Deep Learning for Flow-Based Detection and Classification of Encrypted Botnet Traffic

Lucas Carr*
crrluc003[at]myuct.ac.za
University of Cape Town
South Africa

ABSTRACT

Detection of malicious traffic on a network is critical to ensuring the safety and security of internet systems. Classical approaches to this task increasingly struggle with modern networking procedures, like encryption. Deep learning (DL) offers an alternative approach to traffic classification problems. We address two major problem classes: (1) botnet detection and (2) botnet family classification. For each problem, we explore five implementations of DL architectures: a multi-layer perceptron (MLP), shallow and deep convolutional neural network (CNN v1 and CNN v2), an autoencoder (AE) and an autoencoder + convolutional neural network (AE+CNN). We recognise a lack of surrounding literature which consider the computational requirements as an important aspect of model evaluation. Consequently, our evaluation of models for each respective problem class is based on the classification and computational performance of each model. We further investigate the effect of training the models on an input with a reduced feature space, where we discuss the impact this has in terms of a trade-off between computational and classification performance. For botnet detection, we find that all models attain good (≥ 0.979 accuracy) classification performance on a normal testing set; however, this performance drops fairly substantially when evaluated on a set of unknown botnet families. Furthermore, we observed a clear trend between increased feature space and memory utilisation, while finding no evidence of a trend between inference time and feature space. For botnet classification, we found that models which implement CNN architectures outperform others by a substantial margin (≈ 6 percentage points). We observe the same trend between feature space and memory utilisation, and absence of apparent relationship between feature space and inference time.

CCS CONCEPTS

• **Networks** → **Network monitoring**; • **Computing methodologies** → **Supervised learning by classification**; **Neural networks**;

KEYWORDS

Deep Learning, Machine Learning, Neural Networks, Malware Detection, Malware Classification, Botnets

1 INTRODUCTION

The proliferation of computers and networks as tools essential to modern life has resulted in innumerable benefits. However, adoption of the associated technologies has created new security dynamics to consider; specifically in the form of malicious software, or malware. Malware is an umbrella term that encompasses various types of software designed to infiltrate systems without permission, aiming to cause harm or exploit vulnerabilities, often with a financial motive [15].

While there are numerous sub-categories of malware, we focus attention on botnets, out of recognition that the increasing number of security-vulnerable Internet of Things (IoT) devices offer an ideal landscape for botnets [3]. A botnet defines a distributed network of computers, or *bots*, infected with software that enables the bots to be controlled by a malicious operator, or *botmaster* [1, 17]. A botnet typically leverages one of many additional types of malware - such as a worm - to propagate itself across multiple computers, and can incorporate a centralised or decentralised operating procedure [17]. Moreover, botnets attempt to hide themselves by transmitting normal traffic amongst their botnet traffic [17]. However, a defining characteristic of botnets is the presence of command and control channels, through which the malicious operator is able to transmit instructions or receive information. A common instruction would be a distributed denial-of-service (DDOS) attack, where the bots flood a target to disrupt its service [1, 3]. It is this characteristic of botnets - that the bot must at some point connect to its botmaster - that may be leveraged to build detection models. When a bot connects to the botmaster, a sequence of network flows, defined as a grouping of related traffic, can be extracted from the generated traffic, from which a deep learning (DL) model will be able to learn distinguishing patterns [17]. DL is a field within machine learning which is defined by the use of multi-layered architectures, enabling models to learn complex, hierarchical patterns from data without requiring feature engineering [12, 19].

Preventative measures against malware, and botnets, are not a novel concern; and there are existing approaches to detect and block malicious traffic on networks. These approaches typically deploy a Network Intrusion Detection System, or NIDS [18]. NIDS operate by implementing a broad range of techniques to detect and identify malicious traffic: notably, port analysis, blacklisted IP addresses, and inspecting packet payloads [16, 29]. Recently adopted practices around networking have dampened the effectiveness of these techniques, making NIDS which use them less reliable. Port numbers have become less reliable indicators of application type; additionally, the existence of port-obfuscation enables creators of malware to avoid detection [29]. Similarly, dynamic IP addresses and IP spoofing make systems which filter traffic based on blacklisted IPs unreliable. Finally, approaches which aim to detect malware by inspecting the payload contents of packets flowing through the network face increasing difficulty as more network traffic adopts encryption protocols - a Cisco report from 2017 noted that $\approx 75\%$ of analysed malicious traffic made use of encryption [26].

In recognition of the shortcomings of existing malware detection practices, we define an objective to evaluate the effectiveness of different DL algorithms when tasked with detection and classification of botnet traffic using network flows. More specifically, we implement a series of binary classification models to detect malicious traffic, and secondary of multiclass classification models to classify botnet traffic into respective families. While this approach is not itself novel, much of the existing literature evaluates the effectiveness of DL models in terms of the accuracy, F1-score, False Positive Rate (FPR), and False Negative Rate (FNR) [17, 22, 28]. [1] These metrics provide insight into classification performance; however, we argue that insight into the computational requirements of a model are important. A model which attains good accuracy scores might be impractical due to its computational requirements, especially on smaller, lower-resourced networks. Moreover, models are trained on a datasets made up of only subset of

[1] Definitions for these are found in 6.1 and 9.3

existing botnets, and newer botnets are continuously developed. Evaluating models on an unseen testing set comprised of botnet families present in their training set ignores this concept. As a result is greater uncertainty into a model's ability to generalise to newer botnets.

Many existing machine learning based approaches to malware detection require datasets with hand-selected features, a process which is both time consuming, and requires expert domain knowledge [4]. DL approaches are able to avoid this requirement, as they typically do not require careful feature selection. The consequence of this is that larger feature spaces create models with significantly more trainable parameters, impacting the model's offline performance metrics like inference time and memory utilisation [19].

We propose an alternative, holistic approach to model evaluation which whereby we use the conventional classification performance metrics on a normal testing set, as well as a supplementary testing set of unknown botnet families which aids evaluation of a model's ability to generalise to zero-day attacks. Further, we evaluate the effect that feature space size has on computational performance, in terms of a model's inference time and memory utilisation.

1.1 First Research Objective: Botnet Detection

We implement five binary classification models, an MLP, shallow CNN (v1), deep CNN (v2), AE, and AE+CNN, which serve as models for botnet detection. With respect to each classifier, we aim to:

(1) Evaluate the accuracy, FPR, and FNR on the standard and proto zero-day test set.
(2) Evaluate how reducing the feature space, into 50% and 30% samples, impacts the memory requirements, inference time, accuracy, FPR, and FNR of the models.

1.2 Second Research Objective: Botnet Classification

We implement five multiclass classification models, an MLP, shallow CNN, deep CNN, AE, and AE+CNN, which aim to identify respective families of botnets. With respect to each classifier, we aim to:

(1) Determine the overall accuracy, and accuracy respective to each class, when evaluated on the standard test set.
(2) Evaluate how reducing the feature space, into 50% and 30% samples, effects the memory requirements and inference time, in relation to the overall accuracy of a model.

2 BACKGROUND

2.1 Classification Approaches: Payload vs. Flow Based

Network traffic is made up of discrete blocks of data, called packets, which travel through a network. Approaches to classify network traffic typically make use of training data which captures either the individual packets payload, or network flows. Informally, network flows represent a sequence of packets from end-points on a network - ideally, bi-directional flows, which capture the flow of traffic from the source and destination [29]. Statistical features can be extracted from network flows, which explain metrics such as the rate at which packets flow back and forth, and the mean packet size of the flow [7].

Another approach is to use the core contents of a packet, i.e. the payload, as training data. The notion is that the payload of malicious traffic contains at least part of the malware binary, from which a model would be able to recognise patterns belonging to this binary [11]. Payload based approaches face the difficulty of classifying encrypted traffic - a problem that flow-based analysis avoids, since only the packet headers are required to extract flows from traffic captures, which are not encrypted [12]. There have been implementations of payload-based classifiers which are able to handle encrypted traffic [4, 10, 11]. These approaches typically require thorough processing steps to prepare the data for classification, which makes payload-based approaches ill-suited to real world application. Conversely, aggregated network flows are comparatively easy to extract [18].

2.2 Multilayer Perceptrons

Multilayer Perceptrons (MLPs) were one of the earliest forms of neural networks, and as such, express concepts which are fundamental to the more advanced neural network which are discussed further on. The goal of an MLP model is to, as best as possible, emulate some non-linear function [6]. MLPs offer a relatively simple example of a DL algorithm, and may offer a sense of how difficult a classification problem is.

2.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) have become increasingly popular algorithms for classification [14]. Typically CNNs work with grid-like inputs, such as an image, where a convolutional operation will sweep over the grid, producing a feature map which represents significant areas of the input. These feature maps enable the model to learn identifying spatial patterns in data. Most applications of CNNs use two-dimensional image data, or image-representations of streams of one-dimensional data. However, the fundamental principle of a sequence of convolutional layers that identify increasingly complex patterns in the data holds for inputs which are not grid-like in nature - e.g. a network flow, which is a one dimensional vector [2].

2.4 Autoencoders

Autoencoders (AE) are a type of unsupervised learning algorithm that aim to compress input data into a lower-dimensional "latent" vector. As output, an approximation of the input is reconstructed from the latent vector, through a decoding process [6]. When applying AEs to classification tasks, the decoding process can be replaced with a classification layer (like a softmax or sigmoid layer). The encoding portion of the AE focuses on detecting crucial features in the input data and encoding them into a condensed representation. This process of dimensionality reduction ideally encodes the most significant features which aid the subsequent classification layer.

3 RELATED WORK

With the increasing popularity of DL, numerous applications of malware detection and classification using DL have been proposed. These can usually be categorized as employing either a payload-based, or flow-based approach to the problem. Historically, payload-based approaches to any kind of traffic classification problem faced difficulty when handling encrypted traffic. DL might offer a solution to this, as it is posited that DL algorithms are capable of finding meaningful patterns within the encrypted payloads. The other approach is to use flow statistics of traffic - access to which is not affected by encryption. Flow-based approaches benefit from a comparatively easier pre-processing stage [28]. Importantly, it should be acknowledged that it is not a case of one or the other, and that these approaches can complement each other.

Piskozub et al. [18] propose a multi-phased system to detect malware, classify it into malware type, and further into malware family using bi-directional network flows. The network flows are pruned so as to remove the features relating to IP addresses and port numbers - this is done to prevent the models from learning from features from the specific traffic capture, rather than the behaviour of malware. Their implementation used an AE to create a latent representation of the network flow. This encoding was then fed into a deep neural network which performs a binary classification. The classified malicious traffic was then fed into a type-classifier and family-classifier using the same representation as the binary classifier [18]. The system was evaluated with two approaches; initially a collection of 'clean' datasets containing only malware from one specific family per dataset. This practice of evaluation does not give useful insight into the system's performance, as the dataset is not representative of real-world traffic. Their second evaluation used datasets containing a mix of malicious and benign traffic; under a $1:1$ split of benign and malicious network activity. The binary classification phase attained an F1-score of ≈ 0.830 across the five malware families sampled. For the same ratio of benign and malicious traffic, the type-classifiers attained an aggregated F1-score of ≈ 0.480. Finally, for the malware family classification, an aggregated F1-score of ≈ 9.525. Significantly, the type and family classifiers for worms, viruses, and ransomware performed well with F1-scores above 0.750, while trojans and adware consistently had F1-scores under 0.500 [18]. The paper further acknowledges that the dataset did not contain a balanced representation of malware classes, and families for each class. For instance, there were far more distinct trojan families relative to ransomware, which makes the task of classifying the different trojan families considerably more challenging.

Hadidi et al. [7] evaluate the effectiveness of different approaches to botnet detection. Two of the approaches discussed are signature based, which were not related to the contributions of this paper. The approaches of interest used Support-Vector Machines (SVM), K-Nearest Neighbour (KNN), and Bayesian Networks (BN) to classifying botnet traffic based of either payloads or network flows, using Detection Rate (DR) and False-Positive Rate (FPR) as evaluation metrics. [2] [3] Simulated network traffic was captured in a sandbox environment. Non-encrypted traffic was used, so as to make their payload-classifier able to handle the traffic captures. Notably, in the network flow preprocessing phases, identifying features such as IP addresses and port numbers were removed from the datasets [7]. In the majority of evaluations, payload-based classifiers have been shown to be superior to flow-based methods. Specifically, the payload-based models such as KNN, SVM, and BN have recorded detection rates (DRs) of 1, 0.995, and 0.938 respectively. In contrast, the flow-based models have posted comparatively lower scores, with DRs of 0.968, 0.910, and 0.838. This trend of better performance by payload-based classifiers is also evident in terms of false positive rates (FPRs). In particular, the payload-based KNN and SVM have an excellent FPR of 0, and BN has a rate of 0.112. However, the flow-based methods have reported higher FPRs of 0.035, 0.073, and 0.09 respectively.

Yeo et al. [28] evaluate four different ML architectures (Random Forest, CNN, MLP, and SVM) to be used as binary classifiers for botnet detection. The models were trained on bi-directional network flows extracted from PCAPs in the CTU-13 dataset - a dataset containing botnet traffic from 7 different families. Typical measurements of accuracy, precision and recall were used as evaluation metrics, while the performance of a classifier was evaluated w.r.t an individual botnet family. The performance trend of the four algorithms was consistent across the botnet families: RF performed the best with accuracy, precision, and recall above .93. The CNN maintained results above 0.85 across all metrics for every botnet family, while the MLP and SVM were as low as 0.55.

[2]Detection Rate is identical to Recall

[3]DR and FPR defined as $\frac{TP}{TP+FN}$, $\frac{FP}{FP+TN}$, respectively.

Pektas & Acarman [17] proposed using a deep neural network (DNN) as a binary classifier for botnet detection. They employed the CTU-13 dataset for botnet captures, which was also used in Yeo et al. [28]. To process the data, they constructed a graph representation of the network captures where nodes represent connected hosts. This approach allowed them to extract statistical information about network flows. Alongside source and destination IP addresses and port numbers, they computed five statistical metrics for each flow: mean, median, maximum, minimum, standard deviation. These metrics were applied to the duration, byte size, number of packets and periodicity of each flow. For evaluation, they focused on accuracy, precision, recall, and the F1-score. Initial experimentation assessed the performance of various neural network architectures: three models with two hidden layers sized at (100, 100), (500, 500), and (1000, 1000), and three models with three hidden layers sized at (100, 100, 100), (500, 500, 500), and (1000, 1000, 1000). Interestingly, increased model complexity did not guarantee better performance. The top-performing model, with an F1-score of 0.991, was the two-layer model sized at (100, 100). Further experimentation aimed to asses how excluding certain features from the flows would impact performance; trials were run where one of the five statistic features were removed (e.g. all the 'median' features were removed). The strongest decreases in performance occurred with the removal of the 'standard' and 'maximum' features from flows, indicating that these are likely more important features for the classification task.

Deep Packet, an approach proposed by Lotfollahi et al. [11], is a system which incorporates both feature extraction and classification stages. This approach is not directly related to malware detection or classification, and instead aims to identify, from encrypted traffic, major classes (e.g., P2P traffic), as well as application identification (e.g., Skype). While this is a significant divergence from the aims of our paper, the approach to solving their problem using DL has strong parallels to ours. They propose a five-layered Stacked Auto Encoder (SAE) connected to a softmax layer, and 1D-CNN as classifiers made up of two convolutional layers and a softmax layer [11]. Following convention, Precision, Recall and F1-score were the chosen evaluation metrics. For the task of classification of traffic into major-classes, the 1D-CNN performed the best with a weighted F1-score of 0.930 across all traffic types, relative to the 0.920 achieved by the SAE. The 1D-CNN outperformed the SAE at identifying traffic application type by a more significant margin, with F1-scores of 0.980 and 0.950, respectively.

Marín et al. [12] flow-based and packet-based approaches to detect (a binary classification) and further classify (a multiclass classification) botnet traffic. These approaches implement a 1D CNN which is connected to an LSTM. For the binary classification task, the flow-based approach is the best model by a significant margin, with an accuracy of 0.986, compared to the packet-based model's 0.776. They note the flow-based model is able to achieve this accuracy with a FPR of ≈ 0.025. For the multiclass classification, they were unable to use a flow-based approach due to limitations relating to their dataset. Bearing this in mind, their packet-based model, which aimed to classify traffic into classes of Benign, Neris, Rbot, and Virut, attained accuracies of 0.878, 0.635, 0.999, and 0.547 for each respective class, with an overall accuracy of 0.765.

4 DATASETS AND PREPROCESSING

Detection and classification of botnet traffic using DL algorithms is a task which lends itself towards a supervised learning approach. Supervised learning is a process which requires the use of high quality, labelled datasets with sufficient samples for the training and evaluation process - the existence of such datasets is rare [26].

Network traffic is typically captured through programs like WireShark, where the information is stored in PCAP files. For the task at hand, network flows, and ideally bidirectional network flows, are extracted from these PCAP files, and preprocessed into suitable training data. We developed a preprocessing pipeline which received traffic captures in the form of PCAP files as input, and after a series of steps, outputted datasets in the form of .csv files. Information concerning the original source of the dataset is discussed in 4.1, after which 4.2 describes the preprocessing steps taken in the pipeline.

4.1 Dataset

The Stratosphere Research Laboratory host an online repository [4] of malicious and normal network captures. Specifically, they have created the CTU-13 dataset, which contains network captures of real traffic from seven distinct botnet families [5]. Internally, the dataset is made up of thirteen captures; each capture containing the malware binary, extracted network flows, and a PCAP file containing only botnet traffic from that scenario's capture (these PCAP files have had their normal and background traffic removed due to privacy considerations) [20].

The bidirectional network flows provided by the CTU-13 dataset, which were extracted using the open source tool, openArgus, offer comparatively limited information, relative to what could be extracted when using CICFlowMeter [8]. [5] Consequently, only the PCAP files containing botnet traffic were used from this dataset. These were then supplemented with the Stratosphere Research Laboratory's repository of normal captures, which are captures of network activity which imitate a typical user's activity on a network, and are restricted to contain only benign network activity. The inclusion of benign traffic was necessary in order to facilitate the measurement of True Negatives, and False Positives, as well as encourage models to be able to generalise to a real-world environment [21].

Captures 10 and 11 were omitted from the CTU-13 collection, as they were instances of a malware family which had sufficient representation from the remaining scenarios. The resultant eleven scenarios were supplemented with five 'normal' captures.

For the binary classification process, botnet families Murlo and NSISay were completely excluded from the training and testing sets. This was to enable the creation of an additional testing set, hereafter referred to as the 'proto zero-day' set, which contained botnet families which to which the model had not been exposed. Unlike traditional test sets, which present models with unseen instances of known botnet families, our set introduces entirely new categories. This additional measure is analogous to concept of zero-day attacks, which are malware attacks which have never appeared before [30]. While measuring a model's exact ability to detect zero-day attacks would be impossible, we argue that this approach reasonable indication of the model's performance when encountering previously unseen attacks.

4.2 The Pipeline

Illustrated in Figure 1, the preprocessing pipeline begins with a collection of PCAP files representing traffic captures. These were the eleven botnet captures and five normal captures. Individually, each PCAP represents a capture of either entirely normal traffic, or a single botnet family [5]. The bi-directional network flows were extracted using CICFlowMeter [8]. The extracted flows from each PCAP file would be stored in a corresponding .csv file.

[4]The repositories can be found at: www.stratosphereips.org/datasets-overview
[5]openArgus can be found at: https://openargus.org

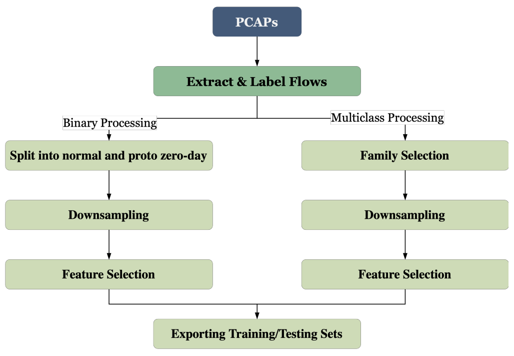

Figure 1: Illustration of the preprocessing pipeline, showing the divergence in processing steps for the binary and multiclass classification datasets

4.2.1 Flow Extraction and Labelling

An essential part of the preprocessing was to assign accurate labels to our data. The CTU-13 dataset includes labelled bi-directional network flows for each scenario, extracted with the openArgus. However, these flow are not as detailed as flows extracted using an alternative tool: CICFlowMeter [8]. A consequence is that flows extracted using CICFlowMeter needed to be labelled manually. The nature of the sourced PCAP files was that they contained either entirely malicious or entirely benign traffic [20]. As such, the labelling process was straightforward to implement: the labels corresponding to the flows generated from the previous stage could be identified by knowing which PCAP file the flows originated from - which was simple to do, given that each PCAP file would produce a single .csv of its extracted flows. The labelling process was automated through a python script, *FlowLabeller.py*.

For the binary classification dataset, the data was either labelled as 0 indicating benign, or 1 meaning malicious. For the multi-class classification, benign traffic was labelled 0, and the malware classes were labelled from 1-7.

4.2.2 Feature Selection

The bidirectional flows are one-dimensional vectors, made up of 82 features. Each feature is a specific measurement of how the data behaves in the flow, from which patterns can be learned during the training process. For example, there is a feature (Total Fwd Pkt) which provides the total number of forward flowing packets in the bi-directional flow.[6] However, we recognised that allowing certain features to persist in the dataset could potentially be detrimental to the models' classification performance.

Features relating to IP addresses (Src IP, Dst IP), and port numbers (Src Port, Dst Port) were removed. The reasons for this were two-fold: dynamic IP address, port-obfuscation and IP spoofing are techniques which make relying on these features for classification a poor idea [7, 18]. Additionally, the models should be able to generalise to unseen data as best as possible. The inclusion of features such as IP addresses and port numbers in the training set is antithetical to this goal, since they are not intrinsic to the identity of the malicious traffic. The 'Flow ID', 'Protocol', and 'Timestamp' features were also removed for this reason. The remaining features are harder to spoof, and associated with the actual behaviour of the botnet malware [18].

The result being that each bi-directional flow is represented as a one-dimensional vector with 75 features. We then create two additional datasets which contained a random sample of 50% and 30% of the features (37 and 22 features, respectively). This would facilitate

[6]The exhaustive list of features is found in 9.4

investigation into how reducing the feature space might lower memory requirements and inference time, and what effect it would have on classification performance.[7]

4.2.3 Balancing

The datasets for the binary classification task were balanced to have an even distribution of benign and malicious samples. The malicious samples were made up of Neris, Rbot, Virut, Menti, and Sogou botnet families. Murlo and NSIS.ay families were excluded as they were used for the creation of the proto zero-day dataset. The approximately 110,000 malicious flows were down-sampled to 59,000, which was the number of benign samples in the dataset; the datasets were split into training, validation, and testing sets in a 72%, 8%, and 20% ratio.

Table 1: Botnet Families and Network Flows from the CTU-13 Dataset with additional Benign Traffic

Family	#Flows	Family	#Flows
Neris	190,028	Sogou	72
Rbot	46,796	dMurlo	11,537
Virut	85,779	NSIS.ay	7,645
Menti	4,810	Benign	56,665

To balance the classes in the datasets for multiclass classification, we ensured that included classes would have a sufficient number of respective samples, so as to allow the model to train well on that class. Empirically, we determined that a class required a minimum of 30,000 samples in order for the classifiers to perform effectively. A consequence of this being that botnet families Sogou, Menti, NSIS.ay, and Murlo were removed from the training set. As indicated in Table 1, these families did not have enough samples for practical up-sampling. The resultant dataset included traffic labelled as benign, Rbot, Virut, or Neris. These classes were then down-sampled to consist of 45000 samples each; the datasets were split into training, validation, and testing sets in a 72%, 8%, and 20% ratio.

5 IMPLEMENTATION

In this section we discuss the how decisions were made concerning the architecture and hyperparameters of each binary and multiclass classification model. As there was overlap in reasoning for the binary and multiclass classification tasks, discussion of architectures for each respective task were grouped.

5.1 Hyperparameter Tuning

The performance of a DL model is heavily influenced by decisions made regarding hyperparameters. Discovering optimal hyperparameters is a process which typically involves references to existing literature, and exploring iterations of training slightly different models and evaluating which yield better results (i.e., Grid-Search). This process is both computationally expensive and time consuming. We adopt an alternate approach using an extension of Keras Tuner called Hyperband [9, 13].

Hyperband presents an efficient means of optimising both hyperparameters and general network topology. Starting from a predefined set of options including ranges of layer sizes, activation functions, learning rates, and dropout rate, Hyperband adopts an early-stopping strategy to identify promising combinations of hyperparameters. These are trained for a small number epochs to assess their performance. The top-performing configurations are kept for further training, while the rest are discarded. This process is iterated until the algorithm converges on a network topology and set of hyperparameters that yield near-optimal performance [9].

[7]The specific features present in these datasets are described in tables 8, 7 and 6 in the Appendix

5.2 Architectures

Expanding on the discussion of the contributions of this paper, our aim is to evaluate five different DL models: an MLP, shallow CNN, deep CNN, AE, and AE+CNN. The split in our aims, between detecting and classifying botnet traffic, requires that for each model we train a binary classifier (for botnet detection) and a multiclass classifier (for botnet classification). The following is an exposition of the topology and hyperparameters of each model - arrived at through implementation of the Hyperband process discussed in 5.1. A tabulated representation of this information can be found in 9.5 in the Appendix.

5.2.1 Multilayer Perceptron

The MLP is, by design, our simplest model. While this model was not implemented under the hope that it would be the most effective model, it is useful in the sense that it's simplicity enables it to act as an indication of how difficult the classification problems are.

The MLP binary classifier has an input layer, connected to a single densely connected layer with 33 neurons using the Tanh activation function. The output of this layer is fed into another densely connected layer with a Sigmoid activation function for classification. The architecture for the multiclass classification model is markedly similar, the difference being an increase in the size of the hidden layer, with 128 neurons, and a classification layer which uses a Softmax function.

5.2.2 Convolutional Neural Networks

For both classification tasks we introduced two CNN architectures inspired by the implementations of 1D-CNNs as per [25, 29]. Each task has a respective shallow CNN (CNN v1), and deeper CNN (CNN v2). Across all models, we adopted the Adam optimiser during the training phase, which has had widespread success in related literature [11, 26, 29]. Furthermore, all networks shared a common filter size of 3×1, with a stride of 1. Following these convolutional layers, MaxPooling was employed as a down-sampling technique to reduce spatial dimensions and retain critical features of the input.

With respect to the binary classifiers, CNN v1 had two 1D convolutional layers made up of 128 and 416 filters, respectively. The output from the final MaxPooling layer was flattened, and fed into a densely connected layer with 352 neurons. A final Sigmoid layer was used for classification. CNN v2 implemented three 1D convolutional layers, with 40, 136, and 232 filters. After the final MaxPooling layer, a dropout of 0.5 was introduced to combat overfitting. The result was flattened, and channelled into a dense layer of 104 neurons, followed by another dense layer of 40 neurons before a final Sigmoid layer for classification. Aside from the classification layer, all applicable layers made use of the Rectified Linear Unit (ReLU) activation function, which introduced non-linearity to the model - a decision determined through the hyperparameter tuning process.

For multiclass classifiers, CNN v1 had two 1D convolutional layers made up of 232 filters each; after the second MaxPooling layer, the output was flattened and inputted to densely connected layer of 72 neurons, before a final Softmax classification layer. CNN v2 implemented three 1D convolutional layers, made up of 232, 104, and 40 filters, respectively. After the third MaxPooling layer we introduced a 0.5 dropout to the model for overfitting. The output was then flattened and inputted to a densely connected layer of 296 neurons, after which another dropout layer of 0.25 was introduced. Two more densely connected layers sized 456, and 168 were implemented before the Softmax classification layer. The dense and convolutional layers used the Tanh activation function - the decision to implement Tanh was made through empirical findings, through the hyperparamter tuning process.

5.2.3 Autoencoders

The architecture of the AEs we implemented for the binary classification and multiclass classification problems were very similar, with differences appearing in the size of the layers. Drawing inspiration from [11], each AE had five fully connected hidden layers, and a classification layer (Sigmoid or Softmax). The sizes of these layers for the binary classifier were [232, 72, 40, 104, 232], whereas the multiclass classifier has layers sized [168, 104, 104, 40, 104]. Both the binary and multiclass classifiers implement a Tanh activation function in each layer.

5.2.4 AE+CNN

The AE + CNN is an ensemble of the previously implemented AE and CNN v1. For the binary classification task, five densely connected layers were used for the encoding process, which had 136, 136, 72, 104, 136 neurons, respectively. The output from the fifth layer was reshaped in order to be suitable input for the CNN. Subsequently, two one-dimensional convolutional layers were implemented with 64 and 32 filters, respectively. Each convolutional layer was followed by a MaxPooling layer, where the final MaxPooling layer was flattened and channelled into a densely connected layer with 8 neurons, connected to the final classification layer which used the Sigmoid activation function. All of the applicable layers used a ReLU activation function, and the Adam optimiser - for the same reasons as before.

The multiclass AE+CNN implemented a similar architecture, with five densely connected encoding layers with 136, 104, 72, 40, 104 neurons, respectively. The same reshaping and subsequent convolutional and MaxPooling layers were included, with 32 and 96 neurons in each respective convolutional layer. These layers, where applicable, implemented a Tanh activation function, as opposed to the binary classification model's ReLU. A final softmax layer was used for classification.

6 EXPERIMENT DESIGN

The following section breaks down the process of experimentation addressing the aims of this work. In recognition of the split in problem type, the discussion is bifurcated into one section concerned with botnet detection, and another which addresses botnet classification. The nature of some of the experiments - measuring memory usage, and inference time - is such that it is difficult to be certain if results accurately represent real performance. This is mostly unavoidable, as factors like background processes, temperatures, memory release are beyond our control. In an attempt to mitigate these factors, experiments were performed on the same M2 Macbook Air with 16Gb of memory, with all unnecessary processes terminated.

6.1 Evaluation Metrics

To evaluate the performance of a model, we use accuracy, FPR, FNR, mean inference time (MIT), and mean memory usage (MMU). While accuracy is a standard metric w.r.t. determining the effectiveness of a classifier, much of the related work prefers F1-score [10, 11, 18]. The advantage F1-score offers is that it provides a fairer representation of a model's performance in the case of unbalanced datasets [27]. In our case, measures were undertaken to ensure a balanced training and testing set; consequently, accuracy was preferred to F1-score.

FPR quantifies the fraction of benign results incorrectly identified as malicious by the model [7]. Conversely, FNR measures the proportion of actual threats misclassified as benign. In the context of intrusion detection systems, both metrics are particularly important. High FNRs undermine the essential purpose of an IDS - to detect threats. On the other hand, an elevated FPR can erode trust in the system. If users are frequently alerted to false threats, they may begin to ignore genuine threats [7]. These metrics are defined as:

$$\begin{aligned}\text{Accuracy} &= \frac{TP + TN}{TP + FP + FN + TN}\\ \text{FPR} &= \frac{FP}{FP + TN}\\ \text{FNR} &= \frac{FN}{FP + FN}\end{aligned}$$

Where TP refers to true positives, TN to true negatives, FP to false positives, and FN to false negatives.

MIT and MMU provide the mean time for a model to make an inference, and memory required to make 100 inferences, respectively. These measurements require multiple iterations of measurements for each model, in order to extract the mean. An alternate approach of taking the worst-case performance of these measurements was considered. The advantage of a worst-case measurement is that it enables us to infer the hardware specifications needed to implement the model without fear of failure, providing an upper bound on the memory usage or inference time. However, we recognise that there are significant difficulties in ascertaining accurate measurements of memory usage or CPU time, (see 6.2.2 for further discussion of these challenges). Consequently, we can be less confident in analyses like the worst-case, which look at a specific value: it is not unlikely that a recorded worst-case measurement was caused by a uncontrollable factor (such as memory not being freed before the recorded activity). On the other hand, the mean of a collection of measurements is more tolerant to these types of errors due to their larger pool of data. The aim is that the trends in model performance, in terms of MIT and MMU, persist through possible erroneous measurements.

6.2 Binary Classification Task

6.2.1 Comparing the Classification Performance of Models Evaluated on Normal and Proto Zero-Day Test Sets

This experiment establishes a baseline evaluation of how MLP, CNN v1, CNN v2, AE, and AE+CNN perform, in terms of classification accuracy, FPR, and FNR on the conventional testing set. Subsequently, these models are evaluated on the additional testing set, as discussed in 4.1 and 4.2.3 to ascertain their ability to generalise to unseen botnet families. Accuracy is determined through the use of the Keras framework's 'evaluate' function. For calculation of FPR and FNR, a model makes predictions on a testing set which are compared with the set of ground truths to determine the FP, FN, TP, TN values used in the formulas outlined in 6.1.

6.2.2 Evaluating the Effect of a Reduced Input Feature Space on Computational and Classification Performance

While DL has largely alleviated the necessity for feature engineering, as is present in machine learning, larger feature spaces typically incur a greater computational cost [11, 19]. We aim to explore the relationship between computational and classification performance of the five DL models when trained on inputs of 100%, 50%, and 30% of the feature space.

We define computational performance as the memory usage (MMU) and inference time (MIT) of a model. Determining accurate values for these metrics presents significant difficulties: during the execution of a program, there are invariably other processes running concurrently. Furthermore, memory management of an operating system is largely beyond our control. Empirically, we found that a when a program iterated over each model, measuring memory use and inference time, there was a consistent increase in memory consumption with each subsequent iteration, irrespective of model complexity. While the specific cause of this was not thoroughly investigated, it presented a risk that models evaluated by the program earlier would have have an advantage over those evaluated later.

To minimize the potential impact these factors may introduce, we elected to measure the inference time and memory utilization across batches ($b = 10$) of the test dataset, each batch comprising of a fixed number of samples ($n = 100$). In this revised approach, the system was rebooted after each subsequent evaluation of a batch. Memory use was measured using the 'Memory Profiler'[8] python package, which provides a list of memory usage taken at snapshots during the program's execution. We record the model and memory usage for each batch, extracting the mean usage from these results. Inference time was determined using the 'timeit' package from Python's Standard Template Library [23]. We measure and record the time taken for a model to make predictions over a batch, extracting the mean from from these records. The experiment requires that we implement this procedure three times for each model, using training and testing sets containing 100%, 50%, and 30% of the available features.

6.3 Multiclass Classification Task

6.3.1 Evaluate the Classification Accuracy of each model overall, and for each Botnet Family

This experiment evaluates the MLP, CNN v1, CNN v2, AE, and AE+CNN models to determine their overall accuracy scores, as well as their accuracy scores for each specific botnet family. Metrics like FPRs and FNRs are not applicable to multiclass classification problems, as they require a binary relation. It is still useful, however, to determine some sense of how each model performs relative to each class in the multiclass classification. We evaluate each model on the normal testing set only, as the concept of the proto zero-day testing set is incongruous with multiclass classification. From this evaluation we are able to determine a general accuracy for each model, as well as the accuracy of each model respective to the available classes. Reiterating the discussion from 6.1, we use accuracy because we have ensured that each class has an equal representation of samples, at 8000. For each model, we make classifications using the Keras Library's 'predict()' method, and store the results in a confusion matrix.

6.3.2 Evaluating the Effect of a Reduced Input Feature Space on Computational and Classification Performance

We evaluate how datasets with reduced feature spaces influence the computational and classification performance of models. More specifically, five models are trained on three datasets containing a random sample of 100%, 50%, and 30% of the total features - this translates to datasets with 74, 37, and 22 features, respectively. We then observe the effect that a reduced feature space might have on a single, or group of models' computational and classification performance. We regard the computational performance to be the MIT and MMU of a model, while classification performance is largely defined as a model's accuracy across all classes, with a secondary focus on the portion incorrectly classified traffic. This latter focus enables us to determine which models might struggle to classify specific families, or if certain families are poorly classified by all models.

The experiment procedure echoes what was outlined in 6.2.2: to mitigate the challenges we face when measuring memory usage and inference time we determine the MIT and MMU by taking measurements over a series batches ($b = 10$) containing a fixed number of samples ($n = 100$), between each measurement the system is rebooted. From these measurements we determine the mean inference time (MIT) and mean memory usage (MMU).

7 RESULTS AND DISCUSSION

In the following we present and discuss the findings from the experiments outline in 6. As with the description of experimental design, there is a natural bifurcation of results into those which involve the binary classification task and multiclass classification task. We begin with the discussion of binary classifiers, followed by multiclass classification results.

7.1 Binary Classifiers for Botnet Detection

7.1.1 Performance on Normal and Proto Zero-Day Test Sets

Figure 2 displays the accuracy, FPR, and FNR of the MLP, CNN v1, CNN v2, AE, and AE+CNN when evaluated on the normal and proto zero-day testing sets. Performance on the normal testing set provides an indication of a model's ability to generalise to unseen traffic that belongs to botnet families which were present in the training set. On the other hand, results relating to the proto zero-day testing set relates to a model's ability to classify traffic from botnet families which were entirely excluded from the training set.

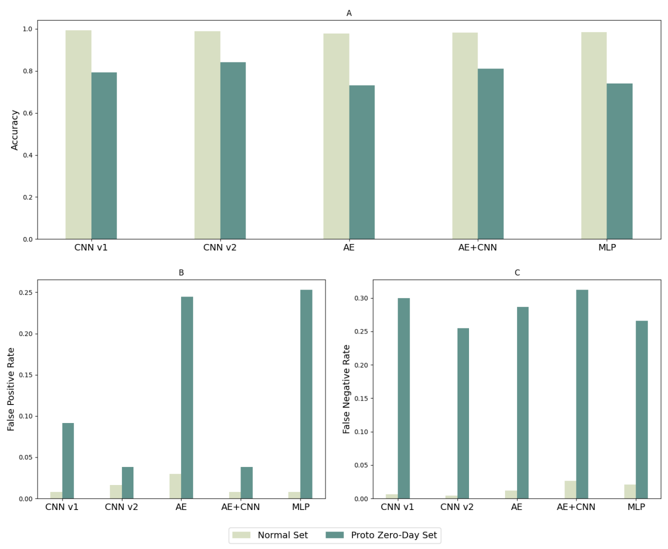

Figure 2: sub-Figures A, B, and C, respectively, show Accuracy, False-Positive Rates, and False-Negative Rates of each model when evaluated on the normal and proto zero-day testing sets.

All models achieved relatively high (≥ 0.979) accuracy scores when evaluated on the normal testing set. the MLP, our least complex model attained a score of 0.986, indicating the the task of botnet detection from known families is fairly simple. The CNN v2 attained the highest classification accuracy of 0.993, while the AE was the lowest with a score of 0.979. We suggest an explanation for the poor performance of the models which implement an autoencoder (the AE+CNN attained

[8]Memory Profiler can be accessed at https://github.com/pythonprofilers/memory_profiler

the second lowest accuracy) is that latent vectors created by the encoding phase fail to capture some distinguishing features of the input data. When comparing best and worst models - CNN v2 and AE - there is an absolute improvement by the CNN v2 of 0.014. While this improvement is small, we calculate that the error rate of the CNN v2, at 0.007, represents a 0.667 reduction in errors relative to the error rate of the AE, at 0.021.[9]

In order to recognise the significance of this reduction, we need to contextualise the problem. Networks are often required to handle a large volume of traffic; for example, a network on a college campus with 10, 000 users may see a typical transfer of 7TB of data every 24 hour period [24]. For an intrusion detection system monitoring this network, a relative reduction in error rate of 0.667 could entail thousands of fewer errors.

These errors can be broken down into two categories: false negatives and false positives. From this, we work out the FPR and FNR for each model. When FPR was evaluated on the normal testing set, the CNN v2, AE+CNN, and MLP were the best performers with FPRs of 0.008. The CNN v2 attained an FPR of 0.016. The poorer performance of the CNN v2 in comparison to the CNN v1 might be explained by the increased complexity of CNN v2, with the additional convolutional layer and fully connected layers causing overfitting. The worst performer w.r.t FPR was once again the AE, with a measure of 0.030. It is difficult to suggest an acceptable tolerance for FPR and FNR: factors around the type of information being secured, the size of the organisation, and general security posture may all influence what might be deemed acceptable. However, the FPRs achieved by the CNN v2, AE+CNN, and MLP, with respective accuracies of 0.993, 0.983, and 0.986, present an improvement on existing work [12, 28].

Evaluation on the proto zero-day testing set showed a decline in performance in terms of accuracy, FPR, and FNR across all models. The best performing model w.r.t. classification performance was the CNN v2, which achieved an accuracy of 0.842, FPR of 0.038, and FNR of 0.255. The same model evaluated on the normal testing set achieved scores of 0.990, 0.016, and 0.004 for each respective metric. The decline in classification performance aligns with our understanding that distinct botnet families are likely to exhibit, at least partially, different behaviour. It is this difference which causes the models to struggle when classifying traffic belonging to families entirely excluded from the training set. However, the model's performance on the proto zero-day testing set remains a significant improvement on guessing. We suggest that an explanation for this improvement rests on the notion that while there are certainly some differences, distinct botnet families must express certain shared behaviour that is not present in benign traffic. This would be behaviour intrinsic to all botnet families, as opposed to behaviour distinct to individual families. [10] This shared behaviour amongst botnet families is what a model would recognise in the proto zero-day testing set.

There was also a more pronounced degree of variability in classification performance of the five models when evaluated on the proto zero-day testing set, relative to the evaluation based on the normal testing set. For instance, the mean accuracy, over all five models, when evaluated on the proto zero-day testing set was 0.783 with a standard deviation of 0.047. In contrast, when evaluated on the normal testing set, we determined a standard deviation of 0.005 around a mean of 0.986. This variability indicates that there is a fairly substantial difference between each models' ability to learn the more complex, intrinsic botnet behaviour which enable better detection of unknown botnet families.

From these two ideas - that good performance on the proto zero-day testing set requires learning more complex behaviour patterns intrinsic to all botnets, and that there is greater variability in model's classification performance when evaluated on the proto zero-day testing set - we make the claim that certain models, specifically models which implement convolutional layers, are significantly more capable of learning these more complex patterns. We ground this claim in the observation that the CNN v1, CNN v2, and AE+CNN achieved accuracies on the proto zero-day set of 0.842, 0.793, and 0.810, respectively, whereas, the AE and MLP achieved accuracies of 0.732 and 0.740. A potential explanation for the efficacy of convolutional layers towards learning these behaviour patterns is that they are particularly good at learning patterns which emerge from the relationship between closely related features - which may represent the more intrinsic behaviour general to all botnets. The AE and MLP are able to achieve high (> 0.979) accuracies on the known botnet families because they can learn the defining features of each individual family, as opposed to learning some underlying pattern seen in all families. This is sufficient for binary classification on known families, but generalises poorly to classification of unknown families, because these 'defining' features may not be present.

7.1.2 Effect of a Reduced Input Feature Space on Computational and Classification Performance

Sub-Figures 3C and 3D show the MIT and MMU for each model when using 30%, 50%, and 100% of the available input feature space. From Figure 3D, we observe a small but clear trend whereby increasing the feature space of a model's input has an associated increase in the memory usage of that model. This result aligns with the position outlined by Sarker [19], that the absence of feature engineering in DL may increase computational requirements of DL models. While the trend is consistent across all models, the proportional increase in memory is small; going from feature spaces of 30% to 100% (that is, 22 to 74 features), we observe a mean increase in MMU of $\approx 10\%$. In the most extreme case, CNN v1, the jump from 30% to 100% of features saw an increase in MMU of ≈ 90 MB, or 20.726%. For CNNs, an explanation for the relatively small increase to MMU when given larger feature spaces is likely found in their use of sparsely connected layers and parameter sharing, which reduce the number of trainable parameters in a model.

From sub-Figure 3D, we observe that the CNN v1 sees the largest increase in memory utilisation when evaluated on larger feature spaces. However, larger feature spaces in this model also result an improvement to classification performance. From sub-Figure 3A, the normal accuracy improves from 0.968 to 0.993, sub-Figure 3E shows that the FPR improves from a rate of 0.027 to 0.008, and FNR (sub-Figure 3F) improves from a rate of 0.037 to 0.006. These improvements to classification performance when using larger feature spaces are significant. Moreover, they are, to lesser extents, observable in the four other models. Consequently, we make the claim that the increase in memory utilisation associated with larger feature spaces is justified by the improvements made to classification performance, given the informal notion that it is easier to buy more memory than it is to attain higher classification accuracy.

This argument is made clearer when we evaluate the impact of a reduced feature space on the proto zero-day testing set. To this end, sub-Figure 3B shows that the best performing model using 100% of features, in terms of accuracy, was the CNN v2 with a score of 0.842.

[9]Formulas and calculations provided in Appendix
[10]We use 'all' here, referring to all families sampled. There may of course be exceptions

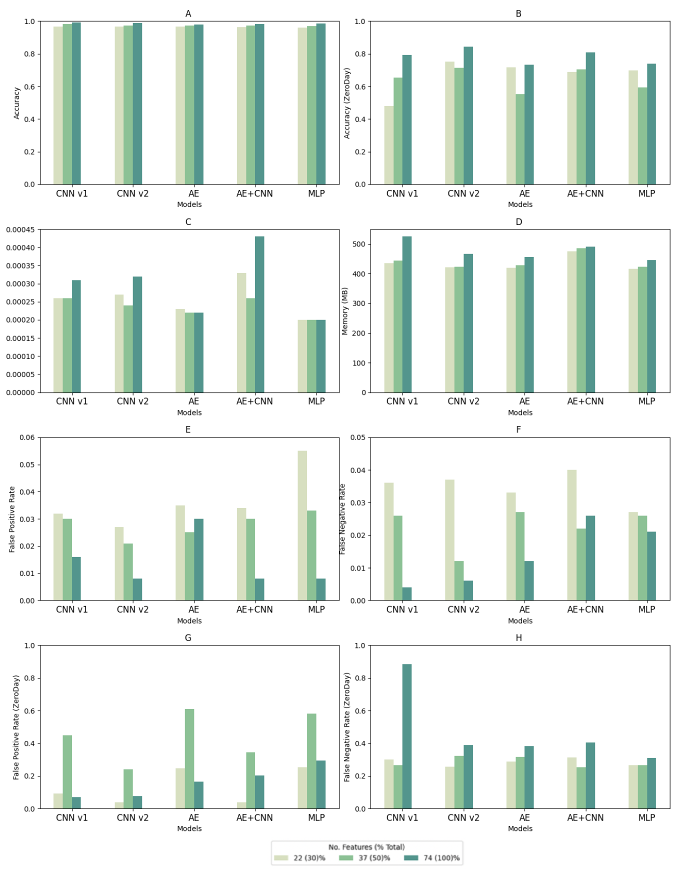

Figure 3: Figures illustrating model performance on testing sets comprised of 30% (22 features), 50% (37 features), 100% (74 features) of total features. The figures show Normal Accuracy (A), Proto Zero-Day Accuracy (B), Mean Inference Time (C), Mean Memory Usage (D), False Positive Rate (E), False Negative Rate (F), False Positive Rate on Zero-Day set (G), and False Negative Rate on Zero-Day set (H).

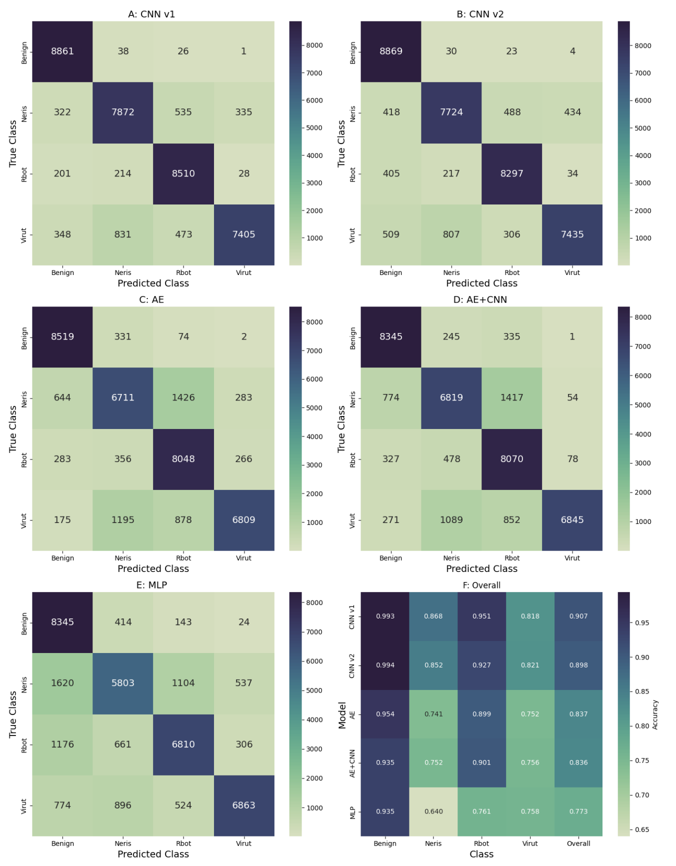

Figure 4: Sub-Figures A to E show the respective confusion matrices for CNN v1, CNN v2, AE, AE+CNN, and MLP models, showing performance of each model w.r.t. individual classes. Sub-Figure F shows the overall accuracy, false-positive rate, and false negative rate of each model w.r.t. each class.

When this model was trained and tested on the set of 30% of features, the accuracy declined to 0.751. Furthermore, sub-Figure 3G shows that the FPR increased from 0.038 to 0.076, and similarly that the FNR declined from 0.255 to 0.389, evident in sub-Figure 3H. We suggest that an explanation for the more significant decline in classification performance when evaluated on the proto zero-day set compared to the normal testing set is that detection of botnet traffic from a range of known botnet families is a fairly simple task, enabling (relatively) high accuracies to be obtained with fewer features. On the other hand, echoing the discussion in 7.1.1, detection of botnet traffic from a range of unknown botnet families requires models to learn complex patterns shared by all botnets, which the datasets with reduced feature spaces are not rich enough to support.

7.2 Multiclass Classifiers for Botnet Classification

7.2.1 Classification Accuracies of each model overall, and for each Botnet Family

The classification performance of each of the five multiclass classifiers is shown in Figure 4; each model has a corresponding confusion matrix which displays its predictions, and enables evaluation of how the model performs w.r.t. each botnet family. Sub-Figure 4F offers a holistic representation of all the model's performance on each respective family. Evident in Figure 4F, the overall accuracies of the models when classifying traffic into respective families of Benign, Neris, Rbot, and Virut were expectedly lower than the binary classification task model accuracies, with a mean accuracy score across all classes of 0.850, compared to 0.986. The best performance was observed in the CNN v1, which achieved an average accuracy across all families of 0.907. We observed that there was again a marked improvement, in terms of overall classification accuracy, seen in the models which used convolutional layers, with the exception of the AE+CNN model. CNNs are known for their efficacy when learning hierarchical relationships between features, which is a useful way of reasoning about the necessary and sufficient conditions when making a classification [6]. In the context of this classification task, this ability may be an explanation for their performance, as they are better able to combine the surface level and more complex features of network flows in order to learn more detailed patterns from the data.

As with the binary classification task, the shallow CNN (v1) outperforms the deep CNN (v2) by a small margin, with accuracies of 0.907 and 0.898, respectively. This is a percentage point increase of 0.009, which is fairly negligible. These findings allow us to make the claim that whichever patterns are learned by the CNNs are able to be learned with a shallow network, and that additional complexity (and associated dropout layers to combat overfitting) is unnecessary.

In the discussion from 7.1.1, concerning binary classification accuracy on the proto zero-day set, we suggested that the models which implemented convolutional layers (CNN v1, CNN v2, and AE+CNN) performed better due to the ability of CNNs to learn complex underlying behaviour patterns present in all botnet families. With respect to multiclass classification accuracy, CNN v1 and v2 are the best performers with overall accuracies of 0.907 and 0.898, respectively; while the AE+CNN has substantially worse classification performance, with an accuracy of 0.837. We have just proposed that a CNNs' ability to capture hierarchical relationships in data may be an explanation for their effectiveness for this problem; consequently, we suggest that an explanation for the relatively poor performance of the AE+CNN, which should benefit from this property of the convolutional layers, is that the encoding phase of the network reduces the complexity of the data to a point where the subsequent CNN is unable to learn the necessary hierarchical relationships, because they no longer 'exist' in the encoded representation. We further suggest that the reason for this is that the typical purpose of an autoencoder is to encode the input into a lower-dimensionality representation, from which an approximation of the original input can be reconstructed. This process may encourage the encoder to prioritise the more 'visible', surface-level patterns as they would be the best way to approximate the input. The consequence being that the encoder begins to act as a bottleneck - the features which aid more complex pattern learning are not present in the encoding, preventing a subsequent CNN from exploiting them. When we compare the results of the AE+CNN to the AE, similar accuracy rates are observed across the board, with overall accuracies of 0.837 and 0.836, respectively. This appears to reaffirm the notion of the encoder acting as a bottleneck for further classification.

Marín et al. [12] implemented a multiclass classifier of a deep CNN fed into an LSTM, trained on a dataset of Benign, Neris, Rbot, and Virut classes. Our best performing model, the CNN v1, showed an improvement on their CNN+LSTM, with an overall accuracy of 0.907 to their 0.765. Apart from the model architecture, a significant difference between our model and theirs is that our CNN v1 uses network flows as data, while they use bytes from the packet payloads, which are encrypted. We suggest that the difference in classification performance is caused by their model struggling to learn meaningful representations from the encrypted data. In support of this notion we refer to their binary classification experiments, where their payload-based classifier obtained an accuracy of 0.650, a result significantly lower than their flow-based classifier at 0.900, and our best performing binary classifier at 0.979 [12].

Notably, the their CNN+LSTM struggled the most when classifying instances of Neris and Virut families. Figures 4B and F quite clearly show that this trend is observable across all models. Marín et al. suggest that an explanation of this is a result of the similarity between the Neris and Virut botnet families, causing models to misclassify one as the other. The results in Figure 4 show that Virut samples are most frequently misclassified as Neris, supporting this notion. However, Neris samples are most frequently misclassified as Rbot, which may indicate that there is some other cause for the models' confusion.

7.2.2 Consequences of Reduced Feature Space on Computational and Multiclass Classification Performance

Figure 5A shows the accuracy of each model given the size of the feature space. As the number of features is reduced we observe an associated decline in classification accuracy. This aligns with the intuition that more features enable a model to use the relationships between features to learn more complex patterns, facilitating better classification performance. The two CNN models show a more substantial decline in accuracy when the features are reduced from 50% to 30%, when compared to 100% reduced to 50%. In some sense, this is an unintuitive result; the larger reduction in feature space is accompanied by a smaller reduction in accuracy. An explanation for this lies in the notion that the CNN's success is a consequence of their ability to learn useful patterns from relations between features. When the feature space is reduced from 100% to 50%, there remains a sufficient number of features

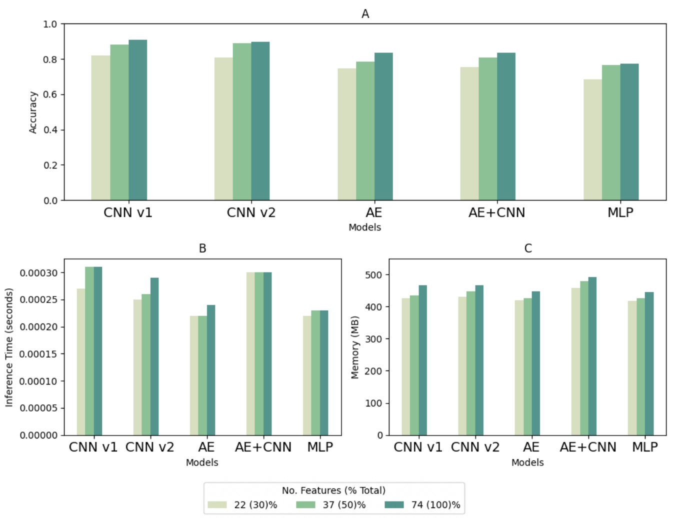

Figure 5: Sub-Figures A, B, and C show the respective accuracy, Mean Inference Time, and Mean Memory Usage when evaluated on datasets of varying feature size.

to enable the models to learn these patterns. In the reduction from 50% to 30%, while fewer features are removed, the resultant dataset is not detailed enough for CNN's to learn useful information. However, an alternate explanation of why the decline in accuracy is more pronounced when going from 50% to 30% of feature space, as opposed to 100% to 50%, is that the process of reducing the feature space uses random sampling to select features. This may have resulted in important features being absent from the 30% dataset. To this end, the cause of the decline might be a result of quality, rather than quantity of features.

We observe that the three models that employ a CNN appeared to use more memory than the AE and MLP. We expect the MLP to use the least memory, as it is the least complex model. However, the CNN v1 has fewer trainable parameters than the AE, while using more memory. We suggest that the cause of this, and a general explanation for why CNNs seem to have the largest MMU, is that the CNN has to store filters and their respective activation maps in memory, which can become fairly expensive [6].

We find that in every model, there is an increase to MMU as the feature space increases. This was an expected result, reaffirming the position outlined in 7.1.2 that larger input feature spaces are associated with an increase to a model's MMU. However, we maintain that this increase to MMU represents a relatively small improvement to computational performance, and is often coupled with a fairly substantial improvement in classification performance. For instance, when going from 30% to 100% of features the MLP's MMU performance declines, with an increased utilisation of only ≈ 28MB; however, there is an accompanied improvement to accuracy of 0.089..

The CNN v1, v2 and AE+CNN models have the three slowest inference times when evaluated on 100% of the feature space. This result matches our expectations, as CNNs operate by executing a convolution operation for each filter across all output elements of the preceding layer [6]. This convolutional process is computationally intensive and is not a requirement in the Autoencoder (AE) and Multilayer Perceptron (MLP) architectures. While these trends seemingly continued as the feature space was reduced, we found no discernible relation between the MIT and the size of the feature space.

8 CONCLUSIONS

In this paper we evaluated the classification and computational performance of DL models for botnet detection, and classification. We further explored the effect reduced feature spaces have on their performance. For our first research objective, we found that all models achieved accuracies ≥ 0.979, FPRs ≤ 0.033, and FNRs ≤ 0.026, suggesting that the classification problem was fairly simple. Classification performance declined on the proto zero-day set overall, and clearly showed that CNNs were more capable algorithms at detecting zero-day traffic. We further observed a clear trend that larger feature spaces where associated with a larger MMU, affirming our expectation that feature selection might improve computational performance. However, in reducing feature space we also observe a substantial decline in classification performance. We found no evidence of a trend between feature space size and MIT; however, we acknowledge that there were serious limitations to the accuracy of measuring MIT. With respect to the second research objective, we found a fairly large differential between the classification performance of models which used convolutional layers and those that did not. We suggest that the efficacy of CNNs at learning hierarchical relations between features is an explanation for this. As with the binary classifiers, we observed a trend where larger feature spaces were associated with slightly greater MMU. However, the larger feature spaces resulted in substantially better classification performance.

We suggest that the most significant observation from this work is found in the discussion of classification performance of binary classifiers on the proto zero-day set. In this area, we observed accuracies high enough to act as a proof of concept, that patterns learned from existing botnet families can be used for detection on unknown families, while still leaving considerable room for improvement.

REFERENCES

[1] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. 2006. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement (IMC '06)*. Association for Computing Machinery, New York, NY, USA, 41–52. https://doi.org/10.1145/1177080.1177086

[2] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. 2018. Mobile encrypted traffic classification using deep learning. In *2018 Network traffic measurement and analysis conference (TMA)*. IEEE, 1–8.

[3] Elisa Bertino and Nayeem Islam. 2017. Botnets and internet of things security. *Computer* 50, 2 (2017), 76–79.

[4] Ronald Cheng. 2017. D 2 PI : Identifying Malware through Deep Packet Inspection with Deep Learning. https://api.semanticscholar.org/CorpusID:53062187

[5] S. García, M. Grill, J. Stiborek, and A. Zunino. 2014. An empirical comparison of botnet detection methods. *Computers Security* 45 (2014), 100–123. https://doi.org/10.1016/j.cose.2014.05.011

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[7] Fariba Haddadi, Duc Le Cong, Laura Porter, and A Nur Zincir-Heywood. 2015. On the effectiveness of different botnet detection approaches. In *Information Security Practice and Experience: 11th International Conference, ISPEC 2015, Beijing, China, May 5-8, 2015, Proceedings*. Springer, 121–135.

[8] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. 2017. Characterization of Tor Traffic using Time based Features. In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*. INSTICC, SciTePress, 253–262. https://doi.org/10.5220/0006105602530262

[9] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The journal of machine learning research* 18, 1 (2017), 6765–6816.

[10] Hyun-Kyo Lim, Ju-Bong Kim, Kwihoon Kim, Yong-Geun Hong, and Youn-Hee Han. 2019. Payload-based traffic classification using multi-layer lstm in software defined networks. *Applied Sciences* 9, 12 (2019), 2550.

[11] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. 2020. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing* 24, 3 (2020), 1999–2012.

[12] Gonzalo Marín, Pedro Caasas, and Germán Capdehourat. 2021. Deepmal-deep learning models for malware traffic detection and classification. In *Data Science–Analytics and Applications: Proceedings of the 3rd International Data Science Conference–iDSC2020*. Springer, 105–112.

[13] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. 2019. KerasTuner. https://github.com/keras-team/keras-tuner. (2019).

[14] Keiron O'Shea and Ryan Nash. 2015. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458* (2015).

[15] Nagababu Pachhala, S. Jothilakshmi, and Bhanu Prakash Battula. 2021. A Comprehensive Survey on Identification of Malware Types and Malware Classification Using Machine Learning Techniques. In *2021 2nd International Conference on Smart Electronics and Communication (ICOSEC)*. 1207–1214. https://doi.org/10.1109/ICOSEC51865.2021.9591763

[16] Eva Papadogiannaki, Giorgos Tsirantonakis, and Sotiris Ioannidis. 2022. Network Intrusion Detection in Encrypted Traffic. In *2022 IEEE Conference on Dependable and Secure Computing (DSC)*. 1–8. https://doi.org/10.1109/DSC54232.2022.9888942

[17] Abdurrahman Pektaş and Tankut Acarman. 2018. Botnet detection based on network flow summary and deep learning. *International Journal of Network Management* 28, 6 (2018), e2039. https://doi.org/10.1002/nem.2039 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nem.2039 e2039 nem.2039.

[18] Michal Piskozub, Fabio De Gaspari, Freddie Barr-Smith, Luigi Mancini, and Ivan Martinovic. 2021. MalPhase: Fine-Grained Malware Detection Using Network Flow Data. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. ACM. https://doi.org/10.1145/3433210.3453101

[19] Iqbal H Sarker. 2021. CyberLearning: Effectiveness analysis of machine learning security modeling to detect cyber-anomalies and multi-attacks. *Internet of Things* 14 (2021), 100393.

[20] Stratosphere. 2015. Stratosphere Laboratory Datasets. (2015). Retrieved March 13, 2020, from https://www.stratosphereips.org/datasets-overview.

[21] Pablo Torres, Carlos Catania, Sebastian Garcia, and Carlos Garcia Garino. 2016. An analysis of recurrent neural networks for botnet detection behavior. In *2016 IEEE biennial congress of Argentina (ARGENCON)*. IEEE, 1–6.

[22] Jos van Roosmalen, Harald Vranken, and Marko van Eekelen. 2018. Applying deep learning on packet flows for botnet detection. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. 1629–1636.

[23] Guido Van Rossum and Fred L. Drake. 2009. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

[24] A Villa and Elizabeth Varki. 2012. Characterization of a campus internet workload. In *Proceedings of CATA*. 140–148.

[25] Wei Wang, Yiqiang Sheng, Jinlin Wang, Xuewen Zeng, Xiaozhou Ye, Yongzhong Huang, and Ming Zhu. 2017. HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection. *IEEE access* 6 (2017), 1792–1806.

[26] Zihao Wang, Kar Wai Fok, and Vrizlynn L.L. Thing. 2022. Machine learning for encrypted malicious traffic detection: Approaches, datasets and comparative study. *Computers &amp Security* 113 (feb 2022), 102542. https://doi.org/10.1016/j.cose.2021.102542

[27] Shane Weisz and Josiah Chavula. 2022. Community Network Traffic Classification Using Two-Dimensional Convolutional Neural Networks. In *e-Infrastructure and e-Services for Developing Countries*, Yahya H. Sheikh, Idris A. Rai, and Abubakar D. Bakar (Eds.). Springer International Publishing, Cham, 128–148.

[28] M. Yeo, Y. Koo, Y. Yoon, T. Hwang, J. Ryu, J. Song, and C. Park. 2018. Flow-based malware detection using convolutional neural network. In *2018 International Conference on Information Networking (ICOIN)*. 910–913. https://doi.org/10.1109/ICOIN.2018.8343255

[29] Yi Zeng, Huaxi Gu, Wenting Wei, and Yantao Guo. 2019. $Deep-Full-Range$: A Deep Learning Based Network Encrypted Traffic Classification and Intrusion Detection Framework. *IEEE Access* 7 (2019), 45182–45190. https://doi.org/10.1109/ACCESS.2019.2908225

[30] Hanxun Zhou, Yeshuai Hu, Xinlin Yang, Hong Pan, Wei Guo, and Cliff C Zou. 2020. A worm detection system based on deep learning. *IEEE Access* 8 (2020), 205444–205454.

9 APPENDIX

9.1 Binary Classification Results

Table 2: Results of Binary Classifiers on Default Test Set

Model	Accuracy			Precision			Recall			FPR			FNR		
	100	50	30	100	50	30	100	50	30	100	50	30	100	50	30
CNN v1	0.990	0.972	0.966	0.990	0.972	0.966	0.990	0.972	0.966	0.016	0.030	0.032	0.004	0.026	0.036
CNN v2	0.993	0.983	0.968	0.993	0.983	0.968	0.993	0.983	0.968	0.008	0.021	0.027	0.006	0.012	0.037
AE	0.979	0.974	0.966	0.979	0.974	0.966	0.979	0.974	0.966	0.030	0.026	0.035	0.012	0.027	0.033
AE CNN	0.983	0.974	0.963	0.983	0.974	0.963	0.983	0.974	0.963	0.008	0.030	0.034	0.026	0.022	0.040
MLP	0.986	0.971	0.959	0.986	0.971	0.959	0.986	0.971	0.959	0.008	0.033	0.055	0.021	0.026	0.027

Table 3: Results of Binary Classifiers on Proto Zero-Day Test Set

Model	Unseen Accuracy			Unseen Precision			Unseen Recall			Unseen FPR			Unseen FNR		
	100	50	30	100	50	30	100	50	30	100	50	30	100	50	30
CNN v1	0.793	0.653	0.480	0.904	0.777	0.671	0.700	0.736	0.116	0.092	0.448	0.071	0.300	0.264	0.884
CNN v2	0.842	0.714	0.751	0.960	0.777	0.909	0.745	0.677	0.611	0.038	0.240	0.076	0.255	0.323	0.389
AE	0.732	0.552	0.716	0.783	0.581	0.824	0.713	0.683	0.619	0.245	0.609	0.163	0.287	0.317	0.381
AE CNN	0.810	0.705	0.687	0.957	0.727	0.785	0.688	0.746	0.597	0.038	0.346	0.202	0.312	0.254	0.403
MLP	0.740	0.594	0.698	0.782	0.610	0.744	0.734	0.736	0.691	0.253	0.580	0.294	0.266	0.264	0.309

Table 4: Computational Performance of Binary Classifiers

Model	Memory (MB)			Inference Time		
	100	50	30	100	50	30
CNN v1	465.89	422.91	422.39	0.00032	0.00024	0.00027
CNN v2	526.16	443.78	435.83	0.00031	0.00026	0.00026
AE	457.02	428.02	420.16	0.00022	0.00022	0.00023
AE CNN	490.94	485.50	475.42	0.00026	0.00023	0.00033
MLP	446.05	422.70	416.67	0.00020	0.00020	0.00020

9.2 Multiclass Classification Results

Table 5: Classification and Computational Performance of Multiclass Classifiers

Model	Accuracy			Memory (MB)			Inference Time		
	100	50	30	100	50	30	100	50	30
CNN v1	0.907	0.881	0.819	465.96	434.63	425.77	0.00030	0.00031	0.00027
CNN v2	0.898	0.889	0.809	465.85	446.88	431.28	0.00029	0.00026	0.00025
AE	0.836	0.786	0.745	447.98	427.21	419.70	0.00024	0.00022	0.00022
AE CNN	0.836	0.807	0.755	492.95	478.88	458.87	0.00028	0.00030	0.00030
MLP	0.773	0.764	0.684	445.69	425.44	418.13	0.00023	0.00023	0.00022

9.3 Formulas

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

$$\text{FPR} = \frac{FP}{FP + TN}$$

$$\text{FNR} = \frac{FN}{FP + FN}$$

$$\text{Error Reduction} = \text{Error}_1 - \text{Error}_2$$

$$\text{Relative Error Reduction} = \frac{\text{Error Reduction}}{\text{Error}_1}$$

$$\text{PI(X, Y)} = \frac{\text{Error}_X - \text{Error}_Y}{\text{Error}_Y}$$

$$\text{Mean accuracy} = \frac{1}{n}\sum_{i=1}^{n} \text{model accuracy}_i$$

$$\text{Standard Deviation, } \sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

9.4 Feature Sets

Table 6: Features present when using 30% of feature space

No.	Feature Name	No.	Feature Name
1	Subflow Bwd Bytes	12	Pkt Length Min
2	Fwd Pkt/Bulk Avg	13	Active Min
3	ECE Flag Count	14	Total Length of Bwd Pkt
4	Fwd Pkts/s	15	Bwd Pkt Length Std
5	Total Length of Fwd Pkt	16	Idle Std
6	Idle Min	17	Fwd IAT Max
7	Subflow Fwd Bytes	18	Fwd Bytes/Bulk Avg
8	Fwd Seg Size Min	19	Fwd Header Length
9	Bwd IAT Std	20	URG Flag Count
10	Fwd Pkt Length Max	21	Average Pkt Size
11	Active Std	22	Label

Table 7: Features present when using 50% of feature space

No.	Feature Name	No.	Feature Name
1	Bwd Pkts/s	20	Fwd IAT Max
2	Fwd IAT Std	21	Fwd Pkt Length Max
3	Fwd PSH Flags	22	Bwd IAT Mean
4	Bwd Pkt Length Std	23	Total Bwd Pkts
5	Total Length of Fwd Pkt	24	Bwd Header Length
6	Subflow Fwd Pkts	25	FWD Init Win Bytes
7	Subflow Bwd Bytes	26	Idle Std
8	Bwd Init Win Bytes	27	Bwd Bulk Rate Avg
9	Flow IAT Std	28	Flow Duration
10	URG Flag Count	29	Bwd Pkt/Bulk Avg
11	Bwd IAT Max	30	Bwd IAT Total
12	Active Mean	31	Pkt Length Variance
13	Pkt Length Mean	32	Idle Max
14	Flow IAT Max	33	Fwd Pkt Length Mean
15	Pkt Length Min	34	Fwd Act Data Pkts
16	Pkt Length Max	35	ACK Flag Count
17	Fwd Bulk Rate Avg	36	Fwd IAT Mean
18	Total Fwd Pkt	37	SYN Flag Count
19	Fwd Pkt Length Std		

Table 8: All the features present in flow extraction from CICFlowMeter

No.	Feature Name	No.	Feature Name
1	Flow ID	43	Fwd Pkts/s
2	Src IP	44	Bwd Pkts/s
3	Src Port	45	Pkt Length Min
4	Dst IP	46	Pkt Length Max
5	Dst Port	47	Pkt Length Mean
6	Protocol	48	Pkt Length Std
7	Timestamp	49	Pkt Length Variance
8	Flow Duration	50	FIN Flag Count
9	Total Fwd Pkt	51	SYN Flag Count
10	Total Bwd Pkts	52	RST Flag Count
11	Total Length of Fwd Pkt	53	PSH Flag Count
12	Total Length of Bwd Pkt	54	ACK Flag Count
13	Fwd Pkt Length Max	55	URG Flag Count
14	Fwd Pkt Length Min	56	CWR Flag Count
15	Fwd Pkt Length Mean	57	ECE Flag Count
16	Fwd Pkt Length Std	58	Down/Up Ratio
17	Bwd Pkt Length Max	59	Average Pkt Size
18	Bwd Pkt Length Min	60	Fwd Segment Size Avg
19	Bwd Pkt Length Mean	61	Bwd Segment Size Avg
20	Bwd Pkt Length Std	62	Fwd Bytes/Bulk Avg
21	Flow Bytes/s	63	Fwd Pkt/Bulk Avg
22	Flow Pkts/s	64	Fwd Bulk Rate Avg
23	Flow IAT Mean	65	Bwd Bytes/Bulk Avg
24	Flow IAT Std	66	Bwd Pkt/Bulk Avg
25	Flow IAT Max	67	Bwd Bulk Rate Avg
26	Flow IAT Min	68	Subflow Fwd Pkts
27	Fwd IAT Total	69	Subflow Fwd Bytes
28	Fwd IAT Mean	70	Subflow Bwd Pkts
29	Fwd IAT Std	71	Subflow Bwd Bytes
30	Fwd IAT Max	72	FWD Init Win Bytes
31	Fwd IAT Min	73	Bwd Init Win Bytes
32	Bwd IAT Total	74	Fwd Act Data Pkts
33	Bwd IAT Mean	75	Fwd Seg Size Min
34	Bwd IAT Std	76	Active Mean
35	Bwd IAT Max	77	Active Std
36	Bwd IAT Min	78	Active Max
37	Fwd PSH Flags	79	Active Min
38	Bwd PSH Flags	80	Idle Mean
39	Fwd URG Flags	81	Idle Std
40	Bwd URG Flags	82	Idle Max
41	Fwd Header Length	83	Idle Min
42	Bwd Header Length	84	Label

9.5 Model Architectures

9.5.1 Binary Classifier Architectures

Table 9: Model Descriptions for Binary Classifiers 100% Feature Space

Model	Layer Type	Kernel	Activation
MLP	Input	0,74	-
	Dense	74,33	Tanh
	Dense	33,1	Sigmoid
CNNv1	Input	0,74, 1	-
	Conv1D	3,1,128	ReLU
	MaxPool1D	2	-
	Conv1D	3,128,416	ReLU
	MaxPool1D	2	-
	Flatten	-	-
	Dense	7072,352	ReLU
	Dense	352,1	Sigmoid
CNNv2	Input	0,74,1	-
	Conv1D	3,1,40	ReLU
	MaxPool1D	2	-
	Conv1D	3,40,136	ReLU
	MaxPool1D	2	
	Conv1D	3,136,232	ReLU
	MaxPool1D	2	
	Dropout	rate=0.5	-
	Flatten	-	-
	Dense	1624,104	ReLU
	Dropout	rate=0.5	-
	Dense	104,104	ReLU
	Dense	104,40	ReLU
	Dense	40,1	Sigmoid
AE	Input	0x74	-
	Dense	74,232	Tanh
	Dense	232,72	Tanh
	Dense	72,40	Tanh
	Dense	40,104	Tanh
	Dense	104,232	Tanh
	Dense	232,1	Sigmoid
AE+CNN	Input	0x74	-
	Dense	74,136	ReLU
	Dense	136,136	ReLU
	Dense	136,72	ReLU
	Dense	74,104	ReLU
	Dense	104,136	ReLU
	Reshape	(136,1)	-
	Conv1D	3,1,64	ReLU
	MaxPool1D	1	-
	Conv1D	3,64,32	ReLU
	MaxPool1D	1	-
	Flatten	-	-
	Dense	4224,8	ReLU
	Dense	8,1	Sigmoid

Table 10: Model Descriptions for Binary Classifiers 50% Feature Space

Model	Layer Type	Kernel	Activation
MLP	Input	0,22,	-
	Dense	22,33	Tanh
	Dense	33,1	Sigmoid
CNNv1	Input	0,37,1	-
	Conv1D	3,1,160	ReLU
	MaxPool1D	2	-
	Conv1D	3,160,224	ReLU
	MaxPool1D	2	-
	Flatten	-	-
	Dense	1568,384	ReLU
	Dense	384,1 Sigmoid	
CNNv2	Input	0,37,1	-
	Conv1D	3,1,8	ReLU
	MaxPool1D	2	-
	Conv1D	3,8,104	ReLU
	MaxPool1D	2	
	Conv1D	3,104,40	ReLU
	MaxPool1D	2	
	Dropout	rate=0.5	-
	Flatten	-	-
	Dense	80,72	ReLU
	Dropout	rate=0.5	-
	Dense	72,136	ReLU
	Dense	136,72	ReLU
	Dense	72,1	Sigmoid
AE	Input	0,37	-
	Dense	37,168	Tanh
	Dense	168,104	Tanh
	Dense	104,200	Tanh
	Dense	200,200	Tanh
	Dense	200,232	Tanh
	Dense	232,1	Sigmoid
AE+CNN	Input	0,37	-
	Dense	37,200	ReLU
	Dense	200,168	ReLU
	Dense	168,104	ReLU
	Dense	104,72	ReLU
	Dense	72,200	ReLU
	Reshape	(200, 1)	-
	Conv1D	3,1,256	ReLU
	MaxPool1D	1	-
	Conv1D	3,256,64	ReLU
	MaxPool1D	1	-
	Flatten	-	-
	Dense	12544,8	ReLU
	Dense	8,1	Sigmoid

Table 11: Model Descriptions for Binary Classifiers 30% Feature Space

Model	Layer Type	Kernel	Activation
MLP	Input	0,22	-
	Dense	22,33	Tanh
	Dense	33,1	Sigmoid
CNNv1	Input	0,22,1	-
	Conv1D	3,1,288	ReLU
	MaxPool1D	2	-
	Conv1D	3,288x,320	ReLU
	MaxPool1D	2	-
	Flatten	-	-
	Dense	1280,480	ReLU
	Dense	480,1	Sigmoid
CNNv2	Input	0,22,1	-
	Conv1D	3,1,72	ReLU
	MaxPool1D	2	-
	Conv1D	3,72,104	ReLU
	MaxPool1D	2	
	Conv1D	3,104,200	ReLU
	MaxPool1D	2	
	Dropout	rate=0.5	-
	Flatten	-	-
	Dense	200,232	ReLU
	Dropout	rate=0.5	-
	Dense	232,104	ReLU
	Dense	104,40	ReLU
	Dense	40,1	Sigmoid
AE	Input	0,22	-
	Dense	22,72	Tanh
	Dense	72,168	Tanh
	Dense	168,232	Tanh
	Dense	232,40	Tanh
	Dense	40,40	Tanh
	Dense	40,1	Softmax
AE+CNN	Input	0,22	-
	Dense	22,40	ReLU
	Dense	40,136	ReLU
	Dense	136,104	ReLU
	Dense	104,40	ReLU
	Dense	40,232	ReLU
	Reshape	(232, 1)	-
	Conv1D	3,1,192	ReLU
	MaxPool1D	1	-
	Conv1D	3,192,32	ReLU
	MaxPool1D	1	-
	Flatten	-	-
	Dense	7296,168	ReLU
	Dense	168,1	Sigmoid

9.5.2 Multiclass Classifier Architectures

Table 12: Model Descriptions for Multiclass Classifiers 100% Feature Space

Model	Layer Type	Kernel	Activation
MLP	Input	0,74	-
	Dense	74,128	Tanh
	Dense	128,4	Softmax
CNN v1	Input	0,74,1	-
	Conv1D	3,1,232	Tanh
	MaxPooling1D	-	-
	Conv1D	3,232,232	Tanh
	MaxPooling1D	-	-
	Dropout	rate=0.35	-
	Flatten	-	-
	Dense	3944,72	Tanh
	Dense	72,4	Softmax
CNNv2	Input	0,74,1	-
	Conv1D	3,1,232	Tanh
	MaxPooling1D	-	-
	Conv1D	3,232,104	Tanh
	MaxPooling1D	-	-
	Conv1D	3,104,40	Tanh
	MaxPooling1D	-	-
	Dropout	rate=0.25	-
	Flatten	-	-
	Dense	280,296	Tanh
	Dropout	rate=0.25	-
	Dense	296,456	Tanh
	Dense	456,168	Tanh
	Dense	168,4	Softmax
AE	Input	0,74	-
	Dense	74,168	Tanh
	Dense	168,104	Tanh
	Dense	104,104	Tanh
	Dense	104,40	Tanh
	Dense	40,104	Tanh
	Dense	104,4	Softmax
AE+CNN	Input	0,74	-
	Dense	74,136	Tanh
	Dense	136,104	Tanh
	Dense	104,72	Tanh
	Dense	72,40	Tanh
	Dense	40,104	Tanh
	Reshape	(104,1)	-
	Conv1D	3,1,32	Tanh
	MaxPooling1D	-	-
	Conv1D	3,32,96	Tanh
	MaxPooling1D	-	-
	Flatten	-	-
	Dense	9600,168	Tanh
	Dense	168,4	Softmax

Table 13: Model Descriptions for Multiclass Classifiers 50% Feature Space

Model	Layer Type	Kernel	Activation
MLP	Input	0,37	-
	Dense	37,96	Tanh
	Dense	96,4	Softmax
CNN v1	Input	0,37,1	-
	Conv1D	3,1,136	Tanh
	MaxPooling1D	-	-
	Conv1D	3,136,232	Tanh
	MaxPooling1D	-	-
	Dropout	rate=0.25	-
	Flatten	-	-
	Dense	1624,104	Tanh
	Dense	104,4	Softmax
CNNv2	Input	0,37,1	-
	Conv1D	3,1,264	Tanh
	MaxPooling1D	-	-
	Conv1D	3,264,296	Tanh
	MaxPooling1D	-	-
	Conv1D	3,296,456	Tanh
	MaxPooling1D	-	-
	Dropout	rate=0.25	-
	Flatten	-	-
	Dense	912,136	Tanh
	Dropout	rate=0.25	-
	Dense	136,360	Tanh
	Dense	360,40	Tanh
	Dense	40,4	Softmax
AE	Input	0,37	-
	Dense	37,136	Tanh
	Dense	136,136	Tanh
	Dense	136,136	Tanh
	Dense	136,200	Tanh
	Dense	200,72	Tanh
	Dense	72,4	Softmax
AE+CNN	Input	0,37	-
	Dense	37,232	Tanh
	Dense	232,200	Tanh
	Dense	200,8	Tanh
	Dense	8,40	Tanh
	Dense	40,168	Tanh
	Reshape	(104,1)	-
	Conv1D	3,1,64	Tanh
	MaxPooling1D	-	-
	Conv1D	3,64,64	Tanh
	MaxPooling1D	-	-
	Flatten	-	-
	Dense	10496,168	Tanh
	Dense	168,4	Softmax

Table 14: Model Descriptions for Multiclass Classifiers 30% Feature Space

Model	Layer Type	Kernel	Activation
MLP	Input	0,22	-
	Dense	22,128	Tanh
	Dense	128,4	Softmax
CNN v1	Input	0,22,1	-
	Conv1D	3,1,136	Tanh
	MaxPooling1D	-	-
	Conv1D	3,136,232	Tanh
	MaxPooling1D	-	-
	Dropout	rate=0.35	-
	Flatten	-	-
	Dense	928,72	Tanh
	Dense	72,4	Softmax
CNNv2	Input	0,221	-
	Conv1D	3,1,200	Tanh
	MaxPooling1D	-	-
	Conv1D	3,200,72	Tanh
	MaxPooling1D	-	-
	Conv1D	3,72,392	Tanh
	MaxPooling1D	-	-
	Dropout	0.25	-
	Flatten	-	-
	Dense	392,296	Tanh
	Dropout	0.25	-
	Dense	296,232	Tanh
	Dense	232,232	Tanh
	Dense	232,4	Softmax
AE	Input	0,22	-
	Dense	22,232	Tanh
	Dense	104,232	Tanh
	Dense	104,232	Tanh
	Dense	232,40	Tanh
	Dense	40,232	Tanh
	Dense	232,4	Softmax
AE+CNN	Input	0,22	-
	Dense	22,232	Tanh
	Dense	232,200	Tanh
	Dense	200,200	Tanh
	Dense	200,72	Tanh
	Dense	72,72	Tanh
	Reshape	(104x1)	-
	Conv1D	3,1,224	Tanh
	MaxPooling1D	-	-
	Conv1D	3,224,64	Tanh
	MaxPooling1D	-	-
	Flatten	-	-
	Dense	4352,200	Tanh
	Dense	200,4	Softmax