NETWORK ANOMALY DETECTION

FYP16021 Final Report

by Tien-Hsuan WU
Supervised by Dr. S. M. YIU

April 2017

ABSTRACT

Most anomaly detection systems found in literature are based on data mining methods, and thus are limited to predefined features. The limitations of the existing systems will significantly reduce the performance of the systems if the features are not selected properly. Recently, as the computation power increases, deep learning becomes a popular area of research. In this project, we used deep learning as the model for anomaly detection, and Keras as the library to implement. The models were first trained with internal network data to classify network packets according to the application layer protocol. The models were then modified and trained to identify anomalies and the performance was evaluated. We obtained training data from Department of Computer Science for packet classification task, and we used publicly available data for the anomaly detection task. The promising result of the anomaly detection shows the potential to integrate deep learning into network intrusion detection system.

ACKNOWLEDGEMENT

I would like to thank my supervisor, Dr. S. M. Yiu, for his unfailing support of my project and teaching me research skills with his patience and immense knowledge.

I would like to express my gratitude to the second examiner, Dr. H. F. Ting, for his effort in reviewing the report.

I wish to acknowledge the help in data collection provided by technical staff of the Department.

Special thanks are given to peers who have offered me valuable suggestions to my project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ANN: artificial neural network

CNN: convolutional neural network

ELU: exponential linear unit

GiB: giga binary byte, 2^{30} bytes

IDS: intrusion detection system

KB: kilobyte in decimal, 1,000 bytes

LReLU: leaky rectified linear unit

LSUV: Layer Sequential Unit Variance

MLP: multiple layer perceptron

PCAP: packet capture (file)

ReLU: rectified linear unit

TCP: transmission control protocol

I. Introduction

The Internet is expanding and its scale is increasing as more and more devices are connected to the Internet. In November 2016, Google indexed 46 billion webpages, and the annual global Internet traffic was expected to reach 1 zettabyte (10^{21} bytes) by the end of 2016. With the popularization of the Internet, its usage has become necessary in various areas. However, alongside the advantages of Internet use is the increasing potential of cyber attack. According to Symantec, there were 54 zero-day (unseen) vulnerabilities discovered each week in 2015, which is twice as many as those in 2014 [1]. Therefore, without appropriate security measures, it is likely that the systems will be compromised, causing great losses to individuals and companies. Intruders may gain unauthorized privileges, or simply overload the server and make it unavailable. Both of these may incur great loss for the system owners.

In order to protect the computers from being hacked, intrusion detection systems (IDS) can be installed. Some common open source IDS are Snort [2] and Suricata [3]. With IDS installed, whenever a system encounters unauthorized access, it can respond by refusing such access request. Moreover, it can generate alerts for human to inspect if there is any system defect.

In our project, we focus on improving the accuracy of intrusion detection methods with a deep learning model as the backend. The entire model was built using Keras, and thus we can take advantage of its high modularity to achieve fast prototyping. A simple code snippet written with Keras library that builds a deep learning model with 3 hidden layers is shown in Figure 1. We first studied optimization techniques with MNIST hand-written digit dataset [4] and CIFAR tiny image dataset [5]. We then focused on determining and building the most effective model. After the model was built, we trained and tested it with some network data and evaluated its performance according to its accuracy. We then analyzed the model and examine the misclassified data to determine the modifications to be made. Finally, we provided suggestions to modifications.

```python
model = Sequential()
model.add(Dense(300, input_shape=(500,))) #hidden layer
model.add(Dropout(0.2))
model.add(Activation('relu'))
model.add(Dense(300)) #hidden layer
model.add(Dropout(0.2))
model.add(Activation('relu'))
model.add(Dense(300)) #hidden layer
model.add(Dropout(0.2))
model.add(Activation('relu'))
model.add(Dense(10)) #output layer
model.add(Activation('softmax'))

sgd = SGD(lr=0.01)

model.summary()

model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

history = model.fit(X_train, Y_train,
                    batch_size=32, nb_epoch=100,
                    verbose=2, validation_data=(X_test, Y_test))

score = model.evaluate(X_test, Y_test, verbose=2)
```

FIGURE 1. Fast prototyping of Keras.

The rest of this report is organized as follows. Section II describes the background of intrusion detection system and deep learning. Section III presents the related works to our project. We discuss the methodology in Section IV. The current deep learning optimization research is summarized in Section V, and our experiment results with network data are reported in Section VI. Finally, Section VII concludes this report.

II. BACKGROUND

A. INTRUSION DETECTION

Intrusion detection systems can be classified into three categories: signature detection systems, anomaly detection systems, and hybrid systems [6, 7]. A signature detection system maintains a misuse database which contains the patterns of abnormal traffic. When a packet arrives, the system will compare it with the misuse database to determine whether such packet is normal. The advantage is that signature detection systems generate a low false positive rate when the misuse database is reliable. This is due to the fact that intrusions detected are supposed to have a high similarity with the abnormal packets. For an anomaly detection system, it uses the pattern generated from normal traffic as the baseline. Any pattern that deviates from the normal traffic is considered anomalous. The advantage is that it can detect unseen (zero-day) attacks. Hybrid systems combine both techniques used in signature detection systems and anomaly detection systems.

B. DEEP LEARNING

Traditional machine learning methods, such as support vector machines and decision trees, are able to find patterns from a set of data. Deep learning is also able to do this. However, what differentiates deep learning from machine learning is the number of learning methods used. In machine learning, typically a single method is used; whereas in deep learning, we can use multiple methods, with each method being based on the result of previous one. The deep structure comes from the multiple steps between the input and the output [8].

One advantage of deep learning the high capacity of the model. In traditional machine learning where only one method is used, it is less desirable to apply it to solve complicated problems such as image recognition and natural language processing. Even if it fits the training data well through more training iterations (epochs), the model is likely to perform poorly on unseen data. By contrast, the problem can be resolved by setting different classification methods for each step when deep learning is used. With a higher capacity of the model, problems of various types can be solved. We can also expect the model is not overfit so that the model trained can truly reflect its performance on unseen data.

In reality when intrusion detection system is implemented, the IDS needs to accommodate to a wide range of network flow patterns. Moreover, when the size of the deep learning model increases, the model is able to capture more features and raise the precision of the detection system. Based on these reasons, we chose deep learning framework to perform anomaly detection tasks.

III. RELATED STUDIES

Data mining techniques and machine learning algorithms can be applied to intrusion detection systems, and these techniques have been extensively studied in the past decade. Clustering and classification are some techniques used in intrusion detection systems (IDS). Münz, Li & Carle proposed an anomaly detection system using k-means algorithm which combines both classification and outlier detection [9]. In [10, 11], the authors combined the k-means clustering with naïve Bayes classification. In [12], the authors further utilized the result from k-means clustering as new features for naïve Bayes classifier. In [13], naïve Bayes classifier is combined with decision tree algorithms.

The abovementioned methods operate on network features only, namely, the connection records. To take payload data from packets into consideration, we need some other techniques. PAYL is a histogram-based classification method that takes payload data as input. It builds a histogram from the input, with frequency of each byte pattern being a bin (see Figure 2), and compares the histogram built from the data with baseline [14].

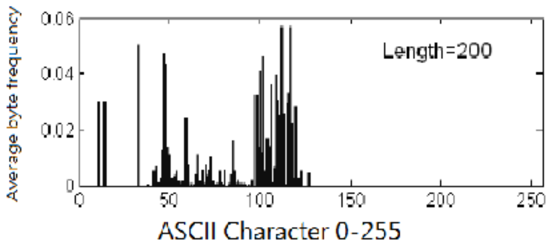


FIGURE 2. Example of byte distribution for a 200-byte packet.

Deep learning techniques can also be implemented to detect anomalies. Niyaz [15] implemented two-stage process of self-taught learning for network anomaly detection. In the first stage, a sparse autoencoder was used for unsupervised feature learning. After the features were learned, they were passed to a softmax classifier in the second stage for anomaly detection. This model can only operate on network features only instead of the entire payload. The performance is evaluated with KDD dataset [16], and precision for 5-class classification based on predefined labels in KDD is 85.44%. Another deep learning application on network data is studied by Wang [17]. He studied the structures of artificial neural network and stacked autoencoder, and then built a system that could classify TCP packets according to their application layer protocols. The weight coefficients of the first hidden layer can be viewed as the importance of the byte features. Most of the protocols can reach 99% precision in the experiment. He suggested that misclassified packets may be anomalous; however, the relationship between those misclassified packets and anomalies was not thoroughly. In our project, we first implemented a similar deep learning model and studied the misclassified packets. We then extended such model to detect anomaly packets.

IV. Methodology

Given a network packet, our first goal is to determine what application layer protocol is used. In Section II.B, we have discussed processing network packets with deep learning models. As packets arriving at a host may be scattered, the order and the temporal information will not be useful. Therefore, we can use a deep learning feedforward network as it only captures the features of each single packet. We used a simple one as an initial implementation, and modify the network to achieve better results.

To implement a deep learning network, it requires some training data to learn from and testing data to evaluate its performance. We retrieved internal-generated Internet packets from Department of Computer Science, University of Hong Kong for the packet classification task. We also used the CTU-13 dataset [18], which consists of the botnet packet capture, as the training and testing data for the anomaly detection. Details of the dataset will be discussed in Section VI.C.

The next step is to adjust the data so that they are suitable for deep learning. We first joined the payload of the packets if they belong to the same session. Since the model requires fixed size of input, we had to truncate payloads that were too large and padded the payloads that were too small. We set the size of the payloads to 500 bytes since previous research showed that most important bytes are located in the first few bytes (see Figure 3) [17]. We did not merely use the first 100 bytes because it contains too limited information, and we did not use the entire 1 KB for the reason that it causes too much time in training process. The detailed data processing steps can be found in Section VI.A.

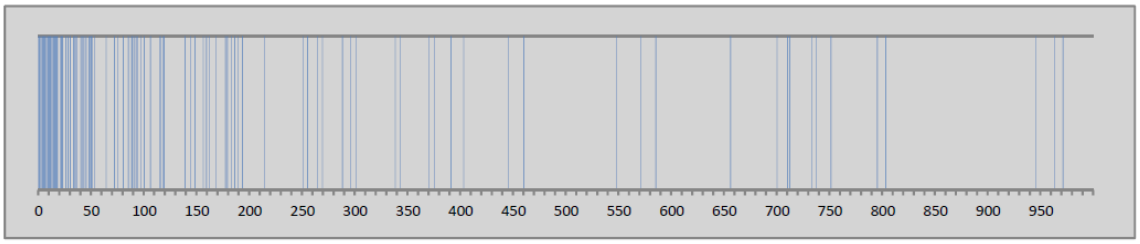


FIGURE 3. The most important 100 locations of the data.

In order to modify the feedforward network, we had to first determine the factors that affect the performance of the system, such as the number of layers of the system and the learning algorithms used. After the neural network was built, we studied the relationship between the byte features and their importance in protocol identification. We would verify the result of the byte features by examining the packets of the neural network.

Our ultimate goal is to identify packet outliers. Based on the system we had built, we used similar techniques to build the model and then improve its performance. The difference is that in the training set, we added some anomalous data, including attack traffic in the Internet. For the output, it should be able to distinguish anomaly packets from normal flow.

We chose Keras for implementing deep learning models because its high modularity allows the program development and prototyping in a relative simple manner. In addition, as we focus on evaluating the existing deep learning structures and the effects of parameters, we can exploit the advantages of modularity and avoid extensive changes in source code during training phase. Another concern in development is the compatibility and extensibility with other packages. In python, which Keras is written in, there are other machine learning packages that can be utilized. Therefore, Keras is particularly suitable for our project.

At the time we started the implementation of the deep learning model, the version of Keras was 1.1.0. Although Keras has recently released an update to 2.0.0, we still use our original version so the efficiency can be compared.

V. Deep Learning Optimization

For a deep learning neural network to classify data samples correctly, the model is required to have enough capacity and the training data have to represent general situations. In this section, we will discuss some methods to build a model that achieves a better performance. Since there is not much research related to deep learning optimization with network data, we used the datasets that are widely studied in the literature for experiment.

A. Dataset for Optimizing Deep Learning Neural Network

MNIST hand-written digit database [4] is one of the widely used labeled data for machine learning. Each grayscale written digit is normalized in size and centered in a 28x28 image. The dataset contains 60,000 training samples and 10,000 test samples for performance evaluation.

Another dataset that is popular in deep learning is CIFAR-100 colored tiny image dataset. Each image has a coarse label and a fine label, which belongs to one of the 20 superclasses and one of the 100 classes (see Table 1). In our experiment, we used fine labels only. The size of each image is 32x32.

FIGURE 4. The MNIST dataset

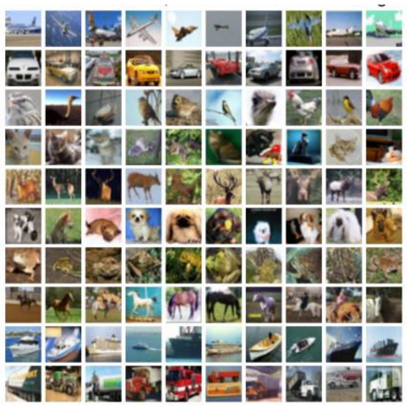


FIGURE 5. The CIFAR-100 dataset

Superclass (Coarse)	**Classes (Fine)**
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

TABLE 1. CIFAR-100 labels

B. Experiments with MLP and CNN

We first built a multiple layer perceptron (MLP) and convolutional neural network (CNN) according to the configuration in Table 2 to test against the MNIST dataset. In this project, all the models were trained on the fyp server (single core Intel i7 CPU, 2GiB memory). Both models were trained with stochastic gradient descent (learning rate = 0.01). The batch size was set to 100 and model was trained for 20 epochs, so there were 12k parameter updates for each model during training. The result is reported in Table 3.

MLP	CNN
Input 28x28 images	Input 28x28 images
400 units	30 filters 3x3
Activation (ReLU)	Activation (ReLU)
Dropout (0.2)	Dropout (0.2)
400 units	30 filters 3x3
Activation (ReLU)	Activation (ReLU)
Dropout (0.2)	Dropout (0.2)
400 units	Maxpooling (2x2)
	120 units
Activation (ReLU)	Activation (ReLU)
Dropout (0.2)	Dropout (0.2)
10-way softmax	10-way softmax

TABLE 2. Experiment configuration for MNIST

	MLP	CNN
No. of parameters	638,810	528,160
Test Accuracy	<u>0.9667</u>	**<u>0.9769</u>**
Test Loss (see Appendix C)	0.1051	0.0783
Train Accuracy	0.9601	0.9719
Train Loss	0.1369	0.0913
Training time	260s	3000s

TABLE 3. Experiment result for MNIST

From Table 3, we can see that convolutional neural network can perform better than a multiple layer perceptron even with fewer number of parameters. The training time for CNN can be improved if we use a GPU and train the model with parallelization.

We also performed a similar experiment on CIFAR-100 dataset. We set different batch size for MLP and CNN: 128 and 32, respectively. After 100 epochs of training, there will be 39k parameter updates for MLP and 156k parameter updates for CNN. As the numbers of parameter updates are different, we also report the performance of CNN after 25 epochs of training (39k parameter updates, similar to that of MLP). The training was done with stochastic gradient descent with learning rate 0.025, decay 10^{-6} and Nesterov momentum 0.9. The configuration is shown in Table 4 and the result is reported in Table 5.

From the result, we can see that as the task becomes more difficult, the difference in parameters and performance for MLP and CNN are greater. One property for CNN is that it captures the relationship of the neighboring area. Thus, in the image classification task, CNN prevails over MLP in detecting edges and contours, making the entire classification more accurate. However, there is a lot of improvement left in the level of accuracy. In the next subsection, some state-of-the-art research in deep learning optimization will be discussed.

MLP	CNN
Input 32x32 images	
3000 units	32 filters 3x3
Activation (ReLU)	Activation (ReLU)
Dropout (0.2)	32 filters 3x3
3000 units	Activation (ReLU)
Activation (ReLU)	Maxpooling (2x2)
Dropout (0.2)	Dropout (0.25)
2000 units	64 filters 3x3
Activation (ReLU)	Activation (ReLU)
Dropout (0.2)	64 filters 3x3
2000 units	Activation (ReLU)
Activation (ReLU)	Maxpooling (2x2)
Dropout (0.2)	Dropout (0.25)
1000 units	512 units
Activation (ReLU)	Activation (ReLU)
Dropout (0.2)	Dropout (0.5)
100-way softmax	

TABLE 4. Experiment configuration for CIFAR-100

	MLP	CNN (25 epochs)	CNN (100 epochs)
No. of parameters	30,327,100	1,297,028	
Test Accuracy	<u>0.2535</u>	**<u>0.4002</u>**	**<u>0.4270</u>**
Test Loss	4.1776	2.3355	2.2208
Train Accuracy	0.5805	0.3235	0.3705
Train Loss	1.5860	2.6801	2.4434
Training time	20000s	6750s	27,000s

TABLE 5. Experiment result for CIFAR-100

C. Optimization Research

The best result in CIFAR-100 uses exponential linear units (ELU) [19] in the activation layers. With non-zero mean activation layers, each layer will produce a bias and propagate to the next layer, which makes the gradient descent less efficient for optimization. ELU is claimed to reduce such bias shift effect. The exponential linear unit is: ($\alpha > 0$)

$$f(x) = \begin{cases} x & , x > 0 \\ \alpha(\exp(x) - 1) & , x \leq 0 \end{cases}, \quad f'(x) = \begin{cases} 1 & , x > 0 \\ f(x) + \alpha & , x \leq 0 \end{cases}$$

Figure 6 from [19] shows the relationship of ELU, leaky ReLU, ReLU and shifted ReLU. The experiment done by Clevert, et al. achieved 75.72% accuracy in CIFAR-100 dataset, and Table 6 shows the model used in their experiment.

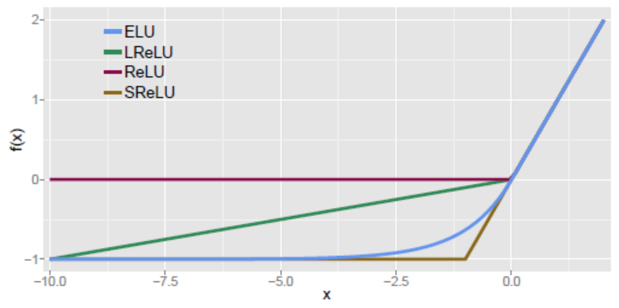


FIGURE 6. Activation functions

Input 32x32 images
384 filters 3x3
ELU (α=1)
Maxpooling 2x2
384 filters 1x1
384 filters 2x2
640 filters 2x2
640 filters 2x2
ELU (α=1)
Dropout(0.1)
Maxpooling 2x2
640 filters 1x1
768 filters 2x2
768 filters 2x2
768 filters 2x2
ELU (α=1)
Dropout(0.2)
Maxpooling 2x2
768 filters 1x1
(continue on the next column)
896 filters 2x2
896 filters 2x2
ELU (α=1)
Dropout(0.3)
Maxpooling 2x2
896 filters 3x3
1024 filters 2x2
ELU (α=1)
Dropout(0.4)
Maxpooling 2x2
1024 filters 1x1
1152 filters 2x2
ELU (α=1)
Dropout(0.5)
Maxpooling 2x2
1152 filters 1x1
ELU (α=1)
100-way Softmax

TABLE 6. Configuration in the original experiment of ELU

Another research in the deep learning optimization focuses on the weight initialization. Initializing the model with Gaussian noise $\mathcal{N}(0, 0.01^2)$ became popular after CNN showed its success in 2012 [20]. Glorot & Benigo [21] proposed a formula to estimate the standard deviation, under the assumption that the relationships between each layer is non-linear. Mishkin & Matas generalized the previous method and named it Layer Sequential Unit Variance (LSUV) [20]. The overall performance on CIFAR-100 is 72.34% accuracy rate.

There are more optimization methods that have been developed recently. Adaptive piecewise linear activation unit [22] is able to learn the activation functions. Fractional max-pooling [23] ameliorates the effect on reducing the size during forward propagation. The pooling is not restricted to the fraction of 1/k where k is an integer, and a pooling fraction between 1/2 and 1 has demonstrated an improved performance. Another method related to pooling is the all convolutional net [24]. The pooling layers are replaced by convolutional layers with an appropriate stride (also called subsampling).

D. Experiments with Advanced Techniques

In this subsection, we experimented with ELU and LSUV discussed previously. As the model used in [19] is too large and is beyond the computation power supported by the server, we used a model with a reduction in size. The model configuration is shown in Table 7.

The training was done with stochastic gradient descent with decay 10^{-6} and Nesterov momentum 0.9. Batch size was set to 100. The learning rate schedule we applied was: 0.005 [1-200 epochs], 0.0025 [201-400 epochs], 0.0005 [401-500 epochs].

Input 32x32 images
80 filters 3x3
80 filters 1x1
ELU (α=1)
Maxpooling 2x2
140 filters 3x3
140 filters 2x2
ELU (α=1)
Dropout(0.1)
Maxpooling 2x2
180 filters 2x2
180 filters 1x1
ELU (α=1)
Dropout(0.2)
Maxpooling 2x2
200 filters 2x2
200 filters 1x1
ELU (α=1)
Dropout(0.3)
Maxpooling 2x2
512 units
ELU (α=1)
Dropout(0.5)
100-way Softmax

TABLE 7. Configuration in our experiment with ELU and LSUV

The training for each epoch took 740 seconds. After 500 epochs of training, the test accuracy reached 70.15% and the training accuracy was 79.15%.

We then used the same model and replaced the ELUs with different activation units. The settings for training were the same, except the schedule for learning rate being: 0.01 [1-100 epochs], 0.001 [101-200 epochs], 0.0001 [201-300 epochs]. The results are compared in Table 8.

CNN	ELU	ReLU	LeakyReLU
Test Accuracy	**0.6837**	0.6523	0.6773
Test Loss	1.1104	1.2186	1.1202
Train Accuracy	0.7076	0.6649	0.6953
Train Loss	0.9757	1.1291	1.1301

TABLE 8. Comparing different activation functions

Rectified linear unit (ReLU) has an activation of 0 when the input is negative; therefore, the mean activation is always non-negative. Both ELU and LeakyReLU have negative activations. Theoretically, ELU performs better than LeakyReLU because the mean of activations is closer to 0. The experiment results are consistent with [19].

VI. WORKING WITH PACKET CAPTURE DATA

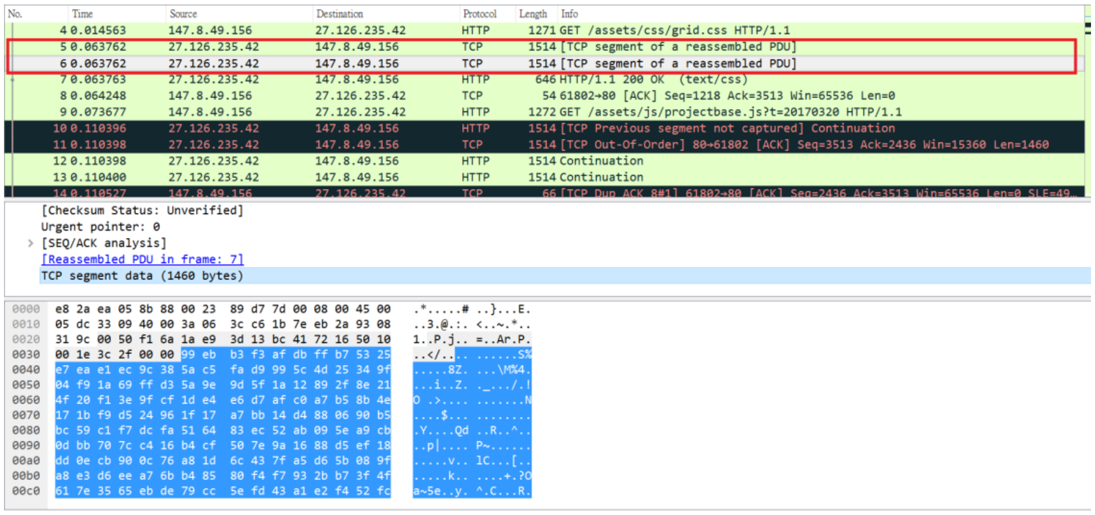
After studying the optimization of deep learning, we now move on to experimenting with network packet capture data. We started with classifying packets according to their application protocols to examine whether the deep learning model has the capability to extract the underlying information from the packet payload. After the capability was confirmed, we moved on to the next phase, anomaly detection. In this section, we begin each phase with the detailed information about dataset we used, and then the experiment results are reported and discussed.

A. DATASET FOR CLASSIFICATION

In order to provide data for the deep learning model, a host was set up to capture network packets that passed through the HKUCS network. The packets were captured in PCAP (packet capture) format, and because of the restrictions of the capturing system, the file size of each capture file is 2 GiB. We sampled 2-hour traffic for the experiment, with a total of 60 PCAP files and 120 GiB of data.

After the data were collected, we chose 10 application protocols that were frequently present in the capture files. The protocols chosen would be the targets of the classification task. We then retrieved packets that belonged to one of the chosen protocols by the port numbers from the raw capture files, and saved the packets into different files according to protocols. As the protocols chosen are widely used and their port numbers are either well-known or registered, we assume that no other packet used the same ports.

The next step was to join the payloads of the packets if they belonged to the same session. We joined the payloads of the packets belonging to the same session because each piece of application data may be encapsulated and transmitted in multiple packets. Consider the example where we visited the website http://www.hku.hk/ (27.126.235.42). The packets captured were shown in Figure 7. Packet 4 is an HTTP GET request and the server replied with HTTP OK and some application data shown in packets 5-7. However, if we look at the payload in packet 6 (see Figure 8), it contains only a part of binary data and it is not realistic to classify this packet solely based on a part of binary data.

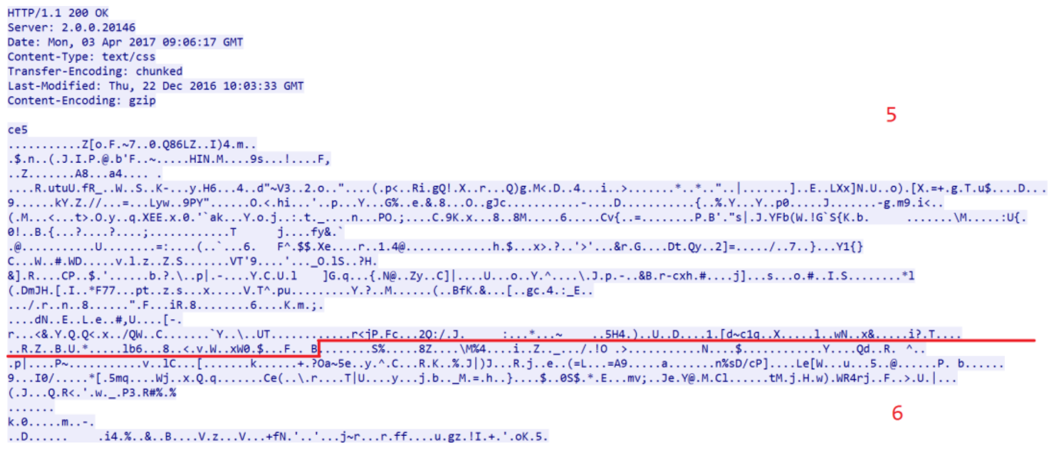


No.	Time	Source	Destination	Protocol	Length	Info
4	0.014563	147.8.49.156	27.126.235.42	HTTP	1271	GET /assets/css/grid.css HTTP/1.1
5	0.063762	27.126.235.42	147.8.49.156	TCP	1514	[TCP segment of a reassembled PDU]
6	0.063762	27.126.235.42	147.8.49.156	TCP	1514	[TCP segment of a reassembled PDU]
7	0.063763	27.126.235.42	147.8.49.156	HTTP	646	HTTP/1.1 200 OK (text/css)
8	0.064248	147.8.49.156	27.126.235.42	TCP	54	61802→80 [ACK] Seq=1218 Ack=3513 Win=65536 Len=0
9	0.073677	147.8.49.156	27.126.235.42	HTTP	1272	GET /assets/js/projectbase.js?t=20170320 HTTP/1.1
10	0.110396	27.126.235.42	147.8.49.156	HTTP	1514	[TCP Previous segment not captured] Continuation
11	0.110398	27.126.235.42	147.8.49.156	TCP	1514	[TCP Out-Of-Order] 80→61802 [ACK] Seq=3513 Ack=2436 Win=15360 Len=1460
12	0.110398	27.126.235.42	147.8.49.156	HTTP	1514	Continuation
13	0.110400	27.126.235.42	147.8.49.156	HTTP	1514	Continuation
14	0.110527	147.8.49.156	27.126.235.42	TCP	66	[TCP Dup ACK 8#1] 61802→80 [ACK] Seq=2436 Ack=3513 Win=65536 Len=0 SLE=49…

```
    [Checksum Status: Unverified]
    Urgent pointer: 0
  > [SEQ/ACK analysis]
    [Reassembled PDU in frame: 7]
    TCP segment data (1460 bytes)
```

```
0000  e8 2a ea 05 8b 88 00 23  89 d7 7d 00 08 00 45 00   .*.....# ..}...E.
0010  05 dc 33 09 40 00 3a 06  3c c6 1b 7e eb 2a 93 08   ..3.@.:. <..~.*..
0020  31 9c 00 50 f1 6a 1a e9  3d 13 bc 41 72 16 50 10   1..P.j.. =..Ar.P.
0030  00 1e 3c 2f 00 00 99 eb  b3 f3 af db ff b7 53 25   ..</.... ......S%
0040  e7 ea e1 ec 9c 38 5a c5  fa d9 99 5c 4d 25 34 9f   .....8Z. ...\M%4.
0050  04 f9 1a 69 ff d3 5a 9e  9d 5f 1a 12 89 2f 8e 21   ...i..Z. ._.../.!
0060  4f 20 f1 3e 9f cf 1d e4  e6 d7 af c0 a7 b5 8b 4e   O .>.... .......N
0070  17 1b f9 d5 24 96 1f 17  a7 bb 14 d4 88 06 90 b5   ....$... ........
0080  bc 59 c1 f7 dc fa 51 64  83 ec 52 ab 09 5e a9 cb   .Y....Qd ..R..^..
0090  0d bb 70 7c c4 16 b4 cf  50 7e 9a 16 88 d5 ef 18   ..p|.... P~......
00a0  dd 0e cb 90 0c 76 a8 1d  6c 43 7f a5 d6 5b 08 9f   .....v.. lC...[..
00b0  a8 e3 d6 ee a7 6b b4 85  80 f4 f7 93 2b b7 3f 4f   .....k.. ....+.?O
00c0  61 7e 35 65 eb de 79 cc  5e fd 43 a1 e2 f4 52 fc   a~5e..y. ^.C...R.
```

FIGURE 7. Packet capture example



```
HTTP/1.1 200 OK
Server: 2.0.0.20146
Date: Mon, 03 Apr 2017 09:06:17 GMT
Content-Type: text/css
Transfer-Encoding: chunked
Last-Modified: Thu, 22 Dec 2016 10:03:33 GMT
Content-Encoding: gzip

ce5
...........Z[o.F.~7..0.Q86LZ..I)4.m..
.$.n..(.J.I.P.@.b'F..~.....HIN.M....9s...!....F,
..Z.......A8...a4.... .
....R.utuU.fR_..W..S..K-...y.H6...4..d"~V3..2.o.."....(.p<..Ri.gQ!.X..r...Q)g.M<.D..4...i..>.......*..*.."..|.......]..E..LXx]N.U..o).[X.=+.g.T.u$....D...
9......kY.Z.//...=...Lyw..9PY"......O.<.hi...'..p...Y...G%..e.&.8...O..gJc..........-....D...........{..%.Y...Y..p0.....J.......-g.m9.i<..
(.M...<...t>.O.y..q.XEE.x.0.'`ak...Y.o.j..:.t._....n...PO.;....C.9K.x...8..8M.....6.....Cv{..=........P.B'."s|.J.YFb(W.!G`S{K.b.      .......\M.....:U{.
0!..B.{...?....?....;............T      j....fy&.`
.@..........U........=:....(..`...6.   F^.$$.Xe....r..1.4@............h.$...x>.?..'>'...&r.G....Dt.Qy..2]=...../..7..}...Y1{}
C...W..#.WD.....v.l.z..Z.S.......VT'9....'..._O.lS..?H.
&].R....CP..$.'......b.?.\..p|.-....Y.C.U.l    ]G.q...{.N@..Zy..C]|....U...o..Y.^....\.J.p.-..&B.r-cxh.#....j]...s...o.#..I.S........*l
(.DmJH.[.I..*F77...pt..z.s...x.....V.T^.pu.........Y.?..M......(..BfK.&...[..gc.4.:_E..
.../.r..n..8......".F...iR.8........6....K.m.;.
....dN..E..L.e..#,U....[-.
r...<&.Y.Q.Q<.x../QW..C.......`Y..\..UT............r<jP.Fc...2Q:/.J.      :...*...~     ..5H4.)..U..D....1.[d~c1q..X.....l..wN..x&.....i?.T....
..R.Z..B.U.*.....lb6...8..<.v.W..xW0.$...F.. B........S%.....8Z....\M%4....i..Z.._.../.!O .>...........N....$............Y....Qd..R.  ^..
.p|....P~...........v..lC...[.......k......+.?Oa~5e..y.^.C...R.K..%.J|)J...R.j..e..(=L...=A9.....a.......n%sD/cP]....Le[W...u...5..@......P. b......
9...I0/.....*[.5mq....Wj..x.Q.q.......Ce(..\.r....T|U....y...j.b.._M.=.h..}....$..0S$.*.E...mv;..Je.Y@.M.Cl......tM.j.H.w).WR4rj..F..>.U.|...
(.J...Q.R<.'.w._.P3.R#%.%
.......
k.0.....m..-.
..D......    .i4.%..&..B....V.z...V...+fN.'..'...j~r...r.ff....u.gz.!I.+.'.oK.5.
```

FIGURE 8. Sample payload of packets

As mentioned in the methodology (Section IV), we used the first 500 bytes of the payload because it was shown that the information is sufficient. The samples with payload size lower than 500 bytes were padded with ASCII 0 (NULL), and the samples with 0 payload were discarded. To prevent the data from biasing towards a certain protocol, we set a limit to each protocol. The maximum number of streams to be used for each protocol was set to be 600. The resulting stream data will be normalized such that the value of each byte ranged from [0, 255] to [0, 1]. Normalization would be helpful when determining the learning rate, which typically has a magnitude of 10^{-2}. Finally, the dataset was split 5:1 for training and testing. The number of streams in each protocol is reported in Table 9.

Protocol	Train	Test
DNS	220	44
FTP	311	62
HTTP	500	100
IMAP	500	100
MYSQL	420	84
NFS	183	36
POP3	189	37
SMTP	500	100
SSH	500	100
TLS	500	100

TABLE 9. Number of samples in each protocol

B. Application Protocol Classification

We first created a multiple layer perceptron using the configuration in Table 10 to for the classification test. Similar to Section V, all the models were trained on the fyp server (single core Intel i7 CPU, 2GiB memory). The learning rate was set to 0.01 for the first 100 epochs, and 0.005 for epochs 101-200. The batch size was 32, and different activations were used, including ReLU, ELU (α=1) and LeakyReLU (α=0.3). After training for 200 epochs, the results are reported in Table 11.

MLP
500-byte input
500 units
Activation
Dropout (0.1)
500 units
Activation
Dropout (0.2)
500 units
Activation
Dropout (0.2)
500 units
Activation
Dropout (0.3)
500 units
Activation
Dropout (0.3)
10-way softmax

TABLE 10. MLP configuration for classification

MLP	**ReLU**	**ELU**	**LeakyReLU**
Test Accuracy	<u>0.8663</u>	<u>0.8073</u>	<u>0.8427</u>
Test Loss (see Appendix C)	0.9537	1.5132	1.0826
Train Accuracy	0.9851	0.9774	0.9833
Train Loss	0.0346	0.0066	0.0392

TABLE 11. Results of classification with MLP

MLP	ReLU		ELU		LeakyReLU	
	Precision	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
DNS	1.0000	0.9565	0.9773	0.9348	1.0000	0.9565
FTP	0.9516	0.8676	0.7639	0.8088	0.9032	0.7467
HTTP	0.8500	0.7727	0.7100	0.7802	0.8300	0.7545
IMAP	0.8500	0.8673	0.7400	0.7708	0.8100	0.7788
MYSQL	0.9405	0.9518	0.9405	0.9518	0.9405	0.8977
NFS	0.8889	1.0000	0.8889	1.0000	0.9167	0.9706
POP3	0.8378	0.9118	0.8919	0.7333	0.8649	0.8000
SMTP	0.8300	0.8924	0.7700	0.7778	0.8000	0.8989
SSH	0.9000	0.8654	0.8700	0.7699	0.8700	0.8878
TLS	0.7300	0.7684	0.6500	0.7222	0.6800	0.8608

TABLE 12. Precision and recall of classification with MLP

Apart from reporting the accuracy of the model as a whole, we also used the **precision** and **recall** to compare the results (see Table 12):

Consider a protocol X and a data sample S. Define true positive (TP), true negative (TN), false positive (FP), false negative (FN) as the following table (Table 13):

Prediction \ Ground Truth	S belongs to protocol X	S does not belong to protocol X
S is classified as protocol X	True Positive (TP)	False Positive (FP)
S is classified as another protocol	False Negative (FN)	True Negative (TN)

TABLE 13. True positive, false positive, false negative, and true negative

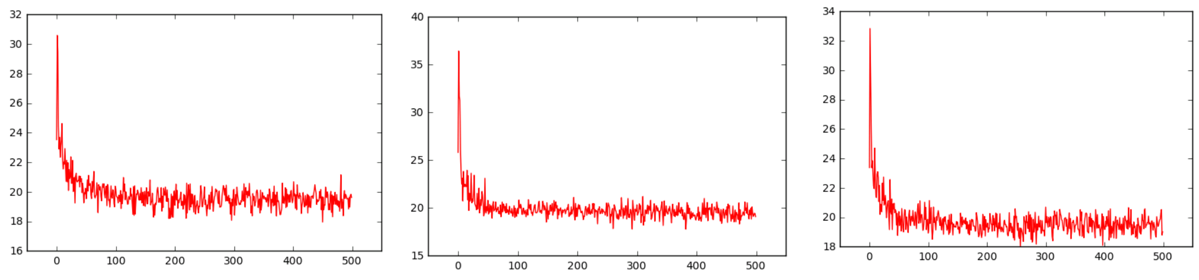
- Among all the samples classified as a particular protocol X, **precision** measures how many samples are truly X.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- Among all the samples that belongs to a particular protocol X, **recall** measures how many samples that are classified as X.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

In addition to the performance of the deep learning neural network, Figure 9 plots the sum of absolute values of the weights that is associated with each input byte and first hidden layer. It can be found out that the first few bytes contribute the most to this classification task, and the result is consistent with [17]. Also, the model that uses ELU stresses relatively more importance on the first few bytes compared with different activations used.



(a) ReLU (b) ELU (c) LeakyReLU

FIGURE 9. Weight associated with each byte in classification

We also constructed a 1D convolutional neural network to see its capability to capture the information of a packet payload. Typically, an image classification task (as in Section V) may use a 2D convolutional neural network to reduce the amount of parameters involved while maintaining the same performance. The convolution operation can be done in one dimension only. Figure 10 shows an example of 1D convolutional neural network. Output[1] is generated by applying convolution to Input[1] and Input[2] with kernel. In our experiment, Input[x] is the normalized ASCII value (between 0 and 1, inclusive) of byte x in the payload. Notice that in other implementation of 1D convolutional neural network, Input[x] can be a tensor of any shape. Therefore, 1D convolutional neural network can be applied to data with a higher dimension, but the convolution operation is done on one of the dimensions only.

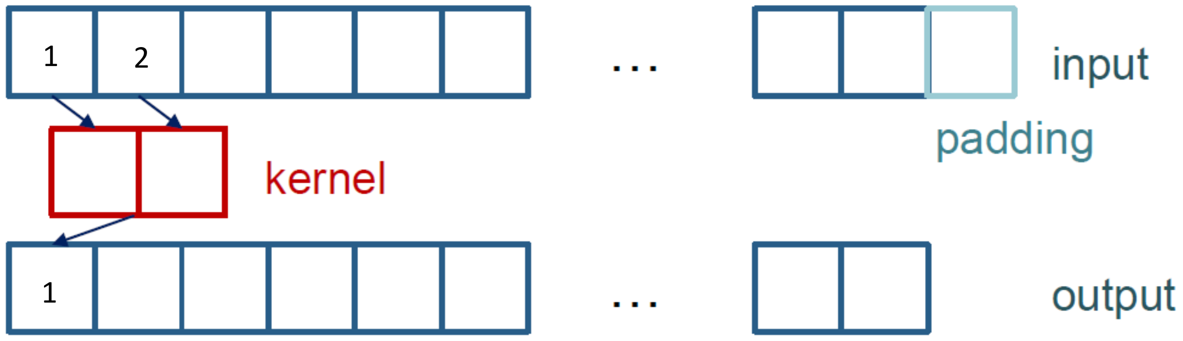


FIGURE 10. 1D convolutional neural network

We used two different activations: ReLU and ELU (α=1). The configuration is shown in Table 14 and the training was done with stochastic gradient descent with learning rate = 0.1. After 50 epochs of training, the result is reported in Table 15.

CNN
500-byte input
200 filters 2x1
Activation (ReLU/ELU)
1D Maxpooling (pool length = 2)
Dropout (0.1)
300 filters 2x1
Activation (ReLU/ELU)
1D Maxpooling (pool length = 2)
Dropout (0.2)
400 filters 2x1
Activation (ReLU/ELU)
1D Maxpooling (pool length = 2)
Dropout (0.2)
10-way softmax

TABLE 14. CNN configuration for classification

CNN	**ReLU**	**ELU**
Test Accuracy	<u>0.8886</u>	<u>0.8165</u>
Test Loss	0.6538	1.3266
Train Accuracy	0.9864	0.9822
Train Loss	0.0341	0.0502

TABLE 15. Results of classification with CNN

We can see that CNN had a lightly better performance when compared with MLP; however, the training time for CNN in our experiment was 100 times more than that of MLP. Nevertheless, the results for both MLP and CNN demonstrates the ability to capture the underlying information of the payload and determine the protocol of each data sample accordingly. In the following subsections, we will apply the same models to detect anomalies.

There is one issue that remains unsolved. During the packet capturing, each PCAP file can store 2GiB of data only. If a file reaches its limit, the packets will then be captured to another file. On average, each capture file contains the packets captured within 2 minutes. In our preprocessing step, when we were joining the payload, the data were joined if the packets belong to the same TCP stream and packets, and the packets exist in the same capture file. As a consequence, if a TCP stream consists of packets stored in different files, there will be more than one samples created, and some of them may contain binary data (same as packets 6 in Figure 7 and Figure 8), or data that is irrelevant to distinguish the protocol used in the packet. Such data samples are noise and exist in both training and testing data.

There are some possible ways to remove noisy data. One way is to examine each of the merged payload and see if it is complete, and discard the samples that have only a part of the entire payload. This approach requires lots of human effort. Another way is to merge the raw data file before joining the payload. The merging process can be done by tools such as tcpdump or tshark (command line tool of wireshark). However, after the raw data files are merged, there will be one large file. Extracting payload according to TCP streams from such a large file will be extremely costly. A more viable approach is to capture the packet only if it is of our interest, but it requires our direct access to control the host that captured packets. In our project, the task of packet classification is to validate the capability of the deep learning neural network when it works on payload data. As it can be seen from Table 15 that the model worked with reasonable accuracy, we moved on to detecting network anomalies.

C. Dataset for Anomaly Detection

To train and evaluate the anomaly detection model, we prepared another dataset that is suitable for this purpose. The CTU-13 dataset [18], prepared by Czech Technical University, contains packet capture files generated by different types of botnets. There are 13 different simulated scenarios, and we chose the scenarios that are not duplicated and contains sufficient amount of data. We joined the payload of the packets using the same approach described in Section VI.A. The selected scenarios, together with their detailed information, are presented in Table 16. The original dataset also contains normal and background traffic, but these packets are not made public due to privacy issues. Therefore, we combined the packets processed in VI.A to form a set of normal data. We first created a dataset (D01) that contains normal flow and anomalous flow with only two different bots. The information of D01 is shown in Table 17.

ID	Scenario (in CTU-13)	Bot	Number of streams
#1	1	Neris	1718
#2	3	Rbot	283
#3	5	Virut	409
#4	6	Menti	215
#5	8	Murlo	1987
#6	12	NSIS.ay	343

TABLE 16. CTU-13 Dataset

Type	Train	Test	Description
Normal	3822	764	All the streams in VI.A
Anomalous	1668	333	#1 and #2 in Table 16

TABLE 17. Anomaly detection dataset D01

We created two more datasets (D02, D03) that contain more scenarios from Table 16. In D03, the number of packets from each scenario is adjusted so that the training will not bias towards any particular scenario. The information of two datasets are shown in Table 18 and Table 19.

Type	Train	Test	Description
Normal	3822	764	All the streams in VI.A
Anomalous	2188	437	#1, #2, #3 and #4 in Table 16

TABLE 18. Anomaly detection dataset D02

Type	Train	Test	Description
Normal	3822	764	All the streams in VI.A
Anomalous	2125	425	#1: 650 samples #2: 283 samples #3: 409 samples #4: 215 samples #5: 650 samples #6: 343 samples

TABLE 19. Anomaly detection dataset D03

D. Anomaly Detection

We set up a multiple layer perceptron (MLP) and 1D convolutional neural network (CNN) to test on the dataset D01. The MLP and CNN had the same configurations the one we used when we were classifying packets, except that in the output layer, the 10-way softmax was changed to 2-way softmax since the output label would be either “normal” or “anomalous.” The modified changes were shown in Table 20 and Table 21.

MLP
500-byte input
500 units
Activation
Dropout (0.1)
500 units
Activation
Dropout (0.2)
500 units
Activation
Dropout (0.2)
500 units
Activation
Dropout (0.3)
500 units
Activation
Dropout (0.3)
2-way softmax

TABLE 20. MLP configuration for classification

CNN
500-byte input
200 filters 2x1
Activation (ReLU/ELU)
1D Maxpooling (pool length = 2)
Dropout (0.1)
300 filters 2x1
Activation (ReLU/ELU)
1D Maxpooling (pool length = 2)
Dropout (0.2)
400 filters 2x1
Activation (ReLU/ELU)
1D Maxpooling (pool length = 2)
Dropout (0.2)
2-way softmax

TABLE 21. CNN configuration for classification

We measured the overall test accuracy (total number of samples classified correctly / total number of samples misclassified), and the precision and recall for the anomaly group. The definition of TP, TN, FP, FN under this situation is shown in Table 22:

Prediction \ Ground Truth	S is anomalous	S is normal
S is classified as anomaly	True Positive (TP)	False Positive (FP)
S is classified normal	False Negative (FN)	True Negative (TN)

TABLE 22. True positive, false positive, false negative, and true negative when used in anomaly detection

The performance of MLP is reported in Table 23 and Table 24, and the performance of CNN is reported in Table 25 and Table 26.

MLP	ReLU	ELU	LeakyReLU
Test Accuracy	0.9727	0.9462	0.9672
Test Loss	0.1185	0.3053	0.1892
Train Accuracy	0.9996	0.9960	0.9991
Train Loss	0.0010	0.0131	0.0017

TABLE 23. Results of detection D01 with MLP

MLP	ReLU		ELU		LeakyReLU	
	Precision	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
Anomaly	0.9489	0.9604	0.9770	0.8702	0.9770	0.9279

TABLE 24. Precision and recall of detection D01 with MLP

CNN	ReLU	ELU
Test Accuracy	0.9681	0.9599
Test Loss	0.1068	0.2025
Train Accuracy	0.9969	0.9918
Train Loss	0.0119	0.0223

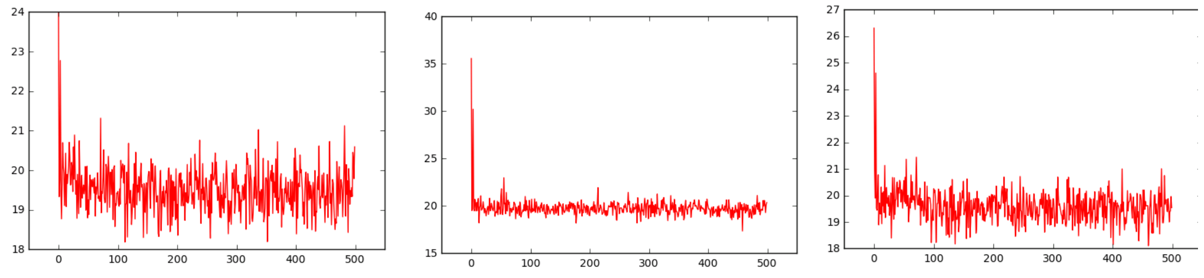
TABLE 25. Results of detection D01 with CNN

CNN	ReLU		ELU	
	Precision	**Recall**	**Precision**	**Recall**
Anomaly	0.9730	0.9257	0.9790	0.8981

TABLE 26. Precision and recall of detection D01 with CNN

From Table 23 and Table 25, we can see that if we use the same configurations as the ones in protocol classification, the performance of MLP is better than CNN. However, in the last 10 training epochs of CNN and MLP (using ReLU as activations), the test accuracy fell in [0.9681, 0.9781] and [0.9699, 0.9754], respectively, and the accuracy was neither strictly increasing nor strictly decreasing. Note that this situation does not apply to classification task, where the test accuracy of CNN [0.8585, 0.8689] is better than MLP [0.8807, 0.8978] in the last 10 epochs. We conclude that the two models have similar performance.

Similarly, we plot the sum of absolute values of the weights that is associated with each input byte and first hidden layer. The first few bytes still contribute the most to this classification task, and the model with ELU stresses relatively more importance on the first few bytes. In the model where ReLU is used, the importance of the first few bytes is not highly valued (notice the scale of y-axis). The reason behind can be further investigated.



(a) ReLU (b) ELU (c) LeakyReLU

FIGURE 11. Weight associated with each byte in detection

As Table 23 and Table 25 suggests that MLP and CNN have the same level of performance, we adopt MLP to train models for dataset D02 and D03 because the training is more efficient with regard to time. The results are reported in Table 27 to Table 30.

MLP	ReLU	ELU	LeakyReLU
Test Accuracy	0.9742	0.9467	0.9600
Test Loss	0.1251	0.3240	0.1925
Train Accuracy	0.9995	0.9963	0.9992
Train Loss	0.0009	0.0107	0.0018

TABLE 27. Results of detection D02 with MLP

MLP	ReLU		ELU		LeakyReLU	
	Precision	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
Anomaly	0.9748	0.9551	0.9633	0.8976	0.9565	0.9351

TABLE 28. Precision and recall of detection D02 with MLP

MLP	ReLU	ELU	LeakyReLU
Test Accuracy	0.9411	0.9697	0.9605
Test Loss	0.3773	0.1614	0.2369
Train Accuracy	0.9973	0.9995	0.9993
Train Loss	0.0103	0.0008	0.0021

TABLE 29. Results of detection D03 with MLP

MLP	ReLU		ELU		LeakyReLU	
	Precision	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
Anomaly	0.9489	0.9604	0.9770	0.8702	0.9770	0.9279

TABLE 30. Precision and recall for detection D03 with MLP

After adding the traffic generated by more different types of botnets, we can see from the results that the level of accuracy using MLP remains the same. The results suggest that deep learning models can well distinguish botnet traffic from normal traffic and detect anomalies.

VII. CONCLUSION

As people nowadays heavily rely on the Internet, it is important to develop an intrusion detection system that helps prevent existing as well as zero-day network attacks. With the increase of computer power, it is feasible to construct a deep learning model to detect anomalies based on payload data of packets. In our project, we have constructed a deep learning neural network that can identify the application layer protocols used in TCP streams based on the payload. The same model was used to detect network anomalies generated by botnets.

We have shown the performance of various activation functions when applied to network data. Some issues arising from this project can be further pursued. Firstly, it is possible to include more information other than payload as the input of deep learning neural network. Secondly, the model can be adjusted to tradeoff between positives against false negatives. Thirdly, more types of anomalies can be included. Finally, some activation functions can be devised to work on network packets.

APPENDICES

A. CODE FOR SORTING PACKETS

Using tcpdump, we can extract the packet that matches the port number from PCAP files.

```
tcpdump -n -r <capture_file_name> -w <output_file_name> 'tcp port <port_number>'
```

The following code reads a PCAP file (consists of many packets and streams) and saves the packets to the same file if the packets belong to the same TCP stream:

Parameter: Command line argument 1 – the PCAP file to be separated into TCP streams

```
#!/bin/bash
maxlen=$(tshark -r "${1}" -Y usb -z conv,tcp| wc -l) #total number of streams
let "maxlen=${maxlen}-6" #-6 for headers
mkdir "${1}_stream"
for stream in $(seq 1 ${maxlen})
do
    tshark -r ${1} -w ${1}_stream/stream-${stream}.pcap -Y "tcp.stream==$stream
" -F pcap
done
```

B. Code for Preprocessing Streams

The following code label the joins the payload of the stream, assign labels (user should input the parameters), and checks the size of each stream.

```python
def preprocess_stream(folder_name, ofname, label, minlen=1, limit=-1, size=500, normalize=False):
    onlyfiles = [f for f in listdir(folder_name) if isfile(join(folder_name, f))]
    counter = 0
    with open(ofname+"_meta", "a") as output_meta:
        for in_file in onlyfiles:
            with open(ofname, "a") as output_file:
                entirepayload = list()
                for ts, pkt in dpkt.pcap.Reader(open(folder_name+"/"+in_file,'r')):

                    eth = dpkt.ethernet.Ethernet(pkt)
                    ip = eth.data
                    tcp = ip.data
                    app = tcp.data

                    payload_str = str(app)
                    if normalize:
                        payload_ascii = [ord(x)/255.0 for x in payload_str]
                    else:
                        payload_ascii = [ord(x) for x in payload_str]
                    entirepayload.extend(payload_ascii)
                    if(len(entirepayload) > size):
                        break
```

(continue on the next page)

```
            if(len(entirepayload) < minlen):
                continue
            elif(len(entirepayload) < size):
                entirepayload += [0] * (size - len(entirepayload))
            entirepayload = entirepayload[:size]
            output_file.write(str(entirepayload)[1:-1])
            output_file.write(" ; ")
            output_file.write(str(label))
            output_file.write("\n")
        counter+=1
        output_meta.write(in_file)
        output_meta.write("\n")
        if(counter == limit):
            break
print("streams output: " + str(counter))
```

C. Train and Test Loss – Categorical Cross Entropy

In the experiment results, categorical (multinomial) cross entropy was used as the loss function. Suppose there are m classes and n samples, let p_{ij} denote the probability that sample i belongs to class j (note that $\sum_j p_{ij} = 1$ in our experiment since softmax layer is used to predict output), and $y_{\mathrm{ij}} = \begin{cases} 1 & \text{sample } i \text{ belongs to class } j \\ 0 & \text{otherwise} \end{cases}$.

Categorical cross entropy is calculated by:

$$\mathcal{L} = -\frac{1}{n}\sum_{i=i}^{n}\sum_{j=1}^{m}\left(y_{ij}\log\left(p_{ij}\right)\right)$$

Consider the following trinary classification example (Prob[x] is the normalized probability that the sample belongs to class x using softmax function):

Sample no.	Label	Prediction	Prob[1]	Prob[2]	Prob[3]
1	**1**	<u>1</u>	<u>**0.95**</u>	0.01	0.04
2	**1**	<u>2</u>	**0.05**	<u>0.55</u>	0.40

Loss: $\mathcal{L} = -\frac{1}{2}(\log_2 0.95 + \log_2 0.05) = 2.2$

Consider another example where each sample can belong to more than one class. Assume the prediction threshold is 0.5.

Sample no.	Label	Prediction	Prob[1]	Prob[2]	Prob[3]
1	**1**	<u>1</u>	<u>**0.9**</u>	0.2	0.4
2	**1, 2**	<u>2, 3</u>	**0.1**	<u>**0.7**</u>	<u>0.55</u>

Loss: $\mathcal{L} = -\frac{1}{2}\left(\log_2 0.9 + (\log_2 0.1 + \log_2 0.7)\right) = 1.99$

REFERENCES

1. Symantec. Internet Security Threat Report 2016 2016. Available from: https://www.symantec.com/security-center/threat-report.

2. Cisco. Snort 2016. Available from: https://www.snort.org/.

3. Open Information Security Foundation. Suricata 2016. Available from: https://suricata-ids.org/.

4. LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE. 1998;86:2278-324.

5. Krizhevsky A, Hinton G. Learning multiple layers of features from tiny images. 2009.

6. Patcha A, Park J-M. An overview of anomaly detection techniques: Existing solutions and latest technological trends. Computer networks. 2007;51:3448-70.

7. Agrawal S, Agrawal J. Survey on Anomaly Detection using Data Mining Techniques. Procedia Computer Science. 2015;60:708-13.

8. Goodfellow I, Bengio Y, Courville A. Deep Learning. 2016.

9. Münz G, Li S, Carle G. Traffic anomaly detection using k-means clustering. GI/ITG Workshop MMBnet; 2007.

10. Chitrakar R, Huang C. Anomaly based Intrusion Detection using Hybrid Learning Approach of combining k-Medoids Clustering and Naïve Bayes Classification. 2012 8th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM); 2012.

11. Muda Z, Yassin W, Sulaiman M, Udzir NI. A K-Means and Naive Bayes learning approach for better intrusion detection. Information Technology Journal. 2011;10:648-55.

12. Varuna S, Natesan P. An integration of k-means clustering and naïve bayes classifier for Intrusion Detection. 2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN); 2015.

13. Farid DM, Harbi N, Rahman MZ. Combining naive bayes and decision tree for adaptive intrusion detection. arXiv preprint arXiv:10054496. 2010.

14. Wang K, Stolfo SJ. Anomalous payload-based network intrusion detection. International Workshop on Recent Advances in Intrusion Detection; 2004.

15. Niyaz Q, Sun W, Javaid AY, Alam M. A Deep Learning Approach for Network Intrusion Detection System. 9th EAI International Conference on Bio-inspired Information and Communications Technologies; 2015.

16. Hettich S, Bay SD. The UCI KDD Archive. Irvine, CA: University of California, Department of Information and Computer Science1999.

17. Wang Z. The Applications of Deep Learning on Traffic Identification. Black Hat; 2015.

18. Garcia S, Grill M, Stiborek J, Zunino A. An empirical comparison of botnet detection methods. computers & security. 2014;45:100-23.

19. Clevert D-A, Unterthiner T, Hochreiter S. Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:151107289. 2015.

20. Mishkin D, Matas J. All you need is a good init. arXiv preprint arXiv:151106422. 2015.

21. Glorot X, Bengio Y, Understanding the difficulty of training deep feedforward neural networks. 2010: Publisher.

22. Agostinelli F, Hoffman M, Sadowski P, Baldi P. Learning activation functions to improve deep neural networks. arXiv preprint arXiv:14126830. 2014.

23. Graham B. Fractional max-pooling. arXiv preprint arXiv:14126071. 2014.

24. Springenberg JT, Dosovitskiy A, Brox T, Riedmiller M. Striving for simplicity: The all convolutional net. arXiv preprint arXiv:14126806. 2014.