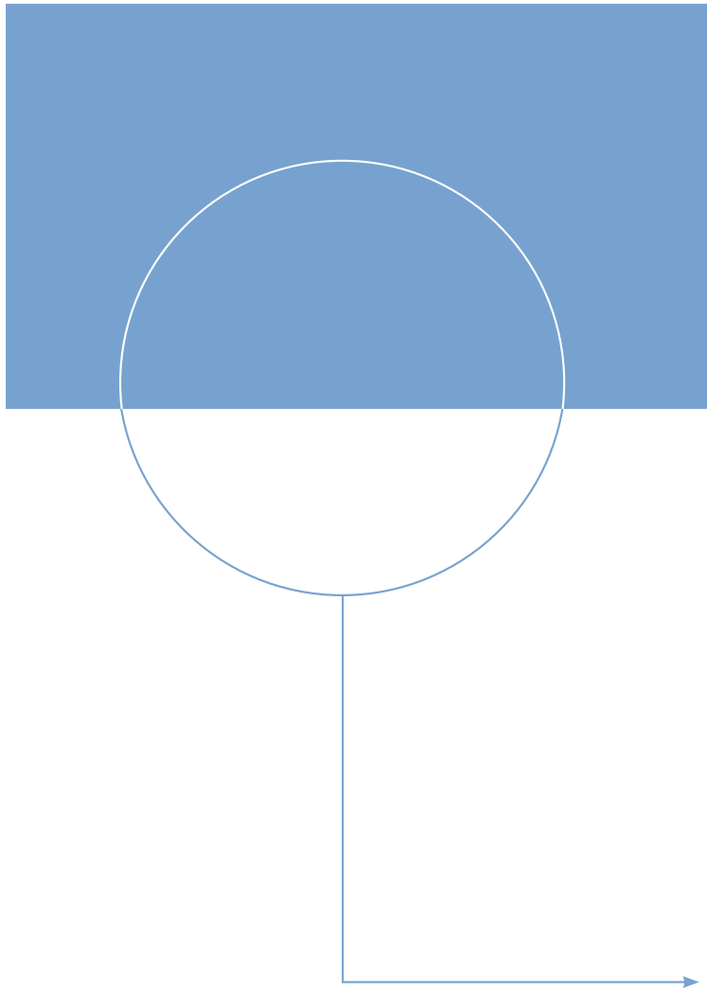


Master's thesis

2022

Lars Gunnar Thingnes

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Information Security and Communication Technology

Lars Gunnar Thingnes

Firewall models in cloud environments

June 2022

Eidsiva

NTNU
Norwegian University of Science and Technology

NTNU
Norwegian University of Science and Technology

NTNU
Norwegian University of
Science and Technology

Firewall models in cloud environments

Lars Gunnar Thingnes

Master of Science in Information Security
Submission date: June 2022
Supervisor: Slobodan Petrovic
Co-supervisor: Håkon Gunleifsen Ph.D

Norwegian University of Science and Technology
Department of Information Security and Communication
Technology

Lars Gunnar Thingnes

Firewall models in cloud environments

Master's thesis in Information Security
Supervisor: Prof. Slobodan Petrović
Co-supervisor: Håkon Gunleifsen Ph.D
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering Department
of Information Security and Communication Technology

NTNU

Avdeling for
informatikk og medieteknikk
Norges Teknisk-Naturvitenskapelige Universitet
Postboks 191
2802 Gjøvik

Faculty of Computer Science
and Media Technology
Norwegian University of Science and Technology
Box 191
N-2802 Gjøvik
Norway

Acknowledgements

I would like to thank Prof. Slobodan Petrović for guidance and feedback throughout the course of this thesis. I cannot begin to express my thanks to Håkon Gunleifsen Ph.D for supervision and discussions through the work of this thesis. I must also send gratitude to my superiors at Eidsiva who has given me time to write this thesis. Finally, I'd like to express my deepest thanks to my family for tolerating my absence through several years with studies, which has culminated with this thesis.

Abstract

This thesis investigates how traditional centralised and zone-based firewalls can be migrated to modern micro-segmented and virtually distributed firewalls. This migration is considered a high risk because it requires a translation from specific firewall rules to more abstract security policies. Further, the primary objective is to investigate if machine learning (ML) can be used to abstract firewall policies from one environment and migrate these policies to another environment. Four different experiments were conducted using a deep neural network (DNN) to classify data from a live production environment into new firewall policies. Network traffic representing the firewall policies, together with data from the infrastructure were analysed by the DNN. The results from the experiments are very promising showing a clear indication that machine learning can be used in such a task. The DNN performs with a very high degree of accuracy if there are enough data for the DNN to analyse. Finally, a model on how an automatic process of migrating firewall policies between different cloud environments is proposed. By training a general DNN, this model can be used to analyse infrastructure environments and automatically migrate the security policy between cloud environments. Although there is much future work, the DNN together with the proposed model is to be considered as groundwork proving that is possible to use of ML in an automated process of moving between different firewall environments.

Sammendrag

Denne oppgaven undersøker hvordan tradisjonelle sentraliserte og sonebaserte brannmurer kan migreres til moderne mikrosegmenterte og virtuelt distribuerte brannmurer. En slik migrering anses som en høy risiko siden den krever en oversettelse fra spesifikke brannmurregler til mer abstrakte sikkerhetspolicyer. Videre er hovedmålet å undersøke om maskinlæring (ML) kan brukes til å abstrahere brannmurpolicyer fra ett miljø og migrere disse policyene til et annet miljø. Fire forskjellige eksperimenter ble utført ved å bruke et dypt nevralt nettverk til å klassifisere data fra et produksjonsmiljø til nye brannmurpolicyer. Nettverkstrafikk som representerer brannmurpolicyene, sammen med data fra infrastrukturen ble analysert av det neurale nettverket. Resultatene fra eksperimentene er svært lovende og viser en klar indikasjon på at maskinlæring kan brukes i en slik oppgave. Det neurale nettverket klassifiserer data med en meget høy grad av nøyaktighet så lenge det er nok data for nettverket å analysere. Til slutt foreslås en modell for hvordan en automatisk prosess for migrering av brannmurpolicyer mellom ulike skymiljøer kan fungere. Ved å trene opp et generelt dypt neuralt nettverk, kan denne modellen brukes til å analysere infrastrukturmiljøer og automatisk migrere sikkerhetspolicyen mellom ulike skymiljøer. Selv om det er mye fremtidig arbeid, er DNN sammen med den foreslåtte modellen å betrakte som grunnarbeid som beviser at det er mulig å bruke ML i en automatisert prosess for å flytte mellom ulike brannmurmiljøer.

Contents

Figures

Tables

Code Listings

Chapter 1

Introduction

1.1 Keywords

Cloud technology, cloud security, firewalls, micro-segmentation, machine learning, deep learning, deep neural networks

1.2 Problem description

National Institute of Standards and Technology (NIST) defines cloud computing as a model for using services on a shared computer environment, through the use of Internet. These resources can be networks, servers, storage, applications, and services that quickly can be allocated and delivered to the customer with little effort by the service provider[1]. The concept of cloud has in the recent years developed further, connecting different cloud environments together or migrating between different cloud environments. Software-Defined Networks (SDN) and overlay networks enable micro-segmentation and firewalling per virtual machine [2]. The idea of migrating virtual machines and containers between different clouds and hardware is motivated by better scaling, more flexibility, lower costs, and decreased time to market. In cloud environments, the network security parameters are converged with both virtual hardware, virtual networks, and software policies. In cloud infrastructure, micro-segmentation is used to provide Information and communications technology (ICT) security. Micro-segmentation has rules that can be classified by virtual hardware and application attributes. This opens for extending the traditional firewall attributes and make more sophisticated and abstract firewall rules. Ultimately, this results in fewer policy rules and simpler management. However, this implies that the firewall rules also can be changed unintentionally, e.g. by changing the name of a virtual machine. Additionally, more firewall attributes in the virtual environment can result in more complex policies. SDN is utilised to disconnect the forwarding-plane of network resources and the management plane. In cloud environments, SDN is used to control the network using automation. If such automation is to be used in its full ex-

tent, there is a need for more general firewall rules that can be used to program new virtual components automatically when needed. This thesis focuses on the ICT-security in cloud firewall models and analysing current firewall rules using machine learning (ML) to create more generic firewall rules. Figure 1.1 shows a scheme on how an on-premise firewall model can be abstracted by ML and migrated to a cloud environment. The outcome of the thesis is to create a model that can make it easier for organisations and companies to make use of cloud technology while maintaining ICT-security.

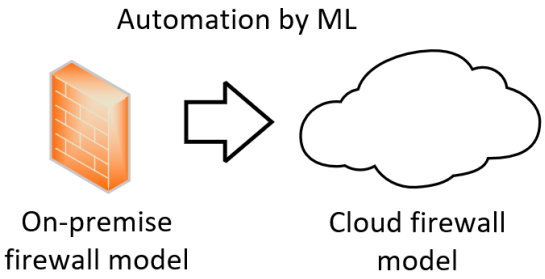


Figure 1.1: Model of the problem description

1.3 Scope of the Thesis

The scope of this thesis is to research what types of firewall models there are in virtual environments and if possible, how to automatically migrate firewall policies between them. By analysing data from one firewall model situated in one virtual environment it may be possible to classify this data into new policies that can be applied in another environment. Such policies can be complex and can consist of many overlapping rules that need to be prioritised by a Security Policy Description Language (SPDL). Such a policy language is not to be considered a part of this thesis but is essential if such a process should work. A hypothesis is that Machine Learning (ML) can provide a classifier for such a problem. The hypothesis is based on the fact that ML has previously been applied to classify other types of data with great success. To test this hypothesis, a *Deep Learning* model is designed and used to classify data from a live environment. Based on the results, it is possible to model a method on how to apply ML in a larger system to automatically migrate firewall policies between different cloud environments.

1.4 Justification, motivation and benefits

In the modern world, the society relies on critical infrastructure to function properly. Companies, such as Internet Service Providers (ISP) and Cloud Service Provider (CSP) provide such critical infrastructure and are responsible for providing

secure and reliable services. Telecommunication infrastructure is defined as critical and can be disturbed if not the necessary security policy is applied [3]. It is vital that the different service providers secure their infrastructure by using firewalls to isolate secure from insecure networks. Traditionally, a perimeter defence has been applied by firewalls to remove unwanted traffic. When moving to a cloud environment, the security can be compromised if the same firewall rules are not correctly ported to the new environment. When moving firewall rules, there is a need to mitigate such security breaches and ensure that moving firewall policies into a cloud environment can be effectuated in a safe way. By providing features that are specific for the different CSPs, there are challenges for organisations to migrate from one CSP to another. Such a risk is called a vendor lock-in and makes it difficult for organisations to adopt a multi-cloud strategy to migrate between many cloud environments [4]. The motivation for this project is to make it easier for organisations and companies to migrate firewall rules to a micro-segmented infrastructure. When an organisation migrates from one cloud model to another, there is a need for migrating the associated firewall rules to the new cloud environment. Such migration of firewall rules is a perquisite for adopting a multi-cloud strategy. If firewall rules cannot be migrated in a secure way, organisations have no secure method to move between cloud-environments that provides the best service at the time.

1.5 Research questions

Bearing in mind the discussion from the previous section, the following research questions are to be answered in this thesis:

- Question 1. How secure are the existing different models in virtualised environments?

  In a cloud environment, there are several considerations that must be taken when creating a secure system. When connecting several different cloud environments together, the traditional perimeter firewall is obsolete. The goal of this research question is to investigate the different firewall models and to outline the differences between them to recommend a model for a multi-cloud environment.

- Question 2. How can a dataset of firewall rules from a traditional perimeter firewall be abstracted into a new set of policy rules in the virtual environment?

  The main objective of this research is to investigate how it is possible to migrate from a traditional perimeter firewall to a micro-segmented environment by using other policy rules than in a traditional firewall architecture. Micro-segmentation provides other protocols with other features than the

traditional IP/port. An hypothesis is that it is possible to abstract traditional policies into more generic policies by using other features than traditional firewall rules.

- Question 3. How can a dataset of firewall rules migrate between cloud environments where the CSP has different firewall models?

  By using abstracted firewall policies, it can be possible to automatically migrate these policies between different cloud environments. Such a process has to be secure in order to maintain the integrity of the policies. How such a migration can be done must be investigated.

1.6 Thesis Outline

Chapter 1 states the problem description, defines the scope, justification, and outlines the research questions of this thesis. Chapter 2 provides the background needed for understanding the concepts and theory discussed in this thesis. It elaborates on cloud computing and cloud technology and the differences between different cloud environments. This chapter also describes micro-segmentation and how this technology can secure a virtual environment. In addition to this, the concepts of *artificial intelligence* and *machine learning* is described and how the output parameters of such technology is interpreted. Chapter 3 focuses on the theoretical contributions of this thesis. It describes the proposed firewall model, which is recommended to be used in virtualised environments. It also describes how machine learning can be applied to classify policies by analysing data from the virtual infrastructure of a live environment. Chapter 4 provides details of the methodology used in this thesis. It specifies the research strategies, and enumerates the methods used e.g., literature review, experiments, and data analysis. Chapter 5 elaborates on the experiments with neural networks that are conducted in this thesis. It describes the setup for the neural network and how the data sets are used in the experiments. The output performance from each experiment is shown and the parameters are discussed. Chapter 6 discusses the findings related to the research questions. The research questions are discussed in the context of the information obtained through the work of this thesis. Finally, Chapter 7 summarises the findings and answers the research questions. It also raises different questions that should be addressed in future work. In Chapter 8, the Python source code used to create dataset and the experiments is presented.

1.7 Contribution

We live in a world of constant innovation and a relatively rapid development of new technology. For both individuals and organisations, it can be difficult to be able to make use of such new technology even if it would benefit the organisation

in many ways. One barrier to making use of new technology is the competence and knowledge on how to make best use of the innovation. One such innovative technology is the use of cloud technology, which converts the physical dimension into an abstract idea. Behind every cloud, there are servers and network devices working together with manual operators providing a service. The goal of this thesis is to make people and organisations make use of cloud infrastructure more easily by providing a method to migrate existing security policies between different environments. In such a way, it can be possible to convert between different types of cloud services. To this end, we have used deep neural networks to model a classifier which is used to analyse data from a live virtual environment.

Chapter 2

Theoretical background

The goal of this thesis is to help people and organisations make use of cloud technology more easily by researching a method for transferring security policies from one cloud to another. To help the reader of this thesis understand the concepts behind the technology used and security used in cloud environments, basic theory is elaborated. In Chapter 5, several experiments with AI and machine learning are performed to investigate if such a method can be used. To understand these concepts, the reader must understand what machine learning is and how this technology can be applied to solve this problem. Thus, in this chapter, the theory of what machine learning is, how it is applied, and how to understand the results obtained by using such technology, is reviewed.

2.1 Cloud models

Yashpalsinh et al. (2012) describe the advantages of cloud computing as *easy management, cost reduction, uninterrupted services, disaster management and green computing* [5]. Cloud computing architectures are differentiated in service models that have unique characteristics. National Institute of Standards and Technology (NIST) defines cloud computing as a model for enabling a rapid pool of resources which can be provisioned within minimal efforts by the service provider [6]. The models are composed of five essential characteristics, three service models, and four deployment models as shown in Figure 2.1.

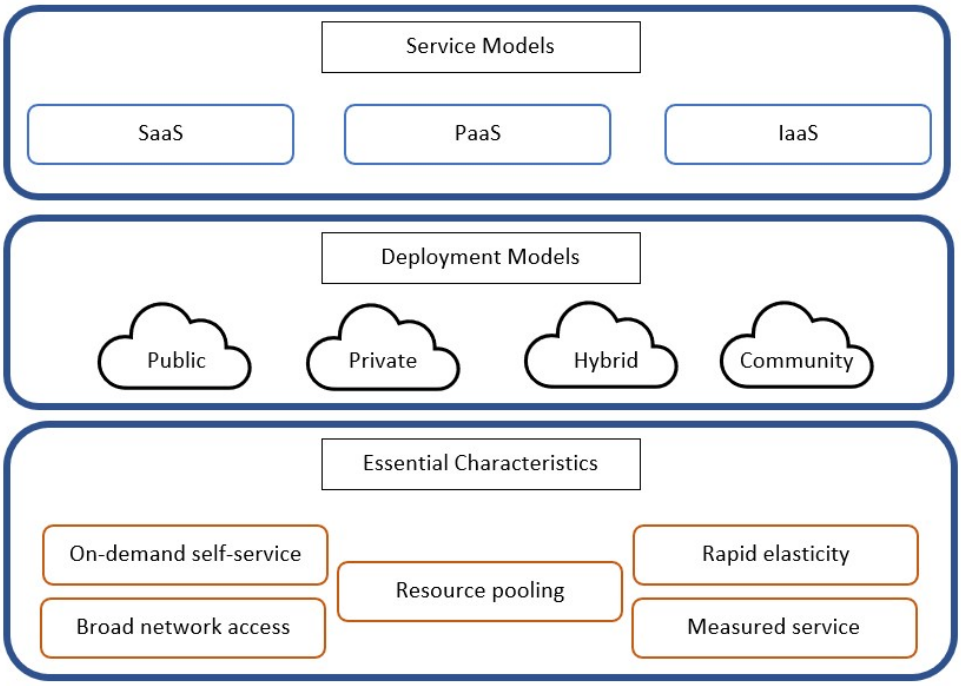


Figure 2.1: The NIST cloud model

The five essential characteristics of the NIST cloud model are [6]:

- **On-demand self-service** - A consumer can make use of computing resources (e.g. servers, network or storage), when needed, without any human touch by the service provider.
- **Broad network access** - Ubiquitous network and technical resources are available and accessed over the network through standard devices such as mobile phones, tablets, computers and other types of end user equipment.
- **Resource pooling** - The computing resources are accessible to multiple consumers in a multi-tenant model. The service provider can dynamically assign and change these resources on demand.
- **Rapid elasticity** - The computing resources can easily be provisioned and released, and preferably automated, to scale after the customer demand. For the consumer, the service seems to be unlimited and can be accessed and increased in any quantity when the service is needed by the customer.
- **Measured service** - The service provider controls and optimises the resources by constantly measuring the load on the infrastructure.

The three service models are Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). NIST defines these three models in the following way [6]:

- **SaaS** - Applications running on the cloud infrastructure that are provided to the consumer. The SaaS lets the customer use the CSPs applications which

are accessible from various clients such as a Web browser. The customer has no access to the underlying Operating System (OS) or infrastructure. Examples of services are Google Mail, Microsoft 365, Salsesforce, Citrix GoToMeeting and Cisco WebEx [7]

- **Paas** - The service provided to the consumer is running on the platform that is provided by the CSP. The PaaS let the customer create applications using the CSPs development framework. The customer has no access to the underlying OS or infrastructure. Examples of services are Microsoft Azure, Forece.com, Heroku and Apache Stratos [7].
- **IaaS** - With IaaS the customers have access to the underlying cloud infrastructure such as the OS, while the virtualisation environment is controlled by the CSP. Examples of this are virtual hypervisors such as KVM [8], VMware ESXi[9], Hyper-V [10] and so forth.

As shown in Figure 2.1, the different models are based upon different deployment models. A deployment model is how the different organisations control their computer resources and are divided into private, community, public or hybrid cloud. NIST defines the different deployment models in the following way [6]:

- **Private Cloud** - Cloud infrastructure, which provides services exclusively to a single organisation for a selection of users.
- **Community cloud** - Cloud infrastructure, which provides services to a community of users from organisations that have common interests. The cloud infrastructure can be operated and managed by one or more organisations, or by a third party. The cloud infrastructure can exist on-premise on one or more of the operators.
- **Public cloud** - Cloud infrastructure which is publicly available, managed, and operated by a business or an organisation. The cloud infrastructure exists on premise of the CSP.
- **Hybrid cloud** - Two or more cloud infrastructures (private, community, or public) that are connected through standardised communication technology.

Different cloud environments can be used to differentiate between which types of data are stored and processed on different deployment models. For instance, personal information is stored in a private cloud, while other data is stored in a public cloud. The use of multiple clouds and expanding the hybrid cloud concept has led to a fifth deployment model called multi-cloud. The concept of a multi-cloud deployment model was first introduced by *Keahey* et al. [11], which built a virtual cluster and created a trusted network distributed in multi-clouds. The concept of multi-cloud computing is a strategy that organisations use to benefit from computer resources provided by multiple service providers available through the Internet [12]. The multi-cloud strategy is made possible using fully virtualised environments, virtualised networks and the use of technology such as SDN and Application Programmable Interfaces (API) to orchestrate the management of the environment [13].

2.2 Data Center Infrastructure

The growth and complexity of Information and communications technology (ICT) has lead to the development of Data Center (DC) to store and process data. The infrastructure in DC is built to operate servers that produces the ICT services. In a traditional DC-architecture, each server controls and operates all the layers of hardware and applications to create services installed on the operating system (OS). To operate more efficient by the concept of virtualisation has been developed. With this technology, several systems can be hosted on the same server sharing the same computer resources. To control the virtualisation, a software component called a *hypervisor* is installed on the server. Popek et al. [14] classified two types of hypervisors: Type-1, native or bare-metal hypervisors and Type-2 or hosted hypervisors. The Type-1 hypervisor is an OS installed directly on the server where guest OS are hosted on the hypervisor. A Type-2 hypervisor is installed as a software application on the OS. Guest operating system is installed on the Type-2 hypervisor, which then operates between the operating system and the guest system. A guest operating system that is installed on a hypervisor is known as a virtual machine (VM) and is a virtual system that operates in an independent enclosed virtual environment containing OS, libraries and binaries and the applications. Deploying VMs with OS, binaries and applications consumes resources. As a result of this, a new type of virtualised lightweight instances called containers have been developed [15]. This is a relatively new type of virtualisation, where an application with all its dependencies is packed in a ready-to-deploy self-contained virtual instance that can be deployed on the infrastructure. Figure 2.2 presents a scheme of the different virtualisation architectures models.

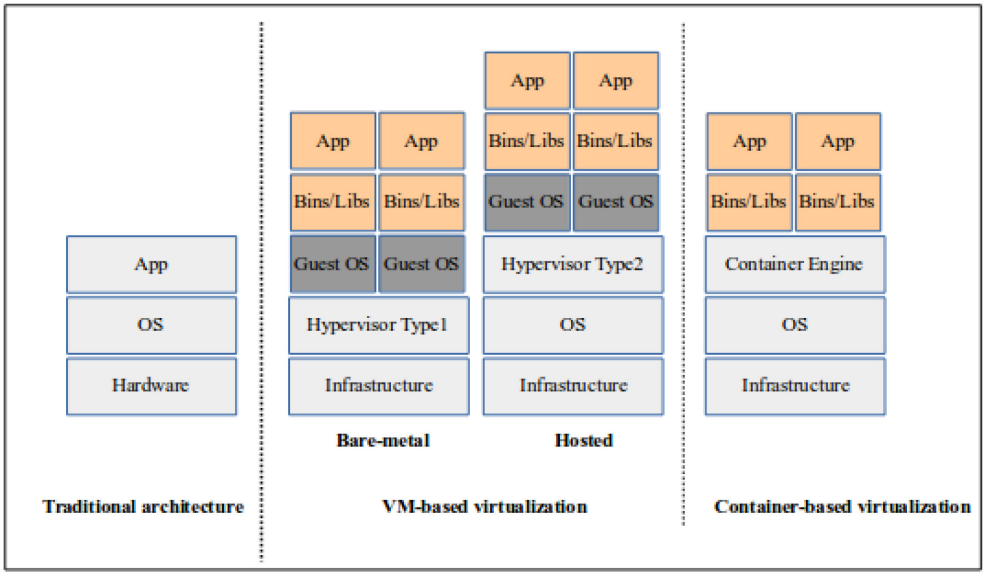


Figure 2.2: Virtualisation architectures [16]

The hypervisor also provides the VM with network connectivity. A hypervisor has a virtual switch (vSwitch) which is a logical network device that separates the data plane from the management plane [17]. In such a way, communication within the same DC is provided directly within the DC. Network communication within the same DC is also referred to as east-west communication. Figure 2.3 presents a simplified scheme of a DC and how traffic east-west in the DC between different VMs uses the vSwitch to communicate with each other. Communication to and from VMs in a DC with services outside the DC is called north-south communication. An example of north-south traffic is communication between VMs located in the DC and internet services.

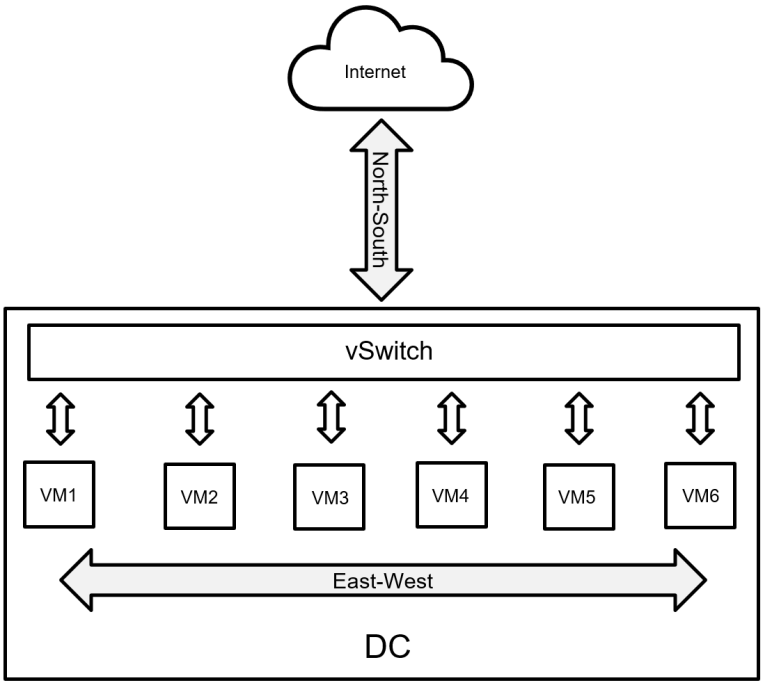


Figure 2.3: Communication in a DC

2.2.1 Network segmentation

Miller et al. (2015), claim that the traditional approach to network security can be compared to medieval castle defence. The network administrators are doing this by building high thick walls around the assets that they want to protect, only opening certain doors that are inspected to keep the adversaries outside. In computer networks, the thick walls are the perimeter firewalls that inspect, drop, or allow communication in and out of the network [18]. The weakness of this type of defence is that if an adversary manages to penetrate the perimeter defence, there are no defence mechanisms in place to limit the adversary to move laterally through the network and compromise more systems in the network. This

vulnerability is known by system administrators and to reduce the risk, networks serving different departments within an organisation are segmented depending on the kind of service that is provided on the network. An example is to segment Operational Technology (OT) from administrative networks. In such a way, the defenders build many smaller forts protecting groups of assets. The reason for such segmentation is to limit the risk that all the digital assets in an organisation are compromised in case of a security breach such as e.g E-mail or phising. Technically, such segmentation is done by dividing the Local Area Network (LAN) [19] into different Virtual Local Area Networks (VLAN) [20] or Virtual Routing and Forwarding (VRF) [21]. Thus, communication is disallowed from one part of the network to access another part of the network. A barrier can be a firewall allowing traffic to flow between the two separated networks while inspecting the traffic. As shown in Figure 2.4, a perimeter firewall controls communication to and from internet services while an internal firewall controls the traffic between different zones in the same network. In this example, an internal firewall has two different zones, each with its own network and connected systems.

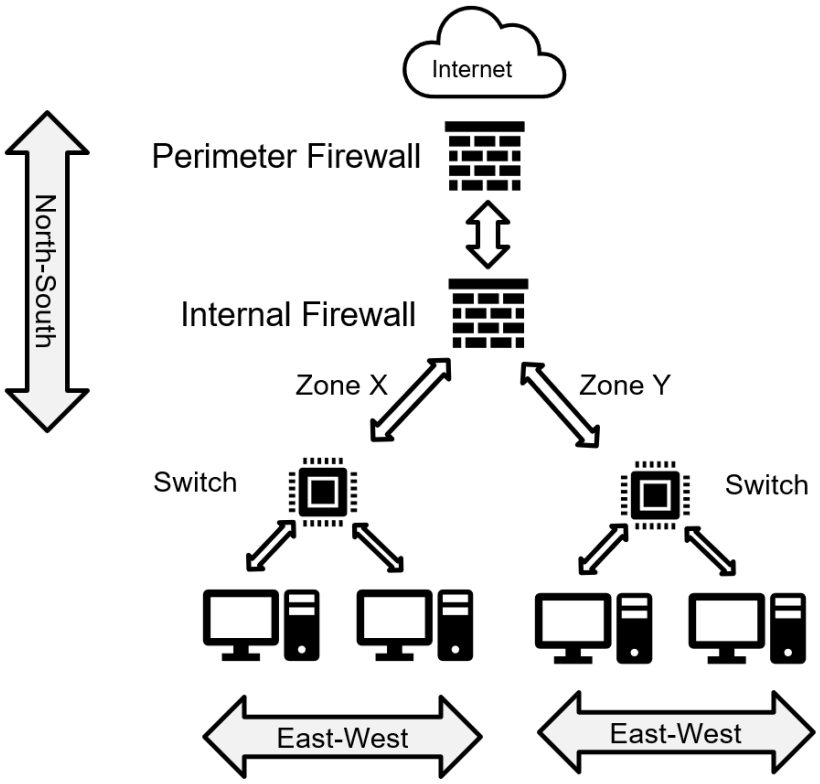


Figure 2.4: Traditional network architecture

In a firewall architecture called bastion host [7], the services that are provided to

be used outside the local network, such as Web-services or Domain Name Services (DNS) are put in a special zone called Demilitarised Zone (DMZ)[22]. Between these different networks, a firewall is configured as a perimeter defence controlling the network traffic going north-south and east-west. Organisations have different sets of guidelines of how things should work within the organisation and how the organisation should react in an interaction with other organisations. Such sets of ideas and plans are called policies that are enforced in a set of rules on how the firewall should behave. In a small network, the number of rules is small. In larger organisations with large computer networks with multiple services, the sets of firewall rules are more complex, resulting in a much more complex set of firewall rules to enforce the company policies.

2.2.2 Firewalls

In traditional computer networks, the system administrator uses a firewall to separate the computer network between trusted and untrusted sections of the network. Firewalls are used to separate different zones to control the network communication between these zones. To mitigate the risk of an infection, and in case of an incident, firewalls prevent the attacks to spread to the whole network. Malware can spread through the network and compromise systems by exploiting vulnerabilities in the infrastructure. A firewall can limit the time it takes for the malware to spread through the computer network and can help the network administrator mitigate the threat. In a computer network, a firewall operates from layer 3 and up to layer 7 in the Open Systems Interconnect (OSI) model depending on the specifications on the firewall. There are two types of firewalls: stateful and stateless. Gouda et al. (2007) defined the difference between a stateful and a stateless firewall by how the firewall inspected packets and either drops or allows packets [23]. A stateful firewall keeps track on active sessions and allows or denies packets based upon what the firewall has accepted previously. Typically, a computer network consists of both stateful and stateless firewalls depending on the network architecture. The firewall rules are configured to allow or deny communication to and from a destination at a specific port. Figure 2.1 presents an example of firewall rules where a specific IP or IP-range is allow to communicate with another IP or IP-range on a specific port or port range.

Table 2.1: An example of firewall rules

No	Protocoll	Source IP	Dest. IP	Dest. Port	Action
1	TCP	10.1.1.1	20.1.1.1	80	Accept
2	TCP	10.1.1.1	20.1.1.1	443	Accept
3	TCP	10.1.1.0/24	20.1.1.4	80	Deny
4	TCP	10.2.2.0/24	20.2.2.10	80	Accept
5	TCP	10.2.3.0/24	20.2.2.10	80	Deny
6	IP	0.0.0.0/0	0.0.0.0/0	0-65535	Deny

2.2.3 Micro-segmentation

Lateral movement of malicious content in a network can be mitigated by adding a more granular level of security. To achieve this, the concept of micro-segmentation has evolved [18]. In a micro-segmented environment, a computer resource, such as a virtual machine, has its unique firewall inspecting the network-traffic to and from this unique computer resource. In such a way, east-west traffic is replaced by a direct connection between the virtual resources. This direct network traffic is inspected without adding any complexity to the perimeter firewall and consuming the network resources. An individual resource in the datacentre has a set of firewall rules, which is managed by the data-center system administrator that monitors the network traffic. The perimeter firewall can still be active inspecting north-south traffic, but the load and complexity of the policies enforced are drastically reduced. Figure 2.5 presents a simplified scheme of a micro-segmented DC. In this environment the firewall is placed between each VM and the vSwitch enforcing firewalling at a more granular level.

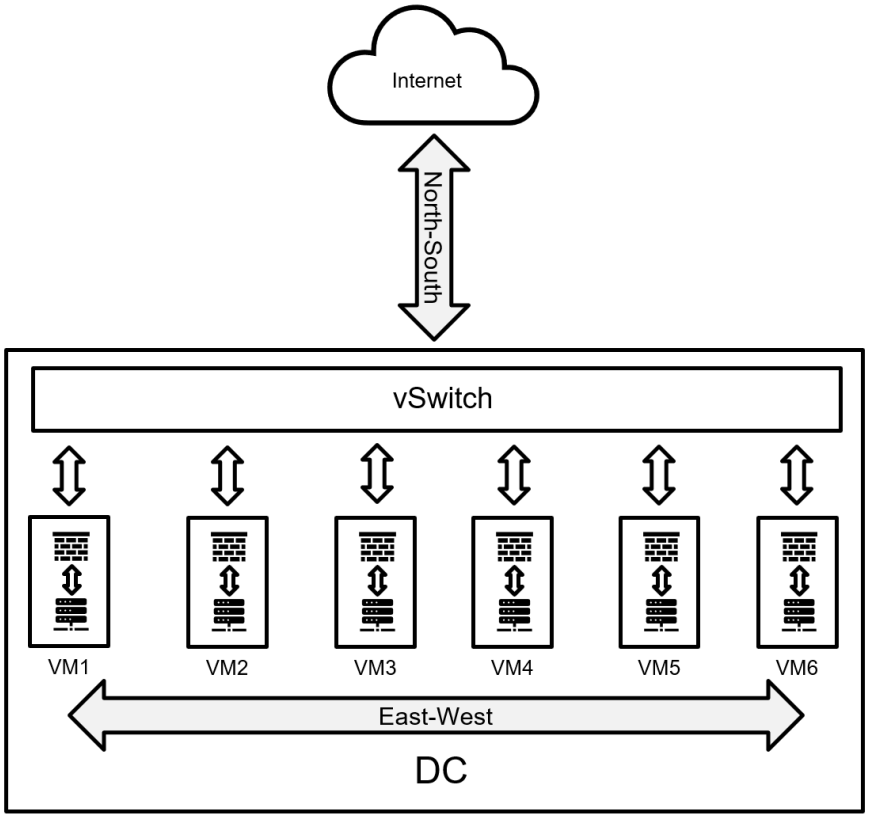


Figure 2.5: Micro-segmentation

As shown in Figure 2.4, traffic moving from one zone to another traverses the firewall. Such network traffic within the same datacentre consumes network re-

sources. If a firewall is on a business-critical system and the company policy demands that the firewall is configured in a High-Availability (HA) configuration, an increase in the network traffic could result in a costly re-investment in the firewalls. Within the same computer segments, east-west traffic is routed directly between the different resources within the datacentre. Chowdhary [24] defined micro-segmentation in the six different capabilities:

- *Distributed stateful firewall:* - Stateful firewalls are implemented to reduce the attack surface on network perimeters. This allows a stateful firewall to be configured on each application.

- *Topology agnostic segmentation* - Application based firewalls that are independent of the underlying network. By this, the firewall is agnostic to hardware vendor, which provides underlying network and supporting both layer 2 and layer 3 topologies.

- *Centralised ubiquitous policy control of distributed services* - Control access or integrate the system into management systems. Security policies can be programmed through Representational state transfer (REST) Application Programming Interface (API).

- *Granular unit-level controls implemented by high-end policy objects* - Grouping of mechanisms for creation of object-based policies. Objects in a micro-segmented system can use dynamic constructs such as operation system (OS), VM name and security tags. Specific static constructs like Active Directory (AD) groups, logical switches, VMs, IP and MAC sets can be used to distinct each application security perimeter.

- *Network based isolation* - Provide connectivity between different data-centres (DC) using secure communication to separate different instances. Such connectivity can be based on overlay networks (i.e VXLAN) or legacy network protocols such as VLAN.

- *Policy-driven unit-level service and traffic steering* - Third-party integration of network solutions for guest capabilities like antivirus, IDS/IPS etc.

2.2.4 Micro-segmentation policies

Firewall rules are often managed by a system administrator and can be tailor-made for each specific computer network. A result of this is that firewall rules need tuning in an ever-changing network topology. It also requires up-to-date administration allowing new services. If firewall rules are configured manually, the risk of human errors can lead to vulnerabilities that can be exploited by adversaries to compromise the network. When moving from a traditional computer

network with one or more perimeter firewalls to a micro-segmented network, the firewall rules must be ported to fit the new architecture. In a data-centre with multiple specialised VMs that provide different types of services, the number of established communication channels can be very high. Figure 2.6 shows an illustration of VMs in a data-centre and how the different VMs can be segmented based on the underlying network they are connected to. The network is segmented in a demilitarised zone (DMZ), application and database (DB). Each department (i.e Finance, HW and Engineering) is also segmented in each zone while making use of resources in different network segments. This creates relationship between different services and resources in the DC, which demands complex policies.

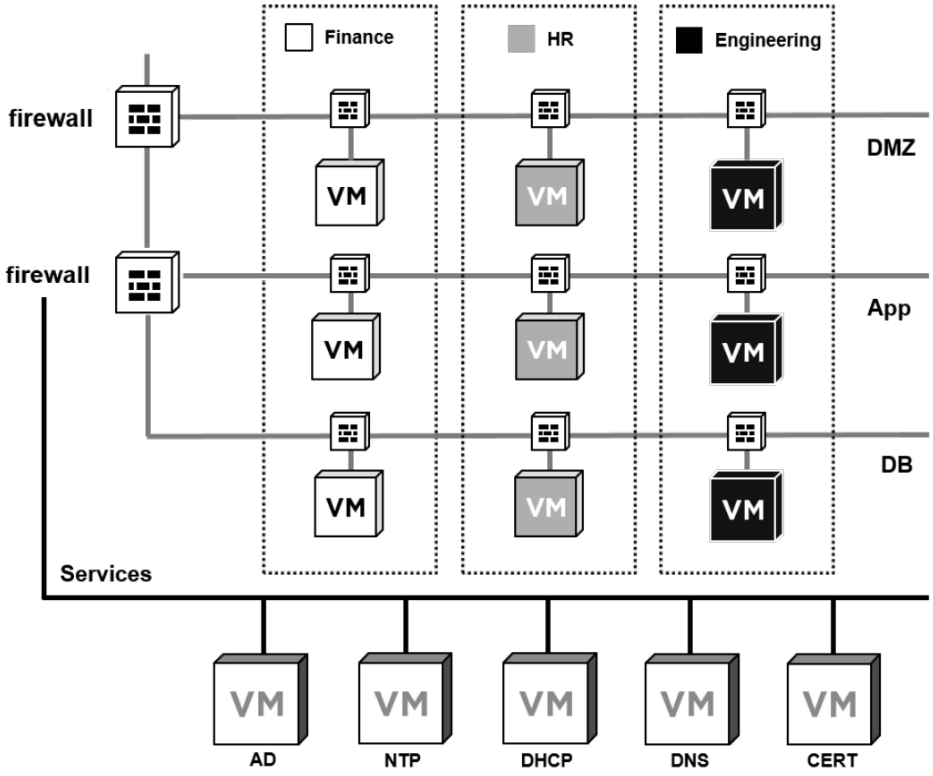


Figure 2.6: Example of various zones in a micro-segmented environment [25]

Sosa et. al [26] described how the use of security groups can be used in a VMware micro-segmented environment to provide policies. A security policy consists of Dynamic Inclusion, Static Inclusion and Static Exclusion. The Dynamic Inclusion can be computer OS name, computer name, VM name, security tag. The Static Inclusion/exclusion can be: VM, vNIC, Security Group, Security Tag, Cluster, Logical Switch, vAPP, IP Sets and MAC sets, Data centre, Resource Pool, Directory Group. Holmes [25] described how intelligent grouping can be defined by a user to security groups. According to Holmes, security groups can be split in the three

classes:

- Infrastructure: Assets from the infrastructure such as IP-address, OS (i.e. Windows, Linux etc.), or VM
- Environmental: Logical constructs such as different departments such as production, development etc.
- Application: Specialised applications such a special database or active directory etc.

As shown in Figure 2.7, a security group is the result of (Dynamic Inclusion + Static Inclusions) – Static Exclusion.

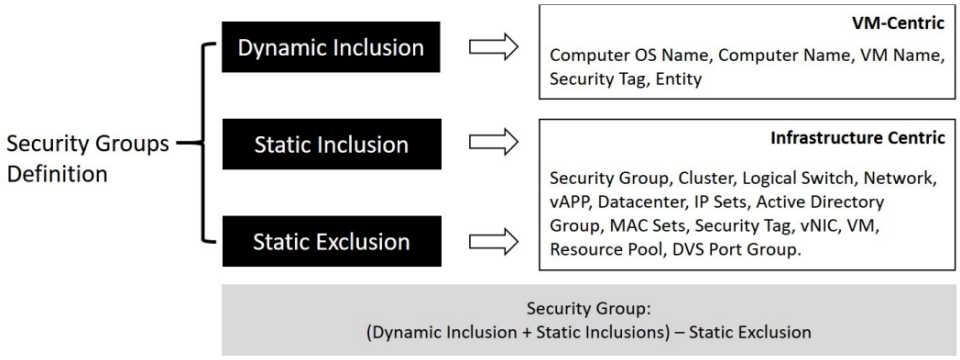


Figure 2.7: Security groups attributes [25]

By configuring security groups, these groups can be used to allow or block communication at a granular level. Figure 2.8 shows an example on how policies can be nested to create specific firewall rules that are programmed on the firewall in the hypervisor.

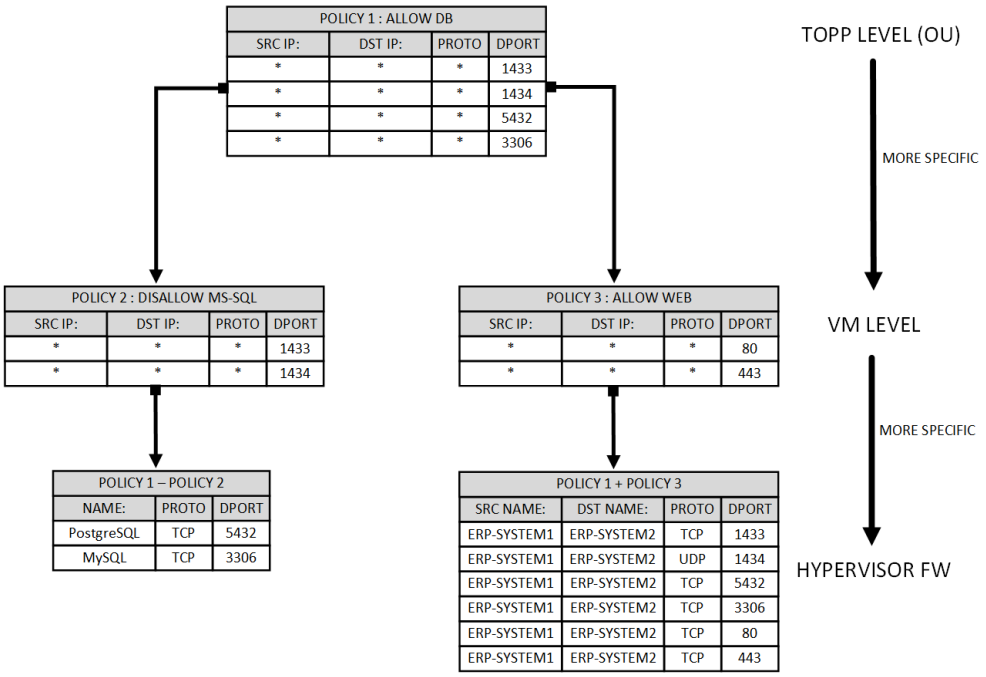


Figure 2.8: Example of how security groups can be nested to create specific firewall rules programmed in the hypervisor

In this example a policy is added at the Organisational Unit (OU) at the top level. Such OU could be a file structure representing different folders with VMs belonging to different departments. These top-level policies are then inherited by the VMs. In this process, more granular attributes specific to the characteristics that enforce this policy are being added. Such attributes could be VM-name, OS etc. After creating such policy groups, the specific firewall policies are programmed down to the firewall in the hypervisor for the specific VMs that are affected. In such a structure, there can be overlapping policies that contradict each other if several policies are applied to the same VM. In such a case, the use of a Security Policy Description Language (SPDL) could be applied to more easily administer such policies [27].

2.2.5 Moving to a micro-segmented environment

When moving from a traditional network architecture to a micro-segmented architecture, the goal is to implement the new design without disrupting existing traffic. To avoid this, a structured process on how to make this transmission is needed. In case of large and complex computer networks such a task can be very complicated and time-consuming if done manually. Holmes [28] defined that micro-segmentation could be divided into network, infrastructure and application. With application, the security is enforced to individual applications or functions and are agnostic to the physical level in the DC. Rules related to the infrastructure are

centric to the DC such as hypervisor clusters, vSwitches and port groups. Network grouping is like the more traditional approach of grouping network elements such as MAC-address, IP-address. Moving from one architecture to another can be a difficult task and there are some commercially available tools that are created to help organisations with such a migration. Tools like VMware v2T Migration Tool [29] and cITopus [30], analyse network traffic and visualise the results to the network administrator. By observing the output, the administrator can create end implement new policies and, in such a way, move to a micro-segmented environment.

2.2.6 Automate micro-segmentation

To migrate firewall policies from one environment to another can be a complex task. While there are several tools available to help network engineers perform such tasks, there are no tools available to automatically move from one firewall model to another. Firewalls are traditionally static policies configured on internal or external firewalls. Moving from such an environment to a micro-segmented environment would demand rewriting of these firewall policies when migrating to the new environment. It could be possible to interpret data and find patterns representing one environment and write programming code to transfer this to another environment. However, such an approach could be difficult because it could be a problem to write code rules for every pattern [31]. Another approach could be to apply artificial Intelligence (AI) to interpret the data and to find these patterns. AI is a great tool for analysing large amounts of data and find patterns and it can be applied on fluctuating environments adapting to new data [31].

2.3 Basic concepts of Artificial Intelligence

AI is a term used to describe the science of Artificial Intelligence [32]. Since the term was first used by John McCarthy at Dartmouth college conference in 1956 [33], the development of different approaches to AI has been researched extensively. Zang et. al (2021) made a study on the history of AI and how the technology has developed in several golden periods. In this study the authors describe how the development of more efficient computers and the Internet has made AI-techniques available to everyone [32]. The development of AI has led to several different methods in how AI is applied.
One such method is Deep Learning (DL) which is a sub-field of Machine Learning (ML) [34]. What differentiates these two techniques is that DL has similarities to biological processes in the brain to solve a specific problem [35]. In the following chapters these techniques are described in further detail.

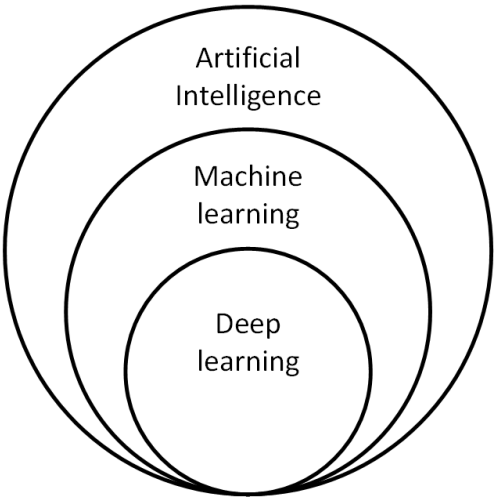


Figure 2.9: Artificial intelligence, machine learning and deep learning [31]

2.3.1 Machine Learning

Arthur Samuel (2000) defined ML as "*the field of study that gives computers the ability to learn without being explicitly programmed*" [36]. This approach to a problem can be very efficient compared to classical computer programming. In classical programming, a human being codes the rules, witch process data to get answers. This can be a complex and time-consuming task and raises the question if a computer can automatically do the same job. In ML, this is exactly what happens and it is a new programming paradigm where there is no coding by humans. Instead of human interaction, the ML-algorithm is trained with the answers and the data to give answer. The training is done by presenting many examples of an answer to the system which then finds the structure in the data [31].

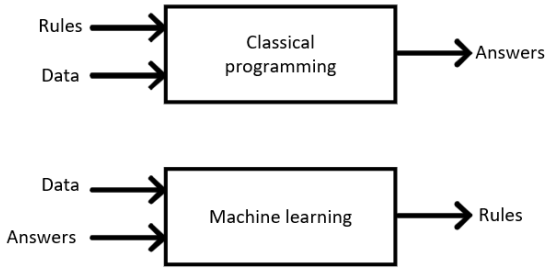


Figure 2.10: The new programming paradigm with ML [31]

2.3.2 Supervised and unsupervised learning

Zhou (2021), wrote that ML can be divided into two sub-fields; Supervised Learning, Unsupervised Learning [37]. These categories of ML have different approaches on how the ML algorithm operates, how data is fed into the system, and consequently how data is processed by the algorithm. In Supervised Learning, labelled data is fed into the ML algorithm. An example of this is the scenario when a computer learns how to differentiate pictures of cats from dogs [38]. In such a case, a person labels the different pictures with cats as cats, and the pictures of dogs as dogs. When the data is fed into the ML-algorithm the computer extracts different features from the pictures and uses the label to tag these features to a certain type of animal. Depending on the algorithm and the quality of the pictures, the computer learns what features are significant for a cat versus a dog within a certain degree of fault tolerance. When the computer is later presented with a picture of a cat it knows how to differentiate a cat from a dog. Géron (2019) stated that most of the ML application today is based on supervised learning although most of the available data is unlabeled [39]. Braiek et al. [40] explained that most ML algorithms require large amount of data and that it can be challenging to preprocess this data before it can be used for training an ML-model. Figure 2.11 shows how data and a problem must be merged into a training model which is evaluated. In this model, the author shows that a problem that we want to solve merges with processed data into a model. The model is then trained, and the performance measured.

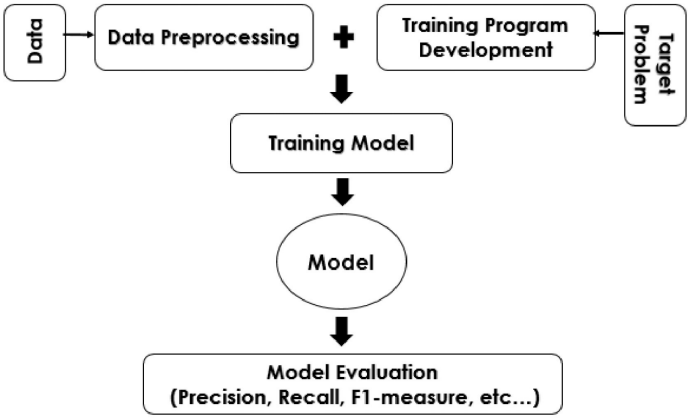


Figure 2.11: ML procedure [40]

Manual labelling can be applied on a dataset before it can be used in supervised ML. Sunhee et al. (2017) investigated how unsupervised learning could be used in combination with supervised learning to detect anomalies in network traffic [41]. The authors suggested a three-step process: (1) clustering of data and labelling the data, (2) use the labelled data to train a supervised model, (3) test the data with the model to detect if a data-point is an anomaly or not. With this method, the authors obtained an accuracy of 88%.

2.3.3 Deep Learning

ML is a mathematical framework implemented on computers that learnins from data to perform a specific task. In the field of ML, several techniques with different characteristics are implemented. Depending on the task that the system is to solve, the type of technique is chosen. One such technique is called a *neural network* or Artificial Neural Network (ANN) [42]. A neural network consists of multiple nodes, called neurons, which interact with each other using one hidden layer. However, in DL the number of hidden layers is greater than one, which constructs a much more complex neural network. The word *deep* in DL describes that the neural network consists of multiple hidden layers [31]. The difference between these types of neural network is shown in Figure 2.12 and 2.13. The number of input nodes in the input layer represents the number of features in the dataset and the number of nodes in the output layer represents the number of labels in the dataset. The operator of the system must find enough nodes in the hidden layer that give the best output performance. Each neuron in the network consists of a weighted input and an activation function that produces an output.

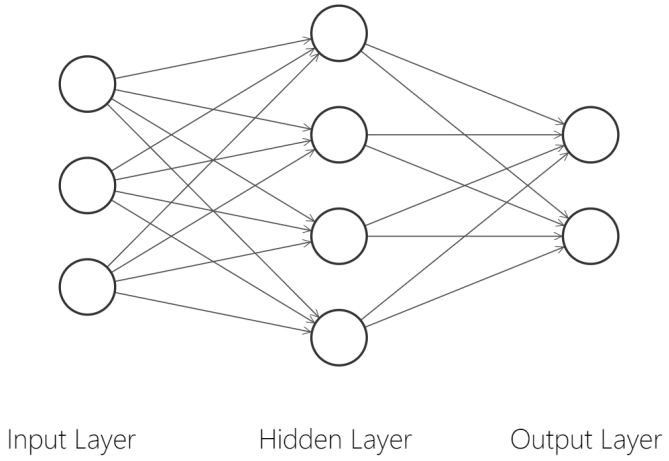


Figure 2.12: Neural network

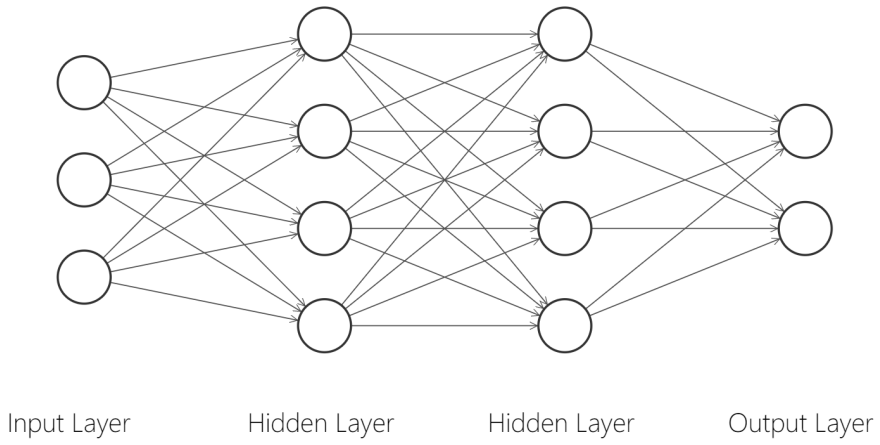


Figure 2.13: Deep Neural network

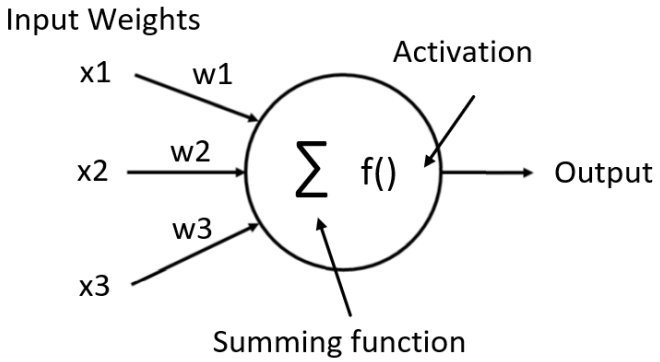


Figure 2.14: Description of a neuron in a neural network [43]

$$Z = \sum_{i=0}^{n} (W_i X_i) \tag{2.1}$$

As shown in Figure 2.14 and in equation (2.1), a neuron's output is the weighted sum of all the inputs (x), which is passed through an activation function f. The activation is a mathematical expression, which activates the output if the signal input is within the threshold of the selected function. The most common functions are the linear, binary step, piecewise linear, sigmoid, Gaussian, and hyperbolic tangent functions [44]. When constructing a neural network, the operator must select one or more activation functions and combine these in the network to solve the given task. The operator also needs to tune the different parameters in the algorithm of the neural network in such a way that the network is as simple as possible while it still performs as intended. This type of tuning of the algorithm is

often called hyper-parameter tuning and is a trade-off between computing power and the quality of the result [37]. When tuning the neural network, a phenomenon called *over-fitting* can occur. Over-fitting occurs when the model is biased and can reduce the system performance. When a neural network is constructed, data is loaded into the algorithm to train the system. The presented data is split into a training set, a test set, and a validation set. In a data set, each individual piece of data is called a sample. When training a model these samples are combined in blocks of data called a batch [31] and for each batch the model makes a prediction. When repeating this process several times with enough data, the network will improve its performance and if constructed correctly, predict the correct value with an increasing level of accuracy. One repetition of the data set is called an *Epoch* [31], and for each epoch the model will learn and try to improve the prediction. The validation dataset is used for hyper-parameter tuning to avoid over-fitting.

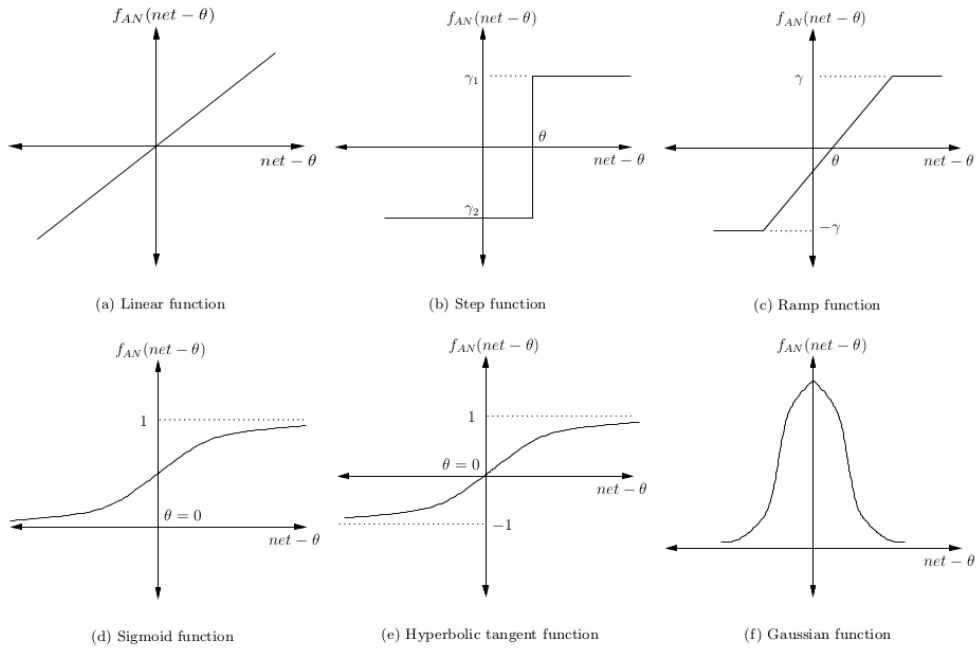


Figure 2.15: Common activation functions used in neural networks [45]

2.3.4 Performance parameters

A confusion matrix is a description of the performance of a classification algorithm [46]. Table 2.16 is an example of a confusion matrix for a binary classifier.

		Predicted	
	Classes	Positive (1)	Negative (0)
Actual	Positive (1)	TP	FN
	Negative (0)	FP	TN

Figure 2.16: Example of a confusion matrix

- True positives (TP): YES predicted as YES
- True negatives (TN): NO predicted as NO
- False positives (FP): Type I error - predicted YES, but is actually NO.
- False negatives (FN): Type II error - predicted NO, but is actually YES.

In machine learning involving more than two classes, the term *multi-class classification* have been used. Figure 2.17, shows an example of a confusion matrix involving ten different classes. In this this matrix the different classes are listed horizontally and vertically in the same order. Horizontally are the actual classes while the classes vertically are the predicted classes. Following the intersection between horizontally actual and the vertically predicted values are values that are true positive values. These values are actual classes that are predicted correctly. Values that are predicted as one class but are not actually that class are false positives. Subsequently, Values that are one class but predicted as another class are false negatives. Every other predicted value that are not actual or predicted is a true negative.

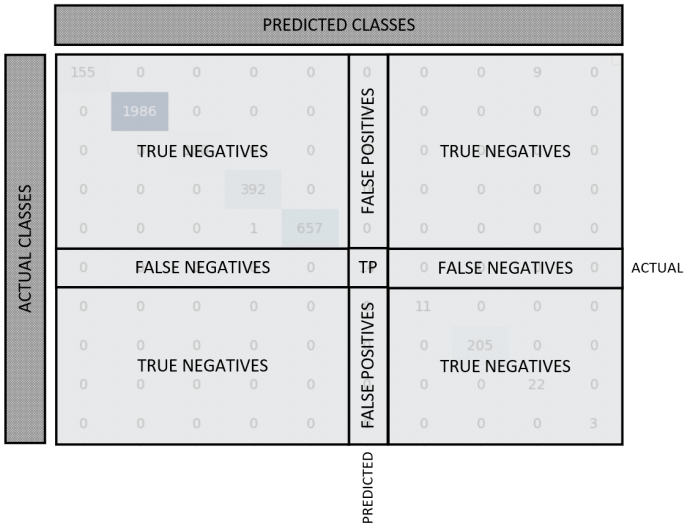


Figure 2.17: Confusion Matrix example for a single class in multi-classification [45]

The precision value is the calculation of positive values that actually are positive.

A high precision value indicates that we can trust that the model classifies the data correctly.

$$Precision(P) = \frac{TP}{TP + FP} \tag{2.2}$$

Recall value is the number of positive predictions divided by the number of positive class values in the test data. This value is also called the true positive rate (TPR) or sensitivity.

$$Recall(R) = \frac{TP}{TP + FN} \tag{2.3}$$

The F/F1-score is the balance between the precision and the recall. The formulas for calculation the different scores is shown in formulas (2.2)-((2.4)

$$F1 - score = 2 \cdot \frac{P}{P + R} \tag{2.4}$$

The accuracy is the calculation of the sum of all TP and TN

$$Accuracy(A) = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.5}$$

The performance parameters in this thesis are displayed in a classification report. In the classification report, the performance of each individual class is displayed together with an average score for all the classes together. Figure 2.18, shows an example of a classification report of a classifier with three classes.

Score per-class

	precision	recall	f1-score	support
0	1.00	1.00	1.00	3228
1	0.99	0.94	0.97	148
2	0.99	1.00	1.00	180
accuracy			1.00	3556
macro avg	0.99	0.98	0.99	3556
weighted avg	1.00	1.00	1.00	3556

Avrage scores

Figure 2.18: Classification report example [47]

The support column is the number of datapoints that has been tested. In this example, the data is unbalanced, which means that one class has many more

samples that the two other classes. The overall performance is calculated in accuracy, macro, and weighted average. The weighted average score is calculated while taken into account the support of each class. The macro average is the the mean of the measurements when the support per class is not taken into account. Another metric is micro average, which is a global average score calculated by the sum of the TP, FN and FP. In most cases micro average and accuracy is the same but, in those cases, where they deviate, the micro average can be displayed. Another way to display the performance on different classes is a Receiver Operating Characteristics (ROC) curve [31]. In such a curve the model performance is checked by calculating Area Under Curve (AUC). The AUC is the ratio between TPR and FPR. A high ROC curve means that the model has a high degree of separation between TP and FP.

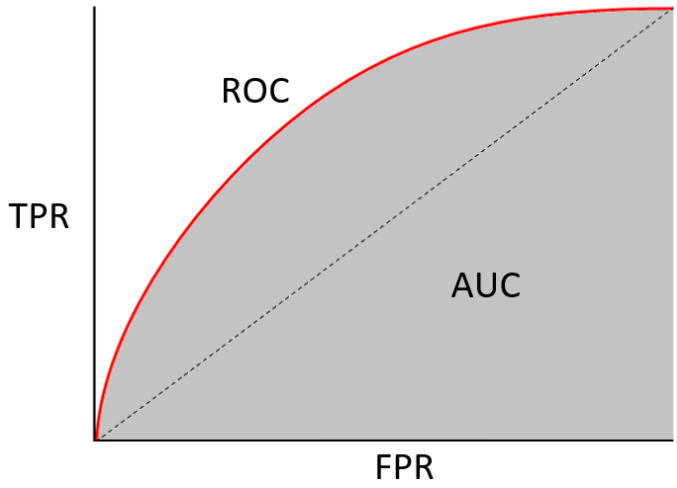


Figure 2.19: ROC curve example

2.4 Applying ML to analyse firewall configurations

Much research has been done to discover security flaws in existing firewall configurations by applying ML to analyse the policies. In traditional network architecture, different types of firewalls are being applied to control traffic between zones in computer networks. Anomalies in the configuration of such firewalls could lead to security gaps. By applying supervised ML to analyse firewall logs it can be possible to detect anomalies [48]. However, using ML to abstract firewall policies and move these policies to another firewall model has not been investigated.

Chapter 3

Theoretical contribution

There are several considerations related to firewall in cloud environments. Cloud services can be provided to customers in different ways depending on the infrastructure and the type of services that are provided. Different cloud environments have different approaches to security and performance. In this thesis, the difference between firewall models is outlined. By this, it is possible to recommend which firewall model is to be used in different virtualised environments. One of the recommended models is kernel-based micro-segmentation. This type of firewall model enforces firewall on a granular level by segmenting both the network, virtual instances, and applications. It is very secure and efficient, but it can be problematic for an organisation to migrate from one firewall model to another. When migrating from one virtual environment to another without compromising security can be a problem. Firewall policies in one environment are not necessarily compatible with the previous model. Migration from one model to another can be an entirely manual process. However, this is not recommended due to the risk of wrong configurations because of human error. In this thesis, we investigate if data from one infrastructure can be abstracted into new policies adapted to the new environment. A hypothesis is that it is possible to achieve this by applying machine learning (ML). The ML in use is a branch of deep neural networks called a Feed Forward Neural Network (FFN). A type of FNN called a Multilayer Perceptron (MLP), are used to test this hypothesis. Policy rules can be nested with general policies at a top level and more specific firewall inherited from these policies. By classifying the top level policies, more specific firewall policies can be derived from these policies. MLPs are good classifiers and have previously been applied for detection cyberattacks in network traffic. These characteristics are the basis for choosing which DNN model is to be applied in this thesis. The proposed neural network is shown in Figure 3.1, consist of two hidden layers of neurons in pyramid model where the number of neurons is reduced for each layer.

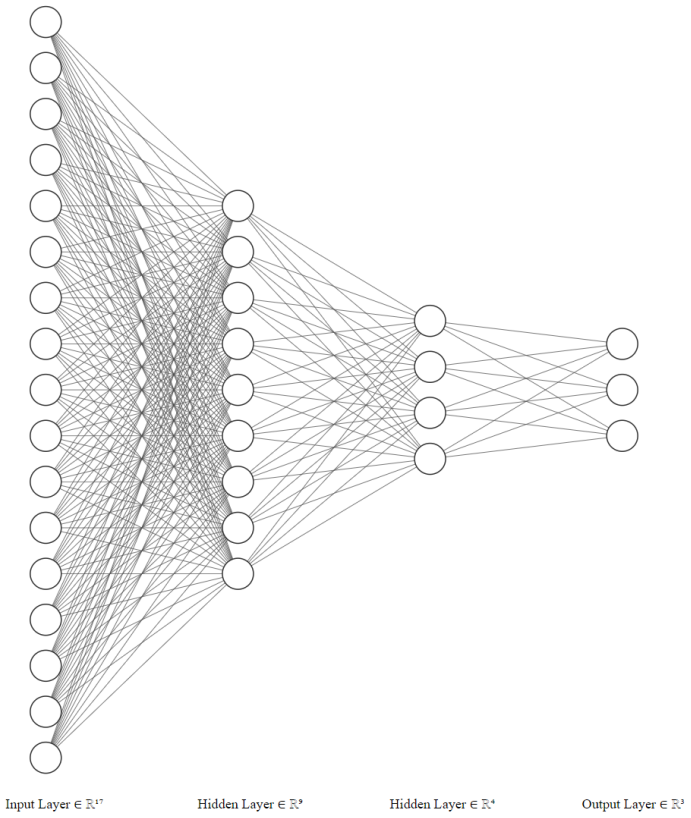


Figure 3.1: Structure of the proposed deep neural network used in this thesis

By labelling the dataset depending on different features and groups in the data set, the output is observed, and the performance evaluated. The performance depends on how accurate the DNN can predict different groups of labelled data. The proposed DNN was found to have a very high degree of overall accuracy for multi-class data set. However, the accuracy decreased for data sets with a limited number of data points. Especially smaller groups of data had a low accuracy resulting in false calculations.

When migrating from one cloud firewall model to another and at the same time maintain security, a structured process is needed to create new policies and copy these policies to the new infrastructure. In this thesis a model for such a process is discussed. In chapter 4, a structured process to abstract firewall policies is proposed. A hypothesis is that such a process can be used in a model to abstract and transfer firewall policies between different cloud environments. In Chapter 6, such a model is discussed and presented.

3.1 The proposed process

When migrating from a perimeter firewall to a micro-segmented environment, a new architecture is used demanding a new set of firewall policies. This thesis proposes a structured process on how to structure data gathered from the old network topology and use these data to abstract new policies that can be used in the new virtual environment. The proposed process consists of seven stages as described in Figure 3.2. With the proposed process, a structured approach to the problem is addressed using machine learning to abstract data into new firewall policies. The data is a sum of different data collected from various sources in a live production environment and analysed by the proposed ML method. The goal of this process is to produce new policies based on a subset of the total amount of data and train a neural network based on these policies. Based on the trained model an analysis can be used on the total dataset to predict new policies for all the VMs in the DC.

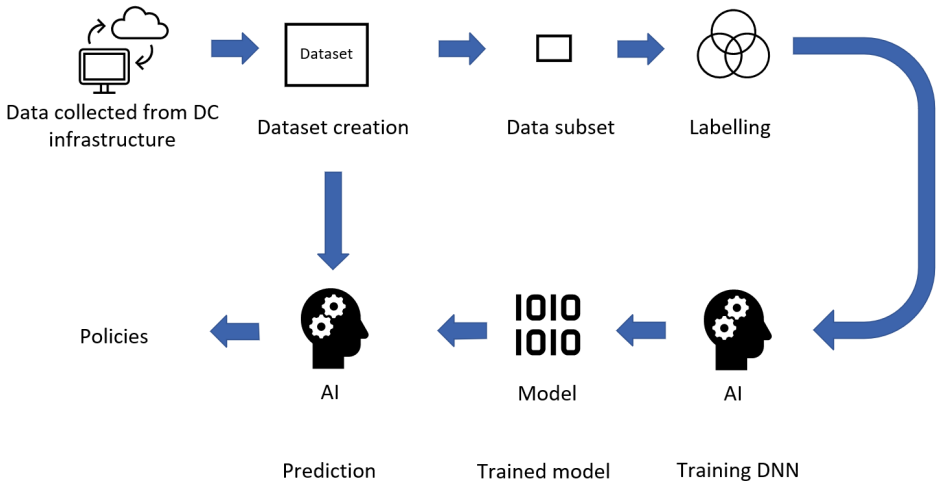


Figure 3.2: Proposed process

1. Data collection: Data from different systems are collected
2. Data set creation: Collected data are merged into one data set
3. Data subset: Subset of the data set is created
4. Labelling: The subset is labelled according to the new policies
5. Neural network: DNNs are trained with the subset
6. Model: The trained model of the DNN is stored
7. Prediction: The model from the DNN can be used a a classifier to predict new firewall policies

In this thesis, data delivered from the infrastructure at Eidsiva are provided as text files. The data were collected from various sources within a live production environment at Eidsiva. This live environment consists of VMs hosted on a hypervisor

and devices which produce different services and a computer network providing connectivity. A perimeter firewall segments different networks into different network zones and enforces network policies between these zones. The configuration from the firewall is saved in a separate file. A prerequisite is that the current firewall policies are enforced as intended allowing network connectivity. Network communication is then collected by applying distinctive features in the infrastructure. All the network traffic traversing north-south, east-west in the data centre has been captured and stored. This data set is a representation of the current firewall policies enforced in the network. Since the amount of network data is large, this dataset is reduced by only gathering data related to VMs in the data centre (i.e east-west traffic) and by only using data from one day. From the hypervisor, attributes related to each virtual machine such as name and operating system are extracted and stored. In total, three sources of information are gathered and used in further analysis. The data collected in the previous stage are stored in separate databases and Structured Query Language (SQL) is used to combine the different databases containing information using the IP-address as index. The size of the dataset is reduced by adding attributes to SQL to only use network traffic that is addressed with known protocols. A subset of the data set is used to train a neural network. This subset is labelled depending on how the new policies can be applied in a new environment. Gan et. al [49] wrote about the difficulty of labelling data. The author claims that labelled data is difficult to obtain and often demands extensive expert knowledge to process. In order to overcome this problem and label the data as accurately as possible, network engineers from Eidsiva have labelled the data sets. When labelled data is obtained, the neural network is constructed and trained. This model can later be used for further analysis on the whole data set predicting policies.

Chapter 4

Research methodology

In this chapter, a research framework for empirical research is described. A part of this framework is literature reviews on related work. Such related work is outlined in the last sections of this chapter.

4.1 Research framework

Oates [50] developed a framework of empirical research. Figure 4.1 shows a graphical representation of this process describing how a researcher can follow different paths in the research process. The framework describes the process of developing research questions, selecting different strategies to generate data, before a quantitative or qualitative analysis of the data is performed.

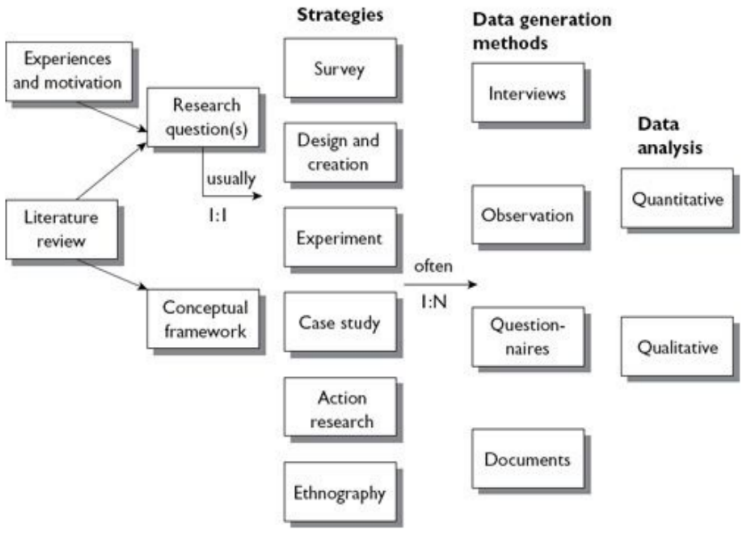


Figure 4.1: Framework of empirical research [50]

In this thesis, this model for a structured research design is followed.

4.1.1 Research strategies

As described in Chapter 1, the motivation for this thesis is to make it easier for organisations and companies to migrate firewall policies between different cloud environments. From this motivation, three research questions are derived.

This section describes the strategies that are used in this thesis to answer the research questions. This thesis uses a qualitative research method to analyse data. An important part of any research project is to perform a *literature review* on previous related work. This is a qualitative approach where information is gathered and studied. By reviewing relevant information and learning the existing knowledge regarding the topic, the research questions can be answered. The knowledge from this review is used to design a series of experiments to test a hypothesis.

When research questions and background theory are studied, a strategy on how to answer the research questions or test different hypothesis can be chosen. In this thesis, the following strategies are used:

- **Design and creation**: An important research in this thesis are new *artefacts*. Oates [50], stated that such artefacts could include models, methods or working demonstrations (i.e. ICT-systems) of such models. In this thesis, a structured process on how to process data to migrate between cloud environments is described. A working proof of concept of some of the elements in such a process is assessed in a series of experiments.
- **Experiments**: After data collection is done, the planned method to analyse the data is to use AI. AI is a term used where computers mimic human behaviour and can be divided into two subsets: Machine-Learning and Deep-Learning [51]. ML is an AI technique that let the computer learn e.g., patterns without being explicitly programmed. Deep learning (DL) is a subset of machine learning (ML), which uses the multi-layer neural networks to analyse distinct factors [34].

The output of the experiments generates different performance parameters that are observed. Based on these observations, the results can be discussed.

4.2 Related work

A goal for this thesis is to apply ML to solve a problem regarding firewall policies. To solve this problem, previous work has been investigated to get inspiration on how to complete such a task. In this section, related work regarding firewall models is being outlined. In addition to this, previous work regarding on how different ML techniques have been applied to analyse text data to solve a problem is also described.

4.2.1 Firewall models in cloud environments

Jekese et al. [52] defined three virtual firewall models to be used in cloud environment. These three models, a, b and c are shown in Figure 4.2. In model a, the firewall is located in front of the virtual switch. This model is less secure since traffic between Virtual Machines (VM) is not inspected. Model b positions the firewall in front of each VM and monitors traffic flowing between the different VMs, and from the VMs to the Internet. Model b can be implemented as an appliance running as a VM on the hypervisor or a kernel process running within the host of the hypervisor. In model c, traffic between VMs and the Internet is monitored as model b but has only one shared firewall for all VMs.

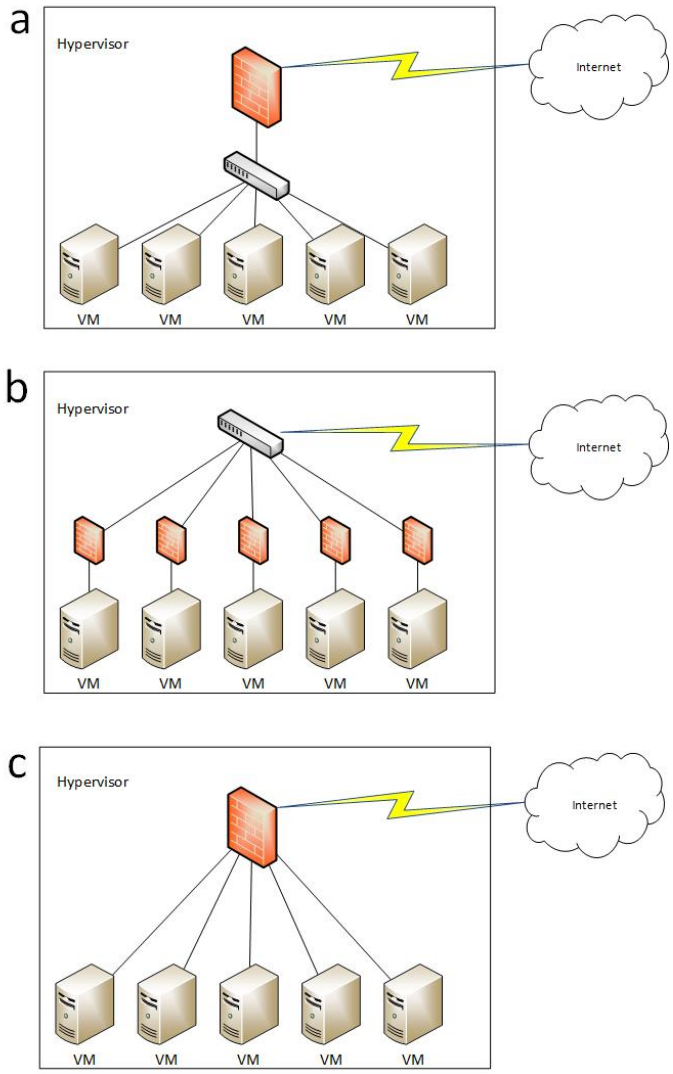


Figure 4.2: Firewall models in cloud environment [52]

NIST Special Publication 800-125B, Secure Virtual Network Configuration for VM Protection defines three classes of firewalls. These classes of firewalls are: Physical Firewalls, Subnet-Level Virtual Firewalls and Kernel-Based Virtual Firewalls [53].

- Physical Firewalls - are dedicated hardware or software firewalls which run their own independent operating system and do not share any resources with any other application.

- Subnet-Level Virtual Firewall - VM with several Virtual Network Interface Cards (vNIC).

- Kernel-Based Virtual Firewalls - installed inside the hypervisor kernel, typically between the vNIC and the VM.

Sections 4.1 to 4.3 in the publication [53], outline the advantages and disadvantages of the different described firewall models. The advantage of the physical firewall is that it is a mature technology that can have sophisticated policy rules, and if installed, other capabilities. The disadvantage of a physical firewall is that in combination with a virtualised architecture, traffic from one VM to another must traverse the firewall, which can lead to congestion of traffic. Another disadvantage with a physical firewall is that it is not potentially integrated with the management system of the virtualised environment and would lead to less automated procedures when applying firewall policies. Virtual firewalls are entirely software based systems that share resources with other virtual instances on the platform where they run. Subnet-Level Virtual Firewalls are special virtual appliances that can be placed on strategic locations within a virtual environment to switch traffic directly between VMs and in such a way minimise network traffic traversing the perimeter firewall. The downside of this firewall model is that in such deployment it only has available the number of resources that it was assigned when configured, which could affect the performance of the firewall. Kernel-based virtual firewalls offer much higher performance compared to subnet-level firewalls because they are tightly integrated with the kernel of the hypervisor. The kernel controls the available hardware resources to the firewall compared to subnet-level firewall where the hardware resources are limited to what is assigned when implemented. Because the firewall is controlled by the kernel, and placed between the VM and the vNIC, the firewall cannot be affected in case VM is infected with malicious content. Based on this analysis NIST recommends four deployment architectures for firewalls in the following sequence [53]:

- Recommendation 1 (VM-FW-R1): Virtual firewalls should be deployed for environments with delay-sensitive VMs in order to route traffic directly between VMs instead of routing traffic outside the virtualised host environment.

- Recommendation 2 (VM-FW-R2): Kernel-based virtual firewalls should be deployed in VMs with input-output (I/O) sensitive applications since packet-

processing in kernel-based virtual firewall processes the packets at native hardware speed in the kernel of the hypervisor.

- Recommendation 3 (VM-FW-R3): In order to achieve easier provisioning of uniform firewall rules, it is recommended that both subnet-level and kernel-based virtualisation has an integrated management platform. Such an integration will enable an effortless way to deploy uniform firewall policies to multiple locations and reduce the risk of human errors due to configuration errors.

- Recommendation 4 (VM-FW-R4): With the use of management platforms, it is recommended that both subnet-level and kernel-based firewall has the ability to abstract rules at a higher level (i.e., security groups) in addition to traditional network attributes like IP addresses, source/destination port and protocol.

Ankur et al. [2], described an SDN-based Stateful Distributed Firewall (SDFW). SDN is a technology that decouples the control of logical networks from the hardware. The authors present the SDFW, which is a distributed firewall designed to prevent lateral movement between hosts in data-centers. The research was based on P4 [54], which is a programming language that allows protocol independent packet processing and stateful packet inspection.

Li et al. [55], investigated security in cloud environments, and proposed a distributed stateful firewall scheme. Firewall as a service (FWaaS), is a component in cloud environments that requires automation of the firewall rules. Current firewalls in cloud environments are static and not flexible. The proposed scheme analyses connection state information in the data plane. This is done by designing a finite state machine and a state table.

4.2.2 Deep Learning Approach for Intelligent Intrusion Detection System (IDS)

In this thesis, ML is applied to analyse and classify data. Such classification has similarities with previous work to analyse network traffic. Vinayakumar et. al [56], investigated how to use machine learning to detect and classify cyberattacks. In this study, the authors investigate the performance of different DNN architectures when predicting cyberattacks on publicly available benchmark IDS dataset. The applied neural network was a type of Feed Forward Network (FFN) called Multi-Layer Perceptron (MLP) that consists of three or more layers of neurons. Five different types of DNN were tested for both binary and multi-class classification against different publicly available datasets of malicious traffic in order to find the neural network with the best performance. The datasets were labelled in different classes depending on the malicious traffic. The authors observed that the proposed

networks learned the categories for the input data after less than 400 epochs. Another observation was that the models were over-fitting when there was no hidden layer with dropout. For the multi-class classification, performance metrics were calculated for the different models of the DNN and for each of the different classes in the dataset. The proposed DNN model outperformed the performance of more classical machine learning classifiers. The authors propose the Scale-Hybrid-IDS-AlertNet (SHIA) framework which is a distributed monitoring and reporting system which implements the proposed DNN. The SHIA framework consists of a processing module and a DNN module. The processing module monitors both computers and network traffic, collects data and stores this data in a database. The DNN module analyses the database with the proposed DNN and passes the results to the network administrator.

Chapter 5

Experimental work and discussion

In Chapter 4, we propose a structured process to gather, process and analyse different data sources in order to create policies for a micro-segmented environment. In this chapter, a series of experiments is conducted to investigate if such a theory is correct by applying machine learning to classify data. Figure 5.2 displays the different data-sources and how a combined data set is fed into a DNN to classify data. The output of the DNN are predictions of firewall policies derived from the data sources. This chapter explains how data is processed and stored in three difference databases. One database stores data from the virtual machines, one stores data from network traffic in the DC and one stores data from the firewall. All these databases are joined into a dataset using Structured Query Language (SQL) [57]. Subsets of the processed data derived from the initial dataset are used in different experiments containing a neural network. The results from these experiments are presented and discussed.

5.1 Setup

5.1.1 Hardware and software setup

In AI, several different approaches can be used to analyse data which is implemented in different programming languages and frameworks. One such framework is Keras [58], which can be installed in a *python* environment. Keras is a DL framework that includes several DL libraries and has previously been used in systems such as recognising patterns in images, e-mails etc [31]. Different libraries are used in the experiments. Table 5.1 lists the version of the OS and the functionality of the installed software packages used in these experiments.

Table 5.1: Installed software

System/package	Version	Functionality
Ubuntu	20.04 LTS	Operating system
Python	3.8.10	Programming language
Pandas	1.3.2	Data structures and data analysis tool
numpy	1.19.5	Mathematical functions
scikit-learn	0.24.2	ML and data anlysis
sqlite3	3.3.1	Small SQL database engine
seaborn	0.10.0	statistical data visualization
matplotlib	3.4.2	Visualization
Keras	2.6.0	ML framework
Tensorflow	2.6.0	ML platform

In this experiment, the Keras framework is used to analyse text data. As shown in Figure 5.1 Keras is installed on top of Tensorflow, whitch is a open-source library for ML controlling the underlying hardware [31]. Tensorflow is compatible with different types of hardware like Tensor processing unit (TPU), Central Processing Unit (CPU) or Graphics Processing Unit (GPU). The training of the neural networks is done on a laptop with an HP Elitebook G2 laptop with an Intel i7-4600U CPU @ 2.10GHz and 8GiB system memory.

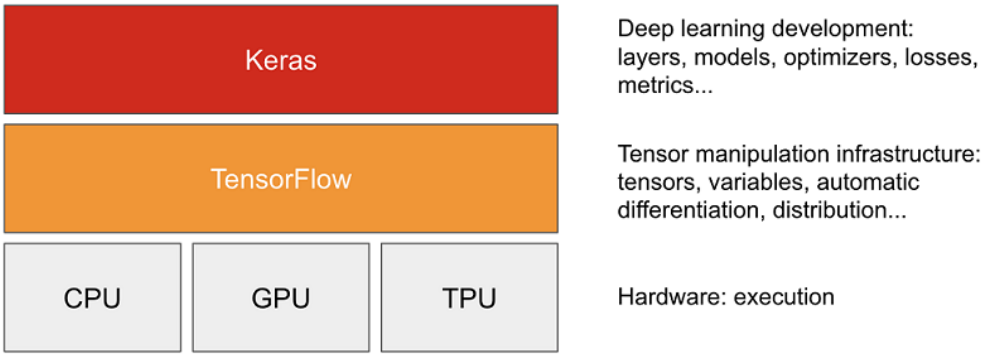


Figure 5.1: Keras and TensorFlow [31]

5.1.2 Data gathering and processing

The data used to conduct the experiments in this thesis where collected by network engineers in Eidsiva. These data sources represent three sources in Eidsiva infrastructure. The data is then merged into a dataset that is processed by the neural network. In this thesis, three sources of data are used to create a dataset. These three sources are information about the VMs, network traffic and information from the firewall. Figure 5.2 shows a simplified process of how data is gathered and processed. The first step in the process is to gather data from the three sources. In this experiment, this is the data gathered by Eidsiva. Data

from the different sources need individual pre-processing in order to be able to join these sources together into one superset. After the individual pre-preocessing, the data is stored in separate databases. These individual processes are explained later in this chapter. The three sources are data from VMs, network traffic and the firewall. When processed, these databases are joined together using SQL into one superset.

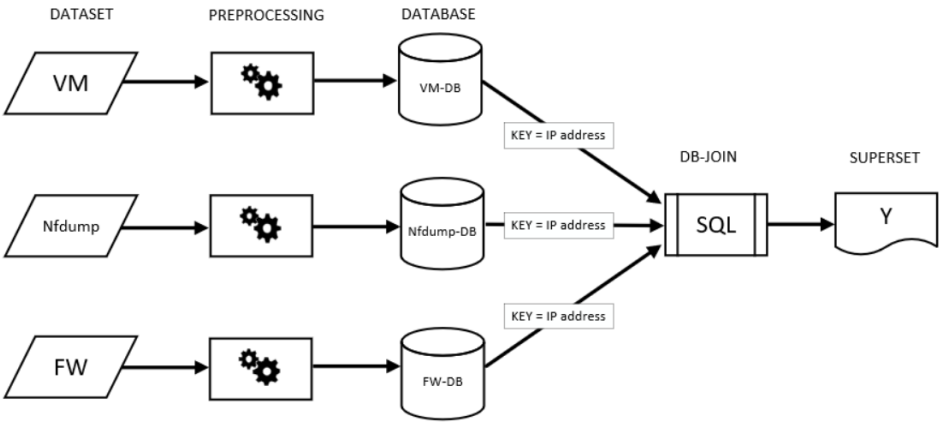


Figure 5.2: Data joining into superset Y

Data from Virtual machines

To conduct the experiments, three datasets are received from Eidsiva. These datasets are infrastructure data regarding VMs operating on the infrastructure, files with network traffic, and data with information from the firewall. Table 5.2, show the different types of data extracted from the virtual environment.

Table 5.2: Extensions from VMs

Type:	**Example:**
Source ip	10.9.2.112
VM-Name	Test
Network name	Labnet
Vlan	100
OS	Ubuntu Linux (64-bit)

Data from network traffic

The DC in Eidsiva is a virtual environment using Vmware vSphere as hypervisor. The virtualised network switch in vSphere is called vSwitch, which can mirror network traffic out of the DC. By mirroring network traffic traversing the vSwitch to a remote host, the network traffic can be captured. Netflow is a data export format, developed by Cisco Systems, to capture network data [59]. In vSphere,

a netflow collector can be configured to collect netflow data from the vSwitch to an external data collector. The network traffic was then captured by Eidsiva using a program called nfcapd [60]. These files where then copied to be used in the experiments.

Nfdump [61], is a tool developed to read and analyse files stored by nfcapd. To limit the data set, only data for one day is extracted and stored for further use. The captured network traffic is filtered on IP-addresses that are located in the DC. Every other network component or device that communicates with a VM but are not located in the DC are excluded. In such a way only east-west traffic is used in the experiments. Nfdump can filter IP-addresses and input files and aggregate the data. By using these features a dataset for one day consisting of aggregated traffic from one destination to another can be created. Code example 5.1 shows the syntax of nfdump. With expression *-R*, a sequence of files is read as a input to the program. The file format is *nfcapd.YYYYMMDDZZZZ*, where the first eight numbers is the date and the last five is a sequence number. By listing all the files stored by nfdump for one day, the first and the second file is put before and after colon in the syntax. The expression *-r* aggregates netflow data at connection level. The next fields limit the source to known source (src) IPv4 addresses in the DC. The IP addresses of VMs in the DC are listed in the brackets. The last expression *-o*, sets the format on the output file to comma-separated values (CSV). The number of output lines depends on how much traffic was generated in the selected period. In this case, data from 25th January was used as input, which generated almost 3.3 million lines.

Code listing 5.1: nfdump example

```
nfdump -R nfcapd.YYYYMMDDZZZZ:nfcapd.YYYYMMDDZZZZ -a 'src ip in
[ xxx.xxx.xxx.xxx yyy.yyy.yyy.yyy zzz.zzz.zzz.zzz ]' -o csv >> FileName.csv
```

Data from firewall

As explained in Chapter 2, a firewall is used to separate the computer networks between trusted and untrusted sections. Eidsiva firewall is connected to the data-centre and is used to separate different network-zones in the DC. Each network zone consists of different vlans and the firewall allows or denies traffic depending on the current firewall policies that are applied to the zone. Data regarding the different zones in the firewall and the subnet that is on the different zones is derived from the firewall configuration and stored as a csv file.

5.1.3 Creating the superset

SQL is used to merge the different datasets. As shown in Figure 5.2, data from various sources is merged together into one data set and stored as a single file that can be analysed by the neural network. In all databases the IP-address of the VMs in the DC is used as key to combine the data sources. Nfcapd is constructed to

provide information of captured network environments, which is not applicable when used in a virtual environment. Because of this, some fields only contain default information and can be filtered out. Table 5.3 lists the output from the nfdump command and which of the tables that are in the dataset.

Table 5.3: Nfdump extensions

Abbreviation	**Extension:**	**Format:**	**Example:**	**Used in dataset:**
ts	Start Time - first seen	date and time	2022-01-25 02:23:32	No
te	End Time - last seen	date and time	2022-01-25 17:46:07	No
td	Duration	seconds	55355.000	No
sa	Source Address	ip-address	10.9.2.112	Yes
da	Destination Address	ip-address	10.9.2.120	Yes
sp	Source Port	number	3306	Yes
dp	Destination Port	number	46795	Yes
pr	Protocol	text	TCP	Yes
flg	TCP Flags	text	...AP.SF	Yes
fwd	Forwarding Status	number	0	Yes
stos	Src Tos	number	8	Yes
ipkt	Input Packets	number	25	Yes
ibyt	Input Bytes	number	2416	Yes

The tables that are not in use are tables not considered to contain relevant information. Since a neural network only accepts numbers, all strings and IP-adresses must be converted to numeric values. For string values a function extracts all unique strings, generates a unique number for each string, and replaces the strings with the generated number. All IP-addresses are converted to binary form using the function INET_ATON() [62].

An initial data set, called Y, is created by joining the three data sources containing network communication joined with information from the VM and Firewall dataset. As shown in Table 5.5, dataset Y is a superset to subsets A-H. Subset A is created by deleting the source port, filtering specific destination port lower that 1024, and summarising the nfdump input packets (ipkt) and input bytes (ibyt). This creates one unique line in the dataset for each set of communication between two VMs in the DC and reduces datset Y to 1719 unique rows. Dataset B is created by selecting a subset of 10% random lines of communication from Y. Table 5.4 list the different columns in the superset Y.

Table 5.4: Superset Y columns

Number	Name	Comment
1	src_ip_aton	Binary representation of source-VM ip address (INET_ATON)
2	src_vm_id	Source-VM id number
3	src_vlan	Source-VM vlan number
4	src_os_id	Source-VM OS number
5	src_os_type	Source-VM OS windows or Unix based
6	dst_ip_aton	Binary representation of destination-VM ip address (INET_ATON)
7	dst_vm_id	Destination-VM id number
8	dst_vlan	Destination-VM vlan number
9	dst_os_id	Destination-VM OS number
10	dst_os_type	Destination-VM OS windows or Unix based
11	dst_port	Destination-VM portnumber
12	protocol	Destination-VM protocol
13	fwd	nfdump forwarding status
14	stos	nfdump Src Tos
15	ipkt	nfdump input packets
16	ibyt	nfdump input bytes
17	base_interface	Firewall interface

The selected data is filtered on destination port lower than 1024 which generates a dataset containing 14223 rows. Figure 5.3 shows a scheme on how the superset (i.e Y) is processed into different subsets, labelled and processed by the DNN.

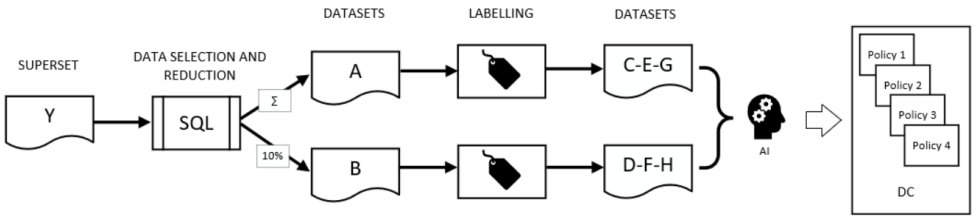


Figure 5.3: Processing superset into policies

Datasets A and B use three firewall interfaces as label which is derived from the firewall dataset. Datasets C and D use VM name as labelling. On these dataset, another column is added with the labelling according to the VM name. An example of this is to label all VMs named "ADMIN" as one group while "DEV" may be labelled as another group. In total, there are 12 groups for dataset C and D. From dataset C and D, a subset is selected to create the datasets E and F. By using the aggregated dataset, it is possible to manually label different VMs into groups and copy the labels to the larger data sets. For dataset G and H, the process is repeated adding other labels. If "DEV" is the basis selected from dataset C, then VM-name containing "TEST" and OS-type Unix is labelled as one group. Figure 5.4 shows a model of how the different datasets are related.

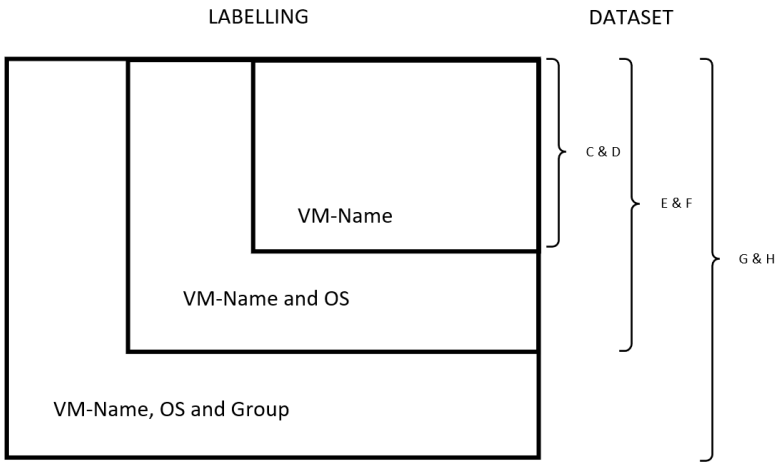


Figure 5.4: Example of dataset labelling

Table 5.5: Overview of data sets

Dataset nr:	Columns	Rows	Labels	Lablel info	Data source
A	16	1719	3	FW	$\Sigma(Y)$
B	16	14223	3	FW	Y
C	17	1719	12	VM name	$\Sigma(Y)$
D	17	14223	10	VM name	Y
E	17	481	13	VM name and dest. OS	$\subset(C)$
F	17	7547	9	VM name and dest. OS	$\subset(D)$
G	17	481	14	VM name, dest. OS, and dest. group	$\subset(C)$
H	18	7547	9	VM name and dest. OS and dest. group	$\subset(D)$

5.2 Experiments

In Chapter 2 we explain how firewall policies in a virtual environment can be created by creating security groups for a micro-segmented architecture. The experiments conducted in this thesis are designed to test if a neural network can be used to analyse data in order to create such new policies. The neural network used in this experiment research is based on the feed-forward neural network as used by Vinayakumar et al. [56], in their experiment regarding classification of cyberattacks.

5.2.1 Experimental design

Four experiments were designed to test if a neural net can predict firewall policies. Derived from the same data set, the labels are changed to test if a neural network can predict different firewall policies to be used in a micro-segmented environment. In the first two experiments test if network and infrastructure features can be used to train a neural network while in the next experiments the complex-

ity increases by adding features like operation system and grouping of VMs. The experiments are as follows.

- Experiment 1: Test if a DNN can learn when data is labelled with firewall interface
- Experiment 2: Test if a DNN can learn when data is labelled with VM name
- Experiment 3: Test if a DNN can learn when data is labelled with VM name and destination OS
- Experiment 4: Test if a DNN can learn when data is labelled with VM name, destination OS and group

Table 5.6 presents an overview of the different experiments, which data set is used, the number of epochs per experiment, and the ratio of train and test split.

Table 5.6: Experiments overview

Experiment	Data set	Dataset rows	Train size (75%)	Test size (25%)	Epochs
1.0	A	1719	1289	430	100
1.1	B	14223	10667	3556	100
2.0	C	1719	1289	430	100
2.1	D	14223	10667	3556	100
3.0	E	481	361	120	100
3.1	F	7547	5660	1887	100
4.0 a)	G	481	361	120	100
4.0 b)	G	481	361	120	500
4.1	H	7547	5660	1887	100

5.2.2 Model design

The implementation of the neural network is done by using Phyton and Keras and is based on the code provided by Vinayakumar [63]. While the input and output layers of a neural network are decided by the dataset, the hidden layers in the network must be designed to make the neural network have the best performance. Hecht-Nielsen [64] stated that the number of neurons in the first hidden layer to be (2n+1), where (n) represents the number of input features. Heaton [65] stated the following rule-of-thumb to select the number of hidden layers in a deep neural network:

- Between the size of the input layer and the size of the output layer.
- 2/3 the size of the input layer, plus the size of the output layer.
- less than twice the size of the input layer.

Since the design and selection of hyperparameters in the deep neural network is complex, a series of tests where conducted to select a model. The test was performed with different hidden layers in the neural network with dataset B. The performance of each neural network was measured after 100 epochs. Table 5.7, show the output performance parameters for different layers.

Table 5.7: Performance parameters of various hidden layers

First layer	Second layer	TPR	FPR	Acc
40	20	0.981	0.008	0.997
20	10	0.974	0.012	0.997
10	8	0.976	0.01	0.998
8	4	0.969	0.012	0.997
9	4	0.974	0.012	0.997
4	2	0.972	0.013	0.994
2	2	0.965	0.017	0.993

The table show that there is not much difference in the performance parameters when the number of neurons in the hidden layers are increased. This may change if other data sources are added, and a more complex labelling is used. By this the number of neurons in the first hidden layer where set to nine and the second to four, reducing the number of layers for each layer. The selected model was then used the experiment 1-4

5.2.3 Implementation

The steps in the code are as follows:

1. Input data:
    - Open and load the csv containing the dataset and calculate the number of columns containing data and unique labels in the column containing the training labels. The input layer to the model, called features, is the number of columns in the data set. The number of features in the output layer is the number of unique labels in the dataset. Figure 5.6 presents an example on how a data set is loaded into a neural network. The figure shows how the number of input features in the model is related to the number of columns (1-6) in the data set. The unique number of labels used to train the model is the seventh column in the dataset. The number of labels in the dataset is related to the number of output features in the neural network. In this example there are two output features, A and B. The rows 1-6 in the dataset contain data and one label, A or B. In such a way, the neural network will learn from the input data, column 1-6, to create the output label, A or B.

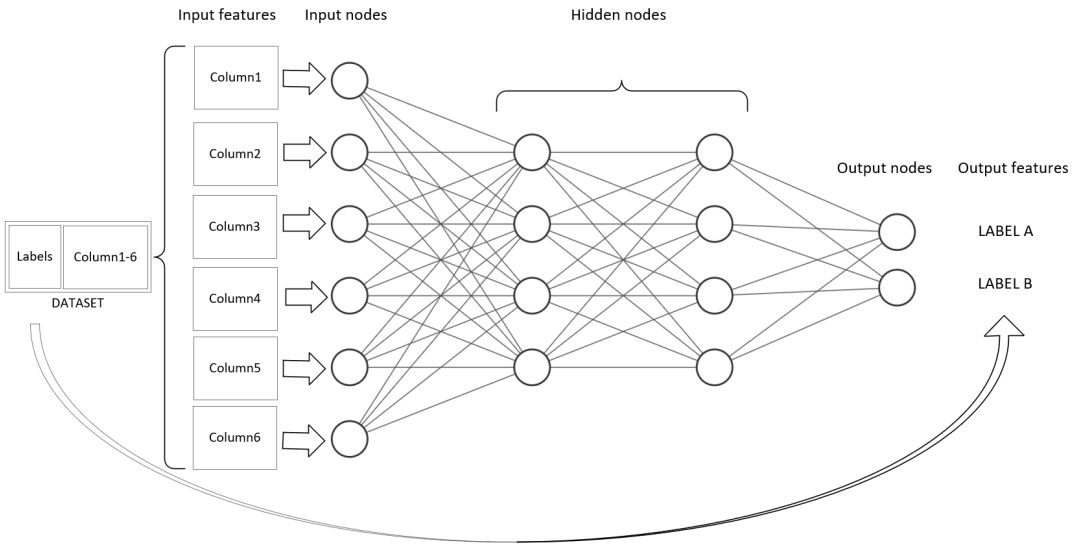


Figure 5.5: How a data set is loaded into a DNN

2. Normalisation:
   - Step two is the normalisation of data using prepossessing module in the scikit-learn library, which remove the means and scale the data to a normal distribution [66] .
3. One Hot Encoding (OHE):
   - OHE is then used to create a binary representation of each label before the data set it is split into a training and validation dataset the using the scikit-learn library[67]. Figure 5.5, show an example of how OHE encodes label into binary.

Label
A
B

One Hot Encoding

A	B
1	0
0	1

Figure 5.6: One Hot Encoding

4. Split dataset into training and testing:
   - The input dataset is randomised using a random generator before it split into a training and testing data set using the scikit-learn preprocessing module [68]. By setting a fixed value to the random generator, the selected data will select the same data each time the code is executed, which will make the experiment reproducible. According scikit-learn documentation, any random number above 0 can be used

[69]. In this experiment number 42 is used. As long as the input and no other parameter is changed, the result should be identical each time the code is executed.

- 80% of the dataset is used for training and 20% for testing. This relates to the Pareto principle which states that *states that for many phenomena, about 80% of the consequences are produced by 20% of the causes* [70]. However, since some of the data sets are relatively small, the ratio is changed to 75% for training and 25% for testing in order to make the model have more data to test on.

5. Initialise model:

   - Initialise neural network using Keras Model functional API [71].The first layer in the neural network is the number of input features derived from the columns in the data set while the hidden layers in the model are selected by the operator. A rule of thumb is to choose the number of neurons in the second layer are to half of the number of input features while the next layers is found by testing and failing. In this experiment, nine neurons are selected for the second layer and four for the third layer. Figure 5.7 shows a model of the neural network used in the experiments using two hidden of nine and four nodes. The only difference between the various experiments are the numbers of input and output features while the second and the third layer in the neural network is unchanged.
   - Rectified Linear Unit (ReLU) is used as activation function in the second and third layer of the neural network.
   - Softmax is used as activation function in the output layer.
   - Loss is calculated by using categorical crossentropy function [72] and with Adam [73] as optimiser.
   - The accuracy class is used to calculate the metrics of the neural network. When training the model, 200 epochs are used. The batch size was set to 32 according to the research done by Masters et al. [74].

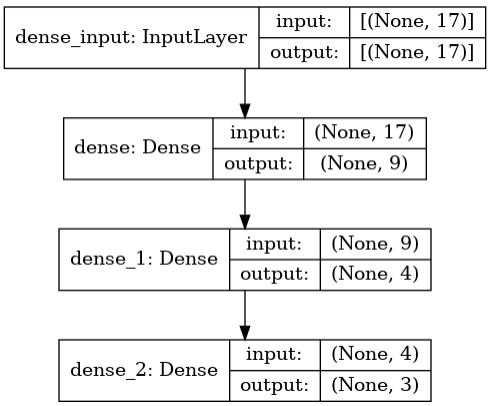


Figure 5.7: Model of the DNN used in the experiments

6. Train model and save model
    - The model from the previous stages executed to start the training process. After training, the model was saved.
7. Evaluate model
    - While training, the model saves its results for each epoch. These results are graphed to visualise the performance of the model. Results such as True Positives, True Negatives, False Positives and False Negatives are also calculated. The performance per output class is also presented in a classification report together with a Confusion Matrix.

The Phyton source code used in this project is made publicly available on Github [75].

5.3 Results performance and discussion

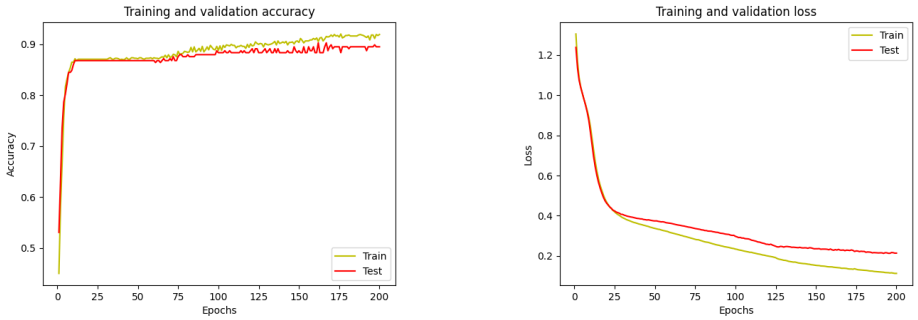
All results from each experiment are presented as graphs for training validation accuracy, training validation loss, confusion matrix and classification report. For each experiment, the results are discussed.

5.3.1 Experiment 1

Test if a DNN can learn when data is labelled with firewall interface.

Results from experiment 1.0

The results from the experiment are as follows:



(a) Accuracy **(b)** Loss

Figure 5.8: Graphed result from experiment 1.0

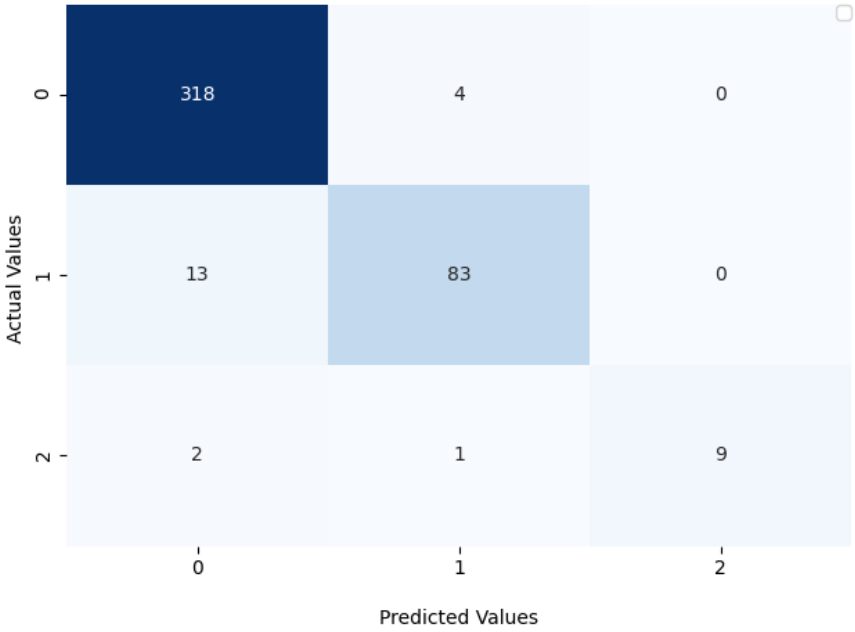


Figure 5.9: Experiment 1.0 confusion matrix

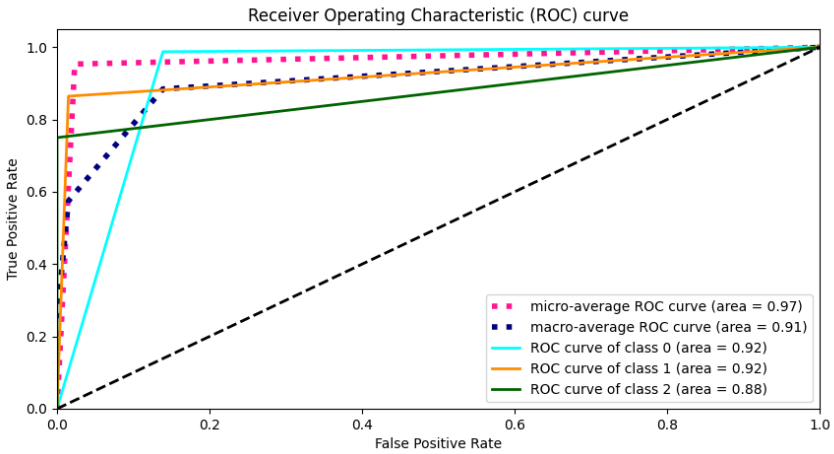


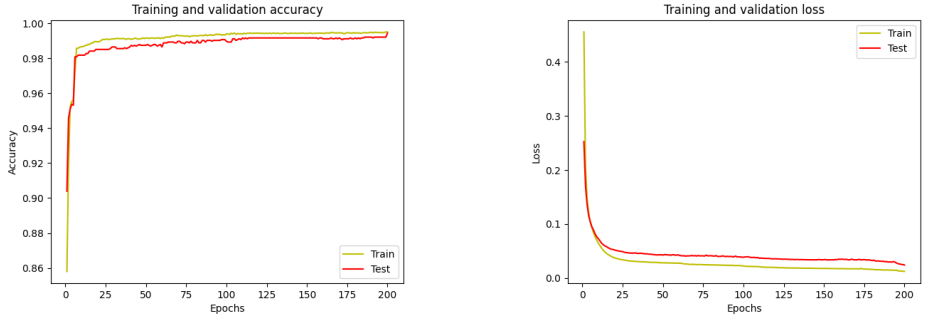
Figure 5.10: Experiment 1.0 ROC

Table 5.8: Experiment 1.0 Classification report

	precision	recall	f1-score	support
0	0.95	0.99	0.97	322
1	0.94	0.86	0.90	96
2	1.00	0.75	0.86	12
accuracy			0.95	430
macro avg	0.97	0.87	0.91	430
weighted avg	0.95	0.95	0.95	430

Results from experiment 1.1

The results from the experiment are as follows:



(a) Accuracy **(b)** Loss

Figure 5.11: Graphed result from experiment 1.1

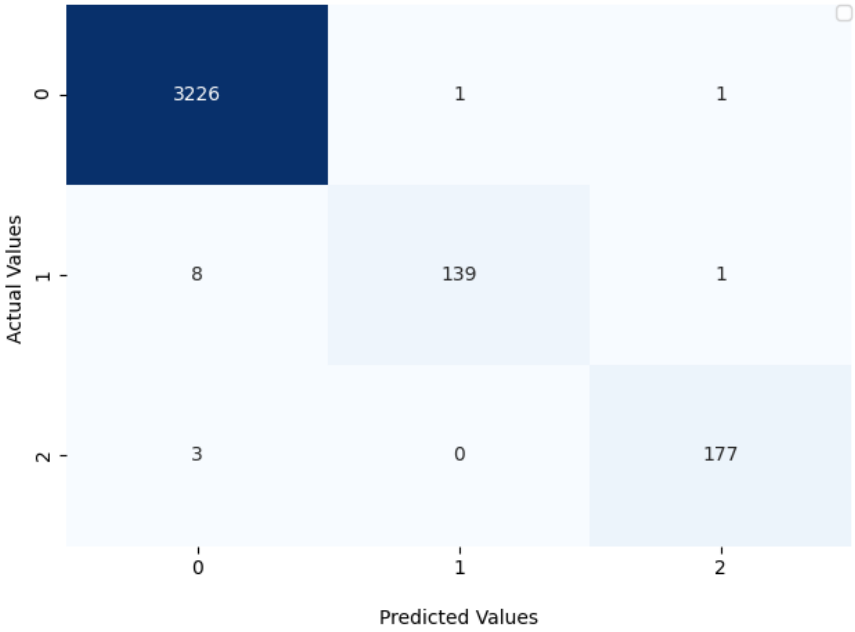


Figure 5.12: Experiment 1.1 confusion matrix

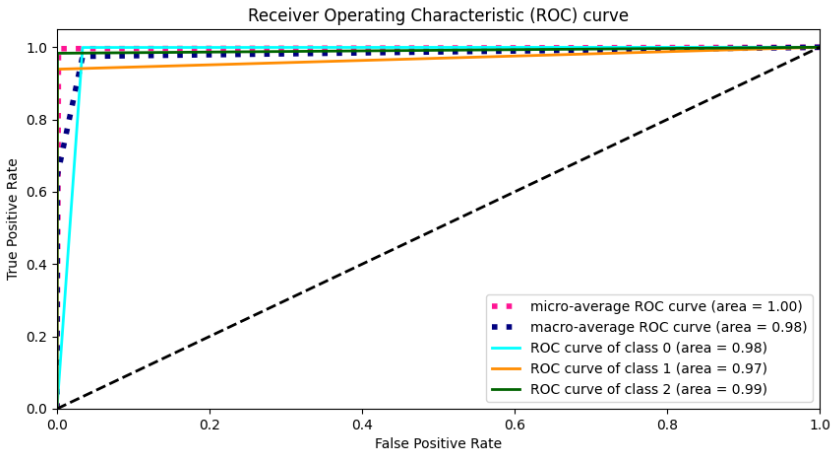


Figure 5.13: Experiment 1.1 ROC

Table 5.9: Experiment 1.1 Classification report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	3228
1	0.99	0.94	0.97	148
2	0.99	0.98	0.99	180
accuracy			1.00	3556
macro avg	0.99	0.97	0.98	3556
weighted avg	1.00	1.00	1.00	3556

5.3.2 Experiment 1 discussion

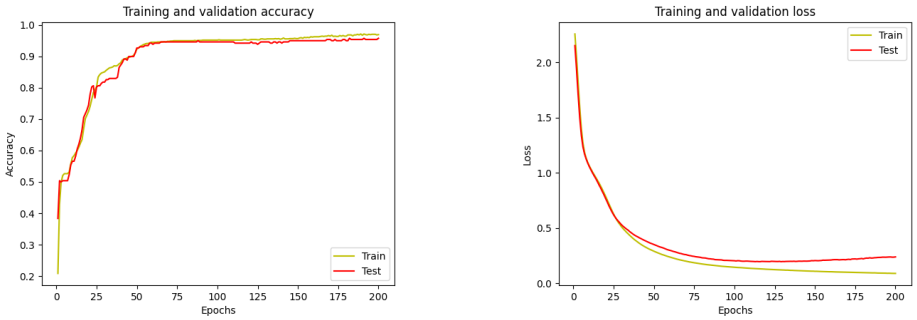
In the first experiment, the DNN finds a connection between the data set and the labelling reaching an accuracy well above 80% after just a few epochs. The accuracy-graph then flats out before it has a small increase in the accuracy from 75 to 200 epochs. This is a very good score indicating that the DNN model works as intended. However, since the data set is not balanced, the score per class is different. The recall is under 90% for two of the classes even though the accuracy is at 95% for the model. By observing the ROC in Figure 5.10, we can see that class 2 does not perform as well as class 0 and 1. Observing the loss function, the two curves start to deviate after 25 which increases to the end. This indicates that the model is overfitting. In contrast, experiment 1.1 shows much better performance for all the three classes in the data set. Observing the accuracy curve, the model trains to very high accuracy after a few epochs. However, also in this experiment we can observe that the graphs in the loss function deviate although this difference is constant. This indicates that the model can learn very fast and after approximately 25 epochs the model has reached the an accuracy above 98%. The difference between these two experiments is the amount of data that is fed into the neural network. Even though the overall accuracy in experiment 1.0 was very well, the performance per class was not that good. In experiment 1.1, all the three classes performed very good. This indicates that the model needs enough data in all the labelled classes in order to train all classes to a high degree of accuracy.

5.3.3 Experiment 2

Test if a DNN can learn when data is labelled with VM name.

Results from experiment 2.0

The results from the experiment are as follows:



(a) Accuracy **(b)** Loss

Figure 5.14: Graphed result from experiment 2.0

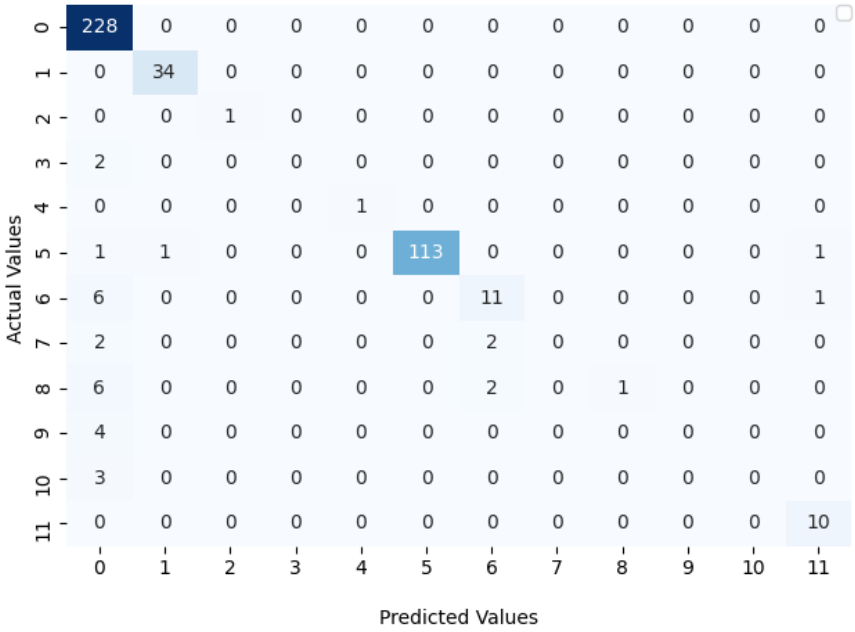


Figure 5.15: Experiment 2.0 confusion matrix

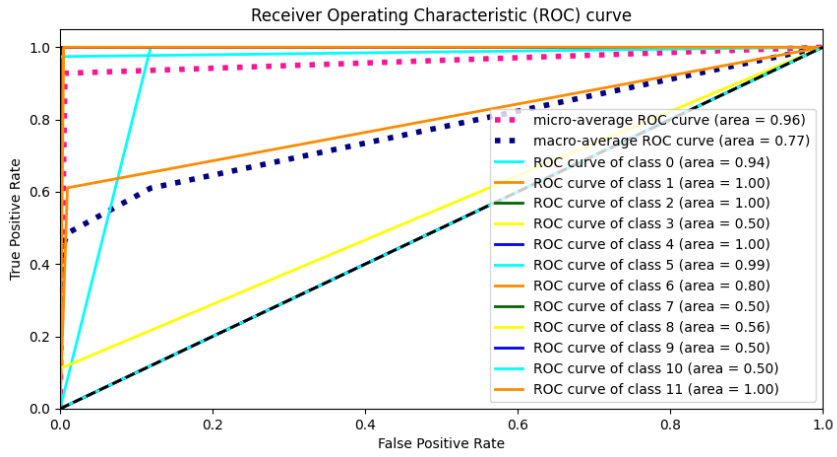


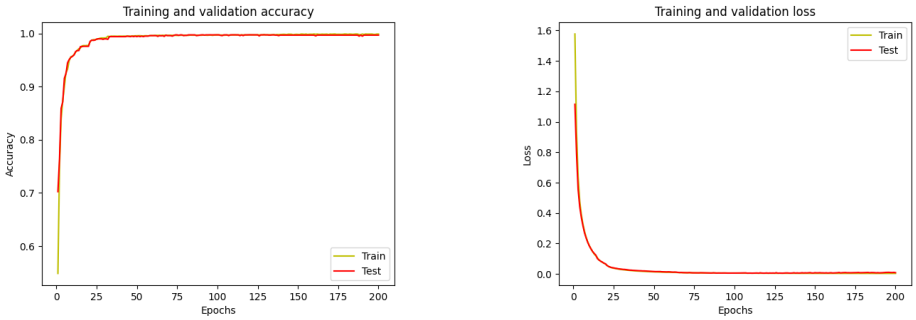
Figure 5.16: Experiment 2.0 ROC

Table 5.10: Experiment 2.0 Classification report

	precision	recall	f1-score	support
0	0.90	1.00	0.95	228
1	0.97	1.00	0.99	34
2	1.00	1.00	1.00	1
3	0.00	0.00	0.00	2
4	1.00	1.00	1.00	1
5	1.00	0.97	0.99	116
6	0.73	0.61	0.67	18
7	0.00	0.00	0.00	4
8	1.00	0.11	0.20	9
9	0.00	0.00	0.00	4
10	0.00	0.00	0.00	3
11	0.83	1.00	0.91	10
accuracy			0.93	430
macro avg	0.62	0.56	0.56	430
weighted avg	0.90	0.93	0.91	430

Results from experiment 2.1

The results from the experiment are as follows:



(a) Accuracy **(b)** Loss

Figure 5.17: Graphed result from experiment 2.1

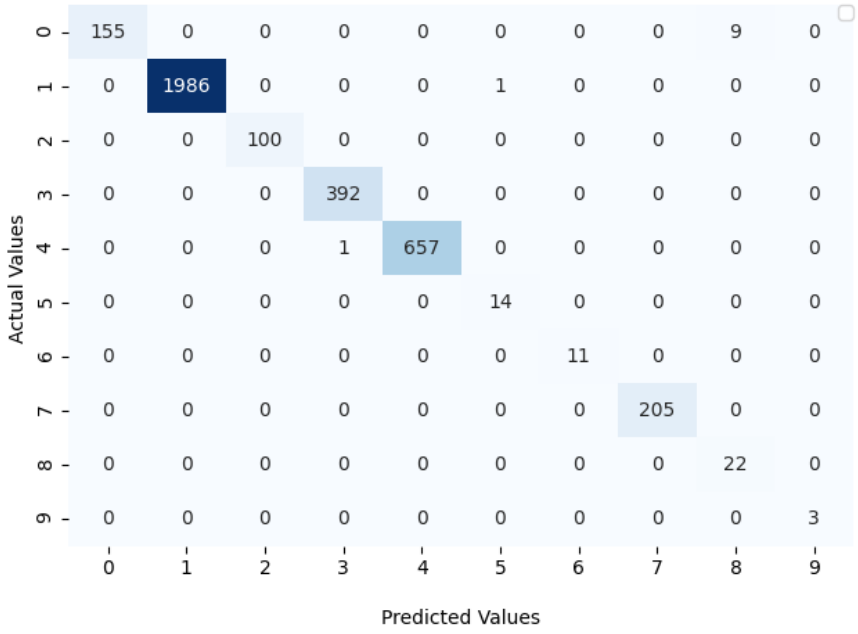


Figure 5.18: Experiment 2.1 confusion matrix

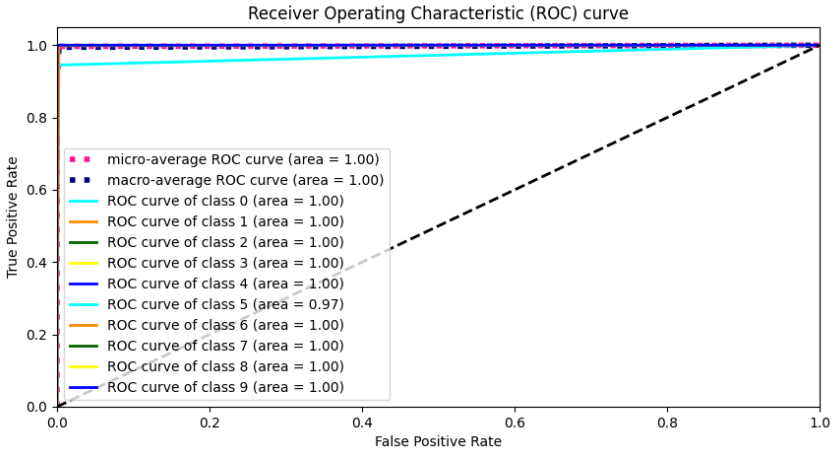


Figure 5.19: Experiment 2.1 ROC

Table 5.11: Experiment 2.1 Classification report

	precision	recall	f1-score	support
5	1.00	0.95	0.97	164
4	1.00	1.00	1.00	1987
8	1.00	1.00	1.00	100
0	1.00	1.00	1.00	392
1	1.00	1.00	1.00	658
9	0.93	1.00	0.97	14
3	1.00	1.00	1.00	11
7	1.00	1.00	1.00	205
6	0.71	1.00	0.83	22
2	1.00	1.00	1.00	3
accuracy			1.00	3556
macro avg	0.96	0.99	0.98	3556
weighted avg	1.00	1.00	1.00	3556

5.3.4 Experiment 2 discussion

In experiment 2, the data is labelled based on the name of the VMs. In the data set containing aggregated network traffic, the performance parameter indicates that there are too little data for some classes for the model to train. As in experiment 1, the data set is imbalanced, meaning that some classes have few samples. As a consequence of this, there are several classes that have calculated precision of zero. For the classes that have enough data, the performance are very good in most cases. The model has an overall accuracy over 90%, which indicates that as a

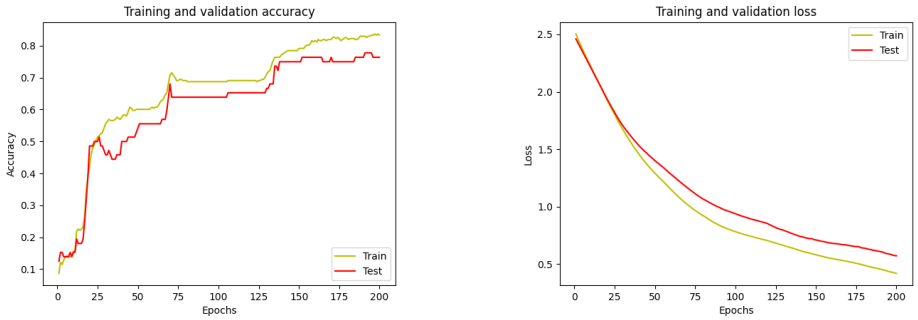
classifier the model does what it is intended to do. However, the average precision is at 56%, which indicates that the model either is not optimal for classifying the data or that the model has not been trained on enough data to optimise the neural network. In experiment 1.1, using a data set with much network traffic, the model predicts a 100% accuracy for almost all the classes. One class has an accuracy at 83% where all the missed labels are classified as another class. This could indicate that the data has been wrongly labelled. The average accuracy for experiment 2.1 is at 98%, which is an strong indication that with enough data the model performs very well.

5.3.5 Experiment 3

Test if a DNN can learn when data is labelled with VM name and destination OS.

Results from experiment 3.0

The results from the experiment are as follows:



(a) Accuracy **(b)** Loss

Figure 5.20: Graphed result from experiment 3.0

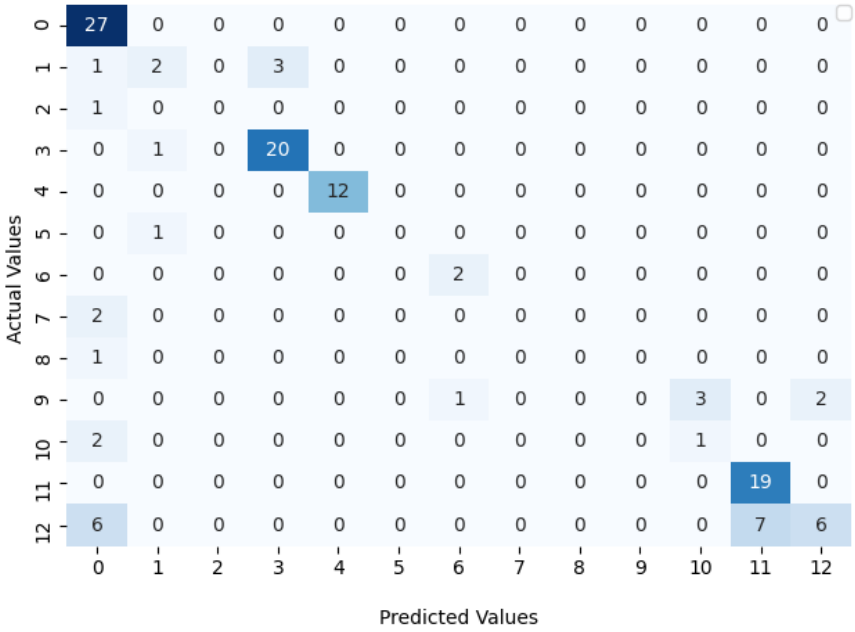


Figure 5.21: Experiment 3.0 confusion matrix

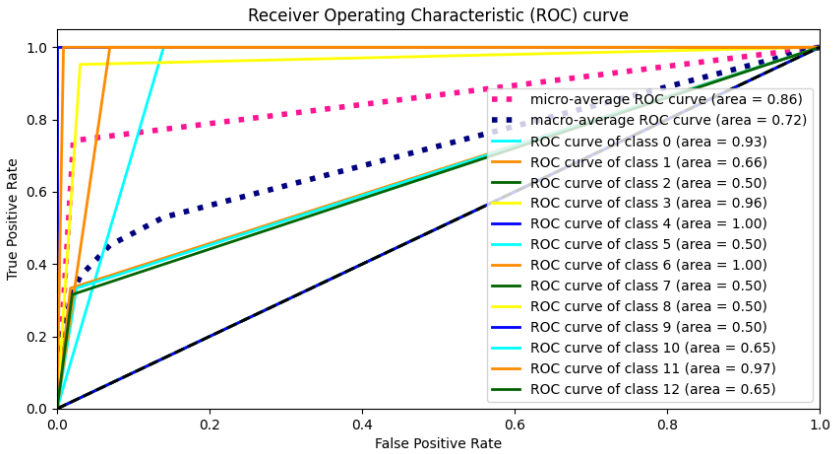


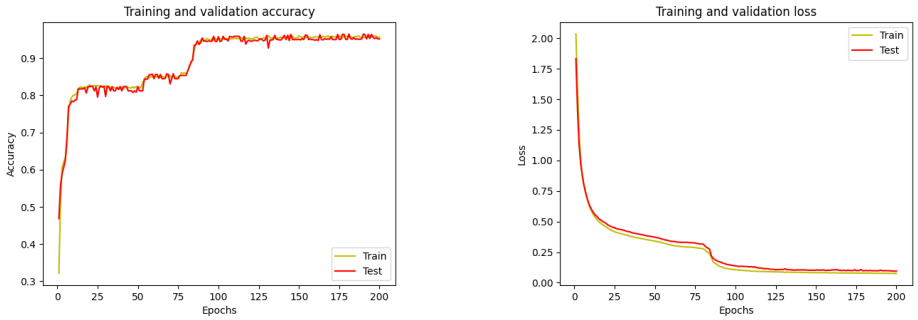
Figure 5.22: Experiment 3.0 ROC

Table 5.12: Experiment 3.0 Classification report

	precision	recall	f1-score	support
0	0.68	1.00	0.81	27
1	0.50	0.33	0.40	6
2	0.00	0.00	0.00	1
3	0.87	0.95	0.91	21
4	1.00	1.00	1.00	12
5	0.00	0.00	0.00	1
6	0.67	1.00	0.80	2
7	0.00	0.00	0.00	2
8	0.00	0.00	0.00	1
9	0.00	0.00	0.00	6
10	0.25	0.33	0.29	3
11	0.73	1.00	0.84	19
12	0.75	0.32	0.44	19
accuracy			0.74	120
macro avg	0.42	0.46	0.42	120
weighted avg	0.68	0.74	0.68	120

Results from experiment 3.1

The results from the experiment are as follows:



(a) Accuracy **(b)** Loss

Figure 5.23: Graphed result from experiment 3.1

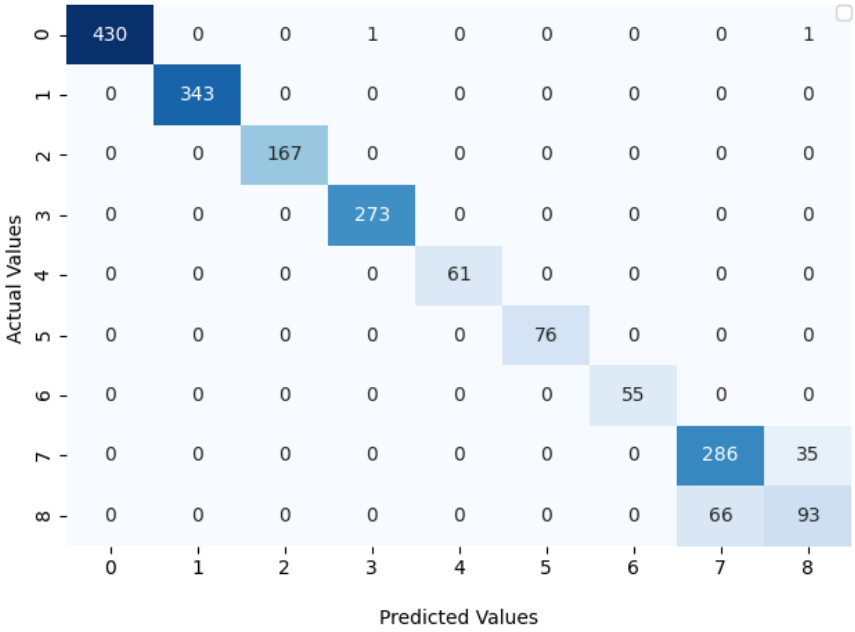


Figure 5.24: Experiment 3.1 confusion matrix

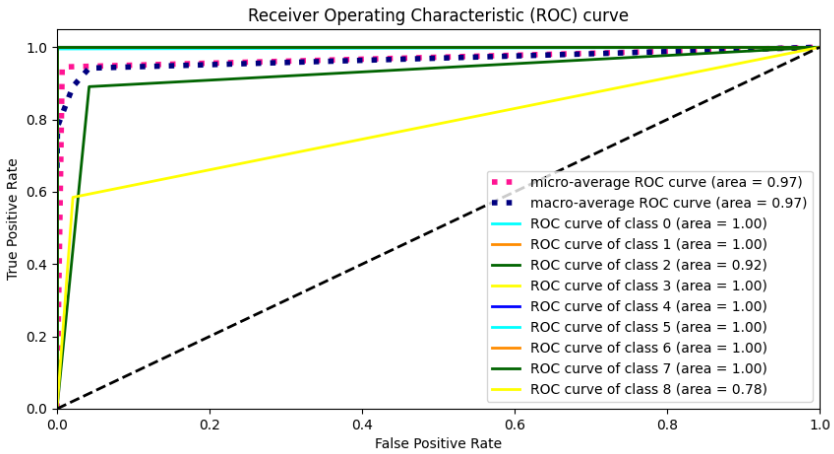


Figure 5.25: Experiment 3.1 ROC

Table 5.13: Experiment 3.1 Classification report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	432
1	1.00	1.00	1.00	343
3	1.00	1.00	1.00	167
4	1.00	1.00	1.00	273
5	1.00	1.00	1.00	61
6	1.00	1.00	1.00	76
7	1.00	1.00	1.00	55
2	0.81	0.89	0.85	321
8	0.72	0.58	0.65	159
accuracy			0.95	1887
macro avg	0.95	0.94	0.94	1887
weighted avg	0.94	0.95	0.94	1887

5.3.6 Experiment 3 discussion

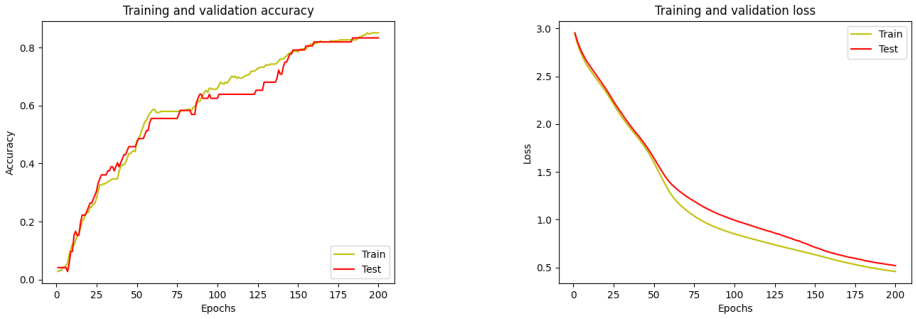
In this experiment, the complexity has increased by labelling the data by destination VM name and OS. In this example, there are also many more labels, which indicates a much more granular segmentation of firewall policies. Because of the increased complexity, the graphs that the model probably needs more epochs in order to learn the different classes. After 200 epochs, the accuracy is 84%. However, as for the previous experiments, the data set containing aggregated network information, the model performs less accurately for classes containing less information. Because the data set is a subset, there are in general less data in the data set. Since this data set is classified into several classes, the number over data per class is limited. Even with little data to train on there is a steady increase in the accuracy. However, the loss deviates indicating that the model is over-fitting but as in the previous experiments this can be related to the fact that there is too little data for the model to train on. In experiment 2.1, the number of samples per class is increased. In this experiment, the model performance is very good for all classes. With enough information, the model learns fast and reaches an accuracy above 80% after approximately 25 epochs. In this experiment, there are three distinct incremental steps in the accuracy and consequently decremented drop in the loss curves. This may relate to when the model discovers the different features in the dataset, which relates to the granular policy (i.e VM name and OS). The classification report shows that the model classifies almost all the classes perfectly. For those classes that have more false positives, there is a chance that these classes has been labelled wrong. The overall macro average f1-score for the model is at 94% which is a very high score indicating that the model can be used as a classifier for this type of data.

5.3.7 Experiment 4

Test if a DNN can learn when data is labelled with VM name, destination OS and group.

Results from experiment 4.0

The results from the experiment are as follows:



(a) Accuracy **(b)** Loss

Figure 5.26: Graphed result from experiment 4.0

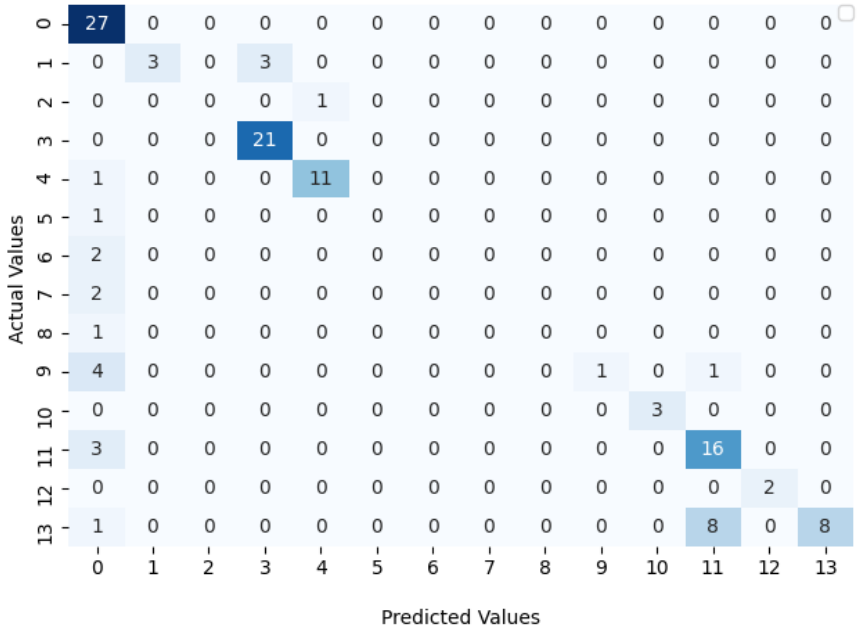


Figure 5.27: Experiment 4.0 confusion matrix

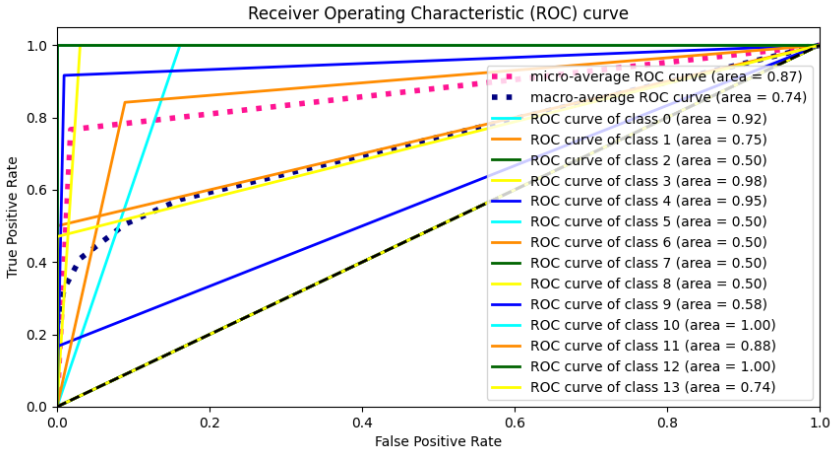


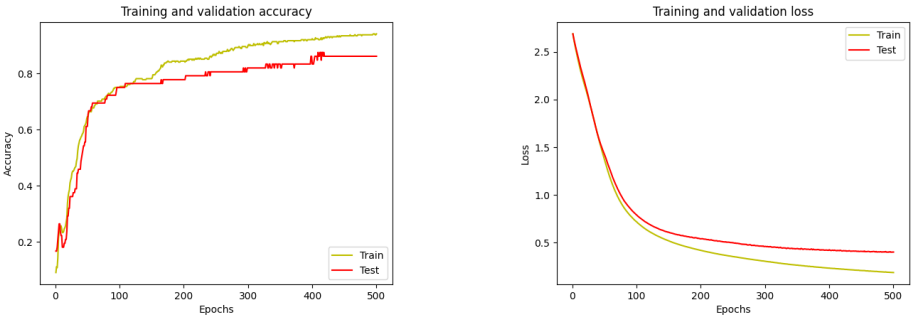
Figure 5.28: Experiment 4.0 ROC

Table 5.14: Experiment 4.0 Classification report

	precision	recall	f1-score	support
0	0.64	1.00	0.78	27
1	1.00	0.50	0.67	6
2	0.00	0.00	0.00	1
3	0.88	1.00	0.93	21
4	0.92	0.92	0.92	12
5	0.00	0.00	0.00	1
6	0.00	0.00	0.00	2
7	0.00	0.00	0.00	2
8	0.00	0.00	0.00	1
9	1.00	0.17	0.29	6
10	1.00	1.00	1.00	3
11	0.64	0.84	0.73	19
12	1.00	1.00	1.00	2
13	1.00	0.47	0.64	17
accuracy			0.77	120
macro avg	0.58	0.49	0.50	120
weighted avg	0.77	0.77	0.73	120

Results from experiment 4.1

This experiment is similar to experiment 4.0 running 500 epochs to test if the neural network can learn from the data when running additional epochs. The results from the experiment are as follows:



(a) Accuracy **(b)** Loss

Figure 5.29: Graphed result from experiment 4.0 with 500 epochs

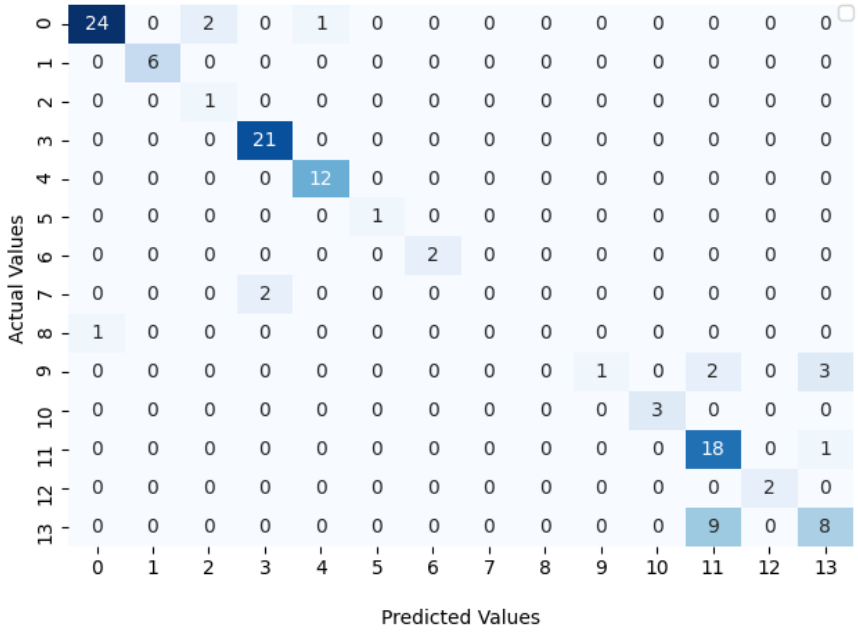


Figure 5.30: Experiment 4.0 with 500 epochs confusion matrix

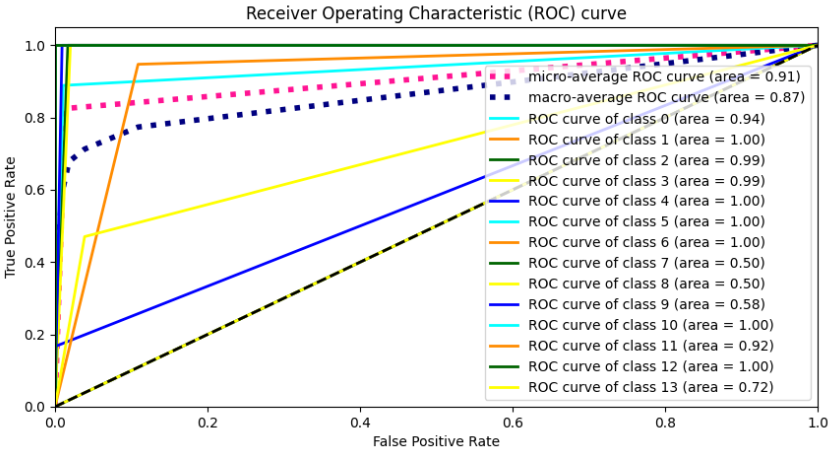


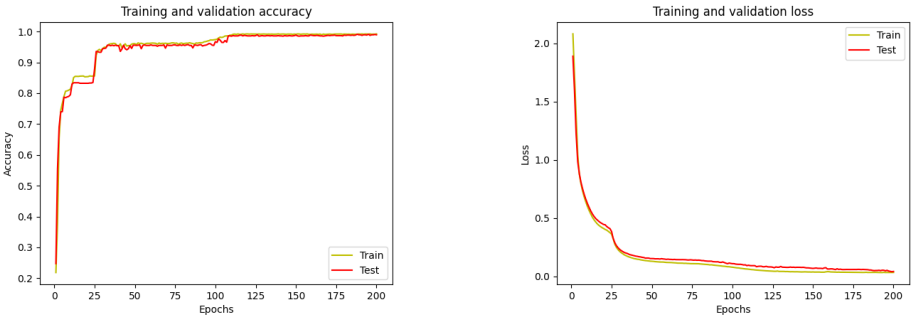
Figure 5.31: Experiment 4.0 with 500 epochs ROC

Table 5.15: Experiment 4.0 Classification report 500 epochs

	precision	recall	f1-score	support
0	0.96	0.89	0.92	27
1	1.00	1.00	1.00	6
2	0.33	1.00	0.50	1
3	0.91	1.00	0.95	21
4	0.92	1.00	0.96	12
5	1.00	1.00	1.00	1
6	1.00	1.00	1.00	2
7	0.00	0.00	0.00	2
8	0.00	0.00	0.00	1
9	1.00	0.17	0.29	6
10	1.00	1.00	1.00	3
11	0.62	0.95	0.75	19
12	1.00	1.00	1.00	2
13	0.67	0.47	0.55	17
accuracy			0.82	120
macro avg	0.74	0.75	0.71	120
weighted avg	0.83	0.82	0.80	120

Results from experiment 4.2

The results from the experiment are as follows:



(a) Accuracy **(b)** Loss

Figure 5.32: Graphed result from experiment 4.2

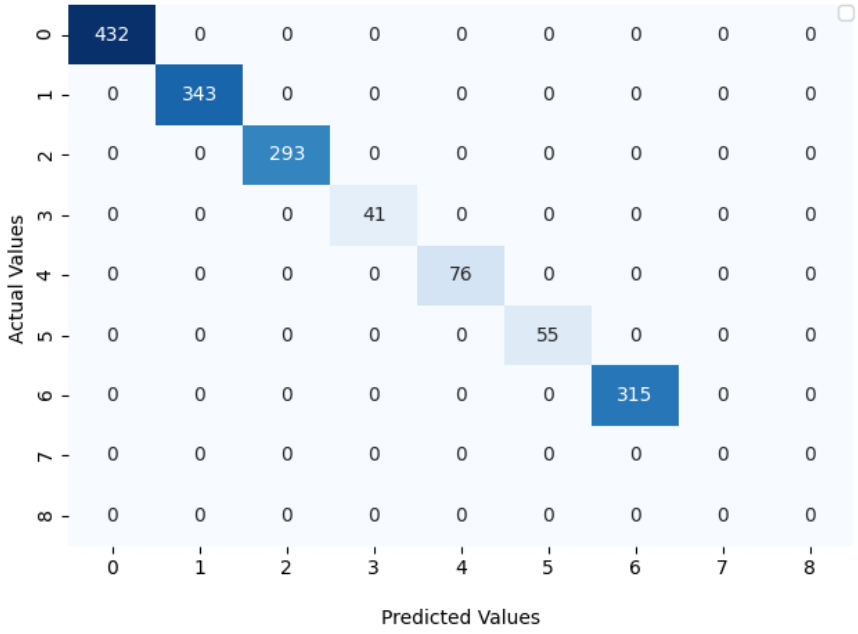


Figure 5.33: Experiment 4.2 confusion matrix

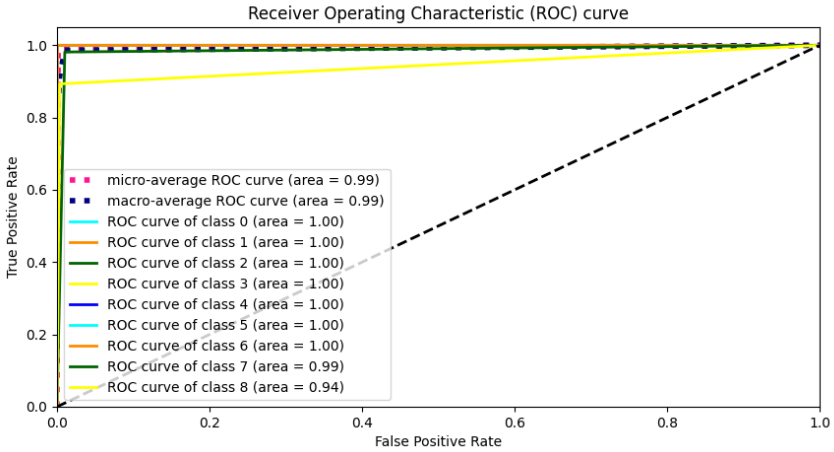


Figure 5.34: Experiment 4.2 ROC

Table 5.16: Experiment 4.1 Classification report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	432
1	1.00	1.00	1.00	343
3	1.00	1.00	1.00	293
4	1.00	1.00	1.00	41
5	1.00	1.00	1.00	76
6	1.00	1.00	1.00	55
7	0.95	0.98	0.97	321
11	0.00	0.00	0.00	0
13	0.00	0.00	0.00	0
micro avg	0.99	1.00	0.99	1561
macro avg	0.77	0.78	0.77	1561
weighted avg	0.99	1.00	0.99	1561

5.3.8 Experiment 4 discussion

In this experiment the complexity of the labelling is increased even more compared to experiment 3 by adding a group name in addition to destination VM and OS. Such a group name can relate to special applications that are grouped into a specific security group. In the first experiment, there is a steady increase in the accuracy which indicates that the model learns more and more for each epoch. However, compared with the previous experiment there is no steep increase in the accuracy curve at the first epochs. The reason for this may be that the labelling is more complex, indicating that the model does not find significant connections

between the input data and the labels that fast. In such a way, the model needs many more epochs to learn from the data. As a result of this, an additional experiment (i.e., experiment 4.1) has been conducted to evaluate this. This experiment is the same as experiment 4.0 but with 500 epochs. This experiment shows that the overall macro average f1-score increases from 50% to 71%. This indicates that the model learns at a steady pace although the accuracy flattens after approximately 400 epochs. The accuracy and loss graph also show that the model is over-fitting which is probably related to the fact that there is too little data for the model to train on. As in the previous experiments, the aggregated data source has little data per class, which affects the overall performance metrics of the model. Some classes do not have enough data for the model to classify the data. Comparing the results with the dataset containing many more samples per class, the results show that the model performs particularly well since the accuracy is high.

Chapter 6

Discussion

In this chapter, the major findings are presented and discussed. In the first section, the findings regarding different firewall models are discussed. Secondly, the experiments together with the performance parameters are discussed. In the last part a method on how to make best use of the research conducted in the previous chapter are discussed.

6.1 Firewall models

In Chapter 4.2, different firewall models are listed. In cloud environments there are three types of firewall models that are used: *physical, subnet-level* and *kernel-based* firewalls. Physical firewalls are located outside the virtualised environment and because of this they are not an integrated part of the hardware in the environment. This affects the performance since the traffic must leave the virtualised environment when two VMs communicate. In addition to this, the management of a physical firewall is not integrated with the virtualised environment. Both subnet-level and kernel-based firewall models are hosted on a virtual environment. The main difference between these two deployment models is how they are hosted in the virtual environment, and this affects the performance and management of the firewall. The subnet-level firewall is a VM hosted on a hypervisor where the firewall application is installed on the operating system on the VM. Such a firewall model can easily be deployed in a virtual environment to segment network traffic into different zones. However, this firewall model has several disadvantages compared to the kernel-based firewall. This because the firewall is an application installed on VM hosted on the hypervisor, which limits the performance of the firewall and the integration and management of the VM and the hypervisor. A kernel-based firewall is to be considered an improvement of the subnet-level firewall. In the kernel-based model, the firewall is hosted between the hypervisor kernel and the network interface. Performance and management in this model are managed through the hypervisor and not at the individual firewall VM as on a subnet-level firewall.

In Chapter 2, the complexity of a micro-segmented infrastructure and how firewall policies in such an environment are implemented and explained. Firewall rules have traditionally been enforced at specific point in computer network to segment different zones from each other. With micro-segmentation, policies are abstracted at a higher-level forming security group containing parameters from both application, infrastructure and the network. The method of abstracting firewall policies is recommended by NIST and enforces security at much more granular level than traditional firewall rules. NIST VM-FW-R4 [53], recommends the implementation of security groups. Such implementation can involve more complex rules and it can be very demanding for an organisation to migrate to such a platform. New policies can be creating by listing all the available resources in a DC and manually create new policies, but this is a very labour-intensive task, and the risk of human error is large. Another approach is to make use of commercially available tools for analysing and monitoring the current infrastructure to create new policies with this tool. Such tools provide insight to the infrastructure of an organisation and would help the organisation migrate to a new deployment model. However, none of the listed tools make use of machine learning to classify the data gathered from the data centre to produce new policies. In the next chapter, a series of experiments are conducted to test if this is possible.

6.2 Experimental result analysis

In this section, the overall results are discussed. An overview of all the different results are then presented and discussed.

6.2.1 Result discussion

In this thesis, a series of experiments has been conducted to investigate if it is possible to train a deep neural network to classify structured data into policies. Table 6.1 shows an overview of the performance results from the experiments. The results from the experiments prove that a DNN can learn very fast from the training data in all four experiments with a high level of accuracy. In all the experiments the results indicate that the DNN performs very well when enough data per class (i.e., label) is processed. Since the performance in all experiments is high, it indicates that the model works as intended. A high performance indicates that the labelling of the data is very accurate. Since the labelling of the data is done with help from a network engineer with in-depth knowledge about the infrastructure, there are strong indications that the datasets are labelled with very high degree of accuracy. There is also a chance that the data is biased. There may be sources of data that are not present in the data set, or the labelling is biased, which could affect the result.

Table 6.1: Performance overview

	Macro-average			Micro-avagerage			
Experiment	**Precision**	**Recall**	**f1-score**	**Precision**	**Recall**	**f1-score**	**accuracy**
1.0	0.95	0.68	0.74				0.92
1.1	0.99	0.98	0.99				1.00
2.0	0.60	0.63	0.61				0.95
2.1	0.86	0.89	0.87	1.00	0.99	1.00	
3.0	0.38	0.43	0.40				0.74
3.1	1.00	1.00	1.00				1.00
4.0	0.33	0.39	0.35				0.58
4.1	0.82	0.79	0.80				0.91
4.2	0.78	0.78	0.78	1.00	1.00	1.00	

6.3 Research questions

- Question 1. How secure are the existing different models in virtualised environments?

  In addition to physical firewalls, there are two models of virtualised firewall. These two models are subnet-level virtual firewall and kernel-based firewall. The subnet-level firewall is a firewall installed as an application on a VM while the kernel-based firewall is installed directly on the hypervisor between the VM and the network interface. The main disadvantage for a physical firewall compared with the virtualised is that the network traffic from one VM to another must flow outside the virtualised environment, which can lead to congestion and latency. Another problem with physical firewall is that there are limited possibilities of integrating the firewall with the virtualised environment and consequently less automated procedures for enforcing policies. In contrast, virtualised firewalls are integrated in the virtualised environment and traffic can be routed directly between VMs in the datacentre. Because there are many more disadvantages for sub-net level firewall than for kernel-based firewalls, we can conclude that the kernel-based firewall is much more efficient. The kernel-based firewall has several advantages compared with the other two deployment models and can be considered as a further development of the subnet-level firewall. Since the kernel-level firewall is placed at kernel level in the hypervisor, the performance is much better. It is recommended for all types of virtualised firewalls to have the ability to abstract higher level security policies. This type of granular level of firewall policies in combination with an efficient management platform is called micro-segmentation. A micro-segmented firewall creates security at a granular level in the virtual environment by segmenting application in addition to network. Such granularity limits the attack surface in the environment and ensures that if one VM is compromised, lateral movement in the infrastructure is mitigated.

- Question 2. How can a dataset of firewall rules from a traditional perimeter firewall be abstracted into a new set of policy rules in the virtual environment?

  The experiments conducted in this thesis show a strong indication that machine learning can be used to solve this problem if enough data are fed into the neural network. However, training such a model can be difficult and demand in depth knowledge of the data and the infrastructure. The experiment in this thesis was derived from a live company environment. Another approach could be to create a simulated environment and train a more generic neural network with data from this network. This model could then be applied on the corporate infrastructure to test how the model would behave on data from this environment. The different data sources must also be discussed. There may be too few data sources in the derived data sets to create policies for all the possibilities in the new environment. How data relates to the neural network is also interesting. The experiment shows that there is an indication that when more complex firewall policies are used, the neural network may need more epochs to train. In the experiments, a neural network with two hidden layers is chosen which performs very well on the datasets used in the experiments. If the complexity of the policies increases even more, it could be interesting to investigate if this affects the performance of the model. Vinayakumar et al. [63] used batch normalisation and dropout in their work with DNN to detect cyberattacks. This was done to speed up the training process and to avert over-fitting. Batch normalisation is a technique, which lets the model learn the optimal scale and mean for each layer in the neural network [39]. Dropout is another technique applied to neural networks which temporarily ignores (i.e, drops) random neurons during training which forces the neural network to train differently [39]. The same approach could be applied to the neural network to investigate if this affects the performance when more complex labelling (i.e, firewall policies) are used. Such an approach may affect training on datasets with few samples per class. The experiments show that model is over-fitting when classes with few samples are analysed. The number of samples needed for each class should also be investigated. In such future work, more balanced datasets should be used to investigate if the model performance for each class improves.

- Question 3. How can a dataset of firewall rules migrate between cloud environments where the CSP has different firewall models?

The experiments conducted to answer question 2, can be applied to migrate from a traditional network architecture to a micro-segmented architecture. By analysing structured and labelled data from one firewall model, machine learning can be

used to recognise new policies. Network traffic represents the policies that are enforced by the firewall. By analysing this traffic, the state of the current firewall policies can be used to transfer between different firewall models. These labels represent the policies in the new cloud model. The challenge in this process is to label the data correctly to represent the new firewall policies. If the labelling is correct, the trained model can be used to classify all VMs in the new model. When migrating between firewall models and at the same time verify that the correct rules are enforced, an automatic process of migrating policies must be in place. When machine learning has classified data from one cloud into a new set of policies, the new policies are created in the new cloud environment by REST API. Such process will demand that the new infrastructure has a distributed firewall with a secure communication such as SDN to maintain the integrity of the new policies. Figure 6.1 shows a scheme of this process.

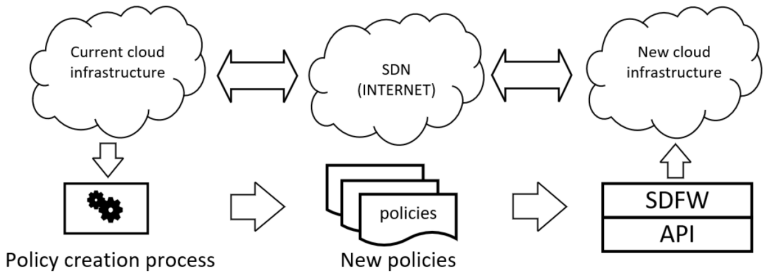


Figure 6.1: Cloud firewall policy migration

In such a process, it can be difficult to create a general model of networks that can be used in different environments. A model created in one virtual environment does not necessarily fit in another virtual environment. Different cloud models may have different policies and there may be a need for a set of different models adapted to different cloud environments. In both cases, the number of true negatives that are not classified must be manually classified. The goal of such a model is to have as little manual work as possible in such a process. However, a human must verify that the process works as intended to ensure that wrong configurations of the firewall happen because of false positives. When generating a DNN, there is also the risk of labelling the data wrongly. Budd et al. [76] described a process called human-in-the-loop (HITL) computing in medicine. Although the risk of wrong conclusions in medicine can be fatal, the risk of faults in the security of ICT-systems can also have severe consequences. HITL computing could benefit from both AI and human experience and intuition and combine the strengths of both to optimise both the process of gathering and processing data, and to verify automatic migrations between cloud environments.

Chapter 7

Conclusion and future work

The focus of this thesis has been firewall models in cloud environments and how firewall policies can be abstracted from one model to another. Different models have been studied and their differences have been outlined. While there are three models in use, one model has several advantages compared to the two others. This firewall model abstracts the firewall policy into a high level of firewall rules which is called micro-segmentation. Moving from one firewall model to another involves the creation of such abstracted firewall policies in the new model. Organisations may find this transaction to be difficult and a structured process to migrate from one model to another is needed. To help organisations make use of cloud technology more easily, a method to abstract firewall policies from one model to another is needed. In this thesis, we investigate if deep neural networks can be used to solve this problem. A series of experiments has been conducted to research the performance of a neural network model when data related to different firewall policies is fed into the neural network.

7.1 Answers to Research Questions

Question 1. How secure are the existing different models in virtualised environments?

There are three models of firewall models for virtual environments: physical, subnet-level, and kernel-based.

Considering security, a high level of abstracting firewall policies is recommended which provides security at a granular level. While physical firewalls can be used to segment network traffic at a physical level, the level of granular segmentation into network, infrastructure, and applications is not possible. Network traffic must also be routed outside the hypervisor if VMs in different zones communicate. This is not the case for subnet-level and kernel-based firewall where both models can provide such granularity. However, the kernel-based firewall has the advantage over subnet-level firewalls that the firewall is hosted between the VM and the

network interface while in a subnet-level model the firewall is an application running as an application inside the VM. If the VM hosting the firewall application is compromised, lateral movement of malware in the system is possible. In addition, the thigh integration of the firewall and the hypervisor in the kernel-based model makes this model network performance much higher than the other models. From this we can conclude that a micro-segmented kernel-based environment is to be considered as the best firewall model for a virtualised environment.

Question 2. How can a dataset of firewall rules from a traditional perimeter firewall be abstracted into a new set of policy rules in the virtual environment?

The main hypothesis was that new policy rules could be abstracted into new policies by analysing data from infrastructure with a traditional network architecture using machine learning. The data from this infrastructure represents the current setup and policies. Previous research has shown that machine learning can be applied to analyse network traffic to detect cyberattacks. The proposed model in this thesis is a machine learning model called a multi-layer perceptron which consists of a deep learning neural network with two hidden layers. From the results, we can conclude that this hypothesis is confirmed by witnessing the proposed model's performance parameters. The model is trained by applying structured data from the old environment. By labelling the data according to new policies to be used in a micro-segmented architecture, the neural network learns from this data and classifies the new policies with a very high degree of accuracy. A keypoint with this experiment is that the model needs enough data to train. If too little data is analysed by the model, the performance parameters decrease significantly. However, since the model learns very fast, not much data is needed to train a model. When increasing the complexity of the labelling representing a higher level of abstraction of the firewall policies, the model computes more epochs before it learns from the data. Overall, the experiments show that the neural network can be used to classify new policies when structured data is presented to the neural network.

Question 3. How can a dataset of firewall rules migrate between cloud environments where the CSP has different firewall models?

We present a model how firewall rules can migrate between cloud environments. If such a model is implemented, migration between firewall models can be fully automated. The motivation with such a model is to make cloud technology much more available for organisations. By automating the process of migrating between firewall models, an organisation can in a relatively easy and secure way make use of cloud technology.

7.2 Summary of contributions

Of the three firewall models in virtual environments, it is only the kernel-based firewall model that can provide firewall policies at a granular level with the desired efficiently. To migrate from one environment to a kernel-based environment can be a difficult task. Because of this, it is desirable to automate such a process. Such a process can include abstracting firewall policies by using ML. A DNN called a multi-layer perceptron is used as a classifier to analyse network traffic representing the current firewall policies in one environment. The performance of the DNN show a very high degree of accuracy as long as there are enough data for the DNN to learn from. To automate the process, a model on how firewall rules can migrate between cloud environments is proposed.

7.3 Future work

Both physical and virtual firewall models are currently in use by CSP to provide different services. Some research has been done regarding different firewall models. In addition to this, the NIST Special Publication 800-125B, documents the different firewall models and their strengths and weaknesses. These characteristics are being outlined and discussed in this thesis. While one firewall model stands out as the best alternative, the other three models are also in use. More emphasis on different models is recommended.

The amount of future work that can be done with abstracting firewall polices using neural networks is huge. This thesis is to be considered as ground work to prove that this is possible. As in other supervised learning processes, the process of labelling data can be complex. A framework for mapping different policies to the structured data is recommended. More emphasis regarding labelling data into new policies should be investigated. One approach to this problem can be to investigate if a semi-supervised learning framework can be used in supervised classification. Such a framework combining unsupervised learning with supervised learning was proposed by Gan et al. [49]. A graphical representation of the structured data could help a user to map data into policies and in such a way help the network engineer label data. The model in this thesis was trained on data derived from one infrastructure. Models from other infrastructures should also be trained and compared to investigate if a model from one infrastructure can be applied to another infrastructure. When data is labelled and a model is trained, this model can be used to predict classes of policies with a high degree of precision. The experiments in this thesis are performed on a feed-forward neural network with two hidden layers. These experiments show that when the labelling representing a more granular policy (i.e., a more complex policy) are used, the time training the model increases. How labelling of different policies affects the performance of the different network architectures are interesting for optimising the best neural network should be further investigated.

Considering migrating from one firewall model to another, a theoretical concept of how such a migration can be done is presented. Much research can be done to study such a concept. To verify if this concept works, a physical implementation (i.e., proof of concept) must be investigated. When new policies are created, these policies have to be copied to the new cloud environment. A framework for copying firewall policies by utilising the API in different cloud environments should be investigated.

Chapter 8

Source Code

Code listing 8.1: Dataset

```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import sqlite3
import os

# Import necessary modules
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
from pandas import read_csv

#Replacing textfields in dataset with unique numbers
def Dataprocessing(data):
    #Replace textfield protocol with numbers
    column_values = data[["pr"]].values.ravel()
    unique_values =  pd.unique(column_values)

    for idx,i in enumerate(unique_values):
        data.pr[data.pr == i] = idx + 1

    #Replace flg with numbers
    column_values = data[["flg"]].values.ravel()
    unique_values =  pd.unique(column_values)

    for idx,i in enumerate(unique_values):
        data.flg[data.flg == i] = idx + 100
    return data

#Get data from firewall dataset
fw_zone = read_csv('fw_zone_aton.csv', delimiter=';')

#Replace interface with number
column_values = fw_zone[["base_interface"]].values.ravel()
unique_values =  pd.unique(column_values)

for idx,i in enumerate(unique_values):
    fw_zone.base_interface[fw_zone.base_interface == i] = idx
```

```
#Get data from VM dataset
vm_data = read_csv('vm_data.csv', delimiter=';')
vm_data = vm_data.drop(columns=['vm_name','vm_network','vm_os'])

#Read data
csv = read_csv('Data25jan.csv', delimiter=',', low_memory=False)

#Selecting all data or a fraction of the data
samples = csv.sample(frac =.001)
#samples = csv

#select relevant data from nfdump file
data = samples.iloc[:,[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]]

#Convert text to number
data = Dataprocessing(data)

#Make the db in memory
conn = sqlite3.connect(':memory:')

#write the tables
vm_data.to_sql('VM', conn, index=False)
fw_zone.to_sql('FW', conn, index=False)
data.to_sql('Data', conn, index=False)

#SQL
comb = 'select␣vm.src_ip␣as␣src_ip_aton,␣vm.id␣as␣src_vm_id,␣vm.vm_vlan␣as␣src_vlan
    ↪ ,␣vm.os_id␣as␣src_os_id,␣dst_vm.src_ip␣as␣dst_ip_aton,␣dst_vm.id␣as␣
    ↪ dst_vm_id,␣dst_vm.vm_vlan␣as␣dst_vlan,␣dst_vm.os_id␣as␣dst_os_id,␣data.sp␣
    ↪ as␣src_port,␣data.dp␣as␣dst_port,␣data.pr␣as␣protocol,␣data.flg,␣data.fwd,␣
    ↪ data.stos,␣data.ipkt,␣data.ibyt,␣data.opkt,␣fw.base_interface␣FROM␣VM␣as␣vm
    ↪ ␣JOIN␣Data␣as␣data␣ON␣vm.sa␣=␣data.sa␣JOIN␣VM␣as␣dst_VM␣ON␣dst_VM.sa␣=␣data
    ↪ .da␣JOIN␣FW␣as␣fw␣ON␣vm.src_ip␣BETWEEN␣fw.aton_start␣AND␣fw.aton_stop␣AND␣
    ↪ dst_port␣<␣1024'
df = pd.read_sql_query(comb, conn)

#write data to file
df.to_csv('dataset.csv',index=False)

print("END")
```

Code listing 8.2: DNN

```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Import necessary modules
from keras.utils import np_utils
from keras.layers import Dense, Dropout, Activation, Embedding
from sklearn.model_selection import train_test_split
from tensorflow.keras import models, layers, utils, backend as K
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from itertools import cycle
from sklearn import svm, datasets
```

```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.datasets import  make_classification
from sklearn.ensemble import RandomForestClassifier

def plot_roc_curve(y_test, y_pred):

  n_classes = len(np.unique(y_test))
  y_test = label_binarize(y_test, classes=np.arange(n_classes))
  y_pred = label_binarize(y_pred, classes=np.arange(n_classes))

  # Compute ROC curve and ROC area for each class
  fpr = dict()
  tpr = dict()
  roc_auc = dict()
  for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

  # Compute micro-average ROC curve and ROC area
  fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_pred.ravel())
  roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

  # First aggregate all false positive rates
  all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

  # Then interpolate all ROC curves at this points
  mean_tpr = np.zeros_like(all_fpr)
  for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

  # Finally average it and compute AUC
  mean_tpr /= n_classes

  fpr["macro"] = all_fpr
  tpr["macro"] = mean_tpr
  roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

  # Plot all ROC curves
  plt.figure(figsize=(10,5))
  #plt.figure(dpi=600)
  lw = 2
  plt.plot(fpr["micro"], tpr["micro"],
    label="micro-average␣ROC␣curve␣(area␣=␣{0:0.2f})".format(roc_auc["micro"]),
    color="deeppink", linestyle=":", linewidth=4,)

  plt.plot(fpr["macro"], tpr["macro"],
    label="macro-average␣ROC␣curve␣(area␣=␣{0:0.2f})".format(roc_auc["macro"]),
    color="navy", linestyle=":", linewidth=4,)

  colors = cycle(["aqua", "darkorange", "darkgreen", "yellow", "blue"])
  for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
        label="ROC␣curve␣of␣class␣{0}␣(area␣=␣{1:0.2f})".format(i, roc_auc[i]),)

  plt.plot([0, 1], [0, 1], "k--", lw=lw)
  plt.xlim([0.0, 1.0])
  plt.ylim([0.0, 1.05])
  plt.xlabel("False␣Positive␣Rate")
  plt.ylabel("True␣Positive␣Rate")
```

```
  plt.title("Receiver␣Operating␣Characteristic␣(ROC)␣curve")
  plt.legend()
  plt.savefig('Experiment␣ROC.png')
  plt.show()

filepath = 'dataset.csv'
data = pd.read_csv(filepath, delimiter=';', header=1)

writePath = 'resultater.txt'
data.columns = range(data.shape[1])

#Get number of input features from data set
features = data.shape[1]

#Get number of unique labels from data set
data_labels = data[features-1].unique()

#Get the number of unique labels in the dataset
length = len(data_labels)

#Copy data to X - labels and labels to y
#Changing pandas dataframe to numpy array
X = data.iloc[:,:features-1].values
y = data.iloc[:,features-1:features].values

#Normalizing the data
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X = sc.fit_transform(X)

#One hot encoding
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder()
y = ohe.fit_transform(y).toarray()

#Linenumbers are copied
indices = np.arange(len(X))

#Split the data in training and testing, 75% as training and 25% as training
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test,indices_train, indices_test = train_test_split(X,y,
    ↪ indices, random_state=42)

# Neural network
model = Sequential()
model.add(Dense(9, input_dim=features-1, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(length, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy
    ↪ '])

#Print and save model design
print(model.summary())
utils.plot_model(model, to_file='model.png', show_shapes=True, show_layer_names=
    ↪ True)

#Train model
training = model.fit(x_train, y_train, epochs=200, batch_size=32, validation_split
    ↪ =0.2)
```

```
#Save model
model.save('model.h5')

#Evaluate model
score = model.evaluate(x_train, y_train, batch_size=32)

#Print(score)
calc_loss = "%s: %.2f%%" % (model.metrics_names[0], score[0]*100)
calc_acc = "%s: %.2f%%" % (model.metrics_names[1], score[1]*100)
print(calc_loss)
print(calc_acc)

#Predicting the Test set rules
#Greater than 0.50 on scale 0 to 1
y_pred = model.predict(x_test)
y_pred = (y_pred > 0.5)

#Plot the training and testing accuracy and loss at each epoch
loss = training.history['loss']
val_loss = training.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper right')
plt.savefig('Experiment loss.png')
plt.show()

acc = training.history['accuracy']
val_acc = training.history['val_accuracy']
plt.plot(epochs, acc, 'y', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='lower right')
plt.savefig('Experiment acc.png')
plt.show()

#Making confusion matrix that checks accuracy of the model.
#Supress warnings
import warnings
warnings.filterwarnings('ignore')  # "error", "ignore", "always", "default", "
    ↪ module" or "once"

from sklearn.metrics import classification_report
cr = classification_report(y_test.argmax(axis=1), y_pred.argmax(axis=1), labels=
    ↪ data_labels)
print('Classification report :\n',cr)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test.argmax(axis=1), y_pred.argmax(axis=1), labels=
    ↪ data_labels)
print(cm)

#Calculate performance parameters
FP = cm.sum(axis=0) - np.diag(cm)
```

```
FN = cm.sum(axis=1) - np.diag(cm)
TP = np.diag(cm)
TN = cm.sum() - (FP + FN + TP)
#print("FP: ", FP,"\nFN: ",FN,"\nTP: ",TP,"\nTN: ",TN)

FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)

#Plot confusion matrix
ax = sns.heatmap(cm, annot=True, fmt='', cbar=False, cmap='Blues')
#ax.set_title('Confusion Matrix\n\n');
ax.set_xlabel('\nPredicted␣Values')
ax.set_ylabel('Actual␣Values␣');
h,l = ax.get_legend_handles_labels()
ax.legend(h,l, borderaxespad=0)
#ax.axis("off")
plt.tight_layout()
plt.savefig('Experiment␣CM.png')
plt.show()

import io
s = io.StringIO()
model.summary(print_fn=lambda x: s.write(x + '\n'))
model_summary = s.getvalue()
s.close()

# Open a file with access mode 'a'
file_object = open(writePath, 'a')
file_object.write(filepath)
file_object.write("\n")
file_object.write(model_summary)
file_object.write("\n")
file_object.write(calc_loss)
file_object.write("\n")
file_object.write(calc_acc)
file_object.write("\n")
file_object.write('\n\nClassification␣Report\n\n{}\n\nConfusion␣Matrix\n\n{}\n'.
    ↪ format(cr, cm))
file_object.write('\nFP:␣{}\nFN:␣{}\nTP:␣{}\nTN:␣{}\n'.format(FP, FN, TP, TN))
file_object.write('\nFPR:␣{}\nFNR:␣{}\nTPR:␣{}\nTNR:␣{}\n'.format(FPR, FNR, TPR,
    ↪ TNR))
file_object.write('\nNPV:␣{}\nFDR:␣{}\nACC:␣{}\n'.format(NPV, FDR, ACC))
file_object.write("\n\n-----------------------------------------------\n\n")
#dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])

# Close the file
file_object.close()

#ROC plot
plot_roc_curve(y_test.argmax(axis=1), y_pred.argmax(axis=1))


print("END")
```

Bibliography

[1] N. I. of Standards and Technology. (2011). 'The nist definition of cloud computing.' visited on 2020-09-26, [Online]. Available: https://www.nist.gov/publications/nist-definition-cloud-computing.

[2] O. Mämmelä, J. Hiltunen, J. Suomalainen, K. Ahola, P. Mannersalo and J. Vehkaperä, 'Towards micro-segmentation in 5g network security,' Jun. 2016.

[3] C. O. T. E. COMMUNITIES. (2006). 'Proposal for a directive of the council on the identification and designation of european critical infrastructure and the assessment of the need to improve their protection ('eci directive').' visited on 2020-09-26, [Online]. Available: https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=COM:2006:0787:FIN:EN:HTML.

[4] J. Opara-Martins, R. Sahandi and F. Tian, 'Critical review of vendor lock-in and its impact on adoption of cloud computing,' in *International Conference on Information Society (i-Society 2014)*, 2014, pp. 92–97. DOI: 10.1109/i-Society.2014.7009018.

[5] Y. Jadeja and K. Modi, 'Cloud computing - concepts, architecture and challenges,' in *2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET)*, 2012, pp. 877–880. DOI: 10.1109/ICCEET.2012.6203873.

[6] P. M. Mell and T. Grance, 'The nist definition of cloud computing,' Gaithersburg, MD, Tech. Rep., 2011. DOI: 10.6028/NIST.SP.800-145. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-145.

[7] W. Stallings, *Computer security : Principles and practice*, eng, Upper Saddle River, 2018.

[8] KVM, *Main page — kvm,* [Online; accessed 26-May-2022], 2016. [Online]. Available: https://www.linux-kvm.org/index.php?title=Main_Page&oldid=173792.

[9] VMware. (2021). 'Master thesis.' visited on 2022-05-01, [Online]. Available: https://customerconnect.vmware.com/en/downloads/info/slug/datacenter_cloud_infrastructure/vmware_vsphere_hypervisor_esxi/7_0.

[10] VMware. (2022). 'Master thesis.' visited on 2022-05-01, [Online]. Available: https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/.

[11] K. Keahey, M. Tsugawa, A. Matsunaga and J. Fortes, 'Sky computing,' *IEEE Internet Computing*, vol. 13, no. 5, pp. 43–51, 2009. DOI: 10.1109/MIC.2009.94.

[12] J. Su. (2017). 'Hybrid cloud vs. multi-cloud: What's the difference?' visited on 2020-09-26, [Online]. Available: https://www.bmc.com/blogs/hybrid-cloud-vs-multi-cloud-whats-the-difference/.

[13] R. Ré, R. M. Meloca, D. N. Roma, M. A. da Cruz Ismael and G. C. Silva, 'An empirical study for evaluating the performance of multi-cloud apis,' *Future Generation Computer Systems*, vol. 79, pp. 726–738, 2018, ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2017.09.003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17301802.

[14] G. J. Popek and R. P. Goldberg, 'Formal requirements for virtualizable third generation architectures,' *Communications of the ACM*, vol. 17, no. 7, pp. 412–421, 1974.

[15] P. Chaurasia, S. B. Nath, S. K. Addya and S. K. Ghosh, 'Automating the selection of container orchestrators for service deployment,' in *2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, IEEE, 2022, pp. 739–743.

[16] L. Helali and M. N. Omri, 'A survey of data center consolidation in cloud computing systems,' *Computer Science Review*, vol. 39, p. 100 366, 2021, ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2021.100366. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S157401372100006X.

[17] vmware docs. (2021). 'Vsphere distributed switch architecture.' visited on 2022-03-29, [Online]. Available: https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.networking.doc/GUID-B15C6A13-797E-4BCB-B9D9-5CBC5A60C3A6.html.

[18] L. Miller and J. Soto, *Microsegmentation for Dummies*. John Wiley & Sons, 2015.

[19] B. Dauti, *CCENT/CCNA: ICND1 100-105 Certification Guide: Learn computer network essentials and enhance your networking skills by obtaining the CCENT certification*. Packt Publishing Ltd, 2018.

[20] E. Bell, A. Smith, P. Langille, A. Rijhsinghani and K. McCloghrie, 'Definitions of managed objects for bridges with traffic classes, multicast filtering and virtual lan extensions,' RFC Editor, RFC 2674, Aug. 1999.

[21] E. Rosen and Y. Rekhter, 'Bgp/mpls ip virtual private networks (vpns),' RFC Editor, RFC 4364, Feb. 2006.

[22] W. Weber, 'Firewall basics,' in *4th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services. TELSIKS'99 (Cat. No.99EX365)*, vol. 1, 1999, 300–305 vol.1.

[23] M. G. Gouda and A. X. Liu, 'Structured firewall design,' *Computer Networks*, vol. 51, no. 4, pp. 1106–1120, 2007, ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2006.06.015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128606001988.

[24] D. Huang, A. Chowdhary and S. Pisharody, 'Microsegmentation: From theory to practice,' in. Dec. 2018, pp. 155–180, ISBN: 9781351210768. DOI: 10.1201/9781351210768-8.

[25] V. N. M.-s. D. 1, *Wade Holmes*. vmware PRESS, 2017.

[26] E. S. Sosa, *Mastering VMware NSX for VSphere*. John Wiley & Sons, 2020.

[27] Z. Ma, Y. Yang and Y. Wang, 'A security policy description language for distributed policy selfmanagement,' in *2015 4th International Conference on Mechatronics, Materials, Chemistry and Computer Engineering*, Atlantis Press, 2015.

[28] V. N. M.-s. D. 2, *Geoff Wilmington*. vmware PRESS, 2017.

[29] V. Docs. (2020). 'Vrealize automation nsx-v to nsx-t (nsx v2t) migration.' visited on 2022-04-23, [Online]. Available: https://docs.vmware.com/en/vRealize-Automation/8.6/vrealize-automation-v2tmigration/GUID-0D1B5D79-66B6-4732-9E04-E6DBD74C9DB5.html.

[30] CITOPUS. (2020). 'Citopus vmware nsx automation and management.' visited on 2022-04-23, [Online]. Available: https://citopus.com/.

[31] C. Francois, *Deep learning with python*, 2017.

[32] C. Zhang and Y. Lu, 'Study on artificial intelligence: The state of the art and future prospects,' *Journal of Industrial Information Integration*, vol. 23, p. 100 224, 2021, ISSN: 2452-414X. DOI: https://doi.org/10.1016/j.jii.2021.100224. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2452414X21000248.

[33] J. McCarthy, M. Minsky, N. Rochester and C. Shannon, 'Artificial intelligence (ai) coined at dartmouth,' *Retrieved October*, vol. 28, p. 2021, 1956.

[34] A. Gulli and S. Pal, *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[35] A. Jain, J. Mao and K. Mohiuddin, 'Artificial neural networks: A tutorial,' *Computer*, vol. 29, no. 3, pp. 31–44, 1996. DOI: 10.1109/2.485891.

[36] A. L. Samuel, 'Some studies in machine learning using the game of checkers,' *IBM Journal of research and development*, vol. 44, no. 1.2, pp. 206–226, 2000.

[37] Z.-H. Zhou, *Machine learning*. eng, Singapore, 2021.

[38] N. I. of Standards and Technology. (2021). 'Cats vs dogs classification.' visited on 2022-03-20, [Online]. Available: https://data-flair.training/blogs/cats-dogs-classification-deep-learning-project-beginners/.

[39] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.", 2019.

[40] H. B. Braiek and F. Khomh, 'On testing machine learning programs,' *Journal of Systems and Software*, vol. 164, p. 110 542, 2020, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2020.110542. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121220300248.

[41] S. Baek, D. Kwon, J. Kim, S. C. Suh, H. Kim and I. Kim, 'Unsupervised labeling for supervised anomaly detection in enterprise and cloud networks,' in *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, 2017, pp. 205–210. DOI: 10.1109/CSCloud.2017.26.

[42] S.-C. Wang, 'Artificial neural network,' in *Interdisciplinary computing in java programming*, Springer, 2003, pp. 81–100.

[43] K. B. Prakash, *Advanced deep learning for engineers and scientists : A practical approach.* eng, Cham, 2021.

[44] V. A. Profillidis and G. N. Botzoris, *Modeling of transport demand: Analyzing, calculating, and forecasting transport demand*. Elsevier, 2018.

[45] S. G. Reid. (2014). '10 misconceptions about neural networks.' visited on 2022-03-20, [Online]. Available: http://www.turingfinance.com/misconceptions-about-neural-networks/.

[46] K. M. Ting, 'Confusion matrix,' in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 209–209, ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_157. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_157.

[47] scikit-learn. (2020). 'Classification report.' visited on 2022-04-20, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html.

[48] E. Ucar and E. Ozhan, 'The analysis of firewall policy through machine learning and data mining,' *Wireless Personal Communications*, vol. 96, no. 2, pp. 2891–2909, 2017.

[49] H. Gan, N. Sang, R. Huang, X. Tong and Z. Dan, 'Using clustering analysis to improve semi-supervised classification,' *Neurocomputing*, vol. 101, pp. 290–298, 2013, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2012.08.020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231212006601.

[50] B. J. Oates, M. Griffiths and R. McLean, *Researching information systems and computing*. Sage, 2022.

[51] H. El-Amir and M. Hamdy, *Deep Learning Pipeline: Building a Deep Learning Model with TensorFlow*, eng, 1st ed. Berkeley, CA: Apress L. P, 2019, ISBN: 9781484253489.

[52] G. Jekese, S. Ramasamy and C. Hwata, 'Virtual firewall security on virtual machines in cloud environment,' *International Journal of Scientific and Engineering Research*, Feb. 2015.

[53] R. Chandramouli, 'Secure virtual network configuration for virtual machine (vm) protection,' Tech. Rep., 2016. DOI: 10.6028/NIST.SP.800-125B. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-125B.

[54] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese and D. Walker, 'P4: Programming protocol-independent packet processors,' *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014, ISSN: 0146-4833. DOI: 10.1145/2656877.2656890. [Online]. Available: https://doi.org/10.1145/2656877.2656890.

[55] J. Li, H. Jiang, W. Jiang, J. Wu and W. Du, 'Sdn-based stateful firewall for cloud,' in *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, 2020, pp. 157–161. DOI: 10.1109/BigDataSecurity-HPSC-IDS49724.2020.00037.

[56] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, A. Al-Nemrat and S. Venkatraman, 'Deep learning approach for intelligent intrusion detection system,' *IEEE Access*, vol. 7, pp. 41 525–41 550, 2019.

[57] vmware docs. (2021). 'Vsphere distributed switch architecture.' visited on 2022-03-29, [Online]. Available: https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.networking.doc/GUID-B15C6A13-797E-4BCB-B9D9-5CBC5A60C3A6.html.

[58] N. Ketkar, 'Introduction to keras,' in *Deep Learning with Python: A Hands-on Introduction*. Berkeley, CA: Apress, 2017, pp. 97–111, ISBN: 978-1-4842-2766-4. DOI: 10.1007/978-1-4842-2766-4_7. [Online]. Available: https://doi.org/10.1007/978-1-4842-2766-4_7.

[59] E. B. Claise. (2004). 'Network working group.' visited on 2022-03-27, [Online]. Available: https://www.ietf.org/rfc/rfc3954.txt.

[60] FreeBSD. (2005). 'Freebsd manual pages - nfdump.' visited on 2022-03-27, [Online]. Available: https://www.freebsd.org/cgi/man.cgi?query=nfcapd&apropos=0&sektion=1&manpath=FreeBSD+8.2-RELEASE+and+Ports&format=html.

[61] FreeBSD. (2004). 'Freebsd manual pages - nfdump.' visited on 2022-03-27, [Online]. Available: https://www.freebsd.org/cgi/man.cgi?query=nfdump&apropos=0&sektion=1&manpath=FreeBSD+8.2-RELEASE+and+Ports&format=html.

[62] J. Su. (2020). 'Inet$_a ton()function in mysql$.' visited on 2022-04-24, [Online]. Available: https://www.geeksforgeeks.org/inet_aton-function-in-mysql/.

[63] V. Ravi. (2018). 'Network-intrusion-detection.' visited on 2022-03-29, [Online]. Available: https://github.com/vinayakumarr/Network-Intrusion-Detection/tree/master/KDDCup%5C%2099/dnn/multiclass.

[64] R. Hecht-Nielsen, 'Kolmogorov's mapping neural network existence theorem,' in *Proceedings of the international conference on Neural Networks*, IEEE Press New York, NY, USA, vol. 3, 1987, pp. 11–14.

[65] P. Jeff Heaton. (2017). 'Heaton research - the number of hidden layers.' visited on 2022-05-01, [Online]. Available: https://www.heatonresearch.com/2017/06/01/hidden-layers.html.

[66] scikit-learn. (2020). 'Sklearn.preprocessing.standardscaler.' visited on 2022-03-29, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

[67] scikit-learn. (2020). 'Sklearn.preprocessing.onehotencoder.' visited on 2022-03-29, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html#sklearn.preprocessing.OneHotEncoder.

[68] scikit-learn. (2020). 'Sklearn.model$_s election.train_t est_s plit$.' visited on 2022-03-29, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html#sklearn.model_selection.train_test_split.

[69] scikit-learn. (2020). 'Glossary of common terms and api elements.' visited on 2022-03-29, [Online]. Available: https://scikit-learn.org/stable/glossary.html#term-random_state.

[70] R. Dunford, Q. Su and E. Tamang, 'The pareto principle,' 2014.

[71] Keras. (2020). 'The model class.' visited on 2022-04-15, [Online]. Available: https://keras.io/api/models/model/.

[72] Keras. (2020). 'Probabilistic metrics.' visited on 2022-04-25, [Online]. Available: https://keras.io/api/metrics/probabilistic_metrics/#categoricalcrossentropy-class.

[73] Keras. (2020). 'Adam.' visited on 2022-04-25, [Online]. Available: https://keras.io/api/optimizers/adam/.

[74] D. Masters and C. Luschi, 'Revisiting small batch training for deep neural networks,' *arXiv preprint arXiv:1804.07612*, 2018.

[75] L. G. Thingnes. (2022). 'Master thesis.' visited on 2022-05-22, [Online]. Available: `https://github.com/larsgthi/Master-thesis`.

[76] S. Budd, E. C. Robinson and B. Kainz, 'A survey on active learning and human-in-the-loop deep learning for medical image analysis,' *Medical Image Analysis*, vol. 71, p. 102 062, 2021.