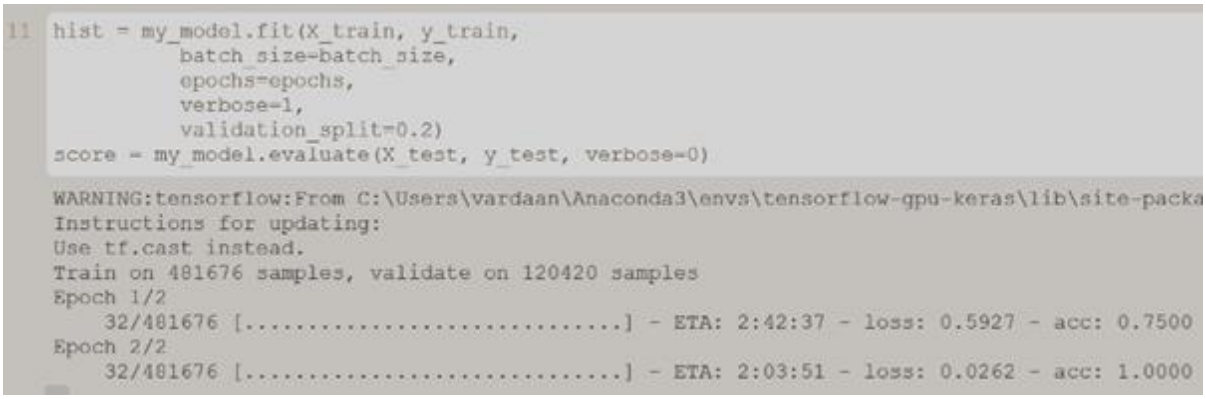

Fig 5.15: Evaluation

- batch_size: Integer or None. Number of samples per gradient update. If unspecified, batch_size will default to 32. It requires less memory. Since you train the network using fewer samples, the overall training procedure requires less memory. That's especially important if you are not able to fit the whole dataset in your machine's memory.

- epoch: One epoch is when an entire dataset is passed forward and backward through the neural network only once. Since one epoch is too big to feed to the computer at once, we divide it in several smaller batches.

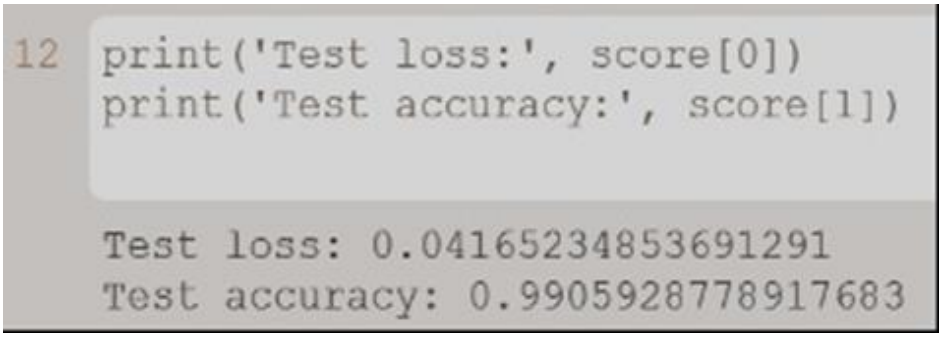

Fig 5.16: Results

Figure 5.16 shows the accuracy generated by the trained model. The accuracy comes out to be 0.9906 with a test loss of 0.0416. The model performs extremely well for the given input data.

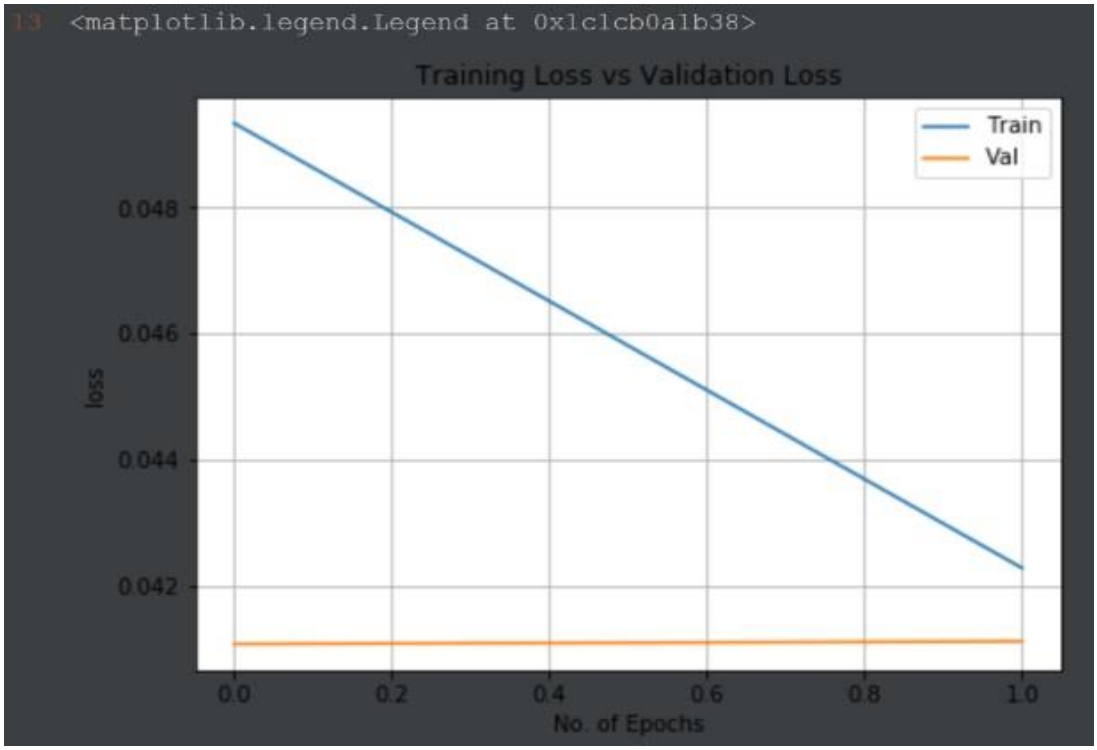

Fig 5.17: Training loss vs No. of Epochs

From Figure 5.17, we can see that the training loss decreases as number of epochs are increased.

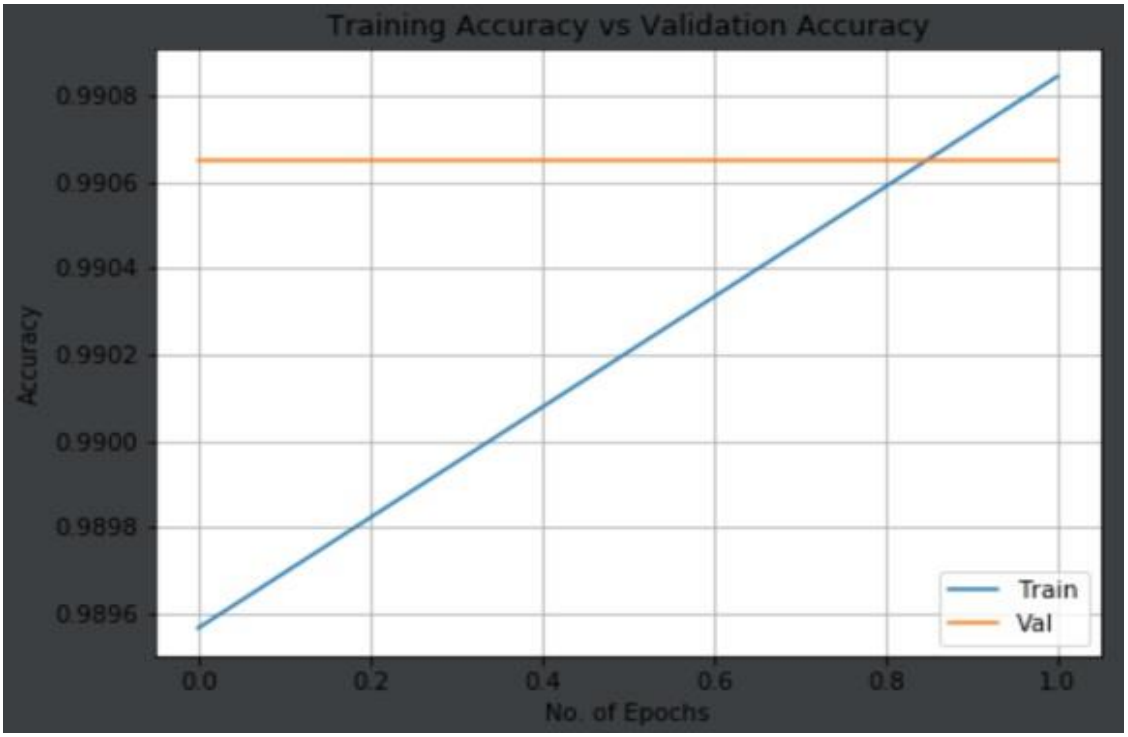

Fig 5.18: Training accuracy vs No. of Epochs

Figure 5.18 shows that the model accuracy increases as number of epochs are increased to train the model.

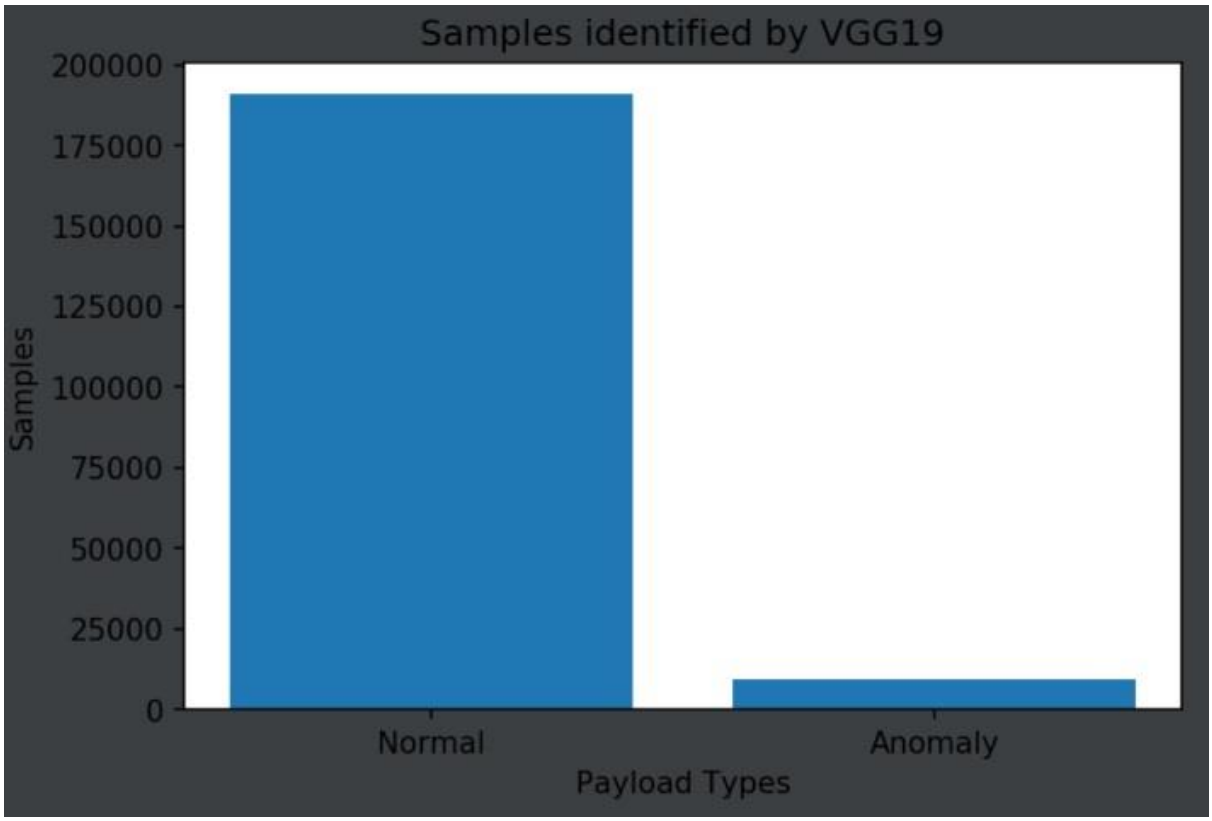

Fig 5.19: Skewness in data

Figure 5.19 shows the skewness of data before being processed by the model. The data was highly skewed and anomalies were present in a minor percentage.

```
21 import sklearn.metrics
   target_names = ['normal','anomaly']
   print(sklearn.metrics.classification_report(y_test,rounded,labels = [0,1], target_names=target_names))
```

	precision	recall	f1-score	support
normal	1.00	0.99	1.00	191182
anomaly	0.85	0.97	0.91	9517
avg / total	0.99	0.99	0.99	200699

Fig 5.20: Classification report

The metrics are defined in terms of true and false positives, and true and false negatives. Positive and negative in this case are generic names for the classes of a binary classification problem.

CHAPTER 6

CONCLUSION

This paper presented an approach for dealing with the class-imbalance problem that consisted of combining different expressions of re-sampling-based classifiers in an informed fashion. In particular, our system was built so as to bias the classifiers towards the positive set in order to counteract the negative bias typically developed by classifiers facing a higher proportion of negative than positive examples. The positive bias we included was carefully regulated by an elimination strategy designed to prevent unreliable attributes to participate in the process. The technique was shown to be effective on ISCX IDS 2012 dataset as compared to a single classifier or another general-purpose method like Decision Trees.

We thoroughly interrogated the data at the outset to gain insight into which features could be discarded and those which could be valuably engineered. To deal with the large skew in the data, we chose an appropriate metric and used an ML algorithm based on deep learning which works best with strongly imbalanced classes. The method we used should therefore be broadly applicable to a range of such problems.

With successful identifications on our model, we can confidently proclaim that our model can be used as a back end engine to an intrusion detection system application that can be mounted on the border of any computer network

CHAPTER 7

REFERENCES

[1] Vijayarani, D.S., & Sylviaa, M.M. (2015). INTRUSION DETECTION SYSTEM – A STUDY.

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

[3] Biswas, S. K. (2018). Intrusion Detection Using Machine Learning: A Comparison Study. International Journal of Pure and Applied Mathematics, 118(19), 101-114.

[4] Estabrooks, A., Jo, T., & Japkowicz, N. (2004). A multiple resampling method for learning from imbalanced data sets. Computational intelligence, 20(1), 18-36.

[5] Canziani, A., Paszke, A., & Culurciello, E. (2016). An analysis of deep neural network models for practical applications. arXiv preprint arXiv:1605.07678.

[6] Javaid, A., Niyaz, Q., Sun, W., & Alam, M. (2016, May). A deep learning approach for network intrusion detection system. In Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS) (pp. 21-26). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[7] Phua, C., Lee, V., Smith, K., & Gayler, R. (2010). A comprehensive survey of data mining-based fraud detection research. arXiv preprint arXiv:1009.6119.

[8] Shiravi, A., Shiravi, H., Tavallaee, M., & Ghorbani, A. A. (2012). Toward developing a systematic approach to generate benchmark datasets for intrusion detection. computers & security, 31(3), 357-374.