Intrusion detection using artificial neural networks and supervised deep learning

JH Smulders

orcid.org/0000-0002-8234-0767

Dissertation submitted in partial fulfilment of the requirements for the degree *Master of Science in Computer Science* at the North-West University

Supervisor: Prof JV du Toit

Graduation May 2022

24971936

ACKNOWLEDGEMENTS

Dedicated to my parents, Frits and Ansie. Dad, you are the wisest man I know and the person I look up to every day. Not only did you instil in me the importance of hard work and discipline, but you have also shown me the value of care and selflessness. Thank you for all the financial and emotional support and guidance that you provided. Mom, you are the kindest, most compassionate person in the world. I owe the completion of this degree to you. Thank you for all the late-night emotional phone calls and for all the care packages sent to see me through the countless nights and weekends spent working on my research. Words cannot begin to express how much I love and admire both of you.

A special thank you to my supervisor, Prof Tiny du Toit, for his guidance through each stage of the process. Prof, thank you for devoting so many of your evenings and weekends to my research and for all your contributions and the encouragement to see it to the end. I am eternally grateful.

ABSTRACT

One of the most challenging problems facing researchers and security experts today is protecting digital assets from the rising number of threats emerging from large networks, such as the Internet daily. Over the last decade, the frequency of cyber-attacks and the number of cybersecurity threats have risen exponentially, fuelled by rapid changing technology and a growing reliance on the Internet and information systems by society as a whole. With the ever-expanding network of data in this digital age, the importance of cybersecurity cannot be overstated. Unfortunately, however, complexities in securing information systems and computer networks against mutating threats and intrusions have given rise to a trend of detection rather than prevention.

Access control, encryption and firewalls have traditionally been recommended as the first line of defence against network and system intrusions. However, given enough time and resources, even the most secure system or network may be breached by an intruder. As a result, intrusion detection systems have been proposed as a second line of defence against cyber-attacks. Intrusion detection systems, which system administrators typically use to detect security breaches in an organisation's network, could be automated and enhanced, using an artificial neural network.

The main purpose of this study is to determine how a best-first search architecture optimisation algorithm could be designed and implemented to automate the construction of an accurate symmetrical autoencoder in an autoencoder-based intrusion detection model. Two autoencoder-based models were developed and trained to achieve this, using a reputable intrusion detection dataset. The initial model, which was developed manually, served as a performance baseline. The second model, which was developed by invoking the best-first search architecture optimisation algorithm, was evaluated against the baseline and other models from relevant literature.

Results from the study suggest that the proposed and developed algorithm can produce symmetrical autoencoder-based intrusion detection models with accuracies comparable to, and in some cases, better than models found in the literature. Furthermore, these results indicate that the best-first search optimisation algorithm may be suitable for automating the construction of an accurate symmetrical autoencoder in an autoencoder-based intrusion detection model.

Keywords: autoencoder, best-first search, classification, cybersecurity, deep learning, intrusion detection, multilayer perceptron, neural network

OPSOMMING

Een van die grootste uitdagings waarmee navorsers en sekuriteitskenners deesdae te doen kry, is die daaglikse beskerming van digitale bates teen die aanslae voortspruitend uit groot netwerke, soos die Internet. Die frekwensie van kubermisdaad en die aantal kubersekureitbedreigings gedurende die vorige dekade het eksponensieel toegeneem. Vinnig veranderde tegnologie en ’n groter afhanklikheid van die Internet en informasiestelsels dra hiertoe by. Met die kontinue uitbreiding van die netwerk van data in hierdie digitale tydperk, kan die belangrikheid van kubersekuriteit nie oorbeklemtoon word nie. Die kompleksiteit van die beskerming van sekuriteitsinformasiestelsels en rekenaarnetwerke teen indringing met die potensiaal van verandering, het aanleiding gegee tot ’n benadering van opsporing, eerder as voorkoming.

Toegangsbeheer, enkripsie en netwerksekuriteitstelsels word tradisioneel aangewend as die eerste verdedigingslinie teen netwerk- en stelselindringing. Met genoegsame tyd en hulpbronne kan dit egter moontlik wees dat die ingewikkeldste sekuriteitstelsel, binne-gedring kan word. Gevolglik kan indringingsopsporingstelsels, waardeur stelseladministrateurs sekuriteitsindringing in ’n maatskappynetwerk waarneem geoutomatiseer en bevorder word deur middel van ’n kunsmatige neurale netwerk.

Die hoofdoel van hierdie studie is om te bepaal hoe ’n beste-eerste soekargitektuur optimiseringsalgoritme ontwerp en geïmplementeer kan word om sodoende die konstruksie van ’n akkurate simmetriese outo-enkodeerder in ’n outo-enkodeerdergebaseerde indringings-opsporingsmodel te outomatiseer. Om dit te verwesenlik, is twee outo-enkodeerder-gebaseerde modelle geskep en geleer deur middel van ’n betroubare indringingsopsporings-datastel. Die aanvanklike handgekonstrueerde model het gedien as ’n prestasiebasislyn. Die tweede model, wat saamgestel is deur die oproep van die beste-eerste soekargitektuuroptimiseringsalgoritme is geëvalueer teen die basislyn en ander modelle van toepaslike literatuur.

Resultate van die studie dui daarop dat die voorgestelde en ontwikkelde algoritme, simmetriese outo-enkodeerdergebaseerde indringingsdeteksiemodelle met akkuraatheid vergelykbaar met en in sommige gevalle beter as modelle in die literatuur kan lewer. Die resultate dui verder daarop dat die beste-eerste soekoptimiseringsalgoritme toepaslik mag wees vir die outomatisering van die samestelling van ’n akkurate simmetriese outo-enkodeerdergebaseerde indringings-opsporingsmodel.

Sleutelwoorde: beste-eerste soek, diep leer, indringingsopsporing, klassifisering, kubersekuriteit, neurale netwerk, outo-enkodeerder, veelvlakkige perseptron

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

CHAPTER 1 INTRODUCTION

The Internet is typically viewed as a secure platform for sharing information, doing commerce, and controlling the physical world (De Bruijn & Janssen, 2017). However, cyber-attacks, intrusions and security breaches happen all the time, and organisations need to be better equipped to deal with them (Arora *et al.*, 2006). Furthermore, information technologies are increasingly incorporated into devices and networks, resulting in information systems and physical infrastructure becoming increasingly intertwined (Ten *et al.*, 2008). Therefore, cybersecurity is critical in our digital age, and it is growing increasingly relevant as we become more reliant on information technology in many parts of our lives. Failing to implement basic security measures to protect information assets can expose organisations to cybersecurity breaches (Coburn *et al.*, 2018).

Cybersecurity breaches can range from having little or no impact to causing a distributed denial-of-service attack, data theft, data manipulation, identity theft, or even gaining control of physical systems and causing bodily harm (De Bruijn & Janssen, 2017). With the high volumes of data passing through networks, such as the Internet daily, provisioning and maintaining an effective and up-to-date network intrusion detection system is one of the significant challenges facing network security experts (Shone *et al.*, 2018). Furthermore, the phenomena known as Big Data and IoT further exacerbate this, resulting in security experts being unable to distinguish malicious traffic from normal network traffic. Similarly, the number of sophisticated and custom-tailored attacks on Internet-connected systems has grown tremendously (Sood & Enbody, 2012).

The COVID-19 pandemic worsened the situation, with manufacturing, healthcare and food and beverage industries all seeing a substantial rise in both the quantity and sophistication of network intrusion activity during the first six months of 2020, compared to the same period in 2019 (Crowdstrike, 2020). The complexity of these intrusions can vary widely, depending on the existing security controls put in place by the organisation.

Traditionally, preventative mechanisms, such as access control, encryption and firewalls have been proposed in the literature as the first line of defence against network and system intrusions (Subba *et al.*, 2016). However, while these mechanisms improve network and system security, they are not guaranteed to rule out intrusion threats completely. This can be demonstrated by the numerous security breaches involving well-known companies, such as Adobe, Equifax, Facebook, Marriott and Yahoo (Daswani & Elbayadi, 2021). Therefore, intrusion detection systems, which can be a complementary second line of defence for securing a system and network, have been proposed to support other cybersecurity measures (Barry & Chan, 2010).

Intrusion detection systems monitor the network for unusual activity to determine whether or not it has been compromised. System administrators often employ network intrusion detection systems to appropriately identify security breaches in an organisation's network (Javaid *et al.*, 2016). However, due to the exponential rate at which everyday data increases, so too are the challenges and complexities of distinguishing between normal and malicious traffic. This challenge can be overcome by applying deep learning techniques to handle the drastic growth in the volume and diversity of data. Subsequently, some of the challenges experienced when differentiating between normal and malicious network traffic are combatted by recognising complex patterns and discovering useful relationships in data (Shone *et al.*, 2018). Therefore, the adoption of artificial intelligence may be a viable approach that may be advantageous to improving and automating network intrusion detection.

An artificial neural network, also called a neural network, is a system of interconnected processing units modelled after the human brain to identify and classify network activity, based on incomplete or limited input sources (Barry & Chan, 2010). According to KS and Ramakrishna (2013), an artificial neural network can be trained to detect intrusions with a high degree of efficiency and accuracy. Furthermore, the high-speed processing afforded by artificial neural networks connected in parallel, not to mention their scalability, highlights the potential for using artificial neural networks to solve complex problems (Izeboudjen *et al.*, 2014). These factors motivate the adoption of neural networks, giving ample reason why they may be ideal to be implemented for the classification problem in this study. In the literature, various artificial neural network architectures have been proposed and applied to the domain of intrusion detection (Drewek-Ossowicka *et al.*, 2021). The autoencoder, one such architecture, has been recommended for its simple implementation and good performance (Javaid *et al.*, 2016).

An autoencoder is a three-layer (input, hidden and output) neural network that consists of two components: an encoder and a decoder (Farahnakian & Heikkonen, 2018). The encoder reduces the size of the input data to a low-dimensional format. It performs feature extraction between the input and hidden layers, while the decoder reconstructs the encoded input data between the hidden and output layers (Wang *et al.*, 2020b). Traditional autoencoders, also known as symmetrical autoencoders, have equal layers or weights in both the encoder and decoder components of the artificial neural network (Qiu & Dai, 2019). Autoencoders are considered unsupervised neural network models, since they are designed to reconstruct their inputs through feature learning (Farahnakian & Heikkonen, 2018). However, supervised learning techniques can also be applied to autoencoder-based neural network models for fine-tuning. These models may even be stacked with other models to achieve a more desirable outcome (Wang *et al.*, 2020b). Selecting the correct number of hidden layers and neurons and defining the values for its

hyperparameters are essential when creating a neural network model (Ma & Khorasani, 2003). Hyperparameters directly control the training algorithm's behaviour and critically impact the model's performance (Van Rijn & Hutter, 2018). They can significantly alter a neural network model's training time, efficiency, convergence, and accuracy (Li & Yang, 2020). Therefore, a way of automating this process by implementing a best-first search hyperparameter optimisation algorithm will be investigated.

The best-first search algorithm, a variant of the graph search algorithm, is a widely used strategy for reducing search time by leveraging heuristic information (Dechter & Pearl, 1985). A node is selected for expansion based on an evaluation function, where the node with the most favourable evaluation metric is traversed first during every iteration (Russell & Norvig, 2016). By continually expanding the best node first, based on knowledge imparted by the heuristic function, the best-first search algorithm can be used to achieve an approximate solution to a search problem rapidly, even when the time available to solve a search problem is limited or undefined (Zhang *et al.*, 2021). Since the best-first search algorithm can quickly arrive at an optimal solution by utilising knowledge acquired through heuristics (Russell & Norvig, 2016), and being based on a priority queue system (Pathak *et al.*, 2018), it will be selected as the basis for the autoencoder architecture optimisation algorithm that will be designed and implemented in this study.

In this research project, the challenge of more accurately and reliably distinguishing between normal and malicious traffic in a computer network is investigated. Specifically, the feasibility of designing and implementing a best-first search algorithm to automate the construction of an accurate symmetrical autoencoder in an autoencoder-based neural network model to perform intrusion detection and applying supervised deep learning techniques to overcome the challenges mentioned above will be explored.

The remainder of this chapter is structured as follows: In Section 1.1, the problem statement is provided and the research question is formulated. The primary aim, followed by the secondary objectives of the study, is outlined in Section 1.2. A brief discussion on the research design, specifically the research paradigm and methodology applied throughout this study, are presented in Section 1.3. Details of the dissertation outline are given in Section 1.4. Finally, in Section 1.5, the context of the study is summarised.

1.1 Problem statement

A variety of security vulnerabilities on the Internet and computer systems has emerged due to the increased use of computer networks (Anwar *et al.*, 2017). Network intrusion detection systems, which are designed to combat these vulnerabilities, have to process and analyse vast amounts

of network traffic (Shone *et al.*, 2018). This inevitably leads to instances where network intrusions are missed or incorrectly classified (Inayat *et al.*, 2016).

This study will empirically model intrusion detection, using a deep artificial neural network to address the problem facing modern security experts of more accurately and reliably classifying network traffic. Furthermore, a best-first search technique will be proposed to automate the construction of an accurate autoencoder which forms part of an intrusion detection system. Consequently, the following research question will be investigated in this study: “How can a best-first search neural network architecture optimisation algorithm be designed to automatically build an accurate symmetrical autoencoder which forms part of an intrusion detection system?”

1.2 Research objectives

The primary aim of this study is to determine how a best-first search algorithm can be designed to automate the construction of an accurate symmetrical autoencoder model, used as part of an intrusion detection system, to classify network traffic as either normal or malicious.

The following secondary objectives have been set to facilitate the achievement of the primary aim:

1. Perform a literature review on:
    (a) traditional computer and network threats and intrusion defence approaches;
    (b) intrusion detection systems, specifically in the context of information systems; and
    (c) deep learning neural network and autoencoder architectures and training techniques.
2. Design and implement a best-first search architecture optimisation algorithm to automate the construction of an accurate symmetrical autoencoder model, used as part of an intrusion detection system;
3. Obtain a representative intrusion detection dataset;
4. Pre-process the dataset and split it into training and testing data for the two symmetrical autoencoder-based neural network intrusion detection models developed in this study;
5. Manually construct and train a suitable symmetrical autoencoder architecture combined with a multilayer perceptron classifier as a baseline intrusion detection model on the dataset prepared in Step 4 to classify network traffic as either normal or malicious;

6. By using the algorithm developed in Step 2, determine an accurate symmetrical autoencoder architecture and train the model combined with a multilayer perceptron classifier on the dataset prepared in Step 4 to classify network traffic as either normal or malicious; and

7. Evaluate the performance of the intrusion detection autoencoder-based neural network model constructed in Step 6 against the performance of the intrusion detection autoencoder-based neural network baseline model constructed in Step 5.

1.3 Research methodology

Due to the rapid rate of technological changes and the necessity for up-to-date knowledge, research has become more vital and increasingly crucial in the modern world. *"Research is a process through which new knowledge is discovered"* (Salkind, 2008:3). It can be described as the process of scientific investigation to satisfy one or more of the following outcomes (Kothari, 2004):

- Become acquainted with a phenomenon or acquire new knowledge about it;
- design of new or improved solutions to scientific or non-scientific problems;
- to determine how often something happens or how often it is associated with something else; and
- to test a hypothesis.

According to Maree (2007), research is conducted in line with a specific research paradigm. A paradigm refers to a set of shared assumptions (Oates, 2006). Similarly, Bernard and Bernard (2000) defined a paradigm as a framework for theory and research that includes basic assumptions, techniques and methods for seeking answers. In other words, a paradigm is a way to think about something collectively. Burrell and Morgan (1979) outline four paradigms or ways of approaching research: radical humanist, also known as critical theory; radical structuralist, also known as the positivist paradigm; functionalist, also known as the realist paradigm; and interpretive, also known as the constructivism paradigm. The objectives of this research stem from the positivist paradigm. Therefore, only the positivist research paradigm will be briefly discussed.

In the positivist paradigm, the object of the study is independent of researchers and science is seen as a way of getting to the truth (Krauss, 2005). Mack (2010) agrees, claiming that scientific

methods are used by researchers, who are unbiased, independent observers, to make predictions in the positivist paradigm. This statement infers that the positivist paradigm involves generating a hypothesis, investigating said hypothesis, and finally, documenting any findings. *"The positivist paradigm asserts that real events can be observed and explained with logical analysis"* (Kaboub, 2008:343). Results from such logical analysis (or experimentation) are unbiased and do not change when observed (Healy & Perry, 2000). In other words, in the positivist paradigm, results are obtained in an objective, reliable manner (Bernard & Bernard, 2000). The work in this study is not based on a hypothesis, but rather on empirical research and experimentation following an extensive review of the literature. Therefore, it is still considered to be conducted using the positivist paradigm (Bharadwaj, 1996). The experimentation follows the design of a best-first search algorithm, developed to automate the building of an accurate symmetrical autoencoder used in an autoencoder-based neural network architecture to model intrusion detection. Since this algorithm takes the shape of an artefact, the design science research methodology will be followed.

According to Hevner and Chatterjee (2010), design science is a set of analytical techniques and perspectives used to study information systems. Design science researchers create simple or innovative artefacts and analyse their performance or use such artefacts, along with abstraction and reflection, to create new knowledge. The term artefact is used to denote something artificial constructed by humans instead of something that occurs naturally (Simon, 1996). Therefore, design science research needs to create a viable artefact in the form of a construct, a model or an algorithm (Hevner & Chatterjee, 2010).

Peffers *et al.* (2006) outline the following conceptual framework for conducting design science research:

1. Problem identification and motivation: A specific research problem is defined and the value of the proposed solution is justified;

2. Objectives of a solution: The objectives through which the identified problem will be addressed are deduced;

3. Design and development: The proposed solution to the problem is developed after determining the required functionality and architecture needed for an artefact;

4. Demonstration: The artefact's efficacy in solving the problem is demonstrated through experimentation;

5. Evaluation: The artefact's actual behaviour is compared to its expected behaviour to measure how well it solves the problem; and

6. Communication: The overall process is communicated or documented, including the problem identified, the importance of the proposed solution, design of the artefact and its effectiveness.

This study follows an experimental design to construct a computational model for intrusion detection; therefore, the design science research approach will be applied. In this chapter, steps one, two and six of the design science research methodology were addressed by defining the research problem, motivating its resolution, setting objectives and outlining the importance and overall process to resolve the research problem.

In the next section, an outline of the remaining chapters is provided by briefly describing each chapter's purpose.

1.4 Dissertation outline

A literature review on common cybersecurity threats and the numerous approaches identified to prevent them is presented in Chapter 2, highlighting the importance of cybersecurity in society and emphasising the growing data concern.

Chapter 3 is devoted to exploring the architecture of an intrusion detection system. Furthermore, the typical application and function of intrusion detection systems will be explained according to existing literature.

In Chapter 4, the deep neural network architecture will be discussed in terms of its history, structure and training. The autoencoder architecture, which forms the basis of the deep neural network considered in this study, will be examined. Finally, the best-first search architecture optimisation algorithm used to automatically build an accurate symmetrical autoencoder model will be presented.

The focus in Chapter 5 is to present the study's experimental design, including the data pre-processing techniques used and the development of the baseline and proposed deep artificial neural network autoencoder-based intrusion detection model. The results obtained from the experimental phase will be presented, along with an evaluation and comparison of the performance of the two models. The knowledge gained from the experimentation is discussed at the end of this chapter.

In Chapter 6, the aim and objectives set out in Section 1.2 will be revisited to determine if they were achieved. Then, a summary of the contributions made in the study will be provided. Finally, possible future work will be identified and discussed.

1.5 Chapter summary

The purpose of this chapter was to provide an overview of the study by introducing the research topic, problem statement, research question and objectives. Additionally, the research methodology adopted to conduct this research was discussed. Finally, a high-level summary for each chapter was provided. The proposed approach is to design and implement a best-first search architecture optimisation algorithm to automate the construction of an accurate symmetrical autoencoder model used as part of an intrusion detection system. In the next chapter, a literature review of traditional computer and network threats and intrusion defence approaches is presented.

CHAPTER 2 CYBERSECURITY THREATS AND APPROACHES

For many years, security has been, and continues to be, a significant concern (Alani, 2021). People take their safety very seriously and generally apply best practices to protect themselves from adverse events, illnesses or harm. Physical security is used to protect a company (or individual's) premises, facilities, buildings, information and other assets (Fennelly, 2004).

Fennelly (2004) mentions three overlapping strategies for implementing physical security:

1. Access control: Controlling who can access an asset;

2. Surveillance: Monitoring said access; and

3. Territorial reinforcement: Preventing the physical breach of access controls.

According to Agrafiotis *et al.* (2018), society is dependent on technology for interaction, business and even production. Innovation has prompted considerable advances in these domains, especially using the Internet. However, this has also exposed people to a slew of digital risks; physical security alone is no longer enough.

The world today revolves around data – it can be found everywhere. Cukier and Mayer-Schoenberger (2013) suggest that the amount of digital data in the world is doubling every three years. By 2020, it was expected that each person on the planet would be generating 1.7 megabytes of data every second. According to Alani (2021), over 4.57 billion users were connected to the Internet by the year 2020, and upwards of 2.5 quintillion[1] bytes of data were being generated by devices and people every day. The popularity of services, such as Social Media, Big Data, Cloud Computing, and the Internet of Things (IoT) further adds to the staggering growth in data volume (Shone *et al.*, 2018). Furthermore, organisations have digitised many parts of their operations, due to the technological advancements made in recent years (Agrafiotis *et al*., 2018).

It all started with the birth of a renowned network known as the Internet. This phenomenon has changed how businesses operate and countries are run and even how people live (Cukier & Mayer-Schoenberger, 2013). The Internet has fundamentally reshaped how people transfer data and share information. However, some people still do not understand what it is, where it comes from, how it can be used, its inherently massive potential, and its many risks. The Internet has

[1] A quintillion bytes is equivalent to 1,000,000 terabytes.

very much become an integrated and indispensable part of everyday life, but this has come at a cost – people are becoming increasingly vulnerable to cybersecurity threats (Hong & Furnell, 2021).

The Internet, the largest computer network in existence today, is a service enabler for many technologies. The Internet reduces geographical distance by enabling access to information immediately and allowing global communication. However, the Internet is also the single most significant contributor to cybersecurity risks.

In this chapter, the second step of the design science research methodology will be addressed by conducting a systematic literature review to synthesise the objectives identified in Chapter 1. The inception of the Internet is presented in Section 2.1. This provides a better understanding of its origin and architecture. This also explains how data traverses through a computer network. The Big Data problem is introduced in Section 2.2, and the security implications thereof are also investigated. IoT is discussed in Section 2.3, along with its numerous vulnerabilities. Cybersecurity and why it is needed is further emphasised in Section 2.4. Common computer security threats are discussed in Section 2.5. Current approaches and tools used for securing information and data are summarised in Section 2.6. A summary of the chapter is presented in Section 2.7.

2.1 The Internet

According to Council (1995:1), the Internet can be defined as follows:

"Internet refers to the global information system that

- *is logically linked together by a globally unique address space based on the Internet Protocol (IP) or its subsequent extensions/follow-ons;*
- *can support communications using the Transmission Control Protocol/Internet Protocol (TCP/IP) suite or its subsequent extensions/follow-ons, and other IP-compatible protocols; and*
- *provides, uses or makes accessible, either publicly or privately, high-level services layered on the communications and related infrastructure described herein."*

In essence, the Internet can simply be described as a large network of interconnected computers worldwide. It is a network made possible using a common set of communication standards, procedures (also known as protocols) and formats, granting people the ability to share information

and communicate with each other, no matter where they are located (Kahn & Cerf, 1999). The Internet is known as the world's largest computer network (Leiner *et al.*, 2009). and can be considered as the most significant enabler of cyber threats and system intrusions. In the next section, a brief history of how the Internet came to fruition is provided.

2.1.1 Origin

According to Leiner *et al.* (2009), the Internet known today is an influential, widespread infrastructure of information, the inception of which revolves around four distinct aspects:

1. Technological evolution: This evolution began with early research on the concept of packet switching and technologies, such as ARPANET (among others), and continues into recent times, expanding on the improvement of the dimensions of scale, performance and higher-level functionality;
2. Operations and management of global network infrastructure: According to Govindan *et al.* (2016), these operations and management are continually evolving;
3. The social aspect: This has given rise to privacy concerns that, at the very least, require some technical skills and a basic understanding of how data is processed in a network environment (Dinev & Hart, 2005); and
4. Commercialisation: Initially plagued by early problems, such as acceptable use policies and lack of network funding (Weis, 2010), the Internet ultimately led to commercial products implementing Internet technology. According to Takano and Kajikawa (2019), these developments have given rise to new opportunities in the emerging technology space, such as IoT and Software as a Service (SaaS).

The history of the Internet will be discussed in the next section, specifically the inception of ARPANET and the World Wide Web.

2.1.1.1 ARPANET

In the Summer of 1962, J.C.R. Licklider of MIT envisioned a globally interconnected set of computers through which everyone could quickly access data and programs from any location (Leiner *et al.*, 2009). Licklider later became the first head of computer research at the Defense Advanced Research Projects Agency (DARPA) program. This project was referred to as the Galactic Network concept by Licklider and Clark (1962). The vision later became a reality, when in the late 1960s, two major research projects emerged from DARPA (Weis, 2010):

1. ARPANET: What started as a network of four interconnected computers built by the Advanced Research Project Agency (ARPA), later re-named DARPA, became operational

in 1969 and today serves as the Internet (Packard, 2020; Leiner *et al.*, 2009). However, early ARPANET usage was limited to DARPA-funded computer science researchers (Weis, 2010). By 1970, the Network Working Group (NWG) created the initial ARPANET host-to-host protocol, later known as the Network Control Protocol (NCP). This eventually gave rise to three of the most popular applications of today: remote computing, file transfer, and electronic mail (Leiner *et al.*, 2009). By the mid-1970s, researchers had determined that they required specific protocols that would allow heterogeneous networks to be interconnected. This was referred to as an open-architecture network (Leiner *et al.*, 2009);

2. TCP/IP: The need of supporting open-architecture networks led DARPA to create a new protocol by 1978 that would eventually be called the Transmission Control Protocol/Internet Protocol (TCP/IP) (Leiner *et al.*, 2009). The Internet Protocol (IP) facilitated the addressing and forwarding of individual packets. At the same time, Transmission Control Protocol (TCP) governed service features, such as flow control and recovery from lost packets. By the early 1980s, TCP/IP was the standard protocol suite used within ARPANET and was also officially adopted as a standard by the US Department of Defense (Kahn & Cerf, 1999).

According to Weis (2010), ARPANET usage was broadened in the 1980s to a much more comprehensive array of researchers. The funding for this evolving network was derived from three primary sources: the federal government, research laboratories, and universities. By 1985, ARPANET's Internet was already a well-established technology that supported a broad community of developers and researchers alike (Leiner *et al.*, 2009) and was also starting to be widely used for delay-tolerant applications, such as electronic mail (e-mail) and file transfer (Weis, 2010). In 1989, Tim Berners-Lee proposed that the world work towards creating a universal linked information system, one whose aim would be to consolidate critical information and references in a single database, providing a way of finding this information afterwards (Berners-Lee, 1989). This proposal for a linked information system ultimately led to creating what became known as the World Wide Web (WWW, Web or W3) (Leiner *et al.*, 2009). The World Wide Web is the enabler that allows people to share information (data) from large database repositories globally and is discussed in the next section.

2.1.1.2 World Wide Web

By 1990, the Internet, based on the TCP/IP protocol, the product of years of research and experimentation, grew exponentially (Campbell-Kelly & Garcia-Swartz, 2013). Unfortunately, the architects of the Internet, who themselves were more focused on advancements in technology than that of the user experience, were soon faced with an increasingly vast infrastructure of information that, without directions, was becoming impossible to navigate by anyone other than

an expert. It was not until 1993 that the Web, which was initially proposed as a tool for creating and reading structured documents, such as software manuals, was beginning to spread outside of the scientific community (Hayes, 1994).

According to Crystal (2001), the World Wide Web can be described as a data store of information hosted by all the computers linked to the Internet, which is mutually accessible using the standardised Hypertext Transfer Protocol (HTTP). W3C (1997:1) described the Web as "*the universe of network-accessible information, an embodiment of human knowledge".* Tim Berners-Lee called the Web a *"way of viewing all the online information available on the Internet as a seamless, browsable continuum"* (Berners-Lee *et al.*, 1993:1).

Some key features of the WWW model proposed by Tim Berners-Lee include the following (Berners-Lee *et al.*, 1992):

- Only a single copy of the information being shared would be required: This single instance could then be referenced from anywhere on the Internet;
- A continually evolving topology of information: The ability to traverse an endless realm of knowledge using links;
- Global reach: The Web stretches seamlessly, eliminating the restriction of distance from globally stored information;
- Indexes represented as documents: An accurate representation of the information being shared would be discoverable by search engines; and
- Paperless library: The documents on the Web would exist virtually and could be updated in real time.

A timeline of how the Internet unfolded is presented in Table 2-1. While the World Wide Web has dramatically added to the usability and convenience of the Internet, its underlying architecture, designed without security considerations in mind, was becoming a breeding ground for network security threats (Djambazova *et al.*, 2008). The architecture of the Internet is discussed briefly in the next section.

Table 2-1: Timeline of Internet development

Year	Event
1962	Galactic Network concept
1969	ARPANET commissioned
1970	Network Control Protocol (NCP) created
1978	Transmission Control Protocol/Internet Protocol (TCP/IP) created
1983	Cutover from NCP to TCP/IP as the official standard for the ARPANET
1989	Tim Berners-Lee proposes a global hypertext project, later to be known as the World Wide Web (WWW)
1991	HyperText Transfer Protocol (HTTP) created
1993	The World Wide Web becomes mainstream

2.1.2 Architecture

The architecture of the Internet is built around two major components:

1. Hardware; and

2. Network protocols.

These two components are explained in this section, whereafter the architecture of the Internet is further discussed.

2.1.2.1 Hardware

According to Singer and Friedman (2014), browsing websites on the Internet follows a simple path between two nodes, including breaking up packets at the source and reassembling them at the destination. Figure 2-1 outlines how computers talk to the Internet:

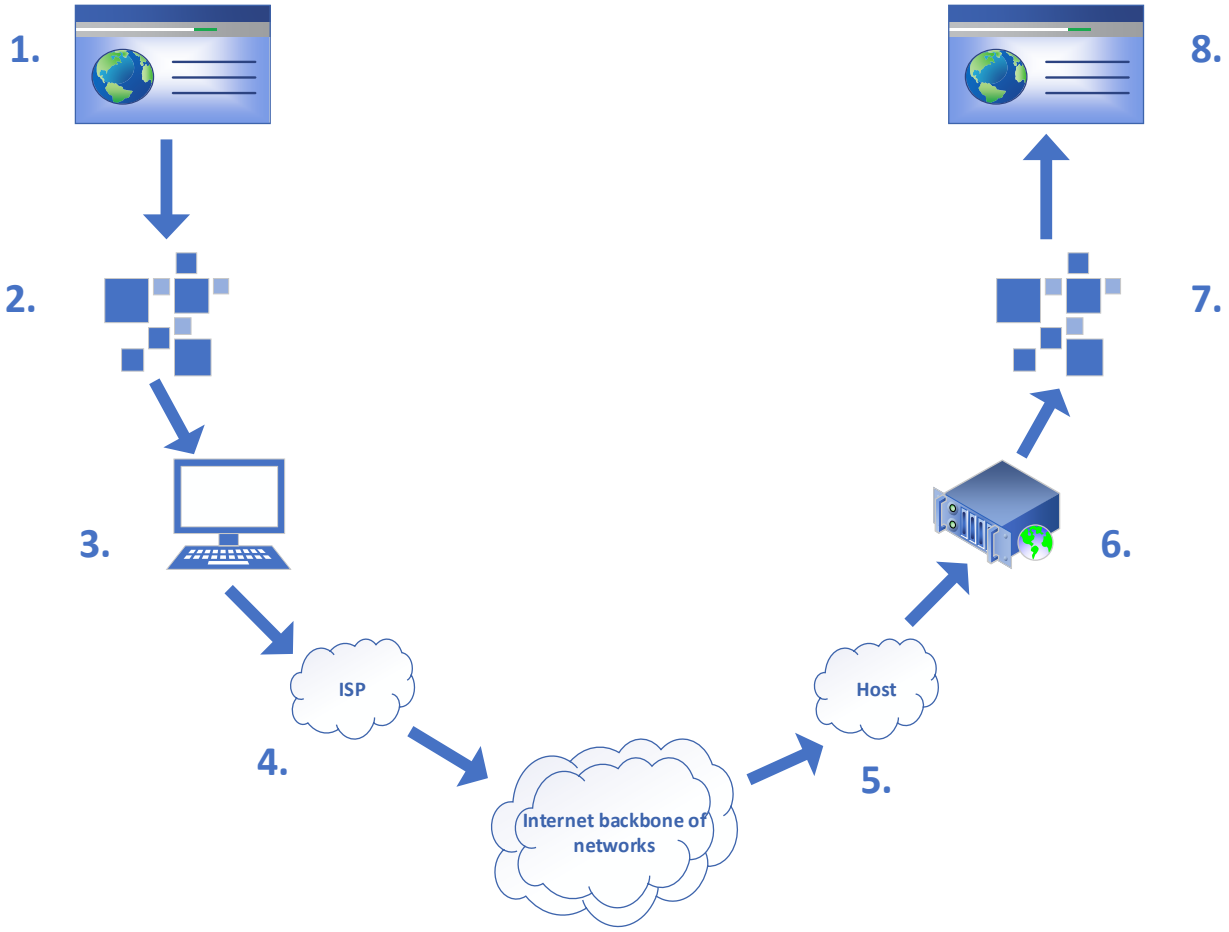


Figure 2-1: How computers talk to the Internet

The steps outlining how computers talk to the Internet are as follows:

1. The URL of the Web page is entered, using a Web browser application, which translates the HTTP protocol request;

2. The operating system breaks up the Web browser's HTTP request into multiple individually addressed IP packets;

3. A client computer (or device) sends packets to the local area network of their Internet service provider (ISP);

4. The ISP routes every individual packet to a larger Internet backbone network;

5. Traffic is then routed to the content host closer to the destination IP address;

6. The host sends traffic to their Web server;

7. The Web server reassembles the packets and interprets the HTTP request to deliver the Web content; and

8. Finally, the Web content is displayed.

The hardware that supports the network and essentially facilitates the communication between devices on the Internet includes a combination of the following (Gralla, 1998):

- Computers: Both client devices and host devices require some form of a computerised system to support the initiation and acceptance of requests for information. These systems run the software (i.e. Internet browsers) that can understand the protocols used to send information, and the servers (i.e. Web servers or storage servers) that distribute the content requested by clients;

- Switches: These are devices that facilitate the interconnection of multiple computers to share information and resources. Switches enable the creation of computer networks; and

- Routers: These devices facilitate the interconnection of multiple switches and their respective networks of devices. Unlike a network switch that simply forwards packets to the next destination, a router can send packets, using the most efficient route, based on predefined priorities. Ultimately, a router enables a network of switches to access the Internet.

The above components, which facilitate the sharing of information, are paramount to the security of data. From a security point of view, any component may have weaknesses that can result in vulnerabilities in the system. A compromised physical component can undermine all additional layers of a system's cybersecurity (Prinetto & Roascio, 2020). However, security vulnerabilities are limited to the hardware components, such as the Internet and the network protocols it needs to function. These network protocols are discussed in the next section.

2.1.2.2 Network protocols

Network protocols, such as the TCP/IP protocol suite, enable computers, smartphones, and other connected devices, supplied from many different computer vendors and running completely different software, to communicate (Fall & Stevens, 2011). This protocol presents the sets of rules

that devices must follow to complete tasks. Ultimately, without this standard collection of rules, devices would not be able to communicate.

TCP/IP protocol layering is depicted in Figure 2-2 as described by Fall and Stevens (2011).

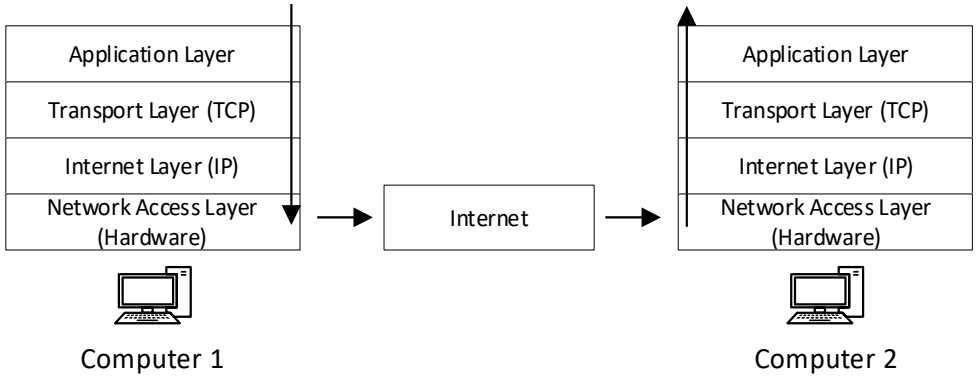


Figure 2-2: Protocol layering based on the TCP/IP suite

The traversal of traffic between two computers through the Internet in the TCP/IP suite context is explained next.

1. The message begins its transmission at the top of the protocol layer stack on computer 1 and works its way down. If the message is long, each protocol layer through which the message passes may break the message into smaller data bits. This is because data sent over the Internet (and most other computer networks) are sent in smaller bits known as packets;

2. The packets would go through the application layer and continue to the transport layer. They will be assigned a port number that acts as a unique identifier for the source and destination application, ensuring that the transferred data arrives correctly at the other end;

3. Upon moving through the transport layer, the packets make their way to the Internet layer where each packet is assigned a destination address, also known as an IP address;

4. Having been assigned a port number and an IP address, the packets are now ready to be transmitted over the network. The hardware layer transforms the packets containing the alphanumeric text of the message into electronic signals. It sends these signals through the network, where it passes through one or more devices, known as routers;

5. On the opposite end of the network, the Internet Service Provider (ISP) directly connects to the Internet. The ISP's router examines the destination address in each packet and determines where to send it;

6. The packets continue to pass through more routers until they eventually reach computer 2. Once here, the packets now start at the bottom of the destination computer's TCP/IP protocol layer stack and work their way up;

7. As the packets travel upwards through the protocol layer stack, all routing data added by the protocol layer stack of computer 1, such as IP address and the port number, are stripped from the packets; and

8. By the time the packets reach the application layer, they would have been reassembled into their original form and interpreted by computer 2.

Part of the attraction of using the Internet to transfer and share information lies in the seamless and user-friendly way transmissions are accomplished using network protocols. However, numerous vulnerabilities in these network protocols can lead to their exploitation, posing serious network security challenges. For example, network packets can be intercepted and changed before reaching their destination. These vulnerabilities have given rise to the concept known as cybersecurity, which is discussed further in Section 2.4.

By working in unison, the combination of hardware and network protocols enable the transmission of information over the Internet. However, this growing network of computers and other devices on the Internet has given rise to a new problem: large volumes of data (Gudivada *et al.*, 2015). The ever-growing data being accumulated on the Internet is discussed in the next section.

2.2 Big data

The world today revolves around data. This mass of data is a resource; it needs to be managed. Big Data refers to the ever-increasing amount of information that organisations are storing, processing and analysing (Tankard, 2012).

2.2.1 Definition

There is more than one formal definition for the term Big Data. Several authors have proposed their interpretations of the phrase Big Data throughout the years. According to Gordon (2013), the phrase Big Data is generally characterised by a combination of the five qualities listed below:

1. Volume: Where a large enough dataset which is to be stored and analysed exists, requiring special consideration;

2. Variety: Where the data is diverse, containing many types of data from multiple sources;

    (a) Structured: Data which is stored in tables or objects having well-defined metadata;

    (b) Semi-structured: Data held as documents with metadata that is contained internally; and

    (c) Unstructured: Data, such as photos, videos and similar forms of binary data.

3. Velocity: Where the data is produced at high rates and operating on older data is not valuable;

4. Value: Where the data in question is of actual importance to the company or organisation that is using it; and

5. Veracity (accuracy): Where the correctness of the data can be measured and assessed for accuracy.

According to Martin (2015), Big Data can briefly be defined as a combination of great size and highly complex data with advanced analytics. Dealing with Big Data, according to De Mauro *et al.* (2016), necessitates more storage and processing power than the average information system can provide. Big Data combines information from diverse sources in new ways to create knowledge, make better predictions or tailor services. According to Pfleeger *et al.* (2015), Big Data implies analysing large amounts of data collected from multiple sources. This large resulting dataset is collected and stored for the sole purpose of analysis to deduce one or more predictions about society.

Despite the opportunities provided by Big Data, data and information processing still pose issues owing to the characteristics of Big Data mentioned above (Zhang *et al.*, 2018). Access to all of this data raises ethical concerns, such as the need for privacy (Zwitter, 2014).

2.2.2 Security implications

Big Data security is a term used to describe any tools that guard both the data and analytics processes against attacks, theft, or other malicious activities that could harm or negatively affect them. Tallon (2013) claims that Big Data is posing a growing threat to personal privacy.

Big Data security challenges are multi-faceted. According to Pfleeger *et al.* (2015), Big Data is frequently associated with large sums of money. Collectors of Big Data may do so to sell the data to advertisers or other buyers. Threats to this model include the theft of information stored online, ransomware, or distributed denial of service (DDoS) attacks that could crash a server. The issue is further exacerbated when companies store sensitive or confidential information, such as credit card numbers or simply contact details. Attacks on an organisation's Big Data storage could result in financial losses (Cerchiello & Giudici, 2016).

Big Data security threats can be grouped into four categories (Keston, 2014):

1. Infrastructure security: Big Data infrastructures are distributed across multiple servers or mainframes over several networked topologies;

2. Data privacy: The datasets can include sensitive, private information of numerous consumers subject to privacy legislation in several countries;

3. Data management: The sheer size and often dispersed nature of Big Data makes it challenging to determine the source and history of its creation and modification; and

4. Integrity and reactive security: Data can be dangerous to an organisation if its integrity cannot be ensured. Similarly, without real-time security monitoring, underlying threats of Big Data can quickly overrun a system.

Ultimately, Big Data relates to the data that is challenging to store, manage, analyse and protect. Furthermore, the high volume, variety and velocity (speed) of data generated on a network, such as the Internet, have made detecting attacks using traditional techniques challenging (Othman *et al.*, 2018). IoT, a significant contributor to the volume of Big Data (Lirette, 2019), is discussed in the next section.

2.3 The Internet of Things (IoT)

IoT, according to Ambrosin *et al.* (2016), is a growing phenomenon that comprises billions of interconnected devices spanning the globe. Gartner predicted that there would be over 20 billion IoT devices by the year 2020, with 65% of companies wholly adopting the technology (Hung, 2017). IoT combines numerous sensors, objects and intelligent nodes that can interact without human involvement (Ambrosin *et al.*, 2016). Unfortunately, while IoT can drive innovation and simplify the interaction between people and objects, it brings new risks for data security and privacy.

2.3.1 Definition

The term IoT was first coined in 1999 by Kevin Ashton (Tripathy & Anuradha, 2017). IoT refers to the myriad of interconnected devices connected to the Internet, providing wider communication and data transmission. Several authors have defined the term IoT over the years, including the following:

- *"The Internet of Things (IoT) describes the network of physical object things that are embedded with sensors, software, and other technologies to connect and exchange data with other devices and systems over the Internet"* (Oracle, 2020:18).

- *"A dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual things have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network"* (Van Kranenburg, 2008:1).

- *"An open and comprehensive network of intelligent objects that can auto-organise, share information, data and resources, reacting and acting in the face of situations and changes in the environment"* (Madakam & Lake, 2015:164).

- *"A worldwide network of interconnected entities"* (Roman *et al.*, 2013:2266).

According to Wortmann and Flüchter (2015), IoT architecture comprises three technological layers: the device layer, the connectivity layer and the cloud layer, as presented in Figure 2-3. Each of these layers is discussed next.

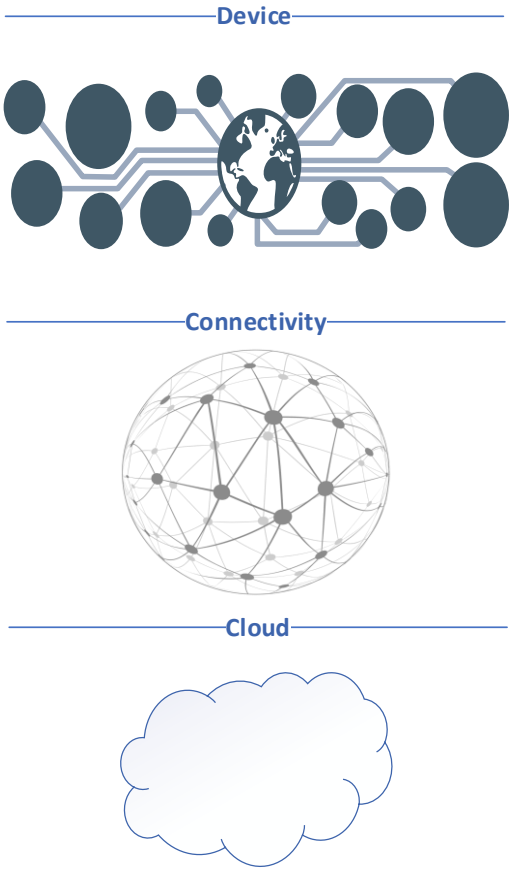


Figure 2-3: Three-tier technology stack of IoT

2.3.1.1 Device layer

The device layer, also known as the Things layer or the Hardware layer (Porter & Heppelmann, 2015), is considered the first IoT architecture level. The IoT device layer includes all the core hardware components of the smart devices connected to the system (Wortmann & Flüchter, 2015). These devices are considered smart devices because they are embedded with sensors, processors, actuators, and networking hardware to perceive their environment (Porter & Heppelmann, 2015). This layer's primary function is to collect useful information from other devices or the environment and transform it into digital signatures shared with users, other smart devices, and applications through the connectivity layer.

2.3.1.2 Connectivity layer

The IoT connectivity layer is positioned between the device layer and the cloud layer in the IoT architecture. This layer facilitates the communication between the device and the cloud, using various networking components and protocols (Wortmann & Flüchter, 2015). The IoT connectivity

layer consists of a physical device or software application that captures the data signatures from the device layer of connected devices and uploads them to the cloud.

2.3.1.3 Cloud layer

In the third layer of the IoT architecture, data from the connected devices are housed in Big Data datastores. It can be subjected to various analytics, transformation and monitoring techniques to derive value for the end users (Porter & Heppelmann, 2015). The cloud layer exposes the device to the greater Internet, allowing for broader communication and the ability to visualise processed data signatures on user-facing devices from anywhere.

2.3.2 IoT risks

The IoT environment is a complex, heterogeneous collection of devices made up of sensors and communication nodes often resource-constrained (Kandasamy *et al.*, 2020). According to Zhang *et al.* (2018), IoT devices generate considerable traffic on the Internet, essentially making them a significant contributor to Big Data. The substantial data contribution of IoT devices leads to an inevitable increase in cyber-attacks (NG & Selvakumar, 2020). Furthermore, IoT devices are often susceptible to the exploitation of confidentiality, integrity and availability vulnerabilities (Butun *et al.*, 2019). Kandasamy *et al.* (2020) describe four interconnected cybersecurity risks in the IoT domain: ethical risk, privacy risk, technical risk and security risk. These four risks are discussed next.

2.3.2.1 Ethical IoT risk

Ethical risk refers to the unforeseen adverse effects of unethical actions. In the context of IoT, ethical risk relates to the immoral actions of people or companies using IoT devices (Kandasamy *et al.*, 2020). For example, a woman from the US ended up suing the manufacturer of a smart sex toy after discovering that it had been collecting and communicating user data to the manufacturer without her permission (Guardian, 2016). This raises the moral issue of informed consent.

2.3.2.2 Privacy IoT risk

IoT devices collect vast amounts of consumer data that can be inspected, analysed and shared, posing a significant risk concerning the use of said data and its privacy (Weber, 2015). Privacy IoT risk is concerned with the safekeeping of sensitive information collected by and stored on devices. The raw data collected using IoT devices are invaluable for extracting usage patterns to support development and improvement. Still, the inherent privacy risk could allow sensitively identifiable information to be shared with unintended persons (Weber, 2015). For example, data

analysis of a client's purchasing habits at a grocery store resulted in the system predicting that she was pregnant, based on the products she had been purchasing in previous weeks. This prompted the automated mailing system to promote baby products directly to her home, startling her family since the prediction was far from accurate (Forbes, 2012). This raises the issue that our personal lives are sometimes no longer entirely private.

2.3.2.3 Technical IoT risk

Technical IoT risk results from hardware or software failure due to poor design (Kandasamy *et al.*, 2020). IoT hardware or software design issues can be attributed to one or more factors, i.e. lack of knowledge of the technology or environment by the user or manufacturer, substandard production standards, and inadequate maintenance or updates during the device's lifetime. One of the most notable examples of a technical IoT risk is Intel's Meltdown flaw identified in 2018 (Meltdownattack, 2018). This vulnerability takes advantage of simultaneous execution, a method used by processors to save time, and uses memory allocation properties to access restricted memory segments (Kaspersky, 2019). This enables a rogue program to access the memory of other programs and the operating system, as well as any information stored within those memory segments.

2.3.2.4 Security IoT risk

According to Kandasamy *et al.* (2020), vulnerabilities in a device can be exploited to gain access to assets with the intent of causing harm. IoT devices are exceptionally vulnerable to attack, since they spend most of their time as unattended devices connected to the Internet (Atzori *et al.*, 2010). The complexity of the IoT communication landscape further exacerbates the challenges of designing and implementing security mechanisms to protect these devices (Roman *et al.*, 2013). An example is the implantable cardiac devices identified by the US Food and Drug Administration in 2017 as having a severe vulnerability that exposes them to hacking (FDA, 2017).

As discussed above, the myriad of devices connected to the Internet poses numerous risks. Due to the complexities and diversity of the systems and data, IoT devices have unique security vulnerabilities (Elrawy *et al.*, 2018). The risks posed by these IoT devices are difficult to categorise within current risk frameworks, making cyber-attacks even more challenging to detect (Liang *et al.*, 2020). This all culminates in the growing need for cybersecurity (Mouheb *et al.*, 2019). Cybersecurity is discussed in the next section.

2.4 Cybersecurity

A recent Cybersecurity Ventures report predicts an exponential rise in cybercrime over the next five years, estimated to cost the world approximately $6 trillion per year by 2021 (Ventures, 2019). The decisions made about cybersecurity shape the future and affect everyone (Singer & Friedman, 2014). Given the growing complexity of computer networks and the sheer number of centralised systems and volume of data being generated globally, companies simply cannot afford to have their network security be only an afterthought (Zhu, 2020). Cybersecurity, which refers to the security of cyberspace, encompasses all the actions taken to protect digital assets from malicious intent (ENISA, 2016).

An effective cybersecurity approach requires the coordination of security efforts throughout an information system on the following levels:

- Network security: The practice of securing a computer network from unwanted users, attacks and intrusions by controlling incoming and outgoing connections to prevent threats from entering or spreading on the network (Cole *et al*., 2011);
- Application security: Keeping software and devices free of threats through constant updates and testing (Altekar *et al*., 2005);
- Information security: Protecting data integrity and privacy, both in storage and in transit (Whitman & Mattord, 2017);
- Operational security: The processes and decisions enforced to ensure that digital assets are secure. This includes the permissions users have when accessing a network and the procedures that determine how and where data may be stored or shared (Littlewood *et al*., 1993); and
- Disaster recovery and business continuity: A plan for how an organisation responds to a cybersecurity incident or any other event that causes the loss of operations or data, detailing how the organisation will restore its operations to a functioning state (Sahebjamnia *et al*., 2015).

Traditionally, physical security was thought to be sufficient for the protection of digital assets. However, given the rapidly changing nature of cybersecurity threats, physical security alone is no longer considered an effective barrier of defence against modern cybersecurity threats. In contrast to the traditional passive physical defence model, the National Institute of Standards and Technology (NIST) has recommended a shift toward more active and continuous security

monitoring in organisations (NIST, 2018). Applying this continuous monitoring enables system administrators to continually view all activity on the network and, subsequently, stay one step ahead of cybersecurity threats.

NIST proposes the cybersecurity framework depicted in Figure 2-4 to help identify and prioritise actions for managing and reducing cybersecurity risk. This framework, which promotes business continuity and resilience to future cybersecurity incidents, is reviewed next.

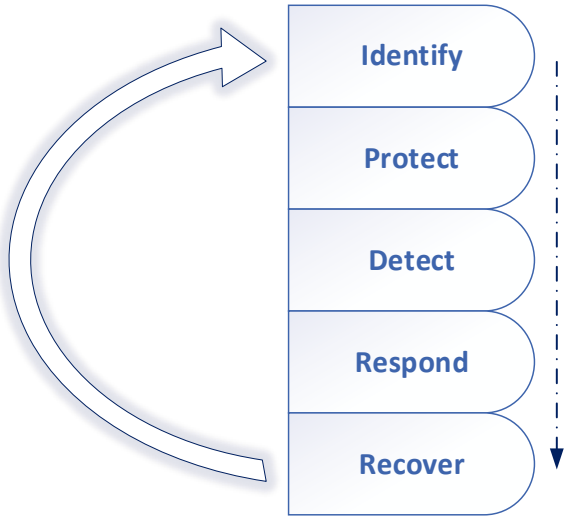


Figure 2-4: Cybersecurity framework

1. Identify: Cybersecurity risks should be managed by proactively identifying potential vulnerabilities and prescribing the resources necessary to support any actions to remediate them successfully;

2. Protect: Every possible measure should be taken to develop and implement appropriate safeguards to prevent or mitigate the impact of a potential cybersecurity incident;

3. Detect: Continuous and proactive monitoring activities should be exercised to detect the occurrence of a cybersecurity incident so that steps can be taken before irreparable harm is done to the system;

4. Respond: Once a cybersecurity incident has been detected, appropriate steps should be taken to mitigate its impact on the system; and

5. Recover: The system should remain resilient and prepared for future cybersecurity incidents and enable the recovery of lost information and the timely restoration of the system to normal operations.

According to Mouheb *et al.* (2019), there has been a dramatic increase in the number and sophistication of cybersecurity threats being observed on the Internet. However, not all of them are malicious. The following section expands on the different types of security threats.

2.5 Types of computer security threats

According to Jouini *et al.* (2014), cybersecurity threats can originate from internal or external entities. The multidimensional threat classification model they proposed is depicted in Figure 2-5 and is summarised next.

1. Internal or external sources can bring on a cybersecurity threat:

    (a) Internal threats originate from within the organisation and are usually the result of employee action to circumvent any controls established to protect an asset; and

    (b) External threats result from individuals obtaining unauthorised access to the network or system through hacking or physical intrusion. Natural disasters also form part of external threats to an organisation.

2. The agents that present a threat to the organisation can be categorised into human (e.g. hackers or disgruntled employees), technological (e.g. hardware or software), and environmental (e.g. fire or lightning):

    (a) Human threats include individuals who intend to circumvent security measures to cause harm to the system;

    (b) Technological threats include the risk of hardware and software failure or loss caused by physical means; and

    (c) Environmental threats are derived from natural disasters, such as fires, floods and lightning that can compromise the availability of a system.

3. The reason for wanting to attack or compromise a system can be motivated by either malicious or non-malicious intentions:

(a) Malicious threats refer to attacks launched to cause damage or disruption to the system. Examples include malware, ransomware, viruses or social engineering. Malicious threats can be intentional or accidental; and

(b) Non-malicious threats result from poor policies, training and ineffective controls. This results in non-intentional harm occurring to the system.

4. Humans perform actions accidentally or to cause harm:

(a) Accidental threats are unintentional threats posed by either human, environmental or technology agents; and

(b) Intentional threats result from malicious actions taken by human agents with the intent of causing harm.

5. Risks associated with these threats include the following:

(a) Unauthorised access to information stored can result in loss of credibility, reputation, market share, and competitive edge;

(b) Loss of information can seriously hamper the business operations and result in substantial financial losses until it can be restored;

(c) Corruption of information occurs when a data element or instance loses its base integrity and transforms into a form that is no longer meaningful; and

(d) Destruction of information caused by the deliberate removal of data to ensure that it no longer exists on the system and cannot be restored.

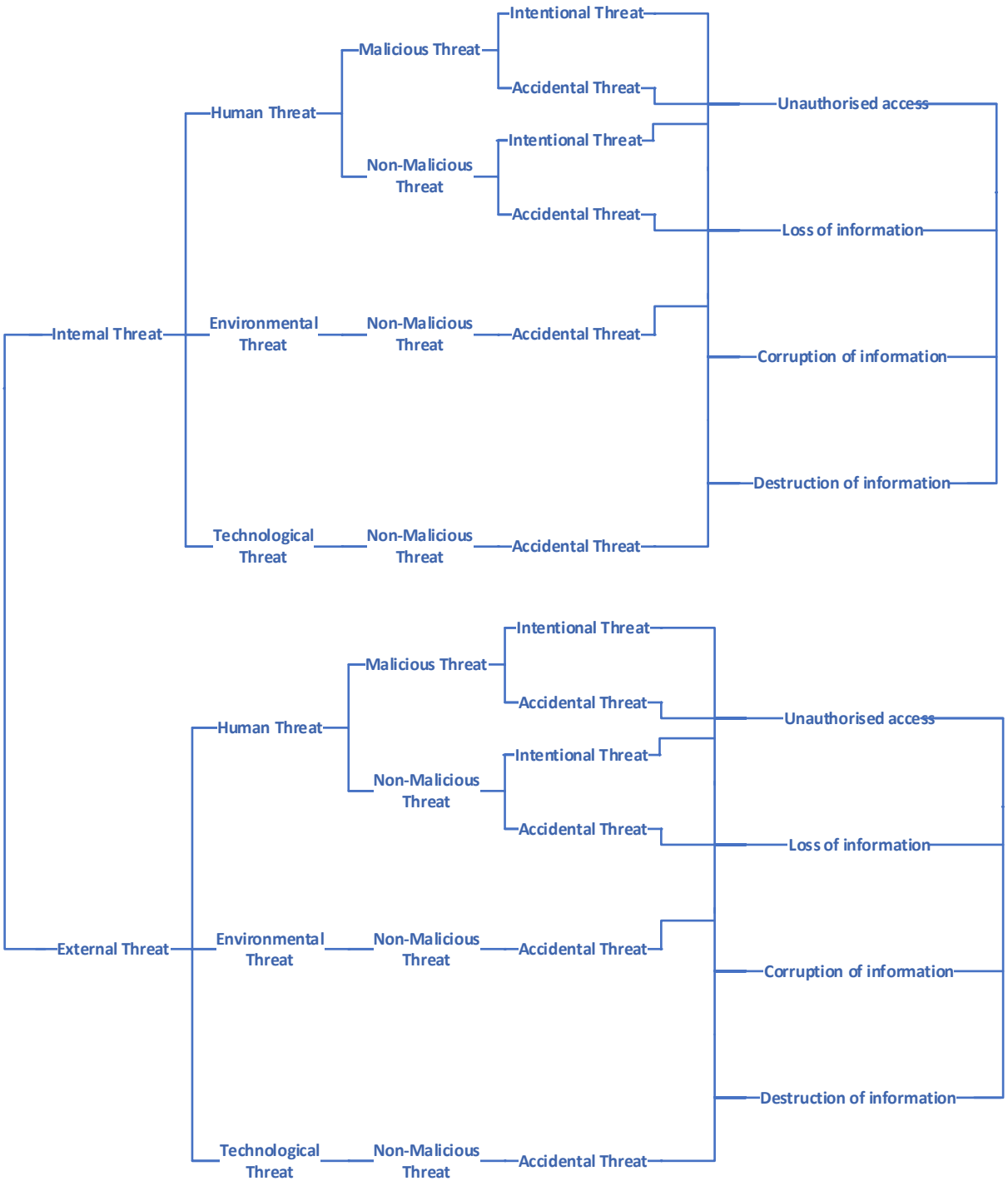


Figure 2-5: Multidimensional threats classification model

As summarised above, computer security threats can originate from different sources, motivations and intentions. Cybercrime is an ongoing threat and protecting against cyberthreats requires various tools and techniques. Popular approaches and tools for securing information and data, according to the literature, are discussed next.

2.6 Approaches for securing information and data

Security is a continuous process undertaken to protect an asset from unauthorised access. There is more than one technique for securing information and data. Ahmed *et al.* (2016) proposed using statistical analysis and machine learning techniques to battle intrusions. Modi *et al.* (2013) encourage implementing a firewall to serve as the first line of defence in a system to prevent unwanted traffic from entering the network. Kizza *et al.* (2013) propose that prevention is perhaps the best course of action for securing a system from intrusion with the caveat of not always knowing against what type of attack to defend. Ultimately, these techniques should be used in unison as a more effective way of securing information and data from intruders. According to Kizza *et al.* (2013), the security of an asset can be assured, provided the following four protection mechanisms are in place:

1. Deterrence: Applying warnings of consequences as the first line of defence to deter intruders from attempting to gain access to an asset;

2. Prevention: Actively working towards preventing intruders from gaining access to an asset through the implementation of physical security measures, firewalls, access controls and intrusion prevention systems;

3. Detection: After intruders have successfully circumvented previous security measures, the detection and alert of said intruders are paramount to the success of any response plan that follows; and

4. Response: The response plan followed upon successful detection of an intruder to stop and prevent future intrusions of the asset.

Bourgeois *et al.* (2014) propose using several tools and techniques for securing information and data. These tools and techniques are briefly discussed next.

2.6.1 Authentication

Authentication is the process undertaken to verify the identity of someone or something. This can be accomplished by identifying someone through something they *know*, *have* or *are*. For example, authenticating someone using a user ID and password, a physical key or access card, or physical characteristics, such as fingerprints or retinal scans.

2.6.2 Access control

Following the successful authentication of a user or system, the next step would be to ensure that access to information and resources is appropriately restricted. Access control systems provide the appropriate level of access and actions to the user or system in line with respective information security policies.

2.6.3 Encryption

Data should be encrypted to ensure that only authorised persons can access data upon its transmission or storage. Encryption protects information sent over the Internet by scrambling readable text (plain text) into encoded text (ciphertext) so that only the intended recipient can view it using the correct decryption key (Grace, 2020).

2.6.4 Backups

Regular backups of important information and data enable the timeous restoration of systems after interruptions caused by user errors, hardware failures or natural disasters (Chervenak *et al.*, 1998, March).

2.6.5 Firewalls

Firewalls protect systems from external threats by filtering traffic entering or leaving the network, based on a predefined set of rules. Firewalls actively monitor attempts to access the system and block unrecognised sources and unwanted traffic from entering the local network accordingly.

2.6.6 Intrusion detection systems

In contrast to the tools and techniques mentioned above, intrusion detection systems (IDSs) do not add any additional security to a system. Instead, they are placed on a network to identify when the network is being attacked and alert security administrators to the intrusion. An IDS is an essential part of every network to alert system administrators to a threat or intrusion so that remedial action can be taken to prevent further harm to other systems or resources (Javaid *et al.*, 2016). As discussed in this chapter, IDSs are an integral part of every computer network security system. The plausibility of automating such IDSs using artificial neural networks and deep learning techniques will form the basis of this study.

2.7 Chapter summary

In this chapter, the concept of and need for cybersecurity was further explored. This included an in-depth discussion about the inherent threats posed to information systems by a large network of devices, such as the Internet. The concepts of Big Data and IoT were also introduced, followed by a discussion of the risks that these phenomena pose to information systems. Finally, common computer security threats were considered and the current approaches and tools used for securing information and data were discussed. The complexities of cybersecurity are evident from these discussions and the various solutions currently being employed to enforce it. Ultimately, even the most secure system or network can eventually be breached by an intruder, given enough time and resources. This study will, therefore, explore the realm of intrusion detection as a second line of defence for networks and computers alike, and how this can be automated and improved, using an artificial neural network. In the next chapter, intrusion detection will be discussed in the context of information systems.

CHAPTER 3 INTRUSION DETECTION SYSTEMS

The creation of the Internet has given rise to the availability and accessibility of vast amounts of information. The introduction of networks has further enabled similar and heterogeneous devices to communicate and share information. In today's massively interconnected world of devices, access to information can be obtained in an instant. However, uninterrupted access to great volumes of data can pose a significant threat to information security. The safekeeping of data is, therefore, of growing concern to organisations and individuals alike.

Traditionally, preventative mechanisms, such as access control, encryption and firewalls have been proposed in the literature as the first line of defence against network and system intrusions (Subba *et al.*, 2016). While these mechanisms improve network and system security, they are not guaranteed to rule out intrusion threats completely.

Intrusion detection systems, which have been proposed by Subba *et al.* (2016) as a complementary second line of defence for securing a system and network, aim at detecting attacks against computer systems and networks or information systems in general.

In this chapter, the second step of the design science research methodology will be addressed by conducting a systematic literature review to synthesise the objectives identified in Chapter 1. An introduction to information security, intruders and intrusion detection is given in Section 3.1. This provides the foundation for the chapter. Background information for intrusion detection systems, including their origin and different implementations, is provided in Section 3.2. Various approaches for intrusion detection are discussed in Section 3.3. Finally, a summary of the chapter is presented in Section 3.4.

3.1 Introduction

In this digital age, almost every person will encounter an information system in some shape or form. Information systems can be found virtually everywhere, from smartphones, laptops, and personal computers to electronic menus and ordering systems in restaurants (Torres, 2016).

3.1.1 Information systems

According to research done by Boell and Cecez-Kecmanovic (2015), the definition of an information system can take one of four distinct views:

1. Technological view: Where the emphasis is placed on the hardware and software components that are used to develop or enable the information system;

2. Social view: Information systems are a social-first, technology-second system, supporting the function of society through the use of technology;

3. Socio-technological view: Concerned with the interaction between technological and social systems; and

4. Process view: Where an information system more accurately resembles a working system devoted to processing and displaying information.

A common theme between these views is the interaction between people, data and technology. This interaction is visually depicted in Figure 3-1.

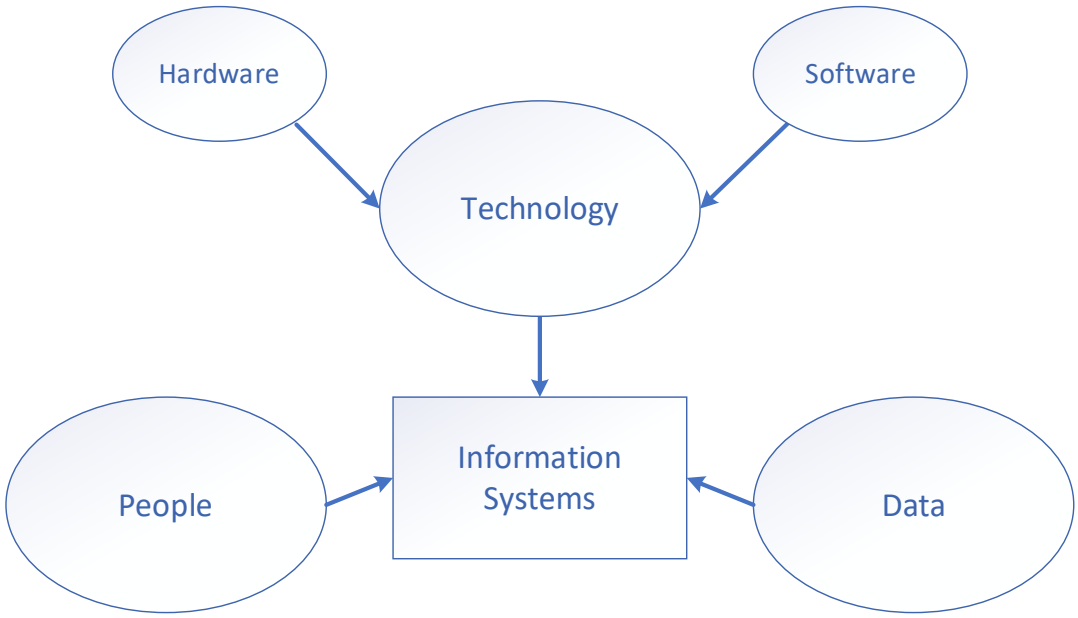


Figure 3-1: The interaction of information systems

According to Pfleeger *et al.* (2015), information systems should be regarded as assets, and assets need to be protected (Fennelly, 2004). The characteristics of availability, integrity and confidentiality make something valuable (Pfleeger *et al.*, 2015).

- Availability: The ability of a system to ensure that something is available and capable of use when needed;

- Integrity: The ability of a system to ensure that the data stored on it is accurate and complete; and

- Confidentiality: The ability of a system to ensure that no unauthorised persons (or devices) can gain access to the data stored on a system.

According to Heady *et al.* (1990), any set of actions that attempts to compromise the integrity, confidentiality or availability of a resource is known as an intrusion. However, to better understand an intrusion, an intruder must first be defined.

3.1.2 Intruders

Intruders are illicit persons or devices that have gained unauthorised access to a system by circumventing their first-line defence mechanisms (Pillai, 2011). The actions of an intruder, which can stem from varying levels of skill, resources and access, are often motivated by financial gain or malice (Abomhara & Køien, 2015). System intrusions can originate from both internal and external sources (Borkar *et al.*, 2017):

- Internal intruders: Users with authorised access to the system, who attempt to gain additional privileges for which they are not authorised; and

- External intruders: Users who are not authorised to use a system, who access it from an outside source, such as the Internet.

According to Yousaf and Yousaf (2017), rapid advancement in Cloud Computing domains, the Internet of Things (IoT), and Big Data has seen intruders evolve. The traditional TCP/IP protocol stack is not capable of defending against modern threats posed by intruders. Furthermore, even the best intrusion prevention systems will inevitably fail (Stuckman & Purtilo, 2011). This has led to additional research in intrusion detection, which is discussed in the next section.

3.1.3 Intrusion detection

Cybersecurity threats have increased dramatically over the last decade, fuelled by rapidly evolving technology and the increased reliance on the Internet and information systems (Bendovschi, 2015). An effective security plan is currently crucial. According to the literature, a variety of approaches can be used to secure the data stored within a system, but the ability to recognise instances of an attack on the system is paramount to an effective security system (Keegan *et al.*, 2016; Lazarevic *et al.*, 2005; Cannady & Harrell, 1996).

Intrusion detection is described as the process of monitoring the events occurring on information systems and networks and evaluating these events for signs of intrusion or system misuse

(Scarfone & Mell, 2012). A system that runs software or hardware to automate the detection of intrusions is known as an intrusion detection system.

Denning and Neumann (1985) identified the following reasons for utilising intrusion detection systems to defend against cyberthreats:

1. Technical limitations on existing systems can make them vulnerable to attack;

2. Due to application and financial considerations, existing systems with security flaws cannot easily be replaced by more secure systems;

3. The development of a system that is completely secure is impossible; and

4. Even highly secure systems can be misused by authorised users.

The field of intrusion detection, including its origin and different implementations, is discussed in the next section.

3.2 Background

Before continuing the discussion on intrusion detection systems, it is first necessary to review their history and architecture.

3.2.1 History

Intrusion detection was first introduced by Anderson (1972), who suggested the need to detect security breaches in computing systems. Anderson (1980) proposed that certain types of threats to a system could be identified by reviewing the system's audit trail for abnormalities and that this process could be automated. Anderson's early work subsequently gave rise to a plethora of research in the field of host-based intrusion detection systems. Five years later, Denning and Neumann (1985) developed the first real-time threat detection system, using a list of expert-written rules. Heberlein *et al.* (1990) later identified the need for a network-based security monitor, which subsequently paved the way for further research in the field of network intrusion detection systems.

3.2.2 Architecture

According to Debar *et al.* (1999), an intrusion detection system can be described as a high-level tool that collects and processes information related to the system (or network) that it has been tasked with protecting. An intrusion detection system uses three types of information:

1. Long-term information related to the technique used to detect intrusions;

2. Configuration information about the current state of the system; and

3. Audit information describing the events that occur on the system.

The general architecture of an intrusion detection system is depicted in Figure 3-2, which will be explained next.

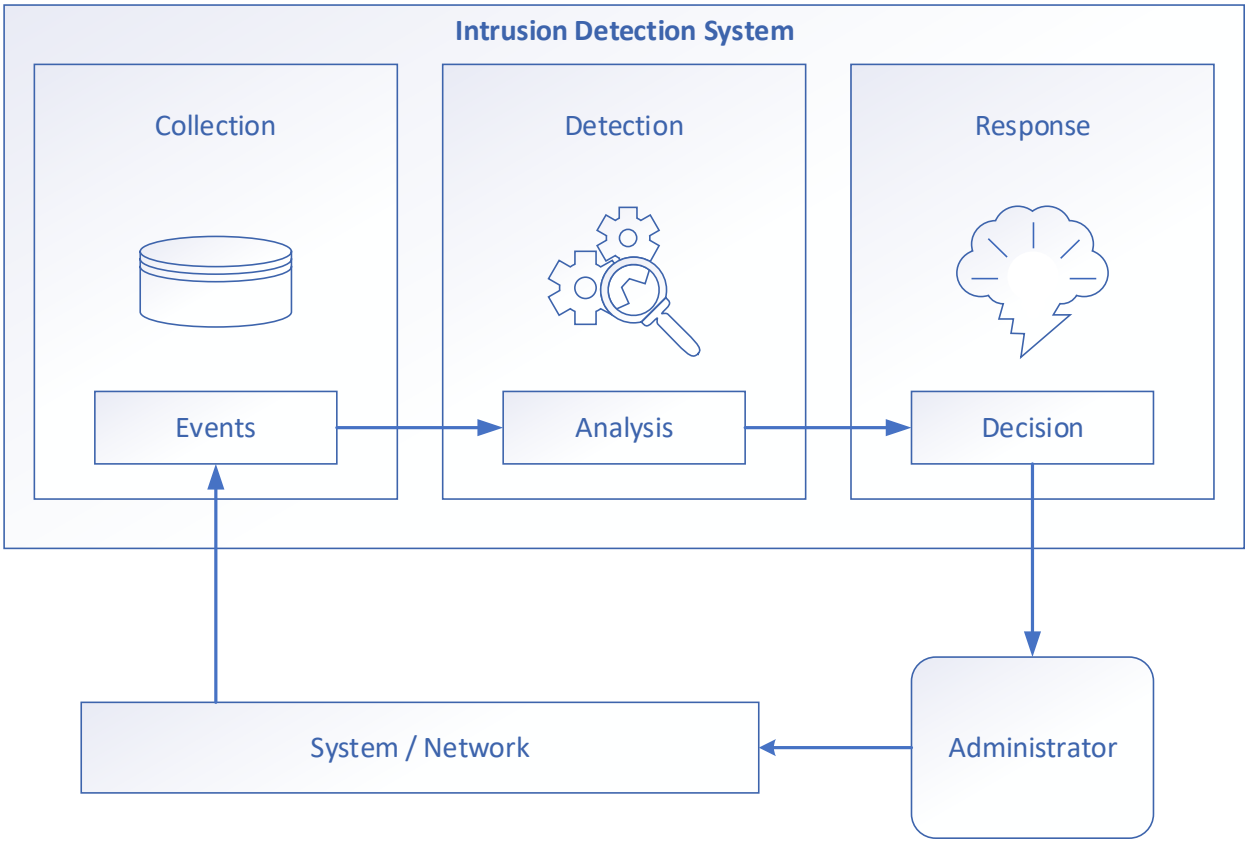


Figure 3-2: Architecture of an intrusion detection system

In the first stage of an intrusion detection system, a system or network generates events collected by the first module of the intrusion detection system. These events are then processed before being stored in the database (Bhati & Rai, 2020). Next, this information is forwarded to the detection module of the intrusion detection system where it can be further analysed.

Upon reaching the detection module of the intrusion detection system, the information is analysed and unnecessary information about the event is discarded (Pillai, 2011). The detection module will then explore what is left of the information, comparing this to its known database of intrusion signatures or a pre-defined set of rules (Ferrag *et al.*, 2020).

Finally, the information is transferred to the response module of the intrusion detection system. Ultimately, a decision is then made to evaluate the probability that the event can be considered symptomatic of an intrusion (Alpern & Shimonski, 2010). This decision, which often resembles an alarm or alert (Biermann *et al.*, 2001), is then passed on to the administrator, who would take the necessary remedial action against the event on the system or network.

3.2.3 Classification

According to Bace and Mell (2001), intrusion detection systems are commonly grouped by information sources (location) and design approaches which determine the scope and detection capabilities of the intrusion detection system, respectively. Variations of intrusion detection systems based on location and approach, as depicted in Figure 3-3, are discussed in the next section.

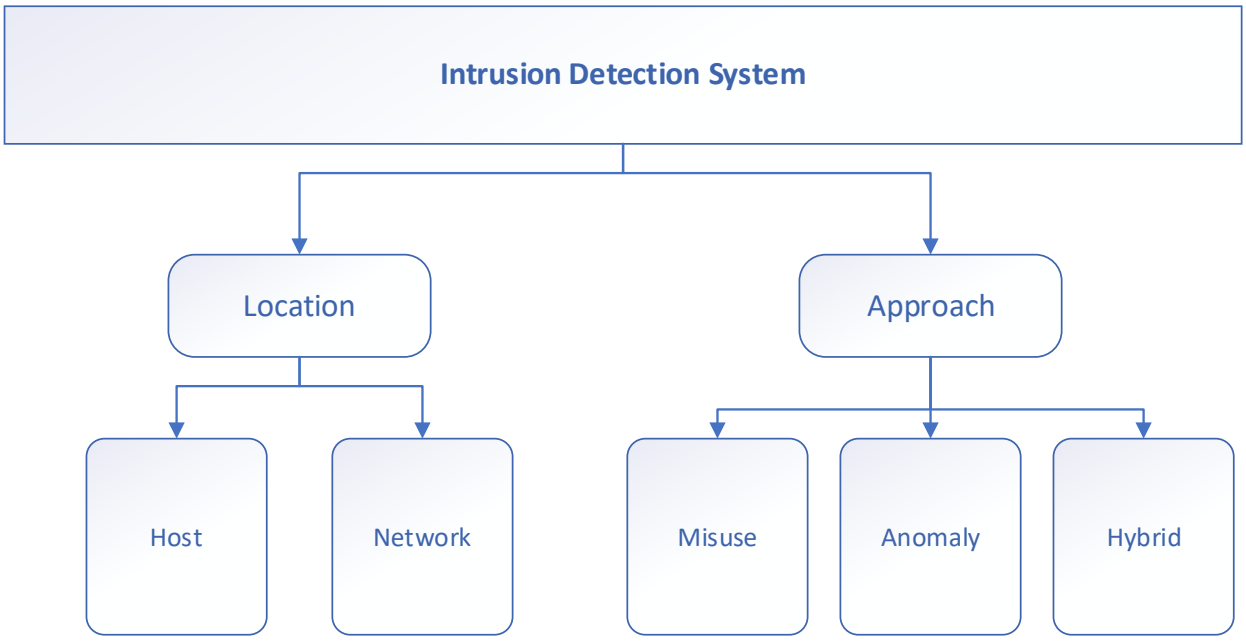


Figure 3-3: Classifications of an intrusion detection system

Intrusion detection systems are commonly classified based on their location within an organisation and according to the information source utilised for their operation (Bridges *et al.*, 2019). Furthermore, the location of an intrusion detection system is critical, since it relies on the data available within its environment to support its function (Modi *et al.*, 2013). This is echoed by Khraisat *et al.* (2019), who identify two general input data sources utilised for intrusion detection systems, namely host-based intrusion detection systems (HIDSs) and network-based intrusion detection systems (NIDSs). These two types of intrusion detection systems are discussed next.

3.2.4 Host-based intrusion detection

According to Bridges *et al.* (2019), a HIDS generally refers to a software component installed on and monitoring a single system. A HIDS gathers data from the host it is protecting. It then uses this information to record the current status of the host and ultimately compares it to a pre-recorded system file for classification purposes (Vokorokos & Baláž, 2010). This gives a HIDS excellent insight into the system's state but poor isolation from it, allowing an intruder with access to the system to either exploit or disable it (Bridges *et al.*, 2019). Another key disadvantage of a HIDS is the significant processing power required by every system to support its function. An example of a HIDS implementation is presented in Figure 3-4.

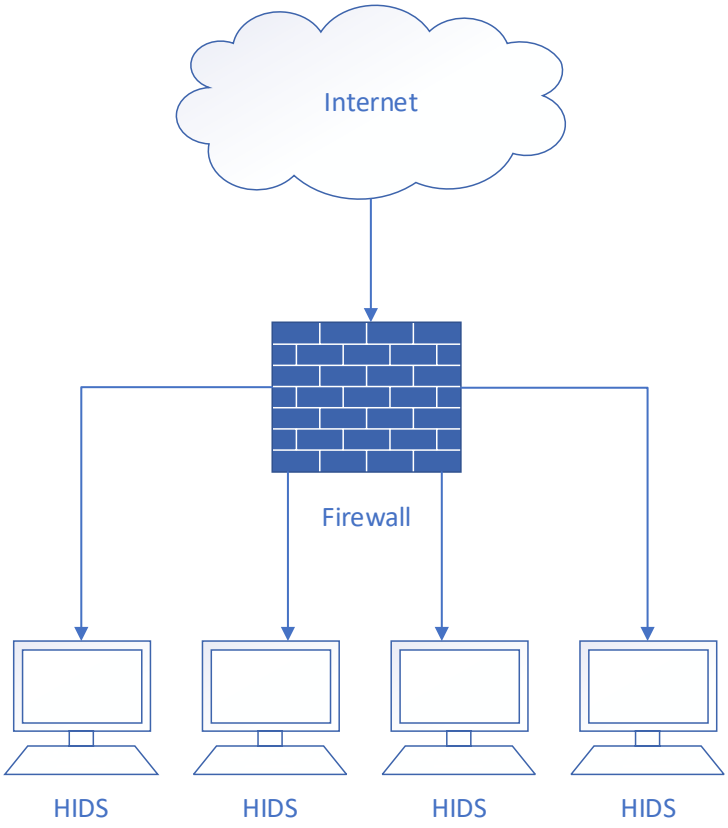


Figure 3-4: Host-based intrusion detection system

In the HIDS implementation depicted in Figure 3-4, an intrusion detection system can be seen running on multiple hosts, with processing performed by each host. Traffic originates from the Internet and passes through a firewall, which is a network security device that monitors incoming and outgoing network traffic and filters data packets based on a set of rules (Al-Shaer & Hamed, 2004). It then makes its way to the HIDS where it is further analysed.

3.2.5 Network-based intrusion detection

Rather than analysing information that originates and remains on a computer, a NIDS utilises packet-sniffing, which is the practice of gathering, collecting, and logging some or all traffic that passes through a computer network, to retrieve information from the TCP/IP stack or other protocol's packets passing through the network. A NIDS detects intrusions by collecting and analysing network packets. According to Singh and Singh (2014), a NIDS tracks and analyses network traffic to protect a system from network-based threats. It works by scanning all incoming packets for signs of suspicious patterns. When threats are detected, the system takes remedial action, such as sounding an alarm or notifying administrators of the breach.

In contrast to a HIDS, a NIDS can span an entire computer network, centralising processing requirements (Pamukchiev *et al.*, 2017). According to Bridges *et al.* (2019), a NIDS is usually represented as an independent physical device situated on the network which monitors multiple systems on a shared network. A crucial drawback of a NIDS is that it may not analyse and classify all traffic on a large network fast enough and may overlook some intrusions. In this study, one possible solution to this problem is investigated by applying artificial neural networks and deep learning techniques to classify network traffic more accurately and reliably. An example of a NIDS implementation is presented in Figure 3-5.

In the NIDS implementation depicted in Figure 3-5, an intrusion detection system can be seen running on a single device with protection spanning the entire network. In the next section, the various intrusion detection approaches will be discussed.

3.3 Intrusion detection approaches

According to Bace and Mell (2001), there are two basic approaches for analysing events to detect intrusions: misuse detection and anomaly detection. However, in later years, to resolve some of the challenges within the first two approaches, researchers have proposed a third approach for intrusion detection: a hybrid approach (Bostani & Sheikhan, 2017; Kim *et al.*, 2014; Depren *et al.*, 2005). These three intrusion detection approaches are discussed next.

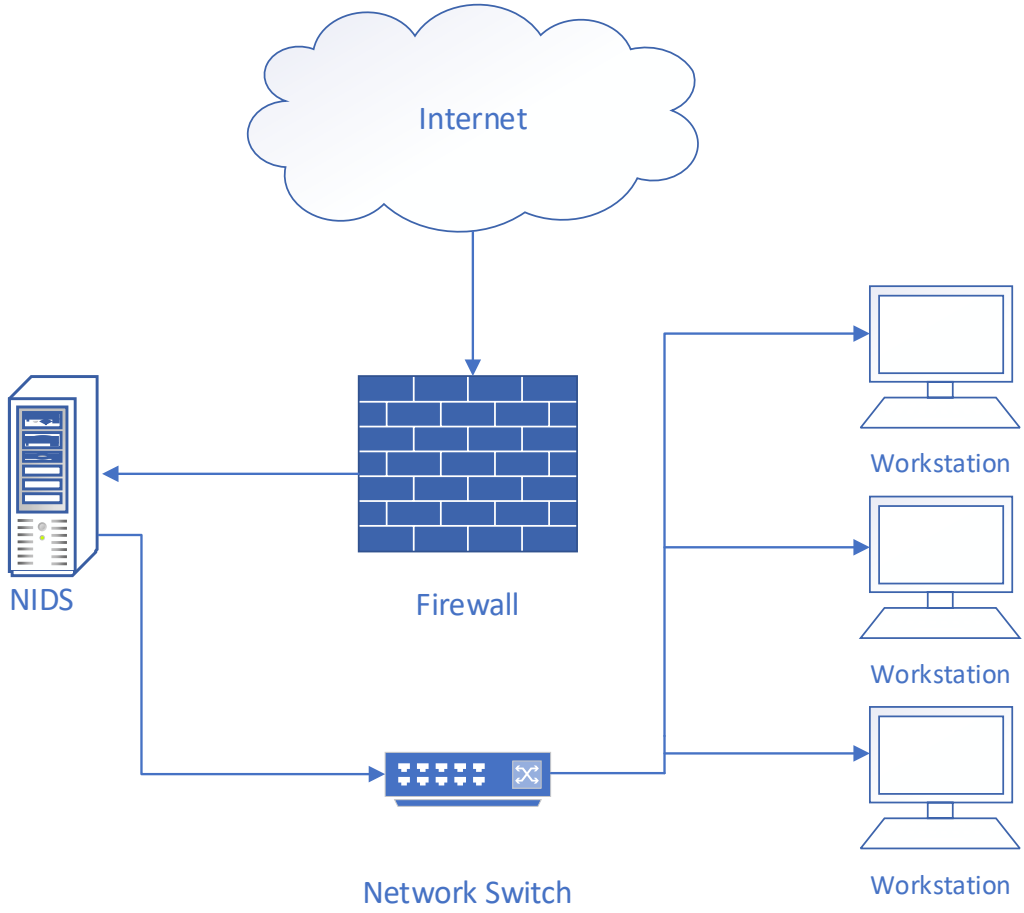


Figure 3-5: Network-based intrusion detection system

3.3.1 Misuse detection

According to Cannady and Harrell (1996), the first general approach to intrusion detection, misuse detection, employs a rule-based system and uses threshold monitoring to determine when a particular established metric has been reached. These metrics are generally one of three types:

1. Event counters: Monitor the number of times a single event occurs over a given period, e.g. the number of authentication requests or how many times a specific file has been accessed;

2. Time intervals: Determine how much time passes between two related events, e.g. the duration between a user's logins during the day; and

3. Resource measurement: Seek to quantify the resources used by the system over a given period, e.g. the volume of traffic transmitted over the network, or CPU expenditure.

Misuse detection, a technique used by most commercial systems (Bace & Mell, 2001), involves comparing a user's behavioural metrics to a knowledge base of established intruder patterns (or signatures) to detect signs of misuse and classifying them accordingly (Mukherjee & Sharma, 2012). In essence, misuse detection is concerned with identifying intrusions resulting from known techniques (Jones & Sielken, 2000).

According to Ferrag *et al.* (2020), a significant disadvantage of a misuse detection approach is that it is highly dependent on frequent updates to the signatures in its knowledge base. As a result, it is unable to detect new attacks on a system. Therefore, it can have a high false negative rate.

Nonetheless, according to Bace and Mell (2001), a misuse detection approach is very accurate and can be used to quickly and reliably diagnose specific intrusion techniques. It allows system administrators to take remedial action before it is too late. Therefore, the most significant advantage of a misuse detection system is that known attacks can be detected with a very low false positive rate.

3.3.2 Anomaly detection

On the other hand, an anomaly detection system defines a detailed and accurate profile of normal user behaviour. Anomaly detection works by recognising variations in behaviour for users or classes of users compared to established and accepted patterns (Cannady & Harrell, 1996). According to Biermann *et al.* (2001), anomaly detection is based on the assumption that a system intrusion will be significantly different from normal system operation. An intruder will behave differently from a typical user. Following this reasoning, an anomaly detection system will learn the description of normal or expected behaviour through observation and treat any deviation from this as a possible intrusion.

The most significant disadvantage of an anomaly detection system is accuracy (Wu & Banzhaf, 2010). It is difficult to distinguish between normal and abnormal behaviour, leading to a high rate of false alarms generated by the system. In addition to this, anomaly detection is dependent on input from the operating system's audit record which imposes significant processing requirements on the system (Cannady & Harrell, 1996).

The main advantage of an anomaly detection system is its ability to detect novel attacks on a system without specifying attack models (Bridges *et al.*, 2019; Jabez & Muthukumar, 2015; Biermann *et al.*, 2001). Anomaly detection systems can create signatures for misuse detection systems (Bace & Mell, 2001).

3.3.3 Hybrid detection

Hybrid approaches to intrusion detection have also been proposed to resolve the disadvantages of misuse detection and anomaly detection systems (Kim *et al.*, 2014). Hybrid intrusion detection systems work by combining misuse detection models and anomaly detection models to develop a new system that aggregates the results of the detection models (Depren *et al.*, 2005).

Hybrid detection systems are, however, not without their drawbacks. According to Kim *et al.* (2014), a hybrid detection system can result in a high false positive rate since it relies on the inherent misuse or anomaly detection system to classify traffic.

The advantage of a hybrid detection system is that it can offer the accuracy of a misuse detection system and the ability to detect never before seen attacks thanks to its anomaly detection component. For the purposes of this study, the anomaly detection approach will be adopted as it is commonly used in artificial neural network implementations. An artificial neural network would be capable of analysing the data from the network, even if the data is incomplete or distorted (Cannady, 1998). Artificial neural networks are discussed in the next chapter.

3.4 Chapter summary

In this chapter, an introduction to information security and intrusions was provided. This was followed by a discussion around the history and architecture of intrusion detection systems. Several variations of intrusion detection systems, based on their location and approach, were identified in the literature. Host-based intrusion detection systems were found to be resource-intensive and difficult to deploy at scale. Therefore, in this study, the emphasis will be on network-based intrusion detection systems and the application thereof. Additionally, three approaches for intrusion detection were discussed, namely misuse detection, anomaly detection and hybrid detection. While anomaly detection may lead to higher false positives, its ability to detect novel intrusions makes it a favourable approach for the implementation of an automated intrusion detection system based on a neural network. In the next chapter, artificial neural networks with a focus on deep learning will be examined.

CHAPTER 4 DEEP NEURAL NETWORKS

Artificial neural networks (ANNs) are powerful computational models capable of solving complex problems in artificial intelligence (Benítez *et al.*, 1997). More recently, deep neural networks (DNNs), which are essentially ANNs with multiple layers between the input and output layers, have emerged as powerful machine learning models with the capacity to learn more complex patterns than traditional ANNs (Szegedy *et al.*, 2013). Factors, such as accuracy, fault tolerance, scalability, and performance, inspired by biological neural networks, make these models effective, efficient and successful in solving complex problems, such as intrusion detection.

In this chapter, steps two, three and six of the design science research methodology are addressed by conducting a systematic literature review to synthesise the objectives identified in Chapter 1 and by determining the required functionality and architecture needed for the design and development of an artefact. An overview of deep neural networks is provided. Firstly, artificial intelligence is briefly introduced in Section 4.1, exploring its history. Secondly, neural networks are described in Section 4.2, emphasising their biological origin, structure, and mathematical design. A rudimentary neural network model known as the perceptron is also introduced, followed by a brief explanation of the different activation functions commonly used when building artificial neural networks. Thirdly, deep learning is discussed in Section 4.3 by providing definitions for it, explaining its components and the learning categories found in this discipline. Backpropagation, a popular training technique used in deep learning models, is also presented, and the deep learning model adopted for this study, namely an autoencoder, is introduced. The best-first search architecture optimisation algorithm used in this study to construct an accurate symmetrical autoencoder model used as part of an intrusion detection system is presented in Section 4.4. Finally, a summary of the chapter is presented in Section 4.5.

4.1 Artificial intelligence

Artificial intelligence (AI) is a broad branch of computer science concerned with creating systems that can function intelligently and autonomously (Li *et al.*, 2017a). AI can be achieved by studying how the human brain thinks and how humans learn and decide when attempting to solve a problem and then applying these findings to develop intelligent systems and applications (Akerkar, 2014). According to Bottou (2014), the following aspects of intelligence have been the subject of AI research over the years:

- Learning: Transforming large amounts of data into valuable intelligence;
- Reasoning: Choosing the best algorithm to achieve the desired result;

- Problem-solving: Systematically searching through a range of possible actions to reach some predefined goal or solution; and

- Perception: Acquiring, selecting, organising and interpreting sensory information.

In this study, the learning and problem-solving aspects of intelligence will be investigated. A brief history of the field of AI will be addressed in the following section.

4.1.1 History

In the broadest sense, AI is used to classify machines that mimic human intelligence (Minsky, 2006). In the early days of AI, computer scientists attempted to recreate aspects of the human mind in the computer (Buchanan, 2005).

The origins of modern AI dates back to the early 1940s when the first mathematical model of an artificial neuron was created by McCulloch and Pitts (1943). In the years that followed, research into the field of computer intelligence became more prevalent. By 1950, Alan Turing published a paper proposing how to develop and test a thinking machine (Turing, 1950). He believed that a computer could be described as thinking if it could conduct a conversation using a typewriter, to imitate a person with no discernible differences. This became widely known as the Turing Test, which is, to this day, considered a benchmark to identify intelligence in an AI system. The name artificial intelligence was officially coined by John McCarthy, six years later, in 1956, at the first-ever AI conference at Dartmouth College (Haenlein & Kaplan, 2019).

Rosenblatt (1958) proposed and created the architecture of classical neural networks, the perceptron - the first system based on a neural network, capable of self-learning through trial and error. The backpropagation method, used to train neural networks, was introduced by Werbos (1974). After John Hopfield described the link between neural networks and physical systems (Hopfield, 1982), by the 1980s, neural networks featuring backpropagation algorithms for training the network had become widely used in AI applications. The backpropagation algorithm allowed for the training of ANNs, using multiple layers of perceptrons or MLPs.

Artificial neural networks are discussed in the next section.

4.2 Artificial neural networks

Artificial neural networks are computational models inspired by the brain's distributed, massively parallel processing structure, enabling it to excel in sophisticated prediction and classification tasks (Hassoun, 1995). In addition, these models leverage some of the features inherent in

biological neurons, such as flexibility, agility, collective computing, fault tolerance and robustness (Yegnanarayana, 1999). The biological inspiration for AI and the architecture of artificial neurons are further examined in the next section.

4.2.1 Biological neurons

In the early 1940s, McCulloch and Pitts (1943) proposed that human cognitive functions could be replicated in machines through ANNs by examining the operation of biological neurons in the brain. Before considering ANNs, it is important to provide a clear understanding of how biological neurons function. The similarities and differences between biological and artificial neurons have been used to defend approaches for artificially replicating biological intelligence (Hunter, 1993). According to Russell and Norvig (2016), the human brain has approximately 100 billion neurons. An example of a biological neuron is depicted in Figure 4-1 below.

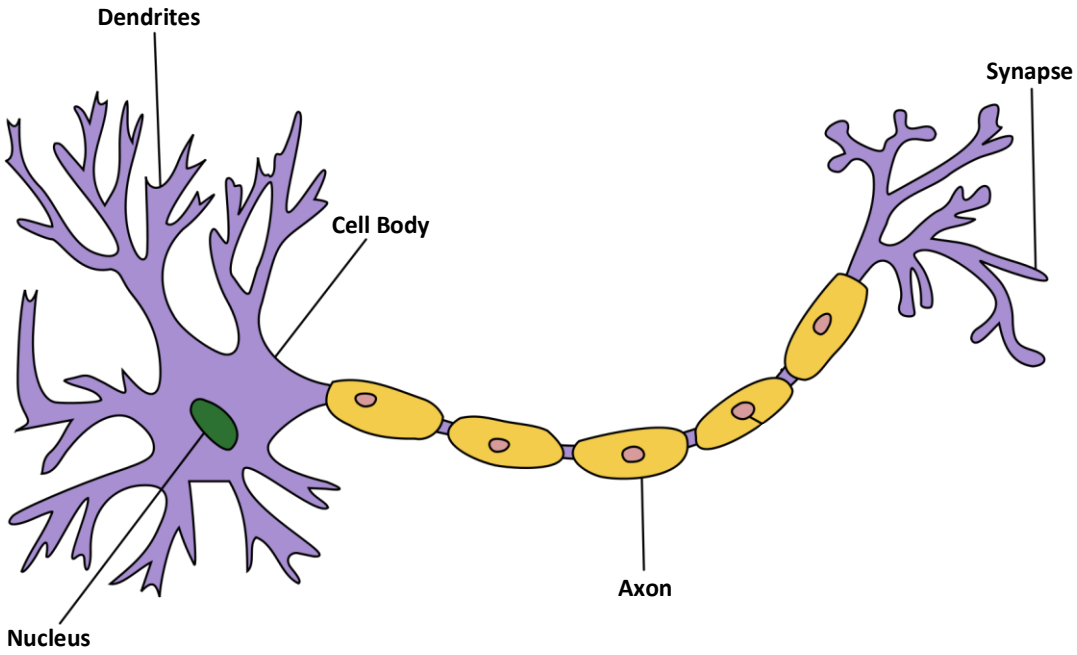


Figure 4-1: Biological neuron (adapted from Jarosz, 2009)

Each of these neurons consists of the following elements (Herculano-Houzel & Lent, 2005; Basheer & Hajmeer, 2000; Johnston *et al.*, 1996):

- Dendrites: The dendrites are responsible for getting incoming signals (or information) from outside which are then passed to the cell body to be processed. A neuron can have multiple dendrites which can carry thousands of input signals each;

- Cell body: Also known as the soma, the cell body is responsible for processing input signals and deciding whether a neuron should fire an output signal. The number of both excitatory (generate an electrical impulse) and inhibitory (prevent the neuron from firing) signals received by a neuron determines whether it is stimulated to fire an output signal;

- Nucleus: The nucleus is a membrane-bound structure found in the cell body of the neuron. It contains the nucleolus and chromosomes, necessary for the coded production of proteins within the cell;

- Axon: The axon is responsible for getting processed signals from the neuron to relevant cells. The axon splits into several branches at the end, forming axon terminals that lead to the synapse; and

- Synapse: The synapse provides a neurochemical connection between an axon and other neuron dendrites. For communication between neurons to occur, an electrical impulse must travel down an axon to the synaptic terminal. These terminals make connections on target cells.

In this section, the structure of biological neurons was discussed. The concept of artificial neurons is discussed in the next section.

4.2.2 Artificial neurons

An artificial neuron is a computational model inspired by biological neurons. Jain *et al.* (1996) drew an analogy between biological neurons and their artificial counterparts, where the axons and dendrites of a biological neuron represent the connections between nodes of an artificial neuron, the synapses represent the weights and the activity in the cell body equates to the threshold of biological and artificial neurons, respectively.

The aim of artificial neural networks is not to recreate the brain in its entirety. Instead, researchers are searching for a better understanding of nature's resources to engineer solutions to problems that traditional computing has not been able to solve (Anderson & McNeill, 1992). In this section, the fundamental architecture of artificial neurons is discussed, along with their mathematical representation related to computer science.

4.2.2.1 Single-input neuron model

A single-input neuron, depicted in Figure 4-2, is a simple unit that receives input signals from a single element. The primary function of the single-input neuron is to transform an input variable p into an output variable a. This process is outlined next.

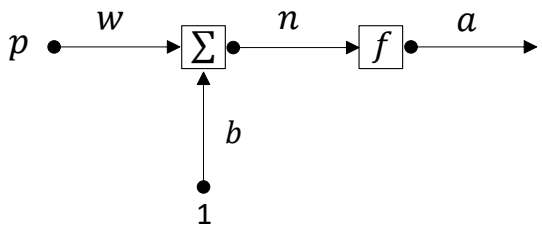


Figure 4-2: Single-input artificial neuron (adapted from Demuth *et al*., 2014)

The first stage of the transformation is to multiply the input p by a weight parameter w, which simulates the synaptic strength of a biological network. Next, an offset parameter or bias b, which here takes a constant input value of 1.0, is added to provide a mechanism for including other influences. The purpose of the bias is to model the fact that biological neurons are affected by factors other than direct inputs. The weight and bias can be either positive or negative to indicate their excitatory or inhibitory nature, respectively. Finally, the product of the input and weight parameters, as well as the bias, are passed to a summing function which results in the parameter n, also known as the net input. This process is summarised in the equation

$$n = \sum wp + b. \tag{4-1}$$

The parameter n is then used in the activation function f to mimic the firing value of the biological neuron, which ultimately leads to the final output a, as given by the equation

$$a = f(n). \tag{4-2}$$

4.2.2.2 Multiple-input neuron model

A multiple-input neuron is an extension of a single-input neuron unit explained in the previous section. Multiple-input neurons can receive input signals from multiple elements and each input would have a corresponding weight value (Hagan *et al.*, 2002). An example of a multiple-input neuron is graphically depicted in Figure 4-3.

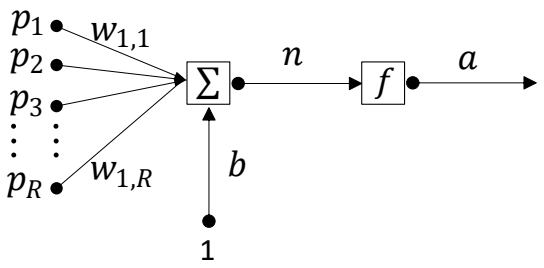


Figure 4-3: Multiple-input artificial neuron (adapted from Demuth *et al*., 2014)

In the multiple-input neuron shown in Figure 4-3, the inputs $p_1, p_2, \ldots, p_R$ are multiplied by the corresponding elements $w_{1,1}, w_{1,2}, \ldots, w_{1,R}$ of the weight matrix $\boldsymbol{W}$. The weighted inputs to which the bias b is added are then summed to give the net input n. This process is summarised in the equation

$$n = \sum_{i=1}^{R} w_{1,i}\, p_i + b. \tag{4-3}$$

The net input n is then used in the activation function f, which ultimately leads to the output a, as given by the equation

$$a = f(n). \tag{4-4}$$

The single-input and multiple-input models described in this section seek to explain the mathematics behind artificial neurons. In the next section, the perceptron, the most rudimentary representation of a fully functional neural network, is discussed.

4.2.3 Perceptrons

First introduced by Rosenblatt (1958) as a hypothetical nervous system for machines designed to classify visual inputs, a perceptron is a neural network unit that performs certain computations to detect features in the input data to deliver an output. Perceptrons are viewed as the building blocks of modern artificial neural networks and consist of five distinct components (Basheer & Hajmeer, 2000; Minsky & Papert, 1969):

1. Input values: Features of the data are taken as input values by the perceptron;

2. Weights: Values that are calculated during the training of the model. The weights are given an initial value and this value is updated iteratively to reduce the training error;

3. Bias: A particular input type that introduces another level of dimensionality to the perceptron. A bias allows for higher quality and faster model training;

4. Weighted summation: Computed by multiplying every feature or input value associated with the corresponding weight values; and

5. Activation function: A function that is added to make a dataset easier to classify by limiting the range of the output produced by an artificial neuron. Their primary goal is to transform an ANN node's input signal into an output signal. Activation functions are further discussed in Section 4.2.4.

The original perceptron model, as proposed by Rosenblatt (1958), was an architecture for the classification of an input into a binary output. In this section, the architectures of two perceptron models, namely the single-layer and multilayer perceptrons, are further discussed. These architectures are currently still used and can classify an input into various output value types.

4.2.3.1 Single-layer perceptron model

As depicted in Figure 4-4, a single-layer perceptron represents the most straightforward artificial neural network implementation.

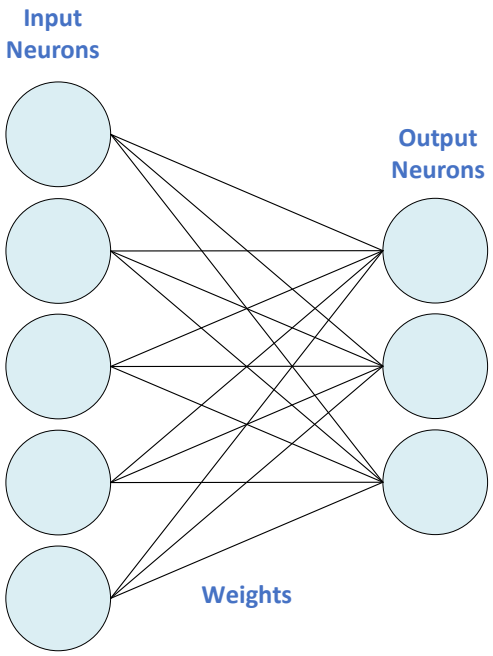


Figure 4-4: Single-layer perceptron (adapted from Rosenblatt, 1958)

The single-layer perceptron from Figure 4-4 has two layers: an input layer and an output layer. Spheres constitute the computational units or neurons, which are interconnected by trainable weights that represent synaptic connections. Rosenblatt (1958) proposed the original perceptron consisting of a single layer of input neurons fully interconnected in a feedforward way to a layer of output neurons.

In supervised learning, discussed in Section 4.3.2.3.1, the perceptron's output is then compared to a known class of a training case, and the weights and bias can be further modified to reduce the model's error rate. Unfortunately, a single-layer perceptron neural network's computational power is limited and is only suitable for classification tasks where the input space is linearly separable (Minsky & Papert, 1969). Nevertheless, a single-layer perceptron can solve some elementary learning tasks. Yet the power of neural networks is realised when many of them are connected in a multilayered architecture, as discussed in the next section.

4.2.3.2 Multilayer perceptron model

As depicted in Figure 4-5, the multilayer perceptron is a layered feedforward artificial neural network, based on Rosenblatt's 1958 perceptron model (Minsky & Papert, 1969; Rosenblatt, 1958). Its architecture comprises multiple perceptrons connected in two or more distinct layers (Ramchoun *et al.*, 2016).

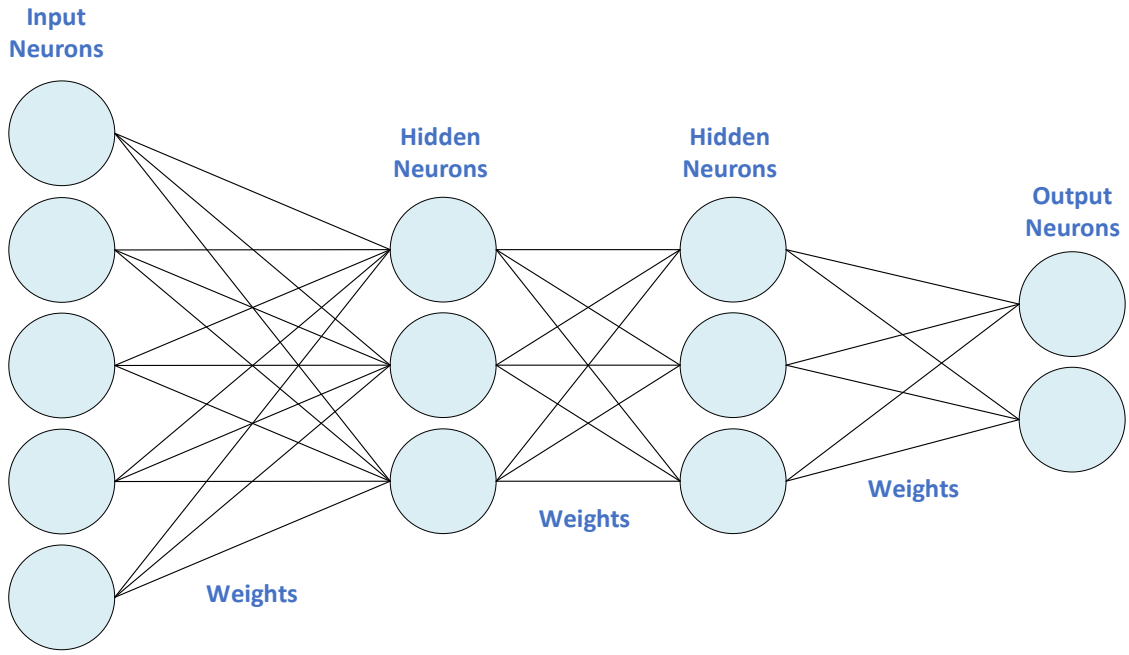


Figure 4-5: Multilayer perceptron (adapted from Minsky & Papert, 1969)

The multilayer perceptron from Figure 4-5 comprises three types of layers: an input layer that receives the signal, a hidden layer that performs the computations and an output layer that makes a decision or prediction about the input. This model is also referred to as a feedforward network, since the information flows from input to output in a unidirectional way, and nodes within the same layer are not interconnected (Popescu *et al.*, 2009). In the next section, the activation functions of a node, which defines the output of that node given an input or set of inputs, are addressed.

4.2.4 Activation functions

Activation functions are applied to artificial neural networks to aid them in learning complex patterns in data, subsequently improving their accuracy (Sharma & Athaiya, 2020). According to Sibi *et al.* (2013), the aim of an activation function in an artificial neural network is to transform a neuron's activation level in one layer into an output signal and to incorporate non-linearity into the neuron's output. The next layer of an artificial neural network then receives this output signal as input. In this section, three commonly used activation functions, namely the logistic sigmoid, hyperbolic tangent and the rectified linear unit are briefly discussed.

4.2.4.1 Logistic sigmoid

In neural networks, logistic functions are often used to incorporate nonlinearity in the model. Any change in the input is not directly proportional to changes in the output and the function compresses (smooths) the neural response signals into a specific interval (Jordan, 1995). The logistic sigmoid (Sigmoid) activation function takes a real value as input and outputs a value between 0 and 1 through a process known as normalisation (Jayalakshmi & Santhakumaran, 2011). This function is represented by the equation

$$f(x) = \frac{1}{1 + e^{-x}}, \tag{4-5}$$

where x represents the net input. The function is graphically depicted in Figure 4-6.

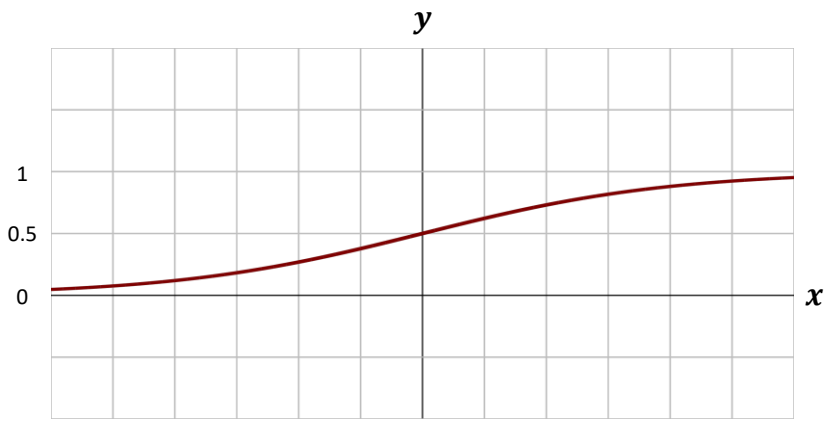


Figure 4-6: Logistic sigmoid activation function

Normalising the input value makes the Sigmoid activation function ideal for supervised classification problems (Özkan & Erbek, 2003). Unfortunately, the Sigmoid activation function suffers from neuron saturation (Glorot & Bengio, 2010). This implies that once the function reaches its peak value, either maximum or minimum, increasing the function's absolute input value would no longer result in a meaningful improvement in its output. Additionally, the Sigmoid activation function requires significant computational power (Mercioni & Holban, 2020).

4.2.4.2 Hyperbolic tangent

The hyperbolic tangent (Tanh) activation function is a shifted variation of the logistic sigmoid that takes a real value as input and produces a scaled output value between -1 and 1 (Özkan & Erbek, 2003). This function is represented by the equation

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}, \tag{4-6}$$

where x represents the net input. The function is graphically depicted in Figure 4-7.

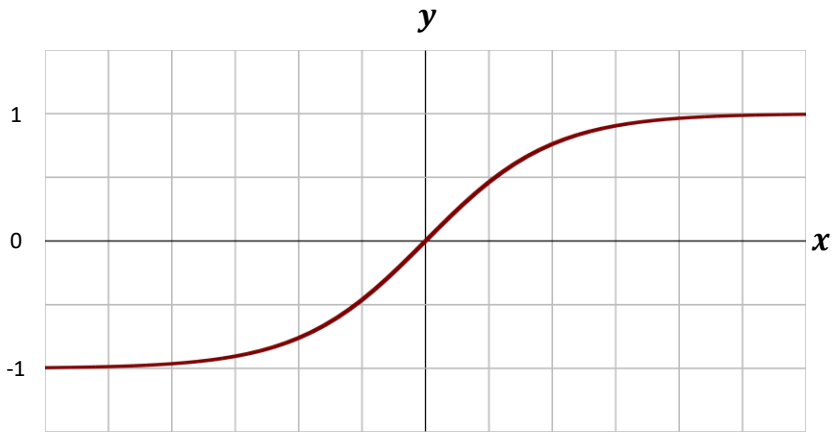


Figure 4-7: Hyperbolic tangent activation function

Like the Sigmoid activation function, the Tanh activation function is non-linear, making it suitable in artificial neural networks for classification problems (Farzad *et al.*, 2019). According to LeCun *et al.* (2012), artificial neural networks learn at a greater rate when their input and hidden layer computations are centred around zero; that is, by having an equal mass on both sides of the x-axis. The Tanh activation function is considered a zero-centred function, since it ranges from -1 to 1. Consequently, optimisation, using Tanh, is easier than when using the Sigmoid activation function (Wang *et al.*, 2020a). However, like the Sigmoid activation function, the Tanh activation function suffers from neuron saturation and higher than expected computational requirements (Tu, 1996).

4.2.4.3 Rectified linear unit

The rectified linear unit (ReLU) is widely considered in the literature as one of, if not the most successful, and commonly used activation functions in the field of deep learning (Banerjee *et al.*, 2019; Hanin, 2019; Jarrett *et al.*, 2009). The ReLU activation function was first introduced by Nair and Hinton (2010). According to Goodfellow *et al.* (2016), models trained using the ReLU activation function are relatively easy to optimise by applying the gradient-descent process, due to the activation function's linear characteristics. Input variables are subjected to a threshold operation by the ReLU activation function, where the function will output the input directly if it is positive. Otherwise, it will output a zero. This function is represented by the equation:

$$f(x) = max\,(0, x)$$

$$= \begin{cases} 0 & for\ x \leq 0 \\ x & for\ x > 0 \end{cases}, \qquad (4\text{-}7)$$

where x represents the net input. The function is graphically depicted in Figure 4-8:

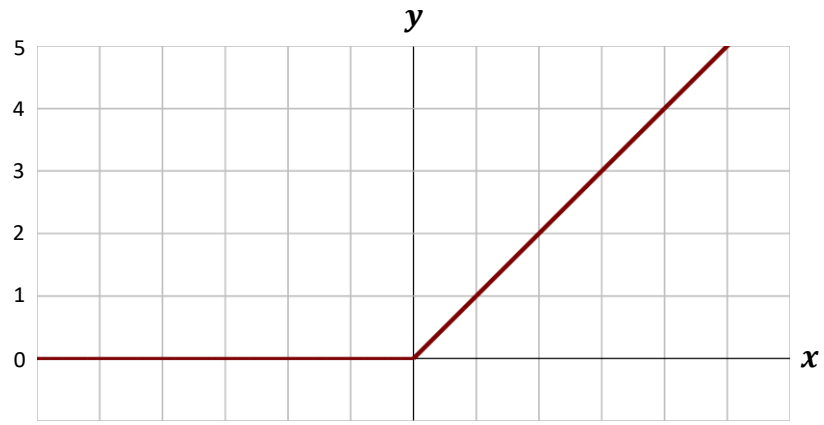


Figure 4-8: Rectified linear unit activation function

As shown in Figure 4-8, the ReLU activation function outputs non-negative values represented by the red line.

Since the left-hand limit and the right-hand limit are not equal, the ReLU activation function is not differentiable at $x = 0$ (Goel *et al.*, 2017). According to Ding *et al.* (2018), the derivative of the ReLU activation function makes it a left-hard saturated function having limited upper and lower bounds. While the ReLU activation function remains the most widely used in deep learning due to its robustness and performance, it is not without its flaws (Croce *et al.*, 2019; Hein *et al.*, 2019; LeCun *et al.*, 2015). For activations of the ReLU function in the region $x \leq 0$, the gradient will be 0, and as a result, the relative weights cannot be changed during descent. This results in the possibility of creating dying neurons, also known as the dying ReLU problem. Consequently, the network can generate regions where neurons do not initialise and remain permanently deactivated, resulting in an output of 0 (Chen *et al.*, 2017). In the next section, deep learning which forms the basis of the two intrusion detection models developed in this study, will be discussed.

4.3 Deep learning

According to Mohri *et al.* (2012), machine learning is a general term that refers to computational methods that rely on prior experience to increase efficiency or make valid inferences. Thus the term experience refers to past information available to the learner. In recent years, deep learning, a subset of machine learning, has emerged as the technique of choice for developing practical applications for text classification (Semberecki & Maciejewski, 2017), computer vision (Shin *et al.*,

2016), speech recognition (Graves *et al.*, 2013), natural language processing (LeCun *et al.*, 2015), autonomous vehicles (Chen *et al.*, 2015), and intrusion detection (Javaid *et al.*, 2016), in the artificial intelligence domain.

The ability of traditional machine learning approaches to process natural data in its raw form is significantly constrained by the number of processing layers used in the architecture of artificial neural networks (Goodfellow *et al.*, 2016). These early machine learning models, also known as shallow networks, contained no more than two hidden layers and an input layer (Deng & Yu, 2014). Although shallow networks are good at solving basic machine learning problems, they cannot keep up with more complicated tasks attributed to processing large amounts of unstructured data (Karatas *et al.*, 2018).

The multilayer perceptron introduced in Section 4.2.3.2 paved the way for further discussions into the field of deep learning. In the next section, the fundamentals of deep learning, specifically its definition and the three components required to teach machines, are discussed. Backpropagation, a popular training technique used in deep learning models, is also detailed. Finally, the deep learning model adopted for this study, namely an autoencoder, is presented.

4.3.1 Definition

Deep learning differs from traditional machine learning in the types of data it uses and the learning techniques it employs (Chauhan & Singh, 2018). According to Han *et al.* (2011), deep learning is a computational architecture that learns from data by integrating multiple processing layers, such as input, hidden, and output layers. While traditional machine learning approaches leverage structured, labelled data to make predictions, deep learning approaches can ingest and process vast amounts of unstructured data. According to Sarker *et al.* (2020), a vital advantage of deep learning over traditional machine learning approaches is that deep learning performs better in various circumstances, particularly when learning from large datasets.

4.3.2 Components

While the field of machine learning is dependent on three main components that facilitate the learning process, namely the data (or dataset), features and algorithms (Flach, 2012), deep learning describes a group of learning algorithms rather than a single method that can be used to learn complex prediction models (LeCun *et al.*, 2015). Since deep learning is a subset of machine learning, the three components required to train a deep neural network are briefly discussed next.

4.3.2.1 Dataset

Machines learn through observing information stored in a dataset (Dey, 2016). The Cambridge Dictionary of English grammar defines a dataset as "*a collection of separate sets of information that is treated as a single unit by a computer"* (McIntosh, 2013). A dataset has both rows and columns, with each row containing one observation. This observation can be an image, audio clip, text, or other vectorised representation. Moreover, deep neural networks commonly utilise the following data types (Patterson & Gibson, 2017):

- Numerical/binary data: Numerical data, or quantitative data, is any form of measurable data that has mathematical significance like simply raw numbers, e.g. the duration of time it took a computer to boot up in seconds;

- Categorical data: Categorical data is non-numerical data sorted by defining characteristics, e.g. gender, race, age or educational level of a person;

- Time-series data: Time-series data consists of data points that are indexed at specific points in time. Unlike numerical data, time-series data has established start and endpoints in time, e.g. server metrics, network data, sensor data or system events; and

- Textual data: Textual (text) data is a combination of words, sentences, or paragraphs that can take many dimensions, e.g. a page could have a hundred individual words, but only a dozen sentences or phrases.

Most machine learning models utilise two types of dataset: training data and testing data (Kotsiantis *et al.*, 2007). All models learn some kind of patterns from the training dataset and apply these patterns to the test dataset for prediction or classification. According to Russell and Norvig (2016), the training dataset used to fit the model may be further divided into a training set and a validation set. These three datasets are described below:

1. Training data: A training dataset is needed for neural networks and other artificial intelligence models to benchmark further application and usage. This dataset serves as the foundation for the program's growing library of data. The training dataset must be accurately labelled before the model can process and learn from it;

2. Validation data: A validation dataset is a subset of the training dataset, formed by separating the training dataset into two distinct parts. This set is used to derive an early estimate of the model's performance. Validation data eliminates bias when evaluating the performance of a model that was fit (trained) on the training dataset; and

3. Testing data: A testing dataset is often derived from the same overall dataset procured for the model. The testing dataset should be treated like a new dataset that should never be used during the model training. Since the testing dataset is not utilised during the training process, it provides an unbiased conclusion of the model's performance.

A deep learning model is generally evaluated on training and validation datasets and then measured on a testing dataset after being fit, respectively. In the next section, features, the second component required to train a deep neural network, are discussed.

4.3.2.2 Features

A feature is an individual measurable occurrence or characteristic of a particular observed problem (Bishop, 2006). In deep learning, features are individual, independent variables that become the input for the model. The model uses the characteristics of these inputs to learn and make predictions. Thus, features are the basic building blocks of datasets and the quality of these features significantly impacts the model's performance.

As described in the previous section, a dataset is a collection of rows and columns. Each column in a dataset represents a unique feature (or variable) for the model to observe (Ali & Yaman, 2013). According to Blum and Langley (1997), the number of features in a dataset defines its dimensionality (vector size) and directly impacts the model's training performance. However, since not all features in a dataset necessarily contribute to the model's ability to learn, feature selection has been proposed in the literature to eliminate irrelevant and redundant features from the dataset which are a burden on the already challenging task of prediction (Li *et al.*, 2017b; Chandrashekar & Sahin, 2014; Guyon & Elisseeff, 2003; Kira & Rendell, 1992).

Features are an integral part of a good deep learning model. The goal of feature selection is to select a subset of variables from the input that can accurately represent the data while reducing the effects from irrelevant variables and still achieving good prediction results. Any given problem can be solved differently. Datasets and features, the first two components required to train a deep neural network, culminate in selecting the right method to solve a problem. In the next section, the most common methods of solving a deep learning problem (also known as learning algorithms) are discussed.

4.3.2.3 Learning algorithms

According to Bonaccorso (2017), learning is characterised as adapting in response to external stimuli while still remembering the majority of previous observations. Thus, learning does not necessarily involve cognition, but is the process of finding statistical regularities or other patterns

in the data (Ayodele, 2010). According to Jordan and Mitchell (2015:255), learning is defined as "*the process of improving some measure of performance when executing some task, through some type of training experience*." Deep learning aims to learn from data, and machines can be trained, using different learning algorithms. These learning algorithms can be classified into several categories, based on their learning objectives (Ayodele, 2010). In this section, the most common learning algorithms, namely supervised learning, unsupervised learning, semi-supervised learning, self-supervised learning and reinforcement learning, are briefly discussed.

4.3.2.3.1 Supervised learning

Supervised learning is the most commonly studied and applied learning algorithm for classification, regression and probability estimation problems (Osisanwo *et al.*, 2017; Ayodele, 2010). In supervised learning, the model receives a set of labelled examples as input (training dataset), learns some kind of patterns from the training dataset and makes predictions for all unseen data points in the testing dataset (Kotsiantis *et al.*, 2007). The success of such a model is measured by its error rate (Bonaccorso, 2017).

4.3.2.3.2 Unsupervised learning

Unlike supervised learning, unsupervised learning algorithms do not rely on a set of labelled training data to model patterns in the input (Sathya & Abraham, 2013). Instead, unsupervised learning explores how models can learn to interpret specific input patterns in a way that represents the overall statistical nature of the input dataset (Dayan *et al.*, 1999). The most common unsupervised learning problem is clustering. This is the process of forming clusters or groups between and among the objects in an area to identify similarities that can be used to classify unknowns in the dataset.

4.3.2.3.3 Semi-supervised learning

Semi-supervised learning studies how to improve classification algorithms, using the power of both supervised and unsupervised learning techniques (Weston *et al.*, 2012). Semi-supervised models are trained, using a small subset of labelled training data and a larger subset of unlabelled training data (Russell & Norvig, 2016). This enables a semi-supervised model to learn insights about the data without requiring as much labelled data as a supervised learning model whilst still being able to deduce patterns in the unlabelled dataset like an unsupervised model.

4.3.2.3.4 Self-supervised learning

Self-supervised learning, a subset of the unsupervised learning paradigm, endeavours to get machines to derive supervision features from the data itself without involving humans (Jing & Tian, 2021). While conventional supervised learning methods rely on the availability of annotated training data, self-supervised learning models can generate labels for unlabelled data to train an unsupervised dataset in a supervised manner (Jaiswal *et al.*, 2021).

4.3.2.3.5 Reinforcement learning

Reinforcement learning algorithms are used when not enough input data can solve a problem (Bonaccorso, 2017). Reinforcement learning trains a model, using a series of delayed rewards or punishments (Russell & Norvig, 2016). Every action has some impact on the environment, and the environment provides feedback that guides the learning algorithm. Reinforcement learning is an algorithm in which the model makes decisions in the environment, based on the steps taken to maximise the reward (Mohri *et al.*, 2012).

In the next section, the backpropagation training algorithm is examined.

4.3.3 Backpropagation training

Backpropagation training is an algorithm widely used for training multilayer perceptron (MLP) neural networks (Han *et al.*, 2011). It is a technique used for fine-tuning the parameters of a neural network by using the error rate from the previous epoch (iteration). In fine-tuning the parameters, the error rate can be reduced, and the model's generalisation improved, making it more accurate.

The backpropagation algorithm (Demuth *et al.*, 2014) receives a set of input values from the dataset, denoting the desired behaviour and outcome of the MLP. This set of values can be represented as follows:

$$\{\boldsymbol{p}_1, \boldsymbol{t}_1\}, \{\boldsymbol{p}_2, \boldsymbol{t}_2\}, \ldots, \{\boldsymbol{p}_Q, \boldsymbol{t}_Q\}, \tag{4-8}$$

where $\boldsymbol{p}_q$ denotes to the input value and $\boldsymbol{t}_q$ refers to the corresponding target output. As each input is applied to the artificial neural network, the mean squared error (MSE) of the model can be calculated. The MSE refers to the average squared difference between the estimated values and the target value or the degree of similarity or, conversely, the level of error between them (Wang & Bovik, 2009). The MSE metric is used with backpropagation training to assess the difference between network output data and target output data. The MSE can be minimised by

adjusting the weight and bias of the inputs, resulting in more precise classification or prediction outcomes.

One layer's output, which serves as the input of the subsequent layer, can be depicted as

$$\boldsymbol{a}^{m+1} = \boldsymbol{f}^{m+1}(\boldsymbol{W}^{m+1}\boldsymbol{a}^{m} + \boldsymbol{b}^{m+1}) \text{ for } m = 0,1,\dots,M-1, \tag{4-9}$$

where M indicates the number of layers within the MLP, and $\boldsymbol{W}^{m+1}$ and $\boldsymbol{b}^{m+1}$ refer to the weight matrix and bias of layer $m+1$, respectively. The first layer of the network will receive inputs from some external stimuli and is the following starting point for the equation above:

$$\boldsymbol{a}^{0} = \boldsymbol{p}. \tag{4-10}$$

The output of the MLP is equivalent to the last layer's output and is represented by

$$\boldsymbol{a} = \boldsymbol{a}^{M}. \tag{4-11}$$

The following equation determines the MSE for a network that has only a single output value:

$$F(\boldsymbol{x}) = E[e^2] = E[(t-a)^2], \tag{4-12}$$

where $\boldsymbol{x}$ represents the vector of weights and biases of the MLP, and E refers to the expected value. This can be further generalised for networks with multiple output values to

$$F(\boldsymbol{x}) = E[\boldsymbol{e}^T\boldsymbol{e}] = E[(\boldsymbol{t}-\boldsymbol{a})^T(\boldsymbol{t}-\boldsymbol{a})]. \tag{4-13}$$

The MSE can be approximated by replacing the expected value for the squared error with the squared error at iteration k, resulting in the following equation:

$$\hat{F}(\boldsymbol{x}) = (\boldsymbol{t}(k)-\boldsymbol{a}(k))^T(\boldsymbol{t}(k)-\boldsymbol{a}(k)) = \boldsymbol{e}^T(k)\boldsymbol{e}(k). \tag{4-14}$$

The weights ($w_{i,j}^{m}$) and biases (b_i^m) can then be adjusted, based on the MSE by calculating the stochastic gradient descent:

$$w_{i,j}^{m}(k+1) = w_{i,j}^{m}(k) - a\frac{\partial \hat{F}}{\partial w_{i,j}^{m}}, \tag{4-15}$$

$$b_i^m(k+1) = b_i^m(k) - a\frac{\partial \hat{F}}{\partial b_i^m}, \tag{4-16}$$

where a denotes the learning rate of the network. Optimisation algorithms, such as stochastic gradient descent use derivates to determine whether to increase or decrease the weights to reduce the MSE (Wang *et al.*, 2017). These derivatives used can be calculated, using the chain rule as follows:

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = \frac{\partial \hat{F}}{\partial n_i^m} * \frac{\partial n_i^m}{\partial w_{i,j}^m}, \tag{4-17}$$

$$\frac{\partial \hat{F}}{\partial b_i^m} = \frac{\partial \hat{F}}{\partial n_i^m} * \frac{\partial n_i^m}{\partial b_i^m}, \tag{4-18}$$

where n_i^m, or the net input for layer m and its first derivatives are defined by

$$n_i^m = \sum_{j=1}^{S^{m-1}} w_{i,j}^m a_j^m + b_i^m, \tag{4-19}$$

$$\frac{\partial n_i^m}{\partial w_{i,j}^m} = a_j^{m-1}, \tag{4-20}$$

$$\frac{\partial n_i^m}{\partial b_i^m} = 1. \tag{4-21}$$

The sensitivity of $\hat{F}$ to changes in the ith element of the net input at layer m, can be defined as

$$S_i^m \equiv \frac{\partial \hat{F}}{\partial n_i^m}, \tag{4-22}$$

which leads to the simplification of the derivatives

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = s_i^m a_j^{m-1} \text{ and} \tag{4-23}$$

$$\frac{\partial \hat{F}}{\partial b_i^m} = s_i^m. \tag{4-24}$$

The stochastic gradient descent algorithm can then be approximated as

$$w_{i,j}^m(k+1) = w_{i,j}^m(k) - a s_i^m a_j^{m-1} \text{ and} \tag{4-25}$$

$$b_i^m(k+1) = b_i^m(k) - a s_i^m. \tag{4-26}$$

These equations are better expressed in matrix form as follows:

$$\boldsymbol{W}^m(k+1) = \boldsymbol{W}^m(k) - a\boldsymbol{s}^m(\boldsymbol{a}^{m-1})^T \text{ and} \tag{4-27}$$

$$\boldsymbol{b}^m(k+1) = \boldsymbol{b}^m(k) - a\boldsymbol{s}^m, \tag{4-28}$$

where

$$\boldsymbol{s}^m \equiv \frac{\partial \hat{F}}{\partial \boldsymbol{n}^m} = \begin{bmatrix} \frac{\partial \hat{F}}{\partial n_1^m} \\ \frac{\partial \hat{F}}{\partial n_2^m} \\ \vdots \\ \frac{\partial \hat{F}}{\partial n_{S^m}^m} \end{bmatrix}. \tag{4-29}$$

However, before the gradient can be determined, the sensitivities ($\boldsymbol{s}^m$) must be calculated and backpropagated. The sensitivity at layer m is calculated, using the sensitivity at layer $m+1$. The following Jacobian matrix is used to derive the sensitivities' recurrence relationships:

$$\frac{\partial n^{m+1}}{\partial n^m} \equiv \begin{bmatrix} \frac{\partial n_1^{m+1}}{\partial n_1^m} & \frac{\partial n_1^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_1^{m+1}}{\partial n_{S^m}^m} \\ \frac{\partial n_2^{m+1}}{\partial n_1^m} & \frac{\partial n_2^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_2^{m+1}}{\partial n_{S^m}^m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_1^m} & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_{S^m}^m} \end{bmatrix}, \tag{4-30}$$

where the i, j element of the matrix is calculated by

$$\begin{aligned} \frac{\partial n_i^{m+1}}{\partial n_j^m} &= \frac{\partial\left(\sum_{l=1}^{S^m} w_{i,l}^{m+1} a_l^m + b_i^{m+1}\right)}{\partial n_j^m} \\ &= w_{i,j}^{m+1} \frac{\partial a_j^m}{\partial n_j^m} \\ &= w_{i,j}^{m+1} \frac{\partial f^m\left(n_j^m\right)}{\partial n_j^m} \\ &= w_{i,j}^{m+1} f'^m\left(n_j^m\right), \end{aligned} \tag{4-31}$$

and

$$f'^m\left(n_j^m\right) = \frac{\partial f^m\left(n_j^m\right)}{\partial n_j^m}. \tag{4-32}$$

The Jacobian matrix can, therefore, be simplified to:

$$\frac{\partial \boldsymbol{n}^{m+1}}{\partial \boldsymbol{n}^{m}} = \boldsymbol{W}^{m+1}\boldsymbol{F}'^{m}(\boldsymbol{n}^{m}) \,, \tag{4-33}$$

where

$$\boldsymbol{F}'^{m}(\boldsymbol{n}^{m}) = \begin{bmatrix} f'^{m}(n_1^m) & 0 & \cdots & 0 \\ 0 & f'^{m}(n_2^m) & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & f'^{m}(n_{S^m}^m) \end{bmatrix}. \tag{4-34}$$

The recurrence relation of the sensitivity can be depicted in matrix form using the chain rule as

$$\boldsymbol{s}^{m} = \frac{\partial \hat{\boldsymbol{F}}}{\partial \boldsymbol{n}^{m}}$$

$$= \left(\frac{\partial \boldsymbol{n}^{m+1}}{\partial \boldsymbol{n}^{m}}\right)^{T} \frac{\partial \hat{\boldsymbol{F}}}{\partial \boldsymbol{n}^{m+1}}$$

$$= \boldsymbol{F}'^{m}(\boldsymbol{n}^{m})(\boldsymbol{W}^{m+1})^{T} \frac{\partial \hat{\boldsymbol{F}}}{\partial \boldsymbol{n}^{m+1}}$$

$$= \boldsymbol{F}'^{m}(\boldsymbol{n}^{m})(\boldsymbol{W}^{m+1})^{T}\boldsymbol{s}^{m+1}. \tag{4-35}$$

These sensitivities can then be sent or propagated backwards through each layer of the artificial neural network, starting at the last layer until the first layer is reached. This process is, therefore, known as backpropagation and can be depicted as

$$\boldsymbol{s}^{M} \rightarrow \boldsymbol{s}^{M-1} \rightarrow \cdots \rightarrow \boldsymbol{s}^{2} \rightarrow \boldsymbol{s}^{1}. \tag{4-36}$$

The last step in the backpropagation process is to determine the starting point $\boldsymbol{s}^{M}$ for the recurrence relation. This information is gathered at the final layer

$$s_i^M = \frac{\partial \hat{\boldsymbol{F}}}{\partial n_i^M}$$

$$= \frac{\partial (\boldsymbol{t}-\boldsymbol{a})^{T}(\boldsymbol{t}-\boldsymbol{a})}{\partial n_i^M}$$

$$= \frac{\partial \Sigma_{j=1}^{S^M}(t_j - a_j)^2}{\partial n_i^M}$$

$$= -2(t_i - a_i)\frac{\partial a_i}{\partial n_i^M}, \tag{4-37}$$

since

$$\frac{\partial a_i}{\partial n_i^M} = \frac{\partial a_i^M}{\partial n_i^M} = \frac{\partial f^M(n_i^M)}{\partial n_i^M} = f'^M(n_i^M), \tag{4-38}$$

s_i^M can be written as

$$s_i^M = -2(t_i - a_i)f'^M(n_i^M). \tag{4-39}$$

Finally, it can be rewritten in matrix form as

$$s^M = -2\boldsymbol{F}'^M(\boldsymbol{n}^M)(\boldsymbol{t} - \boldsymbol{a}). \tag{4-40}$$

In summary, the backpropagation algorithm can be explained in three steps:

1. Propagate the input in a feed-forward manner through the network:

$$\boldsymbol{a}^{m+1} = \boldsymbol{f}^{m+1}(\boldsymbol{W}^{m+1}\boldsymbol{a}^m + \boldsymbol{b}^{m+1}) \text{ for } m = 0,1,\ldots,M-1, \tag{4-09}$$

$$\boldsymbol{a}^0 = \boldsymbol{p}, \text{ and} \tag{4-10}$$

$$\boldsymbol{a} = \boldsymbol{a}^M. \tag{4-11}$$

2. Propagate the sensitivities backwards through the network:

$$s^M = -2\boldsymbol{F}'^M(\boldsymbol{n}^M)(\boldsymbol{t} - \boldsymbol{a}), \text{ and} \tag{4-40}$$

$$s^M = \boldsymbol{F}'^m(\boldsymbol{n}^m)(\boldsymbol{W}^{m+1})^T\boldsymbol{s}^{m+1} \text{ for } m = M-1,\ldots,2,1. \tag{4-35}$$

3. Adjust the network's weights and biases by using the approximate stochastic gradient descent rule as follows:

$$\boldsymbol{W}^m(k+1) = \boldsymbol{W}^m(k) - a\boldsymbol{s}^m(\boldsymbol{a}^{m-1})^T, \tag{4-27}$$

$$\boldsymbol{b}^m(k+1) = \boldsymbol{b}^m(k) - a\boldsymbol{s}^m. \tag{4-28}$$

In the next section, the basis of the intrusion detection model used in this study, is discussed.

4.3.4 Autoencoders

Autoencoders are artificial neural networks that aim to copy their inputs to their outputs by compressing the input into a smaller representation and applying backpropagation to extrapolate the output (Goodfellow *et al.*, 2016).

4.3.4.1 Autoencoder architecture

According to Mac *et al.* (2018), the general architecture of an autoencoder is composed of three main components: an encoder, a decoder and a bottleneck, as depicted in Figure 4-9.

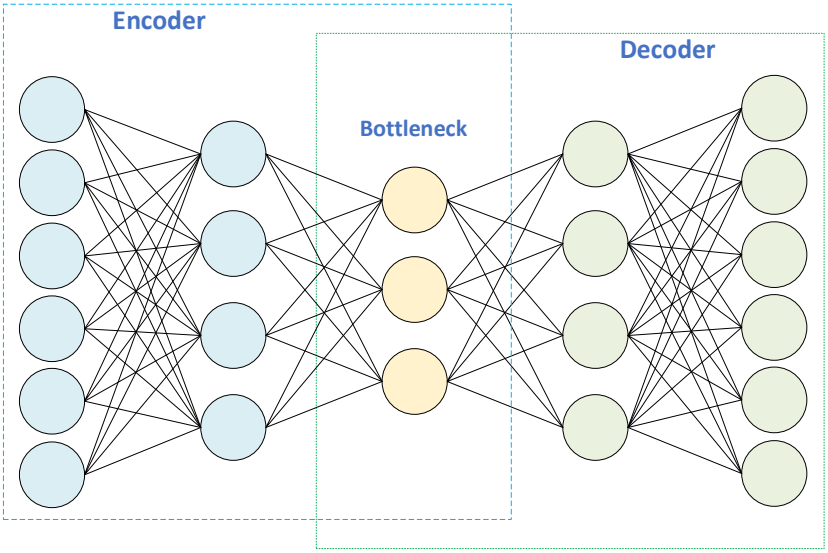


Figure 4-9: General architecture of an autoencoder

The autoencoder in Figure 4-9 consists of encoding and decoding components, which are symmetrical, multilayer feedforward neural networks. This type of autoencoder is considered to be a symmetrical autoencoder, since it has an equal number of layers and nodes in both the encoder and decoder components. Sun *et al.* (2016) argued that the number of encoder and decoder layers does not need to be symmetrical for the model to achieve a good representation of the input. Consequently, they proposed an unsymmetrical autoencoder structure, as illustrated in Figure 4-10.

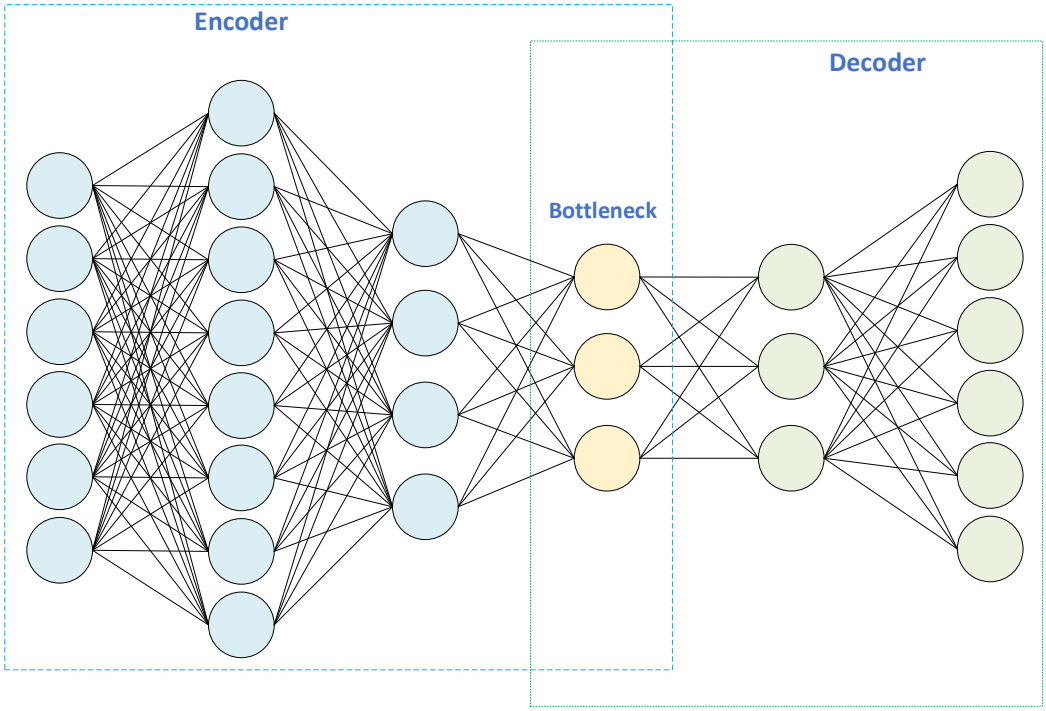


Figure 4-10: Unsymmetrical autoencoder

As shown in Figure 4-10, an unsymmetrical autoencoder is not bound by a symmetrical structure, i.e. an equal number of layers and nodes in the encoder and decoder components. However, since unsymmetrical autoencoders fall outside the scope of this study they will, therefore, not be discussed any further.

The encoder converts input data from a high-dimensional space to a compressed representation in a latent space known as the bottleneck. This compressed representation is then mapped to the decoder, which attempts to reconstruct the input, based on useful features extracted by the encoder during the training process (Zabalza *et al.*, 2016). The difference between the input and the output predicted by the autoencoder is referred to as the reconstruction error (Le *et al.*, 2018). Autoencoders are a subset of feedforward artificial neural networks. They can be trained using backpropagation and gradient descent techniques by adjusting the weights of the inputs to reduce the reconstruction error. This process can ultimately be quantified by the mean squared error or binary cross-entropy loss of the model (Goodfellow *et al.*, 2016).

Stacked (or deep) autoencoders, as depicted in Figure 4-11, can be characterised as an extension of a basic autoencoder, by introducing multiple layers between the encoder and decoder where the output of each hidden layer is connected to the input of the successive hidden layer (Zabalza *et al.*, 2016).

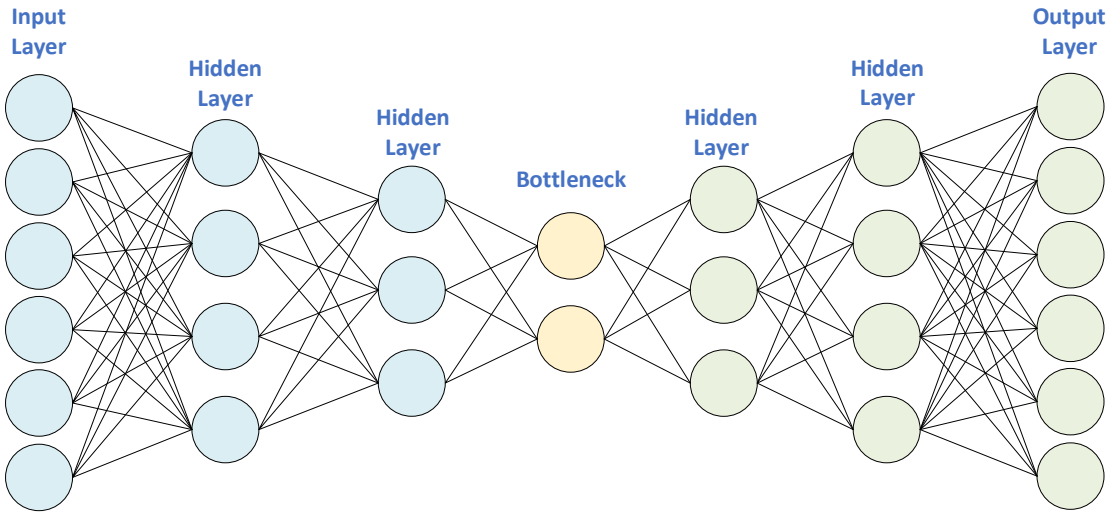


Figure 4-11: Deep autoencoder

Autoencoders are commonly applied in deep learning intrusion detection techniques due to their ability to extract significant levels of abstraction from complex input representations, making them ideal for predicting anomalies, based on similarities identified among features extracted from the inputs (Yousefi-Azar *et al.*, 2017). Deep autoencoders can learn low-dimensional representations even faster and with lower reconstruction error (Witten *et al.*, 2016). Deep symmetrical autoencoders will form the basis of the artificial neural network intrusion detection models constructed in this study. In the next section, autoencoder applications for intrusion detection are discussed.

4.3.4.2 Autoencoder applications for intrusion detection

Autoencoders were first introduced to the machine learning domain in the 1980s to address the problem of training a backpropagation algorithm in an unsupervised manner (Baldi, 2012). Deep autoencoders, which consist of multiple hidden layers, can be optimised, using feedback from an example by adding a supervised loss on the output layer of the model, leading to better generalisation. Generalisation, which refers to the model's ability to adapt to new, previously unseen data, is an important indicator of the model's prediction (reconstruction) capability (Le *et al.*, 2018).

Autoencoders can be used for dimensionality reduction applications to represent data with fewer dimensions while keeping maximum variance or minimising reconstruction loss (Hinton & Salakhutdinov, 2006). According to Zhou and Paffenroth (2017), deep autoencoders effectively detect non-linear characteristics across a wide range of domains, one such domain being

anomaly detection. This is further evidenced through research done by Naseer *et al.* (2018), who applied anomaly detection using a deep autoencoder to develop intrusion detection models. The authors concluded that deep learning is mainly used for feature learning in intrusion detection approaches, which reduces the complexity of the raw features of the dataset. Balın *et al.* (2019) proposed a concrete autoencoder that uses global feature selection to find a subset of the most informative features while concurrently training a neural network to rebuild the input data using the selected features. Dimensionality reduction is an important pre-training step when setting up a new artificial neural network model (Reddy *et al.*, 2020). Techniques, such as feature selection seek to solve two significant challenges by decreasing dimensionality, namely facilitating learning effective classifiers and discovering the most relevant features to better understand the problem the model is trying to address (Guyon & Elisseeff, 2003). According to Nilsson *et al.* (2007), selecting a smaller appropriate feature set from many inputs gives the best possible classification results.

Liu *et al.* (2020) proposed a scalable neural network-based hybrid intrusion detection system (IDS) framework with the combination of host IDS (HIDS) and Network IDS (NIDS), referred to as AlarmNet-IDS. In this framework, features from the original dataset are sent to an autoencoder for feature selection. The resulting features are then passed to a neural network-based decision engine for further processing. This implementation focused on combining autoencoders with deep learning methods to improve the performance of the intrusion detection model. The authors evaluated the performance of the model using the NSL-KDD dataset (UNB, 2009), an improved version of the KDD'99 dataset, which mainly removes some defects, such as redundant data and unbalanced data (Tavallaee *et al.*, 2009). The proposed model was able to achieve an accuracy of 81.04% for binary classification problems, with a precision score of 82.96%.

Al-Qatf *et al.* (2018) developed a self-taught learning based intrusion detection system (STL-IDS) deep learning model by combining a stacked autoencoder (SAE) and support vector machine (SVM) algorithm to automate intrusion detection. STL is a deep learning approach that uses unsupervised feature learning to learn a new representation from a dataset with no labels. Upon combining the new representation with labelled data, a supervised method, such as SVM can then be employed for a classification problem (Raina *et al.*, 2007). By combining deep and shallow learning techniques, the authors were able to leverage the strengths of both methods to reduce the training and testing times of the SVM while still achieving respectable performance results. The proposed STL-IDS model was evaluated, using the NSL-KDD dataset and obtained an accuracy of 84.96% and a precision rate of 96.23% for binary classification problems, respectively.

Wu *et al.* (2020) continued this trend by proposing a hybrid training model for intrusion detection that combines an SAE with an SVM to further improve deep learning training efficiency. The training model, referred to as JSAE-FSVM, utilised an SAE to perform feature dimension reduction to minimise the training time required by the model and ultimately solve the problem of large-scale data processing. The SVM utilised a kernel approximation algorithm to solve the scalability problem associated with traditional SVMs (Mu *et al.*, 2014), making it ideal for large scale deployments. The final model was evaluated using the NSL-KDD dataset and yielded favourable performance results for binary classification problems, achieving an accuracy of 83.8%, with a precision score of 89.6%.

According to Tang *et al.* (2020), existing intrusion detection methods have a high reliance on artificial feature extraction and a low level of precision. The authors proposed an intrusion detection model to combat this, known as SAAE-DNN, based on an SAE to represent the data in a latent layer. The model featured an attention mechanism that enables the network to obtain the key features of intrusion detection in real time (Du *et al.*, 2020) and a deep neural network (DNN) with trainable weights for classification purposes. The authors noted a correlation between the learning rate and the training accuracy of the SAAE-DNN model. Increasing or decreasing the learning rate resulted in better convergence when using the Adam optimiser. The best model evaluated on the NSL-KDD dataset was able to achieve an accuracy of 87.74% and a precision score of 86.47% on a binary classification problem.

The success of deep autoencoders in various classification and prediction problems and their efficiency when dealing with large abstract datasets through dimensionality reduction have motivated the adoption of deep autoencoders in this study to model intrusion detection, using supervised deep learning. The design of such a model is detailed in the next chapter. In the next section, the optimisation algorithm implemented to construct and select an accurate symmetrical autoencoder neural network architecture for this study is discussed.

4.4 Architecture optimisation algorithm

A neural network model's performance is sensitive to changes in its architecture (Lupo Pasini *et al.*, 2021). Moreover, a network architecture that is too simple may result in the model being unable to learn enough useful information from a dataset during the training process, leading to a scenario known as underfitting (Lemley *et al.*, 2017). During underfitting, the model is unable to appropriately capture the relationship between the input and output, resulting in a high error rate and the model being unable to generalise. Conversely, overfitting, a scenario that occurs when a model has too much capacity, can result from a network architecture that is too complex for the

problem it is built to address. Overfitting is an undesirable scenario that leads to a model performing well on training data but poorly on data that it has never seen before (Chauvin, 1990).

The challenge of determining the optimal architecture of artificial neural networks to improve efficiency and accuracy is critical in many applications (Eğrioğlu *et al.*, 2008). However, designing an optimal neural network architecture by hand requires a high level of time, knowledge and expertise in building neural networks (Huang *et al.*, 2018). A variety of approaches, including random search (Bergstra & Bengio, 2012), Bayesian optimisation (Mendoza *et al.*, 2016), the popular genetic algorithm (Miller *et al.*, 1989) as well as heuristic search algorithms, such as best-first search (Dechter & Pearl, 1985) have been proposed in the literature for neural architecture optimisation. For this study, an adapted version of the best-first search algorithm in combination with supervised learning techniques was implemented to find an accurate autoencoder neural network architecture for use in the experiments. Pseudocode for this algorithm is presented in Algorithm 4-1 and explained next.

Algorithm 4-1: Best-first search algorithm

Input: Let S be the minimum error score, and N be the maximum size of the population.

Output: A list containing the architectures and scores of trained and evaluated models, sorted from best to worst

```
model_architecture ← initial autoencoder;
model_score ← train_and_eval_autoencoder(model_architecture);
num_models ← 1;
list_of_models ← [model_architecture, model_score];
best_model ← model_architecture;
best_score ← model_score;
while (best_score > S) and (num_models < N) do
    model_architecture ← model_mutation(best_model);
    if model_architecture not in list_of_models do
        model_score ← train_and_eval_autoencoder(model_architecture);
        add [model_architecture, model_score] to list_of_models;
        num_models + 1;
        sort list_of_models from best to worst;
        if model_score is best score do
            best_model ← model_architecture;
            best_score ← model_score;
        end if
    end if
end while
return best_model and best_score
```

A flow diagram for the best-first search architecture optimisation algorithm developed in this study is provided in Annexure A and elaborated on in Chapter 5. An initial autoencoder model that can be constructed manually, or randomly, is trained and evaluated before being passed to the optimisation algorithm. This serves as a baseline for the best-first search algorithm. The algorithm is initialised by creating an empty list to store the population of evaluated models. Mutation procedures comprise modification of the dropout rate, number of layers, number of nodes and the batch size of the model. Dropout is an effective regularisation technique that reduces overfitting by removing units (both hidden and visible) from a neural network, causing the training process to become noisy and forcing nodes within a layer to learn a better representation from the inputs (Srivastava *et al.*, 2014). Batch size is a hyperparameter that defines the number of training

samples to work through before updating the artificial neural network model's parameters during every epoch (Radiuk, 2017). This step is followed by checking whether the model's new architecture already exists in the population of models before being trained and evaluated. Only new model architectures are trained and evaluated. The score (based on its error rate) and architecture of the new model are then added to the list containing the population of models as a list of objects. If the score of the new model is better than the current best performing model in the population, then the new model is set as the best model. The best-first search architecture optimisation algorithm is executed iteratively for as long as the current best performing model's error rate is greater than S and the population of models is less than N. These two parameters serve as the stopping criteria for the search. Once the stopping criteria have been satisfied, the best autoencoder model and its score are returned from the population of models. The best autoencoder model is then used to encode the input dataset which is subsequently passed to a multilayer perceptron (MLP) classifier as illustrated in Figure 4-12.

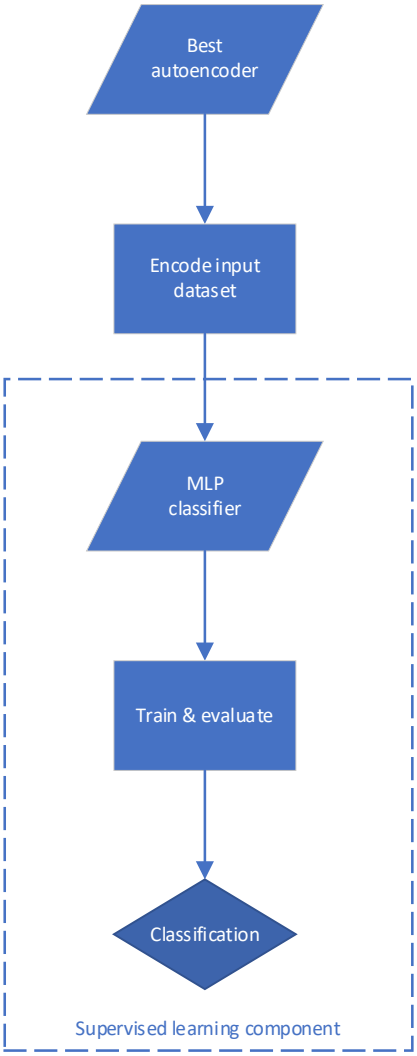


Figure 4-12: MLP classifier supervised learning component

The supervised learning component of the model, as depicted in Figure 4-12, comprises an MLP neural network that is manually constructed and trained on the encoded input dataset to classify network traffic as normal or malicious. The architecture of the MLP classifier is provided in Section 5.3.3.

4.5 Chapter summary

In this chapter, an overview of artificial intelligence and deep neural networks was provided through a literature review. This served as a basis for the architecture and training of multilayer perceptron artificial neural networks. The biological origin of neural networks, common implementations and various activation functions utilised by artificial neural networks were also discussed. A brief introduction to deep learning was then provided, followed by a discussion of the main components that facilitate the learning process, namely data, features and learning algorithms. The backpropagation training technique was also explained. Next, the architecture of an autoencoder model was presented and intrusion detection applications, based on autoencoders from the literature, were discussed. These discussions indicate that autoencoder-based models can be used for intrusion detection classification problems. Lastly, the best-first search architecture optimisation algorithm proposed for this study to construct an accurate symmetrical autoencoder was explained. In the next chapter, experiments performed using the autoencoder-based multilayer perceptron model to classify network intrusions will be discussed. The construction of the autoencoder component by the best-first search architecture optimisation algorithm, as well as results obtained, will also be addressed.

CHAPTER 5 EXPERIMENTAL DESIGN AND RESULTS

The premise of the research and an explanation of the background of the techniques used in this study were presented in the preceding four chapters. The purpose of Chapter 5 is to discuss how the relevant methods were used to construct an autoencoder-based intrusion detection system, using a best-first search neural network architecture optimisation algorithm. Since the premise of this study is to evaluate the feasibility of developing a best-first search architecture optimisation algorithm to automate the construction of an accurate symmetrical autoencoder model, used as part of an intrusion detection system, two experiments were performed in this chapter. Firstly, a baseline autoencoder-based model was manually constructed and its performance was evaluated to represent the reference point. Secondly, the best-first search architecture optimisation algorithm introduced in Chapter 4 was invoked to build an autoencoder-based intrusion detection system that will hopefully yield better performance than the baseline.

In this chapter, steps four, five and six of the design science research methodology are addressed by demonstrating the artefact's efficacy in solving the research problem, evaluating its performance and reporting on the overall results obtained from the proposed solution. The remainder of this chapter is structured as follows: In Section 5.1, the intrusion detection dataset utilised during this study is discussed. All pre-processing techniques applied to prepare the dataset for use are outlined in Section 5.2. Detail around the experimental design of the baseline model and the best-first search architecture optimisation model is provided in Section 5.3, followed by a brief discussion on the performance metrics used to evaluate each model. The experimental results are provided in Section 5.4. Thereafter, the results are discussed in Section 5.5. Finally, a summary of the chapter is provided in Section 5.6.

5.1 The NSL-KDD dataset

Researchers have widely employed the NSL-KDD dataset (UNB, 2009) to model intrusion detection in the literature (Ferrag *et al.*, 2020; Aloqaily *et al.*, 2019; Aljawarneh *et al.*, 2018; Potluri & Diedrich, 2016; Dhanabal & Shantharajah, 2015). This dataset addresses some of the inherent problems and redundancies identified in its predecessor, the KDD'99 dataset (Tavallaee *et al.*, 2009). While the dataset may not be an exact representation of real-world networks, it is still considered an effective benchmark to model intrusion detection, due to the scarcity (for privacy reasons) of available datasets for network-based intrusion detection systems (Shiravi *et al.*, 2012). An overview of the two NSL-KDD data files used for training and testing, when developing the two models in this study, is provided in Table 5-1.

Table 5-1: NSL-KDD data files

File Name	Description	Data Points	Normal	Malicious
KDDTrain+.TXT	The training set, including attack-type labels and difficulty level in CSV format	125 973	53%	47%
KDDTest+.TXT	The testing set, including attack-type labels and difficulty level in CSV format	22 544	43%	57%

Table 5-1 provides a description, the number of data points, and the ratio of normal to malicious network traffic in each file, respectively. The difficulty level mentioned in Table 5-1 refers to the severity and level of complexity of the specific traffic packet. Each data file from the NSL-KDD dataset consists of 43 features that represent the input and target for the model during the training and testing processes. These features are presented and described in Table 5-2 below.

Table 5-2: NSL-KDD feature descriptions

#	Feature Name	Description
1	*duration*	The duration of the connection in seconds
2	*protocol_type*	The connection's protocol
3	*service*	The destination network service that is used
4	*flag*	The connection's status and whether or not a flag has been raised, i.e. no reply, rejected or error
5	*src_bytes*	The number of data bytes transmitted from source to destination in a single connection

Table 5-2: NSL-KDD feature descriptions (continued)

#	Feature Name	Description
6	*dst_bytes*	The number of data bytes transmitted from destination to source in a single connection
7	*land*	If the source and destination IP addresses and port numbers are the same, this variable has the value 1; otherwise, it has the value 0
8	*wrong_fragment*	The total number of wrong fragments in this connection
9	*urgent*	The number of urgent packets in this connection. Urgent packets are packets with the urgent bit activated
10	*hot*	The number of "hot" indicators in the content such as: entering a system directory, creating programs and executing programs
11	*num_failed_logins*	Count of failed login attempts
12	*logged_in*	Login Status: a 1 if successfully logged in; 0 otherwise
13	*num_compromised*	The number of "compromised" conditions
14	*root_shell*	A value of 1 if the root shell is obtained; 0 otherwise
15	*su_attempted*	A value of 1 if the "su root" command is attempted or used; 0 otherwise

Table 5-2: NSL-KDD feature descriptions (continued)

#	Feature Name	Description
16	*num_root*	The number of "root" accesses or the number of operations performed as a root in the connection
17	*num_file_creations*	The number of file creation operations in the connection
18	*num_shells*	The number of shell prompts
19	*num_access_files*	The number of operations on access control files
20	*num_outbound_cmds*	The number of outbound commands in a File Transfer Protocol session
21	*is_host_login*	A value of 1 if the login belongs to the "host" list i.e., root or admin; else 0
22	*is_guest_login*	A value of 1 if the login is a "guest" login; 0 otherwise
23	*count*	The number of connections to the same destination host as the current connection in the past two seconds
24	*srv_count*	The number of connections to the same service (port number) as the current connection in the past two seconds
25	*serror_rate*	The percentage of connections that have activated the *flag* (4) *s0, s1, s2 or s3*, among the connections aggregated in *count* (23)

Table 5-2: NSL-KDD feature descriptions (continued)

#	Feature Name	Description
26	*srv_serror_rate*	The percentage of connections that have activated the *flag* (4) *s0, s1, s2 or s3*, among the connections aggregated in *srv_count* (24)
27	*rerror_rate*	Among the connections aggregated in *count* (23), the percentage of connections that activated the *flag* (4)*, REJ* (rejected)
28	*srv_rerror_rate*	The percentage of connections that have activated the *flag* (4)*, REJ* (rejected), among the connections aggregated in *srv_count* (24)
29	*same_srv_rate*	The percentage of connections that were to the same service, among the connections aggregated in *count* (23)
30	*diff_srv_rate*	The percentage of connections that were to different services, among the connections aggregated in *count* (23)
31	*srv_diff_host_rate*	The percentage of connections that were to different destination machines among the connections aggregated in *srv_count* (24)
32	*dst_host_count*	The number of connections having the same destination host IP address
33	*dst_host_srv_count*	The number of connections having the same port number
34	*dst_host_same_srv_rate*	The percentage of connections that were to the same services, among the connections aggregated in *dst_host_count* (32)
35	*dst_host_diff_srv_rate*	The percentage of connections that were to different services, among the connections aggregated in *dst_host_count* (32)

Table 5-2: NSL-KDD feature descriptions (continued)

#	Feature Name	Description
36	*dst_host_same_src_port_rate*	The percentage of connections that were to the same source port, among the connections aggregated in *dst_host_srv_count* (33)
37	*dst_host_srv_diff_host_rate*	The percentage of connections that were to different destination machines, among the connections aggregated in *dst_host_srv_count* (33)
38	*dst_host_serror_rate*	The percentage of connections that have activated the *flag* (4) *s0, s1, s2 or s3*, among the connections aggregated in *dst_host_count* (32)
39	*dst_host_srv_serror_rate*	The percentage of connections that have activated the *flag* (4) *s0, s1, s2 or s3*, among the connections aggregated in *dst_host_srv_count* (33)
40	*dst_host_rerror_rate*	The percentage of connections that have activated the *flag* (4)*, REJ* (rejected), among the connections aggregated in *dst_host_count* (32)
41	*dst_host_srv_rerror_rate*	The percentage of connections that have activated the *flag* (4)*, REJ* (rejected), among the connections aggregated in *dst_host_srv_count* (33)
42	*label*	Classification of the traffic input: *Normal* or *Malicious*
43	*score*	The severity and level of complexity of the specific traffic packet

The two autoencoder models utilise features 1 through 41 and feature 43 as inputs. The *label* (42) feature from Table 5-2 represents the binary class label or target for the model. The features of the NSL-KDD dataset can be further separated into three data types, as presented in Table 5-3.

Table 5-3: NSL-KDD feature types

Type	Features
Binary	*land* (7) *logged_in* (12) *root_shell* (14) *su_attempted* (15) *is_host_login* (21) *is_guest_login* (22) *label* (42)[2]
Categorical	*protocol_type* (2) *service* (3) *flag* (4)
Numeric	*duration* (1) *src_bytes* (5) *dst_bytes* (6) *wrong_fragment* (8) *urgent* (9) *hot* (10) *num_failed_logins* (11) *num_compromised* (13) *num_root* (16) *num_file_creations* (17) *num_shells* (18) *num_access_files* (19) *num_outbound_cmds* (20) *count* (23) *srv_count* (24) *serror_rate* (25) *srv_serror_rate* (26) *rerror_rate* (27) *srv_rerror_rate* (28) *same_srv_rate* (29) *diff_srv_rate* (30) *srv_diff_host_rate* (31) *dst_host_count* (32) *dst_host_srv_count* (33) *dst_host_same_srv_rate* (34) *dst_host_diff_srv_rate* (35) *dst_host_same_src_port_rate* (36) *dst_host_srv_diff_host_rate* (37) *dst_host_serror_rate* (38) *dst_host_srv_serror_rate* (39) *dst_host_rerror_rate* (40) *dst_host_srv_rerror_rate* (41) *score* (43)

[2] The *label* (42) feature represents the different classifications for the traffic input. Within this feature, there are four classes of attack: Denial of Service (DoS), Probe, User to Root (U2R), and Remote to Local (R2L). For the purposes of this study, however, the *label* (42) feature was converted to a binary value of either normal or malicious network traffic.

Most of the NSL-KDD dataset features shown in Table 5-3 have quantitative data types. However, *protocol_type* (2), *service* (3) and *flag* (4) are categorical. Neural networks, by design, operate on vectors of real numbers (Goodfellow *et al.*, 2016). Therefore, categorical data types need to be converted or encoded into real numbers before employing the autoencoder models in this study. This process, along with several other steps applied to prepare the data for use by the two models developed for this study, is discussed in the next section.

5.2 Pre-processing steps

According to Hamed *et al.* (2018), raw data should be subjected to some preliminary pre-processing steps. These steps are instituted to make the dataset more usable by a model and to improve the efficiency and accuracy of the training and testing processes. The pre-processing steps applied to the NSL-KDD dataset used in this study are summarised in Figure 5-1.

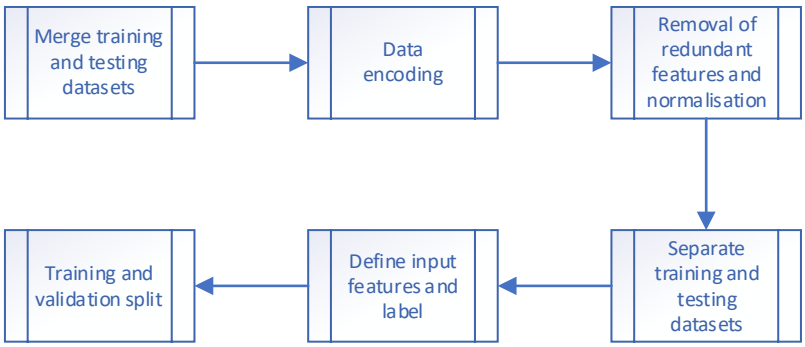


Figure 5-1: NSL-KDD pre-processing steps

The pre-processing steps illustrated in Figure 5-1 are discussed next.

5.2.1 Merge training and testing datasets

Due to the significant size of the dataset and the uniformity of the pre-processing steps applied to both training and testing data, the training and testing sets were combined to form a single dataset before the remainder of the pre-processing techniques were applied.

5.2.2 Data encoding

The categorical features identified in Section 5.1 were converted into real numbers before being applied to the model. This process is divided into output encoding and input encoding steps.

5.2.2.1 Output encoding

Since the model considered in this study performs binary classification to distinguish between normal and malicious network traffic, binary encoding was applied to the *label* (42) feature of the dataset. Consequently, normal traffic is represented by a 0 and all other traffic is characterised by a 1, respectively.

5.2.2.2 Input encoding

As mentioned in Section 5-1, the following features have categorical data types:

- *protocol_type* (2);
- *service* (3); and
- *flag* (4).

Therefore, these features were converted into one-hot encoded vectors, such that each unique attribute for these features is represented by a vector of 1s and 0s, respectively. Thus, for example, after applying categorical encoding, the *protocol_type* (2) feature which has three distinct values would be represented as indicated in Table 5-4.

Table 5-4: Categorical encoding example

Feature Name	Before Encoding	After Encoding
protocol_type (2)	TCP	(1,0,0)
	UDP	(0,1,0)
	ICMP	(0,0,1)

As illustrated in Table 5-4 above, categorical encoding converts a categorical feature's attributes from a string value to a numerical value. Thus, it can be applied to the inputs and included during the training process (Potdar *et al.*, 2017). Upon completion of the encoding procedures, the dataset's dimensionality increased from 42 to 122 input features.

5.2.3 Removal of redundant features and normalisation

Some dataset features may be more relevant than others, while others may fail to add value and are entirely redundant. Subsequently, irrelevant and redundant features should be eliminated from the dataset to prevent overfitting and reduce model training time (Saeys *et al.*, 2007). In addition, data normalisation is an essential pre-processing step that entails transforming or

scaling features into a similar range, such that the model is not biased by larger numeric feature values in the dataset (Singh & Singh, 2020).

In this study, the dataset was checked for uninformative or redundant features that do not benefit the training process. The *num_outbound_cmds* (20) feature is such an example, since it has a standard deviation of 0. In this instance, a standard deviation of 0 indicates the lack of variance in the field. Consequently, this feature was removed from the dataset as it adds no value to the model. Data normalisation was performed to transform the values of numeric features in the dataset to a similar scale without altering the data distribution or having to discard information (Microsoft, 2019). Finally, the remaining features in the dataset were scaled to a range between 0 and 1 to generalise the distribution of the source data.

5.2.4 Separate training and testing datasets

Following the data encoding, redundant feature removal and normalisation steps above, the merged dataset was separated into two independent datasets once again, namely training and testing.

5.2.5 Define input features and label

The input features were separated from the label for both the training and testing datasets to define the input and outputs for the model. After this, the input features and label were defined by instantiating them as disjoint data variables that can be used by the neural network model.

5.2.6 Training and validation split

Since the NSL-KDD dataset already consisted of independent training and testing sets, only an additional validation split was required (Choudhary & Kesswani, 2020). Therefore, in this study, the training dataset was further randomly separated into two parts, one for training and the other for validation purposes, using a ratio of 90% to 10%, respectively. This ratio is in line with other research on the NSL-KDD dataset (Song *et al.*, 2021; Beaugnon *et al.*, 2017). A summary of the sizes of the final datasets is provided in Table 5-5.

Table 5-5: Training, validation and testing datasets

#	Dataset	Number of Data Points	Number of Features
1	training	113 375	122
2	validation	12 598	122
3	testing	22 544	122

Following the aforementioned pre-processing steps, the original two parts of the NSL-KDD dataset had been prepared for use by the model, as shown in Table 5-5. These three datasets were jointly utilised in the experimentation that follows.

5.3 Experimentation

All experimentation was performed on a system with an Intel i7-7700K 4.2 GHz Quad-Core CPU, 16GB of DDR4 3200MHz memory and an NVIDIA GTX 1080Ti GPU to reduce the model training time. Ubuntu 20.04 LTS was selected as the operating system of choice. The baseline model was constructed and the architecture optimisation algorithm was developed and implemented, using a Jupyter notebook running Keras version 2.3.0 and Python 3.7.7 on a TensorFlow version 2.2.0 backend. The second symmetrical model was constructed, using the best-first search architecture optimisation algorithm introduced in Chapter 4 and the source code for it is included in Annexure B.

Before executing the architecture optimisation algorithm, the user can specify the minimum and maximum ranges for the following hyperparameters: the number of hidden layers, the number of nodes per hidden layer, the batch size and the drop-out rate of the models to be considered. Additionally, the early stopping criterion and the total number of models generated by the algorithm can be specified to suit the user's requirements.

Depending on the dataset size, the complexity of the models being constructed, and how quickly each model converges, the algorithm can automate the construction of a relatively large number of models in a reasonably short time. The algorithm returned the best performing symmetrical autoencoder model to be used in the experimentation that follows.

Experimentation was performed to determine whether the best-first search architecture optimisation algorithm could build a more accurate autoencoder-based intrusion detection model than the baseline. Both of the symmetrical autoencoder models constructed were used as input by a multilayer perceptron (MLP) model for intrusion detection classification. The details of these models are discussed next.

5.3.1 Construction of baseline model

The baseline model for this study was manually constructed upon performing a literature review of deep learning and autoencoders in Chapter 4. After selecting the hyperparameters for the model, the architecture presented in Figure 5-2 was chosen.

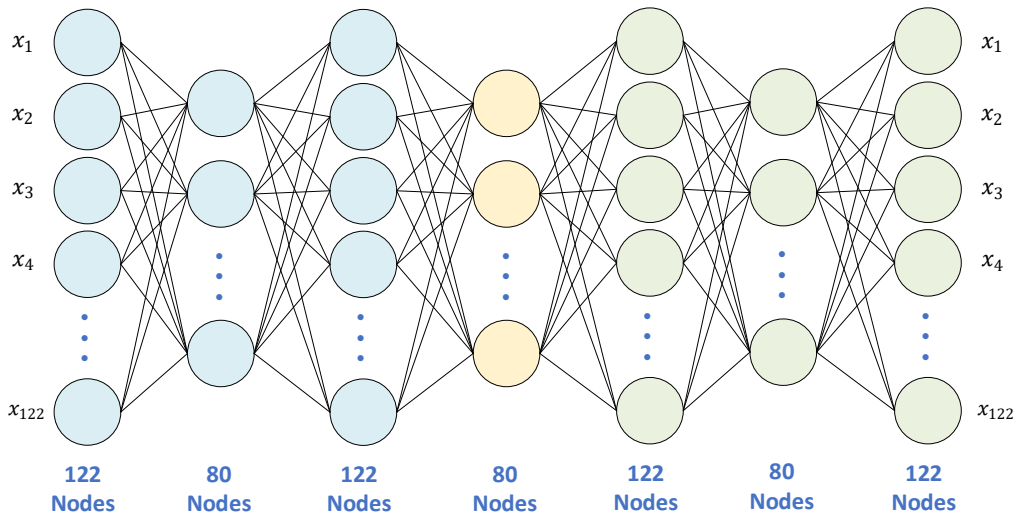


Figure 5-2: Architecture of the baseline autoencoder model

The symmetrical baseline autoencoder model depicted in Figure 5-2 consisted of seven layers. The input and output layers had 122 nodes, which corresponded with the number of input features in the dataset. In addition, there were two hidden layers in both the encoder and decoder components with 80 and 122 nodes, respectively. Lastly, the model had a bottleneck layer of 80 nodes. The ReLU activation function was employed for the hidden layers and the Sigmoid activation function for the output layer. No dropout layers were included in the baseline model, since overfitting did not occur. The model was trained for 136 epochs, using a batch size of 64 and a learning rate of 0.001. Furthermore, this model attained a final binary cross-entropy loss value of 0.0151.

5.3.2 Construction of best-first search optimisation model

The best-first search architecture optimisation algorithm introduced in Chapter 4 was executed to construct a total of 390 unique model architectures. The stopping criteria for the algorithm had the following values: $N = 390$ and $S = 10^{-6}$. It took approximately 128.25 hours to generate 390 candidate autoencoder models on the NSL-KDD dataset. During this process, each iteration involved mutating the current best performing model's hyperparameters, based on a predefined minimum and maximum range, before evaluating the new model on the validation dataset to determine if the performance of the best model found could be further improved. The hyperparameter ranges used by the algorithm are provided in Table 5-6.

Table 5-6: Hyperparameter ranges

Hyperparameter	Range
Layers	[3, 7]
Hidden nodes	[15, 122]
Batch size	[20, 122]
Drop-out rate	[5%, 50%]

A summary of the best performing symmetrical autoencoder model determined by the algorithm is provided in Figure 5-3.

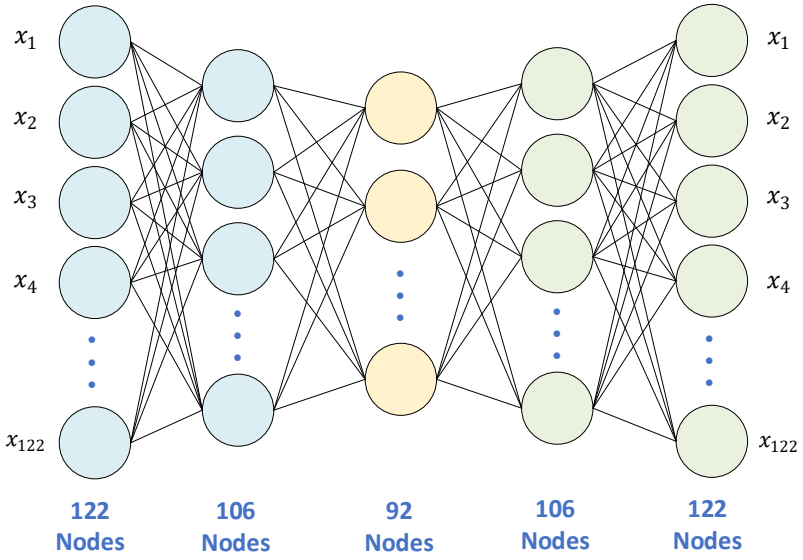


Figure 5-3: Architecture of the best-first search optimisation autoencoder model

As illustrated in Figure 5-3, the best symmetrical autoencoder model selected by the architecture optimisation algorithm consisted of five layers. Similar to the baseline model, the input and output layers of this model had a fixed number of nodes that corresponded with the number of input features in the dataset. In contrast, however, this model had only a single hidden layer for both the encoder and decoder components, consisting of 106 nodes. Finally, a bottleneck layer of 92 nodes was used. The hidden layers of this model implemented the ReLU activation function, while the output layer used the Sigmoid activation function. Additionally, this model had a drop-out rate of 5% applied to all hidden layers and was trained for 234 epochs, using a batch size of 78 and a learning rate of 0.001. Finally, this model attained a final binary cross-entropy loss value of 0.0149.

5.3.3 Construction of the MLP classifier model

Since intrusion detection is modelled as a binary classification problem in this study, a subsequent MLP classifier was manually constructed and trained on the encoded datasets obtained from the two autoencoder models discussed in Section 5.3.1 and Section 5.3.2. The hidden layers of this MLP classifier model were selected to be the same for both autoencoder models and its architecture is presented in Figure 5-4.

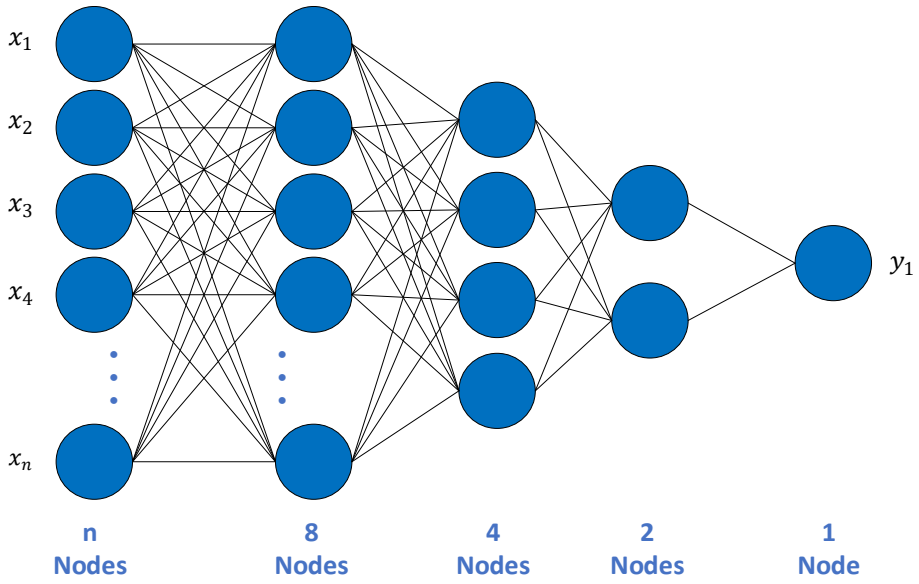


Figure 5-4: MLP classifier architecture

As shown in Figure 5-4, the MLP classifier had five layers, with the encoded dataset providing the input to the classifier. Therefore, the input layer of the MLP classifier had n nodes, representing the size of the output from the encoder. For the baseline model, n had a value of 80, and for the best model found by the best-first search optimisation algorithm, n had a value of 92. Three progressively smaller hidden layers were added to increase the complexity of the model, having 8, 4 and 2 nodes, respectively. Lastly, the output layer was constructed with a single node. The ReLU activation function was employed for the hidden layers of the MLP classifier, and the Sigmoid activation function was used in the output layer. No dropout was included in the MLP classifier, since overfitting did not occur. This model was trained for 83 epochs for the baseline model and 52 epochs for the best-first model until the accuracy of the model plateaued, using a fixed batch size of 64 and a learning rate of 0.0005 for both models. The performance metrics of the abovementioned models are discussed next.

5.3.4 Performance metrics

According to Kotu and Deshpande (2019), the performance of a classification model can be adequately summarised, using a confusion matrix, also known as a truth table. The performance of an intrusion detection classifier can be interpreted, using the following metrics (Hindy *et al.*, 2020; Ieracitano *et al.*, 2018):

- Accuracy: The classification model's ability to select or reject the correct class. Measured as the ratio of successfully classified items compared to the total number of elements evaluated by the model;
- Precision: The classification model's degree of relevance when labelling a class. This metric is measured as the proportion of correctly classified true positives in relation to the total number of predicted positive instances;
- Recall: The proportion of cases where a true positive class was correctly classified by the model; and
- F1 score: The weighted average of the true positive rate and precision of the model. An F1 score of 1 is considered perfect, whereas an F1 score of 0 is regarded as a failure.

An example of a binary classification confusion matrix is provided in Figure 5-5.

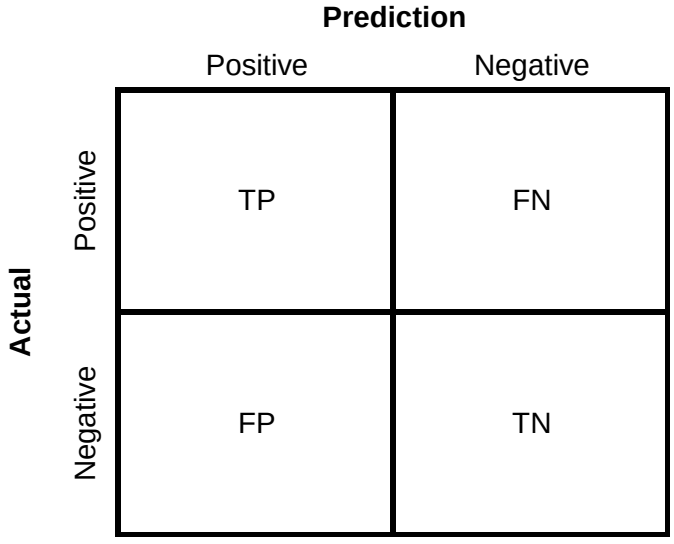


Figure 5-5: Confusion matrix for binary classification (adapted from Ali *et al*., 2019)

The performance metrics mentioned above can be derived, using four general terms provided in Figure 5-5, i.e. True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN), where

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}; \quad (5\text{-}1)$$

$$Precision = \frac{TP}{TP + FP}; \quad (5\text{-}2)$$

$$Recall = \frac{TP}{TP + FN}; \quad (5\text{-}3)$$

and

$$F1\ score = \frac{2 \times Precision \times Recall}{Precision + Recall}. \quad (5\text{-}4)$$

Ben-David (2008) argued that evaluating a classifier's performance by simply measuring the number of incorrectly classified instances can be misleading and that the cost of error must also be considered when evaluating the accuracy of a classifier. Cohen's Kappa statistic (Cohen, 1960), originally proposed as a measure of agreement among observers of psychological behaviour, has also been proposed to quantify the severity of incorrect classification results (Ben-David, 2008). The Kappa statistic considers random chance to assess how much better one classifier performs than another classifier that simply guesses at random, based on the frequency of each class (McHugh, 2012). Calculation of Kappa may be performed according to the following formula:

$$k = \frac{p_o - p_e}{1 - p_e}, \quad (5\text{-}5)$$

where p_o represents the observed accuracy of the model and p_e represents the expected accuracy (random chance) of the model. A Kappa score of 1 is considered perfect, whereas a Kappa score of 0 is considered a failure.

Another approach for visualising the performance of a classifier is by using a Receiver Operating Characteristic (ROC) curve to show the relationship between TP and FP instances (Bradley, 1997). The area under the ROC curve, also known as AUC, is an evaluation metric that represents the degree of separability, or how well a model can differentiate between classes (Ji *et al.*, 2018). An AUC score of 1 represents a model with excellent separability, while an AUC score of 0 indicates the inability to differentiate between two classes.

The consideration of these performance metrics in aggregate will form the basis for discussion in the next section, where the results of each model will be summarised and evaluated.

5.4 Results

Two experiments were conducted, using the NSL-KDD dataset discussed in Section 5.1. The first of these experiments was performed to establish a performance baseline for the study. This baseline classifier model was measured against model accuracy results found in the literature that also used the NSL-KDD dataset. The second experiment was performed using the best autoencoder model constructed, using the best-first search architecture optimisation algorithm discussed in Chapter 4, to determine if the baseline could be improved.

The results obtained by other autoencoder-based classifier models applied to the NSL-KDD dataset, as discussed in Chapter 4, are presented in Table 5-7. The binary classification accuracy scores of the four autoencoder-based models, namely 81.04%, 83.80%, 84.96% and 87.74% will be used as benchmarks to measure the performance of the baseline and best-first search optimisation classifier models.

Table 5-7: Summary of model accuracy results

Model	Source	Accuracy
Hybrid autoencoder-based IDS model with a combination of HIDS and NIDS referred to as AlarmNet-IDS	Liu *et al.* (2020)	81.04%
Hybrid autoencoder-based IDS model that combined an SAE for feature dimension reduction and an SVM to improve deep learning training efficiency referred to as JSAE-FSVM	Wu *et al.* (2020)	83.80%
Self-taught learning-based IDS deep learning model that combined an SAE and SVM algorithm to automate intrusion detection referred to as STL-IDS	Al-Qatf *et al.* (2018)	84.96%
Hybrid IDS model, based on an SAE and DNN with trainable weights referred to as SAAE-DNN	Tang *et al.* (2020)	87.74%

5.4.1 Performance of baseline model

The confusion matrix obtained by evaluating the baseline classifier model on the test dataset is provided in Table 5-8. In addition, specific performance metrics of the model, as identified in Section 5.3.4, are shown in Table 5-9. The learning curve for the baseline autoencoder model is presented in Figure 5-6, and the learning and performance curves for the baseline classifier model are presented in Figure 5-7, Figure 5-8 and Figure 5-9, respectively.

Table 5-8: Baseline classifier model confusion matrix

		Predicted traffic	
		Malicious	**Normal**
Actual traffic	**Malicious**	9 415	3 111
	Normal	296	9 722

Table 5-9: Baseline classifier model performance metrics

Accuracy	84.89%
Precision	96.95%
Recall	75.16%
F1 score	84.68%
Kappa	70.23%
AUC	96.46%

From Table 5-8, it can be observed that the baseline model classified 19 137 out of 22 544 cases of network traffic correctly. As presented in Table 5-9, the baseline model achieved an overall accuracy of 84.89% with a precision score of 96.95%. This accuracy metric is in line with the results achieved by other autoencoder-based classifiers in the literature, as mentioned in Table 5-7. Similarly, the high precision score seems to indicate that the baseline model is well equipped to correctly predict the malicious class on the NSL-KDD dataset. However, a recall score of 75.16%, which represents the model's ability to identify the number of instances of malicious traffic from the dataset, leaves some room for improvement. The baseline model further achieved an F1 score of 84.68%, a Kappa score of 70.23% and an AUC score of 96.46%, respectively. These three metrics are discussed next.

F1 is a measure of the proportion of true positives (malicious traffic correctly classified) that are influenced by false positives (malicious network traffic incorrectly classified as normal) and false negatives (normal network traffic classified as malicious). As a result, the baseline classifier model with an F1 score of 84.68% is capable of classifying malicious traffic with a low rate of false negatives and a similarly low percentage of false positives. A model's Kappa score (Cohen, 1960), can be interpreted as shown in Table 5-10 (McHugh, 2012):

Table 5-10: Interpretation of Kappa score

Kappa Score	Interpretation
0%	No agreement between the predicted and actual entities
1-20%	Slight agreement between the predicted and actual entities
21-40%	A fair agreement between the predicted and actual entities
41-60%	Moderate agreement between the predicted and actual entities
61-80%	Considerable agreement between the predicted and actual entities
81-100%	Almost perfect agreement between the predicted and actual entities

Therefore, based on the information provided in Table 5-10, the baseline classifier model, with a Kappa score of 70.23%, can be seen as performing considerably well on the NSL-KDD dataset compared to a model that simply guesses at random. The AUC score is regarded as an effective approach to summarise the model's overall accuracy and may be interpreted as the greater the AUC score, the better the model's ability to differentiate between two classes (Mandrekar, 2010). The baseline classifier model, with an AUC score of 96.46%, has an excellent level of separability and indicates that it has a probability of 96.46% of correctly distinguishing between normal and malicious network traffic.

The training and validation loss of the baseline autoencoder model, as depicted in Figure 5-6, outlines the learning performance (learning curve) of the model over time in terms of its experience. Learning curves are used to estimate the time required to accomplish an activity as

well as the reduction in effort required for completing subsequent tasks as learning activities occur (Anzanello & Fogliatto, 2011). Goodfellow *et al.* (2016) proposed that two factors can influence how well a learning algorithm will perform on a given problem:

1. Training loss: A higher training loss signifies that the model is underfitting on the training data; and

2. The gap between training and validation loss: When the gap between the training loss and validation loss is too big, overfitting occurs.

A good fit by the model is, therefore, defined by a training and validation loss that gradually declines, with a small gap between the two final loss values. Based on the learning curve of the baseline autoencoder, as shown in Figure 5-6, it can be seen that the training steadily declined, although the validation loss was far more erratic. However, there is a small gap between the final loss values.

The learning curves presented in Figure 5-7 show the training and validation loss of the baseline classifier model. Here, the training loss, represented by the blue line, is an indication of the model's error rate on the training dataset, while the validation loss, represented by the orange line, is the error rate after applying the trained model to the validation dataset after each epoch. The baseline classifier's training loss became less erratic after about 50 epochs, however, its validation loss failed to do the same and kept fluctuating.

The training and validation accuracy of the baseline classifier is provided in Figure 5-8, where the training accuracy, represented by the blue line, is an indication of how accurate the model is on the training dataset, while the validation accuracy, represented by the orange line, is the model's performance, based on the validation dataset. Here, it can be observed that the training accuracy and validation accuracy of the baseline model both saw steady improvements as the training progressed with the gap between these two values remaining relatively consistent throughout.

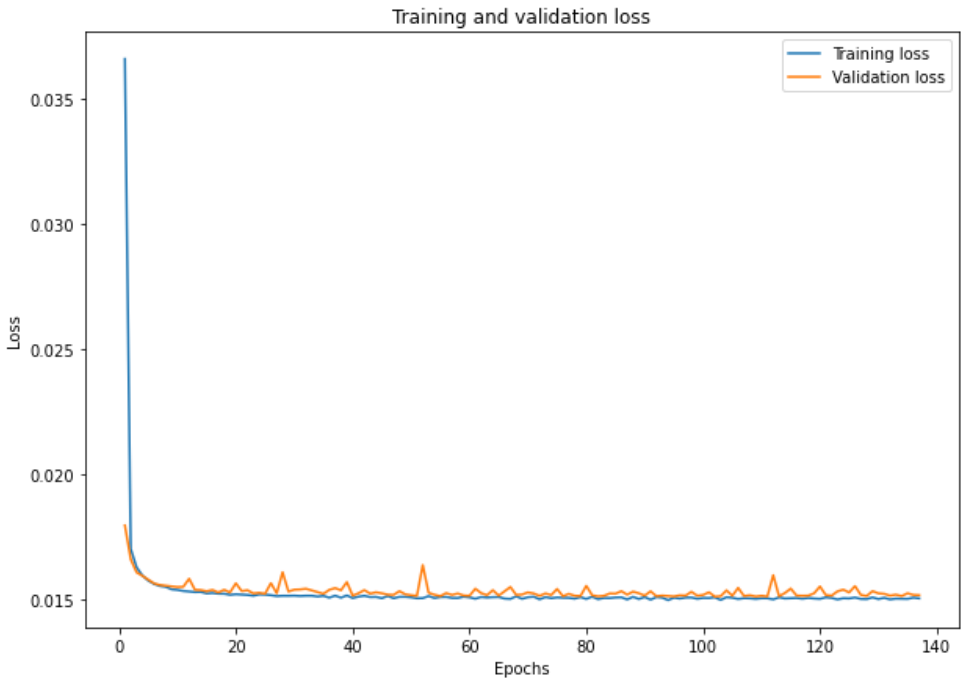


Figure 5-6: Baseline autoencoder training and validation loss

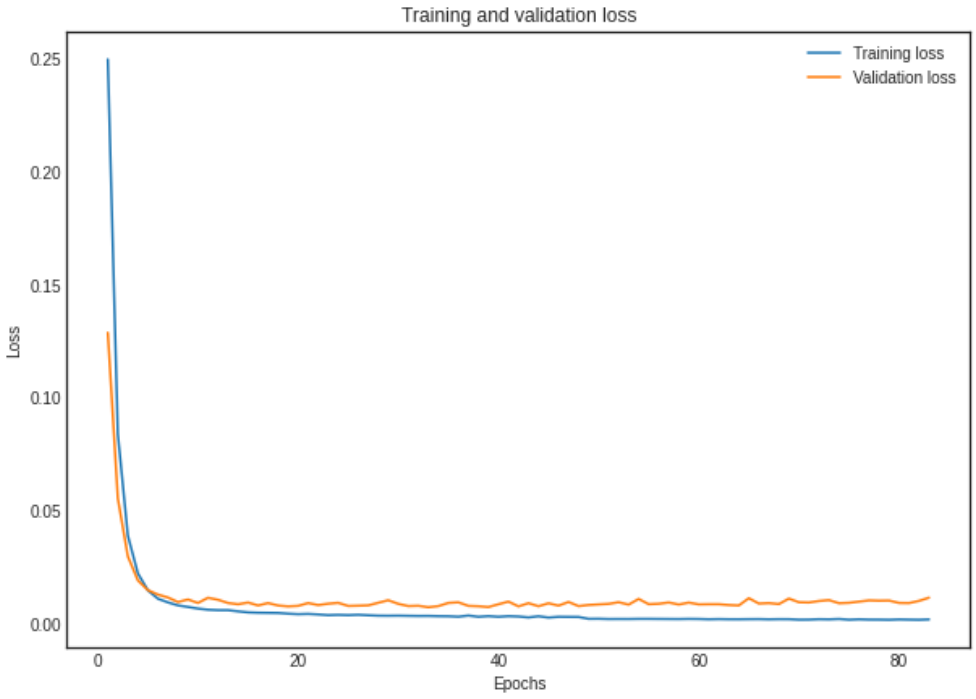


Figure 5-7: Baseline classifier training and validation loss

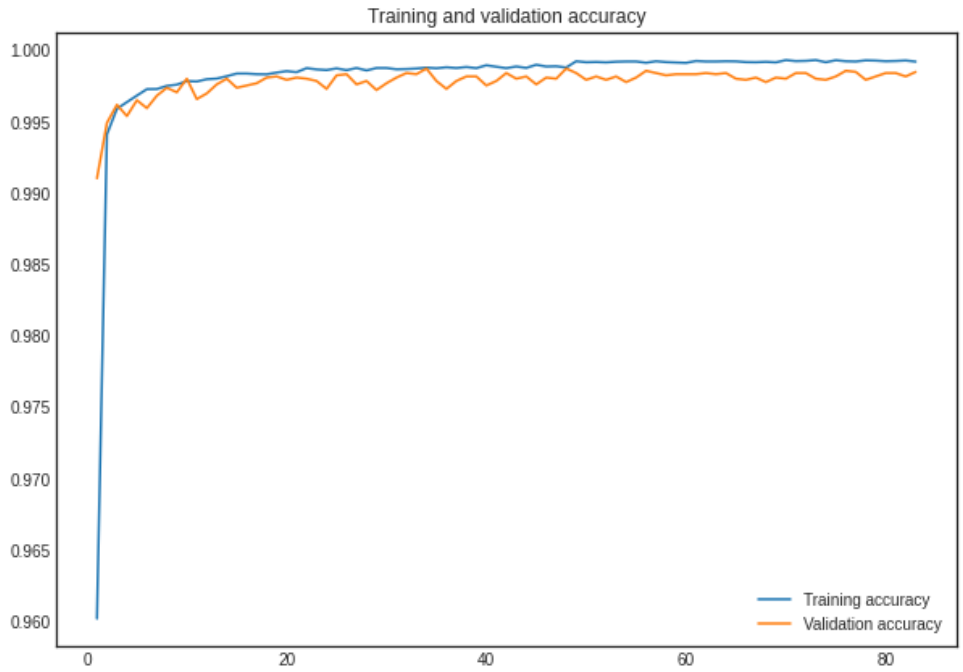


Figure 5-8: Baseline classifier training and validation accuracy

The ROC curve presented in Figure 5-9 expresses the baseline classifier model's performance, based on the trade-off between false positives and false negatives (Fawcett, 2006). Here, the blue line represents the separability threshold and the orange line represents the baseline classifier.

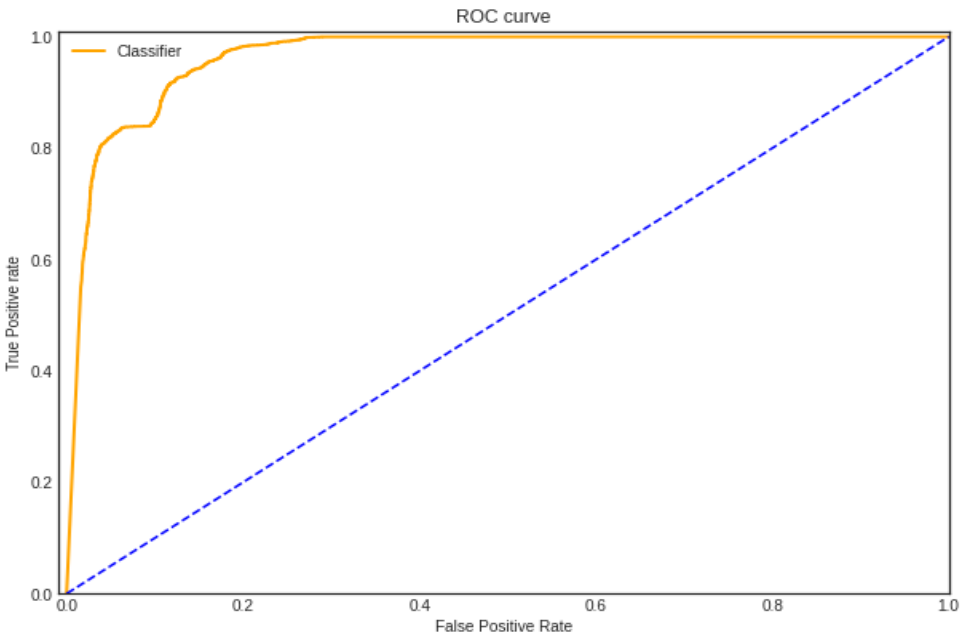


Figure 5-9: Baseline classifier ROC curve

The ROC curve for the baseline classifier model, as shown in Figure 5-9, was plotted using the model's false positive rate on the x-axis against its true positive rate on the y-axis. Based on this curve, the baseline classifier model performed very well on the NSL-KDD dataset.

5.4.2 Performance of best-first search optimisation model

The confusion matrix obtained by evaluating the best-first search optimisation classifier model on the test dataset is provided in Table 5-11. In addition, specific performance metrics of the model, as identified in Section 5.3.4, are shown in Table 5-12. The learning curve for the best-first search optimisation autoencoder model is presented in Figure 5-10, and the learning and performance curves for the best-first search optimisation classifier model are shown in Figure 5-11, Figure 5-12 and Figure 5-13, respectively.

Table 5-11: Best-first search optimisation classifier model confusion matrix

		Predicted traffic	
		Malicious	**Normal**
Actual traffic	**Malicious**	9 299	2 560
	Normal	412	10 273

Table 5-12: Best-first search optimisation classifier model metrics

Accuracy	86.82%
Precision	95.76%
Recall	78.41%
F1 score	86.22%
Kappa	73.82%
AUC	96.77%

According to the confusion matrix presented in Table 5-11, the classifier model, based on the symmetrical autoencoder constructed by the best-first search algorithm, performed better than the baseline, having classified 19 572 out of 22 544 cases of network traffic correctly. Furthermore, the performance metrics shown in Table 5-12 indicate that the former model achieved an overall accuracy of 86.82%, a 1.93% improvement over the baseline. However, the precision score of 95.76% was 1.19% lower than that of the baseline model, indicating that the model may result in slightly more false positives. The recall score has also increased, with the

best model achieving 78.41%, a 3.25% improvement over the baseline. An F1 score of 86.22%, a Kappa score of 73.82% and an AUC score of 96.77% have improvements of 1.54%, 3.59% and 0.31% over the baseline model, respectively.

Based on the learning curve of the best-first search optimisation autoencoder model, as shown in Figure 5-10, it was observed that the gap between the training loss and validation loss was minimal. This indicates that the best-first search optimisation autoencoder model had better convergence on the NSL-KDD dataset than the baseline autoencoder model.

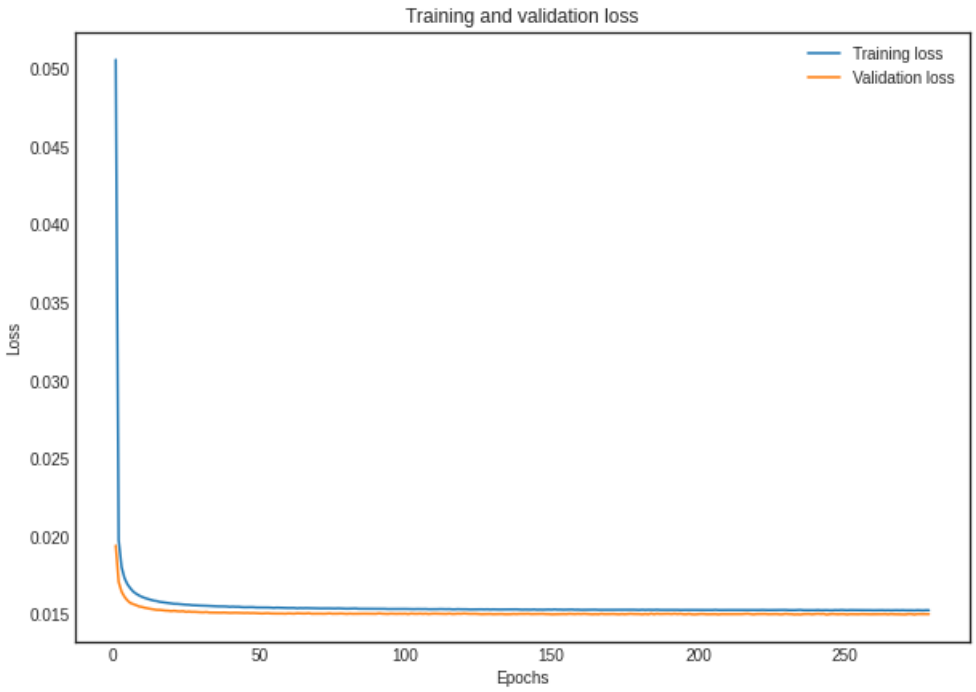


Figure 5-10: Best-first search optimisation autoencoder training and validation loss

The learning curves presented in Figure 5-11 show the training and validation loss of the best-first search optimisation classifier. Here, the training loss, represented by the blue line, is an indication of the model's error rate on the training dataset, while the validation loss, represented by the orange line, is the error rate after applying the trained model to the validation dataset after each epoch. Based on these learning curves, the best-first search optimisation classifier's training and validation loss converged.

The training and validation accuracy of the best-first search optimisation classifier is provided in Figure 5-12, where the training accuracy, represented by the blue line, is an indication of how accurately the model performed on the training dataset, while the validation accuracy, represented by the orange line, is the model's performance, based on the validation dataset.

While the model's validation accuracy was slightly more erratic than that of the baseline, the best-first search optimisation classifier still resulted in a good fit on both the training and validation datasets.

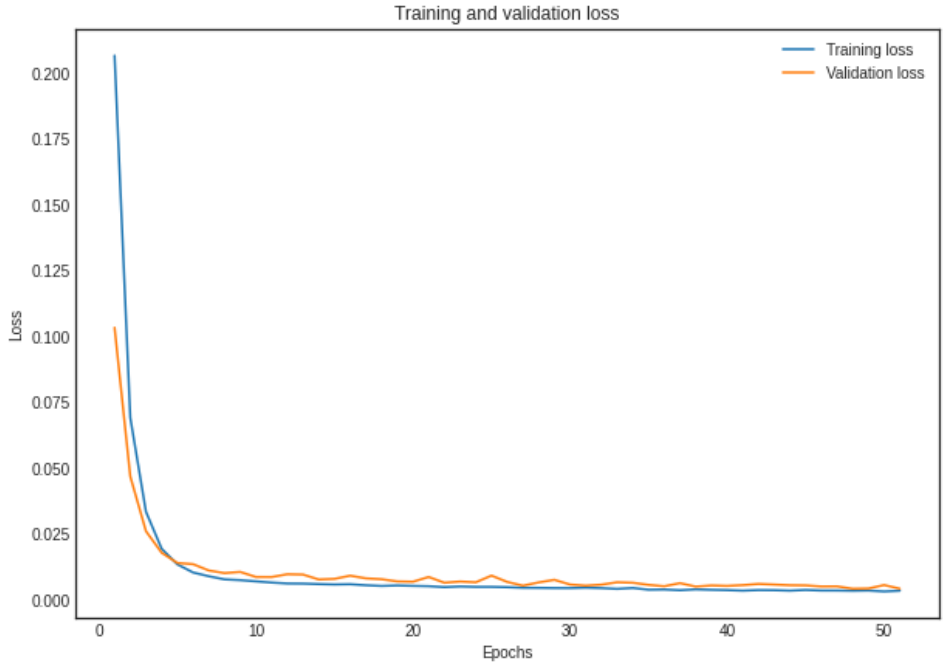


Figure 5-11: Best-first search optimisation classifier training and validation loss

Figure 5-12: Best-first search optimisation classifier training and validation accuracy

The ROC curve of the best-first search optimisation model's classifier, presented in Figure 5-13, details the model's ability to distinguish between normal and malicious network traffic, using the NSL-KDD dataset. Here, the blue line represents the separability threshold and the orange line represents the baseline classifier.

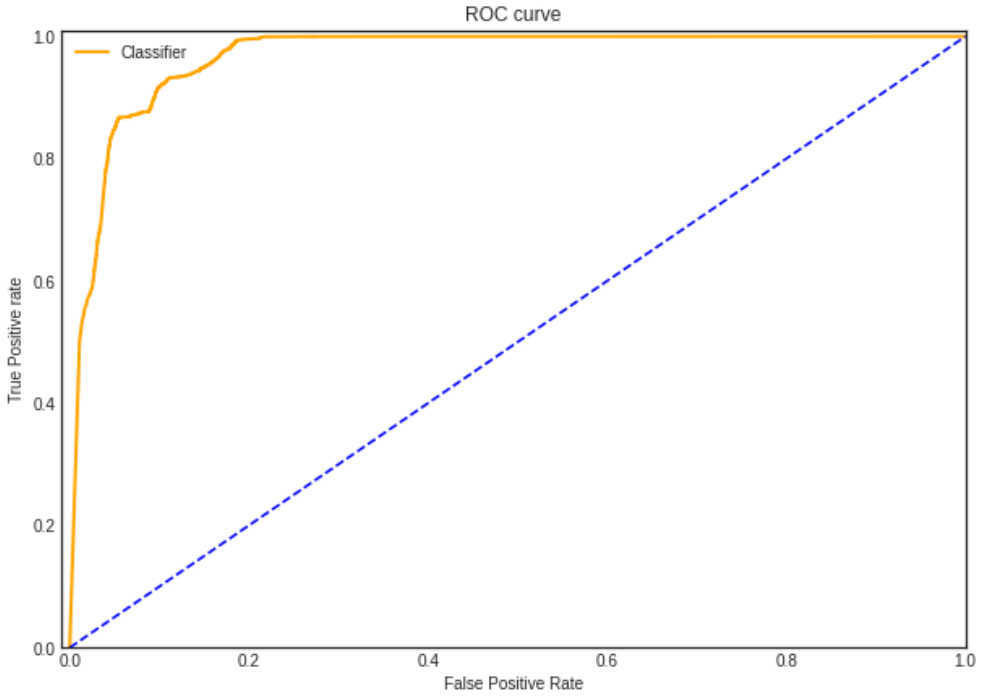


Figure 5-13: Best-first search optimisation classifier ROC curve

The ROC curve shown in Figure 5-13 was created by plotting the classifier model's false positive rate on the x-axis against its true positive rate on the y-axis. After interpreting these results, it was observed that the best-first search optimisation classifier model performed even better on the NSL-KDD dataset than the baseline classifier model, as evidenced by the higher AUC value of the model on the ROC curve.

Performance metrics for the baseline and best-first search optimisation models are summarised in the next section.

5.4.3 Summary of results

A summary of the results is presented in Table 5-13.

Table 5-13: Summary of classifier model accuracy results

Model	Accuracy
Baseline model	84.89%
Best-first search optimisation model	86.82%

The results presented in Table 5-13 conclude that the baseline model, with an accuracy of 84.89% outperformed two of the four autoencoder-based models considered in the literature (Table 5-7). Furthermore, the model constructed, using the best-first search architecture optimisation algorithm, surpassed the baseline model and three of the four autoencoder-based models considered in the literature with a final accuracy of 86.82%, indicating that the best-first search optimisation algorithm is a feasible and accurate solution for automating the construction of a symmetrical autoencoder to model intrusion detection on the NSL-KDD dataset. The model constructed, using the best-first search architecture optimisation algorithm, was outperformed only by the SAAE-DNN model proposed by Tang *et al.* (2020). A discussion of the research results following the experimentation is provided next.

5.5 Discussion of research results

The experimental results show that the proposed best-first search architecture optimisation algorithm developed in this study can be employed to construct an accurate symmetrical autoencoder. The corresponding autoencoder-based classifier is comparable to and even exceeds the performance of similar autoencoder-based models found in the literature, with minimal user intervention required while searching for the best model. It is apparent from the results presented in Table 5-7 and Table 5-13 that autoencoder-based neural networks are well suited for intrusion detection on the NSL-KDD dataset, with all models exceeding 81% classification accuracy. Moreover, the baseline model constructed in this study served as a suitable performance benchmark, since it outperformed two of the four models considered in the literature. The baseline model was ultimately improved by invoking the best-first search architecture optimisation algorithm discussed in Chapter 4. This algorithm demonstrated that the automated construction of an accurate symmetrical autoencoder in an autoencoder-based intrusion detection model can be achieved. Additionally, it was noted that less complex autoencoder architectures sometimes resulted in better convergence and increased performance of the model when using the NSL-KDD dataset. This was observed by evaluating and comparing the performance of the best-first search constructed model, having a relatively simplistic architecture, to that of the baseline model, which had a more complex architecture. Furthermore, the best-first search architecture optimisation algorithm ensured that an accurate symmetrical

autoencoder model was located more efficiently by applying the best-first search strategy during the model's mutation phase, which leverages heuristic information to reduce search times (Dechter & Pearl, 1985), and continually expanding the node with the most favourable evaluation metric during every iteration (Russell & Norvig, 2016). In doing so, the current best performing model's neural network architecture was randomly and iteratively mutated until an increase in the model's performance was noted. By automatically adjusting each new model's hyperparameters through a mutation process, keeping track of each subsequent model's evaluation metric to measure improvements in performance for early stopping, and by restricting the training process only to unique neural network architectures, employing the best-first architecture optimisation algorithm resulted in shorter training times, less user intervention required and utilisation of fewer system resources to achieve the same or better performance results on the NSL-KDD dataset.

5.6 Chapter summary

In this chapter, the intrusion detection dataset utilised during the experimentation was introduced, along with the pre-processing steps applied before it was used. Two experiments were then performed to establish whether the best-first search architecture optimisation algorithm could be employed to construct an accurate symmetrical autoencoder model. These experiments were conducted by manually constructing a symmetrical autoencoder-based neural network model, which served as a performance baseline. The best-first search architecture optimisation algorithm, presented in Chapter 4, was then employed to construct and train a better performing symmetrical autoencoder model, used as part of an intrusion detection system, to classify traffic as normal or malicious. Results from these experiments indicate that the best-first search architecture optimisation algorithm developed in this study is indeed a feasible approach for automating the construction of an accurate symmetrical autoencoder model, used as part of an intrusion detection system. An overview and conclusion of this study will be provided in the next, final chapter.

CHAPTER 6 CONCLUSION

In this study, the aim was to determine how a best-first search architecture optimisation algorithm could be designed and implemented to automate the construction of an accurate symmetrical autoencoder model, used as part of an intrusion detection system. Two autoencoder-based models were constructed and trained to achieve this, using a reputable intrusion detection dataset. The first model, which was manually constructed, served as a performance baseline. The second model, built by the best-first search architecture optimisation algorithm, was evaluated against the baseline and models from relevant literature. Results obtained from the experimentation suggest that the algorithm developed may be suitable for effectively automating the construction of an accurate symmetrical autoencoder that can be used as part of an intrusion detection model.

In this chapter, the aim is to evaluate the research objectives, to provide an overview of the contributions from the study, as well as to present future directions to consider. Through this, step six of the design science research methodology is addressed by communicating and documenting the overall process, including the importance of the proposed solution. Firstly, a description of how the study objectives outlined in Chapter 1 were met is provided in Section 6.1. The contributions of this research are then discussed in Section 6.2. In Section 6.3, there is a discussion of prospective future work that may be investigated. Finally, the chapter is concluded with a summary in Section 6.4.

6.1 Evaluation of research objectives

As stated in Chapter 1, Section 1.1, the research question guiding this study is: “How can a best-first search neural network architecture optimisation algorithm be designed to automatically build an accurate symmetrical autoencoder which forms part of an intrusion detection system?” The primary aim of this study was, therefore, to determine how a best-first search algorithm could be designed to automate the construction of an accurate autoencoder model, used as part of an intrusion detection system, to classify network traffic as either normal or malicious. Seven secondary objectives were established to accomplish the primary aim of the study. Below is a brief overview of how each secondary objective was achieved.

1. Perform a literature review on:

(a) Traditional computer and network threats and intrusion defence approaches;

A literature review was undertaken in Chapter 2 on the common cybersecurity threats, which stem from large networks of interconnected devices, and numerous approaches identified to prevent them. Firstly, a brief introduction to security and the growing concerns this poses to society was provided in Section 2.1 to Section 2.3. Thereafter, in Section 2.4, the concept and importance of cybersecurity were discussed. This was followed by a discussion on the types of computer security threats in Section 2.5 and approaches for securing information and data in Section 2.6.

(b) Intrusion detection systems, specifically in the context of information systems; and

Literature for intrusion detection systems was studied in Chapter 3. In Section 3.1, an introduction to information security and intrusion detection was provided. The various architectures and implementations of intrusion detection systems were then discussed in Section 3.2. This was followed by a summary of the three approaches to intrusion detection, namely misuse detection, anomaly detection and hybrid detection in Section 3.3.

(c) Deep learning neural network and autoencoder architectures and training techniques.

Literature on deep learning, autoencoders and training techniques were presented in Chapter 4. In Section 4.1, artificial intelligence, and the history thereof were briefly presented. Thereafter, the origin and structure of neural networks were discussed in Section 4.2. This was followed by Section 4.3, where the concept of deep learning was defined and its components were described. Learning algorithms were discussed in Section 4.3.2.3, and the backpropagation training technique was explained in Section 4.3.3. Finally, the architecture and applications for autoencoders were addressed in Section 4.3.4.

2. Design and implement a best-first search architecture optimisation algorithm to automate the construction of an accurate symmetrical autoencoder model, used as part of an intrusion detection system

A best-first search neural architecture optimisation algorithm, as discussed in Chapter 4, Section 4.4, was designed and implemented to automate the construction of an accurate symmetrical autoencoder used as part of the second intrusion detection model. This algorithm can be summarised in the following way:

(a) An initial symmetrical autoencoder model that can be constructed manually, or randomly, is trained and evaluated before being passed to the optimisation algorithm. This serves as a baseline for the best-first search algorithm;

(b) The algorithm is initialised by creating an empty list to store the population of evaluated models;

(c) Mutation procedures comprise modification of the drop-out rate, number of layers, number of nodes and the batch size of the model. This step is followed by checking whether the model's new architecture already exists in the population of models before being trained and evaluated. Only new model architectures are trained and evaluated;

(d) The score (based on its error rate) and architecture of the new model are then added to the list containing the population of models as a list of objects. If the score of the new model is better than the current best performing model in the population, then the new model is set as the best model;

(e) The best-first search architecture optimisation algorithm is executed iteratively for as long as the current best performing model's error rate is greater than S and the population of models is less than N. These two parameters serve as the stopping criteria for the search; and

(f) Once the stopping criteria have been satisfied, the best autoencoder model and its score are returned from the population of models.

3. Obtain a representative intrusion detection dataset

All experimentation performed in this study utilised the NSL-KDD dataset that other researchers have widely employed for intrusion detection, as discussed in Chapter 5, Section 5.1.

4. Pre-process the dataset and split it into training and testing data for the two symmetrical autoencoder-based neural network intrusion detection models developed in this study

Before a neural network model could be applied to the NSL-KDD dataset, it had to be prepared. These pre-processing steps were described in Chapter 5, Section 5.2. This process included converting labels and categorical fields to real numbers in Section 5.2.2 and the normalisation of the dataset to transform the values of numeric features in the dataset to a similar scale as described in Section 5.2.3. After preparing the data through various pre-processing techniques, the training and testing datasets consisted of 125 973 and 22 544 data points, respectively, with 122 input features. Additionally, validation data was formed by splitting the training dataset, as discussed in Section 5.2.6, using a 90:10 ratio of training data to validation data.

5. **Manually construct and train a suitable symmetrical autoencoder architecture combined with a multilayer perceptron classifier as a baseline intrusion detection model on the dataset prepared in Step 4 to classify network traffic as either normal or malicious**

The baseline symmetrical autoencoder model for this study was manually constructed upon performing a literature review of deep learning and autoencoders in Chapter 4. A multilayer perceptron classifier was then manually constructed and trained to classify the network traffic encoded by the baseline autoencoder model as normal or malicious. The resultant architectures for each experiment are detailed in Sections 5.3.1 and 5.3.3 of Chapter 5.

6. **By using the algorithm developed in Step 2, determine an accurate symmetrical autoencoder architecture and train the model combined with a multilayer perceptron classifier on the dataset prepared in Step 4 to classify network traffic as either normal or malicious**

The best-first search architecture optimisation algorithm outlined in Chapter 4, Section 4.4, was executed for 290 iterations, whereafter the autoencoder model with the lowest error rate was selected. A multilayer perceptron classifier was then manually constructed and trained to classify the network traffic encoded by the best autoencoder model as normal or malicious. The resultant architectures for each experiment are detailed in Sections 5.3.2 and 5.3.3 of Chapter 5.

7. **Evaluate the performance of the intrusion detection autoencoder-based neural network model constructed in Step 6 against the performance of the intrusion detection autoencoder-based neural network baseline model constructed in Step 5**

The performance of the autoencoder-based neural network intrusion detection model was measured and evaluated against a suitable baseline model, as well as the results from other models in the literature in Chapter 5, Section 5.4, using the following metrics discussed in Section 5.3.4:

- Accuracy: The classification model's ability to select or reject the correct class; measured as the ratio of successfully classified items compared to the total number of elements evaluated by the model;

- Precision: The classification model's degree of relevance when labelling a class; measured as the proportion of correctly classified true positives in relation to the total number of predicted positive instances;

- Recall: The proportion of cases where a true positive class was correctly classified by the model;

- F1 score: The weighted average of the true positive rate and precision of the model. An F1 score of 1 is considered perfect, whereas an F1 score of 0 is considered a failure;

- Kappa score: The Kappa statistic considers random chance to assess how much better one classifier performs than another classifier that simply guesses at random based on the frequency of each class. A Kappa score of 1 is considered perfect, whereas a Kappa score of 0 is considered a failure; and

- AUC score: An evaluation metric that represents the degree of separability, or how well a model can differentiate between classes. An AUC score of 1 represents a model with excellent separability while an AUC score of 0 indicates the inability to differentiate between two classes.

In summary, all the objectives outlined in Chapter 1 were achieved. The contributions of this study to the existing literature will be discussed in the next section.

6.2 Contributions of the study

In this study, a best-first search optimisation algorithm was successfully designed and implemented to automate constructing an accurate symmetrical autoencoder-based neural network intrusion detection model. Furthermore, this research contributed to the existing body of knowledge in the following ways:

- Cybersecurity and the architecture of intrusion detection systems were examined and discussed through a literature review;

- Different implementations and architectures for autoencoders were discussed;

- Steps to prepare the NSL-KDD dataset for use by a neural network were investigated and outlined; and

- The construction of an accurate deep symmetrical autoencoder neural network was automated by designing and implementing a best-first search architecture optimisation algorithm, which proved to be comparable in performance to other autoencoder-based models applied to the NSL-KDD dataset found in the literature.

6.3 Future work

The autoencoders considered during the experimentation in this study were limited to symmetrical architectures. Future research may consider expanding these models to include unsymmetrical autoencoder architectures to examine the possibility of improving the results.

Only a single intrusion detection dataset, NSL-KDD, was used in this study. Future research may consider utilising other or multiple datasets to evaluate the robustness of the model's performance.

The best-first search architecture optimisation algorithm designed and implemented in this study was limited to constructing and training autoencoders, which left the multilayer perceptron classification component to be manually designed and trained after the fact. Future research may consider expanding this algorithm to include constructing and evaluating a multilayer perceptron classifier, or other neural network architectures.

Finally, other neural network learning algorithms, such as reinforcement learning have been growing in popularity. Therefore, implementation thereof could be investigated to determine whether the model's performance can be improved.

6.4 Chapter summary

The initial research question and objectives for this study and how they were addressed are summarised in this chapter. Contributions made to the body of knowledge following this study were also provided, along with recommendations for future research. The results obtained during this study were comparable, and in some cases, better than those of other classification models applied to the NSL-KDD dataset found in the literature. This indicates that the best-first search architecture optimisation algorithm, designed and implemented during this study, may be used to automate the construction of an accurate symmetrical autoencoder-based intrusion detection system.

BIBLIOGRAPHY

Abomhara, M. & Køien, G.M. 2015. Cyber Security and the Internet of Things: Vulnerabilities, Threats, Intruders and Attacks. *Journal of Cyber Security Mobility*, 4(1):65-88.

Agrafiotis, I., Nurse, J.R.C., Goldsmith, M., Creese, S. & Upton, D. 2018. A Taxonomy of Cyber-harms: Defining the Impacts of Cyber-attacks and Understanding How They Propagate. *Journal of Cybersecurity*, 4(1):1-15.

Ahmed, M., Mahmood, A.N., Hu, J. & Applications, C. 2016. A survey of network anomaly detection techniques. *Journal of Network Computer Applications*, 60:19-31.

Akerkar, R. 2014. Introduction to Artificial Intelligence. Second Edition. Delhi, India: PHI Learning Pvt. Ltd.

Al-Qatf, M., Lasheng, Y., Al-Habib, M. & Al-Sabahi, K. 2018. Deep learning approach combining sparse autoencoder with SVM for network intrusion detection. *IEEE Access*, 6:52843-52856.

Al-Shaer, E.S. & Hamed, H.H. 2004. Modeling and management of firewall policies. *IEEE Transactions on network service management*, 1(1):2-10.

Alani, M.M. 2021. Big Data in cybersecurity: a survey of applications and future trends. *Journal of Reliable Intelligent Environments*, 122:1-30.

Ali, N., Neagu, D. & Trundle, P. 2019. Evaluation of k-nearest neighbour classifier performance for heterogeneous data sets. *SN Applied Sciences*, 1(12):1-15.

Ali, Ö.G. & Yaman, K. 2013. Selecting rows and columns for training support vector regression models with large retail datasets. *European Journal of Operational Research*, 226(3):471-480.

Aljawarneh, S., Aldwairi, M. & Yassein, M.B. 2018. Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model. *Journal of Computational Science*, 25:152-160.

Aloqaily, M., Otoum, S., Al Ridhawi, I. & Jararweh, Y. 2019. An intrusion detection system for connected vehicles in smart cities. *Ad Hoc Networks*, 90:101842.

Alpern, N.J. & Shimonski, R.J. 2010. Security Standards and Services. Eleventh Hour Network+. Boston: Syngress. p. 121-142.

Altekar, G., Bagrak, I., Burstein, P. & Schultz, A. 2005. OPUS: Online Patches and Updates for Security. (*In* Proceedings of the 14th USENIX Security Symposium Maryland, USA. p. 287-302).

Ambrosin, M., Anzanpour, A., Conti, M., Dargahi, T., Moosavi, S.R., Rahmani, A.M. & Liljeberg, P. 2016. On the Feasibility of Attribute-Based Encryption on Internet of Things Devices. *IEEE Micro*, 36(6):25-35.

Anderson, D. & McNeill, G. 1992. Artificial Neural Networks Technology. *Kaman Sciences Corporation*, 258(6):1-83.

Anderson, J.P. Division, A.F.E.S. 1972. Computer Security Technology Planning Study. Technical Report ESD-TR-73-51. Massachusetts, USA.

Anderson, J.P. 1980. Computer Security Threat Monitoring and Surveillance. Technical Report 98-17. Pennsylvania, USA.

Anwar, S., Mohamad Zain, J., Zolkipli, M.F., Inayat, Z., Khan, S., Anthony, B. & Chang, V. 2017. From intrusion detection to an intrusion response system: Fundamentals, requirements, and future directions. *Algorithms*, 10(2):1-24.

Anzanello, M.J. & Fogliatto, F.S. 2011. Learning curve models and applications: Literature review and research directions. *International Journal of Industrial Ergonomics*, 41(5):573-583.

Arora, A., Nandkumar, A. & Telang, R. 2006. Does information security attack frequency increase with vulnerability disclosure? An empirical analysis. *Information Systems Frontiers*, 8(5):350-362.

Atzori, L., Iera, A. & Morabito, G. 2010. The internet of things: A survey. *Computer networks*, 54(15):2787-2805.

Ayodele, T.O. 2010. Types of machine learning algorithms. *New advances in machine learning*, 3:19-48.

Bace, R. & Mell, P. 2001. NIST Special Publication on Intrusion Detection Systems. Technical Report 800-31. California, USA.

Baldi, P. 2012. Autoencoders, unsupervised learning, and deep architectures. (*In* Proceedings of ICML workshop on unsupervised and transfer learning Washington, USA. p. 37-49).

Balın, M.F., Abid, A. & Zou, J. 2019. Concrete autoencoders: Differentiable feature selection and reconstruction. (*In* Proceedings of the 36th International Conference on Machine Learning (ICML) California, USA, PMLR. p. 444-453).

Banerjee, C., Mukherjee, T. & Pasiliao Jr, E. 2019. An empirical study on generalizations of the ReLU activation function. (*In* Proceedings of the ACM Southeast Conference Georgia, USA. p. 164-167).

Barry, B.I. & Chan, H.A. 2010. Intrusion detection systems. Handbook of Information and Communication Security. Berlin, Germany: Springer. p. 193-205.

Basheer, I.A. & Hajmeer, M. 2000. Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods*, 43(1):3-31.

Beaugnon, A., Chifflier, P. & Bach, F. 2017. Ilab: An interactive labelling strategy for intrusion detection. (*In* International Symposium on Research in Attacks, Intrusions, and Defenses (RAID) Georgia, USA, Springer. p. 120-140).

Ben-David, A. 2008. Comparison of classification accuracy using Cohen's Weighted Kappa. *Expert Systems with Applications*, 34(2):825-832.

Bendovschi, A. 2015. Cyber-Attacks – Trends, Patterns and Security Countermeasures. *Procedia Economics and Finance*, 28:24-31.

Benítez, J.M., Castro, J.L. & Requena, I. 1997. Are artificial neural networks black boxes? *IEEE Transactions on neural networks*, 8(5):1156-1164.

Bergstra, J. & Bengio, Y. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(2):281-305.

Bernard, H.R. & Bernard, H.R. 2000. Social Research Methods: Qualitative and Quantitative Approaches. California: Sage Publications.

Berners-Lee, T., Cailliau, R., Pellow, N. & Secret, A. 1993. The world wide web initiative. (*In* Proceedings of the International Networking Conference (INET) California, USA).

Berners-Lee, T.J. 1989. Information management: A proposal. *Weaving the Web*, No. CERN-DD-89-001-OC.

Berners-Lee, T., Cailliau, R., Groff, J.F. & Pollermann, B. 1992. World-Wide Web: The Information Universe. *Electronic Networking: Research, Applications and Policy*, 2(1):52-58.

Bharadwaj, A.S. 1996. Integrating Positivist and Interpretive Approaches to Information Systems Research: A Lakatosian Model. (*In* Proceedings of the Americas Conference on Information Systems (AMCIS) Arizona, USA).

Bhati, B.S. & Rai, C. 2020. Analysis of support vector machine-based intrusion detection techniques. *Arabian Journal for Science Engineering*, 45(4):2371-2383.

Biermann, E., Cloete, E. & Venter, L.M. 2001. A comparison of intrusion detection systems. *Computers Security*, 20(8):676-683.

Bishop, C.M. 2006. Pattern recognition and machine learning. New York, USA: Springer.

Blum, A.L. & Langley, P. 1997. Selection of relevant features and examples in machine learning. *Artificial intelligence*, 97(1-2):245-271.

Boell, S. & Cecez-Kecmanovic, D. 2015. What is an Information System? (*In* Proceedings of the 48th Hawaii International Conference on System Sciences (HICSS) Hawaii, USA. p. 4959-4968).

Bonaccorso, G. 2017. Machine Learning Algorithms: A Reference Guide to Popular Algorithms for Data Science and Machine Learning. Birmingham, UK: Packt Publishing.

Borkar, A., Donode, A. & Kumari, A. 2017. A survey on Intrusion Detection System (IDS) and Internal Intrusion Detection and protection system (IIDPS). (*In* Proceedings of the International Conference on Inventive Computing and Informatics (ICICI) Coimbatore, India, IEEE. p. 949-953).

Bostani, H. & Sheikhan, M. 2017. Hybrid of anomaly-based and specification-based IDS for Internet of Things using unsupervised OPF based on MapReduce approach. *Computer Communications*, 98:52-71.

Bottou, L. 2014. From machine learning to machine reasoning. *Machine learning*, 94(2):133-149.

Bourgeois, D., Mortati, J., Wang, S. & Smith, J. 2014. Information Systems for Business and Beyond. Washington DC, USA: Saylor Academy.

Bradley, A.P. 1997. The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms. *Pattern Recognition*, 30(7):1145-1159.

Bridges, R.A., Glass-Vanderlan, T.R., Iannacone, M.D., Vincent, M.S. & Chen, Q. 2019. A survey of intrusion detection systems leveraging host data. *ACM Computing Surveys*, 52(6):1-35.

Buchanan, B.G. 2005. A (very) brief history of artificial intelligence. *AI Magazine*. 26:53-53.

Burrell, G. & Morgan, G. 1979. Sociological Paradigms and Organisational Analysis. London, UK: Heinemann Educational Books.

Butun, I., Österberg, P. & Song, H. 2019. Security of the Internet of Things: Vulnerabilities, attacks, and countermeasures. *IEEE Communications Surveys*, 22(1):616-644.

Campbell-Kelly, M. & Garcia-Swartz, D.D. 2013. The history of the internet: the missing narratives. 28(1):18-33.

Cannady, J. 1998. Artificial neural networks for misuse detection. (*In* Proceedings of the National Information Systems Security Conference (NISSC) Arlington, USA. p. 443-456).

Cannady, J. & Harrell, J. 1996. A comparative analysis of current intrusion detection technologies. (*In* Proceedings of the 4th Technology for Information Security Conference (TISC) Texas, USA. Vol. 96).

Cerchiello, P. & Giudici, P. 2016. Big Data analysis for financial risk management. *Journal of Big Data*, 3(1):1-12.

Chandrashekar, G. & Sahin, F. 2014. A survey on feature selection methods. *Computers Electrical Engineering*, 40(1):16-28.

Chauhan, N. & Singh, K. 2018. A Review on Conventional Machine Learning vs Deep Learning. (*In* Proceedings of the International Conference on Computing, Power and Communication Technologies (GUCON) Galgotias University, India. p. 347-352).

Chauvin, Y. 1990. Generalization performance of overtrained back-propagation networks. *Neural Networks*:45-55.

Chen, C., Seff, A., Kornhauser, A. & Xiao, J. 2015. Deepdriving: Learning affordance for direct perception in autonomous driving (*In* Proceedings of the IEEE International Conference on Computer Vision (ICCV) Santiago, Chile. p. 2722-2730).

Chen, J., Sathe, S., Aggarwal, C. & Turaga, D. 2017. Outlier detection with autoencoder ensembles. (*In* Proceedings of the 2017 SIAM International Conference on Data Mining Texas, USA. p. 90-98).

Chervenak, A., Vellanki, V. & Kurmas, Z. 1998, March. Protecting file systems: A survey of backup techniques. (*In* Proceedings of the Joint NASA and IEEE Mass Storage Conference Maryland, USA. Vol. 99).

Choudhary, S. & Kesswani, N. 2020. Analysis of KDD-Cup'99, NSL-KDD and UNSW-NB15 datasets using deep learning in IoT. *Procedia Computer Science*, 167:1561-1573.

Coburn, A., Leverett, E. & Woo, G. 2018. Solving cyber risk: protecting your company and society: John Wiley & Sons.

Cohen, J. 1960. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20:37-46.

Cole, E., Krutz, R. & Conley, J.W. 2011. Network Security Bible. 2nd. Vol. 768. Indiana, USA: Wiley Publishing.

Council, F.N. 1995. FNC Resolution: Definition of "Internet" Federal Networking Council. https://www.nitrd.gov/fnc/internet_res.pdf Date of access: 30 Sep 2020.

Croce, F., Andriushchenko, M. & Hein, M. 2019. Provable robustness of relu networks via maximization of linear regions. (*In* Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS) Okinawa, Japan. p. 2057-2066).

Crowdstrike. 2020. Threat Hunting Report: Insights from the Crowdstrike Overwatch Team. https://go.crowdstrike.com/rs/281-OBQ-266/images/Report2020OverWatchNowheretoHide.pdf Date of access: 29 Aug 2021.

Crystal, D. 2001. Language and the Internet. Language and the Internet. Cambridge, UK: Cambridge University Press. p. 1-25.

Cukier, K. & Mayer-Schoenberger, V. 2013. The rise of big data: How it's changing the way we think about the world. *Foreign Affairs*, 92(3):28-40.

Daswani, N. & Elbayadi, M. 2021. Big Breaches: Cybersecurity Lessons for Everyone. New York, USA: Apress.

Dayan, P., Sahani, M. & Deback, G. 1999. Unsupervised learning. *The MIT encyclopedia of the cognitive sciences*:857-859.

De Bruijn, H. & Janssen, M. 2017. Building Cybersecurity Awareness: The need for evidence-based framing strategies. *Government Information Quarterly*, 34(1):1-7.

De Mauro, A., Greco, M. & Grimaldi, M. 2016. A formal definition of Big Data based on its essential features. *Library Review*, 65(3):122-135.

Debar, H., Dacier, M. & Wespi, A. 1999. Towards a taxonomy of intrusion-detection systems. *Computer networks*, 31(8):805-822.

Dechter, R. & Pearl, J. 1985. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32(3):505-536.

Demuth, H.B., Beale, M.H., De Jess, O. & Hagan, M.T. 2014. Neural network design. Second Edition. Boston, USA: Martin Hagan.

Deng, L. & Yu, D. 2014. Deep learning: methods and applications. *Foundations trends in signal processing*, 7(3–4):197-387.

Denning, D. & Neumann, P.G. 1985. Requirements and model for IDES - a real-time intrusion-detection expert system. Vol. 8. California, USA: SRI International.

Depren, O., Topallar, M., Anarim, E. & Ciliz, M.K. 2005. An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks. *Expert Systems with Applications*, 29(4):713-722.

Dey, A. 2016. Machine learning algorithms: a review. *International Journal of Computer Science Information Technologies*, 7(3):1174-1179.

Dhanabal, L. & Shantharajah, S. 2015. A study on NSL-KDD dataset for intrusion detection system based on classification algorithms. *International Journal of Advanced Research in Computer Communication Engineering*, 4(6):446-452.

Dinev, T. & Hart, P. 2005. Internet Privacy Concerns and Social Awareness as Determinants of Intention to Transact. *International Journal of Electronic Commerce*, 10(2):7-29.

Ding, B., Qian, H. & Zhou, J. 2018. Activation functions and their characteristics in deep neural networks. (*In* Proceedings of the Chinese Control And Decision Conference (CCDC) Shenyang, China, IEEE. p. 1836-1841).

Djambazova, E., Dimitrov, K., Ioannidis, S., Kirda, E. & Kruegel, C. 2008. Anticipating Security Threats to a Future Internet. *EU FP7 Project FORWARD*.

Drewek-Ossowicka, A., Pietrołaj, M. & Rumiński, J. 2021. A survey of neural networks usage for intrusion detection systems. *Journal of Ambient Intelligence and Humanized Computing*, 12(1):497-514.

Du, S., Li, T., Yang, Y. & Horng, S.-J. 2020. Multivariate time series forecasting via attention-based encoder–decoder framework. *Neurocomputing*, 388:269-279.

Eğrioğlu, E., Aladağ, Ç.H. & Günay, S. 2008. A new model selection strategy in artificial neural networks. *Applied Mathematics Computation*, 195(2):591-597.

Elrawy, M.F., Awad, A.I. & Hamed, H.F. 2018. Intrusion detection systems for IoT-based smart environments: a survey. *Journal of Cloud Computing*, 7(1):1-20.

ENISA. 2016. Definition of Cybersecurity - Gaps and overlaps in standardisation. https://www.enisa.europa.eu/publications/definition-of-cybersecurity Date of access: 27 Jan 2021.

Fall, K.R. & Stevens, W.R. 2011. TCP/IP illustrated, volume 1: The protocols. Massachusetts, USA: Addison-Wesley.

Farahnakian, F. & Heikkonen, J. 2018. A deep auto-encoder based approach for intrusion detection system. (*In* Proceedings of the 20th International Conference on Advanced Communication Technology (ICACT) Chuncheon-si, South Korea, IEEE. p. 178-183).

Farzad, A., Mashayekhi, H. & Hassanpour, H. 2019. A comparative performance analysis of different activation functions in LSTM networks for classification. *Neural Computing Applications*, 31(7):2507-2521.

Fawcett, T. 2006. An introduction to ROC analysis. *Pattern recognition letters*, 27(8):861-874.

FDA. 2017. Firmware Update to Address Cybersecurity Vulnerabilities Identified in Abbott's (formerly St. Jude Medical's) Implantable Cardiac Pacemakers: FDA Safety Communication. https://www.fda.gov/medical-devices/safety-communications/firmware-update-address-cybersecurity-vulnerabilities-identified-abbotts-formerly-st-jude-medicals Date of access: 20 Jan 2021.

Fennelly, L.J. 2004. Effective physical security. Massachusetts, USA: Elsevier Butterworth-Heinemann.

Ferrag, M.A., Maglaras, L., Moschoyiannis, S. & Janicke, H. 2020. Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. *Journal of Information Security and Applications*, 50:102419.

Flach, P. 2012. Machine learning: the art and science of algorithms that make sense of data. Cambridge, UK: Cambridge University Press.

Forbes. 2012. How Target Figured Out A Teen Girl Was Pregnant Before Her Father Did. https://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/?sh=394637a06668 Date of access: 19 Jan 2021.

Glorot, X. & Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. (*In* Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS) Sardinia, Italy. p. 249-256).

Goel, S., Kanade, V., Klivans, A. & Thaler, J. 2017. Reliably learning the relu in polynomial time. (*In* Proceedings of the Conference on Learning Theory (COLT) Amsterdam, Netherlands. p. 1004-1042).

Goodfellow, I., Bengio, Y., Courville, A. & Bengio, Y. 2016. Deep learning. Vol. 1: MIT Press.

Gordon, K. 2013. What is Big Data? *ITNOW*, 55(3):12-13.

Govindan, R., Minei, I., Kallahalla, M., Koley, B. & Vahdat, A. 2016. Evolve or die: High-availability design principles drawn from Google's network infrastructure. (*In* Proceedings of the 2016 ACM SIGCOMM Conference Florianópolis, Brazil. p. 58-72).

Grace, A. 2020. What is encryption and how does it protect your data. https://us.norton.com/internetsecurity-privacy-what-is-encryption.html Date of access: 31 Jan 2021.

Gralla, P. 1998. How the Internet Works. Fourth Edition. Indiana, USA: Que Publishing.

Graves, A., Mohamed, A.-r. & Hinton, G. 2013. Speech recognition with deep recurrent neural networks. (*In* Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP) Vancouver, Canada, IEEE. p. 6645-6649).

Guardian, T. 2016. Tech company accused of collecting details of how customers use sex toys. https://www.theguardian.com/us-news/2016/sep/14/wevibe-sex-toy-data-collection-chicago-lawsuit Date of access: 19 Jan 2021.

Gudivada, V.N., Baeza-Yates, R. & Raghavan, V.V. 2015. Big data: Promises and problems. *IEEE Annals of the History of Computing*, 48(03):20-23.

Guyon, I. & Elisseeff, A. 2003. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157-1182.

Haenlein, M. & Kaplan, A. 2019. A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California management review*, 61(4):5-14.

Hagan, M.T., Demuth, H.B. & Jesús, O.D. 2002. An introduction to the use of neural networks in control systems. *International Journal of Robust Nonlinear Control*, 12(11):959-985.

Hamed, T., Ernst, J.B. & Kremer, S.C. 2018. A survey and taxonomy on data and pre-processing techniques of intrusion detection systems. *Computer and Network Security Essentials*:113-134.

Han, J., Kamber, M. & Pei, J. 2011. Data Mining: Concepts and Techniques. Third Edition. California, USA: Morgan Kaufmann Publishers.

Hanin, B. 2019. Universal function approximation by deep neural nets with bounded width and relu activations. *Mathematics*, 7(10):992.

Hassoun, M.H. 1995. Fundamentals of Artificial Neural Networks. Massachusetts, USA: MIT Press.

Hayes, B. 1994. The World Wide Web. *American Scientist*, 82(5):416-420.

Heady, R., Luger, G., Maccabe, A. & Servilla, M. 1990. The architecture of a network level intrusion detection system. University of New Mexico.

Healy, M. & Perry, C. 2000. Comprehensive criteria to judge validity and reliability of qualitative research within the realism paradigm. *Qualitative market research*, 3(3):118-126.

Heberlein, L.T., Dias, G.V., Levitt, K.N., Mukherjee, B., Wood, J. & Wolber, D. 1990. A network security monitor. (*In* Proceedings of the IEEE Symposium on Security and Privacy California, USA. p. 296-304).

Hein, M., Andriushchenko, M. & Bitterwolf, J. 2019. Why ReLU networks yield high-confidence predictions far away from the training data and how to mitigate the problem. (*In* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) California, USA. p. 41-50).

Herculano-Houzel, S. & Lent, R. 2005. Isotropic fractionator: a simple, rapid method for the quantification of total cell and neuron numbers in the brain. *Journal of Neuroscience*, 25(10):2518-2521.

Hevner, A. & Chatterjee, S. 2010. Design Research in Information Systems: Theory and Practice. Design Research in Information Systems: Theory and Practice. Springer. p. 9-22.

Hindy, H., Brosset, D., Bayne, E., Seeam, A.K., Tachtatzis, C., Atkinson, R. & Bellekens, X. 2020. A taxonomy of network threats and the effect of current datasets on intrusion detection systems. *IEEE Access*, 8:104650-104675.

Hinton, G.E. & Salakhutdinov, R.R. 2006. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504-507.

Hong, Y. & Furnell, S. 2021. Understanding cybersecurity behavioral habits: Insights from situational support. *Journal of Information Security and Applications*, 57:102710.

Hopfield, J.J. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554-2558.

Huang, S., Li, X., Cheng, Z.-Q., Zhang, Z. & Hauptmann, A. 2018. GNAS: A greedy neural architecture search method for multi-attribute learning. (*In* Proceedings of the 26th ACM International Conference on Multimedia Seoul, Korea. p. 2049-2057).

Hung, M. 2017. Gartner Insights on how to lead in a connected world. https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf Date of access: 17 Jan 2021.

Hunter, L. 1993. Artificial Intelligence and Molecular Biology. Vol. 445. California, USA: AAAI Press.

Ieracitano, C., Adeel, A., Gogate, M., Dashtipour, K., Morabito, F.C., Larijani, H., Raza, A. & Hussain, A. 2018. Statistical analysis driven optimized deep learning system for intrusion detection. (*In* Proceedings of the International Conference on Brain Inspired Cognitive Systems (BICS) Xi'an, China, Springer. p. 759-769).

Inayat, Z., Gani, A., Anuar, N.B., Khan, M.K. & Anwar, S. 2016. Intrusion response systems: Foundations, design, and challenges. *Journal of Network and Computer Applications*, 62:53-74.

Izeboudjen, N., Larbes, C. & Farah, A. 2014. A new classification approach for neural networks hardware: from standards chips to embedded systems on chip. *Artificial Intelligence Review*, 41(4):491-534.

Jabez, J. & Muthukumar, B. 2015. Intrusion detection system (IDS): anomaly detection using outlier detection approach. *Procedia Computer Science*, 48:338-346.

Jain, A.K., Mao, J. & Mohiuddin, K.M. 1996. Artificial neural networks: A tutorial. *Computer*, 29(3):31-44.

Jaiswal, A., Babu, A.R., Zadeh, M.Z., Banerjee, D. & Makedon, F. 2021. A survey on contrastive self-supervised learning. *Technologies*, 9(1):2.

Jarosz, Q. 2009. Neuron hand-tuned. https://commons.wikimedia.org/wiki/File:Neuron_Hand-tuned.svg Date of access: 15 Apr 2021.

Jarrett, K., Kavukcuoglu, K., Ranzato, M.A. & LeCun, Y. 2009. What is the best multi-stage architecture for object recognition? (*In* Proceedings of the 12th International Conference on Computer Vision (ICCV) Kyoto, Japan, IEEE. p. 2146-2153).

Javaid, A., Niyaz, Q., Sun, W. & Alam, M. 2016. 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) Brussels, Belgium. p. 21-26).

Jayalakshmi, T. & Santhakumaran, A. 2011. Statistical normalization and back propagation for classification. *International Journal of Computer Theory Engineering*, 3(1):1793-8201.

Ji, H., Wang, Y., Qin, H., Wang, Y. & Li, H. 2018. Comparative performance evaluation of intrusion detection methods for in-vehicle networks. *IEEE Access*, 6:37523-37532.

Jing, L. & Tian, Y. 2021. Self-Supervised Visual Feature Learning With Deep Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis Machine Intelligence*, 43(11):4037-4058.

Johnston, D., Magee, J.C., Colbert, C.M. & Christie, B.R. 1996. Active properties of neuronal dendrites. *Annual review of neuroscience*, 19(1):165-186.

Jones, A.K. & Sielken, R.S. Department of Computer Science. 2000. Computer system intrusion detection: A survey.

Jordan, M.I. 1995. Why the logistic function? A tutorial discussion on probabilities and neural networks. Computational Cognitive Science Technical Report 9503. Massachusetts, USA.

Jordan, M.I. & Mitchell, T.M. 2015. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255-260.

Jouini, M., Rabai, L.B.A. & Aissa, A.B. 2014. Classification of security threats in information systems. *Procedia Computer Science*, 32:489-496.

Kaboub, F. 2008. Positivist paradigm. (*In* Encyclopedia of counselling: changes and challenges for counselling in the 21st century, 2:343).

Kahn, R.E. & Cerf, V.G. 1999. What Is The Internet (And What Makes it Work). http://www.cnri.net/what_is_internet.html Date of access: 27 Jul 2020.

Kandasamy, K., Srinivas, S., Achuthan, K. & Rangan, V.P. 2020. IoT cyber risk: a holistic analysis of cyber risk assessment frameworks, risk vectors, and risk ranking process. *EURASIP Journal on Information Security*, 2020(1):1-18.

Karatas, G., Demir, Ö. & Sahingoz, O. 2018. Deep Learning in Intrusion Detection Systems. (*In* Proceedings of the International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT) Ankara, Turkey. p. 113-116).

Kaspersky. 2019. Spectrology: CPU hardware vulnerabilities in 2019. https://www.kaspersky.com.au/blog/35c3-spectre-meltdown-2019/21886/ Date of access: 20 Jan 2021.

Keegan, N., Ji, S.-Y., Chaudhary, A., Concolato, C., Yu, B., Jeong, D.H.J.H.-c.C. & Sciences, I. 2016. A survey of cloud-based network intrusion detection analysis. 6(1):1-16.

Keston, G. 2014. Security Implications of Big Data. http://www.dbta.com/Editorial/Trends-and-Applications/Security-Implications-of-Big-Data-97720.aspx Date of access: 27 Jul 2020.

Khraisat, A., Gondal, I., Vamplew, P. & Kamruzzaman, J. 2019. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(1):1-22.

Kim, G., Lee, S. & Kim, S. 2014. A novel hybrid intrusion detection method integrating anomaly detection with misuse detection. *Expert Systems with Applications*, 41(4):1690-1700.

Kira, K. & Rendell, L.A. 1992. A practical approach to feature selection. (*In* Proceedings of the International Conference on Machine Learning (ICML) California, USA. p. 249-256).

Kizza, J.M., Kizza & Wheeler. 2013. Guide to computer network security. Heidelberg, Germany: Springer.

Kothari, C.R. 2004. Research methodology: Methods and techniques. New Delhi, India: New Age International.

Kotsiantis, S.B., Zaharakis, I. & Pintelas, P. 2007. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160(1):3-24.

Kotu, V. & Deshpande, B. 2019. Data Science: Concepts and Practice. Second Edition. Massachusetts, USA: Morgan Kaufmann.

Krauss, S.E. 2005. Research paradigms and meaning making: A primer. *The Qualitative Report*, 10(4):758-770.

KS, D. & Ramakrishna, B. 2013. An Artificial Neural Network based Intrusion Detection System and Classification of Attacks. *International Journal of Engineering Research Applications*, 3(4):1959-1964.

Lazarevic, A., Kumar, V. & Srivastava, J. 2005. Intrusion detection: A survey. Managing Cyber Threats: Issues, Approaches, and Challenges. New York, USA: Springer. p. 19-78.

Le, L., Patterson, A. & White, M. 2018. Supervised autoencoders: Improving generalization performance with unsupervised regularizers. *Advances in neural information processing systems*, 31:107-117.

LeCun, Y., Bengio, Y. & Hinton, G. 2015. Deep learning. *Nature*, 521(7553):436-444.

LeCun, Y.A., Bottou, L., Orr, G.B. & Müller, K.-R. 2012. Efficient backprop. Neural networks: Tricks of the trade. Second Edition ed. Berlin, Germany: Springer. p. 9-48.

Leiner, B.M., Cerf, V.G., Clark, D.D., Kahn, R.E., Kleinrock, L., Lynch, D.C., Postel, J., Roberts, L.G. & Wolff, S. 2009. A brief history of the Internet. 39(5):22-31.

Lemley, J., Bazrafkan, S. & Corcoran, P. 2017. Deep Learning for Consumer Devices and Services: Pushing the limits for machine learning, artificial intelligence, and computer vision. *IEEE Consumer Electronics Magazine*. 6:48-56.

Li, B.-h., Hou, B.-c., Yu, W.-t., Lu, X.-b. & Yang, C.-w. 2017a. Applications of artificial intelligence in intelligent manufacturing: a review. *Frontiers of Information Technology Electronic Engineering*, 18(1):86-96.

Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R.P., Tang, J. & Liu, H. 2017b. Feature selection: A data perspective. *ACM Computing Surveys*, 50(6):1-45.

Li, J. & Yang, X. 2020. A Cyclical Learning Rate Method in Deep Learning Training. (*In* Proceedings of the International Conference on Computer, Information and Telecommunication Systems (CITS) Virtual Conference, IEEE. p. 1-5).

Liang, C., Shanmugam, B., Azam, S., Karim, A., Islam, A., Zamani, M., Kavianpour, S. & Idris, N.B. 2020. Intrusion detection system for the internet of things based on blockchain and multi-agent systems. *Electronics*, 9(7):1120.

Licklider, J.C.R. & Clark, W.E. 1962. On-line man-computer communication. (*In* Proceedings of the May 1-3, 1962, Spring Joint Computer Conference California, USA. p. 113-128).

Lirette, C. 2019. What is the Relationship Between IoT and Big Data. https://www.soracom.io/blog/what-is-the-relationship-between-iot-and-big-data/#:~:text=Big%20Data%20and%20IoT,the%20volume%20of%20data%20collected. Date of access: 31 Jan 2021.

Littlewood, B., Brocklehurst, S., Fenton, N., Mellor, P., Page, S., Wright, D., Dobson, J., McDermid, J. & Gollmann, D. 1993. Towards operational measures of computer security. *Journal of computer security*, 2(2-3):211-229.

Liu, J., Xiao, K., Luo, L., Li, Y. & Chen, L. 2020. An intrusion detection system integrating network-level intrusion detection and host-level intrusion detection. (*In* Proceedings of the IEEE 20th International Conference on Software Quality, Reliability and Security (QRS) Macau, China, IEEE. p. 122-129).

Lupo Pasini, M., Yin, J., Li, Y.W. & Eisenbach, M. 2021. A scalable algorithm for the optimization of neural network architectures. *Parallel Computing*, 104-105:102788.

Ma, L. & Khorasani, K. 2003. A new strategy for adaptively constructing multilayer feedforward neural networks. *Neurocomputing*, 51:361-385.

Mac, H., Truong, D., Nguyen, L., Nguyen, H., Tran, H.A. & Tran, D. 2018. Detecting attacks on web applications using autoencoder. (*In* Proceedings of the 9th International Symposium on Information and Communication Technology Da Nang, Vietnam. p. 416-421).

Mack, L. 2010. The philosophical underpinnings of educational research. *Polyglossia*, 19:5-11.

Madakam, S. & Lake, V. 2015. Internet of Things (IoT): A literature review. *Journal of Computer Communications*, 3(5):164.

Mandrekar, J.N. 2010. Receiver Operating Characteristic Curve in Diagnostic Test Assessment. *Journal of Thoracic Oncology*, 5(9):1315-1316.

Maree, K. 2007. First steps in research. Pretoria, SA: Van Schaik.

Martin, K.E. 2015. Ethical issues in the big data industry. *MIS Quarterly Executive*, 14:67-85.

McCulloch, W.S. & Pitts, W. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115-133.

McHugh, M.L. 2012. Interrater reliability: the kappa statistic. *Biochemia medica*, 22(3):276-282.

McIntosh, C. 2013. Cambridge Advanced Learner's Dictionary. https://dictionary.cambridge.org/dictionary/english/dataset Date of access: 10 Apr 2021.

Meltdownattack. 2018. Meltdown and Spectre. https://meltdownattack.com/ Date of access: 20 Jan 2021.

Mendoza, H., Klein, A., Feurer, M., Springenberg, J.T. & Hutter, F. 2016. Towards automatically-tuned neural networks. (*In* Proceedings of the Workshop on Automatic Machine Learning New York, USA. p. 58-65).

Mercioni, M.A. & Holban, S. 2020. The most used activation functions: classic versus current. (*In* Proceedings of the International Conference on Development and Application Systems (DAS) Suceava, Romania, IEEE. p. 141-145).

Microsoft. 2019. Normalize Data. https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/normalize-data Date of access: 02 Oct 2021.

Miller, G.F., Todd, P.M. & Hegde, S.U. 1989. Designing Neural Networks Using Genetic Algorithms. (*In* Proceedings of the 3rd International Conference on Genetic Algorithms (ICGA) Virginia, USA. p. 379-384).

Minsky, M. 2006. The emotion machine: Commonsense thinking, artificial intelligence, and the future of the human mind. New York, USA: Simon and Schuster.

Minsky, M. & Papert, S.A. 1969. Perceptrons: An Introduction to Computational Geometry. Massachusetts, USA: MIT Press.

Modi, C., Patel, D., Borisaniya, B., Patel, H., Patel, A. & Rajarajan, M. 2013. A survey of intrusion detection techniques in cloud. *Journal of network computer applications*, 36(1):42-57.

Mohri, M., Rostamizadeh, A. & Talwalkar, A. 2012. Foundations of machine learning. Massachusetts, USA: MIT Press.

Mouheb, D., Abbas, S. & Merabti, M. 2019. Cybersecurity curriculum design: A survey. Transactions on Edutainment XV. Berlin, Germany: Springer. p. 93-107.

Mu, Y., Hua, G., Fan, W. & Chang, S.-F. 2014. Hash-SVM: Scalable kernel machines for large-scale visual classification. (*In* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Ohio, USA. p. 979-986).

Mukherjee, S. & Sharma, N. 2012. Intrusion detection using naive Bayes classifier with feature reduction. *Procedia Technology*, 4:119-128.

Nair, V. & Hinton, G.E. 2010. Rectified linear units improve restricted boltzmann machines. (*In* Proceedings of the 27th International Conference on Machine Learning (ICML) Haifa, Israel. p. 807-814).

Naseer, S., Saleem, Y., Khalid, S., Bashir, M.K., Han, J., Iqbal, M.M. & Han, K. 2018. Enhanced network anomaly detection based on deep neural networks. *IEEE Access*, 6:48231-48246.

NG, B.A. & Selvakumar, S. 2020. Anomaly detection framework for Internet of things traffic using vector convolutional deep learning approach in fog environment. *Future Generation Computer Systems*, 113:255-265.

Nilsson, R., Pena, J.M., Björkegren, J. & Tegnér, J. 2007. Consistent feature selection for pattern recognition in polynomial time. *The Journal of Machine Learning Research*, 8:589-612.

NIST. 2018. Cybersecurity Framework. https://www.nist.gov/cyberframework/framework Date of access: 27 Jan 2021.

Oates, B.J. 2006. Researching information systems and computing. Researching information systems and computing. London, UK: Sage. p. 282-283.

Oracle. 2020. Emerging Technologies: Driving Financial and Operational Efficiency. https://www.oracle.com/a/ocom/docs/esg-research-oracle-emerging-tech-report.pdf Date of access: 17 Jan 2021.

Osisanwo, F., Akinsola, J., Awodele, O., Hinmikaiye, J., Olakanmi, O. & Akinjobi, J. 2017. Supervised machine learning algorithms: classification and comparison. *International Journal of Computer Trends Technology*, 48(3):128-138.

Othman, S.M., Ba-Alwi, F.M., Alsohybe, N.T. & Al-Hashida, A.Y. 2018. Intrusion detection model using machine learning algorithm on Big Data environment. *Journal of Big Data*, 5(1):1-12.

Özkan, C. & Erbek, F.S. 2003. The comparison of activation functions for multispectral Landsat TM image classification. *Photogrammetric Engineering Remote Sensing*, 69(11):1225-1234.

Packard, N. 2020. The ARPANET Into the Internet: A Tale of Two Networks. *Studies in Media and Communication*, 8(1):37-49.

Pamukchiev, A., Jouet, S. & Pezaros, D.P. 2017. Distributed network anomaly detection on an event processing framework. (*In* Proceedings of the 14th Annual IEEE Consumer Communications & Networking Conference (CCNC) Las Vegas, USA. p. 659-664).

Pathak, M.J., Patel, R.L. & Rami, S.P. 2018. Comparative Analysis of Search Algorithms. *International Journal of Computer Applications*, 179(50):40-43.

Patterson, J. & Gibson, A. 2017. Deep Learning: A Practitioner's Approach. California, USA: O'Reilly Media, Inc.

Peffers, K., Tuunanen, T., Gengler, C.E., Rossi, M., Hui, W., Virtanen, V. & Bragge, J. 2006. The Design Science Research Process: A Model for Producing and Presenting Information Systems Research. (*In* Proceedings of the 1st International Conference on Design Science Research in Information Systems and Technology (DESRIST) California, USA. p. 83-106).

Pfleeger, C.P., Pfleeger, S.L. & Margulies, J. 2015. Security in Computing. Fifth Edition. New Jersey, USA: Pearson Education.

Pillai, M.M. 2011. Applying genetic algorithm techniques in network Intrusion Detection Systems. North-West University.

Popescu, M.-C., Balas, V.E., Perescu-Popescu, L. & Mastorakis, N. 2009. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits Systems*, 8(7):579-588.

Porter, M.E. & Heppelmann, J.E. 2015. How smart, connected products are transforming companies. *Harvard business review*, 93(10):96-114.

Potdar, K., Pardawala, T.S. & Pai, C.D. 2017. A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers. *International Journal of Computer Applications*, 175(4):7-9.

Potluri, S. & Diedrich, C. 2016. Accelerated deep neural networks for enhanced intrusion detection system. (*In* Proceedings of the IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA) Berlin, Germany, IEEE. p. 1-8).

Prinetto, P. & Roascio, G. 2020. Hardware Security, Vulnerabilities, and Attacks: A Comprehensive Taxonomy. (*In* ITASEC. p. 177-189).

Qiu, Y. & Dai, Y. 2019. A Stacked Auto-Encoder Based Fault Diagnosis Model for Chemical Process. *(In* Kiss, A.A., Zondervan, E., Lakerveld, R. & Özkan, L., *eds*. Computer Aided Chemical Engineering. Elsevier. p. 1303-1308).

Radiuk, P.M. 2017. Impact of training set batch size on the performance of convolutional neural networks for diverse datasets. *Information Technology Management Science*, 20(1):20-24.

Raina, R., Battle, A., Lee, H., Packer, B. & Ng, A.Y. 2007. Self-taught learning: transfer learning from unlabeled data. (*In* Proceedings of the 24th International Conference on Machine learning (ICML) Oregon, USA. p. 759-766).

Ramchoun, H., Idrissi, M.A.J., Ghanou, Y. & Ettaouil, M. 2016. Multilayer Perceptron: Architecture Optimization and Training. *IJIMAI*, 4(1):26-30.

Reddy, G.T., Reddy, M.P.K., Lakshmanna, K., Kaluri, R., Rajput, D.S., Srivastava, G. & Baker, T. 2020. Analysis of dimensionality reduction techniques on big data. *IEEE Access*, 8:54776-54788.

Roman, R., Zhou, J. & Lopez, J.J. 2013. On the features and challenges of security and privacy in distributed Internet of Things. *Computer Networks*, 57(10):2266-2279.

Rosenblatt, F. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Russell, S.J. & Norvig, P. 2016. Artificial intelligence: a modern approach. Third Edition. London, UK: Pearson Education.

Saeys, Y., Inza, I. & Larranaga, P. 2007. A review of feature selection techniques in bioinformatics. *Bioinformatics*, 23(19):2507-2517.

Sahebjamnia, N., Torabi, S.A. & Mansouri, S.A. 2015. Integrated business continuity and disaster recovery planning: Towards organizational resilience. *European Journal of Operational Research*, 242(1):261-273.

Salkind, N.J. 2008. Exploring Research. Seventh Edition. New Jersey, USA: Prentice Hall.

Sarker, I.H., Kayes, A., Badsha, S., Alqahtani, H., Watters, P. & Ng, A. 2020. Cybersecurity data science: an overview from machine learning perspective. *Journal of Big Data*, 7(1):1-29.

Sathya, R. & Abraham, A. 2013. Comparison of supervised and unsupervised learning algorithms for pattern classification. *International Journal of Advanced Research in Artificial Intelligence*, 2(2):34-38.

Scarfone, K. & Mell, P. 2012. Guide to intrusion detection and prevention systems (IDPS). NIST Special Publication, 800-94.

Semberecki, P. & Maciejewski, H. 2017. Deep learning methods for subject text classification of articles. (*In* Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS) Prague, Czech Republic, IEEE. p. 357-360).

Sharma, S. & Athaiya, A. 2020. Activation Functions in Neural Networks. *International Journal of Engineering Applied Sciences and Technology*, 4(12):310-316.

Shin, H.-C., Roth, H.R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D. & Summers, R.M. 2016. Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, 35(5):1285-1298.

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.

Shone, N., Ngoc, T.N., Phai, V.D. & Shi, Q. 2018. A Deep Learning Approach to Network Intrusion Detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(1):41-50.

Sibi, P., Jones, S.A. & Siddarth, P. 2013. Analysis of different activation functions using back propagation neural networks. *Journal of theoretical applied information technology*, 47(3):1264-1268.

Simon, H.A. 1996. The Sciences of the Artificial. Third Edition. Massachusetts, USA: MIT Press.

Singer, P.W. & Friedman, A. 2014. Cybersecurity: What Everyone Needs to Know. New York, USA: Oxford University Press.

Singh, A.P. & Singh, M.D. 2014. Analysis of Host-Based and Network-Based Intrusion Detection System. *International Journal of Computer Network Information Security*, 6(8):41-47.

Singh, D. & Singh, B. 2020. Investigating the impact of data normalization on classification performance. *Applied Soft Computing*, 97:105524.

Song, Y., Hyun, S. & Cheong, Y.-G. 2021. Analysis of Autoencoders for Network Intrusion Detection. *Sensors*, 21(13):4294.

Sood, A.K. & Enbody, R. 2012. Targeted cyberattacks: A superset of advanced persistent threats. *IEEE Security & Privacy*, 11(1):54-61.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929-1958.

Stuckman, J. & Purtilo, J. 2011. A testbed for the evaluation of web intrusion prevention systems. (*In* 2011 Third International Workshop on Security Measurements and Metrics, IEEE. p. 66-75).

Subba, B., Biswas, S. & Karmakar, S. 2016. A Neural Network Based System for Intrusion Detection and Attack Classification. (*In* Proceedings of the 22nd IEEE National Conference on Communication (NCC) Guwahati, India. p. 1-6).

Sun, Y., Mao, H., Guo, Q. & Yi, Z. 2016. Learning a good representation with unsymmetrical auto-encoder. *Neural Computing Applications*, 27(5):1361-1367.

Szegedy, C., Toshev, A. & Erhan, D. 2013. Deep neural networks for object detection. (*In* Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS) Nevada, USA. p. 2553-2561).

Takano, Y. & Kajikawa, Y. 2019. Extracting commercialization opportunities of the Internet of Things: Measuring text similarity between papers and patents. *Technological Forecasting and Social Change*, 138:45-68.

Tallon, P.P. 2013. Corporate governance of big data: Perspectives on value, risk, and cost. *Computer*, 46(6):32-38.

Tang, C., Luktarhan, N. & Zhao, Y. 2020. SAAE-DNN: Deep Learning Method on Intrusion Detection. *Symmetry*, 12(10):1695.

Tankard, C. 2012. Big data security. *Network Security*, 2012(7):5-8.

Tavallaee, M., Bagheri, E., Lu, W. & Ghorbani, A.A. 2009. A detailed analysis of the KDD CUP 99 data set. (*In* 2009 IEEE symposium on computational intelligence for security and defense applications Ottawa, Canada, IEEE. p. 1-6).

Ten, C.-W., Liu, C.-C. & Manimaran, G. 2008. Vulnerability assessment of cybersecurity for SCADA systems. *IEEE Transactions on Power Systems*, 23(4):1836-1846.

Torres, A.M. 2016. Electronic Menu and Ordering Application System: A Strategic Tool for Customer Satisfaction and Profit Enhancement. *International Journal of Service, Science Technology*, 9(4):401-410.

Tripathy, B. & Anuradha, J. 2017. Internet of things (IoT): Technologies, Applications, Challenges and Solutions. Florida, USA: CRC Press.

Tu, J.V. 1996. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of clinical epidemiology*, 49(11):1225-1231.

Turing, A.M. 1950. Computing machinery and intelligence. *Mind*, LIX(236):433-460.

UNB. 2009. NSL-KDD dataset. http://www.unb.ca/research/iscx/dataset/iscx-NSL-KDD-dataset.html Date of access: 03 Mar 2020.

Van Kranenburg, R. 2008. The Internet of Things: A critique of ambient technology and the all-seeing network of RFID. Vol. 02. Amsterdam, Netherlands: Institute of Network Cultures.

Van Rijn, J.N. & Hutter, F. 2018. Hyperparameter importance across datasets. (*In* Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD) London, UK. p. 2367-2376).

Ventures, C. 2019. Cybersecurity market report. https://cybersecurityventures.com/cybersecurity-market-report/ Date of access: 26 Jan 2021.

Vokorokos, L. & Baláž, A. 2010. Host-based intrusion detection system. (*In* Proceedings of the 14th IEEE International Conference on Intelligent Engineering Systems (INES) Las Palmas, Spain. p. 43-47).

W3C. 1997. About The World Wide Web. http://www.w3.org/www/ Date of access: 15 Jan 2021.

Wang, J., Wen, Y., Gou, Y., Ye, Z. & Chen, H. 2017. Fractional-order gradient descent learning of BP neural networks with Caputo derivative. *Neural Networks*, 89:19-30.

Wang, Y., Li, Y., Song, Y. & Rong, X. 2020a. The influence of the activation function in a convolution neural network model of facial expression recognition. *Applied Sciences*, 10(5):1897.

Wang, Y., Yang, H., Yuan, X., Shardt, Y.A.W., Yang, C. & Gui, W. 2020b. Deep learning for fault-relevant feature extraction and fault classification with stacked supervised auto-encoder. *Journal of Process Control*, 92:79-89.

Wang, Z. & Bovik, A.C. 2009. Mean squared error: Love it or leave it? A new look at signal fidelity measures. *IEEE signal processing magazine*, 26(1):98-117.

Weber, R.H. 2015. Internet of things: Privacy issues revisited. *Computer Law Security Review*, 31(5):618-627.

Weis, A.H. 2010. Commercialization of the Internet. *Internet Research*, 20(4):420-435.

Werbos, P. 1974. Beyond regression: new tools for prediction and analysis in the behavioral sciences. Harvard University.

Weston, J., Ratle, F., Mobahi, H. & Collobert, R. 2012. Deep learning via semi-supervised embedding. Neural networks: Tricks of the trade. Springer. p. 639-655.

Whitman, M.E. & Mattord, H.J. 2017. Principles of information security. Sixth Edition. Boston, USA: Cengage Learning.

Witten, I.H., Frank, E. & Hall, M.A. 2016. Data Mining Practical Machine Learning Tools and Techniques. Fourth Edition. Massachusetts, USA: Morgan Kaufmann.

Wortmann, F. & Flüchter, K. 2015. Internet of things. *Business Information Systems Engineering*, 57(3):221-224.

Wu, S.X. & Banzhaf, W. 2010. The use of computational intelligence in intrusion detection systems: A review. *Applied soft computing*, 10(1):1-35.

Wu, Y., Lee, W.W., Gong, X. & Wang, H. 2020. A Hybrid Intrusion Detection Model Combining SAE with Kernel Approximation in Internet of Things. *Sensors*, 20(19):5710.

Yegnanarayana, B. 1999. Artificial neural networks. New Delhi, India: Prentice-Hall of India.

Yousaf, A. & Yousaf, O. 2017. Intruders and intrusion detection systems—An overview. (*In* Proceedings of the 14th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON) Phuket, Thailand, IEEE. p. 131-134).

Yousefi-Azar, M., Varadharajan, V., Hamey, L. & Tupakula, U. 2017. Autoencoder-based feature learning for cyber security applications. (*In* Proceedings of the International Joint Conference on Neural Networks (IJCNN) Alaska, USA, IEEE. p. 3854-3861).

Zabalza, J., Ren, J., Zheng, J., Zhao, H., Qing, C., Yang, Z., Du, P. & Marshall, S. 2016. Novel segmented stacked autoencoder for effective dimensionality reduction and feature extraction in hyperspectral imaging. *Neurocomputing*, 185:1-10.

Zhang, Q., Yang, L.T., Chen, Z. & Li, P. 2018. A survey on deep learning for big data. *Information Fusion*, 42:146-157.

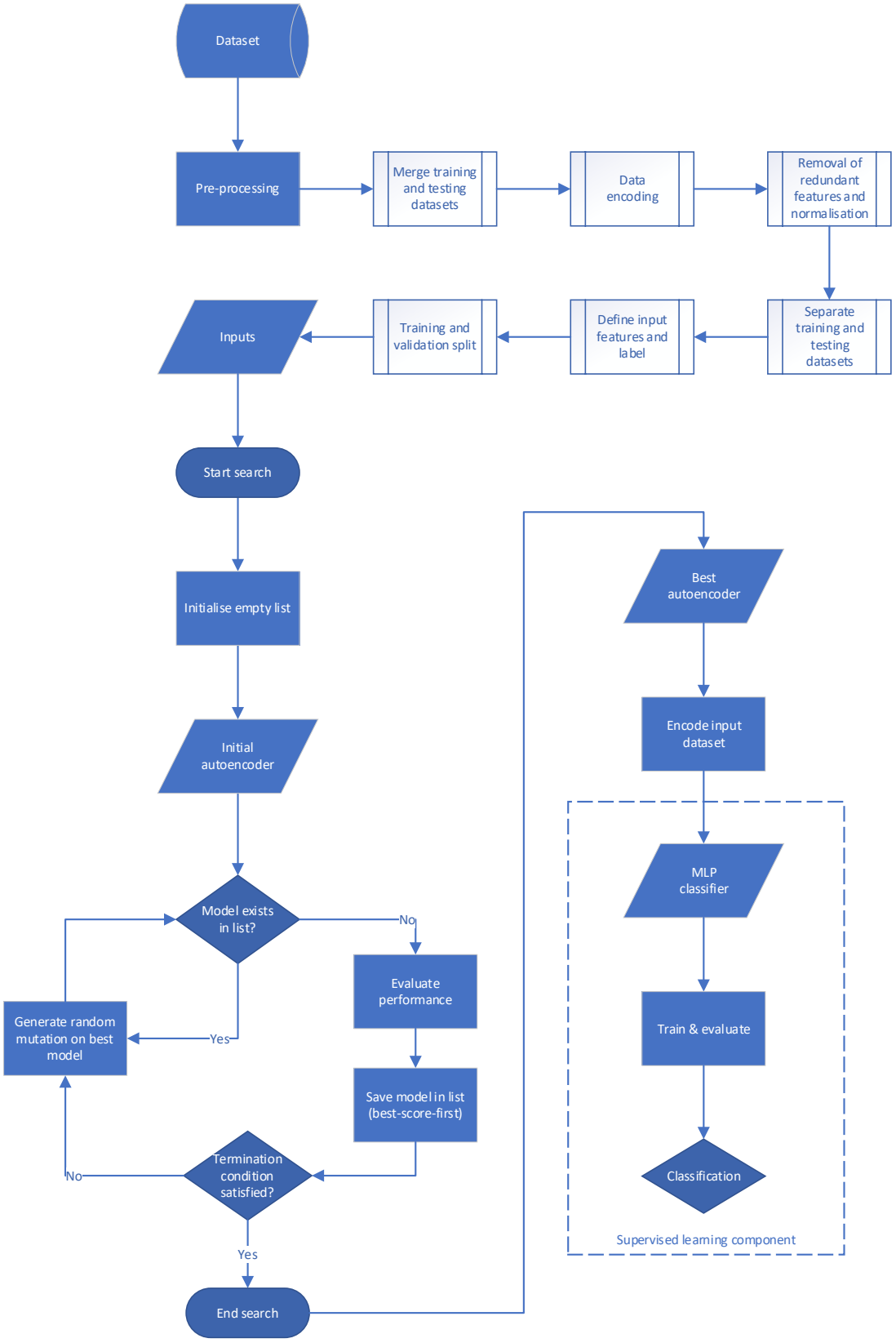
Zhang, W., Sauppe, J.J. & Jacobson, S.H. 2021. Comparison of the number of nodes explored by cyclic best first search with depth contour and best first search. *Computers & Operations Research*, 126:105129.

Zhou, C. & Paffenroth, R.C. 2017. Anomaly detection with robust deep autoencoders. (*In* Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining Halifax, Canada. p. 665-674).

Zhu, W. 2020. Computer Network Security Based on Prevention and Control of Network Virus. (*In* Proceedings of the IEEE International Conference on Civil Aviation Safety and Information Technology (ICCASIT) Weihai, China).

Zwitter, A. 2014. Big Data ethics. *Big Data & Society*, 1(2):6.

ANNEXURE A: FLOW DIAGRAM FOR BEST-FIRST SEARCH ARCHITECTURE OPTIMISATION ALGORITHM



ANNEXURE B: PYTHON SOURCE CODE FOR BEST-FIRST SEARCH ARCHITECTURE OPTIMISATION ALGORITHM

```
## Imports ##

# import packages
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import random
from tensorflow.keras import layers, models, callbacks
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.layers import Input, Dense, Dropout
from tensorflow.keras.models import Sequential, Model
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score,\
f1_score, cohen_kappa_score, roc_auc_score, confusion_matrix


## Function definitions ##

## load_data(): Load training and testing data ##

def load_data():

    # Set dataset path
    print('Setting dataset path...')
    FIELD_NAMES_PATH = #'//PATH TO FIELD NAMES'
    TRAIN_DATA_PATH = #'//PATH TO TRAINING DATASET'
    TEST_DATA_PATH = #'//PATH TO TESTING DATASET'

    # Read dataset files
    print('Reading dataset files...')
    df_train = pd.read_csv(TRAIN_DATA_PATH, header = None)
    df_test = pd.read_csv(TEST_DATA_PATH, header = None)

    # Read field names file
    print('Preparing for field name assignment...')
    with open(FIELD_NAMES_PATH, 'r') as txt_file:
        field_names = txt_file.readlines()

    # Clean field names input file
    field_names_clean = [i.split(':')[0] for i in field_names]

    # Add field for classification of the traffic input
    field_names_clean.extend(['label'])

    # Add field for difficulty level
    field_names_clean.extend(['score'])

    # Assign field names to dataset
    print('Assigning field names to dataset...')
    df_train.columns = field_names_clean
    df_test.columns = field_names_clean
```

```
    # Return train & test datasets
    training_data = df_train
    testing_data = df_test
    print('Dataset successfully loaded \n \n')

    return training_data, testing_data


## preprocess_data(): Clean and pre-process data ##

def preprocess_data(training_data, testing_data, validation_split):

    print('Start dataset pre-processing...')
    # Merge the datasets temporarily to do the encoding and normalisation
    print('Merge training and testing datasets...')
    test_samples_length = len(testing_data)
    df_combined = pd.concat([training_data, testing_data])

    # Data encoding
    print('Data encoding...')

    # Label encoding
    # Create new label field "enc_label" for binary classification
    # [0 = normal, 1 = malicious]
    df_combined['enc_label'] = \
    np.where(df_combined['label'].str.contains('normal'), 0, 1)

    # Drop original "label" field
    df_combined.drop(['label'], axis=1, inplace = True)

    # Rename the "enc_label" field to "label"
    df_combined.rename(columns={'enc_label': 'label'}, inplace = True)

    # Categorical encoding
    # Encode categorical values into binary vector arrays
    df_combined = pd.get_dummies(df_combined, columns = \
                                 ['protocol_type', 'service', 'flag'])

    # Normalisation
    print('Normalisation...')

    # Drop fields with std deviation of 0 to remove uninformative data
    df_combined.drop(['num_outbound_cmds'], axis=1, inplace = True)

    # Normalise the data using MinMaxScaler
    scaler = MinMaxScaler()
    df_combined = \
    pd.DataFrame(scaler.fit_transform(df_combined), \
                 columns=df_combined.columns)

    # Separate train and test datasets
    # Test and train datasets are separated into two independent sets
    print('Separate training and testing datasets...')
    training_data = df_combined.iloc[:-test_samples_length, :]
    testing_data = df_combined.iloc[-test_samples_length:,:]

    # Define features and label data from dataset
    print('Define features and labels...')
```

```
    # return everything except "label" field
    x_train = training_data.drop(columns = 'label').copy()
    # return the label field
    y_train = training_data['label']

    # return everything except "label" field
    x_test = testing_data.drop(columns = 'label').copy()
    # return the label field
    y_test = testing_data['label']

    #Training and validation split
    print('Training and validation split...')
    print('Training:', 1-validation_split, 'Validation:', validation_split)

    # Split training data into training and validation datasets
    x_train, x_valid, y_train, y_valid = \
    train_test_split(x_train, y_train, test_size = \
                     validation_split, shuffle = True, random_state = 42)
    print('End of dataset pre-processing \n \n')

    # Unpack dataset and print detail
    n_train, num_features = x_train.shape
    n_valid, _ = x_valid.shape
    n_test, _ = x_test.shape

    print('Dataset information:')
    print('Number of features:\t\t', num_features)
    print('Number of training samples:\t', n_train)
    print('Number of validation samples:\t', n_valid)
    print('Number of testing samples:\t', n_test, '\n \n')

    return x_train, y_train, x_valid, y_valid, x_test, y_test, num_features


## create_models_list(): Create models list for best-first search algorithm

def create_models_list():

    list_of_models = []
    print('Created empty list:', list_of_models, '\n')
    return list_of_models


## create_initial_autoencoder():
## Create a pre-defined or random initial autoencoder model ##

def create_initial_autoencoder(bfs_search_params):

    # Copy parameters for inital model
    initial_autoencoder_params = bfs_search_params.copy()

    # Pre-defined initialisation
    if bfs_search_params['initial_model'] == 'predefined':
        print('Creating a pre-defined initial model...', '\n')
        # Input layer
        # Set input layer = size of first item in the dictionary
        input_layer = Input(shape = \
                            (bfs_search_params['num_nodes_encoder'][0],))
        encoded = input_layer
```

```
    # Encoder hidden layers
    # Add hidden layers after the input layer of the encoder
    # (idx 1 to n)
    for i in range(1,bfs_search_params['num_layers_encoder']):
        encoded = Dense(bfs_search_params['num_nodes_encoder'][i], \
                        activation = \
                        bfs_search_params['activation_encoder'])\
                        (encoded)
        # Check if Dropout is enabled
        if bfs_search_params['dropout_rate'] != 0.0:
            encoded = Dropout(bfs_search_params['dropout_rate'])\
            (encoded)

    decoded = encoded

    # Decoder hidden layers
    # Add hidden layers before output layer of decoder (idx 0 to n-1)
    for i in range(bfs_search_params['num_layers_decoder'] - 1):
        decoded = Dense(bfs_search_params['num_nodes_decoder'][i], \
                        activation = \
                        bfs_search_params['activation_decoder'])\
                        (decoded)
        # Check if Dropout is enabled
        if bfs_search_params['dropout_rate'] != 0.0:
            decoded = \
            Dropout(bfs_search_params['dropout_rate'])\
            (decoded)

    # Output layer
    # Set output layer = size of last item in the dictionary
    decoded = Dense(bfs_search_params['num_nodes_decoder'][-1], \
                    activation = \
                    bfs_search_params['activation_output'])\
                    (decoded)

# Random initialisation
elif bfs_search_params['initial_model'] == 'random':
    print('Creating a random initial model...', '\n')
    # Start generating new random parameters and update dictionary
    # Number of layers parameters
    initial_autoencoder_params['num_layers_encoder'] = \
    random.randint(bfs_search_params['min_num_layers_encoder'], \
                   bfs_search_params['max_num_layers_encoder'])
    initial_autoencoder_params['num_layers_decoder'] = \
    initial_autoencoder_params['num_layers_encoder'] - 1

    # Number of nodes parameters
    initial_autoencoder_params['num_nodes_encoder'] = \
    np.zeros(initial_autoencoder_params['num_layers_encoder'],\
             dtype = int)
    initial_autoencoder_params['num_nodes_decoder'] = \
    np.zeros(initial_autoencoder_params['num_layers_decoder'],\
             dtype = int)

    # Input and output layer nodes parameters = num_features
    initial_autoencoder_params['num_nodes_encoder'][0] = \
    bfs_search_params['num_features']
    initial_autoencoder_params['num_nodes_decoder'][-1] = \
    bfs_search_params['num_features']
```

```
# Encoder hidden layers parameters
# Update hidden layer nodes after input layer of encoder
# (idx 1 to n)
for i in range(1,initial_autoencoder_params['num_layers_encoder']):
    initial_autoencoder_params['num_nodes_encoder'][i] = \
    random.randint(bfs_search_params['min_num_nodes_encoder'], \
                   bfs_search_params['max_num_nodes_encoder'])

# Force symmetrical structure for encoder sorting nodes high to low
initial_autoencoder_params['num_nodes_encoder'] = \
sorted(initial_autoencoder_params['num_nodes_encoder'], \
       reverse = True)

# Decoder layers parameters
# Mirror intermediate layers parameters
# Update hidden nodes before output layer of decoder (idx 0 to n-1)
for i in range(initial_autoencoder_params['num_layers_decoder']):
    initial_autoencoder_params['num_nodes_decoder'][-i] = \
    initial_autoencoder_params['num_nodes_encoder'][i]

# Force symmetrical structure for decoder sorting nodes low to high
initial_autoencoder_params['num_nodes_decoder'] = \
sorted(initial_autoencoder_params['num_nodes_decoder'])

# Update batch size and dropout rate parameters
initial_autoencoder_params['batch_size'] = \
random.randint(bfs_search_params['min_batch_size'], \
               bfs_search_params['max_batch_size'])
# Check if Dropout is enabled
if bfs_search_params['dropout_rate'] != 0.0:
    initial_autoencoder_params['dropout_rate'] = \
    random.uniform(bfs_search_params['min_dropout_rate'], \
                   bfs_search_params['max_dropout_rate'])

# Provision structure using new parameters dictionary
# Input layer
input_layer = Input(shape=(bfs_search_params['num_features'],))
encoded = input_layer

# Encoder hidden layers
# Add hidden layers after the input layer of the encoder
# (idx 1 to n)
for i in range(1,initial_autoencoder_params['num_layers_encoder']):
    encoded = \
    Dense(initial_autoencoder_params['num_nodes_encoder'][i],\
                    activation = \
                    bfs_search_params['activation_encoder'])\
                    (encoded)
    # Check if Dropout is enabled
    if bfs_search_params['dropout_rate'] != 0.0:
        encoded = \
        Dropout(initial_autoencoder_params['dropout_rate'])\
        (encoded)

decoded = encoded

# Decoder hidden layers
# Add hidden layers before output layer of decoder
# (idx 0 to n-1)
for i in range(initial_autoencoder_params['num_layers_decoder']-1):
```

```
            decoded = \
            Dense(initial_autoencoder_params['num_nodes_decoder'][i], \
                  activation = \
                  bfs_search_params['activation_decoder'])\
                  (decoded)
            # Check if Dropout is enabled
            if bfs_search_params['dropout_rate'] != 0.0:
                decoded = \
                Dropout(initial_autoencoder_params['dropout_rate'])\
                (decoded)

        # Output layer
        # Set output layer = size of last item in the dictionary
        decoded = \
        Dense(initial_autoencoder_params['num_nodes_decoder'][-1], \
              activation = \
              bfs_search_params['activation_output'])(decoded)

    # Define initial autoencoder and encoder models
    initial_autoencoder = Model(input_layer, decoded)
    initial_encoder = Model(input_layer, encoded)

    # Compile model
    initial_autoencoder.compile(optimizer = \
                                bfs_search_params['optimizer'], \
                                loss = \
                                bfs_search_params['loss'])

    print('Summary of initial autoencoder model: \n \n')
    print(initial_autoencoder.summary())

    return initial_autoencoder, initial_encoder, initial_autoencoder_params


## train_autoencoder(): Trains an autoencoder ###

def train_autoencoder(autoencoder, initial_autoencoder_params, \
                      x_train, x_valid):

    print('Training autoencoder model... \n \n')
    ae_history = autoencoder.fit(x_train, x_train,
              validation_data = (x_valid, x_valid),
              epochs = initial_autoencoder_params['epochs'],
              batch_size = initial_autoencoder_params['batch_size'],
              callbacks = initial_autoencoder_params['callbacks'],
              shuffle = True,
              verbose = 2)

    score = autoencoder.evaluate(x_valid, x_valid, verbose = 0)
    print('Score is:', score, '\n \n')

    return score, ae_history


## is_model_in_list() : Check if model is in list_of_models ##

def is_model_in_list(model, list_of_models):

    autoencoder = model[0]
    num_layers_autoencoder = len(autoencoder.layers)
```

```
    num_models = len(list_of_models)

    # If list_of_models is empty, return False
    if num_models < 1 :
        return False

    # Otherwise, check if there is an autoencoder in list_of_models
    # with the same architecture as the given model
    for i in range(num_models):
            num_layers = len(list_of_models[i][0].layers)

            # If number of layers is different, ignore and continue
            if num_layers != num_layers_autoencoder:
                continue

            # Otherwise, check if nodes are equal for all layers
            flag = True
            for j in range(num_layers):
                if list_of_models[i][0].layers[j].output_shape != \
                autoencoder.layers[j].output_shape :
                    flag = False

            # If model already exists return True
            if flag == True:
                return True

    return False


## add_model_to_list(): Add model to list_of_models if it does not exist ##

def add_model_to_list(model, list_of_models):

    if is_model_in_list(model, list_of_models) is not True:
        list_of_models.append(model)
    return list_of_models


## clamp(): Clamp a number between a minimum and maximum value range. ##

def clamp(x, min_value, max_value):

    # If the current configuration is lower than the min defined, return min
    if x < min_value:
        return min_value

    # If current config is greater than the max defined, return max
    elif x > max_value:
        return max_value

    # Else no clamping
    else :
        return x


## model_mutation(): Perform random mutation of given model ##

def model_mutation(best_autoencoder_params):
```

```
# Copy autoencoder parameters to be mutated
new_autoencoder_params = best_autoencoder_params.copy()

# Mutate batch size
# range: -delta to +delta
new_autoencoder_params['batch_size'] = \
best_autoencoder_params['batch_size'] + \
random.randint(-best_autoencoder_params['delta_batch_size'], \
               best_autoencoder_params['delta_batch_size'])
# clamp between: min and max
new_autoencoder_params['batch_size'] = \
clamp(new_autoencoder_params['batch_size'], \
      best_autoencoder_params['min_batch_size'], \
      best_autoencoder_params['max_batch_size'])

# Mutate dropout rate
# Check if Dropout is enabled
if bfs_search_params['dropout_rate'] != 0.0:
    # range: -delta to +delta
    new_autoencoder_params['dropout_rate'] = \
    best_autoencoder_params['dropout_rate'] + \
    random.uniform(-best_autoencoder_params['delta_dropout_rate'], \
                   best_autoencoder_params['delta_dropout_rate'])
    # clamp between: min and max
    new_autoencoder_params['dropout_rate'] = \
    clamp(new_autoencoder_params['dropout_rate'], \
          best_autoencoder_params['min_dropout_rate'], \
          best_autoencoder_params['max_dropout_rate'])

# Mutate number of layers
# range: -delta to +delta
new_autoencoder_params['num_layers_encoder'] = \
best_autoencoder_params['num_layers_encoder'] + \
random.randint(-best_autoencoder_params['delta_num_layers_encoder'], \
               best_autoencoder_params['delta_num_layers_encoder'])
# clamp between: min and max
new_autoencoder_params['num_layers_encoder'] = \
clamp(new_autoencoder_params['num_layers_encoder'], \
      best_autoencoder_params['min_num_layers_encoder'], \
      best_autoencoder_params['max_num_layers_encoder'])
new_autoencoder_params['num_layers_decoder'] = \
new_autoencoder_params['num_layers_encoder'] - 1

# Mutate number of nodes
# Mirror mutation for intermediate layers
# return a new array of given shape and type, filled with zeros
new_autoencoder_params['num_nodes_encoder'] = \
np.zeros(new_autoencoder_params['num_layers_encoder'], dtype = int)
new_autoencoder_params['num_nodes_decoder'] = \
np.zeros(new_autoencoder_params['num_layers_decoder'], dtype = int)

# Set number of nodes for input and output layer (they must be the same)
new_autoencoder_params['num_nodes_encoder'][0] = \
best_autoencoder_params['num_nodes_encoder'][0]
new_autoencoder_params['num_nodes_decoder'][-1] = \
best_autoencoder_params['num_nodes_decoder'][-1]

# for all layers after input layer and before output layer of autoencoder
# (idx 1 to n-1)
for i in range(1,new_autoencoder_params['num_layers_encoder']-1):
```

```
    # test for errors
    try:
        # range: -delta to +delta
        new_autoencoder_params['num_nodes_encoder'][i] = \
        best_autoencoder_params['num_nodes_encoder'][i] + \
        random.randint(-best_autoencoder_params['delta_num_nodes'], \
                       best_autoencoder_params['delta_num_nodes'])
    # handle errors
    except:
        # range: -delta to +delta
        new_autoencoder_params['num_nodes_encoder'][i] = \
        round(new_autoencoder_params['num_nodes_encoder'][i-1]/2) + \
        random.randint(-best_autoencoder_params['delta_num_nodes'], \
                       best_autoencoder_params['delta_num_nodes'])
    # clamp between: min and max
    new_autoencoder_params['num_nodes_encoder'][i] = \
    clamp(new_autoencoder_params['num_nodes_encoder'][i], \
          best_autoencoder_params['min_num_nodes_encoder'], \
          best_autoencoder_params['max_num_nodes_encoder'])
    new_autoencoder_params['num_nodes_decoder'][-i-1] = \
    new_autoencoder_params['num_nodes_encoder'][i]

# Mutation for last encoder layer
idx = new_autoencoder_params['num_layers_encoder']-1
# test for errors
try:
    # range: -delta to +delta
    new_autoencoder_params['num_nodes_encoder'][idx] = \
    best_autoencoder_params['num_nodes_encoder'][idx] + \
    random.randint(-best_autoencoder_params['delta_num_nodes'], \
                   best_autoencoder_params['delta_num_nodes'])
# handle errors
except:
    # range: -delta to +delta
    new_autoencoder_params['num_nodes_encoder'][idx] = \
    round(new_autoencoder_params['num_nodes_encoder'][idx-1]/2) + \
    random.randint(-best_autoencoder_params['delta_num_nodes'], \
                   best_autoencoder_params['delta_num_nodes'])
# clamp between: min and max
new_autoencoder_params['num_nodes_encoder'][idx] = \
clamp(new_autoencoder_params['num_nodes_encoder'][idx], \
      best_autoencoder_params['min_num_nodes_encoder'], \
      best_autoencoder_params['max_num_nodes_encoder'])

print('\n', 'Model successfully mutated', '\n')

# Create mutated model
# Encoder
# Input layer
num_features = new_autoencoder_params['num_nodes_encoder'][0]
input_layer = Input(shape=(num_features,))
encoded = input_layer

# Hidden layers
# for all layers after input layer (idx 1 to n)
for i in range(1,new_autoencoder_params['num_layers_encoder']):
    encoded = \
    Dense(new_autoencoder_params['num_nodes_encoder'][i], \
          activation = \
          new_autoencoder_params['activation_encoder'])\
```

```
                (encoded)
            # Check if Dropout is enabled
            if bfs_search_params['dropout_rate'] != 0.0:
                encoded = Dropout(new_autoencoder_params['dropout_rate'])\
                (encoded)

        # Decoder
        # Hidden layers
        decoded = encoded
        # for all layers after bottleneck and before output layer
        # (idx 0 to n-1)
        for i in range(new_autoencoder_params['num_layers_decoder'] - 1):
            decoded = \
            Dense(new_autoencoder_params['num_nodes_decoder'][i], \
                  activation = \
                  new_autoencoder_params['activation_decoder'])\
                  (decoded)
            # Check if Dropout is enabled
            if bfs_search_params['dropout_rate'] != 0.0:
                decoded = Dropout(new_autoencoder_params['dropout_rate'])\
                (decoded)

        # Output layer
        decoded = \
        Dense(new_autoencoder_params['num_nodes_decoder'][-1], \
              new_autoencoder_params['activation_output'])(decoded)

        # Define encoder and autoencoder
        new_autoencoder = Model(input_layer, decoded)
        new_encoder = Model(input_layer, encoded)

        return new_autoencoder, new_encoder, new_autoencoder_params


## best_model_search(): Perform best-first search ##

def best_model_search(list_of_models, initial_autoencoder_params, \
                      x_train, x_valid):

        # Init variables
        num_models = len(list_of_models)
        best_autoencoder = list_of_models[0][:]
        best_score = list_of_models[0][-1]
        best_autoencoder_params = initial_autoencoder_params
        best_history = Sequential()

        # Begin search
        while best_score > initial_autoencoder_params['epsilon'] and \
        num_models < initial_autoencoder_params['max_num_models']:
            # Create new model by random mutation
            new_autoencoder, new_encoder, new_autoencoder_params = \
            model_mutation(best_autoencoder_params)
            print('Summary of new autoencoder model: \n \n')
            print(new_autoencoder.summary(), '\n')

            # Ignore new model if it already exists and restart the loop
            if is_model_in_list([new_autoencoder, 0, 0], list_of_models):
                continue
```

```
        # Otherwise, train and evaluate the model
        new_autoencoder.compile(optimizer = \
                                initial_autoencoder_params['optimizer'], \
                                loss = \
                                initial_autoencoder_params['loss'])
        new_score, new_history = \
        train_autoencoder(new_autoencoder, new_autoencoder_params, \
                          x_train, x_valid)

        # Add the new model to the list of models
        new_model = [new_autoencoder, new_encoder, new_score]
        list_of_models = add_model_to_list(new_model, list_of_models)

        # Sort list of models according to score (small to large)
        list_of_models = sorted(list_of_models, key = lambda x: x[-1])

        print(num_models+1, '/', \
              initial_autoencoder_params['max_num_models'], \
              'models in list', '\n')
        print('List of scores so far (lower is better):')
        for i in range(len(list_of_models)):
            print(list_of_models[i][-1])
        print('\n \n')

        # Update best model and score - it is always the first model in list
        best_autoencoder = list_of_models[0][0]
        best_encoder = list_of_models[0][1]
        best_score = list_of_models[0][-1]

        # Update best model parameters
        if new_score == best_score:
            best_autoencoder_params = new_autoencoder_params
            best_history = new_history

        # Update number of models
        num_models = num_models + 1

        print('Summary of best autoencoder model: \n \n')
        print(best_autoencoder.summary(), '\n')

    return list_of_models, best_autoencoder, best_encoder, best_score, \
    best_autoencoder_params, best_history


## encode_data(): Encode data using best found encoder ##

def encode_data(x_train, y_train, x_valid, y_valid, x_test, y_test, \
                best_encoder):

    print('Encoding data...')
    # Encode training dataset
    x_train_encoded = best_encoder.predict(x_train)
    x_train_encoded = np.asarray(x_train_encoded).astype('float32')
    y_train_array = np.asarray(y_train).astype('float32')
    print('Encoded training dataset')

    # Encode validation dataset
    x_valid_encoded = best_encoder.predict(x_valid)
    x_valid_encoded = np.asarray(x_valid_encoded).astype('float32')
    y_valid_array = np.asarray(y_valid).astype('float32')
```

```
    print('Encoded validation dataset')

    # Encode testing dataset
    x_test_encoded = best_encoder.predict(x_test)
    x_test_encoded = np.asarray(x_test_encoded).astype('float32')
    y_test_array = np.asarray(y_test).astype('float32')
    print('Encoded testing dataset')
    print('\n')

    return x_train_encoded, y_train_array, x_valid_encoded, \
    y_valid_array, x_test_encoded, y_test_array


## create_mlp(): Create an MLP classifier ##

def create_mlp(mlp_params):

    print('Creating MLP classifier...', '\n')
    mlp_classifier = Sequential()
    mlp_classifier.add(Dense(mlp_params['num_nodes'][1] , \
                             activation =
                             mlp_params['activations'][0], \
                             input_shape = \
                             (mlp_params['num_nodes'][0],)))

    for i in range(1,len(mlp_params['num_nodes'])-1):
        mlp_classifier.add(Dense(mlp_params['num_nodes'][i+1], \
                                 activation = \
                                 mlp_params['activations'][i]))

    mlp_classifier.compile(loss = mlp_params['loss'],
                           optimizer = mlp_params['optimizer'],
                           metrics = mlp_params['metrics'])

    print('Summary of MLP classifier model: \n \n')
    print(mlp_classifier.summary(), '\n')

    return mlp_classifier


## train_mlp(): Train the MLP classifier ##

def train_mlp(mlp_classifier, x_train_encoded, y_train_array, \
              x_valid_encoded, y_valid_array, mlp_params):

    print('Training of MLP classifier: \n \n')
    mlp_history = mlp_classifier.fit(x_train_encoded, y_train_array,
                  validation_data = (x_valid_encoded, y_valid_array),
                  epochs = mlp_params['epochs'],
                  batch_size = mlp_params['batch_size'],
                  callbacks = mlp_params['callbacks'],
                  verbose = 3)

    print('\n \n')

    return mlp_history
```

```
## test_mlp() : Test the MLP classifier ##

def test_mlp(mlp_classifier, x_test_encoded, y_test_array):

    print('Evaluation of MLP classifier on test set: \n \n')
    # The threshold is set to 0.5 to account for unbalanced classes
    mlp_classes = \
    (mlp_classifier.predict(x_test_encoded)>0.5).astype("int32")
    mlp_classes = \
    mlp_classes[:, 0]

    # accuracy: (tp + tn) / (p + n)
    accuracy = accuracy_score(y_test_array, mlp_classes)
    print('Accuracy: %f' % accuracy)
    # precision tp / (tp + fp)
    precision = precision_score(y_test_array, mlp_classes)
    print('Precision: %f' % precision)
    # recall: tp / (tp + fn)
    recall = recall_score(y_test_array, mlp_classes)
    print('Recall: %f' % recall)
    # f1: 2 tp / (2 tp + fp + fn)
    f1 = f1_score(y_test_array, mlp_classes)
    print('F1 score: %f' % f1)
    # kappa
    kappa = cohen_kappa_score(y_test_array, yhat_classes)
    print('Cohens kappa: %f' % kappa)
    # ROC AUC
    auc = roc_auc_score(y_test_array, yhat_probs)
    print('ROC AUC: %f' % auc)
    # confusion matrix
    matrix = confusion_matrix(y_test_array, yhat_classes)
    print('Confusion Matrix:\n', matrix)

    print('\n \n')

## Preparation ##

# Load dataset
training_data, testing_data = load_data()

# Pre-process dataset
validation_split = 0.1
x_train, y_train, x_valid, y_valid, x_test, y_test, num_features = \
preprocess_data(training_data, testing_data, validation_split)

# Initialise empty list for best-first search algorithm
list_of_models = create_models_list()


## Initialise Hyperparameters ##

# Pre-defined model parameters
bfs_search_params = {}
bfs_search_params['num_layers_encoder'] = 4
bfs_search_params['num_nodes_encoder'] = [num_features, 80, 122, 80]
bfs_search_params['num_layers_decoder'] = 3
bfs_search_params['num_nodes_decoder'] = [122, 80, num_features]
bfs_search_params['batch_size'] = 64
```

```
# Global parameters
bfs_search_params['initial_model'] = 'random'
bfs_search_params['num_features'] = num_features
bfs_search_params['activation_encoder'] = 'relu'
bfs_search_params['activation_decoder'] = 'relu'
bfs_search_params['activation_output'] = 'sigmoid'
bfs_search_params['optimizer'] = 'adam'
bfs_search_params['loss'] = 'binary_crossentropy'
bfs_search_params['early_stopping_patience'] = 35
bfs_search_params['dropout_rate'] = 0.1
bfs_search_params['epochs'] = 400
bfs_search_params['callbacks'] = [
    EarlyStopping(patience = bfs_search_params['early_stopping_patience'])]

# Search parameters
bfs_search_params['epsilon'] = 1e-6
bfs_search_params['max_num_models'] = 390
bfs_search_params['delta_num_layers_encoder'] = 1
bfs_search_params['delta_num_layers_decoder'] = 1
bfs_search_params['min_num_layers_encoder'] = 3
bfs_search_params['max_num_layers_encoder'] = 7
bfs_search_params['min_num_layers_decoder'] = 3
bfs_search_params['max_num_layers_decoder'] = 7
bfs_search_params['delta_num_nodes'] = 10
bfs_search_params['min_num_nodes_encoder'] = 15
bfs_search_params['max_num_nodes_encoder'] = 122
bfs_search_params['min_num_nodes_decoder'] = 15
bfs_search_params['max_num_nodes_decoder'] = 122
bfs_search_params['delta_dropout_rate'] = 0.1
bfs_search_params['min_dropout_rate'] = 0.05
bfs_search_params['max_dropout_rate'] = 0.5
bfs_search_params['delta_batch_size'] = 10
bfs_search_params['min_batch_size'] = 20
bfs_search_params['max_batch_size'] = 122

print('Hyperparameters initialised', '\n')


## Initial model ##

# Create initial autoencoder model
initial_autoencoder, initial_encoder, initial_autoencoder_params = \
create_initial_autoencoder(bfs_search_params)

# Train initial autoencoder model
if len(list_of_models) < 1:
    initial_score, initial_history = \
    train_autoencoder(initial_autoencoder, \
                      initial_autoencoder_params, \
                      x_train, x_valid)
else :
  print('List is not empty. Halting training')

# Add initial model to list
initial_model = [initial_autoencoder, initial_encoder, initial_score]
list_of_models = add_model_to_list(initial_model, list_of_models)

# Check number of models in list
print('Number of models: ', len(list_of_models))
```

```
## Best-first search ##

# Initiate the best-first search
list_of_models, best_autoencoder, best_encoder, best_score, \
best_autoencoder_params, best_history = \
best_model_search(list_of_models, initial_autoencoder_params, x_train,
x_valid)


## Encode dataset ##

# Encode dataset using the best encoder model
x_train_encoded, y_train_array, x_valid_encoded, y_valid_array, \
x_test_encoded, y_test_array = \
encode_data(x_train, y_train, x_valid, y_valid, x_test, y_test, best_encoder)


## Define, create, train and test MLP classifier ##

# MLP classifier parameters
mlp_params = {}
mlp_params['num_nodes'] = \
[best_encoder.layers[-1].output_shape[1], 8, 4, 2, 1]
mlp_params['activations'] = ['relu', 'relu', 'relu', 'sigmoid']
mlp_params['loss'] = 'binary_crossentropy'
mlp_params['optimizer'] = 'adam'
mlp_params['epochs'] = 300
mlp_params['batch_size'] = 64
mlp_params['metrics'] = 'accuracy'
mlp_params['callbacks'] = [
    EarlyStopping(monitor = 'accuracy', patience = 10),
    ReduceLROnPlateau(monitor ='val_loss',
                      factor = 0.1,
                      patience = 15,
                      min_lr = 0.0005,
                      verbose = 0)]

# Create MLP classifier
mlp_classifier = create_mlp(mlp_params)

# Train MLP classifier to classify encoded dataset
mlp_history = \
train_mlp(mlp_classifier, x_train_encoded, y_train_array, \
          x_valid_encoded, y_valid_array, mlp_params)

# Test MLP classifier performance on encoded dataset
test_mlp(mlp_classifier, x_test_encoded, y_test_array)
```

ANNEXURE C: CONFIRMATION OF LANGUAGE EDITING

This serves to confirm that I, Isabella Johanna Swart, registered with and accredited as professional translator by the South African Translators' Institute, registration number 1001128, language edited the following dissertation:

Intrusion detection using artificial neural networks and supervised deep learning

by

JH Smulders

Dr Isabel J Swart

23 Poinsettia Close
Van der Stel Park
Dormehlsdrift
GEORGE
6529
Tel: (044) 873 0111
Cell: 082 718 4210
e-mail: isaswart@telkomsa.net

Date: 8 November 2021