

St. Cloud State University

theRepository at St. Cloud State

Culminating Projects in Information Assurance

Department of Information Systems

4-2020

Lightweight Deep Learning Framework to Detect Botnets in IoT Sensor Networks by using Hybrid Self-Organizing Map

Saad Khan

skhan1@go.stcloudstate.edu

Follow this and additional works at: https://repository.stcloudstate.edu/msia_etds

Recommended Citation

Khan, Saad, "Lightweight Deep Learning Framework to Detect Botnets in IoT Sensor Networks by using Hybrid Self-Organizing Map" (2020). *Culminating Projects in Information Assurance*. 100.
https://repository.stcloudstate.edu/msia_etds/100

This Thesis is brought to you for free and open access by the Department of Information Systems at theRepository at St. Cloud State. It has been accepted for inclusion in Culminating Projects in Information Assurance by an authorized administrator of theRepository at St. Cloud State. For more information, please contact tdsteman@stcloudstate.edu.

Lightweight Deep Learning Framework to Detect Botnets in IoT Sensor Networks

by using Hybrid Self-Organizing Map

by

Saad Khan

Thesis Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Information Assurance

May, 2020

Thesis Committee:

Akalanga Maleiwa, Chairperson

Jieyu Wang

Farra Hazem

Abstract

In recent years, we have witnessed a massive growth of intrusion attacks targeted at the internet of things (IoT) devices. Due to inherent security vulnerabilities, it has become an easy target for hackers to target these devices. Recent studies have been focusing on deploying intrusion detection systems at the edge of the network within these devices to localize threat mitigation to avoid computational expenses. Intrusion detection systems based on machine learning and deep learning algorithm have demonstrated the potential capability to detect zero-day attacks where traditional signature-based detection falls short. The paper aims to propose a lightweight and robust deep learning framework for intrusion detection that has computational potential to be deployed within IoT devices. The research builds upon previous researches showing the demonstrated efficiency of anomaly detection rates of self-organizing map-based intrusion. The paper will contribute to the existing body of knowledge by creating a hybrid self-organizing map (SOM) for the purpose of detecting botnet attacks and analyzing its accuracy compared with a traditional supervised artificial neural network (ANN). The paper also aims to answer questions regarding the computational efficiency of our hybrid self-organizing map by measuring the CPU consumption based on time to train model. The deep learning prototypes will be trained on the *NSL-KDD dataset* and *Detection of IoT botnet Attacks* dataset. The study will evaluate the performance of a self-organizing map based k-nearest neighbor prototype with the performance of a supervised artificial neural network based on validation metrics such as confusion matrix, f1, recall, precision, and accuracy score.

Table of Contents

	Page
List of Tables	5
List of Figures.....	6
Chapter	
I. Introduction.....	7
Problem Statement.....	9
Objective of the Study.....	11
Study Questions.....	13
II. Background and Review of Literature.....	14
Introduction	14
Literature Related to the Problem.....	14
Literature Related to the Methodology	17
III. Methodology	20
Introduction	20
Data Analysis.....	21
NSL-KDD Dataset.....	21
Detection of IoT botnet Attacks dataset	24
Performance indicator: Confusion Matrix.....	26
Performance indicator: Accuracy/Recall/Precision.....	27
Performance indicator: Feature Score.....	28
Performance indicator: K-Fold Cross Validation.....	28

Chapter	Page
Performance indicator: CPU Usage.....	28
First Layer of Semi-Supervised Model: The Self-Organizing Map.....	29
Second Layer of Semi-supervised Model: K-nearest neighbor.....	31
Supervised Model: Artificial Neural Network.....	32
Design of the Study.....	33
Tools and Techniques.....	35
IV. Results.....	36
V. Discussion and Conclusions.....	39
References.....	52

List of Tables

Table	Page
I. Mirai Botnet IP addresses were found in 164 countries.....	8
II. Subclasses of intrusion attacks in the NSL-KDD Train+ dataset.....	22
III. Subclasses of intrusion attacks in the NSL-KDD Test+ dataset.....	23
IV. Dataset Properties and Training Summary.....	25
V. Artificial Neural Network performance on NSL-KDD.....	40
VI. Artificial Neural Network Subclass performance on NSL-KDD.....	40
VII. Stratified K-fold Cross Validation over 10 iterations.....	41
VIII. Artificial Neural Network performance on Mirai Botnet Dataset.....	43
IX. Artificial Neural Network performance on Mirai Botnet Dataset.....	43
X. Stratified K-fold Cross Validation over 10 iterations.....	44
XI. Accuracies metrics after fitting the previous fitted network on evaluation data..	46
XII. Accuracies metrics after fitting the previous fitted network on evaluation data..	47
XIII. ANN model and Hybrid SOM model performance on NSL-KDD dataset.....	48
XIV. ANN model and Hybrid SOM model performance on IoT botnet Dataset.....	49
XV. Hybrid SOM on NSL-KDD after Scaling Down Nodes over 10000 iterations...	50
XVI. Hybrid SOM on IoT Dataset after Scaling Down Nodes over 10000 iterations..	51

List of Figures

Figure	Page
1. Confusion Matrix.....	26
2. Simplified Self-organizing Map Architecture.....	30
3. Simplified representation of K-Nearest Neighbor Model.....	31
4. Visual Representation of Perceptron.....	32
5. The methodology design.....	34
6. Training of ANN model on NSL-KDD Test+ Dataset over 3 Epochs.....	39
7. The Multiclass Confusion Matrix for NSL-KDD test+.....	39
8. Training of ANN model on IoT Botnet Attacks Dataset over 3 Epochs.....	42
9. The Multiclass Confusion Matrix IoT Botnet Dataset.....	42
10. Parameters for Hybrid SOM initialized and fitted on NSL-KDD Test+.	45
11. Parameters for Hybrid SOM initialized and fitted on IoT Botnet Attacks.....	46

Chapter I: Introduction

The Internet of things has witnessed extraordinary growth in the past few years and is predicted to reach up to 20 billion devices by 2020. Heterogenous IoT devices do come with unprecedented vulnerabilities that are relatively easier to exploits by attackers. Attackers have established inherent chinks in most of IoT devices and continue to come up with sophisticated intrusion techniques. Hackers target IoTs with default set factory passwords, lack of encryption at rest and in transit, lack of password attempt lockout, outdated firmware, SSH listening permissions, and SQL injection vulnerabilities [1]. Once an IoT device has been successfully breached by a hacker after exploiting these vulnerabilities, the infected device becomes a part of a Botnet. The botnet is a collection of connected devices and computers on a network compromised by an attacker who can get access and successfully control all the hosts and devices connected within the network [2]. In 2016, the number of distributed denial of service attacks had reached an alarming peak of 1.35 terabytes per second that were carried out by Mirai malware, specifically targeting IoT devices [3]. In 2006, a large-scale botnet attack Mirai was able to infect 49,657 unique IPs in 165 countries, as illustrated in Table I, which mostly contained IoT devices such as CCTV devices, baby monitor devices, and routers [4].

Table I

Mirai Botnet IP addresses were found in 164 countries [4]

Country	% of Mirai botnet IPs
Vietnam	12.8%
Brazil	11.8
United States	10.9%
China	8.8%
Mexico	8.4%
South Korea	6.2%
Taiwan	4.9%
Russia	4.0%
Romania	2.3%
Colombia	1.5%

Distributed Denial of Service can be broadly categorized in protocol attacks, application-layer attackers, and volume-based attacks. Protocol attacks concentrate on depleting the victim's server or devices connected to the network. Volume-based attacks are focused on flooding the target's bandwidth rendering the network connection unusable. The application layer targets a web server so that it cannot function correctly [3]. The Mirai botnet specifically targets IoT devices that come with inherent security vulnerabilities [5].

Traditional signature-based detection has a significant vulnerability to zero-day attacks, and malware developers can alter the malware signature to avoid detection.

However, the recent progress in the neural network domain has resulted in a robust implementation of intrusion detection frameworks that have proven to demonstrate higher detection rates for unknown network packets containing malicious payload than traditional signature-based detection [6] [7] [8] .

Problem Statement

In intrusion detection landscape, traditional signature-based detection systems scan files and look for unique attributes and characteristics to determine if an object is a malware or a normal file. The intrusion detection system updates its repository and keeps millions of signatures to identify malicious files. In Cisco 2017 Annual Cybersecurity Report, it is reported that 95% of the malware objects are generated within 24 hours, which means traditional signature-based detection has inherent vulnerabilities [9]. In a situation where the intrusion detections system is not updated in a timely manner, the malicious file can bypass the intrusion detection system and exploit vulnerabilities. Another way attackers exploit signature-based detection is by altering the code within the malware object, register renaming, compressing the code, or by merely adding junk code [9].

To address this problem, the implementation of machine learning and deep learning models to detect zero-day attacks has proven to outperform the traditional signature-based intrusion detection framework [6], [7], [8].

Al-Garadi et al. in his research suggests, “ML and DL frameworks that can efficiently reduce computational complexity should be developed. Developing real-time detection and protection systems are important for providing effective security

mechanisms, particularly for large- scale IoT systems” [8, p. 32] . The research indicates the growing need for machine learning or deep learning frameworks that would reduce the computational complexity so it can be deployed in IoT devices to provide a localized detection framework [8].

A deep learning algorithm has been demonstrated to prevent malicious attacks as well as sophisticated zero-day attacks with high accuracy. However, using deep learning for anomaly detection requires computational resources, and recent studies have been implementing ways to use a deep learning intrusion detection model for the purpose to implement on a live data stream that is computationally efficient [10], [11], [12]. To address the computational complexity drawback, variations of the hybrid self-organizing map have proven to show high detection rates and requires low computational resources [3].

The self-organizing map is an artificial neural network that converts high dimensional input into a 2-D representation. Self-organizing map parameters can be tuned by reducing the number of neurons for speed up the training time but does have to affect the detection rate [3]. In [13] deployed the self-organizing maps in heterogeneous IoT devices by tuning the nodes to reduce computational power and the research concluded, “the detection rate and accuracy are improved because of the well-adaptation to local traffic at the SOM filters” [13, p. 7]. There are plenty of studies regarding SOM detection rate on botnet detection, but there have been few when it comes to SOM with an additional layer(s). In [3] suggested that utilizing an additional layer in self-organizing maps detection performance can be improved significantly.

Objective of the Study

The motivation for this paper is to build a lightweight and robust deep learning prototype to detect IoT botnet and network intrusion. The importance of this study is that it would provide a thorough accuracy analysis for researchers looking to develop a lightweight deep learning model for IoT devices deployed at the edge of the network. To this purpose, we will focus on four main objectives. Firstly, we want to train our hybrid self-organizing map and compare its predictive power with a supervised artificial neural network. Secondly, we will measure what type of distributed denial of service attacks yield higher accuracy results using our hybrid self-organizing maps compared to the artificial neural network. Thirdly, we will compare the computational usage our semi-supervised self-organizing map requires by measuring the time to train measure. Lastly, we will study the tradeoff between scalability and detection accuracy results of our hybrid self-organizing prototype by reducing the number the nodes in our self-organizing map and presenting the comparison.

The results of the performance of our SOM based k-nearest neighbor will contribute to the body of knowledge so researchers can determine our hybrid SOM prototype's effectiveness for Mirai attack detection and network intrusion attacks. This research will survey the performance of SOM based k-nearest neighbors and ANN by training them on the Detection of IoT botnet Attacks dataset and KDD-NSL dataset. We aim to create a deep learning framework for intrusion detection that would yield higher detection rates. These two proposed deep learning models will be measured based on precision, accuracy, confusion matrix, f1 score, false-positive rate, and anomaly metric.

The research will explore the relationship between the metrics to propose which deep learning models perform on Mirai botnet malware on IoT devices and network intrusion connections.

The results of the study will be further examined by looking at the types of DDoS attacks that were more susceptible to detection by our semi-supervised model. The Detection of IoT botnet Attacks dataset contains ack flooding, scan flooding, syn flooding, and UDP flooding, whereas the *NSL-KDD dataset* contains denial of service traffic, user to root traffic (U2R), remote to local (R2L) and probing traffic [14], [15]. The computational usage of our hybrid SOM will be compared with the supervised ANN model. The results will present the time it took to train our prototypes in seconds as a measure of CPU resource usage. These results can be leveraged by future researchers to look at the lightweight capability of hybrid SOM for localized IoT deployment.

This research will leverage the accuracy metric results to answer questions about the practicality of the proposed hybrid self-organizing map prototype being deployed within IoT devices. The paper will look at the tradeoff between scaling down the SOM parameters with the detection rates of anomalous traffic. The results of the tradeoff will be measured by tuning down the nodes in our SOM and then measure the drop of our accuracy metrics based on accuracy, precision, recall false-positive rates, and feature score. We will also measure the time it takes for our model to train and test on our datasets and compare the model after being tuned when nodes of SOM are reduced. The research will look at the training time it took for our hybrid SOM model to be trained and compare it with a traditional supervised ANN model.

Study Questions

1. Is a hybrid self-organizing map better at detecting Botnet IoT attacks and network intrusion attacks than a supervised artificial neural network?
2. Which class of botnet IoT attack gets detected with higher accuracy using a hybrid self-organizing map?
3. Given the additional layer of the k-nearest neighbor algorithm to a self-organizing map, does the proposed semi-supervised prototype has more computation overhead than a supervised artificial neural network?
4. Given the additional layer of the k-nearest neighbor algorithm to a self-organizing map, if we adjust the number of neurons parameters for scalability, how much does the computation performance compromised the detection performance?

To summarize this chapter, so far, the study has provided a brief introduction to the challenges when implementing machine learning for botnet and intrusion detection and have briefly reviewed the framework we will be following to train and test the results of the study. In the following chapter, we will go in detail the literature review of the application of machine learning in the intrusion detection domain, which will serve to give a holistic understanding of research done thus far. In the following chapters, the study will also cover the methodology, dataset, and technologies the author employed to get the result of the studies. The author will address the challenges and limitations of the methodology for future works to consider.

Chapter II: Background and Review of Literature

In this chapter, the study will show research done on the application of self-organizing map in the domain of malware and intrusion detection and share the conclusions and limitations shared by their authors. The papers we discuss in this chapter contains demonstrations of intrusion detection systems based on machine learning algorithms and their successful deployment. Some of the papers in the literature review are focused on large IoT exploits during distributed denial-of-service attacks and the strategy to mitigate them. The papers also include a general outlier detection framework that was not applied to the intrusion detection domain but serves as a robust framework for anomaly detection.

Literature Related to the Problem

Langin et al. [16] created a two-layer self-organizing map for the detection of malignant network traffic. The first layer clusters the traffic, and the second layer classifies the traffic. The self-organizing map is trained on denied firewall log entries. The researchers focus on botnet malware where the hacker is successfully able to infect computers or IoT devices, and subsequently able to get unauthorized access through command and control center. The command and control center get access through multiple protocols such as Peer-to-Peer technology and Internet Relay Chat. The P2P protocol comes with a high level of anonymity since tracing back the source of the attack is incredibly difficult as the traffic is encrypted and comes from a distributed system. The authors talk about traditional intrusion detection approaches and how they are not adequate to mitigate the threat of botnets. The paper critiques misuse detection

and anomaly detection approaches. The research deems misuse detection inadequate for P2P botnet since it assumes advanced knowledge of the botnet and does not anticipate a zero-day attack, however effective when used to detect botnets on Internet Relay Chat protocol. The researchers also critique using anomaly detection for botnet detection since the central assumption is the traffic network in consideration is already benign traffic, which in itself has no guarantee. The methodology in the research has two main steps, clustering steps and classifying steps. The clustering step is where the self-organizing maps are trained based on the denied log entries generated by the host firewall. The logs are queried through MySQL in a matrix table with multiple dimensions, including source IP and port, destination IP and port, time gap, protocol, unique identifier, and date. The queries are stored in a way where each line is a vector, so SOM can be trained to find clusters over the vectors. In the research, Once the bot clusters have been determined using SOM, the study classified the future daily logs to observe local IP addresses with external denied entries. The vector of IP address that showed up in denied firewall entry logs is reviewed. The researchers look for the best matching unit in the vectors to see a correlation with bot then is specified as a suspect. Langin et al. tested the methodology on Southern Illinois University campus, and it states in the paper, "SOM produced 18 suspects in 37 alerts in 96 days" [16, p. 8]

Langin et al. [16] in the paper adds that the limitation of the model lies in its replication since the SOM must be trained on each networks' own firewall denied entries and the resources consumed. The study has emphasized the effectiveness of detecting

malignant botnet traffic on networks that are already infected by botnets and cannot be detected by misuse detection or anomaly detection.

Dao et al. [13] proposed a DDoS prevention framework by deploying smart filters at the edge of a network supervised by a central controller. The proposed framework is termed the MECShield framework that leverages the power of edge computing to localize traffic analysis at the edge of the network. The smart filters work in coordination together and are trained on local traffic using the self-organizing map. The trained SOM matches the malicious traffic with the SOM map to determine DDoS attacks. The smart filters are trained on three datasets, including the CAIDA-attack-traffic dataset, NSL-KDD dataset, and DARPA 2009 dataset. The MECShield framework is compared with a distributed self-organizing map and a Centralized Self-organizing map. The centralized self-organizing map is where the SOM filter is located at the controller site for analysis and receives all the traffic from heterogeneous IoT devices for analysis. The distributed self-organizing map entails the self-organizing map trained by all agents that are merged at the controller site in one central SOM. Eventually, the merged SOM is delivered back to the agents for traffic administration. The results were concluded as follows, "In both criteria, the MECshield performed better than the other schemes. This is because SOM maps in the MECshield agents are separately trained by different local IoT traffic." [13, p. 7]. The CPU usage of the devices indicated that MECshield has the lowest CPU usage that is 36%, whereas the centralized-SOM CPU usage is 45% [13].

Ko, Chambers, and Barrett [3] in their research proposed the best site to deploy an intrusion defense system would at the internet service provider site in case of DDoS

threats since ISP would be able to drop or block any malicious traffic being targeted at the victim. The paper uses a self-organizing map to train their model over a large number of unlabeled data traffic for data mining and feature extraction. The study trained a two-layered self-organizing map that would reduce the blocking of regular traffic that might cause service interruptions. The study aimed to increase the separability of data by taking advantage of additional information available at the ISP site. Based on the feature importance feature, the first layer of SOM was based on global octets per packet mean, global octets per packet standard deviation, local traffic count and so on. The second layer of the self-organizing map took into account features such as global unique protocol, source port, and destination port with local transfer count and so forth. The research concluded, "Deploying the mitigation system within the ISP domain offers a more effective solution, and our proposed hierarchical dual SOM has demonstrated to outperform the K-Mean model by 3.04% and the single SOM by 14.55% on the F1 score" [3, p. 582].

Literature Related to the Methodology

Tian, Azarian, Pecht [17] in their research, developed a new way to implement a self-organizing map to detect anomaly in data containing noise and SOM clusters that are non-convex. Traditionally, the self-organizing map has been used as anomaly detection purposes by taking the average of quantization error or finding the minimum quantization error. The authors describe the quantization error as "the distance between the input data observation and the BMU of the SOM." [17, p. 3]. According to the paper, finding the average of quantization error poses a problem since it assumes the best

matching unit in self-organizing maps to be convex and not sparse, which cannot be used if the best matching units are sparse and not dense. The other traditional method is finding the minimum quantization error, but it is sensitive to noise in the data, which could harm the SOM model. The solution to these two limitations - noise in the data and non-convex SOM clusters - has been proposed where the SOM is trained on healthy data containing little noise. The authors suggest when we fit the SOM trained on healthy data to test data, the nodes that are too sparse or fall under a minimum number of BMU threshold, the node is removed to avoid BMU contaminated with noise. A semi-supervised model (i.e., K-nearest neighbor) is used to classify the data based on the Euclidean distance between the centroids and the observation data points. Ultimately, once the healthy reference has been identified, the anomaly decision is made based on the measure taken by using 99.7 percentile or a standard deviation of 2.7 from the healthy reference.

For this thesis, the method proposed by Tian et al. [17] in their research will be implemented as a hybrid semi-supervised model to detect botnets in our dataset. The current study will use *the detection of IoT botnet attacks dataset* as the training set. The data set includes a benign traffic data which can be used to train as a healthy reference. Once we have trained our SOM based K-nearest neighbor model, we will make our outlier decision based on 99.7 percentile and standard deviation of 3 as a measure for possible botnet traffic.

The papers discussed in this chapter adequately discusses the limitations of these models, one of the main concerns are the implementations of the deep learning

models at the edge of a network for smart IoT malware detection as opposed to detection at a central server. The papers also discuss the inherent threat that IoTs brings to the malware landscape and how relying on the IoT manufacturers for intrusion prevention is not reliable. To this end, one of the aims of this study is to develop a deep learning prototype that computational practical to be deployed at the edge of the network.

Chapter III: Methodology

The methodology can be outlined in the preprocessing stage, training stage, and testing stage. In the preprocessing stage, we will prepare the data so our classifiers are trained and tested while preventing overfitting and multicollinearity. The preprocessing phase implements the sklearn library's `MinMaxScaler` and `normalizer` to scale the input values. The algorithm calculates the mean and standard deviation of the independent variables and gets them centered around 0, keeping the standard deviation to 1. The `MinMaxScaler` is an effective algorithm to scale all the independent values to achieve normal distribution. Once we have achieved feature scaling, we will label encode our categorical features by using pandas's `get_dummies` function. The `get_dummies` function is an effective way to label encode the nominal categorical independent variables in numerical form. The label encoding function creates a new dataframe that contains zeros and ones so it can be quantified and implemented in our deep learning model. To avoid overfitting in our model, we will implement Sklearn's `extra-trees` classifier. This is an ensemble learning method that creates subsets of the dataset, fits randomized decision trees, and uses averaging to decrease variance and improve predictions of our models. The output helps us remove independent features that contribute to overfitting and keep the independent variables that improve the prediction power of our models.

Once we have preprocessed our datasets to reduce dimensionality and scale the data, the two deep learning prototypes will be trained on the *NSL-KDD* dataset and *Detection of IoT botnet Attacks* dataset. The main purpose of using the NSL-KDD

dataset in this study is so it can be used as a benchmark to make our results comparable for future studies. The supervised and semi-supervised models will be trained on two predetermined labeled datasets. The labeled dataset has dependent variable classifying values to either normal traffic and malicious traffic. The semi-supervised model will be evaluated based on an anomaly metric that will conclude the percentage of malicious data that was correctly determined to be an outlier. The anomaly metric determined by calculating the distance between the data observations and the K-nearest neighbors centroids of the observations, similar to the work done by [17] and [18]. The anomaly threshold is determined from the benign traffic by summing the mean with a standard deviation of three. The metrics this study rely on to validate the performance of supervised deep learning model confusion matrix, accuracy score, recall score, and precision score. K-fold cross-validation will be used to test for variance and bias to examine overfitting.

Data Analysis

The training set in this paper refers to the data set that would be used to train our models. The training set includes the NSL-KDD dataset and the Detection of IoT botnet Attacks dataset, which is a simulated network trace of the Mirai attack available on the UCI repository for reproducibility.

NSL-KDD dataset

NSL-KDD is deemed as a replacement of the KDD-99Cup, where the intrusion detection training dataset is involved. NSL-KDD is a subset of its predecessor and has near-even distribution of normal and attack traffic. The NSL-KDD training dataset in this

study include 41 features, and unlike its predecessor, does not contain redundancies in its records. The major limitation of the dataset is it does not identify the hosts/systems under attack. As illustrated in Table II, the total amount of observations in the KDD Train+ dataset contains 125973 records, out of which 53.45% is normal traffic, and 46.55% observations are network attacks. As shown in Table III, total observation for KDD Test+ dataset is 22544, where 43.07% are normal, and 57.03% are attack traffic [19].

Table II

Subclasses of intrusion attacks and their frequencies in the NSL-KDD Train+ dataset

Sub-classes	Train+ Percentage
normal	53.458%
neptune	32.717%
satan	2.884%
ipsweep	2.857%
portsweep	2.327%
smurf	2.101%
nmap	1.185%
back	0.759%
teardrop	0.708%
warezclient	0.707%
pod	0.160%
guess_passwd	0.042%
buffer_overflow	0.024%
warezmaster	0.016%
land	0.014%
imap	0.009%
rootkit	0.008%
loadmodule	0.007%
ftp_write	0.006%
multihop	0.006%
phf	0.003%
perl	0.002%
spy	0.002%

Table III

Subclasses of intrusion attacks in the NSL-KDD Test+ dataset

Subclasses	NSL-KDD Test+ Percentage
normal	43.076%
neptune	20.657%
guess_passwd	5.460%
mscan	4.418%
warezmaster	4.187%
apache2	3.269%
satan	3.260%
processtable	3.039%
smurf	2.950%
back	1.592%
snmpguess	1.468%
saint	1.415%
mailbomb	1.300%
snmpgetattack	0.790%
portsweep	0.696%
ipsweep	0.625%
httptunnel	0.590%
nmap	0.324%
pod	0.182%
buffer_overflow	0.089%
multihop	0.080%
named	0.075%
ps	0.067%
sendmail	0.062%
rootkit	0.058%
xterm	0.058%
teardrop	0.053%
xlock	0.040%
land	0.031%
xsnoop	0.018%
ftp_write	0.013%
loadmodule	0.009%
perl	0.009%
worm	0.009%
phf	0.009%
udpstorm	0.009%
sqlattack	0.009%
imap	0.004%

Detection of IoT botnet Attacks Dataset

Meidan et al. [20] during their research infected nine IoT devices (i.e., doorbell, thermostat, baby monitor, security camera, and webcam) with Mirai and BASHLITE botnets. The researchers made the trace traffic of the dataset available on the University of California Irvine online repository [20]. The paper expounds on the dataset collection method. The data collection method is explained by the authors, “We capture the raw network traffic data (in pcap format) using port mirroring on the switch through which the organizational traffic typically flows.” [20, p. 3]. The data collection step is followed by feature extraction where snapshots of the hosts and protocols are taken. The snapshot resulted in 115 traffic statistics, which are aggregated by the source IP. The second way the data is aggregated is by determining the source of MAC address and IP address to find a distinction between normal traffic and spoofed IP address. Thirdly, the traffic statistics are aggregated by the source and destination of TCP or UDP ports. Lastly, the data statistics are aggregated by source and destination IPs. To understand the traffic trace dataset from this paper the attacks executed are by expounded by the authors. The paper provides a list of attacks that were executed to infect the 9 IoT devices as illustrated in Table IV.

Table IV

Dataset Properties and Training Summary [20]

Device ID	Device Make and Model	Device Type	Number of Benign Instances	Training Time (sec)	Object size (kB)
1	Danmini	Doorbell	49,1548	555	172
2	Ennio	Doorbell	39,100	215	172
3	Ecobee	Thermostat	13,133	54	172
4	Philips B120N/10	Baby Monitor	175,240	292	172
5	Provision PT-737E	Security Camera	62,154	275	172
6	Provision PT-838	Security Camera	98,514	795	172
7	SimpleHome XCS7-1002-WHT	Security Camera	46,585	220	172
8	SimpleHome XCS7-1003-WHT	Security Camera	19,528	190	172
9	Samsung SNH 1011 N	Webcam	52,150	150	172

While testing BASHLITE botnet, scan attempts were executed to find the vulnerability. Spam data attacks were carried out in the form of junk. UDP flooding and TCP flooding for simulating denial of service attacks. Lastly, a combination of spam and connection attempts was made towards specific IP addresses and ports. Similarly, while testing Mirai botnet, devices were scanned for vulnerability. Ack, Syn, UDP, UDPplain

flooding attacks were initiated to test Mirai botnets, which are included in the traffic trace dataset [20].

Performance indicator: Confusion Matrix

The confusion matrix, similar to figure 1, is a widely used evaluation method for measuring the performance of machine learning models. The confusion matrix conveys the number of true positives, true negatives, false positives, and false negatives in the results of our model [6]. False Positive (also called type 1 error) is when our model predicts value to be anomalous but is normal. A false negative is when our model predicts a value to be normal but is anomalous. True positive is when our model predicts a value to be anomalous, and the prediction is correct. True negative is when our model predicts a model to be normal, and the prediction is correct [6]. The dependent variable in our study has been label encoded to binary classification where 0 is labeled as normal traffic, and 1 is labeled as anomalous traffic.

		Confusion Matrix	
		Predicted Normal (0)	Predicted Anamoly (1)
Actual Normal (0)	True Positive	False Positive (Type I Error)	
Actual Anomaly (1)	False Negative (Type II Error)	True Negative	

Figure 1. Confusion Matrix.

Performance indicator: Accuracy

Accuracy measures the overall percentage of values that where our predictions were correct [7]. The accuracy score is determined once we fit our model on the training set and make predictions on the predetermined test set. Mathematically put:

$$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative}$$

During multiclass classification, the accuracy will be determined by computing the subset accuracy of each class.

Precision: Measures the result relevancy. In the research's context, this will tell us the number of correct predictions about the malicious traffic the model correctly was able to classify or detect. The model's chosen outcome is, in fact, the true outcome based on the label provided by the datasets during the test phase [7].

Mathematically put: $Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$.

Out of the total positive results that are predicted, what would be the percentage is the real positive results.

Recall: In the research context, the model's prediction was incorrect, and the traffic is, in fact, malicious [7]. Mathematically would be represented as:

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Performance indicator: Feature Score (also called F1-Score)

F1 score would be a measure of classification model's usefulness, which is obtained through taking the harmonic mean between precision and recall [7]. The score is between 0 and 1. The higher the f1 score entails the high predictive power of the classification model.

Mathematically represented as: $F1 = 2 \times \frac{Precision * Recall}{Precision + Recall}$.

Performance indicator: K-Fold Cross Validation

In order to avoid overfitting in our model, we implement cross validation. We will use the K-Folds cross validation technique to split our data in k number of subsets, we train our model on the k number of subsets and retain the last subset for validation purpose [21]. This validation technique is done k number of times, and eventually, the results are averages in an estimated. In other words, it is a resampling procedure that splits up the dataset in K number of groups and validates groups of train/test splits within a dataset. We can gather the bias and variance present in the results by taking the average and standard deviation of all train-test combinations of k-folds. After we compute the standard deviation of the accuracies generated by k-fold, we can determine if the standard deviation is high enough to signal the presence of overfitting in our model prediction.

Performance indicator: CPU usage

Python's time module will be used to measure the total time it takes to build our training model. The time function shows the total number of seconds it takes for an epoch to be carried out. Calculating the epoch is crucial since in the machine learning

context, an epoch is when our entire dataset is forward and backward propagated through our neural network. Another important way to measure the computation cost of our model is to measure the time it takes to predict dependent variables on the test data.

First Layer of Semi-supervised Model: The Self-organizing Map

The self-organizing map is an unsupervised artificial neural network algorithm implemented for clustering and visualizing data with a high number of dimensions into a topology with far fewer dimensions (generally two dimensions). The algorithm architecture does not contain a hidden layer nor backpropagation like traditional neural networks. In self-organizing maps, the training set dimensions are the input nodes; in other words, each dimension becomes separate input nodes [22]. An important characteristic of SOM architecture is that each output node contains coordinates in relation to the input node, figure 2 shows a schematic representation. Direct mapping is produced when the input nodes that have the closest Euclidean distance to the output node [23].

The mathematical formula for the distance is:

$$Distance: \sqrt{\sum (x_i - w_{1i})^2}$$

w_i represents the weights of each output node. x_i represents the input nodes containing input values from our dataset. After the difference of input and output node is squared and summed, the algorithm takes the square root of the result to measure the distance. The output node having a small distance is considered the best matching unit.

In other words, the output node that has the closest distance to the input node is considered the best matching unit [24]. During the training process, the node updates its weight vectors to move closer to the input node. The algorithm also contains neighbor functions that affect the nodes near the best matching unit to move closer as well; this is how the nodes like each other are clustered together on the two-dimensional map [23].

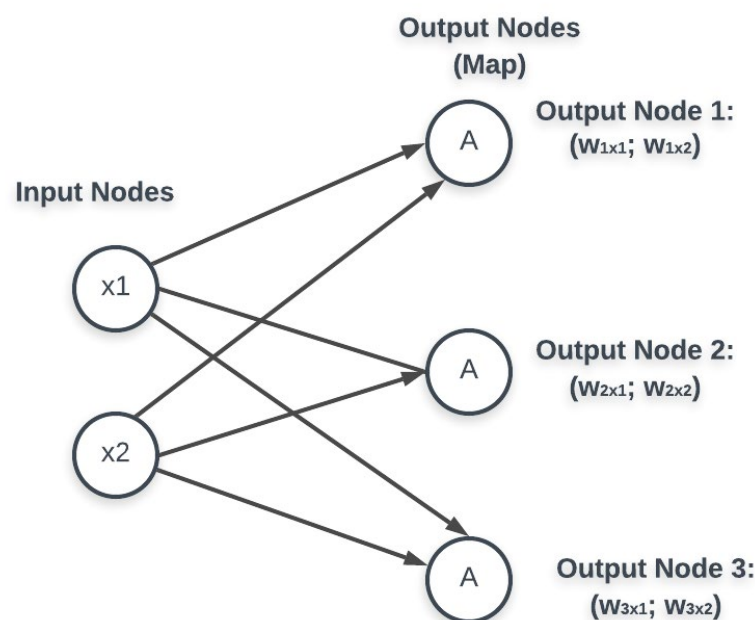


Figure 2. Simplified Self-organizing Map Architecture.

The self-organizing map is easier and intuitive to understand relative to other deep learning algorithms since it focuses on a visual representation of nodes and relationships to their neighbor nodes. The self-organizing map is a robust algorithm that has applications in modern machine learning challenges, including dimension reduction, pattern recognition, image processing, so on and so forth [22].

Second Layer of Semi-supervised Model: K-nearest neighbor

K-nearest neighbor algorithm is a classification machine learning algorithm that classifies a data point based on the distances of its nearest data points. The number of data points nearest is considered when making classification distance. Based on the number of neighboring data points, the new data point is classified [25].

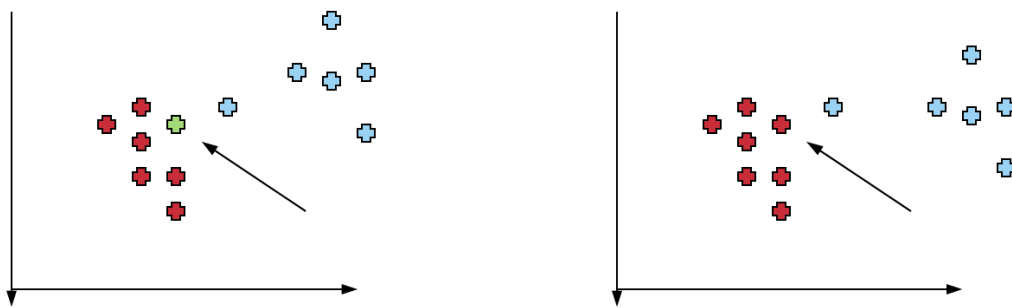


Figure 3 Simplified representation of K-Nearest Neighbor Model.

In figure 3, the green data point represents a new data point as plotted on a two-dimensional chart. Based on the nearest neighbors (closest data points) of the input data point (green), the k-nearest neighbor algorithm classifies it to be red. The classification is based on the distance between datapoint and the count of the data points. The first step to implementing the model, we select the number of K neighbors we take into consideration when classifying. The second step is considering the nearest neighbors based on the Euclidean distance. The third step is considering the number of data points in each category. In the final step, we assign the new data point to the class that has the most neighbors based on the Euclidean distance [25].

Supervised Model: Artificial Neural Network

A supervised learning algorithm consists of an input layer, a hidden layer, and output layer. The neural network learns through synapses, which are assigned weights that are adjusted based on backpropagation and activation function [26], and [27].

Based on the weights, the neural network decides what signals are passed through to achieve higher accuracy in classification problems.

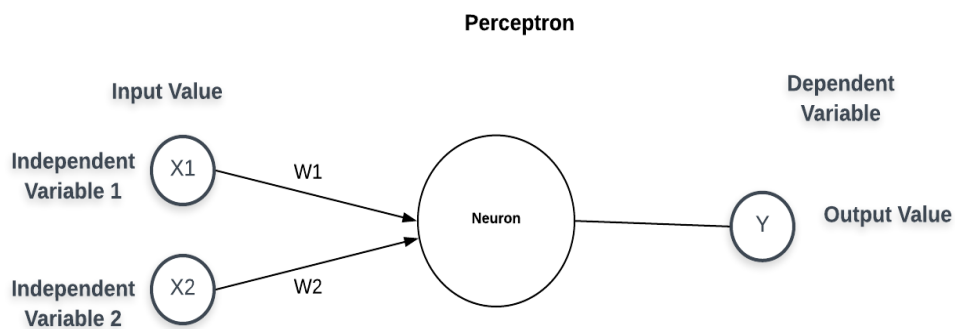


Figure 4. Visual representation of perceptron.

The structure of perceptron, as shown in figure 4, shows that the input variables x_i will be multiplied with weights w_i . The sum of these multiplications is passed through a non-linear activation function ϕ .

$$\hat{y} = \phi \left(\sum_{i=1}^m x_i w_i \right)$$

The activation function we will use in this study is the sigmoid function. The sigmoid function is a non-linear activation function ϕ that takes an input $(\sum_{i=1}^m x_i w_i)$ and converts it into scalar output between 0 and 1 [28].

Sigmoid is mathematically represented as:

$$\phi(x) = \frac{1}{1 + e^{-x}}$$

Backpropagation is a process where the artificial neural network learns from its predicted output \hat{y} by comparing it with the actual output y . The weights are updated with the use of cost function C [29]. Mathematically, cost function can be denoted as:

$$C = \frac{1}{2}(\hat{y} - y)^2$$

The forward and backward propagation process is updated iteratively over the training values to keep adjusted weights based on the cost function C and predicted output \hat{y} get a better prediction from the artificial neural network.

Design of the Study

The framework this study will subscribe to is purely quantitative, where the focus is a robust way to validate the botnet detection performance of our supervised and semi-supervised deep learning models. The framework this study will subscribe to is purely quantitative, where the focus is a robust way to validate the botnet detection performance of our supervised and semi-supervised deep learning models. In figure 5, the design entails importing the Mirai Botnet attack dataset and NSL-KDD datasets into our python environment. The environment that we for this research is Spyder 4.0.0. Once our datasets have been imported, the data will be preprocessed to normalize our data to avoid computational overhead and maintain normal distribution in our datasets. The extra trees classifier will reduce dimensionality in our data to remove noise to avoid bias and variance in our data. Once our data has been preprocessed, we will split our data into training and test subset. The training subset is used to train our models where

the test subset is used to validate our models' predictive accuracy. The results of semi-supervised model will be evaluated based on anomaly metric, f1-score, precision score, recall score and accuracy score.

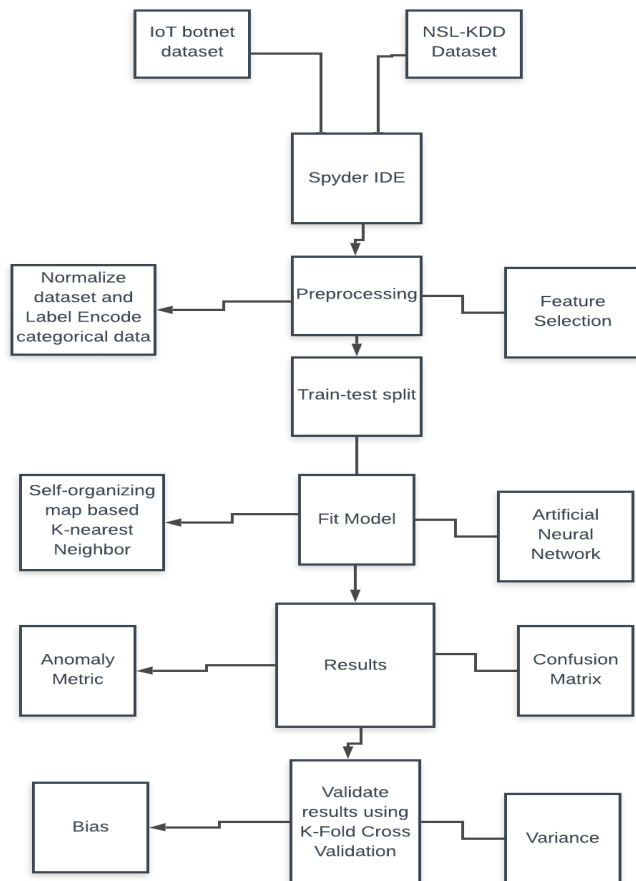


Figure 5. The methodology design.

Tools and Techniques

Python statistical libraries were employed in this study to build, test, and evaluate the deep learning models. Numpy, pandas, scikit-learn, sklearn python libraries were imported into the Spyder Python IDE to conduct this study [30], [31], and [32]. Sklearn

library was employed to validate our results by measuring confusion matrix, recall score, accuracy score, f1 score, and precision score. The standard scaler feature from the Sklearn library was used to separate mean and scale observations to unit variance.

Keras library was used to initiate our deep learning classifier and used to add the input layer, hidden layer, and outer layer for the artificial neural network [33]. The neural network was fitted to the dataset by using Keras library. It was also used for K-fold cross-validation to measure the variance and bias in our models. The grid-search, with the help of Keras, was used to establish best practices by optimizing for best parameters in the supervised models in this study. The self-organizing map were implemented with MiniSom [34] and the self-organizing map based K-nearest neighbor algorithm was implemented by *SOM anomaly detector* [35].

Hardware and Software Environment

The laptop used for this study is an Ideapad 330S. Processor: Intel® Core i5-8250U CPU @ 1.60GHZ 1.80 Ghz Installed RAM: 8.00 GB. Python environment used for this study is Spyder 4.0.0 with Python 3.7.5 version installed.

Chapter IV: Results

Data Preprocessing

ANN on NSL-KDD. The KDDTrain+ dataset was imported to Spyder 4.0 IDE, where it was preprocessed using NumPy, pandas, and sklearn libraries. The data frame, once imported, were preprocessed by label encoding the outcome class to 0, 1, 2, 3, and 4 where 0 represented normal traffic, 1 represented probe, 2 represented root to local attacks, 3 represented denial of service attacks, and 4 represented User to Root attacks. This preprocessing step was used for multiclassification using an artificial neural network. The dataset contained categorical features such as protocol type, service, and flags, which were preprocessed by converting categorical variables into dummy variables. The conversion is necessary, so the ANN model can process the data to get a successful final ANN model. Train_test_split function from the sklearn library was employed to split the dataset into train test subsets; the parameter for test size was set to 25% for validation. The dataset values were normalized using the normalize function, which scales each value to unit norm.

ANN on IoT botnet Attacks Dataset. In the preprocessing stage, each class labels in the dataset were assigned outcome variable as 0, 1, 2, 3, 4. Benign data was assigned 0, malign ACK traffic was assigned 1, malign SCAN traffic was assigned 2, malign SYN traffic was assigned 3, and malign UDP traffic was assigned 4. In the preprocessing stage, each traffic dataset was assigned with an outcome variable as 0, 1, 2, 3, 4 for classification purposes. Benign data was assigned 0, malign ACK traffic was assigned 1, malign SCAN traffic was assigned 2, malign SYN traffic was assigned

3, and malign UDP traffic was assigned 4. The dataset for each traffic type was organized using *vstack* and *hstack* function from the Numpy library. Extra trees classifier was used to reduce dimensionality in our features to control overfitting. The dataset was preprocessed through the train test split, where the test size parameter was set to 25%. Standard scaler function was imported from the Sklearn library to speed up the training speed of the model and standardized all the input data for our model. The backend, sequential, dense, and dropout packages were imported from the TensorFlow Keras library to build out the artificial neural network. The sequential package from Keras was used to initialize our ANN model; then, the classifier and the dense package were added the input layer and the first hidden layer. The parameter for our input and the hidden layer was set to uniform for the kernel initializer parameter, and the rectified linear unit option was used for the ANN activation function parameter. The dropout layer was added to regulate the input of our deep neural network, where the drop out rate is set to 0.2 to control overfitting. The second hidden layer parameters had the units set to 21, kernel initializer is the uniform function, and the activation function is the rectified linear unit. In the output layer, the units' parameter is 5, so our model can classify our multiclass problem. Additionally, in the output layer SoftMax activation function is implemented since it assigns decimal probabilities to the multiclass outcomes. Adam argument was selected to compile the ANN model.

Building the Hybrid SOM model on IoT botnet Attacks Dataset

In the preprocessing step, outcome variables were declared and labeled benign traffic as 0, malign ACK traffic as 1, malign SCAN traffic as 2, Malign SYN traffic as 3,

and malign UDP traffic as 4. The values in the dataset were normalized using normalize function imported from the Sklearn library as it yielded better results compared to MinMaxScaler and Standard Scaler. Extra trees classifier was used to reduce feature size in our dataset to improve the predictive power of our final model and to limit overfitting. For the optimal predictive performance of our model, parameter tuning was performed using Bayesian optimization from the Hyperopt library. Our Hybrid SOM model was initialized by setting the learning rate to 5, learning decay parameter to 0.003, initial radius parameter to 10, radius decay parameter to 0.019, minimum number per best-matching unit parameter to 5, number of neighbors parameter set to 2. The parameters for the hybrid SOM model are based on the results obtained from Bayesian optimization-based hyperparameter tuning. The anomaly detector is fitted on the benign training data, and the number of iterations was set to 5000.

Chapter V: Discussion and Conclusion

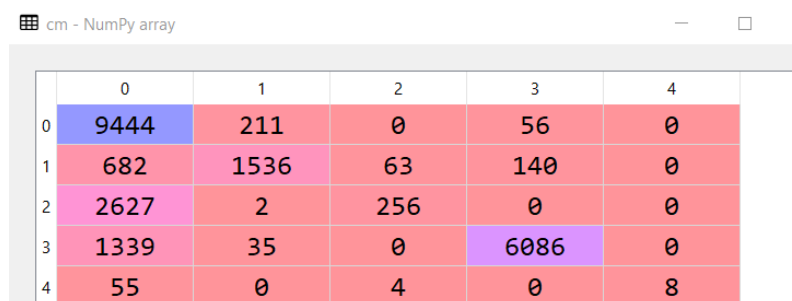
Artificial Neural Network Performance on NSL-KDD dataset

In the multiclass model, the output layer units dimension space is set to 5 to get non-binary output, and the loss function parameter is set to sparse categorical cross-entropy since our classes are mutually exclusive.

```
125973/125973 [=====] - 23s 183us/step - loss: 0.0614 - acc: 0.9823
Epoch 2/3
125973/125973 [=====] - 22s 177us/step - loss: 0.0330 - acc: 0.9905
Epoch 3/3
125973/125973 [=====] - 22s 174us/step - loss: 0.0298 - acc: 0.9916
67.81152749061584 seconds
```

Figure 6. The training of ANN model on NSL KDD Test+ dataset over 3 Epochs.

Figure 6 shows the model has been successfully trained on the NSL KDD train+ dataset. The trained model makes a multiclass prediction on the NSL-KDD test+ dataset. The multiclass prediction is demonstrated in figure 7, which is a confusion matrix generated through the Sklearn library.



	0	1	2	3	4
0	9444	211	0	56	0
1	682	1536	63	140	0
2	2627	2	256	0	0
3	1339	35	0	6086	0
4	55	0	4	0	8

Figure 7. The Multiclass Confusion Matrix for NSL-KDD test+.

Table V

Artificial Neural Network performance on NSL-KDD

Class	True Positive	True Negatives	False Positives	False Negatives
Normal	9444	8130	4703	267
Probe	1536	19875	248	885
R2L	256	19592	67	2629
DOS	6086	14888	196	1374
U2R	8	22477	0	59

Based on the metrics gathered from Table V, the model's predictive power for determining true normal traffic observations are underperforming, which shows when observing a precision score in Table VI.

Table VI

Artificial Neural Network Subclass Performance on NSL-KDD

Class	Accuracy (%)	Precision Score (%)	F1 Score (%)	Recall (%)
Normal	77.95%	66.8%	79.2%	97.3%
Probe	94.97%	86.1%	73.1%	63.4%
R2L	88.04%	79.3%	16.0%	8.9%
DOS	93.03%	96.9%	88.6%	81.6%
U2R	99.73%	100%	21.3%	11.9%
Weighted Average		80.5%	73.4%	76.9%
Multiclass Subset Accuracy score	76.87%			

In the results shown in Table VI, the model has demonstrated greater accuracy in DOS attacks and User2Root attacks. The model has also shown a high number of false negatives for root to local attacks.

Table VII

Stratified K-fold Cross Validation over 10 iterations

Iterations of CV	Accuracies (%) on KDDTest+
1	90.11%
2	89.84%
3	89.09%
4	88.91%
5	89.08%
6	90.28%
7	90.37%
8	88.46%
9	90.15%
10	88.95%
Accuracies Mean	90.11%
Accuracies Variance	0.658%

In Table VII, the stratified K-fold cross validation shows high accuracy over 10 iterations with an average mean of 90.11% and a low variance of 0.658%.

Artificial Neural Network Performance on IoT botnet Attacks Dataset

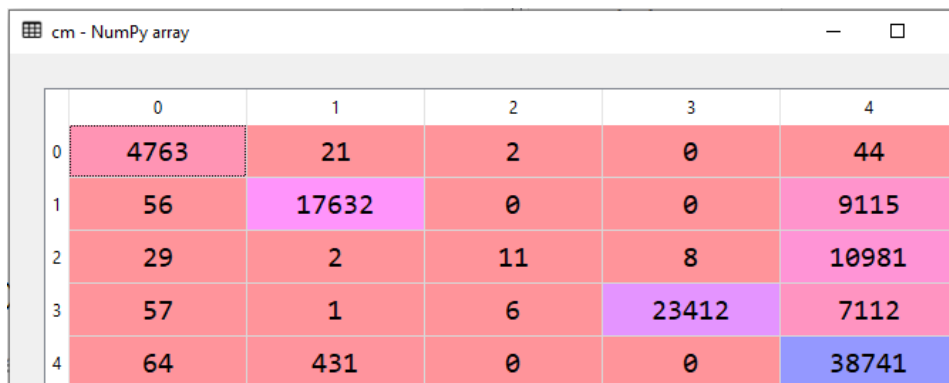
```

Epoch 1/3
337464/337464 [=====] - 42s 126us/sample -
loss: 0.6934 - acc: 0.637240/337464 [=====>.....] -
ETA: 30s - loss: 0.7681 - acc: 0.5947
Epoch 2/3
337464/337464 [=====] - 46s 137us/sample -
loss: 0.5738 - acc: 0.7216
Epoch 3/3
337464/337464 [=====] - 48s 141us/sample -
loss: 0.5447 - acc: 0.7317
136.87066388130188 seconds

```

Figure 8. Training of ANN model on IoT Botnet Attacks Dataset over 3 Epochs.

The ANN model was trained over the IoT botnet Attacks Dataset by train-test split, where the 25% of the dataset was used as test size. The model was trained over three iterations and took 136.87 seconds.



	0	1	2	3	4
0	4763	21	2	0	44
1	56	17632	0	0	9115
2	29	2	11	8	10981
3	57	1	6	23412	7112
4	64	431	0	0	38741

Figure 9. The Multiclass Confusion Matrix IoT Botnet Dataset.

Table VIII

Artificial Neural Network performance on Mirai Botnet Dataset

Class	Accuracy (%)	Precision Score (%)	F1 Score (%)	Recall (%)
Benign	99.75%	95.9%	97.2%	98.6%
ACK	91.44%	97.5%	78.6%	65.8%
Scan	90.19%	57.9%	0.2%	0.1%
SYN	93.61%	100%	86.7%	76.5%
UDP	75.33%	58.7%	73.6%	98.7%
Weighted Average		80.7%	72.2%	75.2%
Multiclass Subset Accuracy score	75.17%			

In Table VIII and Table IX, we observe the predictive power of the model to determine benign, Scan, and SYN traffic. However, the model has high number of false positives in ACK and SCAN traffic compared to its class counterparts.

Table IX

Artificial Neural Network performance on Mirai Botnet Dataset

Class	True Positive	True Negatives	False Positives	False Negatives
Normal	4763	107452	206	67
ACK	17632	85230	455	9171
SCAN	11	101449	8	11020
SYN	23412	81892	8	7176
UDP	38741	46000	27252	495

Table X

Stratified K-fold Cross Validation over 10 iterations

Iterations of CV	Accuracies (%)
1	0.729843
2	0.747711
3	0.768424
4	0.741903
5	0.723552
6	0.755734
7	0.752119
8	0.749393
9	0.757779
10	0.728383
Accuracies Mean	74.54%
Accuracies Variance	1.370%

In Table X, the stratified cross validation indicates a 74.54% mean accuracy score and low variance score of 1.370 over 10 iterations of the ANN model on the Mirai Botnet dataset demonstrating low bias in our trained model.

Hybrid SOM Performance on NSL-KDD

When fitting the hybrid SOM model to our benign dataset that was separated during the preprocessing stage, the parameters were adjusted based on hyperparameter tuning that uses Bayesian optimization, which was implemented using

hyperopt library. Figure 10 shows the final hyperparameter tuning for our anomaly detection model.

```
anomaly_detector = AnomalyDetection(shape=(39,39), input_size=training.shape[1],
                                   learning_rate=9.800275263995829,
                                   learning_decay=0.0029030131893884627,
                                   initial_radius=4.13960012604465,
                                   radius_decay=0.0012626284950839894,
                                   minNumberPerBmu=7.304609930044314,
                                   numberOfNeighbors=2)
```

Figure 10. Parameters for Hybrid SOM initialized and fitted on NSL-KDD Test+.

The *shape* parameter represents the shape of the SOM grid that is made up of x number of rows, and y represents the number of columns allotted to our SOM nodes. The sigma means the spread of the neighborhood function, and this parameter directly affects how the neighboring neurons of the winning nodes will learn from each iteration. The learning rate decides the amount of change that is applied to the self-organizing map after each epoch; the learning rate also exponentially decay after each iteration. The initial radius parameter entails the nodes included within the radius of the BMU initially, and this parameter also diminishes each iteration through exponential decay. The number of neighbors parameter adjusts our K-NN model, in our hybrid framework, to take into account the number of cues near a given data point when classifying. The anomaly detector is trained by fitting it on the benign dataset, and the number of iterations is set to 5000. During our multiclass classification of the NSL-KDD dataset, the subclass accuracy performance is illustrated in Table XI.

Table XI

Accuracies metrics after fitting the previous fitted network on evaluation data

Subclasses	Accuracies (%) on KDDTrain+	Accuracies(%) on KDDTest+
Probe	92.08%	39%
R2L	18.39%	92.812%
DOS	98.377%	2.911%
U2R	17.307%	98.507%

The subclass accuracy performance on Table X demonstrates the high accuracy performance of Hybrid SOM on user to root traffic and root to login traffic when the predictions are mapped on KDDtest+ dataset.

Hybrid SOM Performance on IoT botnet Attacks Dataset

In figure 11, the parameters for hybrid self-organizing maps were selected based on the Bayesian hyperparameter tuning strategy for optimal performance.

```
anomaly_detector = AnomalyDetection(shape=(47, 47), input_size=training.shape[1],
                                   learning_rate=5,
                                   learning_decay=0.003,
                                   initial_radius=10,
                                   radius_decay=0.019,
                                   minNumberPerBmu=5,
                                   numberOfNeighbors=2)
```

Figure 11. Parameters for Hybrid SOM initialized and fitted on the IoT Botnet Attacks dataset.

The limit value is used to determine whether an observation is deemed an anomaly is ascertained by adding the mean and standard deviation of 3. Each

observation is determined to be an outlier if the values in the anomaly metrics are higher than the previously determined limit value. The subclass anomaly score is determined by taking the percentage of the total amount of outlier determined by the model over the total observations in the evaluation data.

Table XII

Accuracies metrics after fitting the previous fitted network on evaluation data

Subclasses	Accuracies (%) on Mirai Botnet Dataset
ACK	32.64%
Scan	99.61%
SYN	23.57%
UDP	28.32%

As suggested by results illustrated in Table XII, the Hybrid self-organizing map has performed significantly better when detecting SCAN attack traffic as opposed to its counterparts such as ACK attacks, SYN attacks, and UDP attacks.

Table XIII

ANN model and Hybrid SOM model performance on NSL-KDD dataset

Subclass	Iterations to train Hybrid SOM	Epoch	Train Time for ANN (s)	Train Time for Hybrid SOM (s)	Accuracies (%) on ANN model	Accuracies (%) on Hybrid SOM
Probe	10,000	3	67.811 seconds	39.239 seconds	94.97%	100%
R2L	10,000	3	67.811 seconds	39.239 seconds	88.04%	6.204%
DOS	10,000	3	67.811 seconds	39.239 seconds	93.03%	86.756%
U2R	10,000	3	67.811 seconds	39.239 seconds	99.73%	100%

The Hybrid SOM model has shown higher predictive power in determining Probe attack traffic and user to root attack. In contrast, the Hybrid SOM model has underperformed in detecting root to login attack traffic. However, the traditional artificial neural network requires high CPU resources while training the model, which can prove to be detrimental in the context of setting it up in IoT devices. Table XIII illustrates that lightweight Hybrid SOM gets fully trained over 10,000 iterations in 39.239 seconds and can outperform the traditional ANN model in detecting probe traffic and user to root

attack traffic. The train time suggests that the Hybrid SOM model is the preferable choice over the conventional neural network for lightweight anomaly detection purposes for network attacks in IoT devices.

Table XIV

ANN model and Hybrid SOM model performance on IoT botnet Dataset

Subclass	Total iterations to train Hybrid SOM	Total ANN Epochs	Train Time for ANN (s)	Train Time for Hybrid SOM (s)	Accuracies (%) on ANN model	Accuracies (%) on Hybrid SOM
ACK	5000	3	54.804 seconds	27.312 seconds	30.27%	32.64%
Scan	5000	3	54.804 seconds	27.312 seconds	99.63%	99.61%
SYN	5000	3	54.804 seconds	27.312 seconds	25.04%	23.57%
UDP	5000	3	54.804 seconds	27.312 seconds	30.30%	28.32%

As per the results illustrated in Table XIV, for Botnet IoT attacks, both the ANN model and lightweight Hybrid SOM predictive accuracy were very close to each other. In our experiment, the Hybrid self-organizing map outperformed the ANN model when predicting ACK traffic attacks. In contrast, SOM hybrid predictions for SCAN attacks, SYN attacks, and UDP attacks were almost at par with the ANN model, which takes nearly twice as long time to train. The SOM model took 37.312 seconds to fully train over 5000 iterations while the ANN model took 54.804 seconds.

Table XV

Hybrid SOM on NSL-KDD after Scaling Down Nodes over 10000 iterations

Subclass	Total Nodes shape Parameter	Total Nodes shape Parameter	Train time for Hybrid SOM	Train time for SOM scaled down	Accuracies (%) on Hybrid SOM	Accuracies (%) on Scaled Down Hybrid SOM
Probe	(36, 36)	(20, 20)	39.239 seconds	8.963 seconds	100%	81.908%
R2L	(36, 36)	(20, 20)	39.239 seconds	8.963 seconds	6.204%	1.941%
DOS	(36, 36)	(20, 20)	39.239 seconds	8.963 seconds	86.756%	87.305%
U2R	(36, 36)	(20, 20)	39.239 seconds	8.963 seconds	100%	100%

Table XV indicates that after scaling down the self-organizing map's nodes for resource optimization, the subclass accuracy was comprised of probe attack traffic, remote to login attack traffic. The scaled-down Hybrid SOM did outperform the Bayesian optimized hybrid SOM when predicting denial of service attacks. The training time was significantly reduced from 39.239 seconds to 8.963 seconds after changing the total self-organizing maps node grid shape from (36, 36) to (20, 20) rows and columns.

Table XVI

Hybrid SOM on IoT Dataset after Scaling Down Nodes over 10000 iterations

Subclass	Total Nodes shape Parameter	Total Nodes shape Parameter	Train time for Hybrid SOM	Train time for SOM scaled down	Accuracies (%) on Hybrid SOM	Accuracies (%) on after Scaled Down Hybrid SOM
ACK	(47, 47)	(24, 24)	27.312 seconds	4.900 seconds	32.64%	25.27%
Scan	(47, 47)	(24, 24)	27.312 seconds	4.900 seconds	99.61%	99.52%
SYN	(47, 47)	(24, 24)	27.312 seconds	4.900 seconds	23.57%	23.55%
UDP	(47, 47)	(24, 24)	27.312 seconds	4.900 seconds	28.32%	25.60%

The comparison, as illustrated in Table XVI, suggests that scaling down the number of SOM nodes by almost half does not have a meaningful impact on the subclass accuracy measures and reduces the training time significantly. The scaled-down self-organizing map has a training time of 4.900 seconds, making it ideal for deep-learning based detection of IoT attacks in a lightweight resource environment such as IoT based devices.

References

- [1] OWASP, "OWASP Internet of Things Project," 2018. [Online]. Available: owasp.org/index.php/OWASP_Internet_of_Things_Project. [Accessed 12/10/2019 December 2019].
- [2] Y. M. Mahardhika, A. Sudarsono and A. Barakbah, "Botnet Detection Using On-line Clustering with Pursuit Reinforcement Competitive Learning (PRCL)," *EMITTER International Journal of Engineering Technology*, vol. 6, no. 1, pp. 1-21, 2018.
- [3] I. Ko, D. Chambers and E. Barrett, "Unsupervised learning with hierarchical feature selection for DDoS mitigation within the ISP domain," *ETRI Journal*, vol. 41, no. 5, pp. 574-584, 2019.
- [4] B. Herzberg, I. Zeifman and D. Bekerman, "Breaking Down Mirai: An IoT DDoS Botnet Analysis," Imperva, 26 October 2016. [Online]. Available: <https://www.imperva.com/blog/malware-analysis-mirai-ddos-botnet/>. [Accessed 11 December 2019].
- [5] J. Fruhlinger, "The Mirai botnet explained: How teen scammers and CCTV cameras almost brought down the internet," CSO, 9 March 2018. [Online]. Available: <https://www.csoonline.com/article/3258748/the-mirai-botnet-explained-how-teen-scammers-and-cctv-cameras-almost-brought-down-the-internet.html>.
- [6] S. A. Aljawarneh, M. Aldwairi and M. O. B. Yassein, "Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model," *Journal of Computational Science*, vol. 25, pp. 152-160, 2018.
- [7] H. Hindy, D. Brosset, E. Bayne, A. Seeam, C. Tachatzis, R. C. Atkinson and X. J. A. Bellekens, "A Taxonomy and Survey of Intrusion Detection System Design Techniques, Network Threats and Datasets," *ArXiv*, vol. 1806.03517, 2018.
- [8] M. A. Al-Garadi, A. Mohamed, A. Al-Ali, X. Du and M. Guizani, "A Survey of Machine and Deep Learning Methods for Internet of Things (IoT) Security," *ArXiv*, vol. abs/1807.11023, 2018.

- [9] J. Cloonan, "Advanced Malware Detection - Signatures vs. Behavior Analysis," *Infosecurity*, 11 April 2017. [Online]. Available: <https://www.infosecurity-magazine.com/opinions/malware-detection-signatures/>. [Accessed 11 12 2019].
- [10] P. Lichodziejewski, A. Zincir-Heywood and M. I. Heywood, "Dynamic Intrusion Detection Using Self-Organizing Maps," in *14th Annual Canadian Information Technology Security Symposium*, Ottawa, 2002.
- [11] Y. Mirsky, T. Doitshman, Y. Elovici and A. Shabtai, "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection," in *Network and Distributed Systems Security Symposium*, San Diego, 2018.
- [12] N. Shone, T. N. Ngoc, V. D. Phai and Q. Shi, "A Deep Learning Approach to Network Intrusion Detection," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41-50, 2018.
- [13] N.-N. Dao , T. V. Phan, U. Sa'ad, J. Kim, T. Bauschert and S. Cho, "Securing Heterogeneous IoT with Intelligent DDoS Attack Behavior Learning," *ArXiv*, vol. 1711.06041, 2017.
- [14] S. Naseer, Y. Saleem, S. Khalid, M. K. Bashir, J. Han, M. M. Iqbal and K. Han, "Enhanced Network Anomaly Detection Based on Deep Neural Networks," *IEEE Access*, vol. 6, pp. 48231-48246, 2018.
- [15] M. Tavallaee, E. Bagheri, W. Lu and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *IEEE Symposium on Computational Intelligence for Security and Defense Applications*, Ottawa, 2009.
- [16] C. Langin, H. Zhou, S. Rahimi, B. Gupta, M. R. Zargham and M. R. Sayeh, "A self-organizing map and its modeling for discovering malignant network traffic," in *IEEE symposium on computational intelligence in Cyber Security*, Nashville, 2009.
- [17] J. Tian, M. H. Azarian and M. Pecht, "Anomaly detection using self-organizing," in *Proceedings of the European Conference of the Prognostics and Health Management Society*, Nantes, France, 2014.
- [18] B. D. Santos, "SELF ORGANISING MAPS: ANOMALY DETECTION," *Superdatascience*, 13 June 2018. [Online]. Available: <https://www.youtube.com/watch?v=iXAuKTT5rPw>.

- [19] S. S. Dhaliwal, A. Al Nahid and R. Abbas, "Effective Intrusion Detection System Using XGBoost," *Information*, vol. 9, p. 149, 2018.
- [20] Y. Meidan, M. Bohadana, Y. Mathov, M. Yisroel, D. Breitenbacher, A. Shabtai and Y. Elovici, "N-BaloT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders," *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12-22, 2018.
- [21] J. N. Bakker, "Intelligent Traffic Classification for Detecting DDoS Attacks using SDN/OpenFlow," 2017.
- [22] M. C. Kind and R. J. Brunner, "SOMz: photometric redshift PDFs with self-organizing maps and random atlas," *Monthly Notices of the Royal Astronomical Society*, vol. 438, no. 4, p. 3409–3421, 2014.
- [23] P. Stefanovic and O. Kurasova, "Outlier detection in self-organizing maps and," *Neural Network World*, vol. 28, no. 2, pp. 106-117, 2018.
- [24] B. Silva and N. C. Marques, "The ubiquitous self-organizing map for non-stationary data streams," *Journal of Big Data*, vol. 2, no. 1, p. 27, 2015.
- [25] G.-F. Fang, Y.-h. Guo, J.-M. Zheng and W.-C. Hong, "Application of the Weighted K-Nearest Neighbor Algorithm for Short-Term Load Forecasting," *Energies*, vol. 12, no. 5, p. 916, 2019.
- [26] Y. LeCun, L. Bottou, G. B. Orr and K.-R. Muller, "Neural networks: Tricks of the trade," *Springer*, vol. 7700, pp. 9-48, 2012.
- [27] X. Glorot, A. Border and Y. Bengio, "Deep Sparse Rectifier Neural Networks," in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, Fort Lauderdale, 2011.
- [28] M. A. Nielsen, *Neural networks and Deep Learning*, Determination Press, 2015.
- [29] C. McDonald, "Machine learning fundamentals (I): Cost functions and gradient descent," Medium, 27 November 2017. [Online]. Available: <https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220>.
- [30] F. P. G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M.

Brucher, M. Perrot and E. Cuchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

- [31] S. Van Der Walt, S. C. Colbert and G. Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22-30, 2011.
- [32] W. G. Mckinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Austin, 2010.
- [33] F. Chollet, "Keras," Github, 2015. [Online]. Available: <https://github.com/fchollet/keras>.
- [34] G. Vettigli, "MiniSom is a minimalistic implementation of the Self Organizing Maps," Github, 2018. [Online]. Available: <https://github.com/JustGlowing/minisom>.
- [35] F. Hoogenboom, "Implementation of Kohonen SOM for anomaly detection purposes.," Github, 2017. [Online]. Available: <https://github.com/FlorisHoogenboom/som-anomaly-detector>.