A Comparative Evaluation of Deep Learning Approaches to Online Network Traffic Classification in Community Networks

Matthew Dicks (DCKMAT004)
University of Cape Town
Cape Town, South Africa
dckmat0041@myuct.ac.za

ABSTRACT

Community networks are networks that are run by the citizens for the citizens. Communities build, operate and own open IP-based networks, which promotes individual and collective participation. These networks run with limited resources compared to traditional Internet Service Providers. Understanding how resources are used is necessary to ensure improved Quality of Service (QoS). The purpose of this paper is to determine whether deep learning can be used for traffic classification to help improve the QoS and traffic engineering in community networks. To do this, the deep learning architectures chosen were tested over a range of time and memory constraints, and they have been compared to a traditional machine learning model. After performing the experiments it was shown that deep learning is the best way to perform traffic classification in a low resource environment. The models provided prediction speeds fast enough to be used for real time classification, and they had a higher test set accuracy than the Support Vector Machine when measured over all memory constraints. The paper also shows that the Long Short-Term Memory architecture significantly outperformed the Multi-Layer Perceptron because it was able to learn the sequential aspects of the packet's byte stream.

CCS CONCEPTS

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

KEYWORDS

Network traffic classification, deep learning, community networks

1 INTRODUCTION

Traffic classification is the process of categorizing network traffic into appropriate classes. In this project, the classes will represent applications such as Gmail, YouTube and Facebook.
Traffic classification has become a vital tool used in advanced network management, especially in the context of Quality of Service (QoS) control, pricing, resource usage planning as well as malware and intrusion detection. This project aims to utilize deep learning to perform traffic classification for the purpose of improving QoS and traffic engineering in community networks. QoS engineering works by prioritizing the traffic of some applications over others depending on the network's requirements [2]. For a QoS system to prioritize some traffic, it needs to first classify the traffic into the appropriate classes. This is why good traffic classification is vital for improving QoS systems.
Previous traffic classification methods used port numbers, Deep Packet Inspection (DPI) or features hand crafted by an expert. There are a number of issues with each of these approaches. Using port numbers for traffic classification is the simplest and fastest way to classify internet traffic. However, due to port obfuscation, random port assignments, port forwarding, protocol embedding and network address translation (NAT), the accuracy of port-based methods has decreased [13]. The accuracy of DPI methods has decreased due to the increase in the amount of encrypted traffic and user privacy agreements. DPI also has a large computational overhead [15], which is not suited to real time classification. Features that have been handcrafted by an expert can suffer a lack of generality because they focus on only a few key features. *Aceto et. al.* [1] notes that these hand-crafted features rapidly become outdated due to the evolution and mix of internet traffic. It is also an expensive method because experts have to be hired and the hand picking procedure is subject to human error [1].
Deep learning is a proposed solution to these issues. Deep learning can learn a hierarchical set of features from high dimensional raw packet data. It therefore removes the need for features to be hand crafted by an expert, and promises to be able to classify encrypted traffic. A system can leverage the power of this approach to build an end-to-end deep learning tool. This will limit the need for feature selection and engineering, which would reduce the amount of prepossessing and overhead. Deep learning includes powerful architectures such as recurrent neural networks that can learn temporal patterns in data. This will become useful because packet data is inherently sequential.

1.1 Project Aims

This project aims to answer three key research questions. Firstly, which of the deep learning architectures provides the highest classification accuracy subject to the resource constraints found in a community network? Secondly, does the more complex Long Short-Term Memory (LSTM) network sufficiently outperform the simpler Multi-Layer Perceptron (MLP), assuming the same computational constraints? Lastly, does the best deep learning network sufficiently outperform a traditional machine learning model, the Support Vector Machine (SVM), assuming the resource constraints found in a community network? The success of this project will be determined by how well these research questions were answered.
To answer these research questions the project aims to build a data pipeline that can take a set of pcap files, and from the pcap files create a dataset that will be used to train and test machine and deep learning models. There will be three model types implemented the SVM, MLP and LSTM. The comparison of the models' performances aims to answer these research questions.

1.2 Structure of Report

The paper will start out by giving an explanation on what a community network is and why resource efficient traffic classification techniques are vital in that context. It will then give a theoretical background on the models chosen and will introduce some of the state of the art research that has been conducted in the field of traffic classification. The paper will then describe the design and execution of the experiments as well as the results. The paper will conclude with a summary of key findings and a discussion on limitations of the paper and future work that would be of use to this field.

2 COMMUNITY NETWORKS

A Community network is a network that is run by the citizens for the citizens, where communities build, operate and own open IP-based networks [3]. These networks promote individual and collective digital participation in rich, poor, rural and urban areas [14]. They also have minimal barriers to entry because their governance, knowledge and ownership is open [14]. These networks are usually run by non-profit organizations who can interact with stakeholders to provide services such as local networking, voice, data and internet access [14]. Community networks are large-scale distributed and decentralized systems with many nodes, links, services and traffic. They are also characterized by their dynamic and heterogeneous behavior [14].

2.1 The potential for traffic classification in community networks

Community networks provide a wide range of application services such as VoIP, content distribution, on-demand and live streaming media, instant messaging as well as back-ups and software updates. The challenge often is that they have to provide these services on links and servers with limited capacity. They also have to manage the diverse and volatile resources in the network [3]. Therefore, to be able to provide a good QoS to the users, network managers should be able to know how their resources are being used. This is where traffic classification comes in, as it provides a way for the citizens and non-profit organizations who are running the network to view what applications are using different resources. This allows them to take more informed actions to try and improve the QoS delivered to their customers. Efficient, real time traffic classification is also vital to ensure the correct traffic is prioritized.
The end goal of this project is to perform traffic classification with deep learning models. These models have a large number of parameters, which makes training and running them computationally expensive. Community networks try to provide their services at a low cost and one of the ways they do this is to use cheaper hardware and resources [3]. Since these models will need to be deployed in low resource environments, they will need to be evaluated not just based on their predictive performance, but also on their computational efficiency.

3 BACKGROUND ON THE MACHINE LEARNING AND DEEP LEARNING MODELS

This section gives a theoretical explanation of each of the models. It also motivate why each model was chosen.

3.1 Machine Learning Model

The Support Vector Machine (SVM) was chosen as the baseline machine learning model. The SVM was developed by the computer science community in the 1990s and has proven to work well in a variety of settings [8]. It is often considered as one of the best out-of-the box classifiers [8].
The Support Vector Classifier (SVC) was the first iteration of the SVM model and it was used to perform two class classification. The SVC models each training example as a point in p dimensional space, where p is the number of predictor variables. Then the algorithm tries to find a p-1 dimensional hyper-plane that linearly separates the two classes, assuming that the classes are linearly separable [5]. Usually the observations in the two classes are not perfectly separable by a linear plane, some observations in one class might fall into the other class's side of the plane. The SVC solves this problem by introducing a soft margin, which allows some observations to be miss-classified [8]. The sensitivity of the model to miss-classified observations is controlled by the hyperparameter C, which will be searched over in the model building process [8]. This parameter also controls the degree to which the model overfits to the data.
What if the data is not linearly separable? This is where an SVC turns into an SVM. An SVM will first transform the feature space using either quadratic, cubic or even higher-order functions of the predictors [8]. These functions are called kernels. If a space is not linearly separable, then a kernel is used to create a new enlarged feature space where the observations are now linearly separable, and an SVC can be fit to this new feature space [8].
Since this project is dealing with multi-class classification, the SVM will have to be able to classify multiple classes. There are two ways in which an SVM can be used to perform multi-class classification and that is one-versus-one or one-versus-many. Due to the emphasis that this project puts on efficiency the one-versus-many approach will be used. For K class classification, K SVM models are trained. To predict the class to which an observation belongs the SVM with the highest value is taken [8].
The SVM model was chosen as a baseline machine learning model for this project because it has the ability, unlike some of the other machine learning models, to perform well on high dimensional data. And since our feature space will have 1480 dimensions, this performance was necessary. The SVM model is also able to use the same type of data that will be used in the deep learning models, which allows for a good comparison.

3.2 Deep Learning Models

3.2.1 *Multi-layer Perceptron.*

The Multi-Layer Perceptron is the quintessential deep learning model and forms the basis for many commercial applications. For example, recurrent neural networks, which will be described in the next section, is formed by a sequence of MLPs [6]. An MLP is a directed acyclic graph that consists of a set of nodes and edges, see Figure 1. The nodes are either the input values, if they are at the input layer, or they are the activation functions. The edges are the weights that form the parameters of the model to be learned. Each node, besides the input nodes, takes a weighted sum of all the nodes in the previous layer as input, and then passes this through a non-linear activation function. In a classification task, the outputs of the last layer are fitted with a softmax function and the neuron, which represents a category, with the highest probability is taken. It can be seen that these models have a large amount of parameters and therefore training and deploying these models is very computationally intensive. Due to this complexity and their low accuracy, these models have not been widely used in traffic classification [15]. Due to their prevalence in the deep learning domain and the fact that it forms the basis for all other deep learning models, this model type was chosen as the baseline deep learning model.

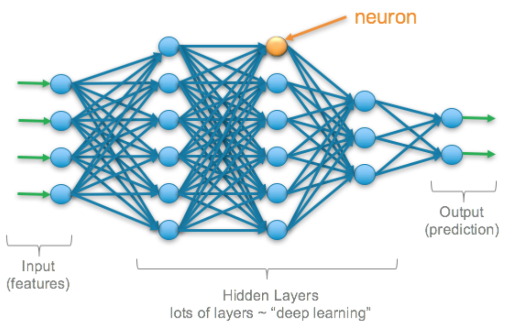

Figure 1: MLP

3.2.2 *Long Short-Term Memory Network.*

Recurrent Neural Networks (RNNs) are used for processing sequential data, where the current state may depend not only on the current input, but also previous inputs, for example language translation and time series predictions [6]. To be able to model these sequences, the network contains loops, as shown in Figure 3. These loops allow RNNs to scale to longer sequences than otherwise would be possible if a different model architecture was used [6]. These networks also take advantage of parameter sharing because they look for information that could occur in multiple locations. Parameter sharing drastically reduces the memory requirements because it reduces the number of parameters that need to be optimized. The problem with RNNs is that they cannot learn long term dependencies due to the vanishing and exploding gradients problem that comes from applying the same function over and over again [6]. To solve this problem, Long Short-Term Memory (LSTM) networks were created. These are gated RNNs where the gates are used by the network to remember or forget states depending on the current time step. This is useful because it allows the gradients to propagate further into the future without exploding or vanishing [6]. In traffic classification approaches, LSTMs are used instead of vanilla RNNs because they can learn these long term dependencies [11, 18]. They are particularly useful when classifying flows into categories because they can learn the temporal patterns between individual packets. This model type has not been used to classify individual packets, but because the bytes in a packet are inherently sequential, this model type may be able to outperform MLPs and other types of networks due to its ability to learn these sequential patterns. The LSTM structure is inherently sequential, which makes them very hard to train and run in parallel. Therefore, they should have longer training and prediction times, compared to other networks. This could use up vital resources in a resource constrained community network.

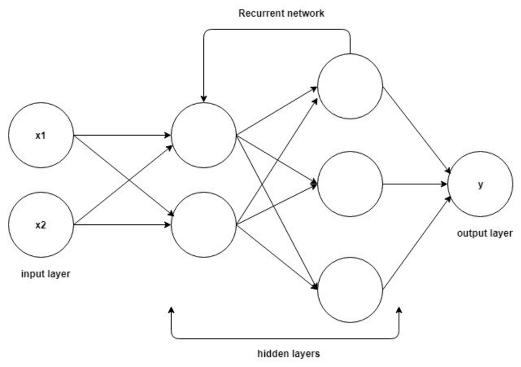

Figure 2: RNN

4 RELATED WORK

This section discusses how deep learning and machine learning have been used in traffic classification. There are two ways to perform traffic classification namely, flow classification and packet classification. Therefore, this section is split into two sections that will discuss the two different approaches.

4.1 Flow Classification

More often than not, the objects of classification are flows [1]. In the literature, there are four prominent approaches for collecting data from a flow [1, 4, 11, 18, 19]. The first approach is to take raw data, in the form of bytes, from some of the packets in the flow [19]. Another approach is to extract raw data from a flow. This mean that you only consider the first N bytes from the flow and you do not care about individual packets [18]. The third approach uses time series data like packet sizes, packets directions and inter-arrival times from individual packets [15]. Flow statistics is the fourth way that data can be extracted from a flow. Examples of flow statistics are means, standard deviations as well as minimums and maximums for packet sizes and inter-arrival times. This approach needs to use more packets from a flow so that estimates do not have too much variance. *Liu et. al.* [15] notes that this may not be suitable for fast real time classification.

Aceto et. al. [1] found that using raw data was better than using hand picked time series features and flow statistics [1]. These findings have shown that increasing the raw information available to the deep learning models results in greater prediction accuracy.

Due to the structure of the data extracted from flows, new model architectures become useful, such as the LSTM and one dimensional Convolutional Neural Network (1D-CNN). These deep learning architectures can find long and short term temporal relationships in the data. The 2D-CNN also gets used to find spatial patterns.

A 1D-CNN and a 2D-CNN were used to classify flows using the first 784 or 1000 bytes of the flow. These bytes are converted either into a 1D vector for the 1D-CNN, or into a 2D image for the 2D-CNN. *Wang et. al.* [19] did not specify how they transformed the inputs into those vectors. The 1D-CNN had an accuracy of 91.25% and the 2D-CNN had and accuracy of 90% [19]. This is not a surprising result since 1D-CNNs are better suited to processing sequential data [10]. *Wang et. al.* [19] also compared the 1D-CNN to the state of the art C4.5 decision tree. The CNN's average recall was 8.1% higher than the average recall of the decision tree [19]. This showed that end-to-end deep learning was better than state-of-the-art machine learning. *Aceto et. al.* [1] also found that 1D-CNNs outperform 2D-CNNs. As mentioned before LSTMs are used to find long term dependencies between inputs and it is currently one of the best models to use when processing sequential data. *Aceto et. al.* [1] showed this to be true by observing that an LSTM model outperformed a 2D-CNN. As expected, the MLP had worse results than the other three deep learning models. A surprising result out of this study was that the LSTM had the most efficient training time [1]. This is a surprising result because the training and running of an LSTM cannot be done in parallel, but most of the other architectures can be run in parallel.

In the studies presented by [1, 19], the 2D-CNN model was not performing as hoped. *Chen et. al.* [4] took a different approach and encoded the static as well as the dynamic aspects of the flow into an image. They used the first nine packet sizes, inter-arrival times and packet directions as input data. This data is encoded as $I_1, I_2, ..., I_N$, where I_j is either the jth packet size, direction or inter-arrival time. To understand the static and dynamic properties of the flow they estimated the marginal distributions, $P(I_j)$, and the conditional distributions, $P(I_{j+1}|I_j)$, of the flow statistics. To estimate the distributions they used the Reproducing Kernel Hilbert Space embedding. This is an efficient and non-parametric approach to represent these distributions [4]. Since there are three statistics per packet there are six distribution estimates related to each packet. They transformed a list of packets into a six channel image. They then used a 2D-CNN to classify the images into application types and achieved an accuracy of 99.84%, which far outperformed the Support Vector Machine (SVM), MLP, Naive Bayes (NB) and decision tree models. They also tested the 2D-CNN on real world data and obtained an accuracy of 88.42%, which was a over 11% higher than the next best model [4]. This approach saw the 2D-CNN work really well since there was a well thought out and a well executed way of transforming flow data into an image.

So far, studies have used 2D-CNN models to find spatial patterns and have used LSTM models to find the temporal patterns. There have been approaches that try to combine the two models to learn both spatial and temporal patterns [11, 18]. *Lopez-Martin et. al.* [11] used the first 20 packets and 6 features from each packet to make a 20x6 matrix, with the rows as the time dimension and the columns as the feature dimension. They passed that matrix into a 2D-CNN-LSTM network. The output of the CNN part is a 3D matrix, with the extra dimension coming from the number of filters. Since the LSTM accepts a 2D matrix, this matrix was squashed, along the feature axis to preserve the time and the filter dimensions which was past into the LSTM. The 2D-CNN-LSTM model obtained an F1 score of 96%, which was an improvement over the standard 2D-CNN and LSTM models, which achieve F1 scores of 94.5% and 95.5% respectively. The plain LSTM model seemed to capture the same information as the 2D-CNN-LSTM [11]. This may be due to the CNN compromising the time dimension of the input matrix by passing over it with 2D kernels. Therefore, the LSTM part of the model did not have a meaningful sequence to learn.

Another approach that did preserve the time aspect of the data was taken by *Huang et. al.* [18]. They took 100 bytes from each of the first 6 packets. They converted the data from each packet into an image until they ended up with 6 images. They used 6 CNN models, one for each packet to extract the spacial features out of the images. Each CNN model had a dense last layer, which created a feature vector for each packet, thus preserving the time dimension. These feature vectors were then run into the LSTM part of the model to classify the flow. The CNN-LSTM model had an accuracy of 99.89% and outperformed the vanilla CNN network [18]. By splitting up the data into 6 images and processing them separately, they allowed the output of the CNNs to be stacked in a manner that would preserve the order of the packets and, therefore, the time. This is a much more suitable input for the LSTM part of the model when you compare it with the previous study.

4.2 Packet Classification

A more fined grained approach would be to classify individual packets. A number of studies show that individual packet classification is possible and can yield very good results [12, 17].

When doing individual packet classification times series features are not useful since the focus is on individual packets. This means that individual packet classification is difficult, but deep learning offers a solution. Since, deep learning has the ability to learn high dimensional data [15], it can therefore learn from the raw data of a packet.

Lotfollahi et. al. [13] used the first 1480 bytes of the IP payload as well as the IP header as input. They masked the IP addresses because they only used a limited number of hosts and servers [13]. This did not allow the model to use the information provided by the IP addresses which would have caused unreliable results. In a similar study, *Chen et. al.* [17] used the same data as *Lotfollahi et. al.* [13] but disregarded the IP header.

One of the most successful deep learning architectures is the Convolutional Neural Network (CNN). This model has lead to good performance in image recognition and object detection. Normally, CNNs are used on 2D images but they can be adapted to be used on 1D vectors. These 1D-CNNs can learn sequential patterns in these 1D vectors. *Lotfollahi et. al.* [13] used a 1D-CNN to classify individual packets into classes and obtained an F1 score of 98% in application classification, and an F1 score of 93% in traffic categorization. The study also used a Stacked Auto-encoder (SAE) to classify internet traffic [13]. Auto-encoders are used as an unsupervised learning algorithm to try an generate output that is as close to the input as possible. This allows the model to learn a more comprehensive feature set that can be used to train supervised learning models [9]. The SAE was trained and then fitted with a soft-max layer to classify the traffic. The model performed slightly worse than the 1D-CNN and achieved F1 scores of 95% and 92% for application classification and traffic characterization respectively [13]. A similar done study by *Chen et. al.* [17] on application

classification saw the CNN and SAE achieve F1 scores of 98.4% and 98.8% respectively. They also compared that with an MLP which had an F1 score of 96.5%. The MLP was a smaller model with only two hidden layers and six neurons each, which probably caused the reduced performance. In both of these studies, the input to the models was a 1D vector that corresponded to the first 1480 bytes of the IP payload data, but in [13] they also added the IP header.
This project aims to determine whether deep learning can be used to perform traffic classification for the purpose of improving the QoS and traffic engineering. To improve the QoS, packets will have to be classified in close to real time so that the QoS algorithm can prioritize them effectively. This need for real time classification of packets means that there is no time to wait for multiple packets to pass though so that a flow can be classified. There is also no way to tell apriori how many packets are going to be in a flow and therefore there is no way to tell how long to wait before the flow must be classified. This is why this project is using packet classification instead of flow classification. There is also far less research done on packet classification, which means that this project will likely yield novel results.

5 EXPERIMENT DESIGN AND EXECUTION

5.1 Obtaining Network Traffic Data

The first step for the project involves accessing the community network traffic data to be used for training and testing the models. We use a dataset that was collected from the Ocean View community network in Cape Town [7]. The data consists of numerous PCAP files that were collected at the gateway of the network, capturing all traffic flowing between the network and the Internet from February 2019. The PCAP files have been copied to a data repository at the University of Cape Town, through which we access the data.

5.2 Preprocessing

This section describes the preprocessing phase. It explains how the PCAP files are transformed into data that can be used to train and test the models

5.2.1 *Flow extraction.*

In each of the PCAP files that were collected, there are numerous flows. Flow extraction is the process of splitting up a PCAP file into smaller PCAP files, where each file contains a single flow. Working with these smaller files made the rest of the preprocessing far easier. It was easier to label the packets in a file containing only a single flow than it was to label packets in the file containing multiple flows. It also helped in the data extraction phase. The library used to perform this function was called *pkt2flow*[1]

5.2.2 *Labeling the packets.*

Classification is a supervised learning task. This means that each example must have a label associated with it, which will allow the neural networks and the SVM to learn from their errors and adjust their parameters accordingly. The packets in the PCAP files do not have labels associated with them already, so we have to provide these labels. The labeling phase takes as input the PCAP files containing single flows and then uses an open-source deep packet inspection library called *nDPI* to label each flow. Each packet associated with a given flow will receive that flow's label.

5.2.3 *Feature extraction and transformation.*

The features used as input into the models are the bytes extracted from the IP payload for each packet. This method was chosen because it enables the use of both encrypted and unencrypted packets. There is also evidence to suggest that using this data as input into deep learning models in the context of traffic classification can yield high prediction accuracy [12, 17].
Python has an open source library called *scapy* that was used to process the packets and flows found in PCAP files. The library has methods that can extract the IP payload from packets and convert it into bytes. These methods were used to obtain the features needed for each packet.
Once the raw data has been extracted from the packets, it has to be transformed into an appropriate format so that a given model can learn from it. The number of bytes in the IP payload is variable, with the maximum number of bytes being 1480. The models to be used need to have the same number features for each packet. It was decided that each packet should have a feature vector of length 1480 and those packets that have less bytes should be padded with zeros. The data was normalized to increase the learning algorithm's stability and decrease the training time, which could save valuable resources in the community network. Due to the limited number of hosts and servers, the first 20 bytes of this 1480 byte vector were masked. This made sure that the models were not learning information relating to the port numbers used in that network but they were learning general patterns found in the IP payload.
For the SVM and the MLP models, the 1D normalized byte stream is an appropriate feature vector. The LSTM requires this byte stream to go through one more transformation. The LSTM model needs the byte stream to be broken up into multiple time steps. This means that the 1D vector has to be transformed into a 2D vector, with the first dimension giving the number of time steps and the second dimension giving the number of observations in a given time step. The obvious way to break up the feature vector would be to consider each of the bytes as an individual time step, but this would create a sequence with length 1480. As discussed before, it is difficult for LSTM models to learn such long sequences due to the vanishing and exploding gradients problem [6]. To solve this problem the feature vector was split up into 40 time steps with each time step consisting of 37 bytes. This reduced the length of the sequence and allowed the gradients to flow back through the network, which increased the model's learning capacity. The reduced sequence length also reduced the number of sequential steps taken by the model, which reduced the model's training and prediction times.

5.3 The Datasets

Once the preprocessing stage was complete, the final dataset consisted of ten classes, each with ten thousand observations. Table 1, summarises the full dataset.
This dataset then gets further subdivided into a training set, a validation set and a test set. The percentage of observations are as follows; 64% is reserved for training the models; 16% is used for validation and hyperparameter tuning; and the last 20% is used as

[1] *pkt2flow* is a simple utility that classifies packets into flows. It takes a single PCAP file as input and returns a set of PCAP files where each file contains a single flow. It is available at: https://github.com/caesar0301/pkt2flow.

Table 1: The set of applications that will be used for classification and the number of observations in each application class.

Class	Number of Observations
WhatsApp	10 000
GoogleServices	10 000
Instagram	10 000
PlayStore	10 000
TeamViewer	10 000
BitTorrent	10 000
WindowsUpdate	10 000
GMail	10 000
Facebook	10 000
YouTube	10 000
Total Observations	**100 000**

the test set. These datasets were created by randomly sampling from the original dataset to ensure that the distributions in the three datasets remain as similar as possible.

5.4 Evaluating the Performance of the Machine Learning and Deep Learning Models

5.4.1 Metrics used for predictive performance.

The models will be evaluated based on two characteristics, namely predictive performance and computational efficiency. The metric used for predictive performance is accuracy, and the formula is as follows:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \times 100$$

5.4.2 Validation Set.

The validation dataset consists of 16% of the full dataset. K-fold cross-validation was not used because it would be too computationally expensive and there was enough data to perform validation analysis on a single validation set. After the training process is complete, this data is used to evaluate the model and give an estimate for the accuracy expected on the test set. This estimate can also be used to test for overfitting, because if there was a vast decrease in the accuracy between the training error and the validation error then this would be an indicator of overfitting. The validation dataset was also used in the hyperparameter tuning process that will be described in Section 5.5.

5.4.3 Test Set.

The validation error rate provides a good estimate for the model's predictive performance, but this set was used to pick the hyperparameters. Therefore, the models would have seen this data. To get a more realistic measure of performance, a test set was created to give an estimate for the accuracy expected on unseen data. Once a model has been trained and the hyperparameters have been chosen, the definitive measure of predictive performance will be the accuracy computed on the test set. This will give us an indication of each model's ability to generalize to new examples and the accuracy computed on the test set will be used to compare each model's predictive performance.

Table 2: This table gives a break down of the range of parameters used in the deep learning models.

15 000	30 000	50 000	100 000	200 000
300 000	350 000	400 000	500 000	600 000
700 000	800 000	900 000	950 000	1 000 000

5.4.4 Metrics used for computational efficiency.

The computational efficiency of the algorithms will be measured in two ways – the time it takes to make a prediction, and the amount of memory needed to make a prediction. Since the algorithms are going to be deployed to produce real-time classifications, the speed of the predictions is key. Therefore, we will be using the average time it takes a model to make a prediction as a metric to analyze computational efficiency. This will be computed by generating sample packets and timing how long the model takes to classify these individual samples. The average time over all these samples will be used as the metric to compare the prediction speed between models. The average time it takes to make a prediction is inversely proportional to the number of packets classified per second, this will also be used to compare the models' prediction speeds.
The amount of memory needed by a model to make a prediction is directly proportional to the number of parameters in the model. So, as the number of parameters increase, the amount of memory needed to make a prediction will increase because more memory will be needed to store the model. The number of parameters in a model will be used as the metric to compare the memory usage.

5.4.5 Overview of the Experiment.

The deep learning models need to be compared subject to memory and time constraints. This means that the predictive performance of these models will need to be evaluated at each of these constraints. For each of the deep learning model architectures, MLP and LSTM, the number of parameters in the architecture will be varied and the test accuracy will be calculated. The number of parameters in the architecture will range from 15 000 to 1 000 000, Table 2 shows the break down of this range. This will allow the models to be compared at the highest and lowest memory requirements. So if a community network can only have a model with 100 000 parameters it will be easy to see which architecture will perform the best. At each of the different parameter levels, the average time it takes to make a prediction and the number of packets classified per second will be calculated. This will allow time constraints to be placed on the models, which will make it easy to see which type has the best accuracy under these constraints. It will also determine whether the deep learning models can support real time classification.
The SVM model has a constant number of parameters so the test accuracy and the average time it takes to classify a packet will be reported for only the single best performing SVM.

5.5 Hyperparameter Tuning

Hyperparameters are values that are set before the training of the networks and therefore are not learned during the training process. Some examples of hyperparameters in neural networks include the network topology, the learning rate of the optimization algorithm, as well as the training batch size and the number of training epochs. Since these parameters have to be defined prior to training, the best way to find the parameters will be to systematically try out different combinations and then pick the best set. The best set will be determined by the accuracy obtained on the validation set. Due to the change of the network topology brought about by varying the number of parameters, the topology will not be a hyperparameter that will be searched. The batch size was selected to optimize the training time, and the number of epochs was determined by the time it took to reach convergence. Early stopping was used in the MLP to limit overfitting when the number of epochs proved to be too much. This means that for the deep learning models the only other hyperparameter that needed to be searched for was the learning rate. For the MLP, three learning rates were checked at every parameter level, namely 0.001, 0.0005 and 0.0001. The LSTM needed higher learning rates to achieve convergence and the three rates searched over were 0.01, 0.005 and 0.001.

5.6 Implementation of Models

The deep learning networks were built in Python using Tensorflow's implementation of the Keras API. Keras was chosen because it provides easy ways of building networks without sacrificing flexibility. It also supports use of input pipelines built in Tensorflow, which will decrease the memory requirements when training the models.

The Support Vector Machine (SVM) was built with Python's scikit-learn library. The decision to use this library was on the basis that scikit-learn includes the ability to make multi-class classifications with an SVM model. This is important because of the need to classify traffic into multiple application classes.

The LSTM and the MLP models were all run in Google's Colab because they allow free use of GPUs. This was needed because a large number of models needed to be trained and GPUs offered a great speed up over CPUs. Google Drive was used to store the datasets and all the experimental results. This was chosen because it integrated better with Colab when compared with local storage.

6 EXPERIMENTAL RESULTS

As stated in Section 5.4.5, the number of parameters for each architecture was varied and at each parameter level, the test accuracy as well as the average time to make a prediction was calculated. This section will present the results obtained by running these experiments.

The first research question was to determine whether the LSTM could attain a higher accuracy when compared with the MLP. The second research question was to check that the increase in accuracy was significant enough to warrant using the added complexity. Figures 3 and 4 present the results obtained trying to answer these questions.

In both plots, the points represent the accuracy and the packets classified per second for each parameter level that was described in Table 2. In Figure 4 the prediction speed was measured in packets per second which was calculated by the following formula:

$$Packets\ per\ second = \frac{1}{Average\ time}$$

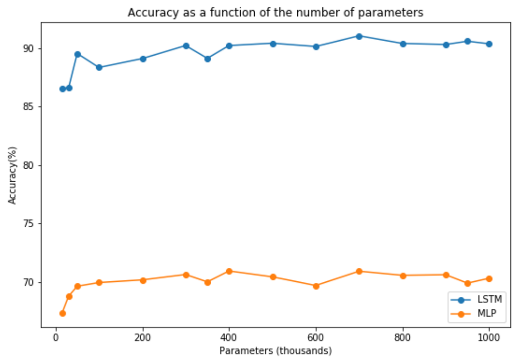

Figure 3: LSTM vs MLP

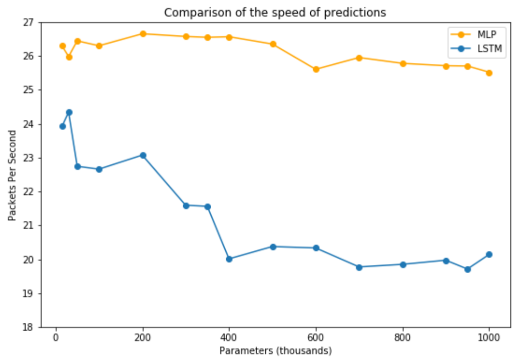

Figure 4: LSTM vs MLP (Speed comparison)

Figure 3 makes it clear that the LSTM out performs the MLP in predictive power across all parameter levels. The LSTM's accuracy ranges form 86.5% to 91% and the MLP's accuracy ranges from 66.8% to 70.5%. There is some noise in the accuracy levels because the optimization process for neural networks is stochastic but the difference between these two models is large enough to say that the LSTM performs better for every memory constraint. This was expected because the LSTM is far more suited to learning sequential information, and the byte stream of a packet is inherently sequential. It is also good to note that there does seem to be a general relationship between the number of parameters and the accuracy. With the accuracy increasing at a decreasing rate as the number of parameters increase. Initially, as the number of parameters increase, the model's accuracy drastically jumps up but, over time, the accuracy starts to plateau. Increasing the number of parameters past a certain point does not lead to a sufficient increase in accuracy because the models have already extracted most of the variation in the data. Increasing the parameters further will likely lead to overfitting and a degrading of the performance on the test set.

Figure 4 shows the results from the speed tests done for the MLP and the LSTM. The results are presented as the number of packets that can be classified in a second. It can be seen that there is some noise in the data, this comes from the stochastic nature of timing how long it takes to run a section of code. For each parameter level, the LSTM's times were averaged over 7000 samples and the MLP's times were averaged over 10 000 samples, to reduce the noise as much as possible. Even though there is noise in the data, it is clear that the MLP can classify more packets a second than the LSTM. The number of packets the MLP can classify per second ranges from 25.5 to 26.6 and for the LSTM the range is 19.7 to 24.3. This was hypothesized in Section 3.2.2, because the structure of the LSTM limits the amount of parallelization, which reduces the speed of its forward pass. The plot also shows that as the number of parameters increase, there is a decrease in the number of packets classified per second. However, what is quite interesting is that the MLP's decrease is linear while the LSTM decreases at a much quicker rate. This is probably due to the increase in the number of calculations done per sequential step in the LSTM. It must also be noted that the smallest number of packets classified per second was 19.7 which is fast enough for real time classification. This means that even the slowest model will be able to pass the speed constraints. Therefore, the only constraint that needs to be looked at more carefully is the memory constraint.

The third research question was trying to determine whether the best deep learning model sufficiently outperforms a traditional machine learning model, the SVM. From Figure 3 we can see that the best deep learning model is the LSTM. Figure 5 compares the SVM with the LSTM.

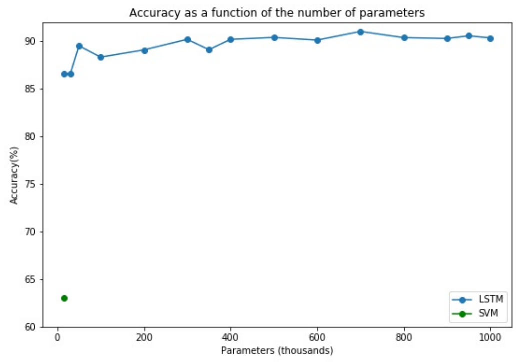

Figure 5: LSTM vs SVM

The SVM has a constant number of parameters because there are a constant number of features. This means that the number of parameters in the SVM could not be varied. The number of parameters in the SVM is 14800. There are 10 classes so 10 SVC lines needed to be fit, each line has 1480 parameters. Therefore, there are 14800 parameters in the classifier. The test accuracy represented by the dot was calculated for the single best performing SVM model. The accuracy obtained on the test set by the SVM was 63% which is far worse than the LSTM at any parameter level. Therefore, under any memory constraint the LSTM will perform better than the SVM.

The prediction speed of the SVM is remarkable, it can classify approximately 3846 packets a second. Once again, a trade-off will need to be made between accuracy and speed but the faster prediction speed provided by the SVM may not be needed because the LSTM is already classifying packets at a speed that is sufficient for real time traffic classification.

Figure 6 incorporates all the information from Figure 3 and 5 to show how all three models compare with each other. The figure shows the MLP sufficiently outperforming the SVM on all parameter levels and it shows how dominant the LSTM is over all the parameter levels.

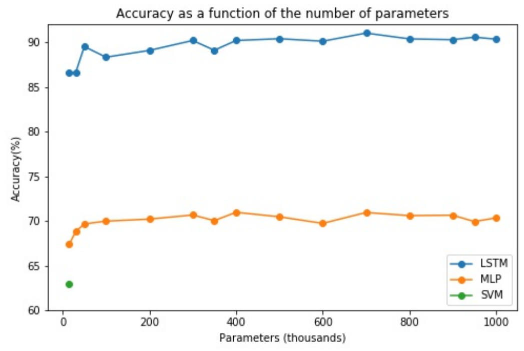

Figure 6: LSTM vs MLP vs SVM

7 DISCUSSION

The results obtained by this project are consistent with the results found in previous work. The deep learning models achieved a higher classification accuracy than the machine learning models, which was also found by the following studies [1, 4]. Furthermore, the LSTM architecture has proven to be the best model by achieving an accuracy of above 90%. This model architecture has shown this type of performance in previous traffic classification studies [1, 11, 18], where the LSTMs in these papers all had an accuracy of greater than 90%. Sequence models were shown to perform well on packet data, which corroborates evidence found by [12, 17]. These two papers gave evidence for the 1D-CNN while this paper has given evidence for the use of the LSTM in packet classification.

Figures 7 and 8 summarize the results found for each of the deep learning models. In both figures, the dotted blue line represents the accuracy obtained over the different parameter levels and the dotted orange line represents the average number of packets classified per second. The dots on the plot represent the accuracy and number of packets for a specific number of parameters.

There are a few similarities between the two deep learning architectures. For both architectures, as the number of parameters increases, the accuracy increases at a decreasing rate. This shows that after a certain point adding more parameters to the model does not increase the accuracy sufficiently to warrant the added complexity. These larger models are also more susceptible to overfitting on real world data. This is good news because it will be advantageous

to have smaller powerful models to run in a community network. They will take up less storage space and they will be able to classify more packets per second. Another similarity is that as the models get larger, the number of packets classified per second drops. The reason for this is that they will need to perform more calculations to make a prediction. It was found that the depth of the models have a larger effect on prediction speed. The deeper the models become, there are more operations that need to be done in sequence, which will drastically slow down the prediction speed.
Even though the accuracy of the LSTM model has a general trend, within this trend there looks to be some noise. The stochastic nature of the optimization algorithm that tries to find the parameter set that minimizes the loss function causes this randomness. However, the noise is not as pronounced as the plot may suggest. The increases and decreases are normally below 1%, but due to the scale of the left y-axis the noise gets exaggerated. The scale of the of the y-axis was set for aesthetic reasons. The same can be said for the noise exhibited in the packet per second line. An interesting observation can be made about the packet per second line for the LSTM model: unlike the MLP, the decrease in the packets classified per second is not linear. Up to about 400 000 parameters, the decline is quite steep but after 400 000, the decline becomes much more gradual. The steep initial decline is probably due to the increased depth found in the models, and after 400 000 the depth remained relatively constant. So, if the network needs more packets to be classified per second and the network managers need to still have a high accuracy, one of the ways to finding a solution would be to increase the width of the model. This will enable the model to have more parameters and a greater chance of learning the sequential information without sacrificing too much speed.

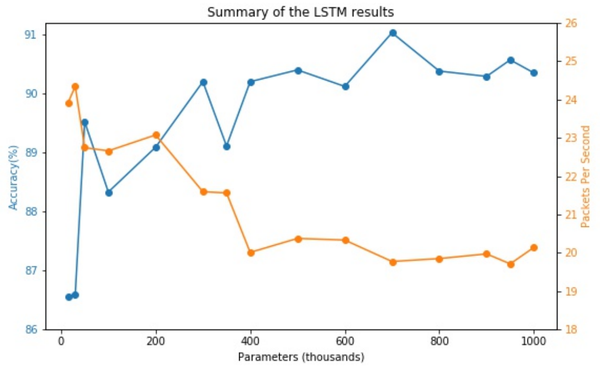

Figure 7: Accuracy vs Speed

The MLP also exhibits some noise as the models get larger, with random decreases in performance at 350 000, 600 000 ad 950 000. However, the general trend does still hold. Once again the decreases in performance were only about 1%. It is interesting to note that the MLP's speed decreases at a linear rate. This could mean that unlike the LSTM architecture, the relationship between the depth and size of the model and the number of parameters is linear. A possible reason for this phenomena is that the sequential nature of the LSTM model exaggerates the decrease.

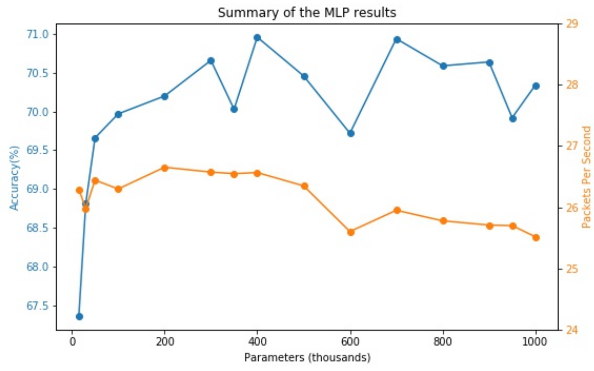

Figure 8: Accuracy vs Speed

The main aim of this project was to find out whether deep learning would be a viable solution to the traffic classification problem in the context of a community network. As previously stated, community networks have to provide their services on less powerful infrastructures. This means that there will be memory and time constraints placed on the classifiers. The LSTM model has a high test accuracy even when the models are really small, proving that even under the strictest memory constraints this architecture can be used to accurately classify traffic. Both of the deep learning models are able to classify traffic in real time, with the slowest model being able to classify 19.7 packets a second. The LSTM architecture will be the best model to deploy unless the network needs faster classification than the LSTM can provide. If this is the case, it will be best not to disregard the LSTM model but to try and make the model shallower and wider, which would reduce the number of operations performed in sequence, while still maintaining a high accuracy by taking advantage of the sequential learning power of the LSTM. Another way to increase the prediction speed would be to take advantage of batch predictions. Batch predictions is where you classify multiple packets at the same time. The time it takes to make a prediction was calculated for only a single packet, but waiting and classifying multiple packets at the same time can reduce the time it takes to make a single prediction. Figures 9 and 10 show the relationship between the batch size and the prediction per second. The dots in the plots represent the the average number of packets classified per second, for a given batch size. In Figure 9 the models used to perform the experiment were the ones with the highest accuracy. The MLP has 400 000 parameters and the LSTM has 700 000 parameters. In Figure 10, both models had 700 000 parameters. These plots clearly show that increasing the batch size allows for faster predictions. This is because the prediction function gets run in parallel, which reduces the overhead found when predicting only a single packet. This plot also shows that the LSTM is slower than the MLP, but it also shows that this difference is not too large even when the LSTM is almost twice the size of the MLP.
The high accuracy obtained by the LSTM architecture over all the parameter levels gives further evidence to suggest that the the data found in packets is sequential in nature and the best way to process it would be to use architectures that take advantage of this

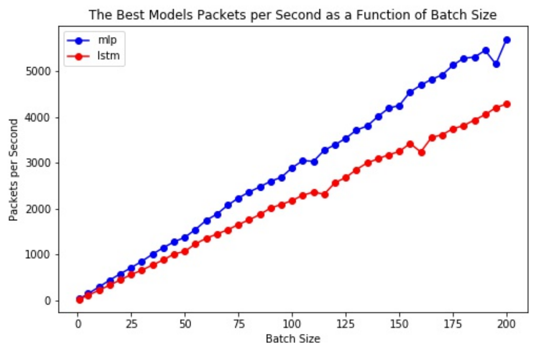

Figure 9: Number of packets classified per second by the best MLP and LSTM models plotted as a function of batch size.

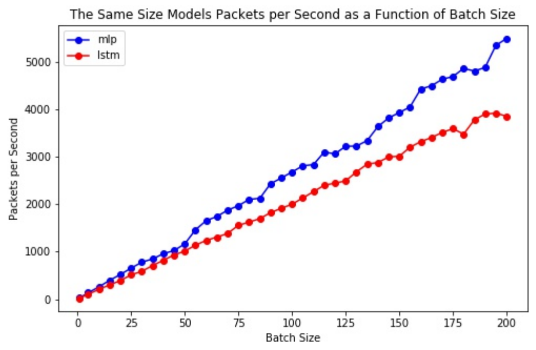

Figure 10: Number of packets classified per second by the same size MLP and LSTM models plotted as a function of batch size.

structure. Some interesting results could be obtained by classifying packets based on neural networks like the 1D-CNN and the Transformer. Transformers use attention and not recurrent connections to process sequential data and have proven to work well on language modeling tasks [16]. Because the actual architecture is not sequential, the Transformer is easily run in parallel. This could help overcome the speed limitations found in the LSTM without sacrificing accuracy.

8 CONCLUSIONS

This paper presented a preprocessing pipeline and three models that could be used for traffic classification, and for the purpose of answering the three research questions. The preprocessing pipeline that was made is able to take a set of pcap files collected from a community network and create a dataset that can be used to train and test machine and deep learning models. Using the models trained on this dataset, the paper found that the deep learning architectures, the MLP and LSTM, were able to perform real time classification, which is vital for improving the QoS algorithm. The LSTM architecture attained the highest classification accuracy of 91%, and significantly outperformed the MLP across all memory constraints. These results answered research questions one and two because it showed that the LSTM is the best deep learning architecture, and it showed that it sufficiently outperformed the MLP when resource constraints were applied. The MLP and the LSTM were also able to obtain a sufficiently higher classification accuracy than the SVM, even under the most strict memory constraints. This answered the final research question because the best deep learning architecture outperformed the SVM even when it was subject to resource constraints. The high accuracy obtained by the LSTM gives evidence that the data found in packets is sequential, and architectures that are built to process this information will do better in the packet classification task.

9 LIMITATIONS AND FUTURE WORK

There were some limitations to this project. Firstly, the labels that were used in the training and evaluation of the models were provided by a Deep Packet Inspection tool *nDPI*. This tool is subject to error, which could have eroded the results obtained in this paper. Secondly, due to limited computational power, multiple models could not be fit at each parameter level, and because of the stochastic optimization processes, there was some noise in the results.
In future studies collecting data that already has labels would lend the results even more credibility. To reduce the noise, multiple models could be fit at each parameter level and the average accuracy could be measured. Another way this study could be enhanced in future research is to collect data from multiple networks so that the models can become more general and robust.

10 DISCLAIMER

This work is based on the research supported wholly/in part by the National Research Foundation of South Africa (Grant Number MND190728459990)

National Research Foundation

REFERENCES

[1] 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.
[2] A. O. Adedayo and B. Twala. 2017. QoS functionality in software defined network. In *2017 International Conference on Information and Communication Technology Convergence (ICTC)*. 693–699.
[3] Bart Braem, Chris Blondia, Christoph Barz, Henning Rogge, Felix Freitag, Leandro Navarro, Joseph Bonicioli, Stavros Papathanasiou, Pau Escrich, Roger Baig Viñas, et al. 2013. A case for research with and on community networks.
[4] Zhitang Chen, Ke He, Jian Li, and Yanhui Geng. 2017. Seq2Img: A sequence-to-image based approach towards IP traffic classification using convolutional neural networks. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 1271–1276.
[5] Ran Dubin, Amit Dvir, Ofir Pele, and Ofer Hadar. 2017. I know what you saw last minute—encrypted http adaptive video streaming title classification. *IEEE Transactions on Information Forensics and Security* 12, 12 (2017), 3039–3049.
[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[7] iNethi Technologies. 2020. https://www.inethi.org.za/deployments/ Accessed: 2020-04-29.

[8] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An introduction to statistical learning*. Vol. 112. Springer.

[9] Jonnalagadda. 2018. *Sparse, Stacked and Variational Autoencoder*. https://medium.com/@venkatakrishna.jonnalagadda/sparse-stacked-and-variational-autoencoder-efe5bfe73b64

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[11] Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. 2017. Network traffic classifier with convolutional and recurrent neural networks for Internet of Things. *IEEE Access* 5 (2017), 18042–18050.

[12] 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.

[13] Mohammad Lotfollahi, Ramin Shirali Hossein Zade, Mahdi Jafari Siavoshani, and Mohammdsadegh Saberian. 2017. Deep Packet: A Novel Approach For Encrypted Traffic Classification Using Deep Learning. arXiv:cs.LG/1709.02656

[14] Leandro Navarro, Pau Escrich, Roger Baig, and Axel Neumann. 2012. Community-Lab: Overview and invitation to the research community. In *2012 IEEE 12th International Conference on Peer-to-Peer Computing (P2P)*. IEEE, 71–72.

[15] Shahbaz Rezaei and Xin Liu. 2019. Deep learning for encrypted traffic classification: An overview. *IEEE communications magazine* 57, 5 (2019), 76–81.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[17] Pan Wang, Feng Ye, Xuejiao Chen, and Yi Qian. 2018. Datanet: Deep learning based encrypted network traffic classification in sdn home gateway. *IEEE Access* 6 (2018), 55380–55391.

[18] 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.

[19] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. 2017. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 43–48.