2015-09

Neural detection of malicious network activities using a new direct parsing and feature extraction technique

Low, Cheng Hong

Monterey, California: Naval Postgraduate School

http://hdl.handle.net/10945/47298

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

NEURAL DETECTION OF MALICIOUS NETWORK ACTIVITIES USING A NEW DIRECT PARSING AND FEATURE EXTRACTION TECHNIQUE

by

Cheng Hong Low

September 2015

Thesis Advisor:	Phillip Pace
Co-Advisor	Monique P. Fargues

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2015	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE NEURAL DETECTION OF MALICIOUS NETWORK ACTIVITIES USING A NEW DIRECT PARSING AND FEATURE EXTRACTION TECHNIQUE	**5. FUNDING NUMBERS**
6. AUTHOR(S) Low, Cheng Hong	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Center for Joint Services Electronic Warfare Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

The aim of this thesis is to develop an intrusion detection system (IDS) software, which learns to detect and classify network attacks and intrusions through prior training data. With the added criteria of operating in real-time applications, ways of improving the efficiency of the IDS without sacrificing the probability of correct classification (PCC) are also considered. Knowledge Data and Discovery Cup 99 data is used to evaluate the IDS architecture. Two neural network (NN) architectures were designed and compared through simulation; the first architecture uses a single NN, while the second uses the merged output of three NNs in parallel. Results show that a three-parallel NN implementation has similar classification performance and a shorter training time than with a single NN implementation. PCC is on the order of 93% for denial-of-service attacks and 96% for normal traffic. The classification results for the R2L and U2R attacks are poor due to the lack of available training data.

14. SUBJECT TERMS intrusion detection systems, neural networks	**15. NUMBER OF PAGES** 75
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

NEURAL DETECTION OF MALICIOUS NETWORK ACTIVITIES USING A NEW DIRECT PARSING AND FEATURE EXTRACTION TECHNIQUE

Cheng Hong Low
Civlian, ST Aerospace, Singapore
M.Sc., National University of Singapore, 2012

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2015

Approved by: Phillip Pace
Thesis Advisor

Monique P. Fargues
Co-Advisor

Clark Robertson
Chair, Department of Electrical And Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The aim of this thesis is to develop an intrusion detection system (IDS) software, which learns to detect and classify network attacks and intrusions through prior training data. With the added criteria of operating in real-time applications, ways of improving the efficiency of the IDS without sacrificing the probability of correct classification (PCC) are also considered. Knowledge Data and Discovery Cup 99 data is used to evaluate the IDS architecture. Two neural network (NN) architectures were designed and compared through simulation; the first architecture uses a single NN, while the second uses the merged output of three NNs in parallel. Results show that a three-parallel NN implementation has similar classification performance and a shorter training time than with a single NN implementation. PCC is on the order of 93% for denial-of-service attacks and 96% for normal traffic. The classification results for the R2L and U2R attacks are poor due to the lack of available training data.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

IDS	Intrusion Detection System
KDD	Knowledge Discovery and Data Mining
NN	Neural Network
PCC	Probability of Correct Classification
RAM	Random Access Memory
SVM	Support Vector Machines
TCP/IP	Transmission Control Protocol/Internet Protocol

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

My sincere thanks go to Professor Phillip Pace and Professor Monique Fargues, who have provided me with ample learning opportunities through their patience, motivation, enthusiasm, and immense knowledge. Their valuable guidance has been instrumental in making this thesis possible. Special thanks to Dr. Martin Kruger, Office of Naval Research, ONR Code 30, Arlington, VA.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Background into the development of an intrusion detection system software prototype and the principal contributions of this thesis are discussed in this thesis

A. BACKGROUND

The growing threat of malicious network activities and intrusion attempts makes intrusion detection systems (IDS) a necessity. Network traffic includes attacker packets, normal packets, and victim packets; thus, IDS must work in a dynamic environment and requires continuous tuning to react against the evolving new attacks that exploit newly discovered security weaknesses. Examples of network attacks include:

- MAC layer denial-of-service (DOS) attacks
- ChopChop attacks
- Passive eavesdropping
- Deauthentication attacks
- Fragmentation attacks
- Duration attacks
- Probing (probe)
- Remote-to-local (R2L)
- User-to-root (U2R)

Typical approaches to intrusion detection include anomaly detection, misuse detection, and combined anomaly/misuse detection [1]. Anomaly detection works by identifying unusual (often malicious) activities which differ from typical user patterns. Misuse detection compares and contrasts a user’s activities with known attackers’ behavior when attempting to access a network. The hybrid or combined approach uses a mix of anomaly and misuse detection.

In the Transmission Control Protocol/Internet Protocol (TCP/IP) Internet model, there are typically four protocol layers used in the communication process between computers within a network. A detailed description of the respective layers can be found in [2]. These layers are shown in Figure 1.

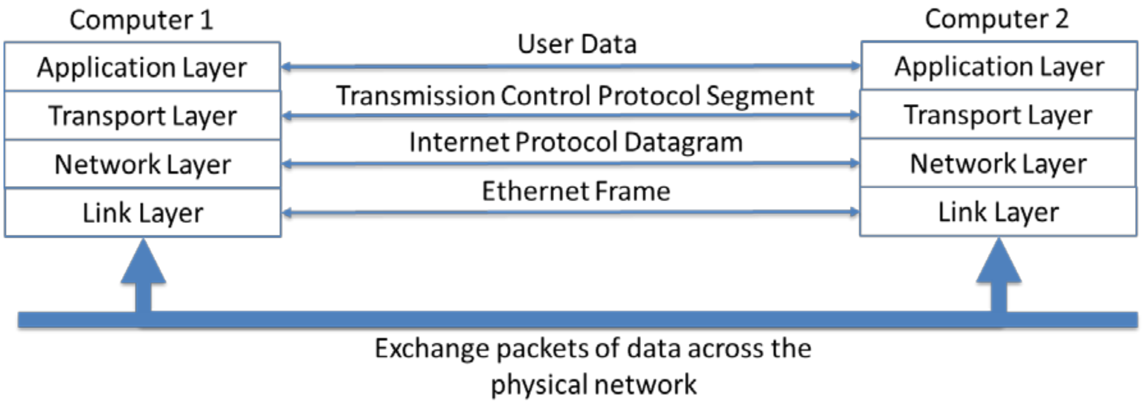


Figure 1. The TCP/IP Internet model as described in [2].

Network data can be collected from any of these protocol layers by various monitoring techniques and then passed to the IDS for analysis and categorization. The IDS first extracts the relevant features of the network data to construct a feature vector; the network data are typically captured from the network TCP/IP header and lower level protocol frames. Further information about headers and protocol frames can be found in Part I of [2]. The feature vector is then processed by a nonlinear network to identify the presence of attack (unauthorized intrusion or misuse) or normal traffic. The results produced by the IDS may then be used by a management system to negate the attack and identify its source.

Typically, the IDS handles a high-dimensionality input feature space, and the testing of the different feature-extraction and classification algorithms is often conducted using the KDD Cup 99 dataset [3]. This dataset was derived from the DARPA IDS evaluation dataset (1998) and has approximately “5 million connection records,” each representing a TCP/IP connection made up of 41 features. Each feature can either be qualitative or quantitative in nature.

In this thesis, the KDD Cup 99 data are enumerated, parsed and normalized to form a set of raw feature vectors used by the feature-extraction process. The feature-extraction process then extracts relevant features to derive a processed feature vector as an input to the classifier. A nonlinear network then uses these feature vectors to detect and classify the intrusion types (DoS, Probe, U2R, R2L, unknown). The training dataset, which consists of a combination of feature vectors with their associated output types is used to globally train and compute the classifier parameters. The probability of correct classification (PCC) is evaluated using the test dataset.

1. Feature Extraction Background

Features used for training and testing of the IDS must retain the anomalies and misuse information critical to the performance of the classifier. Network traffic data may also contain redundant and irrelevant features that can overwhelm and disable the IDS. Consequently, the key goal of feature-extraction is to identify and remove irrelevant, inconsistent, and redundant features, reducing the feature vector size. Also important is to preserve the optimal features to characterize the network data (e.g., normal traffic or attack traffic) while simultaneously maintaining the classifier accuracy.

A significant body of research exists on the process of feature-extraction.

In [4], a hybrid model (wrapper model and filter models) was explored to select an optimal feature vector. The wrapper model "uses the predictive accuracy of the classifier" to evaluate the quality of the feature vector. The filter model uses the "information, consistency or distance measures" to determine the relevance of the feature vector. The two methods are then combined to select an optimal feature vector [4].

Principal component analysis (PCA) and independent component analysis (ICA) were explored in [5]. The PCA transforms a set of features to a lower dimensional space while retaining the variability of the original data; however, the use of PCA to handle dynamic changes in the behavior of the network data was not addressed. To handle such changes, a hybrid data mining technique using a dynamic principle component analysis (DPCA) and a latent semantic analysis (LSA) were combined in [6].

In [7], a feature selection algorithm based on “Mahalanobis distance feature ranking” was used with an exhaustive search to identify appropriate features for identifying each attack type.

A feature selection approach based on a Bayesian network classifier was proposed in [8] and an optimal selection of features using genetic algorithm was presented in [9].

In [10], genetic k-means clustering methods were used to detect unknown attack patterns and remove irrelevant features in order to derive a smaller feature set while giving a higher classification accuracy of attacks.

A multivariate correlation detection system for DoS attack was presented in [11, 12]. This technique was able to detect “known and unknown DoS attacks” by considering “triangle-area-based multivariate correlation analysis” and geometrical correlations between traffic patterns of legitimate network traffic. Another flooding-based DoS attack detection which uses a feature subset selection algorithm was discussed in [13].

In [14], a correlation-based feature extraction method was used to reduce the size of the input feature space through the use of a partial-decision tree that parses out normal and abnormal behaviors. The assumption is that a good feature set contains features that are only highly correlated with the respective outcome types. In [15], irrelevant features are removed from a “ranked feature list based on the mutual information between each feature and the decision variable.” The decision variable refers to the detection and classification of network attacks.

2. Classification Background

Typical IDS tools (for example, SNORT) use rule-based filter techniques, which must be manually configured by a network analyzer for the detection of network attacks; however; the large number of inputs and the complexity of relationships between inputs makes it difficult for the network analyzer to be manually programmed with a comprehensive set of rules.

One possible IDS approach uses a nonlinear network to classify and identify the presence of a network attack (or normal traffic) as discussed in [16]. Specifically, Neural

Networks (NN) were found to be useful for IDS applications, where there is a need to differentiate between different types of network attacks. NNs are described in [17] as statistical and recursive learning models which "fit" a function based on a training dataset that has defined inputs and desired outputs; thus, NNs are suitable for tackling complicated problems which cannot be solved using rule-based or logic-based techniques. The large dimensionality of the network features makes the classification of network attacks complex and creates the case for the use of a NN.

B. PRINCIPAL CONTRIBUTIONS

An IDS software prototype was developed with the following design goals:

- Handle known classes of attacks;
- Handle unknown classes of attacks and classify accordingly;
- Operate in near real-time to classify continuous network traffic;
- Retrain based on new network data with minimal disruption to real-time operations.

The Knowledge Discovery and Data Mining (KDD) Cup 99 data was selected to test and numerically the IDS. The source of the KDD Cup 99 data is based on the NSL-KDD dataset as described in [3]. A raw data preprocessing software module, as described in Section II.C, was developed to parse the raw training and testing network features into a usable format for the subsequent feature-extraction module.

A feature-extraction software module, as described in Section II.D, was considered to reduce the network features, resulting in lower training times needed for the NN implementations.

A classification software module, as described in Section II.E was developed to detect the intrusion using the feature vector output from the feature-extraction software module. The classification software module uses NNs for training and classification of different types of network attacks. In addition, the classification software module was developed to test two different NN implementations and their performance was compared in terms of the PCC and efficiency.

C. THESIS OUTLINE

This thesis is organized as follows. In Chapter II, an overview of the process flow in the IDS and a description of each component of the IDS is detailed. In Chapter III, the results and implications of the IDS are discussed. Finally, conclusions and recommendations for future work are given in Chapter IV.

II. THE NN-BASED IDS

The implementation and design considerations of the NN-based IDS is described in this thesis.

A. OVERVIEW OF PROCESS FLOW IN IDS

The IDS implementation was developed using MATLAB 2014b and is divided into three software modules: raw data pre-processing, feature extraction and classification.

Each software module was designed to work independently with other modules, producing a set of output files used by the subsequent modules in the process flow. Any of the software modules in the process flow can be replaced provided the required inputs are available and in the correct format and syntax. With this built-in flexibility in mind, it is possible to adapt the raw data pre-processing module to allow the IDS to be trained with network traffic data other than the KDD Cup 99 dataset.

For this thesis research, two different classification architectures were explored and compared in terms of accuracy and computing times. The first architecture utilized a single NN for the classification process and is illustrated as a block diagram shown in Figure 2. The second architecture utilized three separate NNs for the classification process and is illustrated as a block diagram in Figure 3. This second structure was inspired by [18], where the training data was randomly separated into an arbitrary number of subsets for the purpose of temporal analysis of network traffic data, and each subset was used to train an individual classifier separately. In this thesis, the training data was split into three subsets, and each NN trains with each subset, respectively. The outputs from each of the three separate NNs were combined to produce the final output. Further details are provided in Section E.

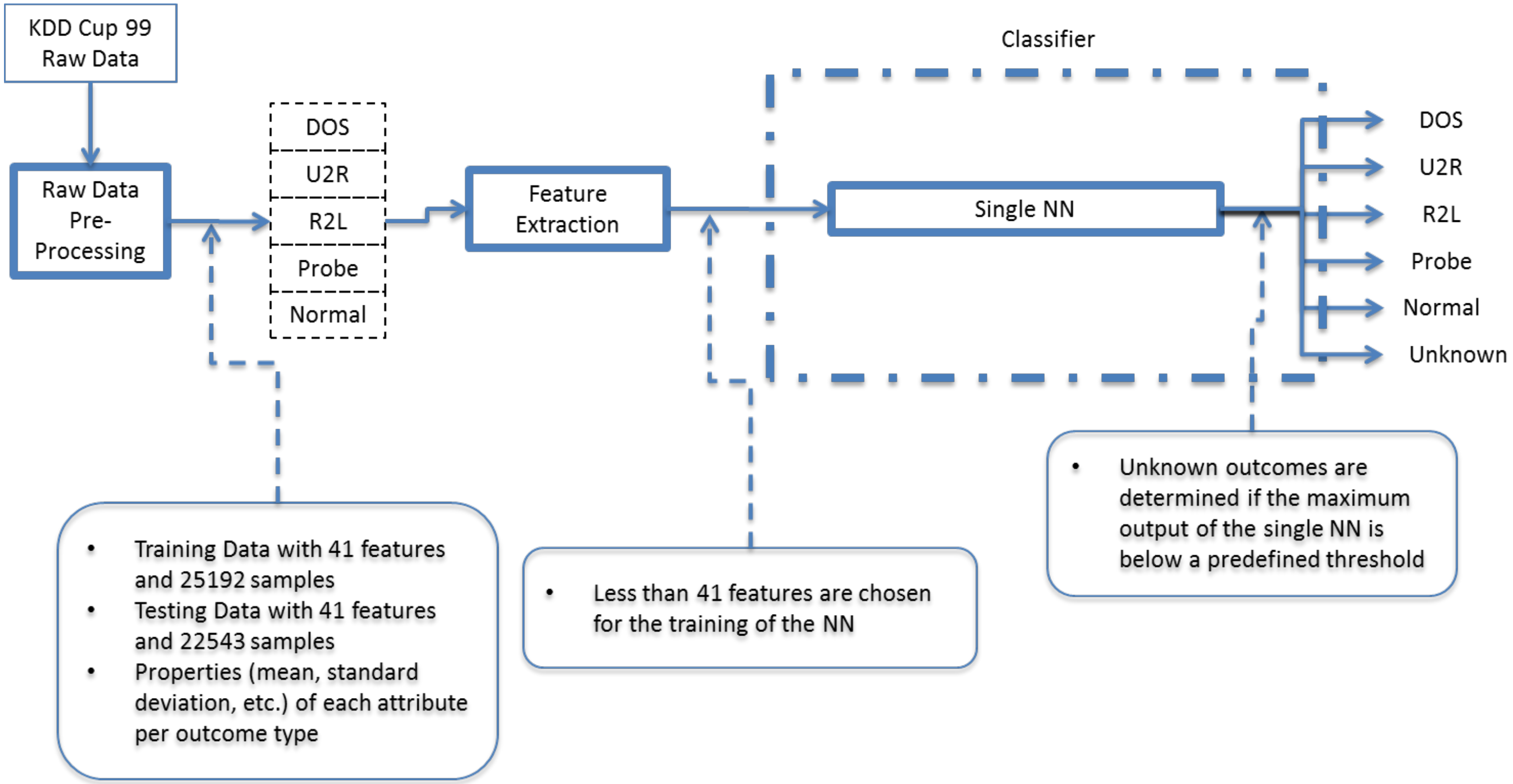
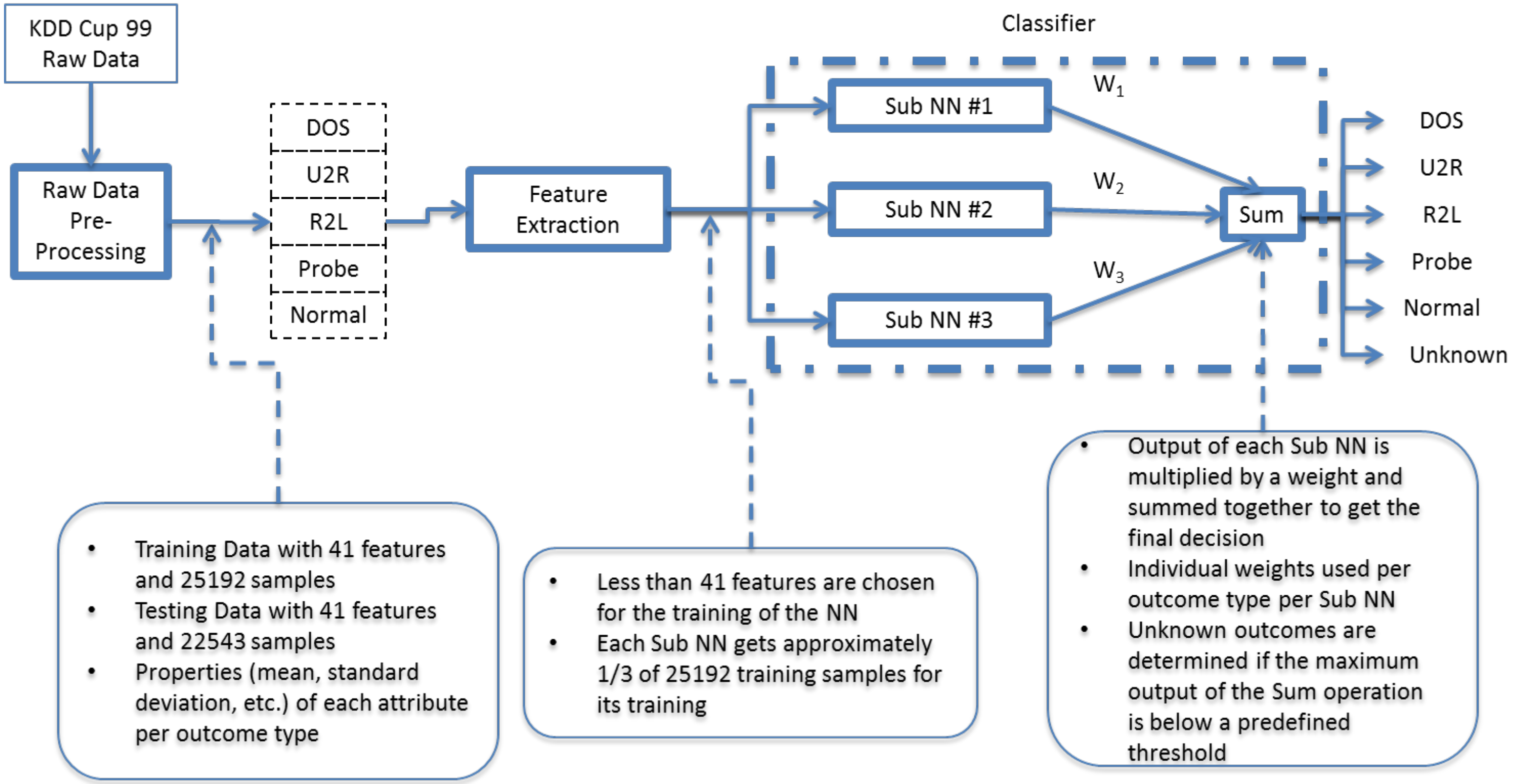

KDD Cup 99 Raw Data
Raw Data Pre-Processing
DOS
U2R
R2L
Probe
Normal
Feature Extraction
Classifier
Single NN
DOS
U2R
R2L
Probe
Normal
Unknown
• Training Data with 41 features and 25192 samples
• Testing Data with 41 features and 22543 samples
• Properties (mean, standard deviation, etc.) of each attribute per outcome type
• Less than 41 features are chosen for the training of the NN
• Unknown outcomes are determined if the maximum output of the single NN is below a predefined threshold

Figure 2. Single NN-based IDS process flow.



- Training Data with 41 features and 25192 samples
- Testing Data with 41 features and 22543 samples
- Properties (mean, standard deviation, etc.) of each attribute per outcome type

- Less than 41 features are chosen for the training of the NN
- Each Sub NN gets approximately 1/3 of 25192 training samples for its training

- Output of each Sub NN is multiplied by a weight and summed together to get the final decision
- Individual weights used per outcome type per Sub NN
- Unknown outcomes are determined if the maximum output of the Sum operation is below a predefined threshold

Figure 3. Three-parallel NN-based IDS process flow.

B. KNOWLEDGE AND DATA DISCOVERY (KDD) CUP 99 DATA

The KDD Cup 99 data set was developed for use in the Third International Knowledge Discovery and Data Mining Tools Competition. The dataset was created through a raw TCP data dump of a simulated U.S. Air Force local-area network (LAN), where the network was subjected to deliberate network attacks.

Each training sample in the training dataset represents a connection summary, and there are a total of 25,192 samples in the training dataset. Each sample can be classified as one of the 22 training attack types and can be further categorized as one of the general five outcome types: DOS, U2R, R2L, Probe or Normal. Refer to Appendix A for the full list of outcome types, and a detailed description of each of the outcome types can be found in [3]. Each connection summary is described using 41 features as listed in Appendix B, which the NN classifier uses as inputs for both training and testing.

In the test dataset, each sample is similarly a connection summary, and there are a total of 22,543 samples in the test dataset. In addition, there are 15 types of attacks which are not found in the training dataset. For the purpose of testing the ability of the IDS to handle unknown attack types, these 15 types of attacks are classified as unknown types in the performance evaluation phase.

A breakdown of the number of outcome types for the testing and training dataset is shown in Table 1.

Table 1. Breakdown of outcome types in the KDD cup 99 training and test datasets.

Dataset Type	Number of Samples per Outcome Type					
	DOS	U2R	R2L	Probe	Normal	Unknown
Training	9234	11	209	2289	13449	0
Testing	5651	37	2199	1106	9710	3750

The KDD Cup 99 dataset was downloaded from a Git repository [19], which uses the NSL-KDD dataset. The NSL-KDD dataset, as discussed in [3], is a corrected version of KDD Cup 99 dataset, which addresses issues and problems in the original KDD Cup 99 dataset.

It must be noted that the KDD Cup 99 dataset is solely used for the purpose of developing the IDS software. It was noted by Brugger [20] that the KDD Cup 99 data is outdated and not fully representative of network traffic attacks; thus, there is a need for the IDS to be validated with updated and more representative network data in future projects.

C. RAW DATA PRE-PROCESSING

The raw data pre-processing stage takes the KDD Cup 99 training data and converts it into a matrix of representative numerical data for the feature-extraction and classification stage. This stage performs the pre-processing described in the following paragraphs.

Transforming semantic features into enumeration allows subsequent stages to process these features as numerical values. In addition, the same set of conversion rules applies to both training and test dataset; thus, both training and test dataset must use the same set of semantics for the respective features. For the “protocol_type,” “service,” and “flag” fields, the enumerations used are shown in Appendix C.

Normalizing features scales features in the range 0 to 1, preventing the training process from being affected by large variations in the feature values.

Formatting raw data for MATLAB input saves the processed raw data in a suitable format for the feature-extraction and classification stage. For example, in this simulation setup, 25,192 training samples were in the raw data, each training sample has 41 features, and each training sample has one of five possible general outcomes; thus, a single training data matrix that encompasses the features and outcomes for every one of the 25,192 training samples was produced as an output.

Calculating mean, standard deviation, and histogram of features per outcome type provides data for the feature-extraction stage to determine if the feature is useful for classification. In this case, the software module calculates the mean and standard deviation of each feature for each outcome type. For example, a 5-by-41 matrix is generated for the mean of the features per outcome type, where there are five outcome types and 41 features. In addition, the histogram of the feature values, for each feature per outcome type, is also created for visual analysis purposes. The histograms generated are shown in Appendix D.

D. FEATURE EXTRACTION

This software module uses the properties of each feature per outcome type to evaluate the features that are useful at the training stage. For this application, an array of 1s and 0s is generated, where a value of 1 indicates to the classification stage that the respective feature is to be used for training, while a value of 0 means the opposite.

The feature-extraction stage uses two methods to remove redundant or irrelevant features.

1. Removal of Features with Small Variations

The first filtering mechanism removes all features which remain relatively constant across different outcome types. For a feature to be considered “constant,” the conditions described in the following paragraphs must be satisfied.

The mean of a feature is approximately the same for all outcome types. In such a case, the classification stage is not able to use this feature to differentiate between different outcome types during the training stage. The mean of the five outcome types are tabulated, and the standard deviation of the five means are then calculated. If this standard deviation is less than the empirical threshold value of 0.0001, the mean values of the five outcome types are considered to be similar.

The standard deviation value of the feature is small for all outcome types. In such a case, features vary within a relatively small range, making this feature useless for training purposes. The standard deviation of the five outcome types are first tabulated. If all five standard deviation values are less than the empirical threshold value of 0.001, this means that the feature does not vary significantly for all outcome types.

2. Removal of Highly Correlated Features

In this step, the correlation coefficient matrix of the features is calculated, and pairs of features which have a correlation coefficient value larger than or equal to 0.97 are identified. For each pair of highly correlated features, only one of the features is selected for the feature vector and used at the classification stage. The correlation coefficient matrix is calculated by

$$R(i,j) = \frac{C(i,j)}{\sqrt{C(i,i)C(j,j)}} \qquad (1)$$

where i and j represent one of the 41 features and $C(i,j)$ is the corresponding covariance matrix. The covariance matrix $C(i,j)$ is defined as

$$C(i,j) = \frac{\sum_{k=1}^{N}(i_k - \overline{i})(j_k - \overline{j})}{N} \qquad (2)$$

where $\overline{i}$ and $\overline{j}$ represents the mean of the respective features and N is the number of training samples in the training dataset. The mean $\overline{i}$ is defined as

$$\overline{i} = \frac{\sum_{k=1}^{N} i_k}{N} \qquad (3)$$

and $\overline{j}$ is also defined similarly.

To summarize, a flow chart of the raw data pre-processing and feature-extraction processes for the KDD Cup 99 data before the classification stage is illustrated in Figure 4, and the list of features used for the classification stage is shown in Table 2.

E. NN IMPLEMENTATIONS

1. The Supervised Learning NN

The NN is a nonlinear process that is trained to perform a particular task (filtering, predicting, identifying patterns, etc.). In this thesis, the NN is implemented using the MATLAB software.

Table 2. List of features used for the classification stage.

S/N	**Name**
1	duration
2	protocol_type
3	service
4	flag
5	src_bytes
6	dst_bytes
7	wrong_fragment
8	urgent
9	hot
10	num_failed_logins
11	logged_in
12	root_shell
13	su_attempted
14	num_root
15	num_file_creations
16	num_shells
17	num_access_files
18	is_guest_login
19	count
20	srv_count
21	srv_rerror_rate
22	same_srv_rate
23	diff_srv_rate
24	srv_diff_host_rate
25	dst_host_count
26	dst_host_srv_count
27	dst_host_same_srv_rate
28	dst_host_diff_srv_rate
29	dst_host_same_src_port_rate
30	dst_host_srv_diff_host_rate
31	dst_host_srv_serror_rate
32	dst_host_rerror_rate
33	dst_host_srv_rerror_rate

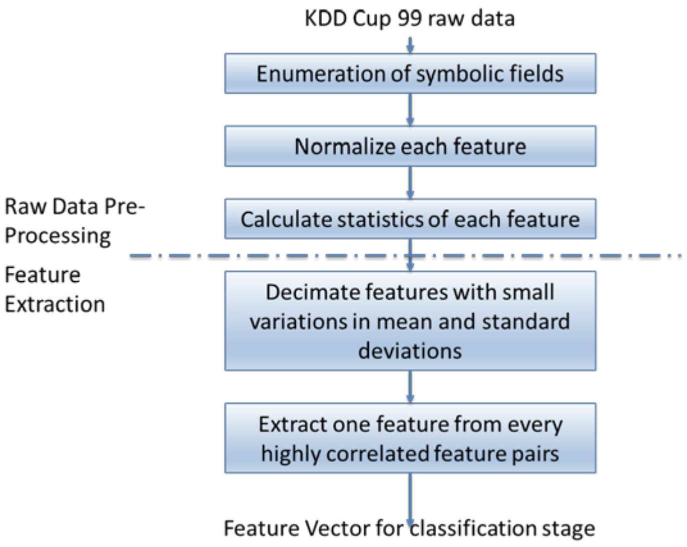

KDD Cup 99 raw data
Enumeration of symbolic fields
Normalize each feature
Calculate statistics of each feature
Raw Data Pre-Processing
Feature Extraction
Decimate features with small variations in mean and standard deviations
Extract one feature from every highly correlated feature pairs
Feature Vector for classification stage

Figure 4. Flowchart of KDD cup 99 data processing prior to classification stage.

The NN consists of an interconnected network of basic computing blocks called neurons. Inter-neuron connections contain synaptic weights, and the synaptic weights are used to retain knowledge obtained through the training process. Through the training process, the synaptic weights are tuned to achieve the training objective. In the NN toolbox provided by MATLAB [21], training functions are defined explicitly as global algorithms for the synaptic weights tuning process. In addition to the training algorithm, the NN toolbox also allows the definition of post-processing algorithms to evaluate the network performance after the training process. A defined proportion of the training dataset also needs to be allocated for validation purposes in the training process, preventing overfitting of the NN to the training dataset.

In this thesis, the NN type of interest is the multi-layer feedforward network. In such networks, the neurons are arranged in layers. The most basic form is the single-layer feedforward network, where the input source nodes directly project to a layer of output nodes. The multi-layer feedforward network extends the single-layer feedforward network through the addition of one or more hidden layers in the network. The NN used in this thesis consists of one hidden layer and one output layer. The hidden layer creates an additional set of synaptic connections between the input nodes and output layer of neurons, which allows the derivation of higher-order statistics and relationships; this is especially important in applications where the inputs have large dimensions.

Each neuron in its respective layer is connected to every other node in the adjacent forward layer. The number of neurons in the single hidden layer is derived based on a rule-of-thumb from [22], where the number of neurons is the sum of two-thirds the number of features and the number of outcome types. If the value calculated is a decimal number, the decimal number is rounded up.

The NN output y_k can be mathematically described as [23]

$$y_k(l) = f_s[\sum_{h=1}^{H} w_{kh} f_{sh}(\sum_{i=1}^{I} w_{hi} x_i(l))] \tag{4}$$

where k is the respective outcome type, x_i is the input, i is the respective feature of the input, I represents the total number of features per input sample and the sample number of the input is l . The total number of hidden layers is H, and h represents the respective hidden layer. The weights from neuron i to neuron h and the weights from neuron h to neuron k are, respectively, defined as w_{hi} and w_{kh}. The hidden layer activation function and the output layer activation function are defined, respectively, as f_{sh} and f_s. The hidden layer activation function f_{sh} uses the tangent sigmoid function

$$f_{sh}(z) = \frac{2}{1+e^{-2z}} - 1, \tag{5}$$

where z is the input to the activation function. The output layer activation function f_s uses the normalized exponential function

$$f_s(z) = \frac{e^z}{\sum_{k=1}^{K} e^{z_k}} \tag{6}$$

where K is the total number of outcome types.

The reader can also refer to [17] for further details on the multi-layer feedforward NN.

2. Single NN Classification Module

The NN is trained using 25,192 training samples. The IDS must classify each network event as one of the five known outcome types or an unknown type. The NN chosen for this module has the characteristics discussed in the following paragraphs.

The NN has a single hidden layer, and the MATLAB "patternnet" function was used to create the NN. For the scenarios in Section III, the number of neurons in the hidden layer is described as follows. In the baseline scenario without feature-extraction, the hidden layer uses 33 neurons, which is calculated based on 2/3(41) + 5 (41 input features and five NN outcome types). In the scenario utilizing feature-extraction, the hidden layer uses 27 neurons, which is calculated based on 2/3(33) + 5 (33 input features and five NN outcome types).

The NN utilizes a supervised learning approach with the training function as scaled conjugate gradient and the performance function as cross-entropy. The training proportion of the training dataset is 90%, and the validation proportion of the training dataset is 10%. The division of the training dataset into training and validation sets was performed randomly. A separate test dataset was used to test the NN.

The NN takes in two input matrices at the training stage: a training feature matrix that contains the feature vectors and a training outcome matrix that indicates the respective output type for every feature vector.

For every test input to the NN, the NN outputs a vector of five numbers, where each element represents one of the five known outcome types. The maximum value of any element is one, and the output of the NN is indicated by the element with the maximum value.

As a result of the supervised learning approach implemented, the NN is unable to classify unknown events type in the test dataset as unknown; thus, an additional capability was included to classify unknown output types. First, a sixth element was added to the NN output vector as a placeholder for the unknown outcome type. The unknown type was determined through the use of a threshold value; if the maximum output value of the NN is below the empirical threshold value of 0.8, this means that the

NN is not able to match its output with one of the known output types with high confidence. In this case, the NN output defaults to the unknown type.

3. Three-Parallel NN Classification Module

The next architecture is comprised of three separate NN modules, where each NN is trained using 1/3 of the 25,192 training samples. Each NN has the same properties as those described in Section 2.

The training set used for each NN is exclusive, and the three NNs work in parallel. To classify unknown event types, the output of each NN is subjected to the same processing as described in Section 2.

Separate outputs are combined based on the number of samples within the training set of the respective NNs. If no training samples of an outcome type are used to train a NN, that respective NN is given a weight of zero in influencing the joint decision of the three NNs. Conversely, if all training samples of an outcome type are used to train a NN, that NN has a maximum weight of one in influencing the joint decision of the three NNs and the other two NNs have no influence on the joint decision.

The weight used for each of the smaller NN module is calculated based on the following:

- For each NN, the number of training samples per outcome type is tabulated as X_{ij}, where i represents one of the three NNs and j represents an outcome type.
- The total number of training samples per outcome type in the full training set is tabulated as Y_j, where j represents a specific outcome type.
- The weight applied to each outcome type for the respective NN is W_{ij}, where i represents one of the three NNs and j represents one outcome type. The respective weight is calculated by $W_{ij} = X_{ij} / Y_j$.
- For unknown outcome types, the default weight is the inverse of the number of NNs used in decision fusion process, i.e. 1/3.

The output of a respective NN for a particular outcome type is o_{ij}, where i represents one of the three NNs and j represents an outcome type. The outputs of each

NN are then multiplied by the respective weights and summed together to create the final output. The fused outcome O_j, where j represents one of the outcome types, is obtained from

$$O_j = \sum_{i=1}^{3} W_{ij} o_{ij} \quad . \tag{7}$$

Similarly to the single NN architecture, the final NN output is indicated by the element with the highest value in the output vector.

The IDS implementation and performance is discussed in the next chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SIMULATION RESULTS AND DISCUSSION

Simulation results obtained for the IDS architectures are discussed in this chapter; the PCC and timing results obtained for the respective scenarios are presented in Sections A and B, and a discussion of the results is given in Section C. To allow comparison and benchmarking of results, all experiments were performed on a MacBook Pro (Retina, 15-inch, Early 2013) with 2.8 GHz Intel Core i7 Processor that was installed with 16 GB 16000 MHz DDR3 Random Access Memory (RAM), and the Operating System used was MAC OS X Yosemite (Version 10.10.3). The MATLAB version used was 2014b.

A. BASELINE CLASSIFIER WITHOUT FEATURE EXTRACTION

This scenario was conducted as a baseline for comparison. The feature-extraction stage is bypassed, and all 41 features were used to train and test the NN.

The same pattern recognition feedforward NN structure was used for the single NN and the three-parallel NN implementation. The NN uses 41 inputs, 33 hidden neurons and five neurons for the output layer. The training samples have five different outcome types, which results in five output neurons. The sixth "unknown" type output is determined by the logic as described earlier in Section II.E.2. An illustration of the NN implemented in MATLAB is shown in Figure 5.

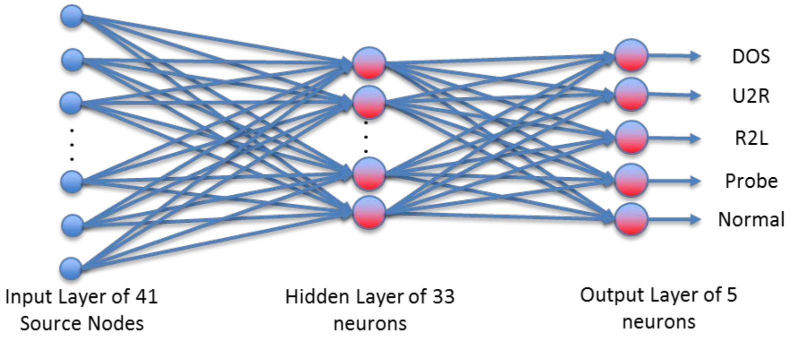


Figure 5. Forty-one input pattern recognition feed-forward NN structure.

1. Probability of Correct Classification Results

Each NN was tested individually with the test dataset. The single NN output was compared with the desired outcome via the "plotconfusion" function provided by MATLAB; detailed confusion matrix results are shown in Appendix E. For the respective three-parallel NN classification module, each parallel NN output and the merged output of the three NNs were compared with the desired outcome via the same method. PCC results are shown in Table 3. Due to the use of random seeds in the training process of the NN, the PCC results vary slightly, but results shown here are representative of the NN average classification performance.

Table 3. PCC for baseline scenario on test dataset.

NN Type	PCC (%)					
	DOS	U2R	R2L	Probe	Normal	Unknown
Sub NN #1	91.0	0.0	0.9	72.6	95.6	34.9
Sub NN #2	93.5	0.0	0.1	73.9	96.6	24.2
Sub NN #3	92.7	0.0	5.2	74.3	96.6	8.6
Merged output of three-parallel NN	93.2	0.0	0.4	73.5	96.5	20.8
Single	91.8	0.0	0.0	73.0	95.8	27.5

2. Timing Results

To measure the amount of time taken to train the respective NN, 30 runs were conducted to find the mean-training time and associated standard deviation for the respective NN implementations. It is also noted that the time taken by the NN to produce the output from the test dataset is almost negligible; for a test dataset of 22,543 samples, the NN takes less than one second to produce the output. Training times for the respective NNs are presented in Table 4. It is also observed that MATLAB has sufficient RAM to execute each simulation run without impacting timing measurements.

Table 4. NN training execution time (average over 30 Runs) – baseline scenario.

NN Type	Average (seconds)	Standard Deviation (Seconds)
Sub NN #1	2.8092	0.6242
Sub NN #2	2.5920	0.7613
Sub NN #3	2.5597	0.7307
Total time for three-parallel NN Implementation (Sequential)	7.9609	2.1162
Single	10.8496	3.636

B. REDUCED FEATURE SIZE CLASSIFIER IMPLEMENTATION

The effects of feature-extraction on classifier performance and the results compared with those from the baseline scenario are considered in this scenario. Recall, the feature-extraction stage keeps only 33 of the 41 original features.

The same pattern recognition feedforward NN structure was used for the single NN and the three-parallel NN implementations. The NN uses 33 inputs, 27 hidden neurons and five neurons for the output layer. The training samples have five different outcome types, which results in five output neurons. The sixth "unknown" type output is determined by the logic as described in Section II.E.2. An illustration of the NN implemented in MATLAB is shown in Figure 6.

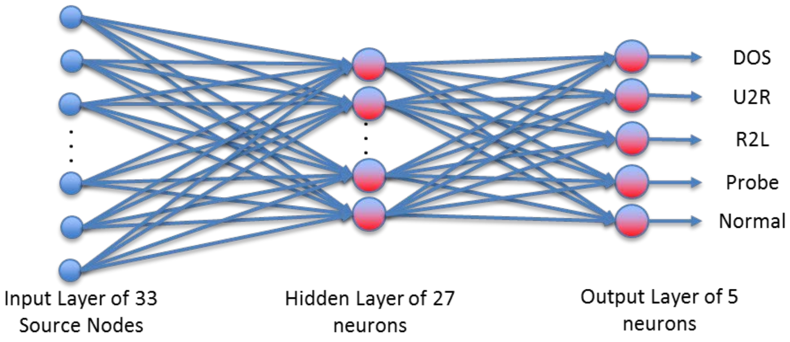

DOS
U2R
R2L
Probe
Normal
Input Layer of 33 Source Nodes
Hidden Layer of 27 neurons
Output Layer of 5 neurons

Figure 6. Thirty-three input pattern recognition feed-forward NN structure.

1. Probability of Correct Classification Results

Similar to the baseline scenario, each NN was tested individually with the test dataset using the same methods as described earlier; detailed confusion matrix results are shown in Appendix F. The PCC results are presented in Table 5. Due to the use of random seeds in the training process of the NN, the PCC results vary slightly, but results shown here are representative of the NN average classification performance.

Table 5. PCC for reduced feature size classifier on test dataset.

NN Type	PCC (%)					
	DOS	U2R	R2L	Probe	Normal	Unknown
Sub NN #1	91.4	2.7	0.0	71.8	95.0	14.8
Sub NN #2	92.0	0.0	0.0	73.9	96.1	24.1
Sub NN #3	91.6	0.0	0.1	73.3	95.7	34.8
Merged output of three-parallel NN	91.9	0.0	0.0	73.1	96.1	25.2
Single	94.2	0.0	0.2	73.9	96.9	17.3

2. Timing Results

Similar to the baseline scenario, training times for the respective NNs are shown in Table 6. To get the reduction in average training time in Table 6, the new average training time is divided by the baseline average training time to get the percentage equivalent. The reduction percentage is then obtained by subtracting this value from 100%. It is also observed that MATLAB has sufficient RAM to execute each simulation run without impacting timing measurements.

Table 6. NN training execution time (average over 30 runs) – reduced feature size scenario.

NN Type	Average (seconds)	Standard Deviation (Seconds)	Percentage Reduction in Average Training Time as compared to Baseline Scenario (%)
Sub NN #1	2.1874	0.6282	22.2
Sub NN #2	1.8743	0.4059	27.6
Sub NN #3	2.0267	0.5745	20.82
Total time for three-parallel NN Implementation (Sequential)	6.0884	1.6086	23.5
Single	8.6757	2.4317	20

C. DISCUSSION OF RESULTS

Results obtained in this study raise the following discussion points.

1. Feature Extraction Impacts on Training-Stage Execution Time

Reducing the feature size does not degrade classification performances in either NN implementation; the feature-extraction stage successfully removed irrelevant features which did not serve to improve the NN classification capabilities.

The main benefit is a reduction in the training time for both NN implementations, as shown in Table 6. This characteristic is beneficial when there is a need to retrain NNs with new network data while minimizing disruption to real-time detection of network intrusions.

2. Comparison of NN Implementations

The overall training time is shorter for the three-parallel NN implementation, and the three-parallel NN implementation has comparable PCC performance to the single NN implementation. This is seen in Table 4 and Table 6, where the overall training time is shorter than that required for the single NN implementation, when all three sub NNs are trained sequentially.

The training time for the three-parallel NN implementation can be further improved when three parallel computing threads are used to train each sub NN. In such a case, the total training time is based on that obtained for the sub NN with the longest training time.

The retraining time for the three-parallel NN implementation can improve further if the feature-extraction stage determines that new training data uses the same set of features as the original training data. In this case, a three-parallel NN only needs to train one sub NN with the latest network data and replace either the oldest or worst performing sub NN. In comparison, a single NN implementation needs to retrain the whole NN with the old and new data.

3. Effects of the Training Dataset

Results show that U2R and R2L outcome types have low PCC for both NN implementations. U2R and R2L outcome types have fewer training samples than DOS, Probe and Normal training samples, as seen from Table 1. As a result, the training dataset is imbalanced, which may have led to insufficient training of the NN in classifying the U2R and R2L outcome types in the testing database.

Classification results obtained for DOS, Probe and Normal outcome types are better. This is due to the large number of training samples for the DOS, Probe and Normal outcome types, which allows the NN to be sufficiently trained.

The use of the threshold method as described in Section II.E.2 to determine unknown outcome types is only able to generate approximately 20% PCC. It is noted that the threshold can be set higher to allow more unknown events to be classified successfully; a higher threshold that the NN output needs to be larger to be classified as one of the five known outcome types. This also comes with the downside of having more of the known outcome types being classified as unknown outcome types.

Results obtained with the NN based classifier considered in this study were presented in this section. Conclusions and recommendations for future work are provided in the next chapter.

IV. CONCLUSIONS AND RECOMMENDATIONS

Conclusions and recommendations for future work are given in this chapter.

The main purpose of this thesis was to develop an NN-based supervised IDS. The IDS considered contains three interchangeable modular software components. The first module pre-processes the raw training and testing data, the second module applies feature-extraction, and the last module performs the NN training and the classification of network events. Single NN and three-parallel NN implementations were developed in the classification module for comparison of PCC and timing performances.

The performance of the IDS implementation was tested using the KDD Cup 99 dataset using separate testing and training sets. Simulations were conducted to investigate the effects of feature-extraction and compute performances obtained with the single NN and three-parallel NN implementations.

Results show the feature-extraction stage removes irrelevant features without impacting PCC while reducing the training time.

While the three-parallel NN implementation is comparable in PCC performance to the single NN implementation, it was shown to be superior in terms of training time. This makes the three-parallel NN implementation a possible candidate for use in real-time applications, when the IDS needs to frequently retrain to handle new types of network attacks.

The following areas are recommended for further work.

Further analysis of the histogram obtained for each feature on a per outcome basis can be performed and used for additional processing in the feature-extraction module.

The IDS considered in this thesis is a signature-based IDS, which detects network attacks or intrusions through patterns in the features. Other approaches such as anomaly-based IDS can be considered to complement signature based IDS, as suggested in [24], where anomaly-based IDS are used to detect behavioral deviations from normal network behavior. Future projects may include the development of a separate anomaly-based IDS to complement the results of the IDS considered in this study.

The IDS can be configured based on the number of sub-NNs present in a parallel NN implementation, threshold values to remove unneeded features, number of neurons in the hidden layer, number of hidden layers, testing-to-validation ratio used for NN training and threshold values to determine unknown outcome types; thus, future work should consider optimizing these parameters.

APPENDIX A. TYPES OF NETWORK ATTACKS FOR KDD CUP TRAINING DATA

Name	**Type**
Back	dos
buffer_overflow	u2r
ftp_write	r2l
guess_passwd	r2l
imap	r2l
ipsweep	probe
land	dos
loadmodule	u2r
multihop	r2l
neptune	dos
nmap	probe
perl	u2r
phf	r2l
pod	dos
portsweep	probe
rootkit	u2r
satan	probe
smurf	dos
spy	r2l
teardrop	dos
warezclient	r2l
warezmaster	r2l

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. FEATURES LIST FOR KDD CUP DATA

S/N	Name	Type
1	duration	continuous
2	protocol_type	symbolic
3	service	symbolic
4	flag	symbolic
5	src_bytes	continuous
6	dst_bytes	continuous
7	land	continuous
8	wrong_fragment	continuous
9	urgent	continuous
10	hot	continuous
11	num_failed_logins	continuous
12	logged_in	continuous
13	num_compromised	continuous
14	root_shell	continuous
15	su_attempted	continuous
16	num_root	continuous
17	num_file_creations	continuous
18	num_shells	continuous
19	num_access_files	continuous
20	num_outbound_cmds	continuous
21	is_host_login	continuous
22	is_guest_login	continuous
23	count	continuous
24	srv_count	continuous
25	serror_rate	continuous
26	srv_serror_rate	continuous
27	rerror_rate	continuous
28	srv_rerror_rate	continuous
29	same_srv_rate	continuous
30	diff_srv_rate	continuous
31	srv_diff_host_rate	continuous
32	dst_host_count	continuous
33	dst_host_srv_count	continuous
34	dst_host_same_srv_rate	continuous
35	dst_host_diff_srv_rate	continuous
36	dst_host_same_src_port_rate	continuous

37	dst_host_srv_diff_host_rate	continuous
38	dst_host_serror_rate	continuous
39	dst_host_srv_serror_rate	continuous
40	dst_host_rerror_rate	continuous
41	dst_host_srv_rerror_rate	continuous

APPENDIX C. ENUMERATION USED FOR SYMBOLIC FEATURES

Enumeration Types for “protocol_type”	
Name	**Enumeration Value**
‘tcp’	1
‘udp’	2
‘icmp’	3

Enumeration Types for “service”	
Name	**Enumeration Value**
‘ftp_data’	1
‘other’	2
‘private’	3
‘http’	4
‘remote_job’	5
‘name’	6
‘netbios_ns’	7
‘eco_i’	8
‘mtp’	9
‘telnet’	10
‘finger’	11
‘domain_u’	12
‘supdup’	13
‘uucp_path’	14
‘Z39_50’	15
‘smtp’	16
‘csnet_ns’	17
‘uucp’	18
‘netbios_dgm’	19
‘urp_i’	20
‘auth’	21
‘domain’	22
‘ftp’	23
‘bgp’	24
‘ldap’	25
‘ecr_i’	26
‘gopher’	27
‘vmnet’	28

‘systat’	29
‘http_443’	30
‘efs’	31
‘whois’	32
‘imap4’	33
‘iso_tsap’	34
‘echo’	35
‘klogin’	36
‘link’	37
‘sunrpc’	38
‘login’	39
‘kshell’	40
‘sql_net’	41
‘time’	42
‘hostnames’	43
‘exec’	44
‘ntp_u’	45
‘discard’	46
‘nntp’	47
‘courier’	48
‘ctf’	49
‘ssh’	50
‘daytime’	51
‘shell’	52
‘netstat’	53
‘pop_3’	54
‘nnsp’	55
‘IRC’	56
‘pop_2’	57
‘printer’	58
‘tim_i’	59
‘pm_dump’	60
‘red_i’	61
‘netbios_ssn’	62
‘rje’	63
‘X11’	64
‘urh_i’	65
‘http_8001’	66

Enumeration Types for “flag”	
Name	Enumeration Value
‘SF’	1
‘S0’	2
‘REJ’	3
‘RSTR’	4
‘SH’	5
‘RSTO’	6
‘S1’	7
‘RSTOS0’	8
‘S3’	9
‘S2’	10
‘OTH’	11

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. HISTOGRAM OF FEATURES PER OUTCOME TYPE

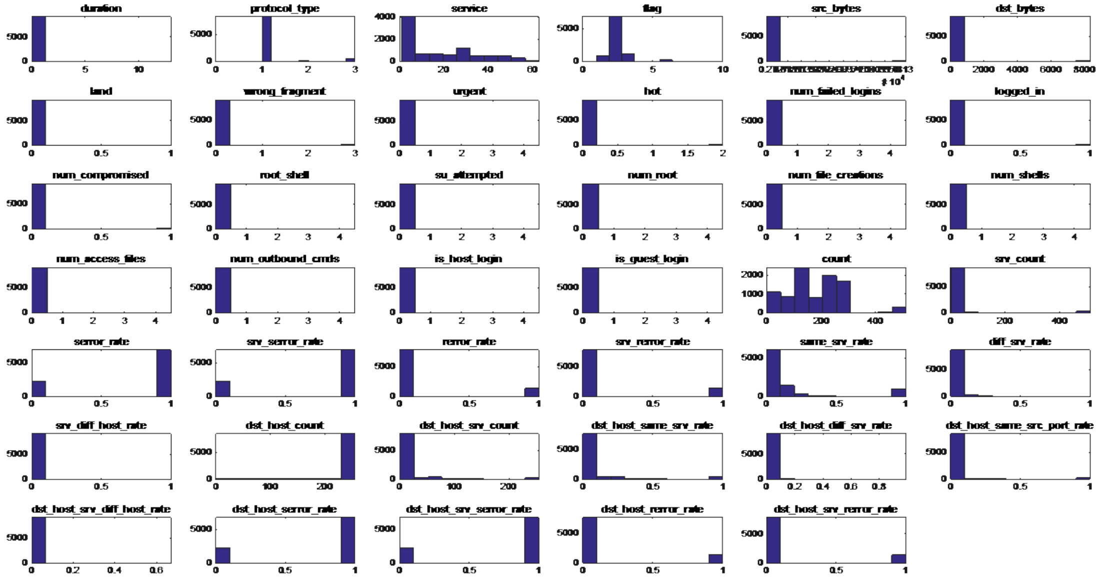


Figure 7. Features of KDD cup 99 data for dos attack; histogram of features values.

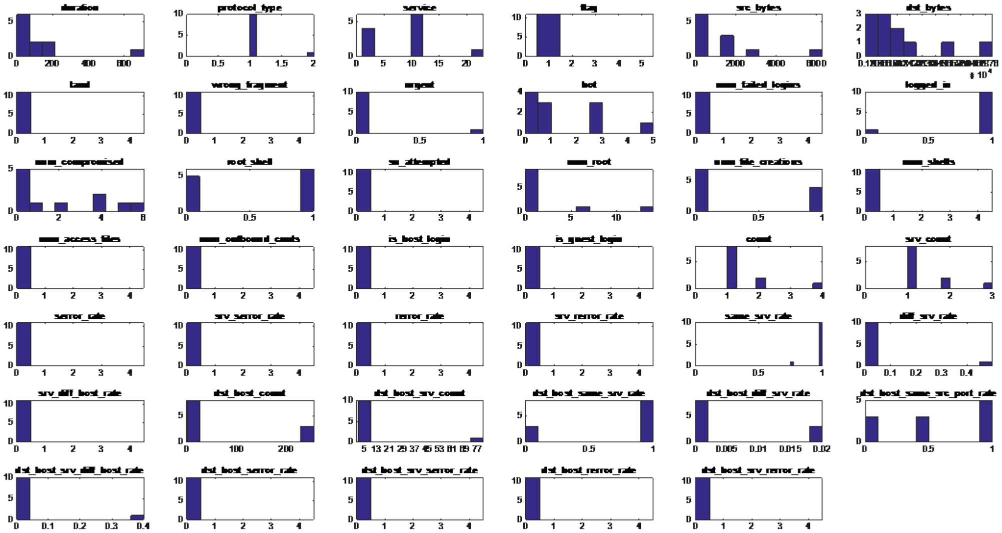


Figure 8. Features of KDD cup 99 data for u2r attack; histogram of features values.

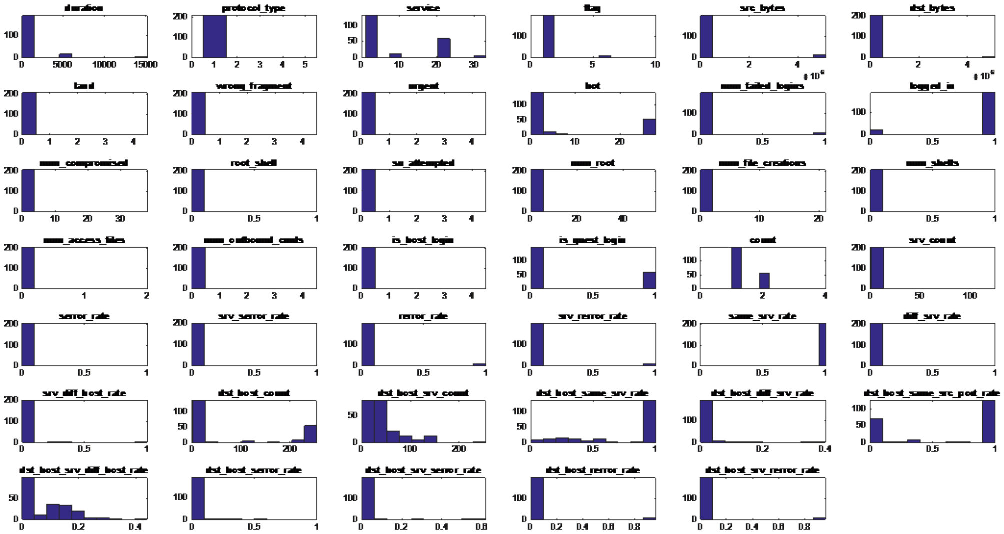


Figure 9. Features of KDD cup 99 data for r2l attack; histogram of features values.

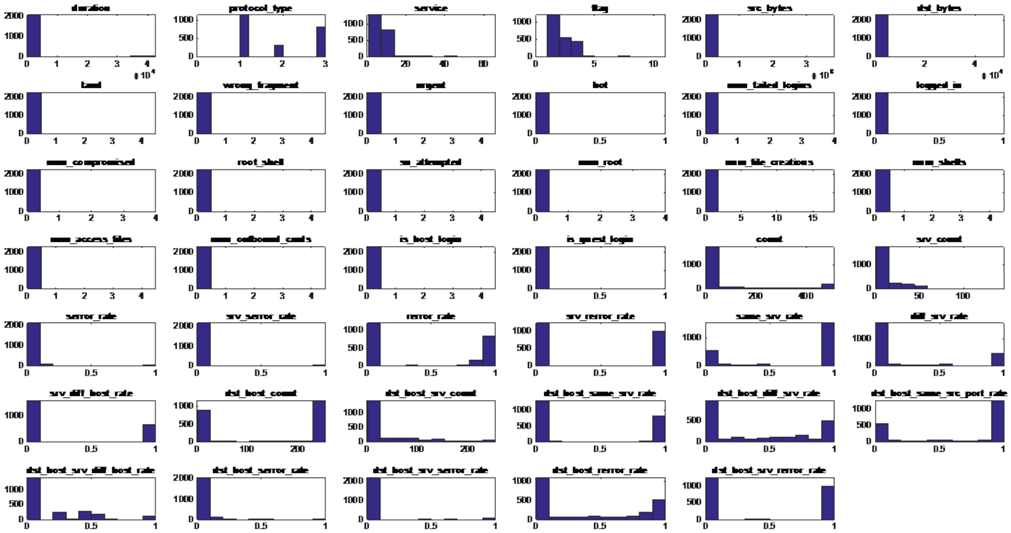


Figure 10. Features of KDD cup 99 data for probe attack; histogram of features values.

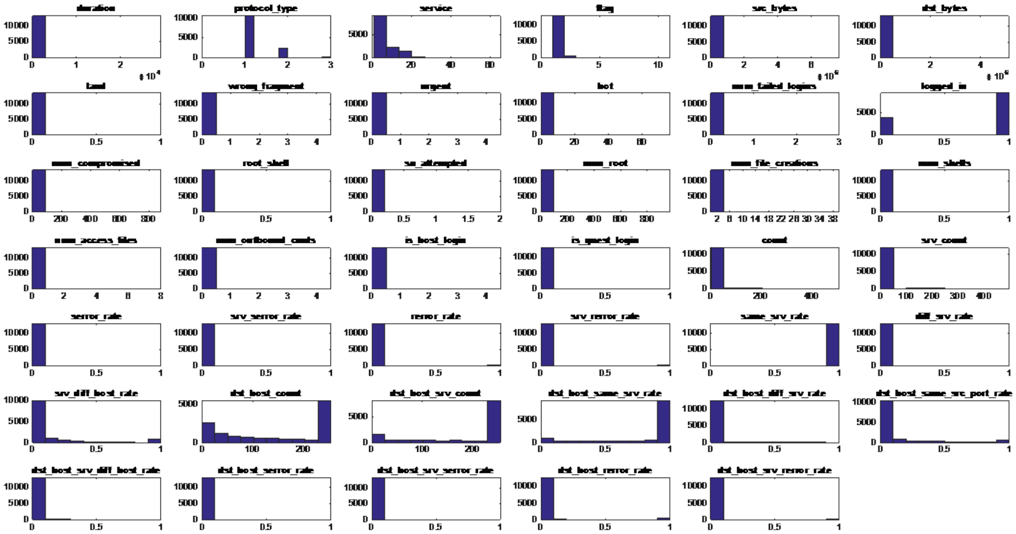
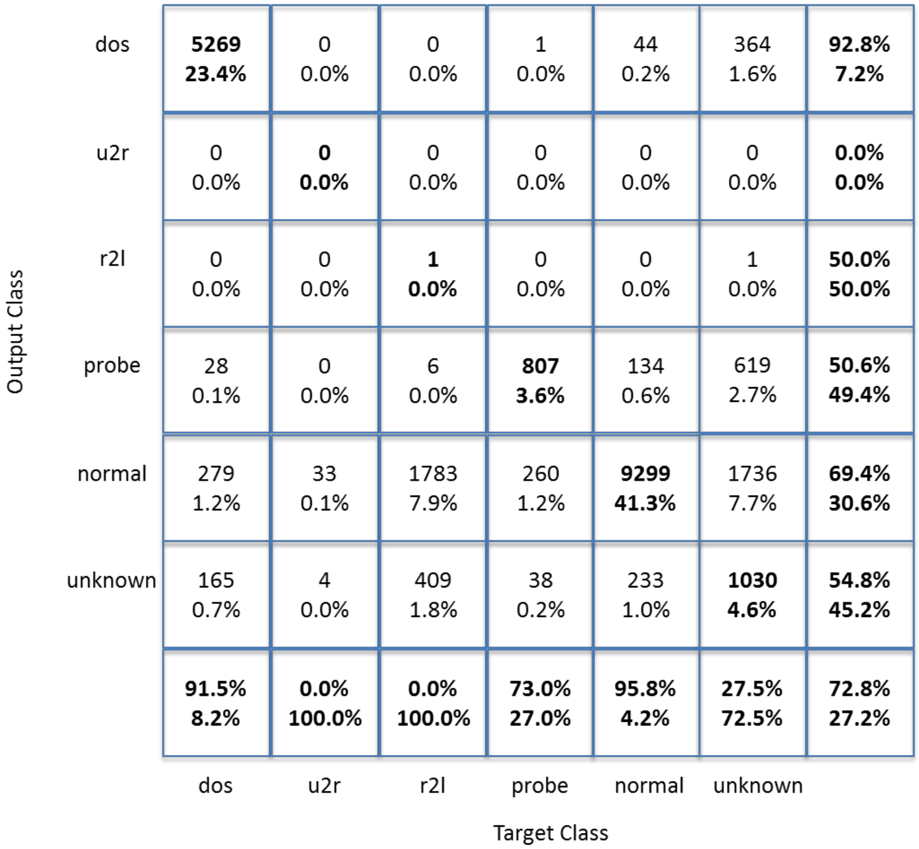


Figure 11. Features of KDD cup 99 data for normal; histogram of features values.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. CONFUSION MATRIX RESULTS FOR BASELINE SCENARIO WITHOUT FEATURE EXTRACTION

The lowest horizontal row of the confusion matrix indicates the percentage of the respective outcome class which was correctly classified for the test dataset. The rightmost vertical column indicates the probability that a given output of an outcome type is correct (True Positive). The diagonal of the matrix indicates the proportion of the respective outcome type in the full testing set. The cell in the rightmost column and the bottom row indicates the overall PCC regardless of outcome type.

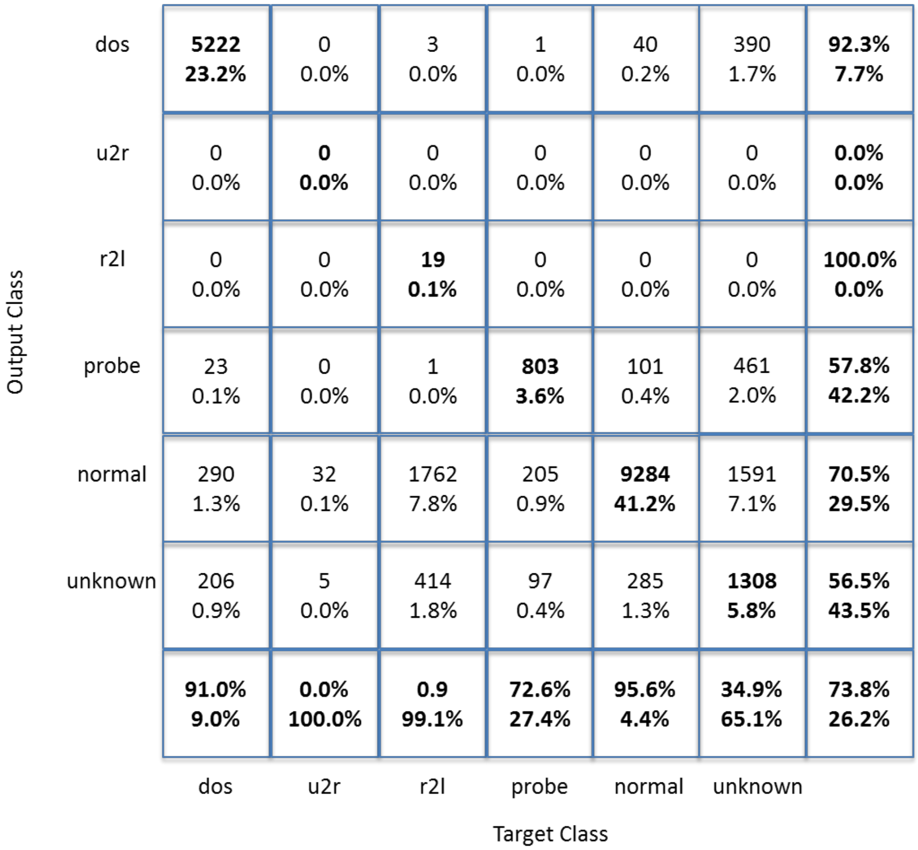


Output Class

	dos	u2r	r2l	probe	normal	unknown	
dos	**5269** **23.4%**	0 0.0%	0 0.0%	1 0.0%	44 0.2%	364 1.6%	**92.8%** **7.2%**
u2r	0 0.0%	**0** **0.0%**	0 0.0%	0 0.0%	0 0.0%	0 0.0%	**0.0%** **0.0%**
r2l	0 0.0%	0 0.0%	**1** **0.0%**	0 0.0%	0 0.0%	1 0.0%	**50.0%** **50.0%**
probe	28 0.1%	0 0.0%	6 0.0%	**807** **3.6%**	134 0.6%	619 2.7%	**50.6%** **49.4%**
normal	279 1.2%	33 0.1%	1783 7.9%	260 1.2%	**9299** **41.3%**	1736 7.7%	**69.4%** **30.6%**
unknown	165 0.7%	4 0.0%	409 1.8%	38 0.2%	233 1.0%	**1030** **4.6%**	**54.8%** **45.2%**
	91.5% **8.2%**	**0.0%** **100.0%**	**0.0%** **100.0%**	**73.0%** **27.0%**	**95.8%** **4.2%**	**27.5%** **72.5%**	**72.8%** **27.2%**

Target Class

Figure 12. Confusion matrix for single NN – test dataset.

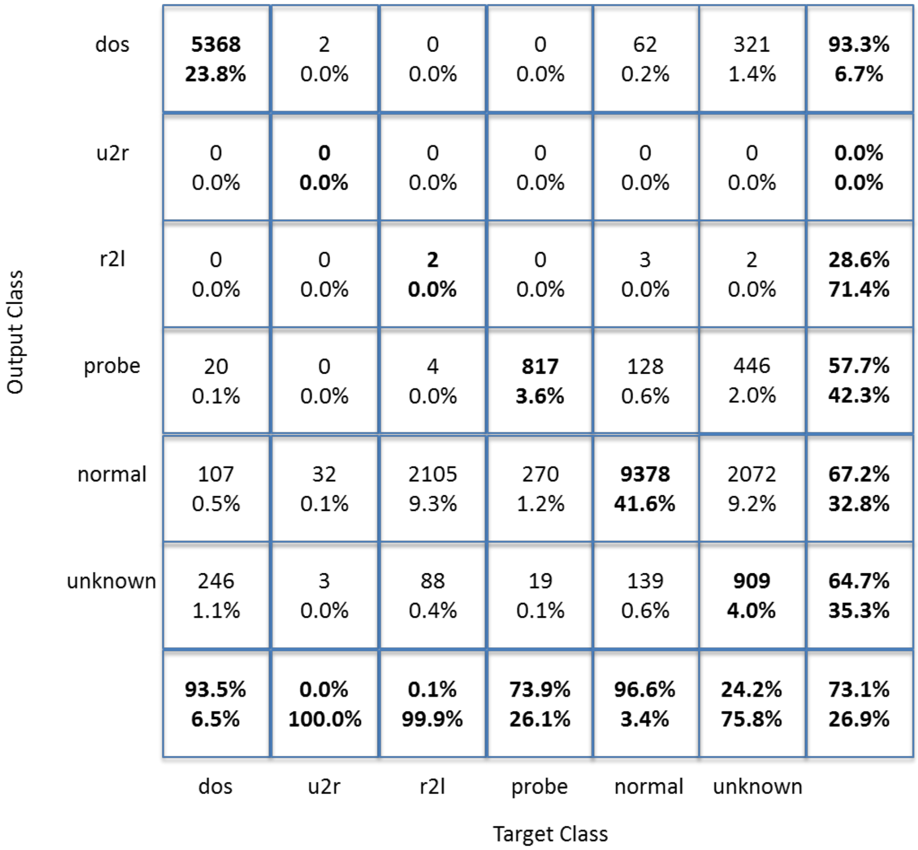


Output Class

dos	**5222 23.2%**	0 0.0%	3 0.0%	1 0.0%	40 0.2%	390 1.7%	**92.3% 7.7%**
u2r	0 0.0%	**0 0.0%**	0 0.0%	0 0.0%	0 0.0%	0 0.0%	**0.0% 0.0%**
r2l	0 0.0%	0 0.0%	**19 0.1%**	0 0.0%	0 0.0%	0 0.0%	**100.0% 0.0%**
probe	23 0.1%	0 0.0%	1 0.0%	**803 3.6%**	101 0.4%	461 2.0%	**57.8% 42.2%**
normal	290 1.3%	32 0.1%	1762 7.8%	205 0.9%	**9284 41.2%**	1591 7.1%	**70.5% 29.5%**
unknown	206 0.9%	5 0.0%	414 1.8%	97 0.4%	285 1.3%	**1308 5.8%**	**56.5% 43.5%**
	91.0% 9.0%	**0.0% 100.0%**	**0.9 99.1%**	**72.6% 27.4%**	**95.6% 4.4%**	**34.9% 65.1%**	**73.8% 26.2%**
	dos	u2r	r2l	probe	normal	unknown	

Target Class

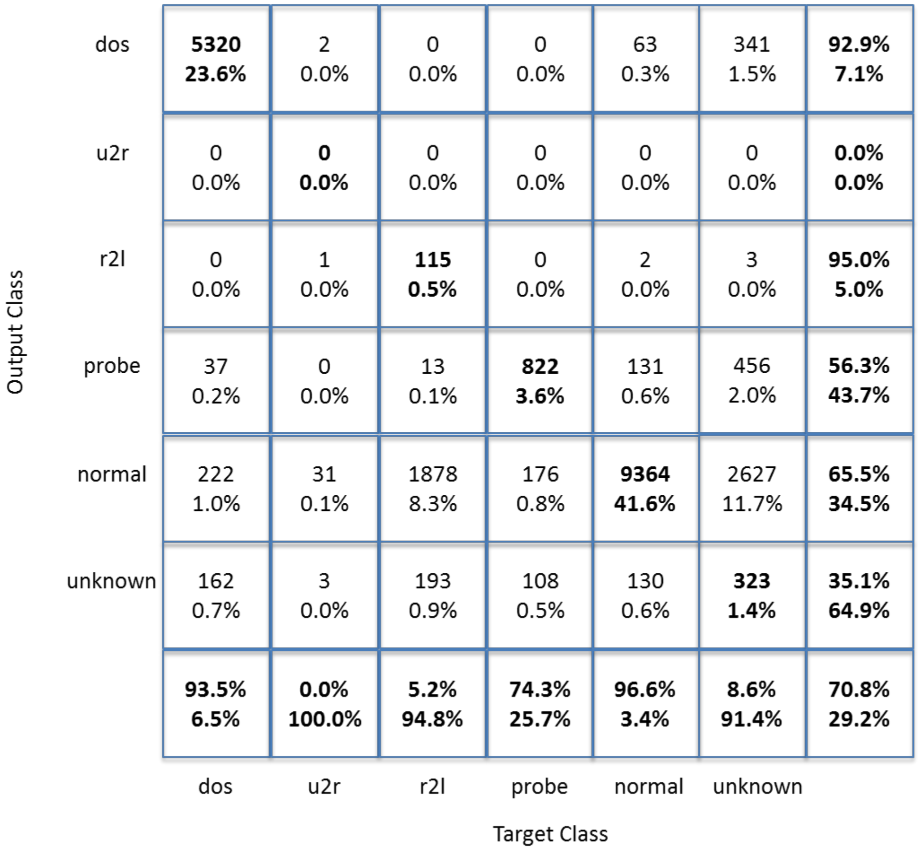
Figure 13. Confusion matrix for sub NN #1 – test dataset.



Output Class	dos	u2r	r2l	probe	normal	unknown	
dos	**5368** **23.8%**	2 0.0%	0 0.0%	0 0.0%	62 0.2%	321 1.4%	**93.3%** **6.7%**
u2r	0 0.0%	**0** **0.0%**	0 0.0%	0 0.0%	0 0.0%	0 0.0%	**0.0%** **0.0%**
r2l	0 0.0%	0 0.0%	**2** **0.0%**	0 0.0%	3 0.0%	2 0.0%	**28.6%** **71.4%**
probe	20 0.1%	0 0.0%	4 0.0%	**817** **3.6%**	128 0.6%	446 2.0%	**57.7%** **42.3%**
normal	107 0.5%	32 0.1%	2105 9.3%	270 1.2%	**9378** **41.6%**	2072 9.2%	**67.2%** **32.8%**
unknown	246 1.1%	3 0.0%	88 0.4%	19 0.1%	139 0.6%	**909** **4.0%**	**64.7%** **35.3%**
	93.5% **6.5%**	**0.0%** **100.0%**	**0.1%** **99.9%**	**73.9%** **26.1%**	**96.6%** **3.4%**	**24.2%** **75.8%**	**73.1%** **26.9%**

Target Class

Figure 14. Confusion matrix for sub NN #2 – test dataset.

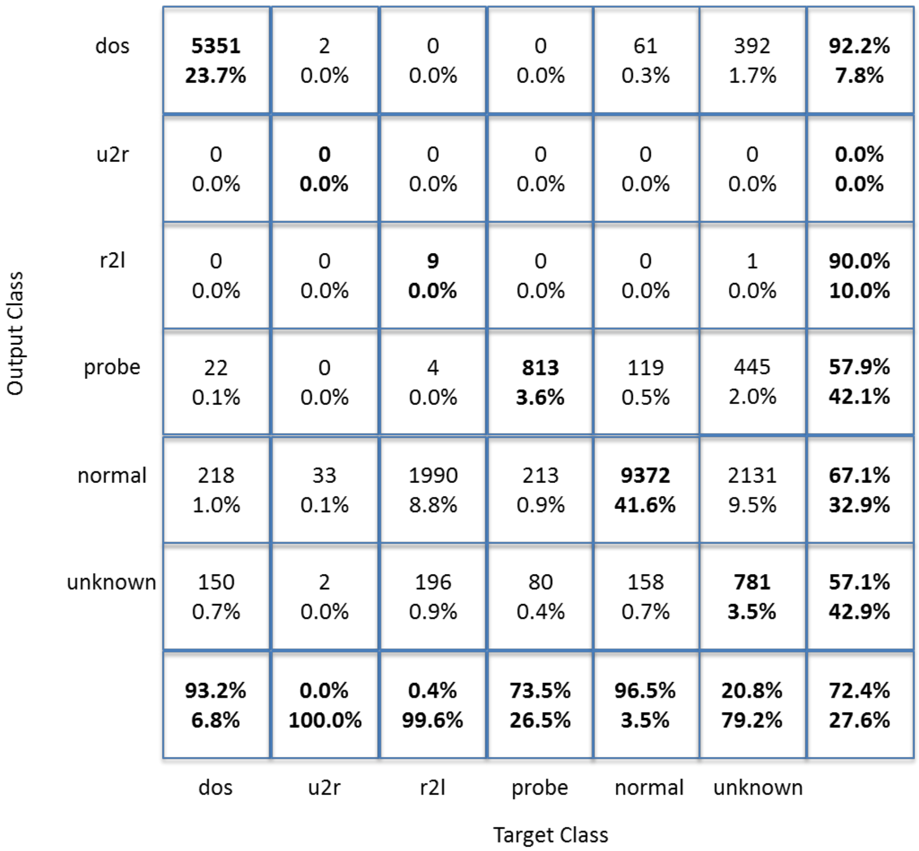


Output Class

dos	**5320** **23.6%**	2 0.0%	0 0.0%	0 0.0%	63 0.3%	341 1.5%	**92.9%** **7.1%**
u2r	0 0.0%	**0** **0.0%**	0 0.0%	0 0.0%	0 0.0%	0 0.0%	**0.0%** **0.0%**
r2l	0 0.0%	1 0.0%	**115** **0.5%**	0 0.0%	2 0.0%	3 0.0%	**95.0%** **5.0%**
probe	37 0.2%	0 0.0%	13 0.1%	**822** **3.6%**	131 0.6%	456 2.0%	**56.3%** **43.7%**
normal	222 1.0%	31 0.1%	1878 8.3%	176 0.8%	**9364** **41.6%**	2627 11.7%	**65.5%** **34.5%**
unknown	162 0.7%	3 0.0%	193 0.9%	108 0.5%	130 0.6%	**323** **1.4%**	**35.1%** **64.9%**
	93.5% **6.5%**	**0.0%** **100.0%**	**5.2%** **94.8%**	**74.3%** **25.7%**	**96.6%** **3.4%**	**8.6%** **91.4%**	**70.8%** **29.2%**
	dos	u2r	r2l	probe	normal	unknown	

Target Class

Figure 15. Confusion matrix for sub NN #3 – test dataset.



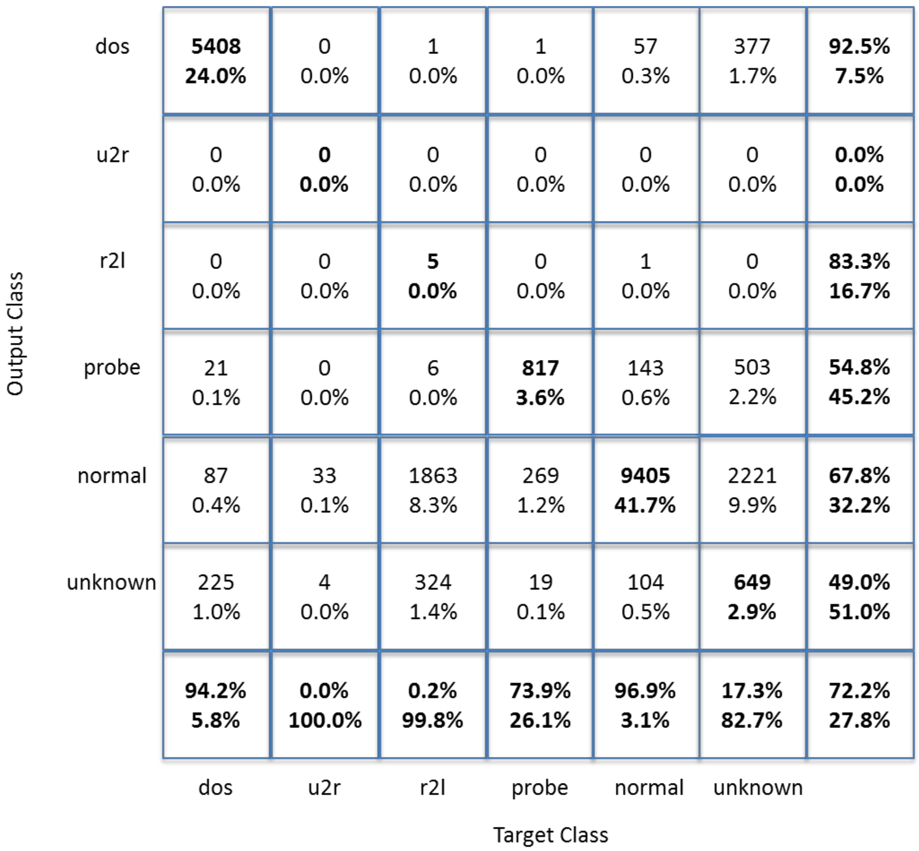
Output Class \ Target Class	dos	u2r	r2l	probe	normal	unknown	
dos	**5351** **23.7%**	2 0.0%	0 0.0%	0 0.0%	61 0.3%	392 1.7%	**92.2%** **7.8%**
u2r	0 0.0%	**0** **0.0%**	0 0.0%	0 0.0%	0 0.0%	0 0.0%	**0.0%** **0.0%**
r2l	0 0.0%	0 0.0%	**9** **0.0%**	0 0.0%	0 0.0%	1 0.0%	**90.0%** **10.0%**
probe	22 0.1%	0 0.0%	4 0.0%	**813** **3.6%**	119 0.5%	445 2.0%	**57.9%** **42.1%**
normal	218 1.0%	33 0.1%	1990 8.8%	213 0.9%	**9372** **41.6%**	2131 9.5%	**67.1%** **32.9%**
unknown	150 0.7%	2 0.0%	196 0.9%	80 0.4%	158 0.7%	**781** **3.5%**	**57.1%** **42.9%**
	93.2% **6.8%**	**0.0%** **100.0%**	**0.4%** **99.6%**	**73.5%** **26.5%**	**96.5%** **3.5%**	**20.8%** **79.2%**	**72.4%** **27.6%**

Figure 16. Confusion matrix for merged output of three-parallel NN – test dataset.

THIS PAGE INTENTIONALLY LEFT BLANK

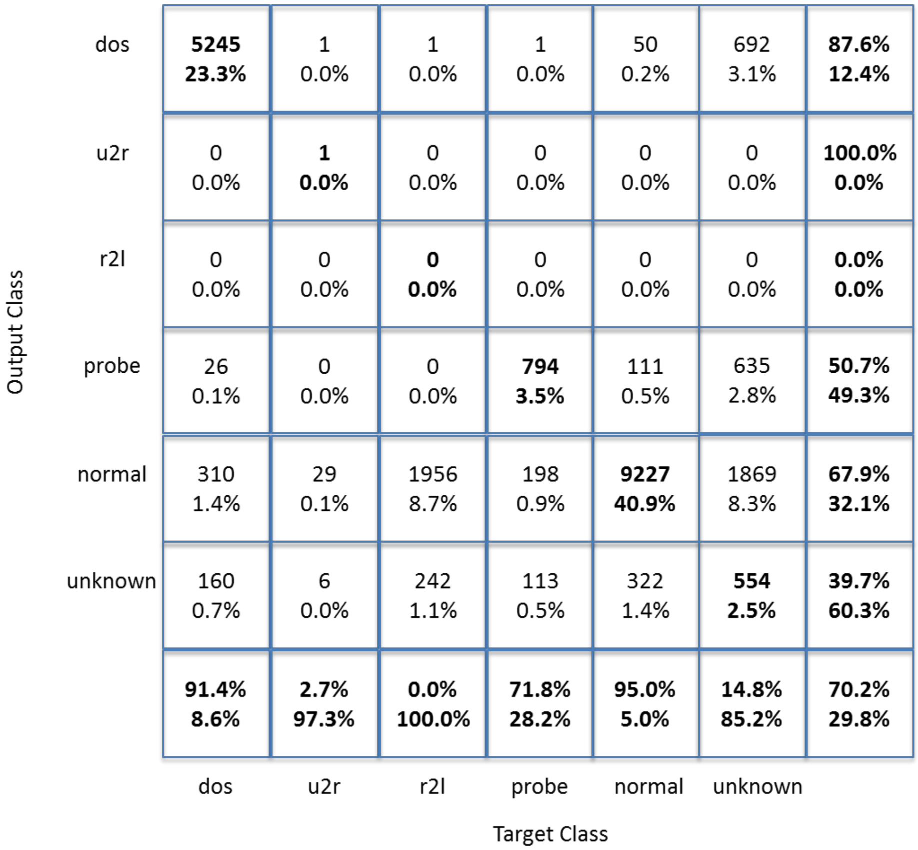
APPENDIX F. CONFUSION MATRIX RESULTS FOR SCENARIO UTILIZING FEATURE EXTRACTION

The lowest horizontal row of the confusion matrix indicates the percentage of the respective outcome class which was correctly classified for the test dataset. The rightmost vertical column indicates the probability that a given output of an outcome type is correct (True Positive). The diagonal of the matrix indicates the proportion of the respective outcome type in the full testing set. The cell in the rightmost column and the bottom row indicates the overall PCC regardless of outcome type.



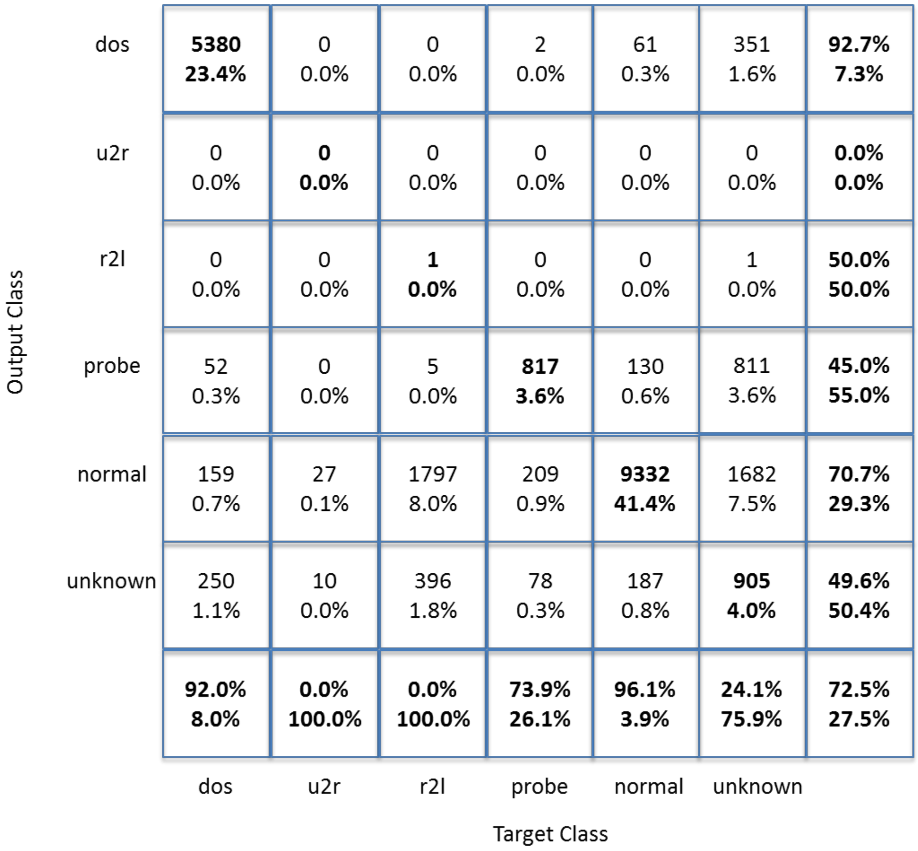
Output Class \ Target Class	dos	u2r	r2l	probe	normal	unknown	
dos	**5408** **24.0%**	0 0.0%	1 0.0%	1 0.0%	57 0.3%	377 1.7%	**92.5%** **7.5%**
u2r	0 0.0%	**0** **0.0%**	0 0.0%	0 0.0%	0 0.0%	0 0.0%	**0.0%** **0.0%**
r2l	0 0.0%	0 0.0%	**5** **0.0%**	0 0.0%	1 0.0%	0 0.0%	**83.3%** **16.7%**
probe	21 0.1%	0 0.0%	6 0.0%	**817** **3.6%**	143 0.6%	503 2.2%	**54.8%** **45.2%**
normal	87 0.4%	33 0.1%	1863 8.3%	269 1.2%	**9405** **41.7%**	2221 9.9%	**67.8%** **32.2%**
unknown	225 1.0%	4 0.0%	324 1.4%	19 0.1%	104 0.5%	**649** **2.9%**	**49.0%** **51.0%**
	94.2% **5.8%**	**0.0%** **100.0%**	**0.2%** **99.8%**	**73.9%** **26.1%**	**96.9%** **3.1%**	**17.3%** **82.7%**	**72.2%** **27.8%**

Figure 17. Confusion matrix for single NN – test dataset.



Output Class \ Target Class	dos	u2r	r2l	probe	normal	unknown	
dos	**5245** **23.3%**	1 0.0%	1 0.0%	1 0.0%	50 0.2%	692 3.1%	**87.6%** **12.4%**
u2r	0 0.0%	**1** **0.0%**	0 0.0%	0 0.0%	0 0.0%	0 0.0%	**100.0%** **0.0%**
r2l	0 0.0%	0 0.0%	**0** **0.0%**	0 0.0%	0 0.0%	0 0.0%	**0.0%** **0.0%**
probe	26 0.1%	0 0.0%	0 0.0%	**794** **3.5%**	111 0.5%	635 2.8%	**50.7%** **49.3%**
normal	310 1.4%	29 0.1%	1956 8.7%	198 0.9%	**9227** **40.9%**	1869 8.3%	**67.9%** **32.1%**
unknown	160 0.7%	6 0.0%	242 1.1%	113 0.5%	322 1.4%	**554** **2.5%**	**39.7%** **60.3%**
	91.4% **8.6%**	**2.7%** **97.3%**	**0.0%** **100.0%**	**71.8%** **28.2%**	**95.0%** **5.0%**	**14.8%** **85.2%**	**70.2%** **29.8%**

Figure 18. Confusion matrix for sub NN #1 – test dataset.

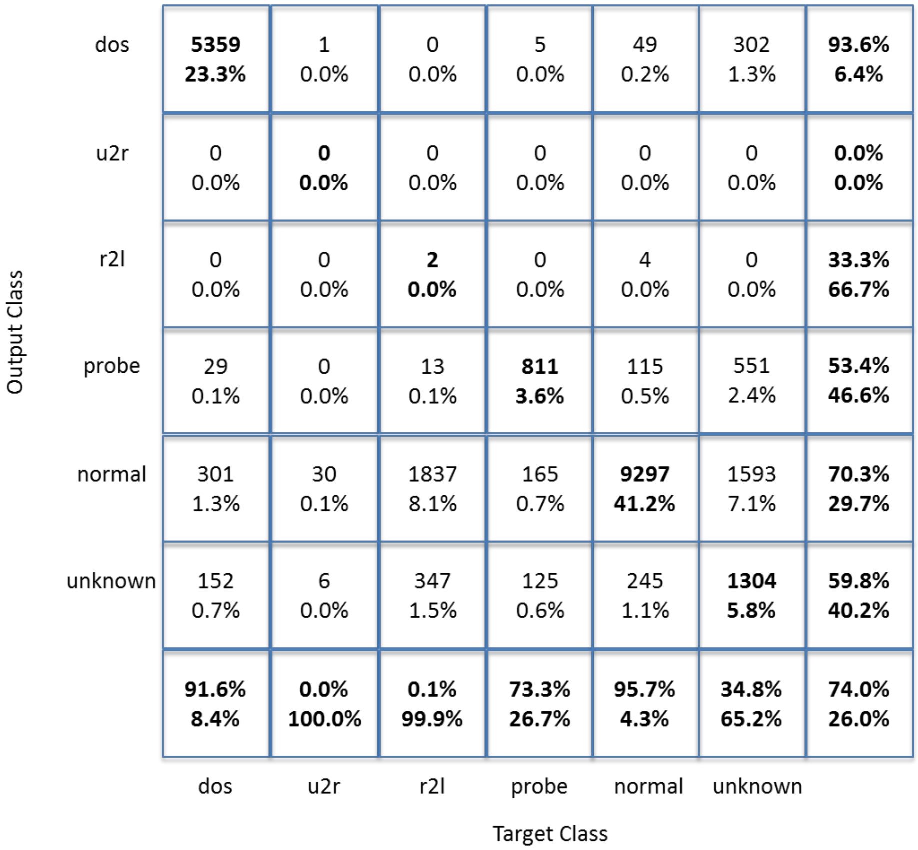


Output Class

dos	**5380** **23.4%**	0 0.0%	0 0.0%	2 0.0%	61 0.3%	351 1.6%	**92.7%** **7.3%**
u2r	0 0.0%	**0** **0.0%**	0 0.0%	0 0.0%	0 0.0%	0 0.0%	**0.0%** **0.0%**
r2l	0 0.0%	0 0.0%	**1** **0.0%**	0 0.0%	0 0.0%	1 0.0%	**50.0%** **50.0%**
probe	52 0.3%	0 0.0%	5 0.0%	**817** **3.6%**	130 0.6%	811 3.6%	**45.0%** **55.0%**
normal	159 0.7%	27 0.1%	1797 8.0%	209 0.9%	**9332** **41.4%**	1682 7.5%	**70.7%** **29.3%**
unknown	250 1.1%	10 0.0%	396 1.8%	78 0.3%	187 0.8%	**905** **4.0%**	**49.6%** **50.4%**
	92.0% **8.0%**	**0.0%** **100.0%**	**0.0%** **100.0%**	**73.9%** **26.1%**	**96.1%** **3.9%**	**24.1%** **75.9%**	**72.5%** **27.5%**
	dos	u2r	r2l	probe	normal	unknown	

Target Class

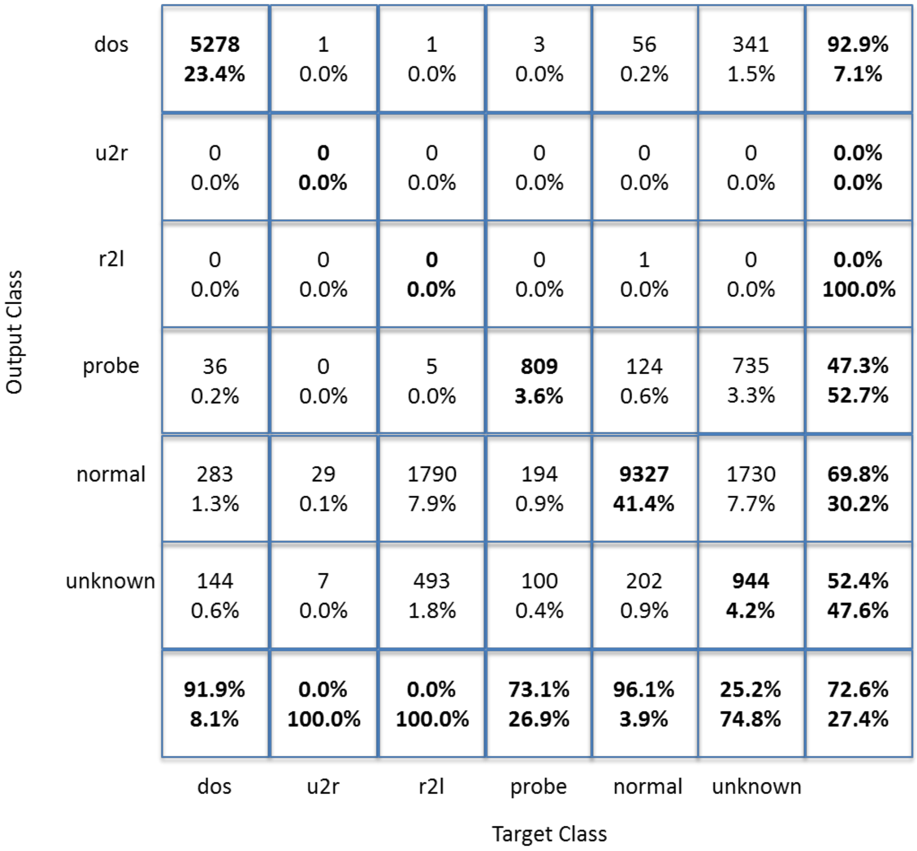
Figure 19. Confusion matrix for sub NN #2 – test dataset.



Output Class	dos	u2r	r2l	probe	normal	unknown	
dos	**5359** **23.3%**	1 0.0%	0 0.0%	5 0.0%	49 0.2%	302 1.3%	**93.6%** **6.4%**
u2r	0 0.0%	**0** **0.0%**	0 0.0%	0 0.0%	0 0.0%	0 0.0%	**0.0%** **0.0%**
r2l	0 0.0%	0 0.0%	**2** **0.0%**	0 0.0%	4 0.0%	0 0.0%	**33.3%** **66.7%**
probe	29 0.1%	0 0.0%	13 0.1%	**811** **3.6%**	115 0.5%	551 2.4%	**53.4%** **46.6%**
normal	301 1.3%	30 0.1%	1837 8.1%	165 0.7%	**9297** **41.2%**	1593 7.1%	**70.3%** **29.7%**
unknown	152 0.7%	6 0.0%	347 1.5%	125 0.6%	245 1.1%	**1304** **5.8%**	**59.8%** **40.2%**
	91.6% **8.4%**	**0.0%** **100.0%**	**0.1%** **99.9%**	**73.3%** **26.7%**	**95.7%** **4.3%**	**34.8%** **65.2%**	**74.0%** **26.0%**

Target Class

Figure 20. Confusion matrix for sub NN #3 – test dataset.



Output Class	dos	u2r	r2l	probe	normal	unknown	
dos	**5278** **23.4%**	1 0.0%	1 0.0%	3 0.0%	56 0.2%	341 1.5%	**92.9%** **7.1%**
u2r	0 0.0%	**0** **0.0%**	0 0.0%	0 0.0%	0 0.0%	0 0.0%	**0.0%** **0.0%**
r2l	0 0.0%	0 0.0%	**0** **0.0%**	0 0.0%	1 0.0%	0 0.0%	**0.0%** **100.0%**
probe	36 0.2%	0 0.0%	5 0.0%	**809** **3.6%**	124 0.6%	735 3.3%	**47.3%** **52.7%**
normal	283 1.3%	29 0.1%	1790 7.9%	194 0.9%	**9327** **41.4%**	1730 7.7%	**69.8%** **30.2%**
unknown	144 0.6%	7 0.0%	493 1.8%	100 0.4%	202 0.9%	**944** **4.2%**	**52.4%** **47.6%**
	91.9% **8.1%**	**0.0%** **100.0%**	**0.0%** **100.0%**	**73.1%** **26.9%**	**96.1%** **3.9%**	**25.2%** **74.8%**	**72.6%** **27.4%**

Target Class

Figure 21. Confusion matrix for merged output of three-parallel NN – test dataset.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] I. Ahmad, A. B. Abdullah, and A. S. Alghamdi, "Remote to local attack detection using supervised neural network," in *Int. Conf. for Internet Technol. and Secured Trans. (ICITST)*, 2010, pp. 1–6.

[2] S. Northcutt and J. Novak, *Network Intrusion Detection: An Analyst's Handbook*, 3rd ed. Indianapolis, IN: New Riders, 2002, pp. 3–21.

[3] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Computational Intell. for Security and Defense Appl.*, 2009, pp. 1–6.

[4] K. El-Khatib, "Impact of feature reduction on the efficiency of wireless intrusion detection systems," *IEEE Trans. Parallel and Distributed Syst.*, vol. 21, no. 8, pp. 1143–1149, 2010.

[5] L. Xie and J. Li, "A novel feature extraction method assembled with PCA and ICA for Network Intrusion Detection," in *Int. Forum Comput. Sci.-Technol. and Appl.*, 2009, pp. 31–34.

[6] I. K. Gbashi, S. H. Hashem, and S. K. Majeed, "Proposed vision for Network Intrusion Detection System using Latent Semantic Analysis and data mining," in *6th Comput. Sci. and Electron. Eng. Conf.*, 2014, pp. 11–16.

[7] Z. Yongli, Z. Yungui, T. Weiming, and C. Hongzhi, "An improved feature selection algorithm based on MAHALANOBIS distance for Network Intrusion Detection," in *Int. Conf. on Sensor Network Security Technol. and Privacy Commun. System*, 2013, pp. 69–73.

[8] F. Zhang and D. Wang, "An effective feature selection approach for Network Intrusion Detection," in *IEEE Eighth Int. Conf. Networking, Architecture and Storage*, 2013, pp. 307–311.

[9] Y. S. Chen, G. Hui, Y. G. Xian, J. X. Ling, Z. L. Nang, and S. T. Jun, "The Solution to How to Select an Optimal Set of Features from Many Features Used to Intrusion Detection System in Wireless Sensor Network," Second WRI Global Congress, 2010, pp. 368–371

[10] G. Sandhya, A. Julian, "Intrusion detection in wireless sensor network using genetic K-means algorithm," in *Int. Conf. Advanced Commun. Control and Computing Technol.*, 2014, pp. 1791–1794.

[11] Zhiyuan Tan, A. Jamdagni, Xiangjian He, P. Nanda and Ren Ping Liu, “Triangle-area-based multivariate correlation analysis for effective denial-of-service attack detection,” in *IEEE 11th Int. Conf. on Trust, Security and Privacy in Computing and Commun. (TrustCom)*, 2012, pp. 33–40.

[12] Zhiyuan Tan, A. Jamdagni, Xiangjian He, P. Nanda and Ren Ping Liu, “A system for denial-of-service attack detection based on multivariate correlation analysis,” *Parallel and Distributed Syst., IEEE Trans.*, vol. 25, pp. 447–456, Feb. 2014.

[13] A. H. S. Aborujilah, S. Musa, A. Shahzad, M. Nazri, and A. Alsharafi, “Flooding based DoS attack feature selection using remove correlated features algorithm,” in *Int. Conf. Advanced Comput. Sci. Appl. and Technol*. 2013, pp. 93–96.

[14] F. Lydia Catherine, R. Pathak and V. Vaidehi, “Efficient host based intrusion detection system using partial decision tree and correlation feature selection algorithm,” in *Int. Conf. Recent Trends Inform. Technol. (ICRTIT)*, 2014, pp. 1–6.

[15] Q. Guangzhi, S. Hariri, M. Yousif, “A new dependency and correlation analysis for features,” *IEEE Trans. Knowledge and Data Eng*., vol. 17, no. 9, pp. 1199–1207, 2005.

[16] S. Mukkamala, G. Janoski, and A. Sung, “Intrusion detection using neural networks and support vector machines,” in *Proc. Int. Joint Conf. Neural Networks*, pp. 1702–1707, 2002.

[17] S. Haykin, *Neural Networks: A Comprehensive Foundation*. 2nd ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2005, pp. 23–59.

[18] M. A. Hogo, “Temporal analysis of intrusion detection,” in *Int. Carnahan Conf. on Security Technol.*, 2014, pp. 1–6.

[19] The NSL KDD dataset. (n.d.). GitHub. [Online]. Available: https://github.com/defcom17/NSL_KDD.git. Accessed Sep. 14, 2015.

[20] KDD Cup ‘99 dataset (Network Intrusion) considered harmful. KDnuggets. [Online]. Available: http://www.kdnuggets.com/news/2007/n18/4i.html. Accessed Aug. 16, 2015.

[21] Neural Network Toolbox. (n.d.). Mathworks. [Online]. Available: http://www.mathworks.com/products/neural-network/features.html#training-algorithms. Accessed Sep. 14, 2015.

[22] Estimating the number of neurons and number of layers of an artificial neural network. (n.d.). StackOverflow. [Online]. Available: http://stackoverflow.com/questions/3345079/estimating-the-number-of-neurons-and-number-of-layers-of-an-artificial-neural-ne. Accessed Aug. 16, 2015.

[23] Pace, P. E., *Detecting and Classifying Low Probability of Intercept Radar*. 2nd ed. Norwood, MA: Artech House, 2009, pp. 624–633.

[24] A. Tartakovsky, I. Nikiforov, and M. Basseville. *Sequential Analysis: Hypothesis Testing and Changepoint Detection*, 1st ed. Boca Raton, FL: CRC Press, 2014, pp. 534–546.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California