



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**NEURAL NETWORK MODEL INTERPRETABILITY FOR
COMPUTER NETWORK OPERATIONS AND DEFENSE**

by

Alexander Huang

December 2023

Thesis Advisor:

Co-Advisor:

Armon C. Barton

Patrick McClure

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE December 2023		3. REPORT TYPE AND DATES COVERED Master's thesis
4. TITLE AND SUBTITLE NEURAL NETWORK MODEL INTERPRETABILITY FOR COMPUTER NETWORK OPERATIONS AND DEFENSE			5. FUNDING NUMBERS	
6. AUTHOR(S) Alexander Huang				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) Effective cyberspace defense and incident response on Navy networks is predicated on the ability to quickly identify, characterize, classify, and respond to network events. A vast amount of network data is collected on shore and enterprise networks alike, but the quantity of data hinders rapid analysis and identification of key events of consequence for network defenders. This research uses machine learning methods to aid in automated decision-making and incident response for network administrators and security operators. We build and test deterministic and Bayesian neural network models as classifiers to discriminate benign traffic from traffic that should be blocked by a firewall network security device. Using Bayesian methods for neural networks, we demonstrate ways to capture and visualize uncertainty and confidence metrics that are not attainable from deterministic models. Finally, we propose class expected saliency maps and class expected Hessians as novel approaches to use machine learning to enhance network traffic analysis and better understand how and why models make predictions. This work provides a proof-of-concept for how model uncertainty and interpretability might be considered in the context of network security and defense.				
14. SUBJECT TERMS classification, anomaly detection, deep learning, artificial neural networks, network traffic analysis, computer networks			15. NUMBER OF PAGES 77	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

**NEURAL NETWORK MODEL INTERPRETABILITY FOR COMPUTER
NETWORK OPERATIONS AND DEFENSE**

Alexander Huang
Lieutenant, United States Navy
BS, United States Naval Academy, 2016

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
December 2023**

Approved by: Armon C. Barton
Advisor

Patrick McClure
Co-Advisor

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Effective cyberspace defense and incident response on Navy networks is predicated on the ability to quickly identify, characterize, classify, and respond to network events. A vast amount of network data is collected on shore and enterprise networks alike, but the quantity of data hinders rapid analysis and identification of key events of consequence for network defenders. This research uses machine learning methods to aid in automated decision-making and incident response for network administrators and security operators. We build and test deterministic and Bayesian neural network models as classifiers to discriminate benign traffic from traffic that should be blocked by a firewall network security device. Using Bayesian methods for neural networks, we demonstrate ways to capture and visualize uncertainty and confidence metrics that are not attainable from deterministic models. Finally, we propose class expected saliency maps and class expected Hessians as novel approaches to use machine learning to enhance network traffic analysis and better understand how and why models make predictions. This work provides a proof-of-concept for how model uncertainty and interpretability might be considered in the context of network security and defense.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1	Introduction	1
1.1	Background	1
1.2	Motivation	3
1.3	Research Objectives	4
1.4	Scope	4
1.5	Contributions	4
1.6	Organization	5
2	Methods	7
2.1	Data Collection	7
2.2	Class Imbalance and Data Sampling	10
2.3	Data Features	11
2.4	Supervised Machine Learning for Classification	14
2.5	Optimization of Neural Networks	17
2.6	Interpreting Machine Learning Models	25
2.7	Performance Metrics	28
2.8	Chapter Summary	30
3	Experiments	31
3.1	Hardware and Software Environment	31
3.2	Classifier Hyperparameters and Architecture	32
3.3	Classifier Results	35
3.4	Feature Attribution and Interactions	41
3.5	Chapter Summary	49
4	Conclusions and Future Work	51
4.1	Conclusions	51
4.2	Limitations	51
4.3	Future Work	52

List of References	53
Initial Distribution List	59

List of Figures

Figure 1.1	PCAP file format.	2
Figure 2.1	Data collection architecture	7
Figure 2.2	Distributions of application ports	9
Figure 2.3	Class imbalance	10
Figure 2.4	Available data features	12
Figure 2.5	The SiLU and ReLU activation functions	17
Figure 2.6	Illustration of dropout in neural networks.	20
Figure 2.7	Example Hessian matrix	27
Figure 3.1	Neural network classifier calibration	39
Figure 3.2	Prediction visualizations with Bayesian models	40
Figure 3.3	Outbound classifier Hessian visualization, allowed class	44
Figure 3.4	Outbound classifier Hessian visualization, blocked class	45
Figure 3.5	Inbound classifier Hessian visualization, allowed class	48
Figure 3.6	Inbound classifier Hessian visualization, blocked class	48

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

Table 3.1	RFC hyperparameters	32
Table 3.2	DNN architecture	33
Table 3.3	Outbound DNN hyperparameters	33
Table 3.4	Concrete dropout model architecture	34
Table 3.5	Concrete dropout model hyperparameters	35
Table 3.6	RFC performance results	35
Table 3.7	DNN performance results	36
Table 3.8	MC dropout model performance	36
Table 3.9	Concrete dropout model performance	37
Table 3.10	Macro performance results.	38
Table 3.11	Expected calibration error	39
Table 3.12	Outbound allowed feature attribution	42
Table 3.13	Outbound blocked feature attribution	43
Table 3.14	Inbound allowed feature attribution	46
Table 3.15	Inbound blocked feature attribution	47

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

ANN	artificial neural network
API	application programming interface
CART	classification and regression tree
CEH	class expected Hessian
CESM	class expected saliency map
CNN	convolutional neural network
CPU	central processing unit
CUDA	compute unified device architecture
DNN	deterministic neural network
ERN	Education and Research Network
GPU	graphics processing unit
HPC	high performance computing
IANA	Internet Assigned Numbers Authority
IMAP	internet mail access protocol
IP	internet protocol
IPv4	internet protocol, version 4
KL	Kullback-Leibler
MAP	maximum a posteriori probability
MC	Monte Carlo

ML	machine learning
MLE	maximum likelihood estimation
MLP	multi-layer perceptron
NPS	Naval Postgraduate School
PCAP	packet capture
RFC	random forest classifier
ReLU	rectified linear unit
SiLU	sigmoid-weighted linear unit
SMOTE	synthetic minority oversampling technique
TCP	transmission control protocol
TPR	true positive rate
TTL	time-to-live
UDP	user datagram protocol

Acknowledgments

Thank you to my advisors for assisting and supporting me through this thesis. I would also like to thank my family for their steadfast support for the last two years.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:

Introduction

1.1 Background

The U.S. Navy operates an expansive portfolio of networked information systems, including afloat, ashore, and expeditionary networks. The requirement for ships and submarines to operate with frequent and extended disruptions to radio-frequency and satellite communications necessitates a separation between shore enterprise networks and shipboard or tactical expeditionary networks, posing a unique challenge for network defenders that must correlate and corroborate massive amounts of network data from many diverse networks and enclaves. Automation is a potential solution to aid human operators with the initial tasks of detection and identification of network events for further investigation.

Network security operators and administrators can be subject to alert fatigue with current log aggregation and intrusion detection systems [1]. Utilization of a machine learning model to autonomously classify large amounts of collected data and present accurate, precise conclusions to the operator has potential to enhance existing signature-based or heuristic-based network defense systems. Automation using machine learning techniques has the potential to be a force multiplier at the tactical edge of network defense, allowing fewer personnel to feasibly assess more information at faster speed. Additionally, operators and administrators should have mechanisms for understanding the how and why a machine learning model arrives at a prediction and when a prediction might incur more or less risk if trusted. To exploit these benefits and address these issues, this research aims to design, implement, and assess a machine learning system trained from network traffic data to assist with network analysis for network administrators and security operators.

Network Data

Two general data formats can be used to represent network traffic data: packet capture (PCAP) records and network flow metadata. PCAP records, shown in Figure 1.1, are a file format containing the raw information of a network packet [2]. Records can be collected with specialized software such as Wireshark or tcpdump [3].

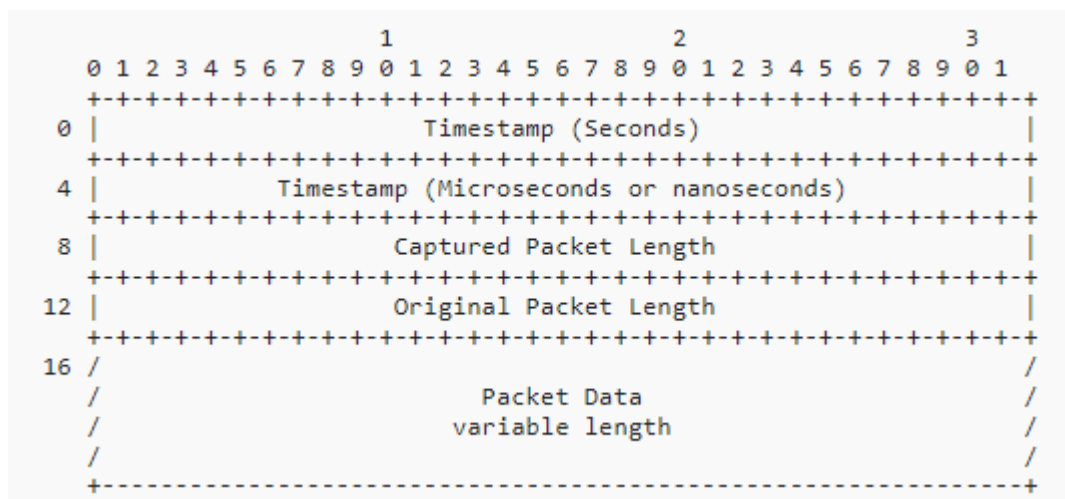


Figure 1.1. File format for PCAP. Source: [2, figure 2].

Network flow metadata can be represented in a variety of file formats. Two common commercial formats are Cisco NetFlow records [4] and Zeek, formerly Bro, logs [5].

Network Defenses

There are three broad objectives for network security and network defenses: confidentiality, integrity, and availability [3]. Network security devices such as firewalls and intrusion detection systems are a crucial component of an overarching security system within a computer network. Both firewalls and intrusion detection systems share common approaches of using defined policies to make decisions on network traffic based on available information about applications, sessions, and protocols derived from collection of network data.

Network Traffic Analysis

Network traffic analysis [3] is the general approach of identifying and deriving useful information about network behavior from collected network data. Knowledge gained by operators from effective network traffic analysis can be applied to tasks like system validation and troubleshooting, performance monitoring, and identification of anomalous behavior.

1.1.1 Prior Work

Prior approaches to network traffic classification and analysis with machine learning examined applications to intrusion detection and user behavior profiling. Ferrag et. al. [6] examined multiple machine learning approaches to intrusion detection, and broadly categorized current approaches into two categories: deep discriminative models and generative/unsupervised models. Included in the former category were deep neural networks, recurrent neural networks, convolutional neural networks, random forests, Naive Bayes classifiers, and support vector machines. Tang et. al. [7] demonstrated the use of deep neural networks as components of an intrusion detection system using network flows. Other approaches incorporated hybrid models that combined random forests using the classification and regression tree (CART) algorithm [8] with other learning methods, including k-Means or k-nn clustering [9], fuzzy logic algorithms [10], and rule-based algorithms [11].

Prior work on using machine learning models to classify network traffic datasets on the same network and using similar data collection and processing methods was done by Capt. Michael Calnan [12] and LT Matthew Straughn [13]. Calnan's work [12] tested three models, a multi-layer perceptron (MLP), a random forest classifier (RFC), and a convolutional neural network (CNN), in which each model was configured as a four-class classifier to discriminate inbound and outbound traffic simultaneously. Semi-supervised approaches to behavioral profiling were also explored, and it was found that machine learning approaches could identify and differentiate broad categories of user behavior from network data. Straughn [13] used representations of network traffic flows as specialized traffic interaction graph data structures as inputs to graph neural networks and compared this approach to CNN and RFC classifiers. Additionally, a mixture of known malicious traffic from a public dataset [14] and collected data from the subject network was used to build a dataset with more reliable ground truth labels for malicious traffic [13].

1.2 Motivation

This research seeks to address operational gaps in network traffic analysis and network defense by examining machine learning approaches to enhance existing alert systems and analysis techniques. The sheer volume of data collected and presented in network defense applications results in alert fatigue for human operators. Machine learning and automation of network traffic analysis can assist with time-sensitive incident response and decision-making

processes. However, an issue with overconfidence in models arises when incorporating machine learning predictions for high-risk decision processes. This impact to operational risk management must be understood by decision-makers and should be mitigated.

1.3 Research Objectives

The primary engineering objective of this thesis is to build and interpret deep learning models for detecting and classifying potentially malicious network traffic.

The research questions are:

1. How can machine learning be leveraged to enhance decision-making for network incident investigation and response?
2. What data features are most important for providing meaningful insight about whether network traffic is potentially malicious?
3. What useful insight could be provided from machine learning with regard to verification of security policies?

1.4 Scope

The scope of this thesis is to:

1. Design, implement, test, and compare deterministic and Bayesian neural network models to classify network traffic.
2. Design, implement, and evaluate saliency map and Hessian-based interpretability methods to provide visualization and transparency of meaningful neural network inputs to network operators.

1.5 Contributions

Major contributions in this research are:

1. The definition of a processing pipeline to convert internet protocol (IP) header information into usable input to machine learning models.

2. The design and evaluation of Bayesian neural network methods for classification of network data, namely Monte Carlo dropout and concrete dropout, and comparison of these methods against deterministic approaches.
3. The adaptation and demonstration of visualization methods for deep neural networks traditionally used in an image classification context to a network traffic classification context. Specifically, we propose two methods to visualize model decisions, the class expected saliency map (CESM) and the class expected Hessian (CEH).

1.6 Organization

Chapter 2 details research methodology and approaches used throughout this research. Chapter 3 describes results and contains analysis of results and products generated through the course of this research. Chapter 4 describes conclusions attained from the work in this thesis and provides recommendations for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2: Methods

This chapter provides a review of network and machine learning (ML) methods used in this thesis. First, dataset collection methodology, characteristics of the analyzed data, and feature engineering are discussed. Then, an overview of the basic ML concepts, the models, the optimization methods, and the performance metrics used for this thesis are presented.

2.1 Data Collection

The dataset was derived from raw packet capture collected from the Naval Postgraduate School (NPS) Education and Research Network (ERN). Figure 2.1 shows the high-level network architecture, including the collection points hosted by the arbiter server.

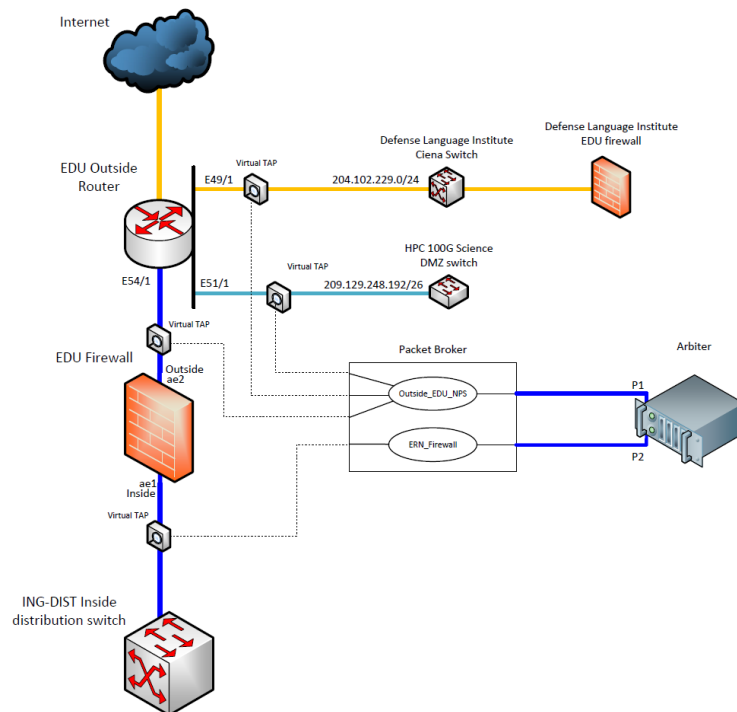


Figure 2.1. Data collection architecture showing main taps of interest on interfaces ae1 (inside tap) and ae2 (outside tap). Source: [15, figure 2].

The inner tap facing the internal network will be referred to as the inside tap, and the outer tap facing the education domain (denoted EDU) outside router will be referred to as the outside tap. The raw packet capture data was collected on December 5-12th, 2022. The raw packets were then processed into labeled packet data, with features derived from the packet header information [15]. Additional information on the collection of packets and processing them into labeled packet data is documented in [15].

The labeled packet data was then used as the starting point for data selection, processing, and feature engineering. First, packets not inbound to the NPS ERN from the Internet or outbound from the NPS ERN to the Internet were filtered out from the dataset. This included network traffic destined for other tenant networks connected to the education domain outside router and traffic from the NPS ERN not destined for the Internet. Then, inbound and outbound traffic were separated into two distinct inbound and outbound datasets. Traffic for either dataset was then labeled with a binary value corresponding to a firewall block or allow decision, with 1 for blocked traffic and 0 for allowed traffic.

In the context of network traffic classification for classifying future traffic, a train, test, and validation data split by time, with testing data being collected at a later time than the training data, is used to develop a classifier. This split by time is representative of the desired real-world use case. This presented a challenge for training supervised machine learning models with this dataset, as the training data did not necessarily closely represent the validation or test data. Figure 2.2 demonstrates the difference in datasets by application ports observed in the outbound allowed and blocked traffic classes.

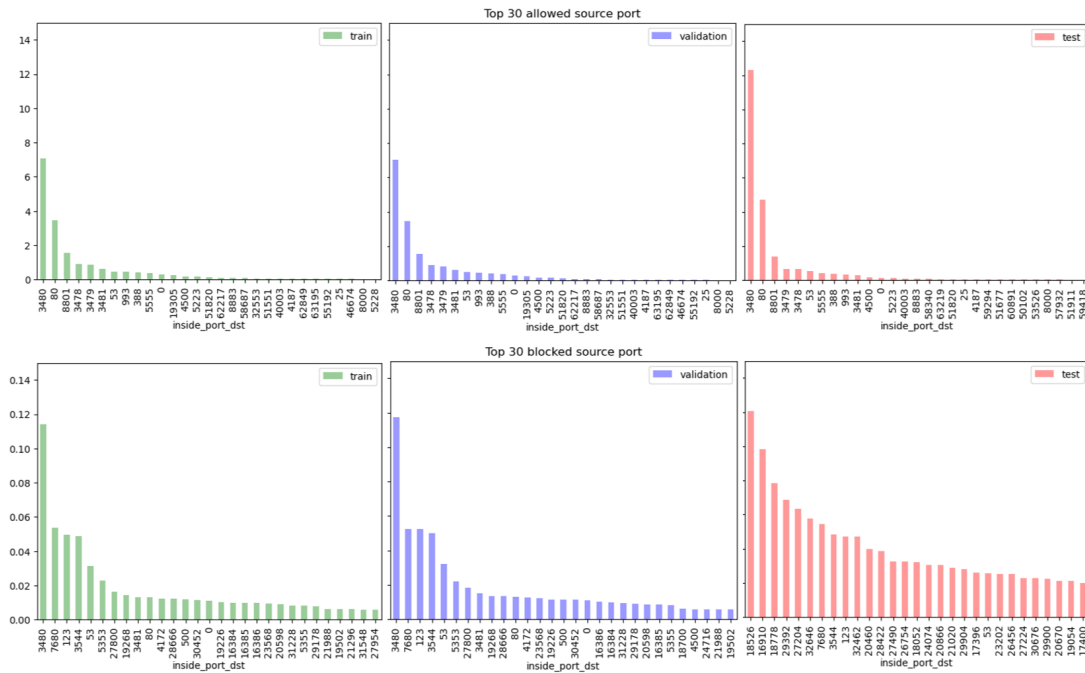


Figure 2.2. Difference in data distributions for top application ports observed between the training (left), validation (center), and test (right) outbound datasets by allowed (upper) and blocked (lower) class. Values are represented in percentages of the respective dataset. Port 443 was the top application port for each histogram, but was removed for clarity. Values for port 443 for the allowed class were 77% for the training/validation sets and 72% for the test set. Values for the blocked class were 0.33% for the training set, 0.36% for the validation set, and 0.24% for the test set.

The data split used for training and validation/test data was Monday through Thursday, December 5-9 from 9 a.m. to 4 p.m. for training, and December 12 from 10 a.m. to 11 a.m. for testing. Random uniform samples were taken from these time periods to create the training and testing sets, and ten percent of the training set was set aside for validation. Another issue for this dataset was a severe class imbalance between the blocked and allowed classes, with the blocked classes consisting of about 2% of the dataset. Figure 2.3 demonstrates this issue.

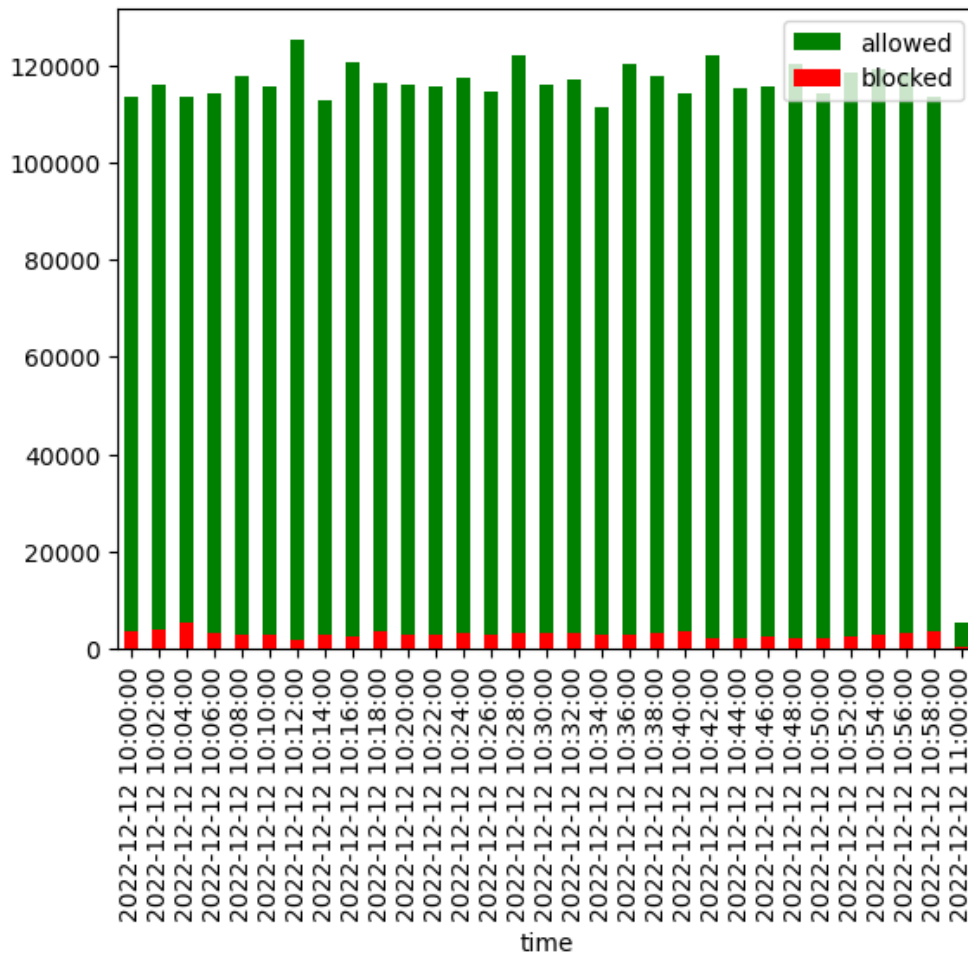


Figure 2.3. Packet counts of test set, sampled from 10:00 am to 11:00 am Pacific Standard Time on December 12, 2022, as a demonstration of class imbalance inherent in this dataset.

2.2 Class Imbalance and Data Sampling

To address the class imbalance inherent to the dataset, sub-sampling of the data was used. Over-sampling of the under-represented class was used to help balance the data distributions between the classes. Specifically, the synthetic minority oversampling technique (SMOTE) algorithm was used to generate new samples through extrapolation from existing samples

from the under-represented class. This can potentially introduce more data variety than random over-sampling alone [16]. Soe et. al. [17] used SMOTE over-sampling with artificial neural networks (ANNs) for distributed denial of service attack detection with an imbalanced network traffic dataset.

2.3 Data Features

The labeled packet data features were derived from a subset of standard internet protocol, version 4 (IPv4) packet header fields. Some fields were not available, such as the header length, type of service, checksum, and fragmentation information. Because of the inside and outside collection points, each data sample may have been observed twice, once for each tap. Thus, there was a corresponding set of inside and outside values for each feature.

For allowed packets, there was a duplicated set of features for each tap, and, for the blocked packets, there was a set of zero values for the tap on the opposing side of the firewall. In the case of inbound traffic, blocked packets had zero values for inside features, and, in the case of outbound traffic, blocked packets had zero values for outside features. Therefore, for the inbound traffic data, all inside features in Figure 2.4 were dropped, and, for the outbound traffic data, all outside features in Figure 2.4 were dropped.

Field	Type	Description
outside_timestamp	float64	The Linux epoch timestamp of the outside packet as float64, enabling microsecond resolution.
inside_timestamp	float64	The Linux epoch timestamp of the inside packet as float64, enabling microsecond resolution.
outside_ip_src	uint32	The source IPv4 address from the outside packet.
inside_ip_src	uint32	The source IPv4 address from the inside packet.
outside_ip_dst	uint32	The destination IPv4 address from the outside packet.
inside_ip_dst	uint32	The destination IPv4 address from the inside packet.
outside_port_src	uint16	If the IP protocol is TCP or UDP, this is the source port of the outside packet. Otherwise it is 0.
inside_port_src	uint16	If the IP protocol is TCP or UDP, this is the source port of the inside packet. Otherwise it is 0.
outside_port_dst	uint16	If the IP protocol is TCP or UDP, this is the destination port of the outside packet. Otherwise it is 0.
inside_port_dst	uint16	If the IP protocol is TCP or UDP, this is the destination port of the inside packet. Otherwise it is 0.
outside_tcp_seq	uint32	If the IP protocol is TCP, this is the TCP sequence number taken from the TCP segment header. Otherwise it is 0.
inside_tcp_seq	uint32	If the IP protocol is TCP, this is the TCP sequence number taken from the TCP segment header. Otherwise it is 0.
outside_tcp_ack	uint32	If the IP protocol is TCP, this is the TCP Acknowledgment number taken from the TCP segment header. Otherwise it is 0.
inside_tcp_ack	uint32	If the IP protocol is TCP, this is the TCP Acknowledgment number taken from the TCP segment header. Otherwise it is 0.
outside_tcp_win_size	uint32	If the IP protocol is TCP, this is the calculated receive window size, accounting for window scaling. Otherwise it is 0.
inside_tcp_win_size	uint32	If the IP protocol is TCP, this is the calculated receive window size, accounting for window scaling. Otherwise it is 0.
outside_tcp_flags	uint16	If the IP protocol is TCP, this contains the TCP flags. Otherwise it is 0.
inside_tcp_flags	uint16	If the IP protocol is TCP, this contains the TCP flags. Otherwise it is 0.
outside_ip_len	uint16	The total length of the outside packet, specifically the IP header plus payload.
inside_ip_len	uint16	The total length of the inside packet, specifically the IP header plus payload.
outside_ip_protocol	uint8	The IP protocol, such as TCP or UDP.
inside_ip_protocol	uint8	The IP protocol, such as TCP or UDP.
outside_ip_ttl	uint8	The time to live value for this packet.
inside_ip_ttl	uint8	The time to live value for this packet.
packet_label	uint8	The packet label, see Table 11.

Figure 2.4. Available data features for collected labeled packet data.
Source: [15, table 10].

The MaxMind *GeoIP2-lite* geolocation database [18] was used to convert IP addresses into geolocation features. IP addresses were converted into country codes and latitude and longitude values using the *GeoIP2-lite* database. Because network address translation is used on the NPS ERN, the source IP addresses resolved to either the same location (United States) or was unable to be resolved by the *GeoIP2-lite* database lookup. Thus, source IP addresses were dropped from the outbound dataset and destination IP addresses were dropped from

the inbound dataset. The *GeoIP2-lite* database also failed to resolve some IP addresses to a proper country code or latitude and longitude. In these cases, the data example was dropped. transmission control protocol (TCP) sequence numbers are relative to individual TCP flows and have no context outside of the relevant TCP conversation. Thus, TCP sequence and acknowledgement numbers, as well as TCP flags, were dropped. Timestamp was found to have a negative impact to classification in [12], and was also dropped. There were samples with “out-of-vocabulary” categorical features, including application ports and geolocation features, in the testing and validation datasets that were not encountered in training. These examples were insignificant in number (less than 0.002%) and were also dropped.

The resulting categorical features included:

- source (inbound) or destination (outbound) country code
- source application port
- destination application port
- source IP protocol
- destination IP protocol

The resulting numerical features included:

- source (inbound) or destination (outbound) latitude
- source (inbound) or destination (outbound) longitude
- packet length
- TCP window size
- packet time-to-live (TTL)

All categorical features were one-hot encoded [19]. All numerical features were scaled per feature and then normalized per sample [19]. The one-hot encoding of the converted country codes, protocol numbers, and port numbers produced a sparse feature matrix that may have impacted classifier performance, but was necessary for the interpretability methods to be discussed later in Section 2.6. To reduce the number of features, application port values greater than 1023 were combined into a single value due to the large number of ephemeral ports used by client-side hosts in various application protocol implementations. The data processing resulted in 381 total features for the inbound datasets and 246 features for the outbound datasets.

2.4 Supervised Machine Learning for Classification

Machine learning is the art and science of learning patterns from data [19]. These methods are often used to enhance human decision processes by providing predictive information or highlighting previously unknown relationships within vast amounts of data. ML can thus be applied to address issues with information coherency from vast amounts of network defense data to enhance decision-making with regard to network defense. Several types of machine learning systems exist, and include supervised, unsupervised, and semi-supervised approaches to building models.

Supervised learning models seek to predict a particular label or continuous value from a set of given input variables, or features. There are two types of problems commonly addressed with supervised learning models: classification problems and regression problems. Classification models predict a particular discrete value, or class, as the label from the set of feature values, and regression models predict a continuous value as the target [19]. This research focused specifically on supervised classification for network traffic data.

2.4.1 Random Forest Classifiers

Random forest classifiers are a type of ensemble model consisting of the aggregation of many trained decision trees [20]. Decision trees have several advantages over other classification methods. They can be easily visualized, with decisions on features denoted in each successive child node from the root. Each node in a decision tree represents a Boolean condition that can be interpreted as a binary value, making each visualized tree simple to interpret. The classification and regression tree (CART) algorithm is a common algorithm for the construction of decision trees for both classification and regression tasks [8], [19]. In the CART algorithm, successive leaf nodes of the decision tree are determined based on Gini impurity [8]. A subtree is then generated based on the impurity for each class given the value of the selected feature.

2.4.2 Artificial Neural Networks

This research uses ANNs to classify network traffic information. ANNs are a category of machine learning models that utilize a layered network of nodes, called neurons, to determine output values from a given input. As applied to classification, a neural network

takes input feature values and outputs probabilities for each class, after which the maximum probability is selected as the predicted class.

Multi-Layer Perceptrons

A MLP is a fully-connected ANN with multiple layers. Each layer performs a linear transformation of the input vector and applies a non-linear activation function. The non-linearity introduced by the activation function and multiple successive layers in a deep model can make ANNs powerful models of complex, multi-dimensional data.

Non-Linear Activation Functions

The softmax activation function is a common activation function for the output layer of an ANN for multi-class classification problems. It is commonly used to convert logits output from the penultimate layer of a model into class probabilities. Equation (2.1) shows the softmax activation function, where z is the pre-activated input vector, and i is a particular index of the input vector corresponding to one of c classes. It can also be used for binary classification problems, provided that predictions and labels are correctly encoded in one-hot representation. This was the activation function used for the output layer for the neural network models in this research.

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^c \exp(z_j)} \quad (2.1)$$

The sigmoid activation function, Equation (2.2), is a commonly used activation function for hidden layers within neural networks and the output layer for binary classification problems. It transforms input values into an output value between 0 and 1 and is both non-linear and continuously differentiable, making it well suited for most applications of neural network models.

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \quad (2.2)$$

However, there may be cases in deep networks where particularly small gradient values exist due to the function being saturated for very large positive and negative numbers. This

circumstance may cause issues with vanishing gradients [21] with deeper neural networks, where gradients are diminished into smaller and smaller values over time until gradient updates cease entirely.

The rectified linear unit (ReLU) activation function [22], Equation (2.3), addresses the issues of the sigmoid activation function, and provides several other advantages.

$$\text{ReLU}(z) = \max(0, z) \quad (2.3)$$

Inputs that are less than or equal to 0 result in a 0 output, and positive inputs are left unchanged. Because this is a relatively simple step function, it is computationally inexpensive to compute, while still providing non-linear transformation of input values. ReLU is also resistant to vanishing gradients, as gradients for positive inputs are constant at 1. As long as the output values are expected to be positive, ReLU can work well for many applications. However, ReLU is not second-order differentiable and is not continuous at 0. This limits its usefulness in the application explored in this research to be discussed in Section 2.6.2, which relied on the use of a Hessian matrix of second-order derivatives of the predicted probabilities of the model with respect to the inputs.

The swish activation function, Equation (2.4), is a modification of the sigmoid-weighted linear unit (SiLU) [23], Equation (2.5). In its original form, the swish function [24] includes a trainable parameter, β , to scale the input value.

$$\text{swish}(z) = z * \sigma(\beta z) \quad (2.4)$$

$$\text{SiLU} = z * \sigma(z) \quad (2.5)$$

In the implementation of the swish activation function used for this research, β was set to 1. This made the swish activation equivalent to SiLU activation. For consistency, references to swish activation in this thesis incorporate this modification. Figure 2.5 illustrates the similarity between the ReLU and SiLU, or swish, activation functions.

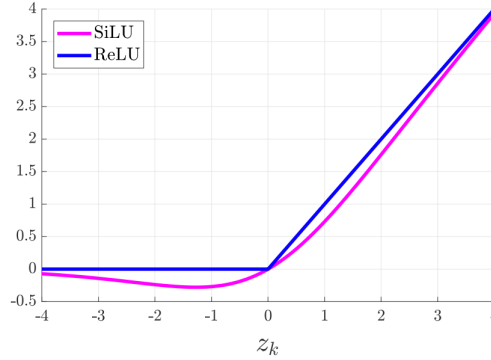


Figure 2.5. The SiLU activation function compared to the ReLU activation function. Source: [23, figure 1].

For large input z values, swish is approximately equal to ReLU, with activation close to 0 for negative input values of z and close to z for positive input values of z . Thus, it retains many of the previously discussed characteristics of ReLU and the associated advantages over the sigmoid activation. The swish activation function was used because it is second-order differentiable and incorporates advantages of ReLU due to its similarity.

2.5 Optimization of Neural Networks

Training a classification model involves finding the optimal parameter matrix θ given an $n \times d$ input feature matrix X and an $n \times c$ target matrix Y , where n is the number of examples, d is the number of input dimensions, and c is the number of classes. Because closed-form solutions are intractable for large models and datasets, we used stochastic gradient descent optimization to find a good estimation of the optimal model parameters. Gradient descent estimates optimal parameters for the model by optimizing an objective function. The update procedure for gradient descent is shown in Equation (2.6), where η is the learning rate and $\mathcal{L}$ is the loss, an objective function to be minimized to optimize the set of model parameters θ at step t .

$$\theta_t = \theta_{t-1} - \eta \nabla_{\theta} \mathcal{L}(Y, X, \theta) \quad (2.6)$$

Learning rate is multiplied by the loss gradient with respect to the model parameters, and represents the amount of step adjustment for the weights between each epoch. Momentum adds additional magnitude to gradient updates based on a moving average of past gradients [25]. This adds consideration for the effect of previous weight changes to the next adjustment [19]. Adam is a common optimization algorithm used for gradient descent. It uses adaptive moment estimation to update gradients, which equates to variable adjustment of momentum between epochs and reduces the need to manually adjust the learning rate with a learning rate schedule throughout training.

This optimization procedure requires the definition of a reasonable loss function. For machine learning with some input dataset, there are several ways to define this loss function derived from fundamental concepts in probability.

2.5.1 Optimization Methods for Deterministic Models

Deterministic methods for model optimization find a point estimate for the set of model parameters given some data. For maximum likelihood estimation (MLE), the likelihood of the data given the parameters is maximized, as shown in Equation (2.7) [26], where $\mathcal{D}$ the dataset comprised of input features X and the output labels Y .

$$\text{MLE} = \underset{\theta}{\operatorname{argmax}} p(\mathcal{D}|\theta) \quad (2.7)$$

MAP estimation expands MLE by estimating the posterior probability using both the likelihood, $p(\mathcal{D}|\theta)$, and some prior distribution, $p(\theta)$. L2 regularization is a common method used in maximum a posteriori probability (MAP) estimation, which is equivalent to using a zero-mean Gaussian prior with isotropic covariance for $p(\theta)$. Using a Gaussian prior in this manner can constrain model parameters during training.

Applying the natural logarithm allows for the simplification of the MAP loss function into a sum of likelihood, prior, and evidence terms. Equation (2.8) [26] represents the objective of MAP estimation.

$$\begin{aligned}
\text{MAP} &= \underset{\theta}{\operatorname{argmax}} p(\theta|\mathcal{D}) \\
\underset{\theta}{\operatorname{argmax}} p(\theta|\mathcal{D}) &= \underset{\theta}{\operatorname{argmax}} \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \\
\underset{\theta}{\operatorname{argmax}} \log p(\theta|\mathcal{D}) &= \underset{\theta}{\operatorname{argmax}} \log p(\mathcal{D}|\theta) + \log p(\theta) - \log p(\mathcal{D})
\end{aligned} \tag{2.8}$$

Because the evidence term $p(\mathcal{D})$ represents the probability of the dataset, it is a constant that does not impact the calculation of the loss gradient and can be ignored in the loss function. To use a gradient descent training procedure, the objective function is negated to define a loss function to minimize over the parameters, shown in Equation (2.9) [26], where the first term of the loss equation is the negative log-likelihood, and the second term is the prior.

$$\begin{aligned}
\underset{\theta}{\operatorname{argmax}} p(\theta|\mathcal{D}) &= \underset{\theta}{\operatorname{argmin}} -p(\theta|\mathcal{D}) \\
\underset{\theta}{\operatorname{argmin}} -p(\theta|\mathcal{D}) &= \underset{\theta}{\operatorname{argmin}} -\frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \\
\underset{\theta}{\operatorname{argmin}} -\log p(\theta|\mathcal{D}) &= \underset{\theta}{\operatorname{argmin}} -\log p(\mathcal{D}|\theta) - \log p(\theta) \\
&= \underset{\theta}{\operatorname{argmin}} -\log p(Y|X, \theta) - \log p(\theta)
\end{aligned} \tag{2.9}$$

Another method of regularization used was dropout. Dropout deactivates all connections to a neuron in the network with drop probability p_{drop} for each forward propagation during training. This is achieved by sampling a dropout variable r from a Bernoulli distribution with $p = 1 - p_{drop}$. During testing, parameters are multiplied by $1 - p_{drop}$. For a single linear neuron i , the pre-activation output z_i is expressed in Equation (2.10) [27] for θ_i consisting of k weights, w_i , and bias b_i .

$$z_{train,i} = r \left(\sum_{j=0}^k w_{i,j} x_{i,j} + b_i \right), \quad r \sim \text{Bern}(1 - p_{drop}) \tag{2.10}$$

Similarly, testing time for a single neuron is expressed in Equation (2.11) [27].

$$z_{test,i} = (1 - p_{drop}) \left(\sum_{j=0}^k w_{i,j} x_{i,j} + b_i \right) \quad (2.11)$$

This dropout procedure results in a model trained from many “thin” neural networks and grants a regularizing effect by preventing the model from placing too much weight on any one particular neuron in training [28]. This is illustrated in Figure 2.6.

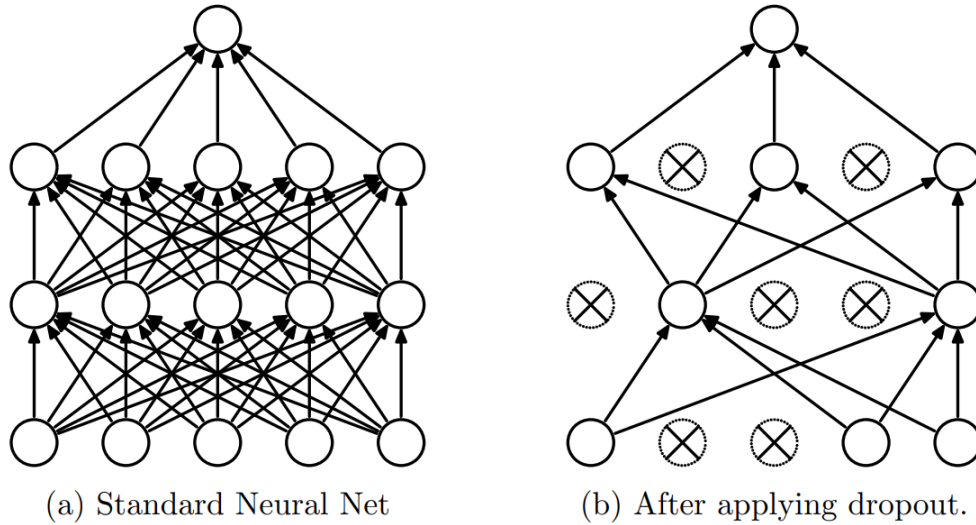


Figure 2.6. Standard neural network (a) with dropout applied at training time (b). All connections are then restored at testing time. Source: [28, figure 1].

A common negative log-likelihood for classification is cross-entropy, defined in Equation (2.12) [26] for c classes.

$$-\log p(y_i|x_i, \theta) = -\sum_c y_c \log p(y_c) \quad (2.12)$$

For the network traffic “blocked” or “allowed” binary classification problem with two classes, $y = 0$ and $y = 1$, we used Equation (2.13) for binary cross-entropy loss.

$$-\log p(y_i|x_i, \theta) = -p(y_i = 0|x_i, \theta) \log p(y_i = 0|x_i, \theta) - p(y_i = 1|x_i, \theta) \log p(y_i = 1|x_i, \theta) \quad (2.13)$$

2.5.2 Optimization Methods for Bayesian Models

Bayesian methods provide an alternative to point estimation methods by providing an estimate of the posterior distribution.

Variational Inference

Variational inference uses the Kullback-Leibler (KL) divergence between some variational distribution parameterized by ϕ , $q_\phi(\theta)$, and the posterior distribution, $p(\theta|\mathcal{D})$ [29]. Generally, the KL divergence between two distributions, $p(\theta)$ and $q_\phi(\theta)$ is given by Equation (2.14) [30]. We considered the continuous case for our parameters, where the expectation is defined by using an integral over the parameters.

$$\text{KL}[q(\theta)|p(\theta)] = \mathbb{E}_{q(\theta)} \left[\log \frac{q_\phi(\theta)}{p(\theta)} \right] = \int q_\phi(\theta) \log \frac{q_\phi(\theta)}{p(\theta)} d\theta \quad (2.14)$$

Thus, the general objective function for variational inference models is shown by Equation (2.15) [30], where the objective is the KL divergence between the posterior distribution and the variational distribution.

$$\begin{aligned} \underset{\phi}{\text{argmin}} \text{KL}[q_\phi(\theta)|p(\theta|\mathcal{D})] &= \underset{\phi}{\text{argmin}} \int q_\phi(\theta) \log \frac{q_\phi(\theta)}{p(\theta|\mathcal{D})} d\theta \\ &= \underset{\phi}{\text{argmin}} -\frac{1}{n} \int q_\phi(\theta) \sum_{i=1}^n \log p(\mathcal{D}_i|\theta) d\theta + \frac{1}{n} \text{KL}[q_\phi(\theta)|p(\theta)] \end{aligned} \quad (2.15)$$

This formulation separates the objective function into a data term and the KL divergence between the variational distribution, $q_\phi(\theta)$, and a prior distribution, $p(\theta)$, as a regularization term. Minimizing the KL divergence between q_ϕ and p effectively makes q_ϕ more similar

to p [25]. The data term is equivalent to the negative log-likelihood as discussed in Section 2.5.1.

Bayesian Model Averaging

Bayesian model averaging, shown in Equation (2.16) [31], can be used to approximate the data term for the variational inference loss objective. In this case, $p(\theta|\mathcal{D})$ is approximated by $q_\phi(\theta)$, so $\theta_m \sim q_\phi(\theta)$.

$$p(y|x, \mathcal{D}) \approx \frac{1}{m} \sum_{j=1}^m p(y|x, \theta_m), \quad \theta_m \sim q_\phi(\theta) \quad (2.16)$$

where $p(\theta|\mathcal{D}) \approx q_\phi(\theta)$

Monte Carlo Dropout

Monte Carlo (MC) dropout is an extension of standard dropout [28] in which neurons are dropped at both training time and testing time [19]. MC dropout approximates sampling model weights from a mixture of mean-field Gaussian distributions for regularization [32]. The theoretical variational distribution for MC dropout is given in Equation (2.17) [27], where $\theta = rw$ for dropout variable r and weights w .

$$q(\theta_i) = p_{drop}N(0, \sigma_q^2 I) + (1 - p_{drop})N(w_i, \sigma_q^2 I) \quad (2.17)$$

The prior regularization term for MC dropout can then be approximated by sampling from this mixture of Gaussians q . Because p is a hyperparameter, it is not trainable and the entropy term is constant with respect to the weights. The prior regularization term is then Equation (2.18) [27].

$$\begin{aligned}
\text{KL}[q_\phi(\theta_i)|p(\theta_i)] &\approx \sum_{i=0}^1 \frac{p_i}{2} \sum_{j=0}^{k-1} \left(\frac{w_{i,j}^2}{\sigma_p^2} + \frac{\sigma_q^2}{\sigma_p^2} - 1 - \log 2\pi - \log \sigma_q^2 - \log \sigma_p^2 \right) - \mathcal{H}(p) \\
&\approx \frac{1 - p_{drop}}{2\sigma_p^2} \sum_{j=0}^{k-1} w_i^2 + C, \\
\text{where } \mathcal{H}(p) &= - \sum_{i=0}^1 p_i \log p_i \text{ and } C \text{ is a constant with respect to } w_i
\end{aligned} \tag{2.18}$$

This method allows models trained with dropout to be used as a Bayesian approximation [32], and gives an inexpensive way to obtain improved uncertainty estimates from those models without changing model architecture or re-training.

Concrete Dropout

Concrete Dropout [33] is another extension of dropout in which the probability of a neuron being dropped, p_{drop} , is a trainable parameter rather than a hyperparameter. The dropout variable r is sampled from a continuous approximation of the Bernoulli distribution, shown in Equation (2.19) [27].

$$\begin{aligned}
r_i &= \text{sigmoid} \left(\frac{1}{t} (\log(1 - p_{drop,i}) - \log(p_{drop,i} + \log u - \log(1 - u))) \right), \\
\text{where } u &\sim \mathcal{U}(0, 1)
\end{aligned} \tag{2.19}$$

This continuous approximation is differentiable, allowing p_{drop} to be learned from data via backpropagation. The regularization term for variational inference loss is a modification to Equation (2.20) [27], as the entropy term $H(p)$ is no longer constant with respect to w_i .

$$\text{KL}[q_\phi(\theta_i)|p(\theta_i)] \approx \frac{1 - p_{drop}}{2\sigma_p^2} \sum_{j=0}^{k-1} w_i^2 + \mathcal{H}(p) + C \quad (2.20)$$

The trade-off for the ability to learn p_{drop} is increased complexity of the model in terms of the number of trainable parameters and the number of mathematical operations, which increases resource requirements to train and test a model.

2.5.3 Uncertainty Quantification

Because Bayesian neural networks produce distributions of parameters, training with Bayesian methods produces a distribution of models from using the training data. This allows for the quantification of different measures of uncertainty for predictions given an input.

Total uncertainty can be measured by finding the entropy of the model prediction, shown in Equation (2.21) [34], [31].

$$\text{total uncertainty} = - \sum_{i=0}^c p(y = c|x, \mathcal{D}) \log p(y = c|x, \mathcal{D}) \quad (2.21)$$

Factors of data collection such as precision of the collection apparatus or the collection methodology can produce noise and uncertainty within the data. This type of uncertainty is aleatoric uncertainty, and it cannot be reduced by the model since it is a property of the data itself. Measures of aleatoric uncertainty can indicate when the quality of the data collected or the data processing could be improved [35], [36]. Aleatoric uncertainty was measured by finding the expectation of the entropy of the model predictions, approximated in Equation (2.22) [34], [31], for m model samples.

$$\text{aleatoric uncertainty} = -\frac{1}{m} \sum_{i=1}^m p(y|x, \theta_m) \log p(y|x, \theta_m) \quad (2.22)$$

Uncertainty also exists about the parameters of the model itself due to issues with model training or lack of data in specific areas within the domain. This type of uncertainty is epistemic uncertainty, and can often be improved with more training or by training with more data [35], [36]. Because the sum of aleatoric and epistemic uncertainty is total uncertainty, epistemic uncertainty was obtained by taking the difference of total uncertainty and aleatoric uncertainty [34].

These uncertainty measures can quantify the level of confidence in a model prediction. In the context of network traffic classification, providing a measure of uncertainty for model predictions gives operators of a machine learning system an understanding of when a model prediction might not be reliable, and additional human analysis is required to reach a decisive conclusion.

2.6 Interpreting Machine Learning Models

This research examined feature attribution and feature interactions as two approaches for interpreting machine learning models.

2.6.1 Feature Attribution Methods

Feature attribution methods examine the importance of input features with respect to an output prediction. They provide a measure of how much the change in a specific feature will change the output prediction. When calculated across an entire dataset, feature attribution methods can grant general insight into how, and how much, a single feature contributes to the predictions of the model. For random forest classifiers trained using the CART algorithm, Gini impurity [8] can be used to derive feature importance values [37], [38]. The mean decrease in Gini impurity for each tree within the forest gives an indicator of importance, with larger magnitudes corresponding to higher feature importance.

Saliency maps are a method of feature attribution for differentiable machine learning models, such as neural networks. Gradient-based saliency maps derive attribution scores from the parameter gradients of a neural network model, and can be generated from both deterministic and Bayesian neural network models. Simonyan et. al. [39] introduced saliency maps for pixel attribution in images with CNN image classifiers. Their proposed method for deriving saliency maps was to calculate the derivative of the class score given by the output layer

of the neural network with respect to the input. For sample image, they found the partial derivative of the output class score for its class with respect to the input. The resulting map of gradients gave attributions for the input. Baehrens et. al. [40] defined these gradients as “explanation vectors” for a class and showed that large values highlight features that influence the classification to that class by the model. A positive sign of the gradient indicated increasing the feature value lowered the probability of the class. Erhan et. al. [41] examined ways of visualizing how layers deeper than the input layer in a neural network transform features with respect to the input space. They identified the neurons of most consequence by finding the maximum activations within a layer through gradient ascent. Other applications of saliency maps include methods like Grad-CAM [42] and SmoothGrad [43] that were shown to highlight parts of an image that contribute the most to a classification result for image classifiers.

In this research, using the numerical output of the gradient-based saliency map methods yielded information about whether certain network packet features, such as the IP address or the application port, contribute to a block or allow prediction. We adapted the approach proposed by Simonyan et. al. [39] and generated a CESM for a class c by calculating the gradient of the probability of the predicted class with respect to the features of an input sample x , as shown in Equation (2.23).

$$\text{CESM}(x, \theta, c) = \nabla_x p(y = c | x, \theta) \quad (2.23)$$

2.6.2 Feature Interaction Methods

Feature interaction methods examine how multiple features together contribute to a prediction. This can be achieved using the second-order derivatives of model weights with respect to model inputs in a Hessian matrix to derive interaction values for pairs of features corresponding to an output label.

Hessian matrices are square matrices of second-order derivatives of some function f with respect to pairs of variables x_i and x_j variable vector x . An example is shown in Figure 2.7.

$$H_f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_0^2} & \frac{\partial^2 f}{\partial x_1 \partial x_0} & \frac{\partial^2 f}{\partial x_2 \partial x_0} \\ \frac{\partial^2 f}{\partial x_0 \partial x_1} & \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_2 \partial x_1} \\ \frac{\partial^2 f}{\partial x_0 \partial x_2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix}$$

Figure 2.7. Hessian matrix for second-order differentiable function f and vector x of variables x_0 , x_1 , and x_2 .

Janizek et. al. [44] proposed Integrated Hessians, which applied to either deterministic or Bayesian models. Cui et. al. [45] proposed a method called Group Expected Hessians to calculate feature interaction values for certain groups within data. This method was more restrictive, as it required the training of a Bayesian neural network model.

We used the Group Expected Hessians method as inspiration to derive feature interactions from Hessians produced by a neural network. First, the matrix of first-order derivatives, or Jacobian matrix, of the gradients produced in Equation (2.23) was generated. Because a gradient is a vector of first-order derivatives, the Jacobian of a gradient of the class predicted probability with respect to the inputs is equivalent to the Hessian matrix of the class predicted probability with respect to the inputs.

Then, we separated Hessians by group and found the expected Hessian of each group by averaging across the group. These groups were defined by the two classes of blocked or allowed network traffic. The expected Hessians for either class were analyzed separately to gain insight on model decisions for either blocked or allowed network traffic rather than summed together.

As with saliency maps for feature attribution, we used raw values instead of absolute values from Group Expected Hessians to find pairwise feature interactions that contributed to block or allow decisions. The goal was to provide more insight for model decisions and learned characteristics of the network security device not given by single-feature attributions alone.

This modification to Group Expected Hessians, referred to as a CEH, is shown in Equation (2.24), where c is either class 0 for allowed traffic or class 1 for blocked traffic.

$$\text{CEH}(x, \theta, c) = \mathbb{E}_{p(x|y=c)} [H_{p(y=c|x, \theta)}(x)] \quad (2.24)$$

Since calculating the Hessian matrix required calculating the derivative of model gradients, a second-order differentiable activation function was needed. Swish was selected as the activation function for this reason and the characteristics discussed in Section 2.4.2

2.7 Performance Metrics

Several common metrics were used to assess the performance of supervised classification models.

2.7.1 Accuracy

Accuracy is the ratio of correct predictions to total samples, shown in Equation (2.25) [19].

$$\text{Accuracy} = \frac{\# \text{ correct samples}}{\# \text{ total samples}} \quad (2.25)$$

High accuracy is generally an indicator of good performance for a classifier, but in cases where the classes are imbalanced, it may not be very reliable. For example, if one class is extremely over-represented, the model can achieve good accuracy by only predicting that class at the detriment of the underrepresented class. This issue applied to network traffic dataset used due to the class imbalance discussed in Section 2.2. Precision and recall are metrics that can more adequately capture the performance of a classification model when class imbalance exists.

2.7.2 Precision

The number of true positives is the number of correct positive predictions, while the number of false positives is the number of incorrect positive predictions. Precision, shown

in Equation (2.26), measures the proportion of correct positive predictions out of all positive predictions made by the model [19].

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \quad (2.26)$$

2.7.3 Recall

The number of false negatives is the number of incorrect negative predictions. Recall, also known as sensitivity, is shown in Equation (2.27). It measures the proportion of true positives out of all samples in the positive class present in the dataset and is synonymous with true positive rate (TPR) [19].

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \quad (2.27)$$

2.7.4 F₁ Score

F₁ score, Equation (2.28), is the harmonic mean of precision and recall and can be used as a combination of both metrics [19]. Because the harmonic mean weights low values higher than a normal mean calculation, the low precision or recall has a higher negative impact on the F₁ score. A good F₁ score is, thus, indicative of both low false positive and false negative rate and a good representation of performance across a combination of both metrics.

$$\text{F}_1 \text{ Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (2.28)$$

2.7.5 Calibration

Calibration metrics [46] were used to establish whether predicted probabilities from a classifier were accurate representations of the actual probabilities in the data. Calibration plots, also known as reliability diagrams, were used to visualize how well the classifiers tested were calibrated. To generate a calibration plot, data was first placed in uniform bins. Then, the true probability was plotted along the horizontal axis and the predicted probability

was plotted along the vertical axis. A perfectly calibrated classifier produces a predicted probability exactly equal to the true probability for each input sample, represented by a line with slope of one on the plot. Plots below this line indicate overconfidence in classifier predictions, while plots above the line indicate underconfidence.

Expected Calibration Error

Expected calibration error [47] estimates the difference between the confidence and the accuracy of predictions for B number of uniform bins. The computed error is the sum of the average difference between accuracy and predicted confidence for each bin and quantifies how well predicted probabilities match true probabilities of the data distribution [46]. Equation (2.29) gives expected calibration error.

$$\text{ECE} = \sum_{i=1}^B \frac{|i|}{n} \cdot |\text{Accuracy}(i) - \mathbb{E}_i[p(y|x, \theta)]| \quad (2.29)$$

2.8 Chapter Summary

In this chapter, we reviewed the characteristics of the network dataset that was used in this research and the data collection and data processing methodologies. Then, machine learning concepts for deterministic and Bayesian classification models and optimization methods were discussed. Finally, we explained methods for model interpretability and discussed metrics for assessing model performance.

CHAPTER 3: Experiments

This chapter provides an overview of the design methodology used for, and the results of, the experiments in this thesis. The software and hardware tools used for research environments and experiments, data collection and processing methodology, and the configuration of machine learning models are covered. Model architectures, loss functions, and parameters for deterministic and Bayesian models are detailed and discussed. Results for tests on four models trained on inbound and outbound network traffic are presented. Finally, the methodology for obtaining feature interactions and visualization for model interpretability is explained.

3.1 Hardware and Software Environment

Experiments conducted for this research required the use of specialized software packages in a contained environment and access to high-end computational hardware due to the size of the dataset and computational complexity of training deep neural networks.

3.1.1 Software

Several software tools provided a comprehensive environment for machine learning suitable for this research. TensorFlow [48] is a software application programming interface (API) that provides a comprehensive set of tools to build, train, evaluate, and share machine learning models. It includes pre-packaged functions and algorithms for end-to-end machine learning projects, but is also open-source and allows for customization of various objects such as model classes, layers, and objective functions. Additionally, TensorFlow is compatible with the NVIDIA compute unified device architecture (CUDA), enabling the use of powerful graphics processing units (GPUs) for computation at scale. Keras [49] extends the base TensorFlow API with additional functionality specifically designed for deep learning and deep neural networks. Scikit-learn [50] is another machine learning and data science API that provides tools to process data, train and evaluate models, and visualize data and results. Imbalanced-learn [51] is a package compatible with Scikit-learn that provides tools to specifically address data imbalance challenges for classification.

3.1.2 Hamming High Performance Computing

The Hamming high performance computing (HPC) center is a hybrid cluster supercomputer consisting of many partitioned nodes capable of central processing unit (CPU) and GPU computation at a large scale. A distributed network filesystem enables relatively fast and scalable storage for large datasets, such as the NPS ERN dataset used in this research.

GPU computation was necessary and key to this research, as use of these resources greatly shortened training and evaluation time for deep neural networks. Model training and testing was performed on a combination of NVIDIA RTX A6000, NVIDIA RTX 6000, and NVIDIA A100 GPUs. All experiments were conducted on single GPUs.

3.2 Classifier Hyperparameters and Architecture

The architectures, configuration, and hyperparameters for the four types of classifiers tested are enumerated in this section. Hyperparameters were found using grid search and the validation set.

3.2.1 Random Forest Classifier

Hyperparameters for the baseline inbound and outbound RFC are listed in Table 3.1.

Table 3.1. Random forest classifier hyperparameters.

Hyperparameter	Outbound Value	Inbound Value
maximum depth	800	800
number of estimators	100	100
minimum samples per split	10	20
minimum samplers per leaf	10	20
SMOTE sampling ratio	0.6	0.45

3.2.2 Deterministic Neural Networks

The model architectures and hyperparameters for the outbound and inbound traffic deterministic neural network (DNN) classifiers are listed in Table 3.2 and Table 3.3.

Table 3.2. DNN classifier model architecture.

Layer	Configuration
Input	246 neurons (outbound), 381 neurons (inbound)
Dense	512 neurons, L2 regularization, swish activation
Dropout	$p_{drop} = 0.2$
Dense	512 neurons, L2 regularization, swish activation
Dropout	$p_{drop} = 0.2$
Output	2 neurons, softmax activation

Table 3.3. DNN classifier hyperparameters.

Hyperparameter	Outbound Value	Inbound Value
regularization coefficient	1e-4	1e-6
optimizer	Adam	Adam
base learning rate	1e-6	1e-5
batch size	4096	8192
number of training epochs	100	100
SMOTE sampling ratio	0.1	0.3

3.2.3 Monte Carlo Dropout Neural Network

Monte Carlo dropout as a Bayesian approximation was tested using the same DNN models discussed above, except with dropout sampling used during testing. The architecture and hyperparameters were thus the same as listed above for the DNN models. The mean of

the predictions across 20 models was taken to approximate the expected prediction. The hyperparameters used are the same as those for the inbound and outbound DNN classifiers listed in Table 3.3.

3.2.4 Concrete Dropout Neural Network

Concrete dropout was added to the same base architecture of the DNN, with minor differences due to the characteristics of the layer wrapper implementation used [33]. The specific implementation of concrete dropout used for these experiments dropped the inputs of the respective layer provided to the layer wrapper. In order to apply dropout to the neurons of the hidden layer, the layer wrapper was applied to the second dense layer and the output layer of the model. Because the concrete dropout implementation used incorporates weight regularization, kernel and bias regularization were removed from the hidden dense layers. The resulting architecture and hyperparameters are listed in Table 3.4 and Table 3.5. Dropout regularization was set to $1/n$ for n examples in the training set.

Table 3.4. Concrete dropout neural network classifier model architecture.

Layer Wrapper	Layer	Configuration
	Input	246 neurons (outbound), 381 neurons (inbound)
	Dense	512 neurons, swish activation
ConcreteDenseDropout	Dense	512 neurons, swish activation
ConcreteDenseDropout	Output	2 neurons, softmax activation

Table 3.5. Concrete dropout model classifier hyperparameters.

Hyperparameter	Outbound Value	Inbound Value
dropout regularization	$1/n$	$1/n$
weight regularization	1e-4	1e-6
optimizer	Adam	Adam
base learning rate	1e-6	1e-6
batch size	4096	8192
number of training epochs	100	100
SMOTE sampling ratio	0.1	0.3

3.3 Classifier Results

The RFC was used as a baseline model for comparison to the neural network models. Table 3.6 lists results for the RFC, and Table 3.7 lists performance metrics for both outbound and inbound fully connected DNN classifiers. While the outbound RFC performed well, there was a notable difference of about 15% in precision compared to the outbound neural network models in Table 3.7. The inbound RFC performed similarly to the inbound neural network models, with a slight reduction in accuracy and precision.

Table 3.6. Random forest classifier performance results.

Classifier	Class	Precision	Recall	F ₁ -score	Accuracy
outbound	0 (allowed)	0.9970	0.9848	0.9909	0.9823
	1 (blocked)	0.5976	0.8845	0.7133	
inbound	0 (allowed)	0.9991	0.8570	0.9226	0.8588
	1 (blocked)	0.1100	0.9568	0.1974	

Table 3.7. Deterministic neural network performance results.

Classifier	Class	Precision	Recall	F ₁ -score	Accuracy
outbound	0 (allowed)	0.9974	0.9923	0.9948	0.9900
	1 (blocked)	0.7475	0.8991	0.8163	
inbound	0 (allowed)	0.9990	0.8631	0.9261	0.8648
	1 (blocked)	0.1141	0.9541	0.2039	

Table 3.8 lists results for the MC dropout classifiers. Both outbound and inbound MC dropout models performed comparably to the inbound DNN models. The concrete dropout model also yielded similar results to the other two classifiers, as shown in Table 3.9. As both Bayesian models performed similarly compared to the deterministic counterpart, this shows that performance need not be traded for the ability to estimate epistemic uncertainty for these datasets.

Table 3.8. MC dropout model performance results.

Classifier	Class	Precision	Recall	F ₁ -score	Accuracy
outbound	0 (allowed)	0.9974	0.9926	0.9950	0.9903
	1 (blocked)	0.7555	0.8992	0.8211	
inbound	0 (allowed)	0.9990	0.8632	0.9262	0.8649
	1 (blocked)	0.1142	0.9540	0.2040	

Table 3.9. Concrete dropout model performance results.

Classifier	Class	Precision	Recall	F1-score	Accuracy
outbound	0 (allowed)	0.9975	0.9923	0.9949	0.9900
	1 (blocked)	0.7489	0.9005	0.8177	
inbound	0 (allowed)	0.9990	0.8630	0.9260	0.8647
	1 (blocked)	0.1140	0.9541	0.2037	

Table 3.10 lists overall results for all tested classifiers. Amongst the outbound classifiers, the MC dropout model had the best overall performance, with minor improvement against the other two neural network models and more significant improvement over the RFC. Performance amongst the inbound neural network classifiers was very similar between any of the four classifiers, but the neural network models had slightly higher precision than the RFC.

However, neither accuracy nor F_1 -score adequately reflect the substantial trade-off between recall and precision for the inbound classifiers. In all inbound classifiers, recall alone was at a more reasonable 0.9 or above. In use cases prioritizing security above other considerations for network performance and usability, it may be acceptable to trade higher recall performance for lower precision. This equates to accepting more false positives in return for a lower false negative rate. In terms of a firewall, this would result in lesser chance of potentially malicious or non-compliant traffic being allowed inbound to the network in exchange for a greater chance of benign, authorized traffic being blocked.

Table 3.10. Summary of classifier performance metrics by macro average between the blocked and allowed class.

Outbound Classifier	Precision	Recall	F1-score	Accuracy
RFC	0.7973	0.9347	0.8521	0.9823
DNN	0.8724	0.9457	0.9056	0.9900
MC dropout	0.8764	0.9459	0.9080	0.9903
Concrete dropout	0.8732	0.9464	0.9063	0.9900
Inbound Classifier	Precision	Recall	F1-score	Accuracy
RFC	0.5546	0.9069	0.5600	0.8588
DNN	0.5566	0.9086	0.5650	0.8648
MC dropout	0.5566	0.9086	0.5651	0.8649
Concrete dropout	0.5565	0.9085	0.5649	0.8647

3.3.1 Uncertainty

For the deterministic model, aleatoric uncertainty was obtained by finding the entropy across model predictions. For the Bayesian models, aleatoric uncertainty was obtained by finding the entropy across the expectations of model predictions across 20 sampled models. Epistemic uncertainty was obtained for the MC dropout and concrete dropout models by finding the expectation of entropy across 20 sampled models, as discussed in Section 2.5.3. For most samples, epistemic uncertainty was low, indicating low prediction variability due to model parameter uncertainty. While aleatoric uncertainty was also low for a majority of samples, there was a notable number of samples for which aleatoric uncertainty was relatively high. Examination of samples by class revealed consistently high aleatoric uncertainty in the blocked class, and a wide range of low to high aleatoric uncertainty for the allowed class. Generally high total uncertainty with respect to the training data, particularly for the blocked class, indicated the presence of noise within the input data that may be improved with better data collection or data processing.

3.3.2 Calibration

Uncertainty about predicted probabilities for the neural network models was evaluated through calibration as discussed in Section 2.7.5. Figure 3.1 shows calibration for all neural network classifiers.

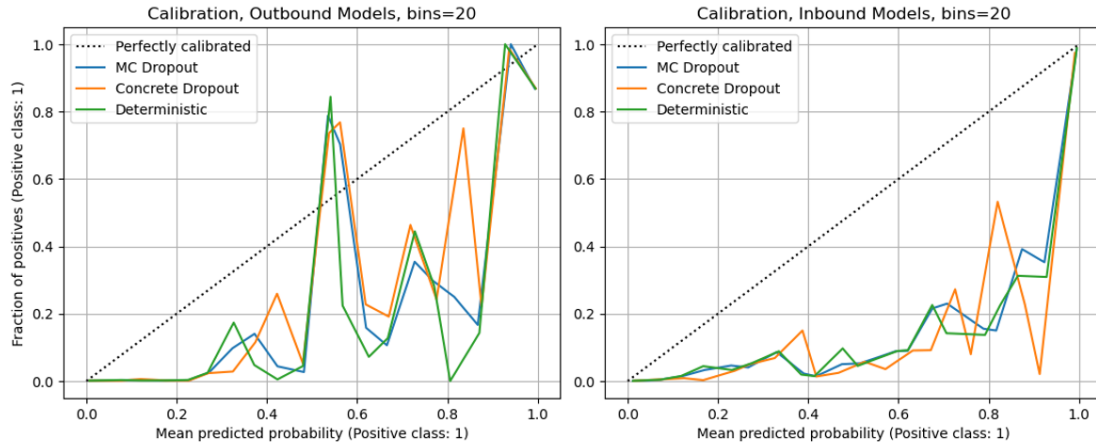


Figure 3.1. Calibration for deterministic, MC dropout, and concrete dropout outbound (left) and inbound (right) models.

While all classifiers were still generally overconfident about predicted probabilities, concrete dropout had the least calibration error for the outbound model, and the deterministic model had the least calibration error for the inbound model, as shown in Table 3.11. MC dropout model results were comparable to those of the models with the least error for both classifier types.

Table 3.11. Expected calibration error for neural network models for 15 bins.

Classifier	DNN	MC Dropout	Concrete Dropout
Outbound	0.0346	0.0330	0.0327
Inbound	0.0929	0.0931	0.1004

3.3.3 Bayesian Model Predictions

Because model parameters are sampled from a distribution, Bayesian models allow for variation in model predictions. The amount of deviation in a particular prediction between different models was represented in a confidence interval visualization, as shown in Figure 3.2.

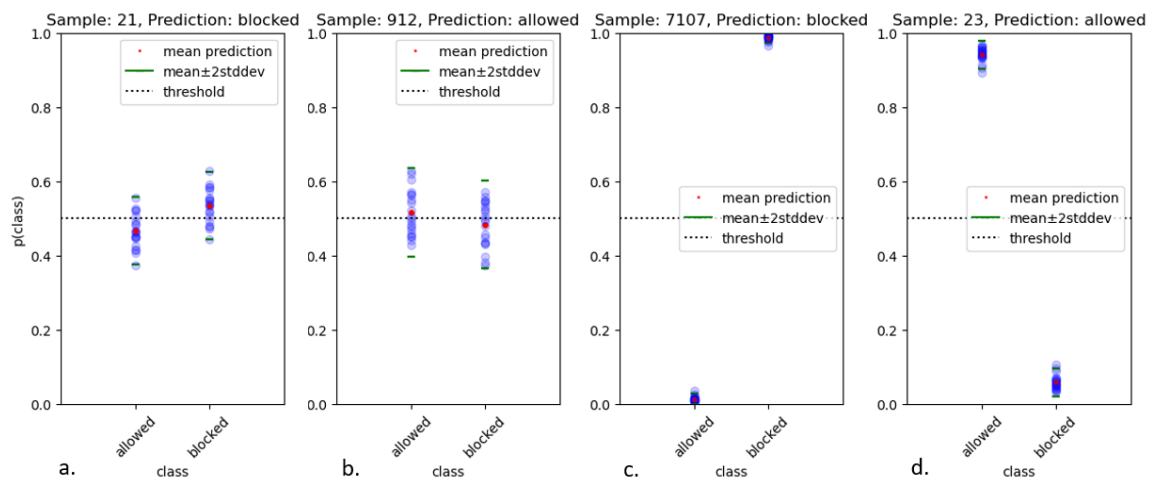


Figure 3.2. Four correct predictions on test examples for the concrete dropout outbound classifier. Plots (a) and (b) depict low confidence and high uncertainty predictions; plots (c) and (d) depict high confidence and low uncertainty predictions.

This cannot be done with deterministic models, which produce the same prediction for each iteration of testing. The two predictions in Figure 3.2, though correct, may warrant additional investigation by a human operator based on the low confidence and large confidence interval observed. The concern of alert fatigue, where intrusion detection systems present large queues of alerts for human review, might also be addressed using these concepts of uncertainty and confidence with machine learning. An operator might prioritize predictions of low confidence and high uncertainty for manual investigation during alert triage and allow the rest of the system to process the other alerts. Such a policy would reduce the amount of positive alerts requiring manual review and thus prevent large numbers of false positive alerts from inflicting alert fatigue on operators.

3.4 Feature Attribution and Interactions

Saliency maps were generated with the method discussed in Section 2.6. Top twenty feature attributions for the neural networks were obtained by sorting the resulting gradients by magnitude. Positive values in the feature attributions reflect a positive contribution for the allow decision, while negative values reflect a negative contribution. The TCP and user datagram protocol (UDP) application port numbers are found in the Internet Assigned Numbers Authority (IANA) Service Name and Transport Protocol Port Number Registry [52]. The IPv4 protocol numbers are found in the IANA Protocol Numbers registry [53].

CEH diagrams were generated using the method discussed in Section 2.6. The top gradients for feature attribution were used to filter the generated Hessians and visualize the most interesting feature interactions that might contribute most to classification decisions by the model. The resulting visualizations display interaction values between the top features and the total feature set.

3.4.1 Outbound Class Expected Saliency and Class Expected Hessian

Table 3.12 and Table 3.13 show the feature attributions for the outbound DNN classifier by allowed and blocked class. Some of the feature attributions can provide some insight into the behavior of the firewall and the outbound network traffic. For example, the high positive attributions to allow predictions and corresponding high negative attributions to block predictions for domain name services (port 53) may have indicated the presence of a server providing this service within the network that regularly communicated with servers and clients external to the network.

Table 3.12. Top 20 attributed features to allow prediction for outbound DNN.

No.	Feature	Value
1.	inside_port_dst_143	-0.0961066
2.	inside_port_dst_585	-0.09012788
3.	inside_port_dst_53	0.09010026
4.	inside_port_src_53	0.08662078
5.	inside_ip_protocol_6	0.078464106
6.	inside_ip_protocol_1	-0.0718925
7.	inside_ip_len	0.06349483
8.	inside_ip_protocol_50	0.061263487
9.	inside_ip_dst_LAT	-0.058016103
10.	inside_ip_protocol_17	-0.04722565
11.	inside_port_src_949	-0.045993797
12.	inside_port_src_443	0.043224886
13.	inside_port_dst_123	-0.041875236
14.	inside_port_dst_388	0.041652843
15.	inside_port_dst_HIGH	0.040386517
16.	inside_ip_dst_SG	-0.039773364
17.	inside_port_src_123	-0.038724985
18.	inside_ip_dst_CN	-0.037877966
19.	inside_ip_dst_NL	0.03659649
20.	inside_ip_ttl	0.034470417

Table 3.13. Top 20 attributed features to block prediction for outbound DNN.

No.	Feature	Value
1.	inside_port_dst_53	-1.3209271
2.	inside_ip_protocol_6	-1.1260004
3.	inside_port_dst_143	1.0151772
4.	inside_port_src_53	-0.9634426
5.	inside_port_dst_585	0.949271
6.	inside_ip_dst_LAT	0.8031542
7.	inside_ip_protocol_17	0.7356974
8.	inside_ip_len	-0.72487247
9.	inside_ip_ttl	-0.71340877
10.	inside_ip_protocol_1	0.7008425
11.	inside_ip_protocol_50	-0.6495795
12.	inside_port_dst_HIGH	-0.6391647
13.	inside_port_dst_443	0.52507436
14.	inside_port_src_949	0.5138083
15.	inside_port_dst_123	0.46952713
16.	inside_port_src_123	0.4355466
17.	inside_ip_dst_NL	-0.42460907
18.	inside_ip_dst_SG	0.41434813
19.	inside_port_dst_388	-0.41289136
20.	inside_port_dst_80	0.41218174

The high negative attributions to allow predictions and corresponding high positive attributions to block predictions for internet mail access protocol (IMAP) (port 143) and IMAP with secure socket layer encryption (port 585) indicated that emails may be regularly blocked. IMAP is not secured with an encryption protocol, and the Service Name and Transport Protocol Port Number Registry [52] indicates port 585 is deprecated. Attribution of these two email application ports to block decisions may have been due to the imple-

mentation of an email filtering solution or email gateway to reduce spam. Alternatively, a policy may have been in place that enforced the use of or gave preference to a more recent secure protocol such as IMAP with transport layer security (port 993).

A majority of the features with high attribution were application ports that correspond to popular application services such as email, domain name services, internet control messaging protocol (IPv4 protocol 1), and transport layer protocols (IPv4 protocols 17 and 6) for common network traffic. It was expected that common applications and protocols would have more significant impact on decisions of the model and potentially of the firewall being modeled.

Port 949 was notable out of the application ports in the outbound feature attributions, as it is not registered by IANA, and the outbound model learned a relatively high attribution to a block decision for that port. A network or security operator may consider investigating the use of this unregistered port and associated applications within the network.

The CEH visualizations in Figure 3.3 and Figure 3.4 display interaction values for allowed and blocked predicted classes of the outbound DNN classifier. Features on the horizontal axis with interaction values less than 0.01 were removed for readability.

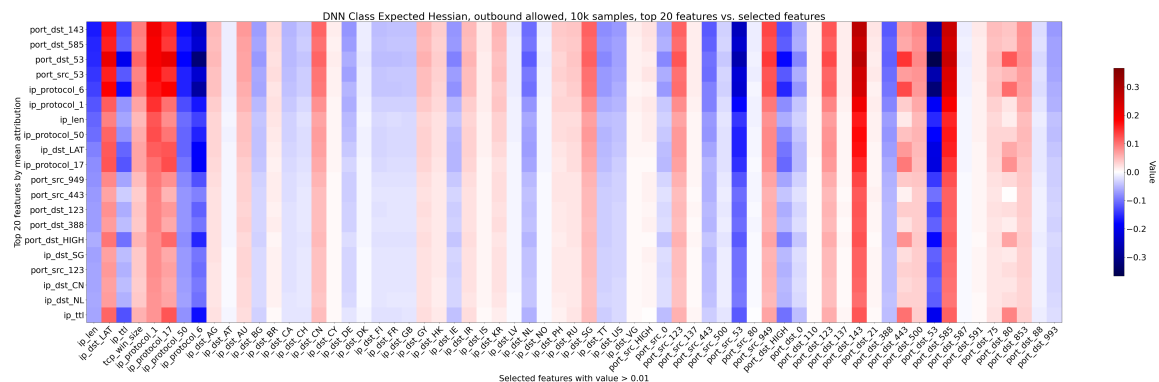


Figure 3.3. Expected Hessian visualization showing top feature interactions for top 20 attributed features across 10,000 test dataset samples for the outbound allowed class.

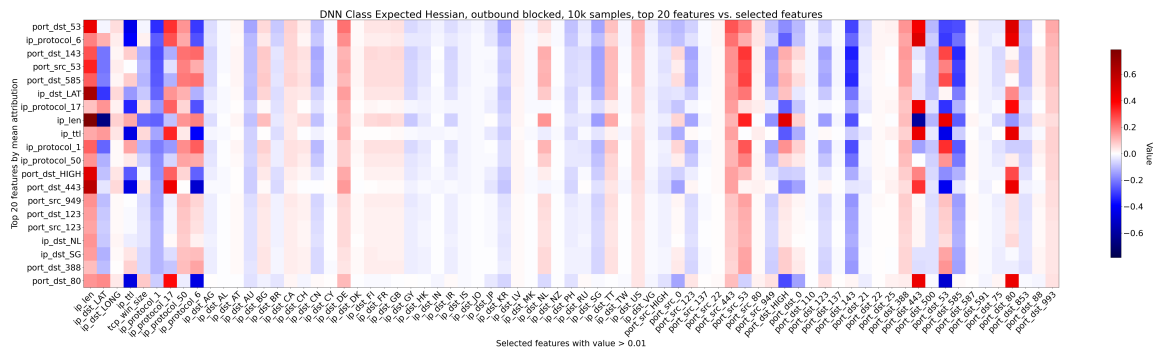


Figure 3.4. Expected Hessian visualization showing top feature interactions for top 20 attributed features across 10,000 test dataset samples for the outbound blocked class.

Each position on the heatmap represents positive or negative interaction between the top feature on the vertical axis and the feature on the horizontal axis. Increases in feature values on the horizontal axis enhanced or reduced the effect of the attribution of features on the vertical axis, with red representing enhancement and blue representing reduction. A red square on the heatmap would mean the feature on the horizontal axis enhanced the respective attribution of the feature on the vertical axis. If the attribution of the feature on the vertical axis was positive, then the interaction with the second feature made that attribution more positive. If the attribution was negative, then the interaction made it more negative.

Several interesting observations were made from the feature interactions displayed. For the allowed class, the China (CN) and Russia (RU) geolocation features slightly enhanced the feature attributions of the top 20 features. Services with positive attribution that would result in higher probability of a packet being allowed included domain name service (port 53) and encrypted web traffic (port 443), indicating that geolocation to China or Russia alone was not necessarily a strong condition for a block decision in conjunction with common applications. Conversely, for the blocked class, the China and Russia geolocation features slightly reduced the feature attributions. Geolocation by itself may not be a good indication of whether traffic might be contrary or compliant to security policy. While it might be expected that Chinese or Russian destinations might contribute more to a block decision, there may have been many cases where traffic to those destinations was valid and authorized.

3.4.2 Inbound Class Expected Saliency and Class Expected Hessian

Feature attributions for the inbound DNN classifier are shown in Table 3.14 and Table 3.15. The inbound model learned that some expected application ports corresponding to common services, such as secure shell (port 22) and email (ports 587, 25, and 465) are allowed into the NPS ERN network, along with encrypted tunneling (IPv4 protocol 50).

Table 3.14. Top 20 attributed features to allow prediction for inbound DNN.

No.	Feature	Value
1.	outside_ip_src_RU	-0.054533627
2.	outside_port_src_53	0.05051326
3.	outside_ip_src_CN	-0.03466427
4.	outside_ip_src_TT	0.033177946
5.	outside_port_src_587	0.032524038
6.	outside_port_src_25	0.031701952
7.	outside_port_dst_22	0.030728133
8.	outside_port_dst_123	-0.030379076
9.	outside_ip_ttl	-0.029814437
10.	outside_port_src_21	-0.028313633
11.	outside_ip_src_IS	-0.027055694
12.	outside_ip_src_NG	0.026314452
13.	outside_ip_protocol_50	0.025400858
14.	outside_port_src_443	0.025357028
15.	outside_ip_src_AG	0.025324926
16.	outside_ip_src_HK	-0.02458259
17.	outside_port_dst_58	-0.023167392
18.	outside_ip_src_UA	-0.02219936
19.	outside_port_src_465	0.02211467
20.	outside_ip_src_LAT	0.021546658

Table 3.15. Top 20 attributed features to block prediction for inbound DNN.

No.	Feature	Value
1.	outside_port_src_53	-0.9440721
2.	outside_ip_src_RU	0.87091404
3.	outside_ip_src_TT	-0.6190797
4.	outside_port_dst_22	-0.6150029
5.	outside_ip_ttl	0.5723791
6.	outside_port_dst_123	0.5685847
7.	outside_ip_src_CN	0.55828905
8.	outside_port_src_443	-0.5520477
9.	outside_port_src_21	0.5275082
10.	outside_port_src_587	-0.4998415
11.	outside_port_src_25	-0.49619323
12.	outside_ip_src_NG	-0.48698092
13.	outside_ip_protocol_50	-0.44962156
14.	outside_ip_src_IS	0.44344342
15.	outside_ip_src_AU	-0.44107243
16.	outside_ip_src_HK	0.42980725
17.	outside_port_src_HIGH	0.4262788
18.	outside_ip_src_IT	0.41256237
19.	outside_ip_src_AG	-0.4001467
20.	outside_port_dst_58	0.38027176

When combined with the outbound attribution results, this presents a clearer picture of two-way traffic for email and virtual private network or remote tunneling services on the network. Conversely, file transfer protocol (port 21) had a high attribution to a block decision, indicating that inbound file transfers using this unsecured service may not have been allowed.

Traffic inbound from China, Russia, and Hong Kong (HK) had a negative attribution to an allow prediction and corresponding positive attribution to a block prediction. The same case exists for Italy (IT) and Israel (IL). Unlike for outbound traffic, it may have been the case that inbound traffic from these locations is restricted or constrained. The attributions for the inbound blocked class in general should be considered in the context of the low precision performance of the inbound classifier. However, useful observations were still made for the blocked class to corroborate those of the allowed class.

CEH diagrams generated for the inbound classifier are shown in Figure 3.5 and Figure 3.6. For the allowed class, interaction values less than 0.02 were filtered; for the blocked class, interaction values less than 0.1 were filtered.

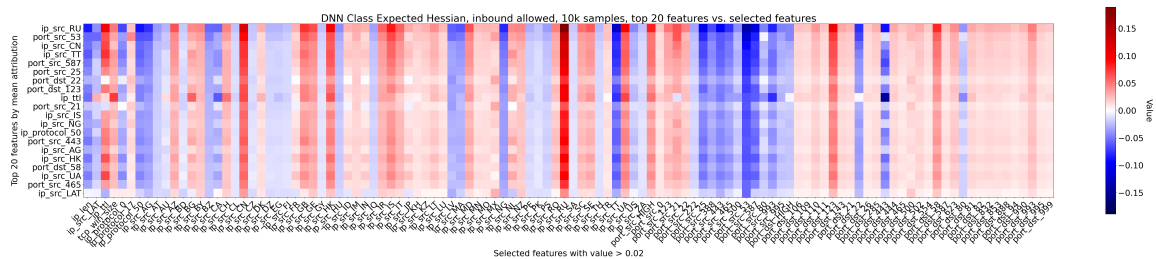


Figure 3.5. Expected Hessian visualization showing top feature interactions for top 20 attributed features across 10,000 test dataset samples for the inbound allowed class.

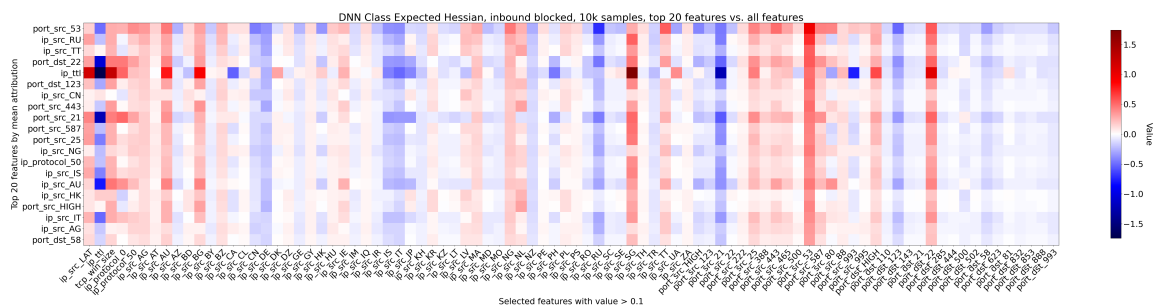


Figure 3.6. Expected Hessian visualization showing top feature interactions for top 20 attributed features across 10,000 test dataset samples for the inbound blocked class.

Strong enhancements to all feature attributions for the allowed class were observed for the China and Russia geolocation features. Since most common application ports had negative attributions to a block decision, this again indicates that geolocation to these nations does not necessarily correlate to block decisions. The enhancement to the negative attribution for file transfer protocol differs from the other application ports, and indicates that geolocation to China and Russia further reduced the probability of unsecured file transfers being allowed.

For the inbound blocked class, the Hessian matrix was more sparse, but there were relatively strong interactions for secure shell (port 22) and domain name service (port 53) traffic enhancing the effect of all attributions. Those services enhanced the probability of a block decision when combined with geolocation to China, Russia, Italy, and Israel, and reduced the probability of a block decision when combined with locations such as Trinidad and Tobago, Afghanistan, and Australia.

3.5 Chapter Summary

This chapter presented an overview of research experiments conducted. This included a description of software APIs and packages used in programming and hardware used to run tests in an experimental environment. Next, configurations for model architecture and hyperparameters were discussed. Finally, this chapter presented results in terms of performance metrics and included a discussion of various visualizations for analysis using techniques and methods discussed in previous chapters.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:

Conclusions and Future Work

This chapter discusses conclusions to be drawn from the research and results, potential limitations of the experiments conducted, and recommendations for additional topics of research for future work.

4.1 Conclusions

Through this research we demonstrated the use of machine learning, particularly Bayesian machine learning methods with deep neural networks, to perform network traffic classification at the packet level. The use of Bayesian methods does not negatively impact model performance and rather provides some minor benefits with respect to defined performance metrics and predictive overconfidence. Class expected saliency maps and class expected Hessians were presented as usable, human-readable interpretations of neural network models. Both interpretation methods were shown to provide coherent insights to model decisions within the context of network traffic analysis.

4.2 Limitations

Data quality was a consistent issue throughout all experiments and greatly impacted model effectiveness for both inbound and outbound classification tasks. Both class imbalance and distribution mismatch between training, validation, and testing datasets contributed to this problem, and the impact to model performance was observed in significant predictive uncertainty in the blocked class for both inbound and outbound classifiers. “Out-of-vocabulary” samples not seen by the model in training were dropped from the validation and testing datasets, but discarding valid data can negatively impact model training and testing. While the outbound classifiers were able to achieve respectable performance with hyperparameter tuning and the use of sampling methods to correct for class imbalance, the inbound traffic remained a problem for machine learning models tested. Finally, the network traffic in the dataset used was derived from firewall observations. While monitoring of firewall decisions might be useful to network administrators, the dataset was limited for the task of intrusion

detection for security operators, as traffic blocked by a firewall may not necessarily be malicious.

4.3 Future Work

For future work, the data limitations discussed above might be addressed by collecting network traffic over a longer period of time or using a dataset with malicious traffic. The current dataset contains just over a week of data; a month or more of packet collection may be necessary. Collection of more data may also resolve the issue of “out-of-vocabulary” samples not seen in training that were dropped in data processing for the validation and testing datasets used in this research. A different dataset with true malicious traffic patterns and labels may be better for testing the machine learning methods discussed for intrusion detection purposes.

Classification of network traffic flows was not tested in this thesis. As demonstrated by Calnan [12] and Straughn [13] on similar data from the same network, flows can represent temporal information not discernible from individual PCAP records alone that may be beneficial for machine learning. Training machine learning models on datasets derived from commercial network flow data formats rather than datasets derived from raw PCAP might also be more applicable to operational use cases where raw PCAP is not collected or stored long-term.

List of References

- [1] S. Rebensky, M. Carroll, A. Nakushian, M. Chaparro, and T. Prior, “Understanding the last line of defense: Human response to cybersecurity events,” in *HCI for Cybersecurity, Privacy and Trust*, 2021. Available: https://link.springer.com/chapter/10.1007/978-3-030-77392-2_23
- [2] M. Richardson, “PCAP Capture File Format,” IETF, Dec. 22, 2020. Available: <https://datatracker.ietf.org/doc/id/draft-gharris-opsawg-pcap-00.html>
- [3] S. Thakare, A. Pund, and M. A. Pund, “Network traffic analysis, importance, techniques: A review,” in *2018 3rd International Conference on Communication and Electronics Systems (ICCES)*, 2018. Available: <https://ieeexplore.ieee.org/document/8723955>
- [4] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, “Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, May 2014. Available: <https://doi.org/10.1109/COMST.2014.2321898>
- [5] V. Paxson, “Bro: A system for detecting network intruders in real-time,” *Computer Networks*, vol. 31, no. 23–24, Dec 1999. Available: [https://doi.org/10.1016/S1389-1286\(99\)00112-7](https://doi.org/10.1016/S1389-1286(99)00112-7)
- [6] M. A. Ferrag, L. Maglaras, S. Moschoyiannis, and H. Janicke, “Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study,” *Journal of Information Security and Applications*, vol. 50, no. 102419, Apr. 2020. Available: <https://doi.org/10.1016/j.jisa.2019.102419>
- [7] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, “Deep learning approach for network intrusion detection in software defined networking,” in *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, 2016. Available: <https://ieeexplore.ieee.org/document/7777224>
- [8] L. Breiman, J. Friedman, C. Stone, and R. Olshen. *Classification and Regression Trees*. Milton Park, Oxfordshire, UK: Taylor & Francis, 1984. Available: <https://books.google.com/books?id=JwQx-WOmSyQC>
- [9] Y. Y. Aung and M. M. Min, “Hybrid intrusion detection system using k-means and classification and regression trees algorithms,” in *2018 IEEE 16th International Conference on Software Engineering Research, Management and Applications (SERA)*, 2018, pp. 195–199.

- [10] A. F. A. Pinem and E. B. Setiawan, "Implementation of classification and regression tree (CART) and fuzzy logic algorithm for intrusion detection system," in *2015 3rd International Conference on Information and Communication Technology (ICoICT)*, 2015. Available: <https://ieeexplore.ieee.org/document/7231434>
- [11] T. Komviriyavut, P. Sangkatsanee, N. Wattanapongsakorn, and C. Charnsripinyo, "Network intrusion detection and classification with decision tree and rule based approaches," in *2009 9th International Symposium on Communications and Information Technology*, 2009. Available: <https://ieeexplore.ieee.org/document/5341005>
- [12] M. C. Calnan, "Multi-dimensional profiling of cyber threats for large-scale networks," M.S. thesis, Dept. of Comp. Sci., NPS, Monterey, CA, USA, 2022. Available: <https://hdl.handle.net/10945/71108>
- [13] M. N. Straughn, "Classifying TCP network traffic flows via traffic interaction graphs and machine learning," M.S. thesis, Dept. of Comp. Sci., NPS, Monterey, CA, USA, 2023. Available: <https://hdl.handle.net/10945/72396>
- [14] E. C. P. Neto, S. Dadkhah, R. Ferreira, A. Zohourian, R. Lu, and A. A. Ghorbani, "CICIoT2023: A real-time dataset and benchmark for large-scale attacks in IoT environment," *Sensors*, vol. 23, no. 13, June 2023. Available: <https://doi.org/10.3390/s23135941>
- [15] B. Allen, "NETCOM 2023 network traffic analysis," unpublished.
- [16] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, June 2002. Available: <https://doi.org/10.1613/jair.953>
- [17] Y. N. Soe, P. I. Santosa, and R. Hartanto, "DDoS attack detection based on simple ANN with SMOTE for IoT environment," in *2019 Fourth International Conference on Informatics and Computing (ICIC)*, 2019. Available: <https://ieeexplore.ieee.org/document/8985853>
- [18] "MaxMind GeoIP2-lite Database". Accessed Sep. 3, 2023. Available: <https://dev.maxmind.com/geoip/geolite2-free-geolocation-data>
- [19] A. Geron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed. Boston, MA, USA: O'Reilly Media, Inc., 2019.
- [20] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, 2001. Available: <https://doi.org/10.1023/a:1010933404324>

- [21] L. Ven and J. Lederer, “Regularization and reparameterization avoid vanishing gradients in sigmoid-type networks,” arXiv, 2021. Available: <https://doi.org/10.48550/arXiv.2106.02260>
- [22] A. F. Agarap, “Deep learning using rectified linear units (ReLU),” arXiv, 2019. Available: <https://doi.org/10.48550/arXiv.1803.08375>
- [23] S. Elfwing, E. Uchibe, and K. Doya, “Sigmoid-weighted linear units for neural network function approximation in reinforcement learning,” arXiv, 2017. Available: <https://doi.org/10.48550/arXiv.1702.03118>
- [24] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for activation functions,” 2017. Available: <https://doi.org/10.48550/arXiv.1710.05941>
- [25] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Caimbridge, MA, USA, 2016. Available: <http://www.deeplearningbook.org>
- [26] “Maximum likelihood estimation and maximum a posteriori estimation,” class notes for Bayesian Methods for Neural Networks, Dept. of Comp. Sci., Naval Postgraduate School, Monterey, CA, USA, winter 2023.
- [27] “Dropout-based variational inference,” class notes for Bayesian Methods for Neural Networks, Dept. of Comp. Sci., Naval Postgraduate School, Monterey, CA, USA, winter 2023.
- [28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, no. 56, 2014. Available: <http://jmlr.org/papers/v15/srivastava14a.html>
- [29] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, “Weight uncertainty in neural networks,” arXiv, 2015. Available: <https://doi.org/10.48550/arXiv.1505.05424>
- [30] “Variational inference,” class notes for Bayesian Methods for Neural Networks, Dept. of Comp. Sci., Naval Postgraduate School, Monterey, CA, USA, winter 2023.
- [31] “The Laplace approximation and harnessing parameter uncertainty,” class notes for Bayesian Methods for Neural Networks, Dept. of Comp. Sci., Naval Postgraduate School, Monterey, CA, USA, winter 2023.
- [32] Y. Gal and Z. Ghahramani, “Dropout as a Bayesian approximation: Representing model uncertainty in deep learning,” in *Proceedings of The 33rd International Conference on Machine Learning*, 2016. Available: <https://proceedings.mlr.press/v48/gal16.html>

- [33] Y. Gal, J. Hron, and A. Kendall, “Concrete dropout,” arXiv, 2017. Available: <https://doi.org/10.48550/arXiv.1705.07832>
- [34] S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft, “Decomposition of uncertainty in bayesian deep learning for efficient and risk-sensitive learning,” in *Proceedings of the 35th International Conference on Machine Learning*, 2018. Available: <https://proceedings.mlr.press/v80/depeweg18a/depeweg18a.pdf>
- [35] J. Davis, J. Zhu, J. Oldfather, S. MacDonald, and M. Trzaskowski, “Quantifying uncertainty in machine learning,” AWS, Aug. 24, 2020. Available: <https://docs.aws.amazon.com/prescriptive-guidance/latest/ml-quantifying-uncertainty/welcome.html>
- [36] Y. Gal, “Uncertainty in deep learning,” Ph.D. dissertation, Dept. of Eng., Univ. of Cambridge, Cambridge, England, UK, 2016.
- [37] C. Strobl, A.-L. Boulesteix, A. Zeileis, and T. Hothorn, “Bias in random forest variable importance measures: Illustrations, sources and a solution,” *BMC Bioinformatics*, vol. 8, no. 1, Feb 2007. Available: <http://doi.org/10.1186/1471-2105-8-25>
- [38] S. Nembrini, I. R. König, and M. N. Wright, “The revival of the gini importance?” *Bioinformatics*, vol. 34, no. 21, May 2018. Available: <https://doi.org/10.1093/bioinformatics/bty373>
- [39] K. Simonyan, A. Vedaldi, and A. Zisserman, “Deep inside convolutional networks: Visualising image classification models and saliency maps,” arXiv, 2014. Available: <https://doi.org/10.48550/arXiv.1312.6034>
- [40] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K.-R. Mueller, “How to explain individual classification decisions,” arXiv, 2009. Available: <https://doi.org/10.48550/arXiv.0912.1128>
- [41] D. Erhan, Y. Bengio, A. C. Courville, and P. Vincent, “Visualizing higher-layer features of a deep network,” working paper, Université de Montréal, Montreal, QC, CA, 2009. Available: <https://api.semanticscholar.org/CorpusID:15127402>
- [42] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-CAM: Visual explanations from deep networks via gradient-based localization,” *International Journal of Computer Vision*, vol. 128, no. 2, Oct 2019. Available: <https://doi.org/10.1007/s11263-019-01228-7>
- [43] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg, “Smoothgrad: Removing noise by adding noise,” arXiv, 2017. Available: <https://doi.org/10.48550/arXiv.1706.03825>

- [44] J. D. Janizek, P. Sturmfels, and S.-I. Lee, “Explaining explanations: Axiomatic feature interactions for deep networks,” arXiv, 2020. Available: <https://doi.org/10.48550/arXiv.2002.04138>
- [45] T. Cui, P. Marttinen, and S. Kaski, “Learning global pairwise interactions with bayesian neural networks,” arXiv, 2019. Available: <https://doi.org/10.48550/arXiv.1901.08361>
- [46] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, “On calibration of modern neural networks,” arXiv, 2017. Available: <https://doi.org/10.48550/arXiv.1706.04599>
- [47] M. P. Naeini, G. F. Cooper, and M. Hauskrecht, “Obtaining well calibrated probabilities using bayesian binning,” in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4410090/>
- [48] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, 2015. Available: <https://dl.acm.org/doi/10.5555/3026877.3026899>
- [49] F. Chollet *et al.*, *Keras*, ver. 2.13.1, 2015. Available: <https://keras.io>
- [50] F. Pedregosa, 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. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, 2011. Available: <https://www.jmlr.org/papers/v12/pedregosa11a.html>
- [51] G. Lemaître, F. Nogueira, and C. K. Aridas, “Imbalanced-learn: A Python toolbox to tackle the curse of imbalanced datasets in machine learning,” *Journal of Machine Learning Research*, vol. 18, no. 17, Jan 2017. Available: <http://jmlr.org/papers/v18/16-365.html>
- [52] Internet Assigned Numbers Authority, “Service Name and Transport Protocol Port Number Registry,” Nov. 17, 2023. Available: <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>

[53] Internet Assigned Numbers Authority, “Protocol Numbers,” Nov. 7, 2023. Available: <https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>

Initial Distribution List

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California



DUDLEY KNOX LIBRARY

NAVAL POSTGRADUATE SCHOOL

WWW.NPS.EDU

WHERE SCIENCE MEETS THE ART OF WARFARE