Towards Detecting and Classifying Network Intrusion Traffic Using Deep Learning Frameworks

Ram B. Basnet[1]*, Riad Shash[1], Clayton Johnson[1], Lucas Walgren[1], and Tenzin Doleck[2]
[1]Colorado Mesa University, Grand Junction, CO 81501 USA
rbasnet@coloradomesa.edu, {rshash, cpjohnson, lawalgren}@mavs.coloradomesa.edu
[2]McGill University, Quebec, CA
tenzin.doleck@mail.mcgill.ca

Abstract

Recent breakthroughs in deep learning algorithms have enabled researchers and practitioners to make significant progress in various hard computer science problems and applications from computer vision and perception, natural language processing and interpretation to complex reasoning tasks such as playing board games (e.g., Go, Chess, etc.) and even overthrowing human champions. Considering the expected acceleration and increase in computer threats, in this article, we explore the utility and capability of deep learning algorithms in the important area of network intrusion detection. We apply and compare various state-of-the-art deep learning frameworks (e.g., Keras, TensorFlow, Theano, fast.ai, and PyTorch) in detecting network intrusion traffic and also in classifying common network attack types using the recent CSE-CIC-IDS2018 dataset. Experimental results show that fast.ai, a highly opinionated wrapper for PyTorch, provided the highest accuracy of about 99% with low false positive and negative rates in both detecting and classifying various intrusion types. Our results provide evidence of the utility of various deep learning frameworks detecting network intrusion traffic.

Keywords: Intrusion Detection, Deep Learning, Network Security, Web Security, Brute Force, Machine Learning, IDS

1 Introduction

In an ever-changing world of computer technology and rapidly evolving cyber threats, attackers and defenders are in a constant loop of cat-and-mouse game. Naturally, a growing body of work is concerned with the protection and security of computer systems. One of the crucial tools—besides network firewall(s)—defenders must add to their security layer a defense-in-depth strategy to effectively deter advanced persistent cyber threats, a Network Intrusion Detection System (NIDS). A NIDS differs from a firewall in that it monitors the flow of incoming and outgoing network traffic, raising alarms if a threat is detected. A firewall on the other hand acts as a "guarded gate" by allowing traffic to and from trusted devices or network defined in the ruleset. If a trusted network is already compromised, a firewall can be useless like those walls and guarded gates that protected the ancient city of Troy after the infamous Trojan Horse was already inside the city during the Trojan War [20].

There are two types of NIDS that vary based on the detection method used. First and earliest used are signature-based NIDS (SNIDS): Snort [16] and Suricata [18] fall into this category. A SNIDS looks for patterns in network traffic and compares it to pre-installed known attack patterns called rules. Although SNIDS are a great solution for detecting known attacks, they fail to detect new or previously unrecognized attacks [33]. Antivirus software are similar in this regard; they rely on continuous virus signature

Journal of Internet Services and Information Security (JISIS), volume: 9, number: 4 (November, 2019), pp. 1-17

*Corresponding author: Ram B. Basnet, Department of Computer Science and Engineering, Colorado Mesa University, 1100 North Aveneue, Grand Junction, CO 81501 USA, Tel: +1 970 248 1682, Web: https://rambasnet.github.io

updates to remain effective [31]. The second type of NIDS is the anomaly-based NIDS (ANIDS), which rely on various statistical machine learning models to automatically learn patterns and create rules that distinguish normal traffic from malicious attacks.

Recent years have seen a surge in the use of machine learning. Indeed, machine learning has generated a lot of attention and curiosity owing to it being introduced in many industrial and consumer applications with high success rates. Many companies discovered the benefits of incorporating deep learning into their products or services that produced incredible results. For example, PayPal, a worldwide payment system, utilizes deep learning to recognize fraud patterns. Google uses various deep learning implementations to power visual/voice recognition, search, research along with providing rapid development APIs giving developers easy access to tools not previously available [5]. The underlying theme is that some applications are too difficult to hard code algorithms to search for patterns in enormous amounts of data. Since ANIDS uses deep neural network models under the hood, it is well suited for detecting new or previously unknown attacks [31]. However, the caveat with ANIDS is that they are limited (like all machine learning applications) by the dataset used to train its model.

Many related works in this stream of research rely on datasets known to have shortcomings and issues to train ANIDS models. Some problems with many existing datasets include but are not limited to: poor representation (outdated attack traffic and no representation of modern attack types), redundancy, anonymity (due to privacy or ethical issues), simulated traffic, lack of traffic diversity, and the lack of an all-encompassing dataset [26]. Furthermore, a recent taxonomy and survey of IDS research of the past decade shows that 63.8% used the KDD'99 dataset [25]. The KDD'99 dataset is known to have redundancy and data corruption negatively affecting experimental results [26, 29].

Against this backdrop, a dataset with proper feature selection, labeled traffic, and no anonymity/-modifications is highly desired [33]. Sharafaldin et al. [26] at the Canadian Institute of Cybersecurity proposed a new IDS dataset that contains seven of the most common attack classes along with unaltered real-world benign traffic where privacy or ethics is not a concern.

With this newly published CSE-CIC-IDS2018 dataset, we are interested in how well we can train hypothetical ANIDS models using popular, high-level machine learning frameworks including: Keras, TensorFlow, Theano, fast.ai, and PyTorch. We built and trained models using this dataset and report the relevant metrics. Our goal is to show that it is possible to train deep neural networks using the frameworks mentioned above and that further progress can be made without a lot of effort owing to the rapid progress and advancement of machine learning APIs.

The rest of the paper is organized as follows. We first cover related work followed by an overview of the deep learning frameworks used. Next an overview and survey of the dataset is presented along with all the modifications that were made prior to training. Following that, we present the experimental setup and results along with a discussion. We end with some concluding remarks and highlight avenues for future research.

2 Related Work

The tremendous progress and breakthroughs in Deep Learning, which "allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction" [28], algorithms and high-level APIs has rendered machine learning an accessible tool, spawning new opportunities and progress in IDS research. A broad range of research has been conducted on intrusion detection. In this section, we present a brief summary of the relevant research on intrusion detection.

2.1 Ensembles

There are multiple recent works that resolve to apply multiple classifiers or autoencoders to the problem of an IDS integrated with artificial intelligence. Mirsky et al. [30] created an algorithm called KitNET that is utilized in their Kitsune program. This newly-developed NIDS sought to solve multiple issues within the community of IDSes, mostly pertaining to resource constraints in a network where an IDS is distributed to the edge devices. This program uses multiple encoders in an attempt to improve accuracy. Ahmim et al. [21] built an algorithm using three classifiers, and tested the algorithm on the CICIDS2017 dataset. The output from two of the classifiers would feed into the third with the original input data. Their use of the REP Tree, JRip Algorithm, and Forest PA classifiers demonstrated a 96.67% accuracy rate. Similarly, [36] attempted to solve the IDS problem using three classifiers. Their correlation-based feature selection Bat algorithm (CFS-BA), when tested on the CICIDS2017 dataset, had marginal effects on most of the data. However, there was a large improvement (compared to the typical DNNs) when feature selection was applied to the SQL injection attacks [36]. Further, the "ensemble" of classifiers outperformed all others they compared it against. Overall, these studies highlight that techniques involving multiple classifiers and autoencoders appear to be successful.

2.2 Surveys

Hindy et al [25] organized and composed a comprehensive taxonomy and survey of IDS published research from 2008-2018. The review identified that 97.25% of surveyed research used machine learning where Artificial Neural Networks (ANN), K-means clustering, and Support Vector Machine (SVM) comprised the most commonly used algorithms. As previously mentioned, 63.8% of surveyed research used the now deprecated KDD'99 (1998-1999) dataset. Considering the constant evolution of cyber threats and methods combined with benign traffic from twenty years ago, outdated datasets like KDD'99 hinders the real-world effectiveness of models despite having high accuracy rates [25].

2.3 Ambitious Projects

Anomaly detection solutions incorporating special localities, patterns, and relationships of the dataset were proposed utilizing Convolutional Networks (CNN). CNNs are specially formulated for multi-dimensional data (images, video, sound) requiring transformation of input into a 2D array mimicking an image. Citing similarities of network traffic and packet structure to human language, CNNs are ideal candidate algorithms to be considered for anomaly detection [35]. Using the NSLKDD dataset, Naseer et al. [32] implemented a Deep Convolutional Neural Network (DCNN) and Convolutional Autoencoder (ConvAE) among other Neural Networks and conventional Machine Learning algorithms. All models were trained without using the testing dataset during training. The test dataset was used exclusively for evaluation. The authors believe that introducing the test dataset during training can result in overfitting - dataset memorization - that severely can impact performance of the model when encountering new data. In this study the CNN based algorithms were among the top performing and showed promising results. Using the KDD'99 data set with feature reduction, Xiao et al. [35] constructed a CNN employing Batch Normalization (BN) that displayed great performance while also observing a considerable reduction in classification time compared to other models using non-CNN algorithms. Short classification processing time is a requirement for real-time IDS and the CNN constructed addresses that criteria very well [35].

Clustering algorithms like those represented in [24] and [22] appear to be useful in providing optimistic solutions. In the case of [24], the use of a multilayer perceptron network in combination with the artificial bee colony and fuzzy clustering algorithms returned 98.42% correct labels, compared to 96.18% for a selection of relevant features technique.

Vinayakumar et al. [34] created a hybrid IDS that utilizes host log information (HIDS) and network information (NIDS). Additionally, this paper serves to categorize and measure the effectiveness of multiple different strategies against different datasets. Experiments show that the decision tree, random forest, and adaboost classifiers have the best and most consistent performance across multiple datasets. Across these experiments, DNNs also demonstrate exceptional performance compared to other machine learning algorithms [34]. This provides further motivation for continuing down the path of DNNs for an IDS.

Khan et al. [27] had tremendous success with a two-stage technique that encouraged feature learning for abnormal traffic through the KDD99 and UNSW-NB15 datasets. The strategy attempts to avoid bogging down the feature learning neural network with benign traffic by taking the abnormality of the traffic into account during its second stage. Since the KDD99 dataset primarily contains outdated data, they also tested on the UNSW-NB15 dataset. The goal of this was to demonstrate the ability of their network to react to unseen data. The network was 99.99% accurate on KDD99 and 89.13% on UNSW-NB15 [27]. The topology put forth by this team is unique and appears to be efficient as it requires only 10 features. Any work moving forward from this could improve the efficiency and accuracy further.

Lee et al. [29] evaluated three deep learning models: Deep Neural Network (DNN), Self-Taught Learning (STL) using an Autoencoder and a Recurrent Neural Network (RNN). Using the KDD'99 and improved NSL-KDD datasets the authors trained models and compared their performance. The DNN was a generic model where inputs get multiplied by weights and then passed to an activation function with back propagation. The STL model consisted of an Unsupervised Feature Learning stage using an Autoencoder followed by passing the output to a soft-max regression (SMR). The RNN applied Long-Short-Term Learning (LSTM) in the model's hidden layer. The highest performing algorithm was STL with Autoencoder yielding an accuracy of 98.9% [29].

3 Deep Learning Frameworks

In this section, we briefly describe the various deep learning frameworks we used in our experiments. Using each framework, we created simple multilayer perceptron neural net models by stacking at least three linear classifiers called layers.

3.1 Keras

The Keras interface is a portable, high-level neural networks API, written in Python, usable for TensorFlow, CNTK, and Theano as a back-end, and was originally developed as part of the Open-ended Neuro-Electronic Intelligent Robot Operating System (ONEIROS) [10]. This interface has multiple advantages when it comes to research and development; primarily being its portability. Since Keras is written to support three major deep learning frameworks and potentially more in the future [10], minimal alterations are required to switch the framework in use.

3.2 TensorFlow

TensorFlow is a machine learning open-source platform developed by Google with extensive industry use. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in machine learning (ML) and developers easily build and deploy ML powered applications [3]. This framework is available in Python, Java, JavaScript, and C++, while also supporting the Internet of Things (IoT) devices [3].

3.3 Theano

Theano is a Python-based deep learning framework primarily distinguished for its speed. This library utilizes various techniques to achieve speeds that, when used in conjunction with a modern GPU, surpasses custom C programs [19] [12]. We chose to include it because of its speed advantage and quick drop-in replacement for Keras backend framework.

3.4 fast.ai

fast.ai [7] was designed with the purpose of making deep learning solutions more accessible to researchers and developers of diverse backgrounds. fast.ai is an open-source Python-based library that uses PyTorch—another popular deep learning framework discussed in the next section—as backend. fast.ai optimizes PyTorch and simplifies the experimentation process for deep learning researchers.

3.5 PyTorch

PyTorch [14] is a deep learning framework popular within academia and industry due to its wide support and research to release features. This development framework is supported by the largest cloud services available while allowing intense customization and optimization. Additionally, PyTorch is available for use in C++, allowing for further optimization and development of models [14].

3.6 Summary

These frameworks were chosen primarily for their usability and applicability. Both Keras-TensorFlow and fast.ai are meticulously ocumented and open-source to decrease the effort required to move from research to deployment. fast.ai has demonstrated intense simplification in its boasts of four-lined training scripts [3], while TensorFlow has multiple case studies from large groups such as Intel, PayPal, GE Healthcare, ARM, Twitter, etc. [5]. PyTorch demonstrates its capabilities through reputation within the research communities while Theano brags intense speeds. Additionally, all of these frameworks are continuously maintained and updated. To demonstrate the ease of training IDS through deep learning, we concluded that these frameworks were appropriate.

4 Our Approach

As mentioned earlier, the challenge with the existing literature is its reliance on datasets with shortcomings and issues to train ANIDS models, including, but not limited to: poor representation (outdated attack traffic and no representation of modern attack types), redundancy, anonymity (due to privacy or ethical issues), simulated traffic, lack of traffic diversity, and the lack of an all-encompassing dataset. The current study builds on existing research to highlight the utility of various deep learning frameworks (e.g., Keras, TensorFlow, Theano, fast.ai, and PyTorch) in detecting network intrusion traffic and also classifying common network attack types. To do so, we exploit a recent CSE-CIC-IDS2018 dataset [26], which overcomes some of the common limitations of previous datasets.

4.1 Dataset

We downloaded and used a recent network intrusion datasets generated and published in 2018 [2]. For more information about the datasets, including the experiments and testbeds used to generate the datasets, we refer the readers to [26]. The datasets consist of benign (normal) network traffic and malicious traffic generated from several different network attacks that are briefly described below.

Brute force attack This is a common network attack type where attackers try to guess online passwords using every key combination or try to check if certain hidden web page (e.g., admin login page) exists using fuzzing techniques in websites. Dataset contains brute force attacks on two common network services SSH and FTP with traffic data labeled as SSH-Bruteforce and FTP-Bruteforce respectively. Traffic generated by Heartbleed attack against the notorious Heartbleed Bug [9] was also categorized into this attack type.

Denial of Service (DoS) attack DoS attack is a very common type of network attack where attackers target a service or network by sending an overwhelming number of bogus requests to temporarily deny the legitimate users. Attackers typically use compromised system(s) also called bots to flood the target system or the network service. Authors of the datasets used commonly available tools such as Hulk, GoldenEye, Slowloris and Showhttptest to generate DoS attack traffic [26]. Distributed Denial of Service (DDoS) is more sophisticated denial of service attack where attackers use multiple botnets consisting of tens of thousands of compromised systems possibly from around the world to flood the target systems or services by generating overwhelming amount of network traffic. Low Orbit Ion Cannon (LOIC) was installed on a group of Windows machines to perform DDoS attack and generate the traffic captured in the dataset [26]. Both DoS and DDoS attacks are categorized either as DoS-GoldenEye or DoS-Slowloris based on the tools used to generate the DoS attack traffic.

Bot attack Attackers use botnet—group of compromised network systems and devices working as a single virtual network—to carry out various nefarious Internet attacks such as stealing and sniffing data via keyloggers and sniffers, send spam, carry out phishing attacks, provide backdoor access to compromised systems, etc. Botnet traffic was generated using the Ares tool—a Python-based botnet that can provide remote shell, file upload/download, capturing screenshots and key logging. [26].

Web attack Web attack category consists of three common attacks: cross-site scripting (XSS) [6] (BruteForce-XSS), SQL-Injection [17] (SQL-Injection), brute force administrative and user passwords (BruteForce-Web), etc. on modern web applications. Damn Vulnerable Web App (DVWA) [1] was used as a victim service and homegrown automated code using Selenium framework was used as an attacking tool to generate Web attack traffic [26].

Infiltration attack Infiltration attacks are usually carried out from inside the networks once some systems are compromised by exploiting critical vulnerabilities in software applications e.g., PDF Readers, Internet browsers, etc. After successful attacks, these systems provide persistent backdoors thereby enabling attackers to launch further internal attacks—technique called pivoting—such as internal IP sweeping, full port scanning and service enumeration using network scanners, launching phishing attacks, etc. To generate infiltration attack traffic, authors used Kali Linux as an attack machine and Windows, Ubuntu and Mac systems in the victim network [26].

Benign traffic In order to generate realistic real-world benign or normal network traffic authors have proposed and used their own system that is responsible for profiling the abstract behavior of human interactions. Dataset contains more than 5 million benign traffic samples mimicking real-world behavior of 25 internet users using various day-to-day applications and services such as HTTP, HTTPS, FTP, SSH, email, etc. [26].

Table 1: Number of samples and network traffic types in each dataset

Dataset	Traffic Type	Number of Samples Remaining	Number of Samples Dropped
02-14-2018.csv	Benign	663,808	3,818
	FTP-BruteForce	193,354	6
	SSH-Bruteforce	187,589	0
02-15-2018.csv	Benign	988,050	8,027
	DoS-GoldenEye	41,508	0
	DoS-Slowloris	10,99	0
02-16-2018.csv	Benign	446,772	0
	DoS-SlowHTTPTest	139,890	0
	DoS-Hulk	461,912	0
02-22-2018.csv	Benign	1,042,603	5,610
	BruteForce-Web	249	0
	BruteForce-XSS	79	0
02-23-2018.csv	Benign	1,042,301	5,708
	BruteForce-Web	362	0
	BruteForce-XSS	151	0
	SQL-Injection	53	0
03-01-2018.csv	Benign	235,778	2,259
	Infiltration	92,403	660
03-02-2018.csv	Benign	758,334	4,050
	BotAttack	286,191	0
Binary-class	Benign	5,177,646	
	Attack	1,414,765	

4.2 Data Analysis and Cleanup

After downloading the dataset, we analyzed the data to learn about their characteristics and cleaned it up if required. The dataset consists of original traffic in pcap files, logs and the preprocessed and labelled, and feature selected CSV files. We used the labelled CSV files with a total of 80 traffic features extracted using CICFlowMeter. The dataset includes normal (benign) and attack traffic comprised of various common attack types—stated in the previous section—distributed among seven CSV files. Because of plenty of available samples and also to keep experiments simple, we dropped samples with Infinity, NaN, or missing values. We converted timestamps to Unix epoch numeric values. We parsed and removed column headers that were repeated in some datafiles. About 20,000 samples were dropped as a result of the data cleanup process. Table 1 shows the summary of datasets we use for our experiments after the data cleanup step. *Number of Samples Remaining* column is the total samples left after dropping number of samples from each category represented in the last column. Each dataset contains certain number of traffic samples belonging to benign and one or more attack types repeated across multitple datasets as summarized in Table 1 and 2.

4.3 Features

Each of the cleaned dataset contains 79 features; out of which 2 (*Destination Port and Protocol*) are treated as categorical using 1-to-n encoding and the rest are all numeric. The original dataset also con-

Table 2: Total number of traffic data samples for each type among all the datasets

Traffic Type	Number of Samples
Benign	5,177,646
FTP-BruteForce	193,354
SSH-Bruteforce	187,589
DOS-GoldenEye	41,508
Dos-Slowloris	10,990
Dos-SlowHTTPTest	139,890
Dos-Hulk	461,912
BruteForce-Web	611
BruteForce-XSS	230
SQL-Injection	87
Infiltration	92,403
BotAttack	286,191
Total Attack Samples	1,414,765

tains data with best feature selection; we do not use this data in our experiments, however. Some examples of the numeric features are *Timestamp, Flow Duration, SYN Flag Count, Packet Length Min, Packet Length Max*, etc. More details about all the features can be found in [26].

5 Experimental Setup

We experimented with several state-of-the-art deep learning frameworks discussed above in section 3 for each dataset. There are two common approaches to evaluate machine learning models: n-fold cross-validation (normally 10-fold) and train-test split (normally 70-30 or 80-20). n-fold cross-validation is normally used when the number of samples in some categories are small or disproportionate whereas train-test split is used when the dataset contains a large number of samples in every category. We used both the approaches where appropriate.

We combined all the individual files and generated two other datasets to experiment with binary-class and multi-class classification. For binary-class classification we relabeled all the attack traffic as 1 and benign traffic as 0. For multi-class classification we either left the labels as they were (textual) or converted them into one-hot *(a.k.a. one-of-K scheme)* encoding depending on the framework requirements.

5.1 Performance Metrics

There are several metrics available to evaluate and compare the performance of machine learning classifiers. We used the following widely used performance metrics in literature to evaluate our deep learning classifiers:

Accuracy Accuracy is the percentage of correctly classified samples over the total number of samples evaluated. Accuracy by itself may not be a good performance measure when the number of samples being evaluated is disproportionate. Therefore, we also generated and reported confusion matrices that overcome the drawbacks of accuracy measure.

Confusion matrix Confusion matrix provides a visual way to display the detail performance of classifiers showing total number of samples correctly and incorrectly classified in each category. Confusion matrix also helps us calculate false positive rate—also called recall or sensitivity—which is the % of negative class samples classified as positive class and false negative rate that is % of positive class samples incorrectly classified as negative—another important metrics used in classification problems that can be further used to calculate commonly used measures such as Precision and F1-measure. A confusion matrix is very useful in evaluating not only binary-class classifier, but also multi-class classifiers as in our case. Some datasets are fairly large—especially the ones that include all the samples for Binary-Class and Multi-Class classification experiments—with more than 6.5 million samples and file size of more than 2 GB. A regular computer with 12 GB of memory took several hours for some experiments using CPU while running out of memory for larger datasets. This inspired us to setup an NVIDIA TITAN Xp GPU described below. We then repeated all our experiments to compare the time differences and accuracies.

5.2 GPU Setup

We mostly used two different environments where we simultaneously ran our experiments. First was a cloud service called Salamander which provided ML ready servers [15]. We rented a cloud server with 4 vCPUs, 61 GB RAM running and an NVIDIA Tesla K80 GPU.

We built the second GPU testbed in our research lab. The GPU test bed consists of a virtual machine running Ubuntu 18.04 64-bit LTS in a VMware ESXi server environment. We allocated 12 vCPUs and 26 GB of RAM to the VM. When running fast.ai experiments on CPU to compare CPU and GPU execution time, the vCPU count was increased to 18 to decrease training time. NVIDIA Titan Xp GPU was made available to it as PCI Device 0. The latest NVIDIA graphics drivers were installed along with Anaconda Python package manager, virtual environment and required software packages for our deep learning experiments such fast.ai, pytorch, tensorflow-gpu, cudatoolkit, cudnn, etc. Our setup also provided HTML5-based clientless remote terminal access using Apache Guacamole for remotely accessing the VM and running experiments remotely.

We also experimented with Google Colab [8] environment, but the default machine provided by Colab was not adequate enough to run our experiments especially on binary-class and multi-class classification experiments where the amount of RAM required exceeded 12 GB.

6 Results

We applied the deep-learning frameworks on each dataset and recorded performance results such as accuracy, standard deviation, time, etc. We experimented with various parameters provided by each framework to create optimal deep learning classifier to use as the baseline classifier. We also experimented normalizing the numeric features, but the performance difference was statistically insignificant to justify normalizing numeric values for all the experiments. Table 3 presents a summary of the performance results on each dataset using CPU. Since CPU experiments were taking several weeks with Keras-TensorFlow on Binary-Class and Multi-Class classification on the whole dataset, we decided not to run Keras-Theano experiments on these datasets. Moreover, since Theano results are very similar to Keras-TensorFlow for individual files, we felt that the Keras-Theano would very unlikely achieve superior results to fast.ai.

Figure 1 shows that accuracy results for individual dataset over several epochs. Accuracies peak quickly after just a couple of epochs for most datasets. Figure 2 illustrates the accuracy results over 10 Epochs for Multi-class and Binary-class experiments.

Table 3: Performance results on each dataset using CPU

Dataset	Framework	Accuracy (%)
02-14-2018.csv	fast.ai	99.85
	Keras-TensorFlow	98.80
	Keras-Theano	98.58
02-15-2018.csv	fast.ai	99.98
	Keras-TensorFlow	99.32
	Keras-Theano	99.17
02-16-2018.csv	fast.ai	100.00
	Keras-TensorFlow	99.84
	Keras-Theano	99.41
02-22-2018.csv	fast.ai	99.87
	Keras-TensorFlow	99.97
	Keras-Theano	99.97
02-23-2018.csv	fast.ai	99.92
	Keras-TensorFlow	99.94
	Keras-Theano	99.95
03-01-2018.csv	fast.ai	87.00
	Keras-TensorFlow	72.20
	Keras-Theano	72.04
03-02-2018.csv	fast.ai	99.97
	Keras-TensorFlow	98.12
	Keras-Theano	93.95
Multi-Class	Keras-TensorFlow	94.73
	fast.ai	98.31
Binary-Class	Keras-TensorFlow	94.40
	fast.ai	98.68

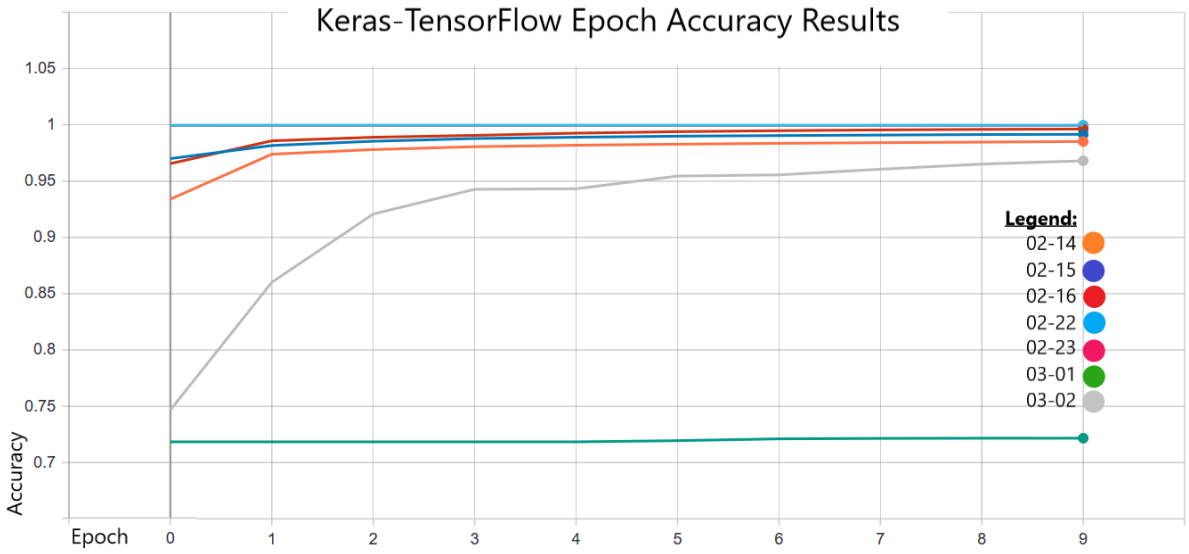


Figure 1: Accuracy results on all datasets over 10 epochs

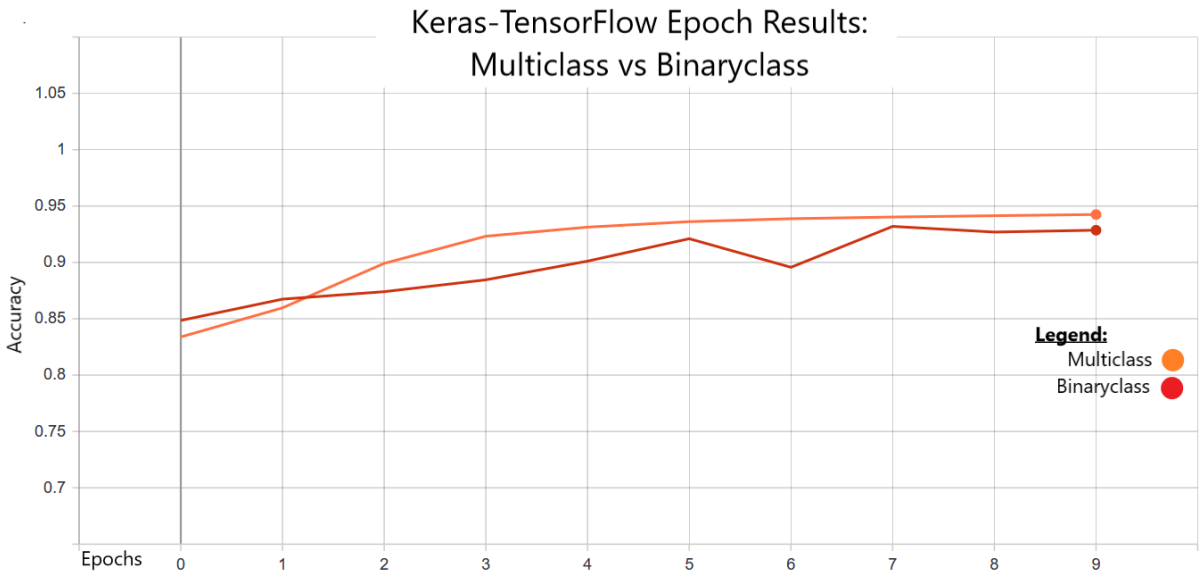


Figure 2: Accuracy results on binary and multi-class classififcation experiments over 10 epochs

Table 4: Attack sample distribution and detection accuracy on each dataset

Dataset	% of Attack Sample	% Attack Flagged as Attack	% Benign Flagged as Attack
02-14-2018.csv	36.46	100.00	0.00*
02-15-2018.csv	5.04	99.85	0.00*
02-16-2018.csv	57.39	100.00	0.00*
02-22-2018.csv	0.00*	0.02	0.00
02-23-2018.csv	0.00*	61.61	0.00*
03-01-2018.csv	28.16	73.19	10.16
03-02-2018.csv	27.40	99.85	0.00*
Binary-Class	21.50	94.60	0.21
Multi-Class	21.50	93.9	0.48

*small non-zero value

Table 4 summarizes the attack sample distributions and detection accuracy on each dataset. In Table 5, we provide the results of the 10-fold cross validation time comparison using the CPU and GPU on our GPU setup.

6.1 Confusion Matrix

As mentioned earlier, confusion matrix provides a visual way to display the detail performance of classifiers showing total number of samples correctly and incorrectly classified in each category. The confusion matrices were generated from fast.ai experiments for each dataset as presented in Table 6 and Figure 3.

7 Discussion

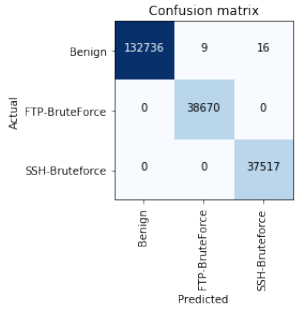
From the performance results (Table 3), we can observe that the various frameworks tested resulted in high accuracy rates. fast.ai framework API was easier to use compared to Keras-TensorFlow Keras-

Table 5: 10-fold cross validation time comparison using CPU and GPU on our GPU setup

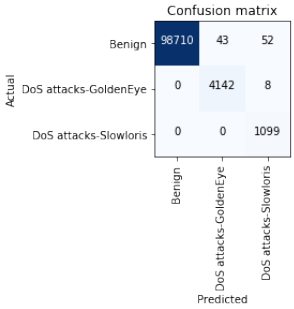
Dataset	CPU Time (mins)	GPU Time (mins)
02-14-2018.csv	1193.84	100.36
02-15-2018.csv	1299.55	103.16
02-16-2018.csv	433.63	104.51
02-22-2018.csv	3091.34	102.83
02-23-2018.csv	1938.74	104.43
03-01-2018.csv	80.07	33.23
03-02-2018.csv	1503.18	104.34
Binary-Class	19441.55	632.36

Table 6: Confusion metrices generated from fast.ai experiments on each dataset

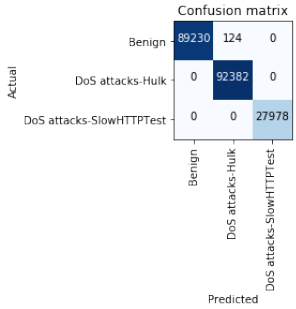
02-14-2018.csv



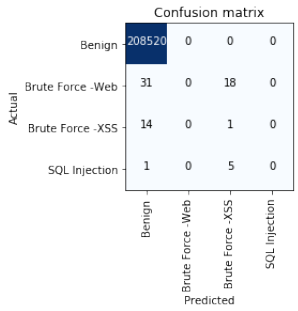
02-15-2018.csv



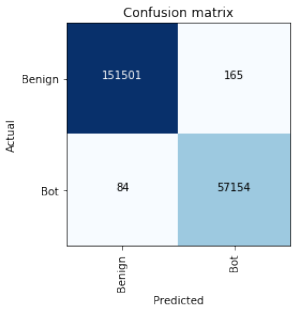
02-16-2018.csv



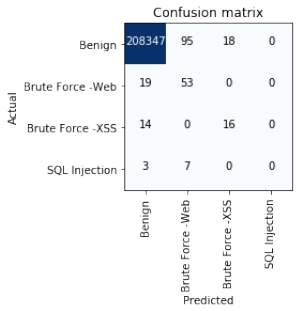
02-22-2018.csv



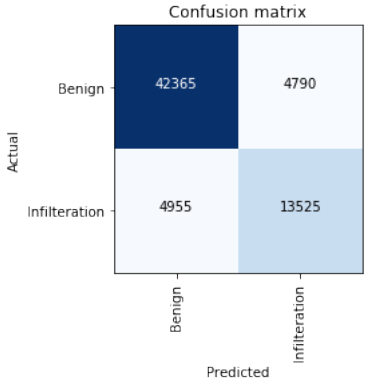
02-23-2018.csv



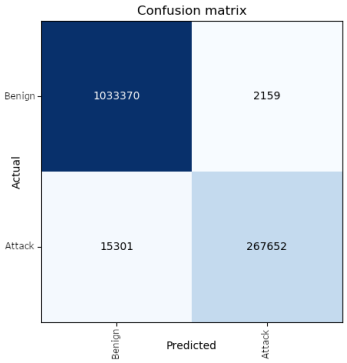
03-01-2018.csv



03-2-2018.csv



Binary-Class



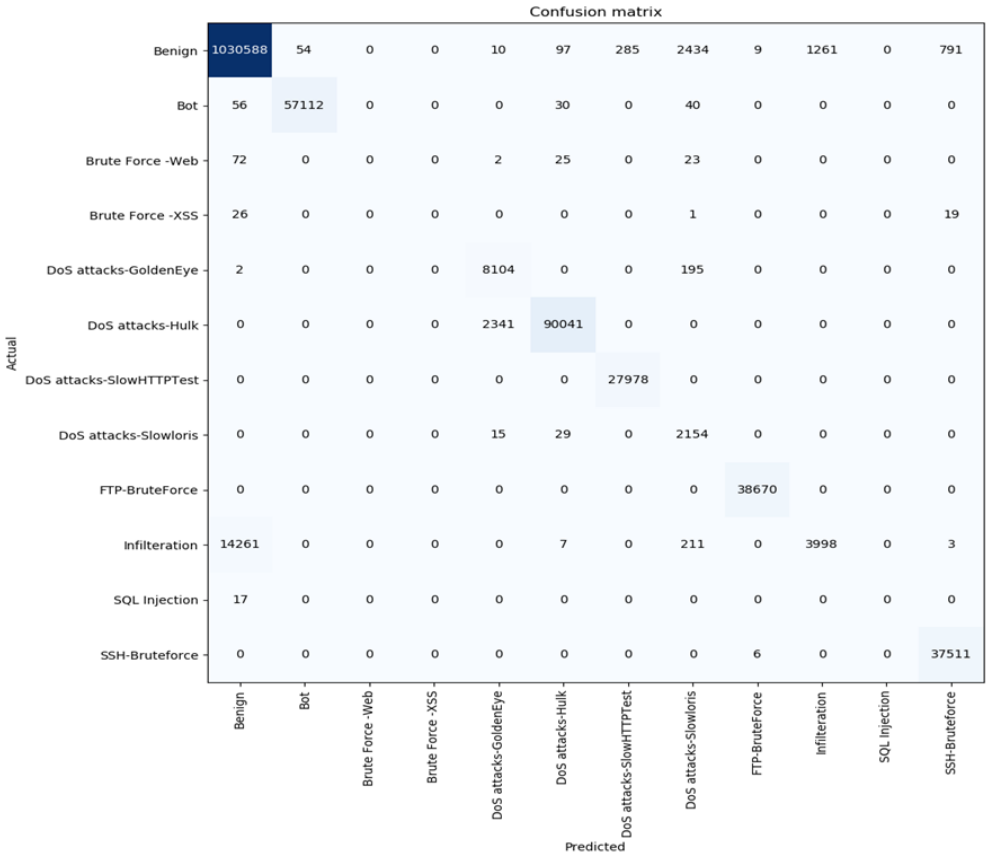


Figure 3: Confusion matrix generated from fast.ai experiments for multi-class classification

Theano and also consistently gave slightly better performance (accuracy) results compared to both TensorFlow and Theano frameworks. This could be because fast.ai automatically chooses and fine tunes various hyper-parameters out-of-the-box providing optimal and easy-to-interpret models via its APIs [11]. Scripts incorporating fast.ai utilized up to 23-54% of the GPU as well. Even though overall accuracies are stellar (99%) for all individual datasets except for 03-01-2018.csv, the false positive and false negative rates are surprisingly much higher for 02-22, 02-23, and 03-01-2018.csv files. This discrepancy is due to the fact that the number of attack samples ($< 0.01\%$) is far less compared to the number of benign samples for 02-22 and 02-23-2018.csv datasets. One possible explanation is that our DL models might be underfitted due to proportionally much smaller number of attack samples. Interestingly on 03-01-2018.csv dataset, accuracy is very low compared to other datasets. A larger number of infiltration samples were misclassified as benign resulting in high false negative rates as well as over 10% of benign samples were misclassified as attack thus driving the overall accuracy down. Since the infiltration attack traffic was generated from the compromised systems within the victim network, we hypothesize that the traffic patterns are very similar in characteristics to benign samples and vice versa and the models were unable to distinguish one from another well. On 02-18-2018.csv dataset, 100% of attack samples were correctly classified as attacks whereas a smaller number of benign samples were classified as FTP-BruteForce (9) and SSH-BruteForce (16) with the overall false negative rate of less than 0.01%.

When all the data samples were combined into one dataset we obtained reasonable results of about 95% accuracy for binary and multi-class classification using Keras-TensorFlow and significantly better results of about 99% accuracy using fast.ai for the same experiments. TensorFlow utilizing GPUs gave

similar accuracies and loss metrics using the Salamander.ai cloud service. What surprised us initially was the lack of any considerable performance improvement when using our GPU. Further investigations pointed to a data starvation problem. Owing to the tremendous ability of GPU's parallel processing, GPUs can train deep neural networks incredibly fast [4]. However, when combining CPUs with a high-end GPU, the bottleneck almost always ends up being the CPUs. We suspect our scripts caused the GPU to keep waiting for the CPU to feed it data; thus, resulting in our observed low utilization range of 7% - 15%.

One solution is to rewrite our scripts to incorporate TensorFlow's Dataset API (tf.data) which enables the construction of asynchronous and optimized data input pipelines [13]. This enables the CPU to deliver data to the GPU before the current step has finished yielding an increase in GPU utilization effectively reducing training time [23]. We leave this for our future work.

7.1 Research Challenges

It bears mentioning that there were some challenges in terms of computational time and resource bottlenecks. We should note that for CPU experiments, we first started with Keras-TensorFlow on Binary-Class and Multi-Class classification on the whole dataset. The experiments took several weeks to complete. Given that Keras-TensorFlow and Keras-Theano produce similar results for individual files, we decided not to run additional experiments with Keras-Theano on Binary-Class and Multi-Class classification on the whole dataset. In future work, we can explore the differences (if any) between Keras-TensorFlow and Keras-Theano on Binary-Class and Multi-Class classification on the whole dataset. Furthermore, for binary and multi-class classification using TensorFlow (utilizing GPUs), we noticed that there was little performance improvement when using our GPU. As we probed the problem further, we realized that there was a data starvation issue. The low utilization ranges of 7%- 15% could be the result of the GPU waiting for the CPU to feed it data. We know that GPUs are incredibly fast at training deep neural networks; however, when combining CPUs with a high-end GPU, CPUs tend to be the bottleneck.

8 Conclusion and Future Work

Deep learning has gained significant attention as advancements in deep learning have enabled new opportunities for both research and practice [33-35]. In this article, we compared three state-of-the-art deep learning frameworks namely Theano, TensorFlow, and fast.ai in detecting and classifying various intrusion types. We applied the aforementioned frameworks on a recent dataset published in 2018. From our results, we observed that fast.ai outperformed the other two frameworks consistently among all the experiments yielding accuracy results up to 99% with a very low false positive and negative rates of less than 1%. Furthermore, we ran our experiments on CPU and GPU and noticed that fast.ai's GPU utilization was much higher compared to Keras framework using both Theano and TensorFlow as backends.

When comparing just the training and testing times by fast.ai, we noticed the GPU training and testing times were anywhere from 3 to more than 20 times faster than the CPU counterparts. As a future work, we would like to test our models with additional datasets to further validate the models and eventually build, deploy and test the deep learning powered IDS system in a lab environment mimicking real-world IT infrastructures. Hyperparameters tuning and comparing various deep learning algorithms such as RNN (Recurrent Neural Network), CNN (Convolutional Neural Network), Reinforcement Learning (RL) on the same dataset are some interesting future directions as well.

8.1 Acknowledgment

This research project was supported by the state of Colorado through funds appropriated for cybersecurity law dubbed "Cyber Coding Cryptology for State Records." Any opinions, findings and conclusions, or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding sources.

References

[1] Dvwa — damn vulnerable web application. http://www.dvwa.co.uk/ [Online; Accessed on November 2, 2019].

[2] Ids 2017. https://www.unb.ca/cic/datasets/ids-2017.html [Online; Accessed on November 2, 2019].

[3] About fast.a. https://www.fast.ai/about/ [Online; Accessed on November 2, 2019], 2019.

[4] Basics of gpu computing for data scientists. https://www.kdnuggets.com/2016/04/basics-gpu-computing-data-scientists.html [Online; Accessed on November 2, 2019], 2019.

[5] Case studies — tensorflow. https://www.tensorflow.org/about/case-studies [Online; Accessed on November 2, 2019], 2019.

[6] Cross-site scripting (xss) - owasp. https://www.owasp.org/index.php/Cross-site_Scripting_(XSS) [Online; Accessed on November 2, 2019], 2019.

[7] fastai. https://docs.fast.ai/ [Online; Accessed on November 2, 2019], 2019.

[8] Google colaboratory. https://colab.research.google.com/notebooks/welcome.ipynb [Online; Accessed on November 2, 2019], 2019.

[9] Heartbleed bug. http://heartbleed.com/ [Online; Accessed on November 2, 2019], 2019.

[10] Home - keras documentation. https://keras.io/ [Online; Accessed on November 2, 2019], 2019.

[11] How to make the most out of the top-down fast.ai course, practical deep learning for coders, part 1. https://medium.com/@georgezhang_33009/how-to-make-the-most-of-the-top-down-fast-ai-courses-ae70814c736f [Online; Accessed on November 2, 2019], 2019.

[12] "optimizations - theano 1.0.0 documentation". http://www.deeplearning.net/software/theano/optimizations.html#optimizations [Online; Accessed on November 2, 2019], 2019.

[13] Performance — tensorflow core. https://www.tensorflow.org/guide/performance/overview#input_pipeline_optimization [Online; Accessed on November 2, 2019], 2019.

[14] Pytorch, 2019. https://pytorch.org/features [Online; Accessed on November 2, 2019].

[15] Salamander. =https://salamander.ai/ [Online; Accessed on November 2, 2019], 2019.

[16] Snort - network intrusion detection and prevention system. http://www.csie.ntu.edu.tw/~cjlin/papers/features.pdf [Online; Accessed on November 2, 2019], 2019.

[17] "sql injection - owasp. https://www.owasp.org/index.php/SQL_Injection [Online; Accessed on November 2, 2019], 2019.

[18] Suricata. https://suricata-ids.org/ [Online; Accessed on November 2, 2019], 2019.

[19] "theano at a glance - theano 1.0.0 documentation". http://www.deeplearning.net/software/theano/introduction.html [Online; Accessed on November 2, 2019], 2019.

[20] Trojan war. https://www.history.com/topics/ancient-history/trojan-war [Online; Accessed on November 2, 2019], 2019.

[21] A. Ahmim, L. Maglaras, M. Ferrag, M. Derdour, and H. Janicke. A novel hierarchical intrusion detection system based on decision tree and rules-based models. In *Proc. of the 15th International Conference on Distributed Computing in Sensor Systems (DCOSS'19), Santorini Island, Greece*, pages 228–233, May 2019.

[22] T. Bajtos, A. Gajdos, L. Kleinova, K. Lucivjanska, and P. Sokol. Network intrusion detection with threat agent profiling. *Security and Communication Networks*, 2018:1–17, March 2018.

[23] T. Genthial. Building a data pipeline. https://cs230-stanford.github.io/tensorflow-input-data.html [Online; Accessed on November 2, 2019], 2019.

[24] B. Hajimirzaei and N. Navimipour. Intrusion detection for cloud computing using neural networks and artificial bee colony optimization algorithm. *ICT Express*, 5(1):56–59, March 2019.

[25] H. Hindy, D. Brosset, E. Bayne, A. Seeam, C. Tachtatzis, R. Atkinson, and X. Bellekens. A taxonomy and survey of intrusion detection system design techniques, network threats and datasets. https://arxiv.org/pdf/1806.03517.pdf [Online; Accessed on November 2, 2019], 2018.

[26] A. L. I. Sharafaldin and A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proc. of the 4th International Conference on Information Systems Security and Privacy (ICISSP'18), Funchal, Madeira, Portugal*, pages 108–116. ICISSP, January 2018.

[27] F. Khan, A. Gumaei, A. Derhab, and A. Hussain. "tsdl: A two-stage deep learning model for efficient network intrusion detection". *IEEE Access*, 7:30373–30385, March 2019.

[28] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[29] B. Lee, S. Amaresh, C. Green, and D. Engels. Comparative study of deep learning models for network intrusion detection. *SMU Data Science Review*, 1(1):8:1–8:13, 2018.

[30] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection, 2018. https://arxiv.org/pdf/1802.09089.pdf [Online; Accessed on November 2, 2019].

[31] G. Nascimento and M. Correia. Anomaly-based intrusion detection in software as a service. In *Proc. of the 41st International Conference on Dependable Systems and Networks Workshops (DSN-W'11), Hong Kong, China*, pages 19–24. IEEE, 2011.

[32] S. Naseer, Y. Saleem, S. Khalid, M. K. Bashir, J. Han, M. M. Iqbal, and K. Han. Enhanced network anomaly detection based on deep neural networks. *IEEE Access*, 6:48231–48246, August 2018.

[33] A. J. Q. Niyaz, W. Sun and M. Alam. A deep learning approach for network intrusion detection system. In *Proc. of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (BICT'15), New York, USA*, pages 21–26. ICST, December 2016.

[34] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman. Deep learning approach for intelligent intrusion detection system. *IEEE Access*, 7:41525–41550, April 2019.

[35] Y. Xiao, C. Xing, T. Zhang, and Z. Zhao. An intrusion detection model based on feature reduction and convolutional neural networks. *IEEE Access*, pages 42210–42219, March 2019.

[36] Y. Zhou and G. Cheng. An efficient network intrusion detection system based on feature selection and ensemble classifier. https://arxiv.org/abs/1904.01352 [Online; Accessed on November 2, 2019], 2019.

Author Biography

Ram B. Basnet is an associate professor of Computer Science at Colorado Mesa University (CMU). He received his BS in Computer Science from CMU in 2004 and MS and PhD in Computer Science from New Mexico Tech in 2008 and 2012, respectively. His research interests are in the areas of information assurance, machine learning, and computer science pedagogy.

Riad Shash received his BS in Computer Science from Colorado Mesa University (CMU) in 2019 and is pursuing his MS degree in Computer Science at the University of Colorado Boulder. He is a former treasurer of the Cybersecurity club at CMU.

Clayton Johnson is pursuing his BS degree in Computer Science, Mathematics, and Physics at Colorado Mesa University (CMU) and Professional Certificate in Cyber Security. He was a president of the Computer Science club at CMU, lab assistant, and a tutor for CS.

Lucas Walgren is pursuing his BS degree in Computer Science and Professional Certificate in Cybersecurity at Colorado Mesa University (CMU). He is an officer of the Cybersecurity club at CMU.

Tenzin Doleck received his PhD from McGill University in 2017. He is currently a post-doctoral fellow at the University of Southern California.