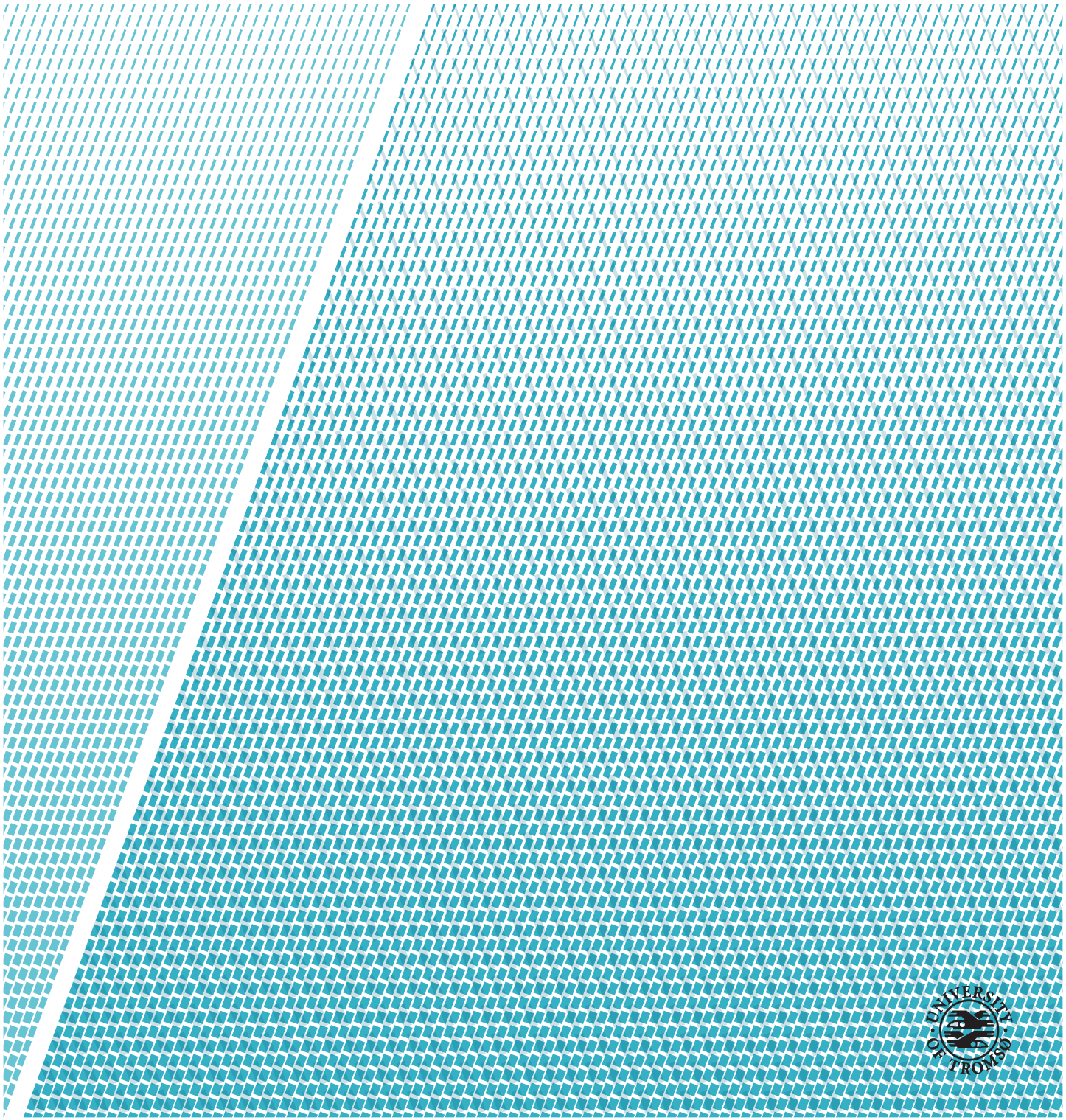
UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

Department of Computer Science and Computational Engineering

Detecting attacks on servers using machine learning models

Hans Victor Andersson Lindbäck

Master Thesis in Applied Computer Science



Abstract

With today's storage of information moving from paper to physical hard drives and the cloud, safety of these new information platforms are of great importance. Today an average server suffers several instances of abuse weekly, or even daily. The purpose of this project is to design a system for detecting abusive data traffic coming to or from a server by using machine learning algorithms. Also of importance is the new GDPR guidelines and how they affect the future development of data usage in AI. This project is aiming to use data already gathered by industry to check performance of their networks and systems. That is why a metrics based system using sequential data to see patterns in the network flow is investigated. The project is a combined effort between UiT Narvik and Arctic Circle Data Center, hereby called ACDC, where the data is provided by ACDC and the development is done by UiT. Included in this thesis is: a review of today's threat profile and how this effects industry, a review of today's research into anomaly detection using machine learning, a risk evaluation of the project and a review of the different attack data sets viable for machine learning on this topic. It concludes with a recommendation for the best models and data sets for an anomaly detection tool. The thesis includes an in depth explanation of the relevant theory and machine learning models as well as a simplified review of the different anomaly types.

Acknowledgements

I would like to thank my advisor, Bernt Arild Bremdal, who took the time to support and advise me throughout the course of this project.

I would also like to thank Arne Handal and Øyvind Bakksjø at Arctic Circle Data Center. Without them this project would not have been possible.

My classmates also deserve a great amount of thanks, not only for the times we have shared during this thesis, but for the support they have given me throughout my time at UiT - Narvik. I would not have made it through without you.

I thank my parents for supporting me and always being the place I can turn to when things get difficult. A short sentence in a master thesis is not enough to describe how much I appreciate you or how much you mean to me.

Finally, I would like to thank my girlfriend Aina Fosli who has made me the happiest man on earth. I can't wait to start the next chapter of our life together, in a new place.

Contents

Abstract i

Acknowledgements iii

List of Figures vii

List of Tables ix

1 Introduction 1
- 1.1 Task given by client - Arctic Circle Data Center . . . 3
- 1.2 Situation today - State of art . . . 3
- 1.3 Project Specification . . . 4
  - 1.3.1 Risks connected to project . . . 6

2 Evaluation of relevant data sets 7
- 2.1 KDD Cup 99 Intrusion Detection . . . 7
- 2.2 CAIDA "DDoS Attack 2007" . . . 8
- 2.3 DARPA 2009 . . . 8
- 2.4 CICIDS 2017 . . . 9
- 2.5 CTU-13 . . . 9
- 2.6 UNSW-NB15 . . . 10

3 Attack types in data set 13
- 3.1 Fuzzers . . . 13
- 3.2 Backdoors . . . 14
- 3.3 Denial of Service - DoS . . . 14
- 3.4 Exploits . . . 14
- 3.5 Generic attacks . . . 15
- 3.6 Reconnaissance . . . 15
- 3.7 Shellcode . . . 15
- 3.8 Worms . . . 16

4 Relevant background theory and research 17
- 4.1 Recurrent Neural Networks - RNN . . . 17

4.2 Long Short Term Memory RNN's - LSTM 18
4.3 Attention mechanisms 19
4.4 Fully Convolutional Networks - FCN 19
4.5 Squeeze and Excite blocks 21

5 Relevant machine learning models 25
5.1 HIVE-COTE 25
5.2 Residual Networks 26
5.3 LSTM-FCN, ALSTM-FCN and MALSTM-FCN 27

6 Methods 29
6.1 MALSTM-FCN 29
6.2 Processing of UNSW-NB15 data set 30
6.3 Process of running the program 32

7 Results 33

8 Discussion 45

9 Conclusion 49

10 Future Work 51

Bibliography 53

A Code and data sets related to project 57

List of Figures

1.1 Rudimentary points of access for an attacker in a cloud setting 2

2.1 Setup for capture of the UNSW-NB15 data set. Picture from Moustafa and Slay (2015) 11

4.1 Example of semantic segmentation on images. Picture from Heinrich (2018) 20
4.2 Squeeze and excite block between layers. Picture from titu1994 (2018) 23

5.1 Skip connection in ResNets. Picture from Tsang (2018) . . . 26
5.2 Different ResNet implementations. Picture from Tsang (2018) 27
5.3 Example of LSTM-FCN implementation used in my project. Picture from titu1994 (2018) 28

6.1 Operations done to the original training and test sets from the UNSW-NB15 dataset. 31

7.1 Results of simulations using different LSTM-FCN implementations 35
7.2 Results of simulations using different batch sizes in the MALSTM-FCN model 37
7.3 Results of MALSTM-FCN simulations using different amount and types of keys in the data set 40
7.4 Pictures of MALSTM-FCN on the 10 key set with different epoch values 42

List of Tables

7.1 Results of a time comparison between the time needed to run the script and the 72 hours of data in the UNSW-NB15 extended data set. 43
7.2 Final accuracy results for each key set with optimized epoch number 44

1

Introduction

Digitalization is one of the biggest buzzwords in industry today and as companies move their data from paper to the cloud, new safety issues arise. These new safety issues are to be handled not only by the companies themselves, but also by the providers of such cloud services. Server companies need to be able to handle security of not only data, but also the safety of network traffic and computing resources on server space that the customers are renting. As seen in Alert Logic (2017), the number of attacks on privately hosted and hybrid based cloud services are the most common. For hosted clouds the average number of attacks were 684 over an 18 month period. The results are even worse for hybrid solutions where there were 977 in the same time span.

Losing data, data access or data privacy is expensive. In 2015 according to Armin et al. (2015) the cost of cyber crime in the EU was 0.4% of the GDP. This equals a sum of 13 billion euros per year and the numbers were only expected to rise. Germany alone had a loss of 2.6 billion euros a year. The annual revenue from cyber criminality is estimated to be around 15 billion euros per year and the market for cyber security products and services is 50 billion euros and rising. It was also the second most reported crime in 2016. These sums represent how widespread the effects of cyber crime is and the numbers have only risen since 2015.

Securing data has come a long way in the last 20 years but the security of computing power and network traffic is lagging behind. This can be seen in Aruna et al. (2013) where most, if not all, risks listed are tied to network traffic

or server resources and not data saved on location or site. As shown in Fig. 1.1 there are more entry points in a cloud setting than for local data storage.

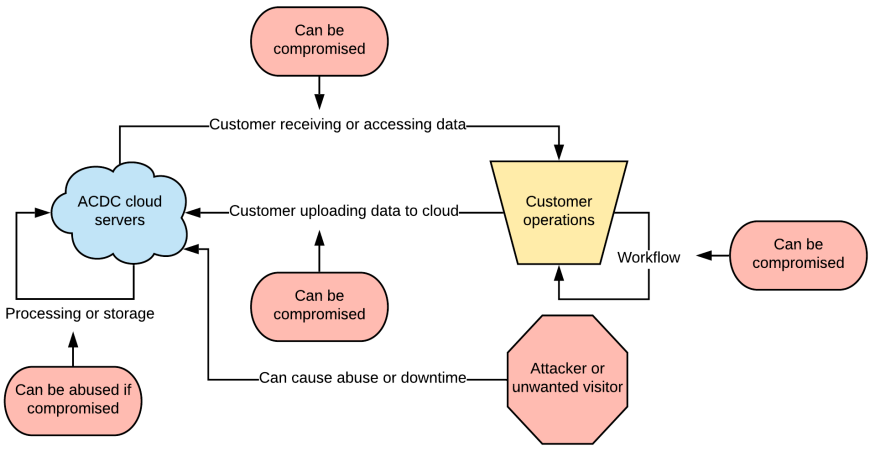


Figure 1.1: Rudimentary points of access for an attacker in a cloud setting

The discovery rate of cyber crime is also worth noting. While more physical crimes are often discovered within minutes, hours or days, cyber attacks are on average detected about 6 months after the attack is initiated. That gives hackers a 6 months head start on cyber security experts and thereby make the offenders very hard to locate and prosecute.

This report proposes a solution to some of the problems mentioned above. I propose the creation of a system based on machine learning, metrics and continuous learning to be implemented on cloud servers with low impact on their computational resources. Using a MALSTM-FCN implementation trained on parts of the UNSW-NB15 data set, I managed to achieve an accuracy of 89%. The script is low impact on memory strained systems and uses a minimal amount of computational resources compared to other machine learning models. It requires a nominal amount of preprocessing the data and is able to handle data streams of unstable quality.

This all resulted in a metrics based program with a focus on cyber security, and privacy, which is easy to deploy on most platforms.

1.1 Task given by client - Arctic Circle Data Center

This thesis's task, agreed upon by UiT Narvik and Arctic Circle Data Center (ACDC), was to detect network traffic or computing resource anomalies in ACDC's virtual machines using machine learning algorithms. These results would then be used to create a rudimentary prototype for ACDC. The success criteria was the ability to detect different types of anomalies or unwanted traffic like:

- Attacks from the outside
- Attacks generated from within (participation in unwanted traffic or attacks)
- Altered traffic patterns from customers

Thereafter demonstrate a possibility to integrate this solution into existing ACDC platforms. ACDC were responsible for delivering the data needed for training and testing the machine learning implementation.

ACDC are currently using a system based on OpenStack running on Ubuntu Linux. They measure general volume of network traffic by exporting their traffic metrics, using SNMP, to a Prometheus database. The data is then visualized in a Grafana front end to be viewed by personnel if needed. This gives them an overview of bytes received. Currently this is done only on the host level not at a per virtual machine (VM) basis. They are now working to implement metrics per virtual machine into their systems.

1.2 Situation today - State of art

Today's development in the use of machine learning to detect attacks on both servers and personal computers is large and growing. Currently there are several major research sites and hundreds of articles showing successful results in this field. For example, in He et al. (2017) they managed to acquire a success rate of 99.7% in detecting four different types of DDOS attacks. This was done by using characteristics of the packet flow in the system. They analyzed the amount of Diffie-Helmann key exchanges, the ratio of in and out going packages and the ratio between SYN and ACK messages sent. Diffie-Hellman key exchange is a method for transferring cryptographic keys safely between parties connected over public channels. It is widely used to secure various internet services today. By counting the amount of these exchanges, one can see how many connections have been made between a server and it's

clients.

Najafabadi et al. (2014) also had successful results in using machine learning for detecting attacks. They specialized in brute force detection and used a different methodology to the one used by He et al. (2017). They utilized several attributes of the network like source port, connection duration and more to expose attacks. From these attributes they made a decision tree with several layers. This gave them a very high success and low false alarm rate.

Similar results and research can be shown in Kirubavathi and Anitha (2016) and many more. However these are all very specified solutions with a hard coded program for every type of attack. If you implement one of these you are still completely vulnerable for other types of attacks like mining or any type of malware on the machine. Not to mention the human work needed to implement similar solutions for every known type of attack. There are currently no software available that checks for all known vulnerabilities using the above methods, on the market today.

Lastly the big problem with these solutions are that they are indiscriminate about data. In the last years there have been a greater focus on data privacy with the EU General Data Protection Regulation or GDPR being one of the biggest changes for data scientists working in the field today. This means that the solutions referenced might not be allowed to access and utilize the data they need in the future. Several companies have already been punished for breaching these regulations. Especially solutions used in Najafabadi et al. (2014) for discovering brute force attacks can be privacy invading given that it records with whom and for how long the user or attacker is connected.

This brings us to solutions that avoid using what might in the future be protected data. In Mendoza and Bedford (2018) the MITRE corporation released their research into metrics driven attack detection with dynamic thresholds. While many metrics based protection systems use a static value when it detects anomalies, the MITRE solution uses forecasting to create a dynamic threshold. This is then used as the new limit and any action outside the expected parameters are handled as anomalies. This is a good solution when it comes to privacy, but it currently has no further use other than detection of unexpected behavior in the network metrics.

1.3 Project Specification

After seeing the types of research that has been done in the field of network and server protection using many different tools and algorithms, I have got

a general idea of what would be a good implementation for the client ACDC. Since they are currently in the process of establishing data for per VM metrics and they are today not using any anomaly detection software, any minor result is better than what they have. It is also easier for them to implement a metrics based detection system into their current platform since they have experience working with such data.

The development of current machine learning based detection systems have mostly been indiscriminate in terms of data and user privacy has not been a priority. From a business perspective, the lack of needing approval from each customer to use the system is a bonus, as in the case of GDPR, many businesses are still coping with the changes. As such I recommend using a future proof, not customer reliant and easily integrate-able solution. This is done by using a metrics based platform as mentioned in Mendoza and Bedford (2018).

Their solution has future work that can be implemented to improve the performance of their detection systems. One such thing is the time frame used. Mendoza and Bedford (2018) collected data with 20 second time intervals and averaged this into one hour time aggregates. By making the system more efficient one might be able to make this more detailed and thereby be able to detect more fine tuned anomalies. This can be done by picking different algorithms for the time forecasting and detection algorithm, reducing complexity of the data set and/or increasing computational power.

Another part of their project that can be improved is the ability to run multiple sequential data processes at the same time and use them in conjunction to classify or detect abuse. With a multivariate version of an LSTM-FCN model it is possible. This might be the most important feature that can be implemented into the functioning structure mentioned in Mendoza and Bedford (2018).

Several machine learning algorithms are used to make forecasts in sequential data today and many have good results. In Dietterich (2002) several suitable candidates are mentioned, included but not limited to; Long Short Term Memory or LSTM, Hidden Markov models, Gated recurring units or GRU, Neural Networks Auto Regression and CRF. These are just some of the candidates for the final implementation and testing which models give the best results will be a part of the work done in this project.

Thereby, the goal for the final implementation will be; a detection system using aspects of a VM's network metrics and resource usage to do a forecasting with the help of a suitable machine learning algorithm. This prediction should give dynamic parameters for what is the expected and abnormal behavior. It will hopefully be able to learn long and short term trends and take this into account when detecting attacks. The solution should be implementable on

ACDC's platform without extensive efforts.

1.3.1 Risks connected to project

The project specification is in its whole viable. There is a relatively unexplored market for this type of anomaly detection and research supports it's capabilities for this kind of scenario. It is also a solution which would be easy to integrate into ACDC's already existing information platform. However, this kind of idea has no proven record of matching the 99,7 percent accuracy in attack detection that has been proven using other methods. This however, it makes up for in data privacy and in being a future proof solution.

The biggest risks were mainly connected to collecting the training and test data needed to use the machine learning algorithms. ACDC did not currently possess the data but were working to extract them during the course of the project. If they were unable to extract the data in time, the solution could be tested on dummy data generated by other server applications, lowering the total risk factor.

Another risk is that while the solution will be able to detect easily distinguishable patterns like with a brute force or DDOS attack, it might struggle with more fine tuned attacks like detecting mining or similar actions.

2

Evaluation of relevant data sets

With reference to 1.3.1 - Risks connected to project, ACDC were unable to provide detailed attack or anomaly data in time for this project. Therefore a data set of attack patterns was required. This set needed to be relevant for the user case of ACDC and also be of a certain size to be able to support long term anomaly detection.

Most importantly, the data set needs to represent real life network traffic and not be overly synthetic, as many such sets have a tendency to be. There are many sets available, all with their own strengths and weaknesses.

2.1 KDD Cup 99 Intrusion Detection

This is the data set used for The Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99, The Fifth International Conference on Knowledge Discovery and Data Mining. The KDD 1999 intrusion data set was the industry standard for attack detection when it was released and for many subsequent years.

The data set has been used and analyzed by a multiple of researchers. This has

resulted in a good understanding of the different attributes in the set. Many classify it as overly optimistic giving better results in attack detection than if used for a real life setting. This could be mitigated by mixing it with real life data, however according to Protic (2018) the data set will never represent attacks accurately since the attacks were done over a virtual network.

It is also known for not being completely representative of today's attack patterns. Much have changed in the cyber crime industry since 1999 and the KDD data set just no longer represents attacks accurately. This is a problem that can not be overcome and therefore makes this data set a poor choice for my project.

2.2 CAIDA "DDoS Attack 2007"

This data set was captured by CAIDA (Center for Applied Internet Data Analysis) and contains approximately one hour of anonymous traffic traces from a DDoS attack made in 2007. The type of denial-of-service attack used here attempts to block access to the targeted server by consuming computing resources and using all bandwidth of the network connecting server to Internet.

The data set is often used to detect DDOS attacks in a short time span and it is one of the most widely used data sets for DDOS detection. It gives an accurate depiction of today's DDOS attack patterns and is representative of a real attack. The problem is that since the data set only spans one hour it does not fit what the task specification requires; long term attack detection. This data set will not test seasonality, trends or other attributes.

It is also locked behind IMPACT (Information Marketplace for Policy and Analysis of Cyber-risk and Trust) and they did not grant access to this data set for my project.

2.3 DARPA 2009

The DARPA 2009 data set was captured by Colorado State University and created to aid in the evaluation of network intrusion detectors. The data set lasts for a period of 10 days. It comprises of synthetic HTTP, SMTP, and DNS background traffic merged with real life traffic between a /16 local subnet and the Internet. The data set consists of a variety of security events and attack types, including denial of service attacks and worms.

The data set is used by industry today for intrusion detection training. It shares some of the issues with the KDD data set since it is synthetic but it handles the problem better by introducing realistic traffic to the simulation. The data set has a size of 6.5 terabyte giving it more than enough size to fit the criteria of long term attack detection.

However, the problem with such a huge data set is that it's not easy to process and has a lot of useless data inside. The creators themselves acknowledge this and has tried to split the data set up into smaller pieces relevant for different attack types, but they have not completed the segmentation.

The data set is also distributed by IMPACT and they did not give access to the data set. It is also not preprocessed for machine learning and it would take extensive efforts to utilize it. Hence, the DARPA 2009 data set is not suitable for use in this project.

2.4 CICIDS 2017

The CICIDS 2017 dataset, Sharafaldin et al. (2018), was compiled by the University of New Brunswick and contains benign traffic and up-to-date attacks, both of which resembles real-world data. It also includes the results of a network traffic analysis with labeled flows based on the time stamp, source and destination IPs, and source and destination ports. The set was captured over 5 days and was heavily focused on creating realistic background traffic. It includes seven varied attack types.

The set is labeled and suitable for machine learning. It reproduces real life scenarios well and is of an acceptable size. Since the set is relatively new it has not been as extensively researched as the others, but it has been created using techniques resulting from modern set analysis.

This set is well suited for this project but is not a perfect fit.

2.5 CTU-13

The CTU-13, García et al. (2014), is a data set of botnet traffic that was captured in the CTU University of the Czech Republic, in 2011. The goal of this data set was to have a large capture of real botnet traffic mixed with normal and background traffic. The CTU-13 data set consists of thirteen captures (called scenarios) of different botnet samples.

The distinctive characteristic of the CTU-13 data set is that it's manually analyzed and have labeled each scenario. The fact that botnets are one of the major issues facing the server hosting industry today makes this set very relevant. Data from this set is comparable to a real life scenario and could be used for machine learning.

However, it lacks certain keys are needed to perform well in a metrics based program. If the final solution of this project were to be indiscriminate about data, then this set would be a good fit for learning to detect botnets.

2.6 UNSW-NB15

The raw network packets of the UNSW-NB15 data set, Moustafa and Slay (2015), was created by the IXIA PerfectStorm tool in the Cyber Range Lab of the Australian Centre for Cyber Security (ACCS), for generating a hybrid of real modern normal activities and synthetic attack behaviors.

The Tcpdump tool was utilised to capture 100 GB of raw traffic. The setup for this capture can be seen in Fig. 2.1 The data set has nine types of attacks, namely, Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms. They have used twelve algorithms to generate totally 49 features with class labels. This makes it a complex set that is easy to understand and preprocess for machine learning.

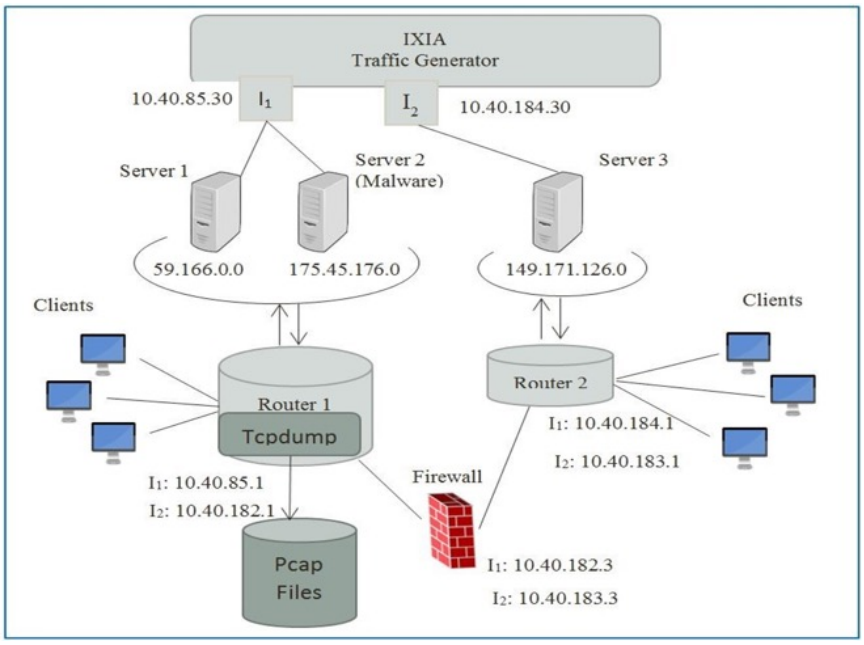


Figure 2.1: Setup for capture of the UNSW-NB15 data set. Picture from Moustafa and Slay (2015)

The creators also produced training and testing data sets ready for machine learning. These sets include all the attack types that are available in the original set. Number of records in the training set is 175,341 and the test set consists of 82,332 readings.

Since the set has many of the most used attack types that server providers suffer from, it is ideal for my project. It also reproduces real life activities with it's mix of synthetic and real life, "normal", behavior. It is a relatively new set compared to others available and is of a good size for long term analysis.

UNSW-NB15 was chosen as the sole data set for the project since it was a complex data set of a nice size, while still managing to remain simple to work with. It is also better documented than the others with multiple pages describing the information inside and it's different uses. Since ready made training and testing data sets were available, this is the best candidate for a project with a short time frame such as this.

The UNSW-NB15 will hopefully be able to make up for the lack of data provided by the client and have a similarity that still gives it relevancy to the final product. It will also give an insight into how the algorithm chosen for the project will be able to handle the speed of close to real time data. This can be tested by

using any number between 4 and 49 of the data labels provided to test the efficiency.

3

Attack types in data set

The UNSW-NB15 data set contains in total nine attack types. These all have different methods used to cause harm and effect the network flow differently.

3.1 Fuzzers

Fuzz testing is a technique used to uncover coding errors and security loopholes in operating systems, software or networks. It involves inputting huge amounts of randomized data, called fuzz, into the test subject in an attempt to make it crash. If a vulnerability is found, tools called fuzzers can be used to identify the cause of such issues.

A fuzzer discovers vulnerabilities that can be exploited by buffer overflow. Like in the case of; Denial of Service, cross-site scripting and SQL injections. These attacks are often used by malicious hackers intent on doing the most amount of damage in the least amount of time.

3.2 Backdoors

Backdoors is a method used by hackers to gain unauthorized access to a network, often from a remote location. Attackers use this to be able to gain repeated access to a network without being logged by a system administrator. This enables the hacker to be able to use the network, network resources or computing resources, without the knowledge of others.

3.3 Denial of Service - DoS

Denial of Service attacks aim to monopolize network or computing resources so that valid users cannot use them. There are several types of such attacks with the most used being flooding type attacks like SYN flood or ICMP flood. An ICMP flood attack, also called a ping flood, is a type of DoS where the attacker sends spoofed information packets that hit every computer connected to the network, thereby taking advantage of any wrongly configured devices.

A SYN flood is a type of DoS that use a vulnerability in the normal TCP sequence. The relevant sequence is the three way handshake made between a server and host. It works like this: The server target gets a request to begin the handshake process. However, in a SYN flood this handshake is never completed. This leaves the connected port occupied and thereby unavailable for further processes. This is repeated over and over by the attacker until all open ports are occupied and the server is unusable.

A DoS attack is different from a DDoS (Distributed Denial of Service) attack. A DoS attack is often done using one computer and one network while DDoS uses multiple computers and connections to flood the target. DDoS is often used for global attacks, distributed via botnets, and are generally more severe.

3.4 Exploits

A computer exploit is a type of attack where the attacker takes advantage of a particular known system vulnerability as an attack point. There are often many such vulnerabilities available for any type of system, especially any system which is not continually updated and suffers from having an overwhelming amount of known vulnerabilities. 90% of attacks on systems and computers today are using known vulnerabilities and among them, more than 90% have an existing safety fix waiting in a pending update. This is what many hackers use to gain access into systems today.

The threshold for such hacker attacks is also much lower than for other types since there are tools available and the techniques needed to use them are available with a simple Google search. If you use Nmap queries over an IP range to find the version of operating system on a computer for example, there are several sites which show the known vulnerabilities for this system version and how to abuse such exploits.

3.5 Generic attacks

Generic attacks focus on what can be called the "user" side of a system. Within this type are brute-force attacks that use dictionaries of words to basically test different passwords to access a user's account or gain access to a network. Another generic attack can be to try and insert code in text fields available to the user.

In our data set brute-force attacks were made. This is computationally expensive and very time consuming but can often be run quite secretively if certain tests are not run on the server side. These types of attacks would have been easier to detect if I had CPU usage metrics included in the data set.

3.6 Reconnaissance

A reconnaissance attack is where a malicious party tries to gain information about or within a target network. These reconnaissance scanners are used for unauthorized discovery and mapping of target networks and systems, including services or vulnerabilities.

Before any bigger attack can or should be launched, there is done reconnaissance of the target to be able to detect the best angle of attack. These are difficult to detect and do no damage, except uncover information. After this is done the attacker can decide for themselves what kind of attack angle has the highest chance of success on this type of system or network.

3.7 Shellcode

A shellcode is usually a small code script that is used as a payload in the exploitation of a software vulnerability. The name shellcode comes from the behaviour of this script, since it often opens a command shell from where the

attacker can do malicious activity or gain control of the system. However, any piece of code that has a similar function can be called a shellcode.

Since it's function as a payload is beyond that of just opening a command shell many argue that the name shellcode is insufficient. Shellcode is often written directly in machine code. There are two main types of shellcode, local and remote. A local shellcode is used when the attacker has limited access and has the opportunity to exploit a vulnerability. Remote shellcode is used to gain access into a network or computer from a distance.

3.8 Worms

A worm is malware that multiplies and spreads itself through a network of computers. A worm is self replicating without human interaction and does not need to be used as part of a computer program to do damage.

Worms are often transmitted via software vulnerabilities or via spam messages in email or IM (Instant Message) form. If a user opens these links or attachments a worm is downloaded and installed. Once installed the worm goes to work with silently infecting the machine without user's knowledge.

Worms often modify and delete files. They can even download or inject further malicious software onto a system or network. Some worms have the sole purpose of replicating and taking up so much resources that the system is unusable or the network is stuffed with traffic. In addition to these options it can steal data, install backdoor software and allow remote access to a malicious user or even remote control.

A relevant example of a very serious attack using a worms was a ransomware cryptoworm attack called WannaCry. It spread through a known exploit that had been released to the public a few months before. Ironically, the fix was already released by Microsoft in a patch prior to the attack. However, the worm spread through non-updated systems and managed to infect more than 200 000 computers across 150 countries. The damage caused by the worm was calculated to be in the hundreds of millions to billions of dollar range. Later the worm was traced back to North Korea, with the US, United Kingdom and Australia announcing it formally in December 2017.

4

Relevant background theory and research

4.1 Recurrent Neural Networks - RNN

Recurrent neural networks (RNN) is a class of artificial neural network where the connections between separate units form a directed graph along a sequence. In our case this displays temporal connections between layers. RNN's use their internal state to process input sequences and are applicable for the task of time series anomaly detection. According to Pascanu et al. (2013), RNN's have a hidden state h which is up to date at a time step t as shown in Eq. (4.1),

$$h_t = tanh(Wh_{t-1} + Ix_t) \tag{4.1}$$

where *tanh* is the hyperbolic tangent function, W is the weight matrix and I is the projection matrix. y_t is a prediction that can be calculated using the hidden state h and W as seen in Eq. (4.2),

$$y_t = softmax(Wh_{t-1}). \tag{4.2}$$

Further, by using softmax we create a normalized probability distribution of all classes utilizing a logistic sigmoid function σ. By using the hidden state h as an input to another RNN we can create deeper networks as shown in Eq. (4.3),

$$h_t^l = \sigma(Wh_{t-1}^l + Ih_t^{l-1}) \tag{4.3}$$

4.2 Long Short Term Memory RNN's - LSTM

Long Short Term Memory RNN's (LSTM) is another type of RNN. What makes this type different is that it deals with the problem of vanishing gradients. Vanishing gradients results in normal RNN's having only a short term memory when it comes to solving tasks. This is a bottleneck for the algorithm and hinders performance. LSTM solves this problem by using gating functions in their state mechanics. An LSTM has a hidden vector h and a memory vector m, which control the state updates and outputs at each time step. Graves (2012) depicts the computations done at each time step as seen in Eq. (4.4):

$$\begin{aligned} g_u &= \sigma(W_{t-1}^u + I^u x_t) \\ g_f &= \sigma(W_{t-1}^f + I^f x_t) \\ g_o &= \sigma(W_{t-1}^o + I^o x_t) \\ g_c &= \sigma(W_{t-1}^c + I^c x_t) \\ m_t &= g^f \odot m_{t-1} + g^u \odot g^c \\ h_t &= tanh(g^o \odot m_t \end{aligned} \tag{4.4}$$

where σ is the logistic sigmoid function and $\odot$ is element wise multiplication.

LSTM's are capable of learning temporal dependencies. Long term dependencies of long sequences however, can prove challenging for the algorithm. A solution to this problem has been proposed by Bahdanau et al. (2014) using attention mechanisms to learn these long term dependencies.

4.3 Attention mechanisms

Attention mechanisms aim to solve the problem of long term dependencies by using a context vector C on a target sequence y. It is a method commonly used on neural translation of documents. According to Bahdanau et al. (2014), the context vector c_i is dependable on the sequence of annotations $(h_1, ..., h_{T_x})$, where an encoder maps the input sequence. h_i is comprised of information from the whole input sequence. However, it focuses on the regions surrounding i. We compute the context vector by using the weighted sum of the annotations h_i as shown in Eq. (4.5):

$$c_i = \sum_{j=1}^{T_x} a_{ij} h_j \tag{4.5}$$

the weights a_{ij} are calculated as shown in Eq. (4.6):

$$a_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})} \tag{4.6}$$

where e_{ij} is $a(s_{i-1}, h_j)$. e_{ij} is the alignment model and measures how well the input at index j matches with the output at index i. This is done by using the hidden state of the RNN, s_{i1} and the annotation at position j, h_j, of the input sequence.

In Bahdanau et al. (2014) they used a feed forward network to establish the parameters of the alignment model, a. The network was trained with all the other components of the model. It also calculates a soft alignment which back propagates the gradient of the cost function.

4.4 Fully Convolutional Networks - FCN

A traditional Fully Convolutional Network (FCN) has a goal of creating "semantic segmentation". This is an output with the same size as the original input and with a rough likeness to the original. However, the semantic segmentation result, has the data from the input classified into predefined classes. In an image this would result in being able to classify objects contained within the scope as shown in Fig. 4.1

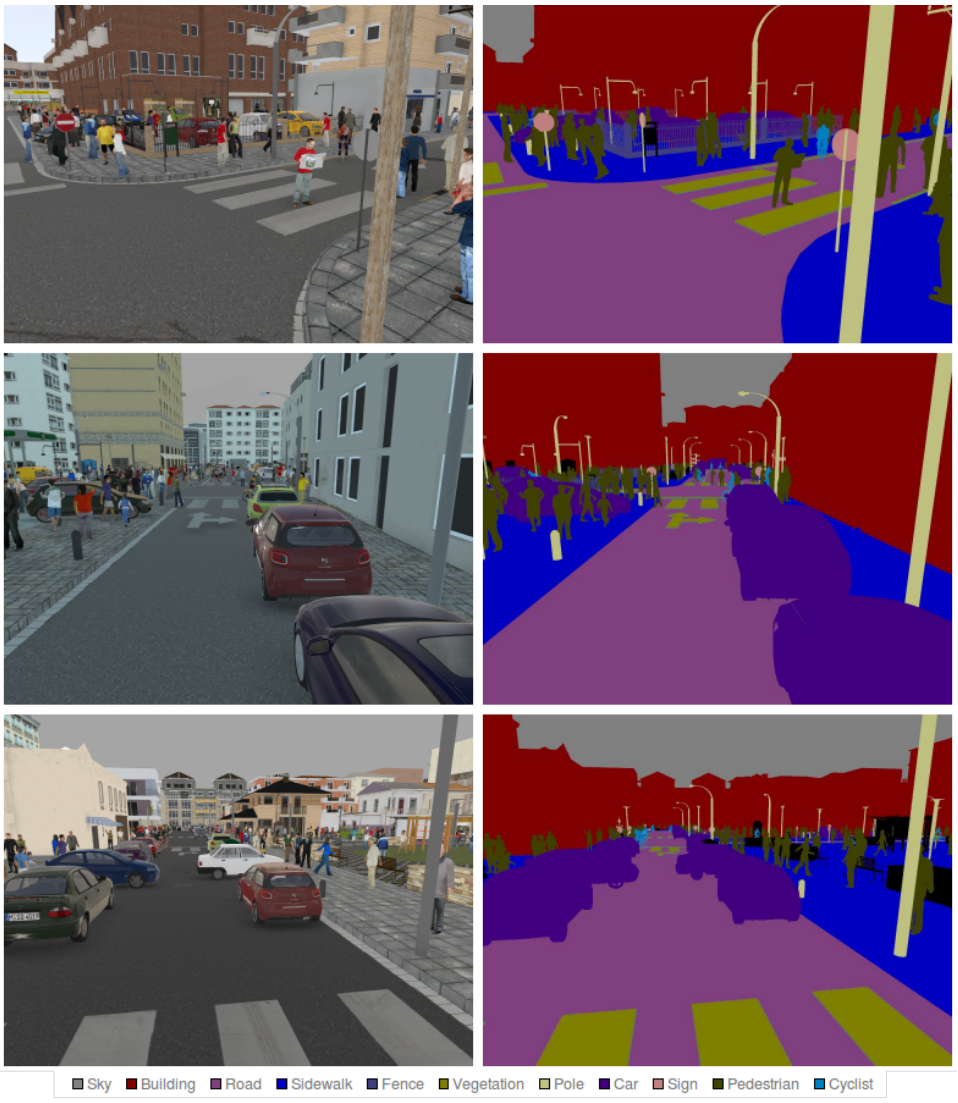


Figure 4.1: Example of semantic segmentation on images. Picture from Heinrich (2018)

In our case, Temporal Convolution Networks function as the feature extraction module of the FCN part. Normally, basic convolution blocks contain a convolution layer and a batch normalization. The batch normalization is then followed by the activation function. In this case the activation function is a Rectified Linear Unit (ReLU).

Temporal Convolution Networks often have an input type of time series. In Lea et al. (2016) they defined $X_t \in \mathbb{R}^{F^0}$ to be the input feature vector with length F^0 for time t. $t > 0 \leq T$, where T is number of time steps for a sequence. Every

frame has an action label, y_t where $y_t \in \{1, ..., C\}$ and C is number of classes. Every one of the L convolutional layers has a 1D filter applied to it. This is to make sure that the evolution of the input signals are captured over time.

Lea et al. (2016) also use a tensor $W^{(l)} \in \mathbb{R}^{F_t \times d \times F_{t-1}}$ and biases $b^{(l)} \in \mathbb{R}^{F^t}$ to give parameters to the 1D filter. The layer index is $l \in \{1, ..., L\}$ and the filter duration is d. The i-th component of the activation $\hat{E}_t^{(l)} \in \mathbb{R}^{F^t}$ for the l-th layers is a function of the incoming normalized activation matrix$E^{(l-1)} \in \mathbb{R}^{F^{t-1} \times T_{l-1}}$ from the previous layer, as shown in Eq. (4.7):

$$\hat{E}_{i,t}^{(l)} = f(b_i^{(l)} + \sum_{t'=1}^{d} \langle W_{i,t',.}^{(l)}, E_{.,t+d-t'}^{(l-1)} \rangle) \tag{4.7}$$

for each time t where $f(\dot{})$ is a ReLU.

4.5 Squeeze and Excite blocks

In Hu et al. (2017) they propose a *Squeeze-and-Excitation* block. This acts as a computational unit for any transformation of $\mathrm{F}_{tr} : \mathrm{X} \longrightarrow \mathrm{U}, \mathrm{X} \in \mathbb{R}^{W' \times H' \times C'}, \mathrm{U} \in \mathbb{R}^{W \times H \times C}$. Outputs of F_{tr} are represented by $U = [\mathrm{u}_1, \mathrm{u}_2, ..., \mathrm{u}_C,]$ where u_c is shown in Eq. (4.8) as:

$$\mathrm{u}_c = \mathrm{v}_c * \mathrm{X} = \sum_{s=1}^{C'} \mathrm{v}_c^s * \mathrm{x}^s \tag{4.8}$$

Convolution operations are depicted as $*$ and the 2D spatial kernel is v_c^s. A single channel v_c works on the corresponding channel X. They also model the channel-wise inter dependencies to adjust the filter responses in two ways, *Squeeze* and *Excitation*.

The *Squeeze* segment uses the contextual information on the outside of the direct input field, by taking advantage of a global average pool to create channel-wise statistics. The transformation output U, is condensed through spatial dimensions $W \times H$ to calculate the channel-wise statistics, $z \in \mathbb{R}^C$. The c-th element of z is computed as shown in Eq. (4.9):

$$z_c = F_{sq}(u_c) = \frac{1}{W \times H} \sum_{i=1}^{W} \sum_{j=1}^{H} u_c(i, j). \tag{4.9}$$

For temporal sequence data, U, the transformation output is condensed, or *squeezed*, by using the temporal dimension T to calculate the channel-wise statistics $z \in \mathbb{R}^C$. The c-th element of z is then computed as shown in Eq. (4.10):

$$z_c = F_{sq}(u_c) = \frac{1}{T} \sum_{t=1}^{T} u_c(t). \tag{4.10}$$

The aggregated information gained from the *Squeeze* segment is then followed by an *Excite* segment. The objective with this operation is to capture channel-wise dependencies. A gating mechanism with a sigmoid activation is used to achieve this, as shown in Eq. (4.11):

$$s = F_{ex}(z, W) = \sigma(g(z, W)) = \sigma(W_2 \delta(W_1 z)), \tag{4.11}$$

where δ is a ReLU, $W_1 \in \mathbb{R}^{\frac{C}{r} \times C}$ and $W_2 \in \mathbb{R}^{\frac{C}{r} \times C}$. W_1 and W_2 are used to limit the model complexity. W_1 are parameters of the dimensional reduction layer and W_2 are parameters of the dimensional increasing layer.

Then the output is scaled as shown in Eq. (4.12):

$$\tilde{x}_c = F_{scale}(u_c, s_c) = s_c \cdot u_c \tag{4.12}$$

where $\tilde{X} = [\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_C]$ and $F_{scale}(u_c, s_c)$ refers to channel-wise multiplication between feature map $u_c \in \mathbb{R}^T$ and the scale s_c.

In Fig. 4.2 we can see an example of how a *Squeeze and Excite* block can be implemented between convolutional layers.

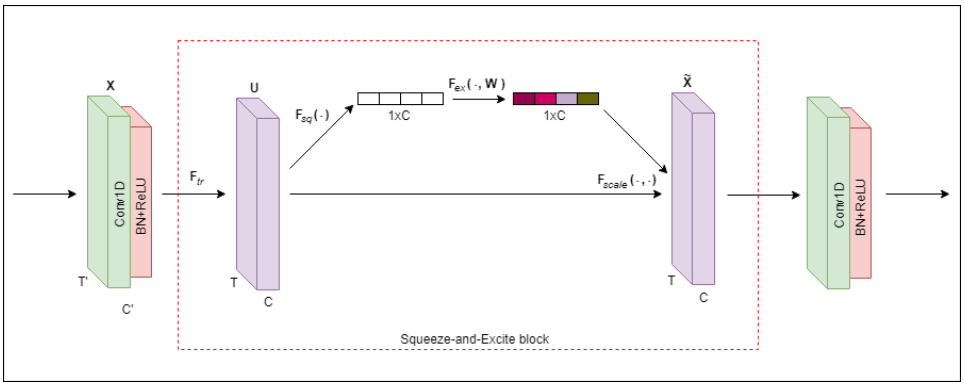


Figure 4.2: Squeeze and excite block between layers. Picture from titu1994 (2018)

5

Relevant machine learning models

5.1 HIVE-COTE

COTE (Collective Of Transformation-based Ensembles) is an ensemble of 35 classifiers that instead of being able to ensemble different classifiers over the same transformation, ensembles different classifiers over different time representations. Lines et al. (2018) extended COTE with a HIVE (Hierarchical Vote) system making the HIVE-COTE. This has shown to be a significant improvement to the original COTE by using a new hierarchical structure with a probability based voting, including two new classifiers and two additional representation transformation domains. HIVE-COTE is currently considered to be the state-of-art in time series classification when evaluated over the 85 datasets from the UCR/UEA archive.

However, to achieve the high accuracy HIVE-COTE is very computationally intensive and therefore impractical to use for real time problems. It requires 37 classifiers to be trained as well as cross validating each parameter of the algorithms described in Lines et al. (2018). In addition, the decisions made by the 37 classifiers are not easily interpreted even by domain experts making it a struggle to fully understand the decisions made by a single classifier. The HIVE-COTE also needs a very processed and stable data set to work upon, making real life implementations even harder to implement

These attributes in conjunction makes the approach unusable for our application.

5.2 Residual Networks

Residual Networks (ResNet) are a type of Convolutional Neural Network (CNN) which aims to solve the problem with vanishing and exploding gradients. ResNet introduces skip connections to fit the input from previous layers to the next without modification of the input. This enables us to have deeper networks. In 2015 this implementation won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in image classification, detection and localization.

To solve the problem of vanishing or exploding gradients, a skip connection is added to add the input x to the output after few weight layers as shown in Fig. 5.1:

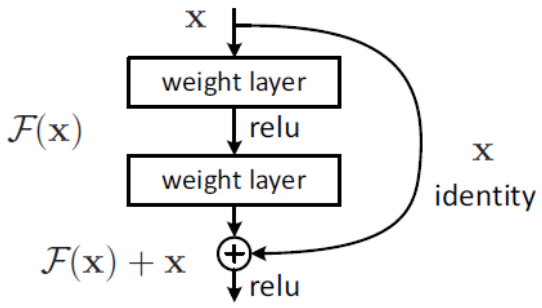


Figure 5.1: Skip connection in ResNets. Picture from Tsang (2018)

Therefore, the output is $H(x) = F(x) + x$. The weight layers are used to learn residual mapping: $F(x) = H(x) - x$. That is why even if there is vanishing gradient for the weight layers, we always still have the identity x to transfer back to earlier layers.

In 5.2 we see the ResNet architecture. The 34-layer residual network is a plain ResNet implementation with the skip connections. The VGG-19 is the is the state of the art approach used in ILSVRC 2014 and the 34 layer plain network is treated as the deeper network of the VGG-19 with more convolutional layers.

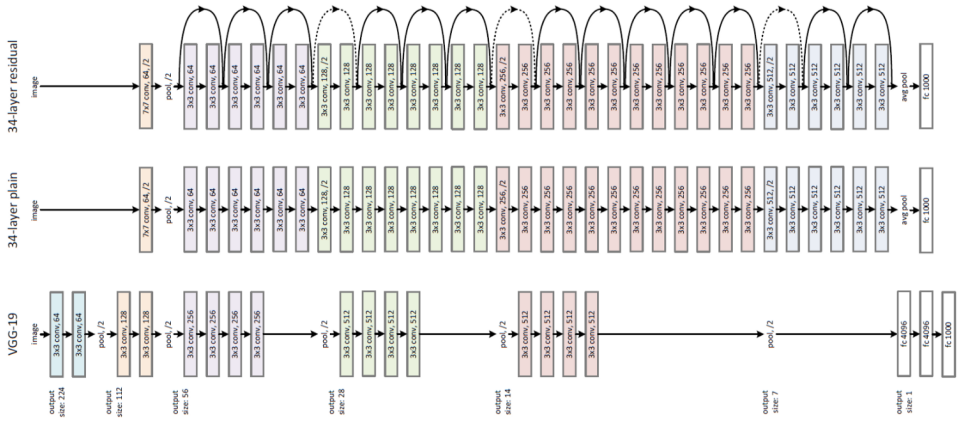


Figure 5.2: Different ResNet implementations. Picture from Tsang (2018)

Training ResNets are difficult and become even harder with the addition of further layers. They can have between dozens and thousands of layers and can in extreme cases take weeks to train. One is recommended to distribute the ResNet over several GPUs. However this requires a significant amount of resources and scaling deep learning models across multiple machines is notoriously difficult.

This makes a ResNet implementation unsuitable for our task since we are hoping to be able to run real time learning as we analyze the data.

5.3 LSTM-FCN, ALSTM-FCN and MALSTM-FCN

LSTM-FCN is an augmentation of a fully convolutional network with LSTM sub-modules for time series classification created by Karim et al. (2018). The augmentation managed to significantly improve the performance of fully convolutional networks with a minimal increase in model size and requires very little preprocessing of data to be effective. At release it managed to match or outperform most rivaling models with far better computation efficiency than the closest classification algorithms.

LSTM-FCN is efficient on various multivariate time series classification tasks, including tasks relevant to this project, like activity and action recognition. Furthermore the models are small enough to be able to deploy on memory constrained systems.

An ALSTM-FCN is a LSTM-FCN implementation with an added attention block

to the LSTM module to enhance performance. This attention mechanism allows to better visualize the decision process of the LSTM cell, as well as greatly diminish problems with long term dependencies. This improves performance on many sets and, in our case especially, helps with training stability. Allowing better results in fewer epochs.

MLSTM-FCN and MALSTM-FCN are other versions of LSTM-FCN. However, this one has the *Squeeze and Excite* blocks implemented in between the FCN layers as seen in Fig. 5.3. This further improves the performance of regular models on most test sets according to Karim et al. (2019).

The greatest strengths of all these LSTM-FCN models are; the lack of data preprocessing needed, the allowance of data that is unreliable in quality, it's low memory usage and high efficiency. This makes it a well suited model to be deployed on platforms with strict requirements and tolerances on resource usage and real time training on data.

This is the profile of our user case so these models are applicable for our cause and were therefore selected for further use in the project.

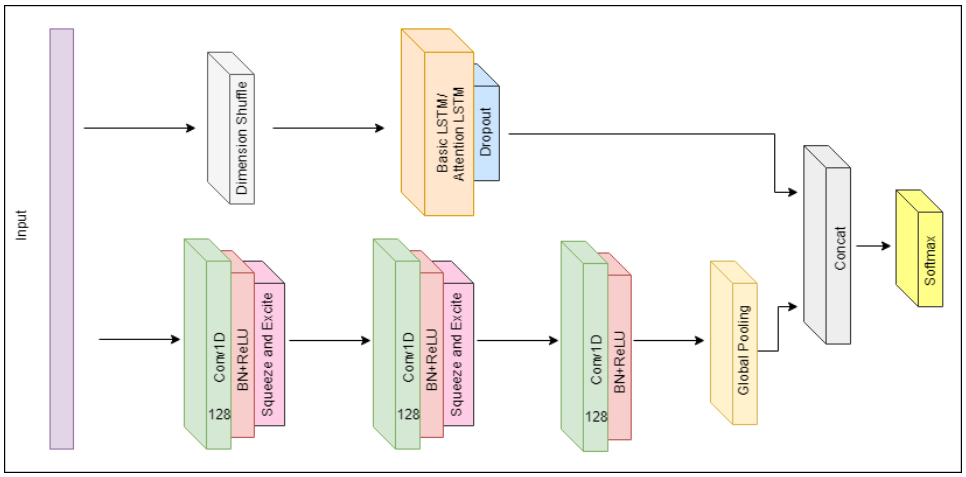


Figure 5.3: Example of LSTM-FCN implementation used in my project. Picture from titu1994 (2018)

6 Methods

6.1 MALSTM-FCN

The original MALSTM-FCN implementation created by Karim et al. (2019) and chosen for this project had open source code available to build upon. I used the code, found in titu1994 (2018), as a foundation for the project, and used their script for analyzing netflow called *netflow.py* as a starting point. Since analyzing netflow has close similarities to my own problem specification it already had many of the attributes needed for my final implementation. Many of the values within were also already tuned closely to their optimal performance for this type of time series classification.

The entire code basis supplied by Karim et al. (2019) consisted mainly of 6 scripts where 4 are utility code for the 2 others.The 4 utility files are *constants.py, generic_utils.py, keras_utils.py* and *layer_utils.py*.

keras_utils.py contains altered keras utilities needed to run and evaluate the LSTM-FCN scripts.

generic_utils.py is used by *keras_utils.py* to load data sets, calculate data set metrics and select the cutoff ranges for the sets.

layer_utils.py contains the entire MALSTM-FCN implementation and is the model used in our program.

constants.py contains the constants for each of the datasets. In our case this needs to be continually altered, based on the amount of keys we want as our input.

The two scripts that are used for running the program and creating the data sets are *generate_sets.py* and *attack_detection.py*.

generate_sets.py is used to preprocess the csv files i take into my program. In the original implementation, they handled the input data as matlab files. As my program was created to be quick to use and needed to be implementable and editable on most platforms, csv was a better choice as file format. Therefore, I rewrote and simplified the script to create processed data sets from csv files instead.

attack_detection.py is my reworked version of the *netFlow.py* script. It is the script containing the LSTM, ALSTM, MLSTM and MALSTM-FCN models which uses layers from both keras and *keras_utils.py*. It consists of mainly the four different models and matplotlib code for visualising the results. I altered the original script by adding the visualization code and experimented by altering values, like; batch size, epochs, dropout and activation function, to incrementally improve performance. Much of the entire 6 script program also needed to be altered to facilitate the csv file implementation.

6.2 Processing of UNSW-NB15 data set

The UNSW-NB15's premade training and test sets had a relatively good accuracy from the start but through some simple alterations the accuracy was greatly improved and epochs needed to obtain valid results were reduced. The original training set had a size of 82332 and the test set had a size of 175341 readings. This is bad for most machine learning algorithms since generally you want a significantly bigger training than validation set. Therefore I did some alterations to the existing data.

As shown in Fig. 6.1, I merged the training and test sets to create a new training set with a size of 257673 readings. After this I flipped the existing test set to create a new one. This dramatically improved performance. It also allowed me to utilize the entirety of the data sets instead of halving my test set. I called these new files the *Extended* versions.

I also created multiple different subsets from the data set which represented different data gathering and privacy actions needed to collect the data. The four most relevant are the original extended data set, the extended set which

only includes keys using direct metrics, a similar metrics set which included 2 averaged values and the extended set which uses metrics as well as upload and download packet statistics.

The original extended set consists of the entire 38 keys that could be gathered from the network and is not concerned with privacy of the netflow.

The extended metrics set is mainly consisting of 4 keys which show source and destination bytes as well as upload and download speed. 2 keys of average values was added to the metrics set to create the third set. These are the easiest sets of data to acquire and is as secure and private as possible.

The final data set consisted of all data that could be gathered from metrics and an upload and download packet analysis. It does not analyze the contents of these but just counts successfully transmitted or lost packages. It also contains all the values from the extended metrics set. This implementation is harder to acquire from a data gathering perspective but it is still very simple for a knowledgeable individual to acquire. It is very similar in a privacy and security aspect to the metrics set, but gives better results.

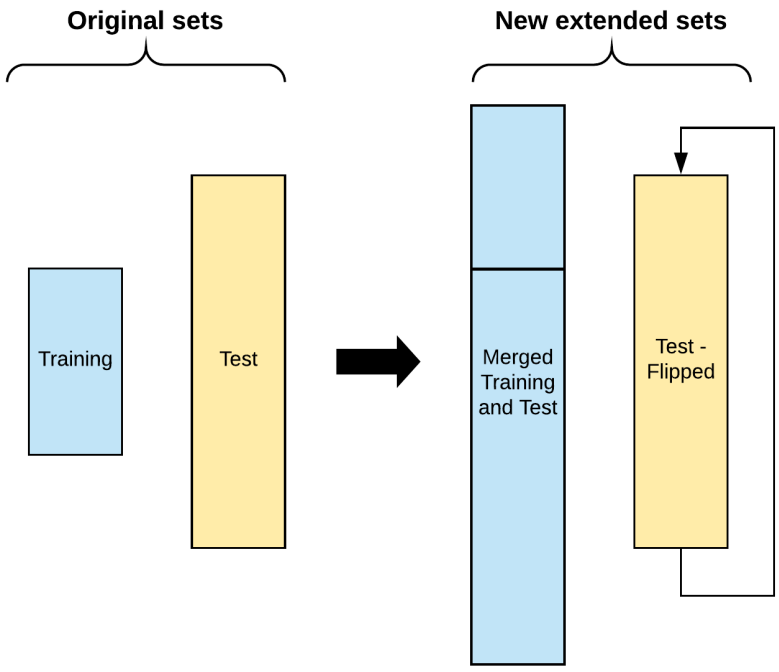


Figure 6.1: Operations done to the original training and test sets from the UNSW-NB15 dataset.

6.3 Process of running the program

The program referenced in Appendix A, is as mentioned split into separate parts. Therefore, three steps are needed each time we want to run it with a new data set:

- Step 1: First we need the *generate_sets.py* file. This creates the *.npy* training and test files that we need to execute the final script. The *generate_sets.py* has several string values which equates to distinct map folders. Each folder contains a data set with different attributes. After choosing the string corresponding to the folder you want, we run the program and get two outputs. One of them is the 4 *.npy* training and test files needed. These will be saved in the same folder as the *generate_sets.py* file. The other output is information in the python console describing the shape of the data set. The top number is the width and the bottom one is the length. We need the width value for step 2.

- Step 2: Now using the width value from the python console in Step 1, we input this into *constants.py*. The value needs to be inputted to the correct position. There are several values here all corresponding to the 47 data sets which the original MALSTM-FCN creators used in their tests. I created a 48th entry. Under the *MAX_NB_VARIABLES* we input the width value under item 48.

- Step 3: The final step is to run the *attack_detection.py* file. In the file there are 4 different *generate_model* methods. The first is the MLSTM_FCN, the second is the MALSTM_FCN, the third is the LSTM_FCN and the fourth is the ALSTM_FCN. In the trained model variable, one can change batch_size and epoch number. Choose whichever model and values are needed for your test and the program is ready to be run.

All these steps are only needed if one is testing a new data set. If the data set is the last one to have been through step 1 and 2, only step 3 is needed the next time.

7

Results

All simulations were run on a Nvidia GeForce 1070, with max Q design, laptop GPU and the loss and accuracy functions were categorical cross entropy.

Results from simulations run upon the UNSW-NB15 data set were largely conclusive.

There are several different LSTM-FCN models that have been tested for this task. As shown in Fig. 7.1 the different models produce rather different results. The two models producing the highest accuracy on both sets are the MLSTM, shown in Fig. 7.1a, and MALSTM, shown in Fig. 7.1b.

If we pay attention to Fig. 7.1c, we see that the results of the LSTM seem nearly as good as the MALSTM implementation. However, if we focus on the training accuracy it falls behind by ca. 1-1.5% and the training and validation loss is slightly higher. Important to note here is that the training and test set used, were smaller and more akin to each other than they would be in a real life setting. If the set was more diverse and had larger periods of non attack data, then the special attributes of the MALSTM, MLSTM and ALSTM would be easier to present. This also means that the LSTM could very likely be over performing on this set type compared to how it would perform in the client's situation.

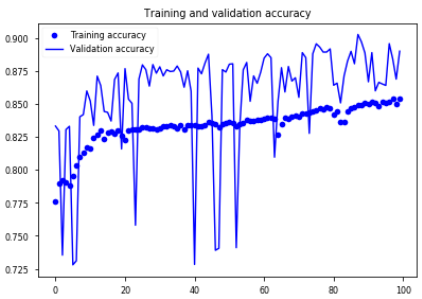
From the images we can see that the MLSTM's and MALSTM's results are generally 1-2% higher than the models without *Squeeze and Excite* blocks. Of

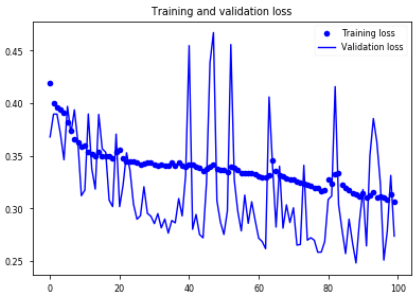
these two, the MALSTM is the most stable one and the one producing best results.

The most stable models are the LSTM and MALSTM. This shows tendencies towards the *Squeeze and Excite* and *Attention* blocks might provide better and more stable solutions if used together. The S&E blocks giving better performance while the *Attention* helping with stability. In Fig. 7.1d we see the worst results with both low accuracy and stability. This supports the theory.

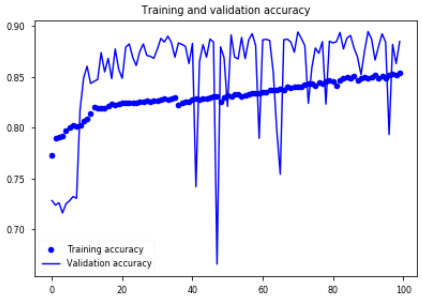
The reason behind validation accuracy being higher than training accuracy in earlier epochs might have something to do with the dropout only being applied to the training set, while the validation set remains untouched. Then in later epochs the system is more stable and handles the dropout better. Another reason could be that the validation set is plain easier than the training set, and therefore yields better results in earlier epochs. The same could be argued for loss values in the MALSTM-FCN implementation.

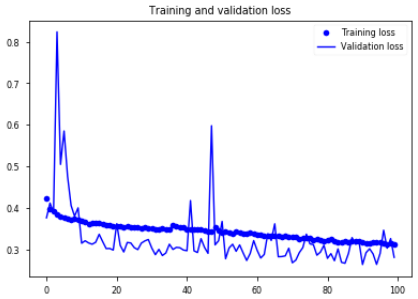
After having run these simulation and analyzing theory behind the different models, MALSTM-FCN was chosen as the best solution and was used for further experimentation.



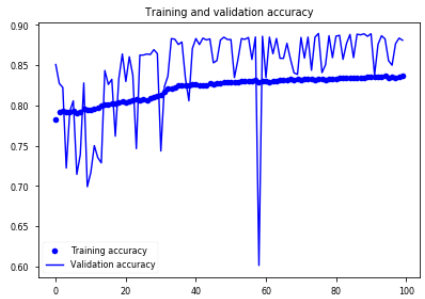


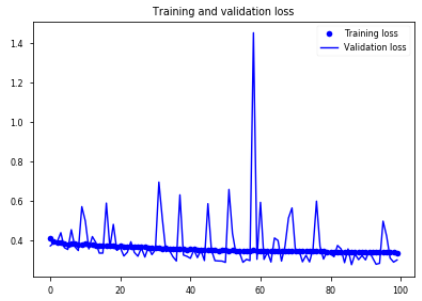
(a) MLSTM-FCN model's results after 100 epochs



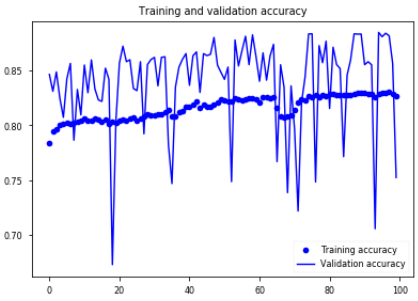


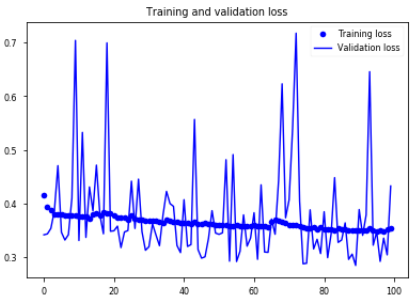
(b) MALSTM-FCN model's results after 100 epochs





(c) LSTM-FCN model's results after 100 epochs





(d) ALSTM-FCN model's results after 100 epochs

Figure 7.1: Results of simulations using different LSTM-FCN implementations

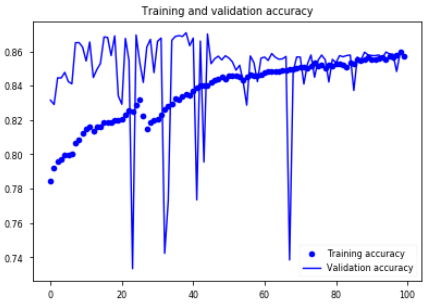
The batch size used for the model affected results significantly. Since the data

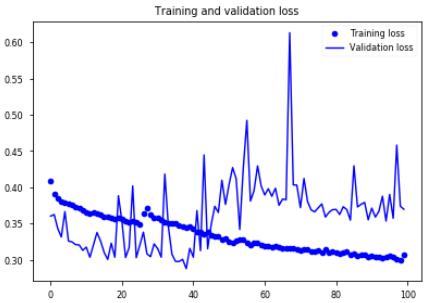
sets that this implementation uses are quite large, the program needs to be not just accurate, but also fast. Therefore, three different batch size values were experimented with; 512, 2048 and 9148. As shown in Fig. 7.2, the results differ from each other. With a batch size of 512, the accuracy results are good but the training and validation losses divert from each other after ca. 45 epochs. This would most likely only worsen if more epochs were applied.

With 2048, as shown in Fig. 7.2b, the results are more unstable but the overall training and validation loss and accuracy are converging. Not only this, but the execution time of the program was also roughly halved. This shows quite promising results. Especially, since the values are converging, we know that the training is improving the final results.

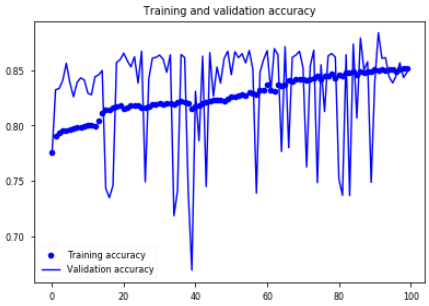
9148 was the worst batch size with unstable and diverging results. It was however, the quickest solution, again roughly halving the time of 2048. This is not enough though to make it a viable solution.

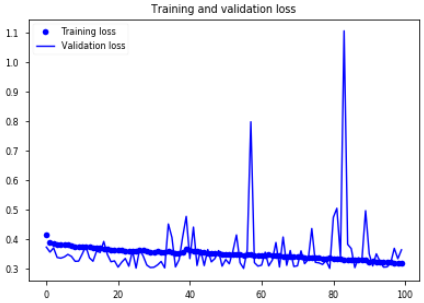
The chosen batch size was 2048. It gave the best and most stable results and was much faster to run simulations with. Run time was important since, hopefully, in the final implementation of this project we will scan real time data traffic and be able to give live feedback.



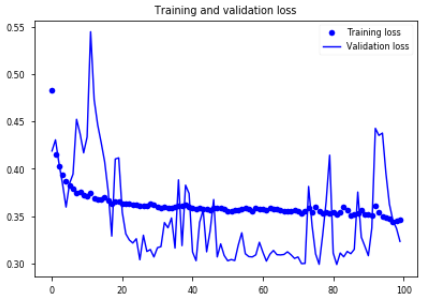


(a) 100 iterations of simulations with a batch size of 512





(b) 100 iterations of simulations with a batch size of 2048



(c) 100 iterations of simulations with a batch size of 9148

Figure 7.2: Results of simulations using different batch sizes in the MALSTM-FCN model

I created multiple sub sets of the original training and validation sets. These gave different readings in results, performance and stability. I ended up using 4 sets as representations for different real life implementations. The four sets were; 38 key set, 10 key set, 6, key set and 4 key set. As shown in Fig. 7.3, the results generally differ, with some being more similar.

The 38 key set shown in Fig. 7.3a is the most stable and the one with highest accuracy. This comes from having access to the entirety of data from our set without a care for privacy or security. Interestingly though, the loss is unstable when compared to the 10 key set, shown in Fig. 7.3b. It is however, very hard to beat the accuracy performance of this set. This can most likely be attributed to the others being sub sets of this one. In short the 38 key set is what the others will be compared against to see if their performance is within acceptable parameters.

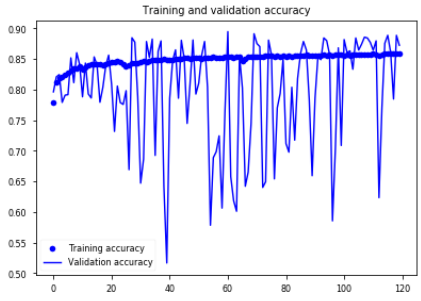
In the 10 key set we see other results. As seen in Fig. 7.3b, the stability suffers by 1-1.5% from the lack of the other 28 keys but the final result is quite close to the original. It also has more stable loss values throughout it's training epochs. The set has a tendency to overfit after 90 epochs but this is not a big issue since it reaches respectable results in just 85. In all, the set is quite close in performance to the original and has a much better run time. The 10 chosen keys replicate data that could be gathered from metrics, a basic check for lost packages and package count. This is within my set of privacy parameters used for this project.

The 6 key set is even less stable and has the same tendency as the 10 key set to worsen in performance after 85-90 epochs. As seen in 7.3c, before the loss values start diverging at ca. 85, it is still behind the 10 key solution by ca. 1-2%. These 6 keys use only up and download metrics statistics. The run time is very good and the data needed is equal to the 4 key set making this the best option if one is hoping to use solely metrics as the method of choice.

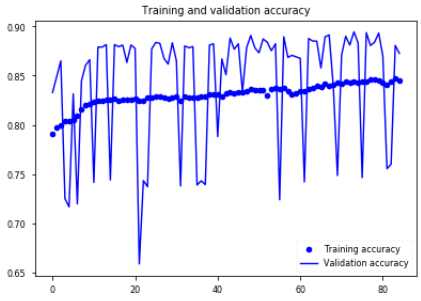
The 4 key set was the first metrics based set I made and all others were compared between this one and the 38 key set. As seen in 7.3d, it is more chaotic in its results than the others and does suffer in performance. Its loss values never really converge and the accuracy is very unstable. It shows how well the results are on a pure metrics based solution with the minimal amount of possible data. As shown in 7.3d, it is the worst performing model and implicates that other models are made better from an addition of key values.

The 10 key data set was chosen for the final implementation and is recommended if the user wishes to maintain complete privacy in their flow of data. It performs close to the original set in the tests and has a nominal amount of data intrusion. It also has a much faster run time than the original extended

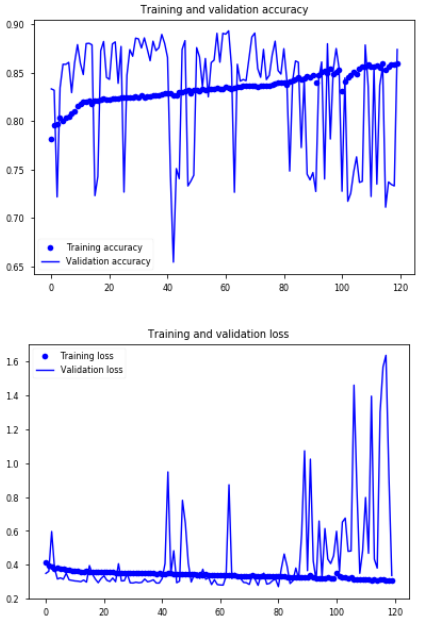
set and is close, time wise, to the 6 key set. If the user has more computing resources or is not focused in providing extra private data flows, then the 38 key set may be preferred.



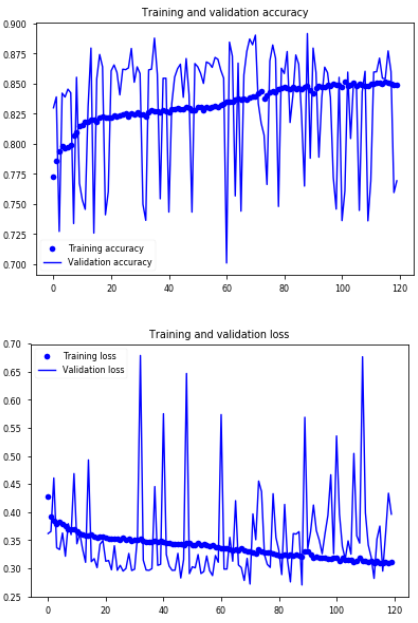
(a) MALSTM-FCN model on 38 key set over 120 epochs



(b) MALSTM-FCN model on 10 key set over 85 epochs



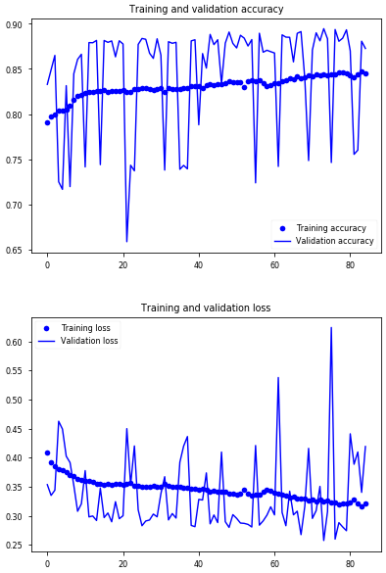
(c) MALSTM-FCN model on 6 key set over 120 epochs



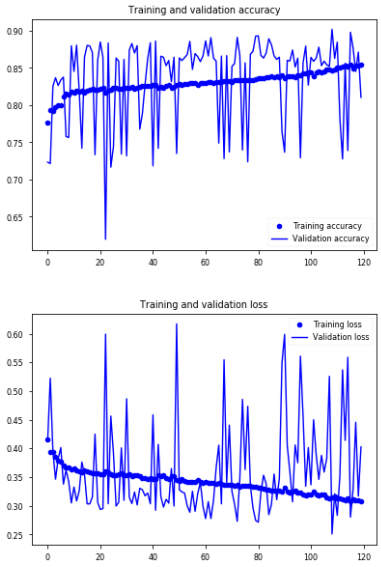
(d) MALSTM-FCN model on 4 key set over 120 epochs

Figure 7.3: Results of MALSTM-FCN simulations using different amount and types of keys in the data set

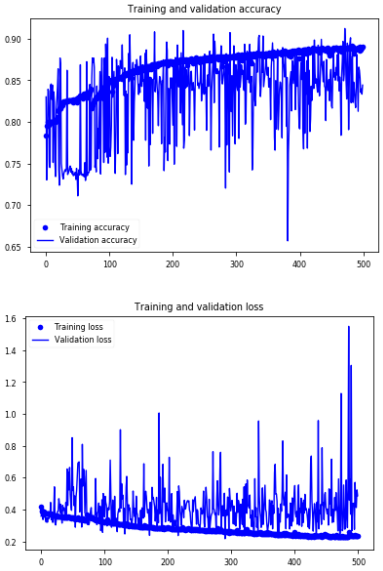
The number of epochs needed for the 10 key set is also very relevant. As seen in Fig. 7.4, after 80-120 epochs the values overfit and affect the validation set results negatively. I therefore chose an epoch number of 85 since this ensures a good result before any overfitting happens. This however might be different on another data set. The UNSW-NB15 set is, as mentioned, not truly indicative of real life situations since it is to difficult, with too many attacks at the same time.



(a) MALSTM-FCN model on 10 key set over 85 epochs



(b) MALSTM-FCN model on 10 key set over 120 epochs



(c) MALSTM-FCN model on 10 key set over 500 epochs

Figure 7.4: Pictures of MALSTM-FCN on the 10 key set with different epoch values

A very important aspect mentioned here is time to run the simulation for each set. As shown in Table. 7.1, we can see that the times are different. In the UNSW-NB15 extended data set we have 257673 readings that we train upon.

This evens out to about 72 hours of data if the readings were made once per second. We also test upon 175341 readings during the simulations. We do not take this into account since we need to test the data against a set in real time training as well but we do not know what the size of this set will be. When we run the different key sets we see that all of them are capable of running quite smoothly on real time applications, with even the 38 keys and 500 epochs model processing 95 seconds of data per second. This is the slowest of all the sets, and run on the slowest model, MALSTM-FCN. Still it shows promise for use in real time.

If we evaluate the set that was recommended, the 10 key set, it runs very fast and will process 428 seconds of data per second. This is even better and is so fast that one can probably have multiple readings per second very comfortably. This is also when run on a laptop so a dedicated server will most likely increase the performance several times over.

In summary, I state that the MALSTM-FCN model with a 10 key set works very well for real time data collection and analysis.

Number of keys	Epochs	Seconds of data trained upon per real time second
4	120	594.338s
6	120	513.237s
10	85	427.844s
38	500	95.094s

Table 7.1: Results of a time comparison between the time needed to run the script and the 72 hours of data in the UNSW-NB15 extended data set.

Lastly, we review the final accuracy on each of the sets when run with the MALSTM-FCN model with a close to optimized amount of epochs. As seen in 7.2, the results follow suit from what we have gathered so far. The 38 key set is 1.5-2% higher than the closest competitor, the 10 key set, and has an accuracy of 89.7%. This shows real promise when you take into account the difficulty of the data set compared to real life data. The model has good capabilities for this type of time series analysis and it shows. If a more realistic data set was used it would most likely increase the performance significantly.

The drawback with the 38 key set is as stated the time needed to run and the privacy aspect. The 10 key set is therefore the one I recommend for quick and private use. It is close in performance with 88.07% accuracy and only 5.5% of

the time needed to run. It is also very protecting of user privacy, requiring only metrics and some simple package statistics. Compared to the others it is the most preferred set and model combination.

Number of keys	Epochs	Runtime in seconds	Final Accuracy
4	120	423	0.84830
6	120	491	0.85625
10	85	589	0.88068
38	500	10626	0.89708

Table 7.2: Final accuracy results for each key set with optimized epoch number

8

Discussion

Even though the results are quite conclusive some of the basic parameters can be further discussed.

The choice of a 10 key set, as the best solution for this project, may not be as relevant for all applications. The set is several times faster than the 38 key one and is, arguably, close in performance as well. However, the 38 key set is still capable of learning on 95 seconds of data per second on a Nvidia GeForce 1070 laptop card. The performance would probably increase if implemented in a proper and dedicated computing environment. The 38 key set is then well inside the realm of being used for real time data processing and the resources needed to execute it would be small. Even if it is several times that of the 10 key set.
Some cloud server providers might not see the importance of direct data privacy as well and this would also impact the decision. Therefore, the choice of best set would most likely not be the same for every client or business.

On the same note, the 4 and 6 key sets also give usable results and are the most GDPR friendly solutions. With a final accuracy of about 85%, these two are also relatively close in performance to the others. This shows that upload and download metrics might be a better indication of attacks and anomalies than previously thought. Since the 4 key solution is able to process 594 seconds of data per second, several operations can be run simultaneously on a data set. With a different implementation containing multiple scripts, we can most likely improve performance past the 90% mark of the 38 key set. We could run

multiple 4 key set MALSTM-FCN implementations at the same time and cross check results to get better performance. We can add safeguards to alert us of false positives and much more since the actual data processing part has such a low impact on computing resources. With further research and development this could end up being the better solution.

The overfitting problem is of interest as well. Since the MALSTM-FCN implementation would not run on the sets tested here when used by a cloud service provider or other business, the overfitting would be affected. Most likely, since real life data would have better similarities and unique qualities to learn upon, the overfitting would decrease. While the data would most likely have more similarities between each training round, it would also have segments that would make it unique, giving less general bias in the sets. MALSTM-FCN is better suited for larger amounts of data and I believe that it's performance would benefit from real time data extraction. Therefore, I believe that in a real life situation, amounts of epochs can be increased resulting in a minimum of 1-2% increase in accuracy across all key set types.

Another important note is the time to run each model and set. The tests were as stated done on a medium grade laptop GPU with other processes running in the background simultaneously. In the future I recommend the models should be tested in the client's computing environment to better tell what implementation is best for their amount of resources. Alternatively, the models should be run on hardware representing what could be found in the today's industry, and results should be used for giving final recommendations to clients.

The task given by the client was to detect three cases of interest; attacks from the outside, attacks from within and altered network patterns in a customers traffic. With the data set I ended up using, two of these cases were tested. Attacks from the outside was the most prominent data type in our set while only shellcode and worms could be classified as being able to do attacks from within. Therefore, my results are skewed towards detecting attacks from outside and is not tested on altering traffic patterns at all. However, since both abuse from within and outside could be classified as altered network traffic patterns, the program has promises for detecting such anomalies as well. If the MALSTM-FCN model focuses on attack patterns, safe traffic or both is difficult to tell from the results.

Another issue with my results are that they do not show the model's tendency to fail or succeed based on what type of attack it is classifying. This makes it difficult to discuss what degree of accuracy the model has in relation to attack types or anomaly detection. The model could have perfect accuracy for certain attacks and fail for all the others for all we know. This can be attributed to two things. The time schedule did not allow for me to develop

such stats into my program and the data set has mixed ongoing attacks making it difficult to separate the results. At times there are 7 attack types running simultaneously and this muddles the results. This is also not representing real life attacks.

The chosen data set was not as applicable to our case as first imagined. The biggest factors to this is the difficulty of the set, the muddled attacks within and the lack of certain anomaly types. The set is not fully representable of the client's environment and this affects the final solution. However, none of the other sets analyzed here would likely have had better results. This is due to most of them not having the metrics or the size needed, as well as finely labeled data in their sets. They also had a bigger attack to normal traffic ratio and this would be even further from the client's situation.

The risk of not receiving data from the client was well founded and actions to lessen the impact of this were made. The idea of using data created elsewhere was satisfactory and, in the end, made the project viable. Other ideas could have been implemented but they would have been very time consuming and, while not perfect, the current solution worked. The biggest issue was that the final results of the project might not be as closely relevant for the client as first hoped, but this was expected. All things considered the choice of solution was correct. It has not been tested directly on client's data traffic but it shows promise for this type of application. This fits within the original task parameters provided by the client.

When putting my final results up against the state of the art solutions available today, we see interesting trends. My program outperforms all known tests run on pure metrics data with a large amount. It does not match up to more specified solutions but for the solutions aimed to catch all types of anomalies, it performs as well or better then its rivals. The solution is less computationally expensive and is of a smaller size which makes it easier to deploy on memory or resource strained systems. All things considered it pushes boundaries of the current network anomaly detection.

The strongest points of my solution is that it is robust, efficient and accurate. It handles data of varying quality, it outperforms several state of the art models and requires a very small amount of preprocessing of data to have satisfying results. The weak parts is that it requires significant amounts of data to reach it's full potential and that it is currently untested on multiple facets of attack detection. Even though it shows promise, since it is untested the results could still be lower than expected. The method for running the program is also needlessly complicated and could be vastly improved. Overall however, most of the weak parts are or can be addressed. The idea needs more testing, and the large amount of data needed to reach optimal performance is perfect for

a real time solution anyway. The strong points does in my opinion make this solution a rival to, if not surpassing, state of the art.

9

Conclusion

The results of both the research and development have been quite conclusive. The MALSTM-FCN outperforms other LSTM-FCN implementations and other types of machine learning models in this project setting. It is the best performing one of the LSTM-FCN's and is equal to or better in results than the industry leading models like ResNet and HIVE-COTE. It is also much better in efficiency and will have a fraction of the preprocessing needed and run times of the other models.

The model does not manage to reach the levels of results that more specified solutions manage to acquire. For example (Najafabadi et al., 2014), which acquired a 99.7% accuracy on detecting DDoS attacks. This type of solution is very good for it's specified use, but will be useless for any other attack types. Creating the keys needed is much more difficult, the preprocessing of data takes more effort and the time needed to run this type of program will be much larger. One also needs to create this specified type of program for multiple attack types and run them simultaneously. That is why even this solution has it's drawbacks.

At an 88.07% accuracy however, my implementation is quite good as a security net aiming to detect all types of abuse. It will most likely have a higher accuracy the more data it gets to train upon and will likely perform better on real life data. Since it is less cluttered with overlapping attacks than the UNSW-NB15 dat set, it will be easier to predict and categorize the results.

Therefore, I recommend a MALSTM-FCN implementation using 10 keys based on metrics and basic package statistics. This should be used for the task of detecting anomalies in network traffic, with processed data being used to further train the model. It is able to be implemented on a real time network and is efficient in it's use of computing resources. It outperforms other models in almost every aspect and is highly recommended as the backbone of an application used for detecting network traffic or computing resource abuse.

10

Future Work

There are multiple segments of this report that can be used as a basis for further work.

One is the creation of a new network anomaly data set that specifically aims to have realistic noise, clear and concise attacks in their own time frames and labels on the different attack types. The work put into researching data sets prove that there is definitely room for improvement in the current sets available for attack detection. Creation of such data would greatly help any future work into this field.

The different metrics sets, models and theory explained here can also be used for a detection radar where multiple hits for suspicious activity within a time frame results in detection. This will most likely even further increase performance. Putting effort into creating a finished tool based on the result from this report would most likely be very beneficial for any cloud service provider currently on the market.

Further work can also be put into improving the LSTM-FCN implementations or finding new areas of use for this type of model. It is a new and unexplored model that I believe have much more use than it has currently been tested for. Testing has mostly been focused on image processing and, very recently, for time series. This means that anything outside these boundaries is unexplored. Even within these fields the research possibilities are not depleted. Similar inquisitions can be done into the *Squeeze* and *Excite*, and *Attention* blocks.

Another possibility is analysis into if the creation of multiple sub scripts more specified into certain attack types is viable for a real time solution. As well as which keys and values are best to find certain types of attacks. This is a very interesting thought for anyone interested in creating a jack of all trades type of application for server providers facing issues with cyber crime.

Bibliography

Alert Logic (2017). Cloud security report, 2017. https://www.alertlogic.com/assets/industry-reports/alertlogic-cloud-security-report-2017.pdf.

Armin, J., Thompson, B., Ariu, D., Giacinto, G., Roli, F., and Kijewski, P. (2015). 2020 cybercrime economic costs: No measure no solution. In *2015 10th International Conference on Availability, Reliability and Security*, pages 701–710.

Aruna, E., Shri, A. A., and Lakkshmanan, A. (2013). Security concerns and risk at different levels in cloud computing. In *2013 International Conference on Green Computing, Communication and Conservation of Energy (ICGCE)*, pages 743–746.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). arxiv:1409.0473, 2014. In *Neural Machine Translation by Jointly Learning to Align and Translate*.

Dietterich, T. G. (2002). Machine learning for sequential data: A review. In Caelli, T., Amin, A., Duin, R. P. W., de Ridder, D., and Kamel, M., editors, *Structural, Syntactic, and Statistical Pattern Recognition*, pages 15–30, Berlin, Heidelberg. Springer Berlin Heidelberg.

García, S., Grill, M., Stiborek, J., and Zunino, A. (2014). An empirical comparison of botnet detection methods. *Computers and Security*, 45:100–123.

Graves, A. (2012). *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385. Springer.

He, Z., Zhang, T., and Lee, R. B. (2017). Machine learning based ddos attack detection from source side in cloud. In *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 114–120.

Heinrich, G. (2018). Image segmentation using digits 5. https://devblogs.nvidia.com/image-segmentation-using-digits-5/.

Hu, J., Shen, L., and Sun, G. (2017). Squeeze-and-excitation networks. *CoRR*, abs/1709.01507.

Karim, F., Majumdar, S., Darabi, H., and Chen, S. (2018). Lstm fully convolutional networks for time series classification. *IEEE Access*, 6:1662–1669.

Karim, F., Majumdar, S., Darabi, H., and Harford, S. (2019). Multivariate lstm-fcns for time series classification. *Neural Networks*.

Kirubavathi, G. and Anitha, R. (2016). Botnet detection via mining of traffic flow characteristics. *Computers and Electrical Engineering*, 50(C):91–101.

Lea, C., Vidal, R., Reiter, A., and Hager, G. D. (2016). Temporal convolutional networks: A unified approach to action segmentation. *CoRR*, abs/1608.08242.

Lines, J., Taylor, S., and Bagnall, A. (2018). *HIVE-COTE: The Hierarchical Vote Collective of Transformation-based Ensembles for Time Series Classification*.

Mendoza, M.-A. and Bedford, H. (2018). *Machine Learning for Anomaly Detection on VM and Host Performance Metrics Use machine learning techniques to reduce the number of false alerts sent to IT system operators. Discover alert conditions not detected by conventional IT system monitoring*.

Moustafa and Slay (2015). The unsw-nb15 dataset description. https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets/.

Moustafa, N. and Slay, J. (2015). Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 Military Communications and Information Systems Conference (MilCIS)*, pages 1–6.

Najafabadi, M. M., Khoshgoftaar, T. M., Kemp, C., Seliya, N., and Zuech, R. (2014). Machine learning for detecting brute force attacks at the network level. In *2014 IEEE International Conference on Bioinformatics and Bioengineering*, pages 379–385.

Pascanu, R., Gulcehre, C., Cho, K., and Bengio, Y. (2013). arxiv:1312.6026. In *How to Construct Deep Recurrent Neural Networks*.

Protic, D. (2018). Review of kdd cup '99, nsl-kdd and kyoto 2006+ datasets. *Vojnotehnicki glasnik*, 66:580–596.

Sharafaldin, I., Habibi Lashkari, A., and Ghorbani, A. A. (2018). Toward gener-

ating a new intrusion detection dataset and intrusion traffic characterization. *Proceedings of the 4th International Conference on Information Systems Security and Privacy*.

titu1994 (2018). titu1994/mlstm-fcn. https://github.com/titu1994/MLSTM-FCN.

Tsang, S.-H. (2018). Review: Resnet — winner of ilsvrc 2015 (image classification, localization, detection).

A

Code and data sets related to project

Appended to this report is a zipped file containing the entirety of images, data sets and code used for this project. The zipped file contains three folders, a requirements text file and *attack_detection.py*. The three folders are; *data*, *utils* and *weights*.

data contains all of the data sets and images used for this report, as well as *generate_sets.py*.

utils contains all the utility files of the project as well as *constants.py*.

weights is where the weights made from the simulations are put.