Resource-constrained Real-time Network Traffic Classification using One-Dimensional Convolutional Neural Networks

Jonathan Tooke and Josiah Chavula

Computer Science Department
University of Cape Town, South Africa
jonathantooke@gmail.com, jchavula@cs.uct.ac.za

Abstract. Real-time network traffic classification is vital for networks to implement Quality of Service (QoS) traffic engineering. Deep learning techniques have proven to be effective for classification tasks, even when the traffic is encrypted. The pursuit for higher accuracy has incentivized implementations of deep learning models that are larger and slower, and require higher computational resources. This poses a problem for real-time online classification, particularly in low resource environments. This paper considers the trade-off between prediction speed and accuracy for the packet-based network traffic classification tasks when computing resources are limited. We build and compare 1D Convolutional Neural Network (1D-CNN) and the Multilayer Perceptron (MLP) models of various sizes with varying packet payload lengths used as input. These deep learning models are further compared to Support Vector Machine (SVM) models across the same metrics. The models are evaluated on six different sets of hardware constraints that are likely to be found in low-resource community networks. The study finds a clear trade-off between prediction rate and attainable accuracy. Our results suggest that MLP can achieve sufficiently fast prediction in community networks with middle-range CPUs, and for the most powerful of CPUs, a 1D-CNN should be the preferred model.

Keywords: deep learning, neural networks, network classification, community networks, quality of service, machine learning

1 Introduction

Network traffic classification is a problem that has undergone several evolutionary steps in line with improvements in network security standards and the growth of computational power. Improvements in network security, such as the reduction in dedicated application ports and an increase in the prevalence of encryption have made traditional network traffic classification algorithms less effective at best, and obsolete at worst [21]. At the same time, the growth in computational power and big data has resulted in the ever-increasing popularity of machine learning (ML) frameworks for this classification task.

The focus of this study is on packet-based real-time or online network traffic classification, which requires that traffic is classified using individual packets in near real-time. The online classification task will be considered from the perspective of a community network wanting to make use of QoS engineering, which requires a traffic classifier. Community networks typically experience slow internet speeds compared to traditional internet service providers, and rely on inexpensive hardware at the network gateway [5]. The slow internet speed experienced in these networks makes QoS engineering especially appealing, as the benefits of prioritizing latency-sensitive packets becomes more pronounced when there is a bottleneck in the network. The resource constraints imposed by the inexpensive hardware in community networks entails that a traffic classification tool must be lightweight and efficient enough to avoid adding latency to the network.

Deep learning techniques have shown significant success for the network classification task in recent studies [2, 13, 21]). Despite their capability of achieving higher accuracy on modern internet traffic than traditional ML [21], large deep learning models come with considerable computational overheads as compared to their traditional ML counterparts due to the sheer number of calculations that need to be performed. For low-resource environments, such as in community networks, it is not clear if deep learning models that are constrained in complexity due to computational limits would perform adequately, or outperform traditional ML models that are not as computationally intensive.

This paper investigate the trade-off between speed and model complexity, in the context of a resource constrained community network, and employing two prominent deep learning architectures; the One Dimensional Convolutional Neural Network (1D-CNN) and the Multilayer Perceptron (MLP). Two key research problems are pursued in this paper. Firstly, we evaluate whether 1D-CNNs and MLP outperform the baseline models on the basis of classification accuracy. Secondly, we compare the models' speed for real-time classification, in the context of varying levels of hardware constraints.

We build and explore varying sizes of these deep learning architectures using a randomised grid search technique, and evaluate the prediction speed and classification accuracy using a dataset collected from a community network in South Africa. The 1D-CNN and MLP models are further compared across the same metrics to a traditional and the more lightweight Support Vector Machine (SVM). Additionally, these models are built with varying input lengths to investigate the potential performance implications of only considering a smaller portion of each packet's payload as features.

2 Background

2.1 Quality of Service (QoS)

QoS engineering is used to manage network traffic with the goal of reducing latency, packet loss and jitter. This can be done by prioritizing traffic from some applications over others according to predefined network rules [4]. Successful

QoS engineering will improve user experience by ensuring that latency-sensitive traffic (such as video calls) can be prioritised over traffic that can be delayed for longer without frustrating users (such as email) . The router can only choose which packets to prioritize if it can first classify the incoming packets. This is done using online network traffic classification tools, such as the machine learning ones implemented in this paper.

2.2 Community Networks

Community networks are a solution to providing internet access in rural areas across the globe. They are typically formed by a small group of people coming together to develop a network infrastructure in their local community and then creating an access point to connect their network to the wider internet [16, 17]. Typically, community networks are distributed, decentralized low-resource systems that use low-cost hardware and wireless technologies to connect network nodes [6]. For this reason, community networks typically experience slow internet speeds, making QoS engineering particularly useful to ensure good user experience. However, due to the inexpensive hardware [5], computationally intensive traffic classifiers, such as large deep learning models, may be too slow for online classification. Smaller deep learning models and traditional ML models may be required under these circumstances.

To contextualise the performance of the models in a community network, the machine learning models trained in this paper use network traffic data samples collected from a community network in South Africa. Six sets of low-resource hardware specifications are considered, and the classification speed of the models on these hardware systems is evaluated in terms of accuracy and speed of classification. The effect of reducing the length of the packet payload as the input to deep learning models is evaluated in the context of speed and accuracy for online classification.

2.3 Support Vector Machine (SVM)

The SVM is a traditional machine learning model that is typically used for binary classification problems. In a binary classification problem, the SVM attempts to distinguish classes along a maximum-margin hyperplane, which is essentially a straight line that splits the two data classes most convincingly in the features space [18]. In situations where the data is not linearly separable, a kernel function is used to transform the data into higher dimensions to allow for a linear separation. To prevent overfitting with this approach, and since most real-world datasets contain outliers, the basic SVM algorithm has been modified to allow for some values to fall on the wrong side of the hyperplane. This is known as the soft margin. In the case of multi-class classification, a one-versus-rest approach is used, where each class is separated from all of the other classes in the dataset. The one-versus-rest is the approach that is used in this paper.

2.4 Multilayer Perceptron (MLP)

The multilayer perceptron is a type of feedforward artificial neural network. It consists of an input layer, one or more hidden layers, and an output layer. Each layer (l) consists of one or more nodes that are each connected to every node in the subsequent layer [20]. For each connection in the network, a weight value (w) is stored, and these values are adjusted as the network is trained. Additionally, each non-input node is assigned a non-linear activation function (σ) and stores a bias value (b) that is also learned during training. The data for the network is passed to the input layer, with each node in the input layer representing a feature of the data. Each subsequent node in the network calculates its output (a) by applying an activation function to the sum of the outputs of the previous layer multiplied by the connected weight and adding its bias value as shown by the equation below.

$$a_j^l = \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l) \tag{1}$$

In this paper, the MLP is used for multi-class classification with a softmax activation function at the output layer. Adam optimization [10] is used as the optimizer with dropout [23] and early stopping as regularization mechanisms.

2.5 1D Convolutional Neural Networks (1D-CNN)

A convolutional neural network is a more advanced deep learning model. The main difference to the MLP is that the input data is fed through convolutional layers first which performs the feature extraction [13]. Thereafter, a dimensionality reduction technique, such as max pooling is used, and the output of this is connected to one or more fully-connected layers.

In a 1D-CNN, the convolutional layers take a one-dimensional vector as input. A kernel of some length is slid across each sub-region of the input data, calculating the dot product between the kernel and the sub-region of the input data at each point and appending the result to the output vector. This operation results in dependencies between subsequent features being captured by the model.

The equation for the output, z, from one kernel operation is given below. Where w is the kernel vector with a different weight value at each position a. The value l is the layer in the network and m is the length of the kernel.

$$z_i^l = \sigma(\sum_{a=0}^{m-1} w_a z_{(i+a)}^{l-1}) \tag{2}$$

Max pooling is typically used after convolutional layers in order to reduce the dimensions of the output. This works by taking the maximum value from subsequent sub-regions of the output and discarding the other values. The size of the sub-region can be chosen and tuned.

Similar regularization can be used for CNNs as mentioned in Section **??** for the MLP, although dropout is less common following directly from convolutional layers. The same optimizer and output layer structure can be used as well.

3 Related Work

Deep learning techniques have been studied in recent years for for network traffic classification, partly due to the additional challenges imposed by the encryption of internet traffic [21]. In the context of network traffic classification, these deep learning techniques can be broadly categorised into flow-based [3] and packet-based approaches [14].

3.1 Flow Classification

More often than not, the objects of classification are flows [3], loosely a sequence of packets sent from a particular source application to a particular destination application. There are four prominent approaches for collecting data from a flow [3, 7, 12, 26, 27]. The first approach is to take raw data, in the form of bytes, from some of the packets in the flow [27]. 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 [26]. The third approach uses time series data like packet sizes, packets directions and inter-arrival times from individual packets [22]. 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. This may not be suitable for fast real time classification [22].

Using raw data has been shown to work better than using hand picked time series features and flow statistics [3]. 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.

In another study [27], 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. In the study, 1D-CNN achieved an accuracy of 91.25% and the 2D-CNN had and accuracy of 90% [27]. This is not a surprising result since 1D-CNNs are better suited to processing sequential data [11]. The study [27] 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 [27]. This showed that end-to-end deep learning was better than state-of-the-art machine learning. In a similar study [3], 1D-CNNs also outperformed 2D-CNNs.

3.2 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 [14, 24]. When doing individual packet classification, times series features are not useful since the focus is on individual packets. Since deep learning has the ability to learn high dimensional data [22], it can therefore learn from the raw data of a packet.

Another study [15] 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. 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 [24] used the same data 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. In one study [15], 1D-CNN was used 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 [15]. 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 [15]. A similar study [24] 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.

Packet-based classification using a 1D-CNN has been shown to be successful. Two main studies made use of 1D-CNNs for encrypted traffic classification. The first used flow-based features [25], making it less relevant for online classification. The second used the packet payload as features, where each byte is a feature [13]. Studies [13] suggest that 1D-CNNs are ideal architectures for the traffic classification task using the packet payload since they are able to recognize dependencies between successive bytes in order to learn key patterns that enable successful classification. Deep learning models are able to learn the distinguishable patterns within the encrypted data that characterize applications, despite the content itself being inaccessible due to encryption. Results presented by the studies [13] show the effectiveness of the 1D-CNN approach, with their model attaining an F1 score of 0.95, outperforming all prior similar works in

literature that performed classification on the same public dataset. This shows that 1D-CNNs are promising for this packet-based classification task.

Flow-based classification based on inter-packet features are less suited for QoS mechanisms that need require real-time traffic classification. This paper therefore focuses on packet-based online classification. From the reviewed literature, there has not been critical evaluation of computational resources required for deep learning models that can be used for online packet-based classification in low-resource networks. MLP has rarely been used due to its complexity and low accuracy [21], but it has mostly been evaluated for flow-based traffic classification tasks. This paper aims to determine whether 1D-CNN and MLP deep learning can be used to perform real-time packet-based traffic classification, given the resource constraints and requirements of low-resource community networks. We focus on online classification, i.e., where packets need to be classified in near real-time, and where the first few packets of a flow are used for classification. A previous study on traffic classification in community networks [**?**] evaluated computational efficiency and accuracy of LSTM and Multi-Layer Perceptron (MLP) models. The study showed that LSTM models attain higher out-of-sample accuracy than traditional support vector machines classifiers and the simpler multi-layer perceptron neural networks, given the same computational resource constraints.

4 Methodology

In this section, we describe the preprocessing of raw community network traffic data into the format required by the learning models. Thereafter, we specify the requirements for systems that the models must be able to run on, as well as the required throughput rates. Finally, we provide the approach for building and evaluating the different models through the use of a randomised grid search over possible model architectures and hyperparameters.

4.1 Preprocessing

Before building the traffic classifier models, the data, which is in the form of pcap files, needs to be transformed into the required models' input format. The data is converted to a packet-payload representation, where each feature is a byte from the payload. The IP and TCP headers are excluded to prevent the model from using attributes such as port numbers and IP addresses as features since these change with usage and configuration, and are therefore not necessarily representative. A bash script is written to automate the transformation process from raw pcap input to csv output, and the steps followed by the script are outlined in full detail below.

Dataset The data used for training the models is collected from the network gateway of a community network in South Africa (details withheld for blind-review). A sample of 145 raw pcap files from this network, totalling 14.2GB in size, was used with the files picked over a date-range from March to May 2019.

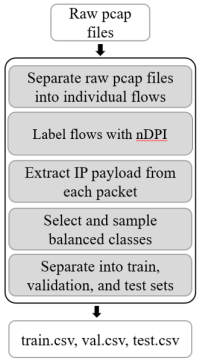

Fig. 1: Preprocessing Steps

Separating raw pcap files into flows This step takes raw pcap files as input and produces the pcap flows extracted from the raw files as output. A flow consists of a number of packets that share the same source IP address and port, destination IP address and port, and protocol [21]. Each raw pcap file in the dataset consists of many flows that represent traffic from different application classes. The tool *pkt2flow*[1] was used to separate the flows in each raw pcap file, and saved each flow to its own pcap file.

Label flows with nDPI This step takes pcap flows as input and produces a csv file with a label for each flow as output. The open-source deep packet inspection tool, *nDPI*[2], was used to produce the labels for each of the flows extracted in the step above.

Extract IP payload from each packet This step takes the flows produced by pkt2flow and extracts the raw 1480 byte IP payload from each packet in each flow and saves them to a csv file where each byte is a feature, following the approach by [13]. For packets that have less than 1460 bytes in the payload, the remainder of the payload is zero-padded. Additionally, the label produced by nDPI is assigned to this csv file, so that each packet payload has a label. The python library *scapy*[3] is used to perform the IP payload extraction.

Select and sample balanced classes nDPI produced labels for 65 different classes, but only 10 of the classes were selected for this study on the basis of having sufficient samples in the dataset. The application classes used were BitTorrent, GoogleDocs, Instagram, GMail, YouTube, Cloudflare, WhatsApp, GoogleServices, Facebook, and TeamViewer. For each each of the application

[1] *pkt2flow* Available at: https://github.com/caesar0301/pkt2flow
[2] *nDPI* Available at: https://github.com/ntop/nDPI
[3] *scapy* Available at: https://github.com/secdev/scapy

classes, 10 000 packets were sampled from the dateset to ensure a balanced dataset. A study [13] showed that this was a good amount of data per class.

Separate into train, validation, and test sets The processed dataset, consisting of 100 000 labelled byte-vectors (10 000 per class), is split into train, validation, and test sets. Of this, 20% of is reserved for the test set, 16% for the validation set, and 64% for the training set.

4.2 Hardware Constraints Considered

Performance of the various models built in the experiment are evaluated on the six resource constraints provided in the Table **??** below.

Table 1: Resource constraints considered

	Effective CPU Speed
1	300 MHz
2	600 MHz
3	1.2 GHz
4	2.4GHz
5	4.8GHz
6	9.6GHz

A number of assumptions are made around the resource constraints. Firstly, there will be about 25% of the CPU's resources available for the model to perform its classification. Secondly, differences in CPU architecture beyond clock speed are ignored and the models are assumed to be perfectly parallelizable where the effective CPU speed listed above is the number of CPU cores multiplied by the clock speed per core. Thirdly, the community network servers do not have access to a GPU. Fourthly, the RAM constraint is not considered. Lastly, the upload speed is fixed at 10Mbps, which is used to calculate the required classification throughput.

4.3 Required Classification Throughput

The required throughput refers to the number of packets that need to be classified per second to match the number of packets transferred per second by the router. To calculate required throughput, the maximum network upload speed is divided by the expected size of each packet. The required throughput is the number of packets that must be classified per second. A network speed of 10Mbps (1250000 bytes/s) is based on our community network. The expected packet size is calculated by taking the mean of the payload lengths of the sample taken from the community network, giving 970 bytes per packet. We use this value to calculate throughput, without including the IP header size as well. This produces

the calculation below.

$$RequiredThroughput = 1250000/970 = 1289 packets/s \quad (3)$$

4.4 Input Length

To investigate the effect of including less features in the input layer on model accuracy and performance, the packet lengths are considered. During training, models will only take the first n bytes from the payload as input. The number of bytes will be varied across the different models built as per Table 2.

Table 2: Number of bytes considered

Number of bytes (n)	Payload length
1460	100%
1095	75%
730	50%
365	25%

4.5 Evaluation Criteria

Classification accuracy is the metric chosen for evaluating the correctness of the models' predictions. Additionally, the models are evaluated on the speed at which they can predict the test set of 20 000 packets on a CPU, with a batch size of 32. The speed of prediction is evaluated on the basis of whether it would be sufficiently fast when subjected to varying resource constraints.

$$Accuracy = Number\ of\ correct\ predictions Total\ number\ of\ predictions \times 100 \quad (4)$$

4.6 Architecture and Hyperparameter Selection

To find the best set of hyperparameters and model architectures for the varying resource constraints, a randomised grid search technique is used to search over both the model architecture and training parameters at the same time. The architecture parameters considered are designed to produce models of varying size and complexity.

Model architectures MLP models are randomly constructed with between one and three hidden layers. Ten different values for the number of nodes in each layer are considered. However, the number of nodes considered is capped at the input size, reducing the number of options and complexity for smaller inputs used. For the CNN models, one or two 1D convolutional layers are used, followed by a max pooling layer, and then one or two dense layers. Details regarding the architecture parameters considered can be found in the Appendix B.

Hyperparameters Other parameters searched over include the dropout rate, the batch size, the learning rate and the activation function. The input length (number of features) is also varied. For the convolutional layers, values for filter size, kernel size, stride and padding are searched over. Different values for the max pooling layer are also considered. Details regarding the hyperparameters considered can be found in the Appendix B.

4.7 Experiment Procedure

Varying the input length A method is written to transform the input data to the input lengths specified in Section 4.4 so that the models can be built and evaluated on inputs of varying sizes.

Establishing a baseline with SVM The first step of the experiment involves setting a baseline classification accuracy on the dataset with a traditional machine learning model using the scikit learn LinearSVC library (SVM classifier) [19]. The model was trained and evaluated across all 4 input lengths specified. A linear kernel along with a one-vs-the-rest scheme is used as per the scikit-learn documentation.

Training deep learning models 300 MLP and 1D-CNN models are trained across the hyperparamater and model architecture search space for each of the 4 input lengths outlined in Section 4.4 using Keras [8] with Tensorflow [1]. This results in 1200 different MLP models and 1200 different CNN models that can be evaluated. The hyperparameters chosen for each of these models are saved to a csv file and the model itself is also saved. Early stopping is used to reduce training time and provide a regularization effect. A Tela V100 GPU is used to do this training and it takes between 24 and 48 hours to train each set of 1200 models.

Performance evaluation The models are now loaded into a CPU instance where their prediction speed on the test set of 20 000 packets is recorded using a batch size of 32. This was performed on an Intel Xeon CPU with 2 cores and a clock speed of 2.30GHz per core (effective CPU speed of 4.6GHz). This rate is later adjusted to match the CPU requirements of the hardware constraints specified in the Section 4 in order to provide a model allocation per hardware

specification. The prediction speed is added to the csv file generated above, along with the size of the saved model in megabytes. The hardware specs for the CPU instance used is retrieved with a bash script and saved to a text file for analysis.

Comparison to hardware constraints The ratio between the clock speed on the system that the model was tested on and the hardware constraints under consideration is calculated and the prediction rate is scaled by that same ratio. Thereafter, the calculated prediction rate is reduced by 75% to account for the assumption that only 25% of the system's resources should be allocated to the traffic classification. The prediction rate is now compared to the required throughput calculated in Section 4.3 to see if the hardware under consideration can support the model.

5 Results and Discussion

Related literature [2] suggests that 1D-CNN should outperform the MLP on accuracy, and in general, deep learning models perform better than traditional ML on modern network traffic [21]. This suggests that the MLP and 1D-CNN should outperform the SVM on accuracy. However, our analysis focuses on the prediction rate and accuracy attainable for the different models, as well as the trade-off between the two, given different resource and performance constraints. Additionally, we provide insights into the effect of reducing the payload length. For all of the graphs plotted below, prediction rate is calculated by predicting on the 20 000 packets in the test set and then calculating the number of packets predicted per second.

5.1 SVM Accuracy and Prediction Rate

Table 3: SVM Results

No. bytes (n)	Accuracy (%)	Prediction Rate (packets/s)
1460	**64.59%**	**142116**
1095	47.95%	193205
730	48.09%	260783
365	48.24%	322039

Table 3 shows that the SVM is capable of predicting at an accuracy of just under 65% when the full payload is used as the input. The prediction rate improves as the number of bytes included decreases, showing the improvement in performance of considering less features. However, this comes at the cost of a decrease in accuracy of just over 15% as soon as the full payload of 1460 bytes is no longer considered. There seems to be no significant difference in the accuracy of SVM models that use less than the full payload, suggesting that the model does not extract any additional useful information from bytes 365 to 1095.

5.2 Model Accuracy Comparison

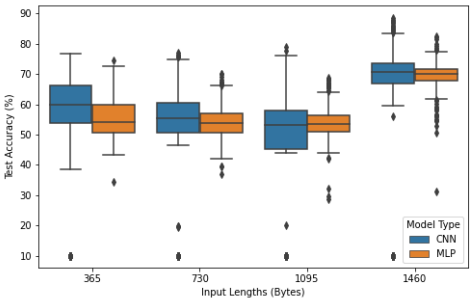

Fig. 2: Accuracy Distribution by Model and Input Length

Figure 2 shows the difference between the distribution of accuracy on the test set for each of the deep learning models using each of the input lengths considered. It is clear that models with the full payload are capable of achieving a higher accuracy than models that use less bytes as input, but there does not seem to be a significant difference between the accuracy attainable for input lengths of any other size. The higher accuracy achieved by the 1460-byte models could be attributed to either some very useful features in the later bytes in the payload or that the payload length is a useful feature that the models can only learn when they have the full payload.

Table 4: Maximum Model Accuracy Comparisons

	1D-CNN	MLP	SVM
Accuracy (%)	88.64%	82.48%	64.59%
Input Length (n)	1460	1460	1460
Prediction Rate (packets/s)	1115	34880	193205

Table 4 shows a comparison between the maximum accuracy attained by the models of each class as well as the input length used and the prediction rate. There is a clear difference in accuracy across the classes, with the 1D-CNN outperforming the MLP, which outperforms the SVM. However, the prediction rate is about 30 times slower for the CNN as compared to the MLP, which is 6 times slower than the SVM. The hyperparameters used for these models can be found in the Appendix B.

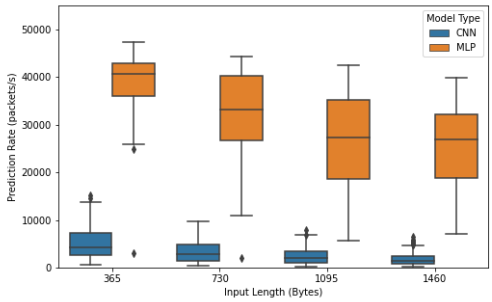

Fig. 3: Prediction Rate Distribution by Model and Input Length

5.3 Model Prediction Rate Comparison

Figure 3 provides insight into the differences between the prediction rates of the MLP and the 1D-CNN, as well as the differences between the prediction rates of models trained with the different input lengths. It is clear from the diagram that the CNN is considerably slower than the MLP across input lengths of all sizes. Additionally, a trend of models performing faster with a smaller input length can be seen for both the MLP and CNN.

5.4 Maximum Accuracy Attainable by Input Length

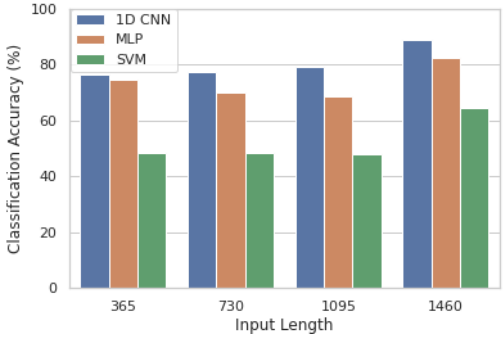

Fig. 4: Best Accuracy Attainable by Each Model for each Input Size

Figure 4 shows the maximum accuracy that each model class can attain for each input length. The highest accuracy attained by every model class was done with the full payload, indicating that the full payload should be used if accuracy is the only concern. However, Section 5.4 showed that considering less features

can result in a faster prediction rate, so it is possible that using a smaller payload could be beneficial for some hardware-model combinations. There does not seem to be a significant difference between the highest accuracy attainable for models with between 365 and 1095 bytes as input, suggesting that bytes 365 to 1095 do not add significant information.

5.5 Relationship Between Accuracy and Prediction Rate

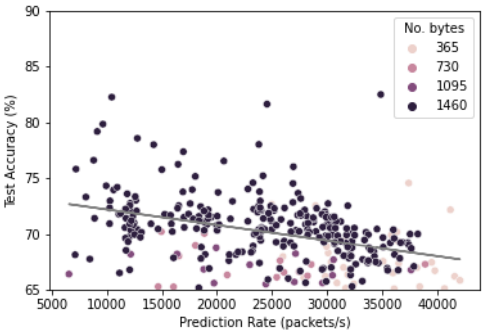

Fig. 5: MLP Accuracy vs Prediction Rate for Models with Accuracy above 65%

Figure 5 shows the relationship between the test accuracy attainable and the prediction rate for the MLP models. There appears to be an inverse relationship, with the models attaining a higher accuracy at the cost of a slower prediction rate. The number of bytes considered by each model is also shown by adjusting the hue of the dots. A trend of the dots getting darker as the accuracy improves and then prediction rate declines can also be observed, indicating that the smaller input sizes demonstrate faster classification at the expense of accuracy for the MLP models.

Figure 6 shows the relationship between the test accuracy attainable and the prediction rate for the 1D-CNN models. A similar inverse relationship exists between test accuracy and prediction rate, but it is worth noting that the models are considerably slower than those presented in Figure 5. As with Figure 5, a trend of the dots getting darker as the accuracy improves can be seen indicating that the smaller input sizes demonstrate faster classification at the expense of accuracy for the 1D-CNN models.

5.6 Model Allocations by Hardware Constraints

The following model allocations are made for the resource constraints outlined Section 4. Note that the CPU speed in the table is taken after taking the clock speed and reducing it to the 25% available for classification. Furthermore, the

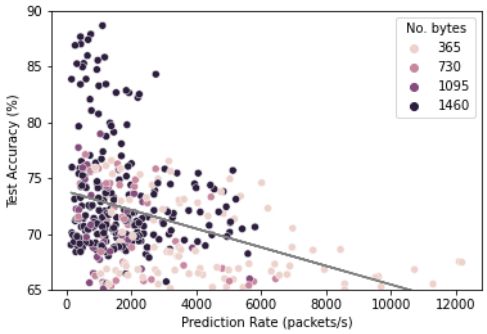

Fig. 6: 1D-CNN Accuracy vs Prediction Rate for Models with Accuracy above 65%

prediction rate is adjusted to be representative of what might be expected on each of the CPUs under consideration, and this adjusted rate is used to allocate the model with the highest accuracy that can meet the required throughput rate of 1289 packets/s for each of the CPUs. The model name is a unique identifier which follows the format [Model class]_[Number of bytes used as input]_[id] where id is a number between 0 and 299.

Table 5: Model Allocations by CPU speed

CPU Available (GHz)	Model Name	Accuracy	Rate (pk/s)
0.075	SVM_1480	64.59%	2317
0.15	MLP_365_136	72.16%	1343
0.3	MLP_1460_127	82.48%	2274
0.6	MLP_1460_127	82.48%	4549
1.2	MLP_1460_127	82.48%	9099
2.4	CNN_1460_197	84.29%	1438

Table 5 shows models that were selected for each hardware system. The model with the highest classification accuracy is selected, subject to it meeting the required prediction speed on the given hardware system. Firstly, it is only for the weakest processor that has its highest accuracy (within required speed) obtained through an SVM, with an accuracy of 64.59%. For the most powerful processor, the best performance is obtained a CNN with an accuracy of 84.29%. The other hardware configurations achieve the best performance through MLP model, with an accuracy of 82.48%. The second weakest processor uses an MLP with 365 bytes as input and produces an accuracy of 72.16%. The highest accuracy CNN reported in Section 5.4 is too slow even for the fastest CPU considered. The hyperparameters for these models can be found in the Appendix B.

6 Conclusions and Future Work

Some key conclusions can be drawn from the results of the experiment. Firstly, similar to related literature, the 1D-CNN has performed better than the MLP on accuracy. However, our analysis finds 1D-CNN to significantly slower for online classification than the MLP, with the highest accuracy 1D-CNN predicting 30 times slower than the highest accuracy MLP. The highest accuracy 1D-CNN achieved an accuracy of 88.64% and the highest accuracy MLP achieved an accuracy of 82.48%. The best SVM model uses the full payload as features and achieves an accuracy of 64.59%, significantly worse than the CNN and MLP, but performs six times faster than the best MLP and 180 times faster than the best 1D-CNN. The highest accuracy 1D-CNN is too slow for even the fastest of the resource constraints considered.

Secondly, it has been shown that there is a general inverse relationship between the prediction rate and the attainable accuracy, for both within and between model classes. This means that low-resource environments may have to select a faster model with a lower accuracy to meet the throughput requirements of an online traffic classifier. Our analysis on the use of packet payload lengths of less than the full 1460 bytes revealed that this does come at a significant accuracy cost, but with a faster prediction rate. Despite performing worse than full-payload models, the models with 365, 730, and 1095 bytes as features performed similarly to each other, suggesting that there is no consequential information captured from bytes 365 to 1095. Despite the lower accuracy, considering smaller input lengths can be worthwhile for lower resource environments, as is shown by MLP_365_136 in Table 5 which only uses the first 365 bytes as input and was selected for the second slowest resource environment.

For a community network, the recommended model would depend on the required throughput, available processor resources, and available RAM. Only the first two have been considered here, but it is reasonable to infer that models with a faster prediction rate will generally require less RAM as less calculations are performed and thus less values need to be stored. For low-resourced community networks, it appears that the MLP is a good option as it perform sufficiently fast to meet the processor constraints, while achieving a reasonable accuracy. The CNN should be used in community networks that have high-end CPUs or access to a GPU. The SVM should only be used in community networks that have exceptionally slow processors.

While this study has provided insight into the potential of using deep learning models for online traffic classification in low resource networks, there are a number of ways in which the work could be extended. For example, it is useful to investigate the effects of using different number of classes for classification on accuracy and performance. This study has only considered the effectiveness of 1D-CNN, MLP and SVM on for the online traffic classification task. Other deep learning models used in related literature with good performance include the LSTM, 2D-CNN and Stacked Autoencoder. The performance of other traditional machine learning models could be investigated and compared.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). pp. 265–283 (2016)
2. Aceto, G., Ciuonzo, D., Montieri, A., Pescapé, A.: Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges. IEEE Transactions on Network and Service Management **16**(2), 445–458 (2019)
3. Aceto, G., Ciuonzo, D., Montieri, A., Pescapé, A.: Mobile encrypted traffic classification using deep learning. In: 2018 Network Traffic Measurement and Analysis Conference (TMA). pp. 1–8. IEEE (2018)
4. Adedayo, A.O., Twala, B.: Qos functionality in software defined network. In: 2017 International Conference on Information and Communication Technology Convergence (ICTC). pp. 693–699 (2017)
5. Braem, B., Blondia, C., Barz, C., Rogge, H., Freitag, F., Navarro, L., Bonicioli, J., Papathanasiou, S., Escrich, P., Baig Viñas, R., Kaplan, A.L., Neumann, A., Vilata i Balaguer, I., Tatum, B., Matson, M.: A case for research with and on community networks **43**(3) (2013)
6. Braem, B., Blondia, C., Barz, C., Rogge, H., Freitag, F., Navarro, L., Bonicioli, J., Papathanasiou, S., Escrich, P., Baig Viñas, R., Kaplan, A.L., Neumann, A., Vilata i Balaguer, I., Tatum, B., Matson, M.: A case for research with and on community networks. SIGCOMM Comput. Commun. Rev. **43**(3), 68–73 (Jul 2013). https://doi.org/10.1145/2500098.2500108, https://doi.org/10.1145/2500098.2500108
7. Chen, Z., He, K., Li, J., Geng, Y.: 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). pp. 1271–1276. IEEE (2017)
8. Chollet, F., et al.: Keras (2015), https://github.com/fchollet/keras
9. Jonnalagadda: Sparse, stacked and variational autoencoder (2018), https://medium.com/@venkatakrishna.jonnalagadda/sparse-stacked-and-variational-autoencoder-efe5bfe73b64
10. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
11. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature **521**(7553), 436–444 (2015)
12. Lopez-Martin, M., Carro, B., Sanchez-Esguevillas, A., Lloret, J.: Network traffic classifier with convolutional and recurrent neural networks for internet of things. IEEE Access **5**, 18042–18050 (2017)
13. Lotfollahi, M., Siavoshani, M.J., Zade, R.S.H., Saberian, M.: Deep packet: a novel approach for encrypted traffic classification using deep learning. Soft Computing **24**(3), 1999–2012 (2019). https://doi.org/10.1007/s00500-019-04030-2
14. Lotfollahi, M., Siavoshani, M.J., Zade, R.S.H., Saberian, M.: Deep packet: A novel approach for encrypted traffic classification using deep learning. Soft Computing **24**(3), 1999–2012 (2020)
15. Lotfollahi, M., Zade, R.S.H., Siavoshani, M.J., Saberian, M.: Deep packet: A novel approach for encrypted traffic classification using deep learning (2017)
16. Micholia, P., Karaliopoulos, M., Koutsopoulos, I., Navarro, L., Baig Vias, R., Boucas, D., Michalis, M., Antoniadis, P.: Community networks and sustainability: A

survey of perceptions, practices, and proposed solutions. IEEE Communications Surveys Tutorials **20**(4), 3581–3606 (2018)
17. Micholia, P., Karaliopoulos, M., Koutsopoulos, I., Navarro, L., Vias, R.B., Boucas, D., Michalis, M., Antoniadis, P.: Community networks and sustainability: a survey of perceptions, practices, and proposed solutions. IEEE Communications Surveys & Tutorials **20**(4), 3581–3606 (2018)
18. Noble, W.S.: What is a support vector machine? Nature biotechnology **24**(12), 1565–1567 (2006)
19. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
20. Ramchoun, H., Amine, M., Idrissi, J., Ghanou, Y., Ettaouil, M.: Multilayer perceptron: Architecture optimization and training. International Journal of Interactive Multimedia and Artificial Intelligence **4**(1), 26 (2016). https://doi.org/10.9781/ijimai.2016.415
21. Rezaei, S., Liu, X.: Deep learning for encrypted traffic classification: An overview. IEEE Communications Magazine **57**(5), 76–81 (2019)
22. Rezaei, S., Liu, X.: Deep learning for encrypted traffic classification: An overview. IEEE communications magazine **57**(5), 76–81 (2019)
23. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research **15**(1), 1929–1958 (2014)
24. Wang, P., Ye, F., Chen, X., Qian, Y.: Datanet: Deep learning based encrypted network traffic classification in sdn home gateway. IEEE Access **6**, 55380–55391 (2018)
25. Wang, W., Zhu, M., Wang, J., Zeng, X., Yang, Z.: End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In: 2017 IEEE International Conference on Intelligence and Security Informatics (ISI). pp. 43–48 (2017)
26. Wang, W., Sheng, Y., Wang, J., Zeng, X., Ye, X., Huang, Y., Zhu, M.: Hast-ids: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection. IEEE Access **6**, 1792–1806 (2017)
27. Wang, W., Zhu, M., Wang, J., Zeng, X., Yang, Z.: End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In: 2017 IEEE International Conference on Intelligence and Security Informatics (ISI). pp. 43–48. IEEE (2017)

A Hardware Constraints

- 300 MHz single core ARM processor as found on very low-range home routers.
- 600 MHz single core ARM processor as found on low-range home routers.
- 1.2 GHz single core ARM processor as found on middle-range home routers.
- Intel Pentium 4 Processor. 1 core @ 2.80 GHz. 2GB RAM. As found in entry level servers.
- Intel Core i3-5010U process. 2 cores @ 2.10 GHz/core. As found in some community network servers.
- Intel Core i5-8259U processor. 4 cores @ 2.3Ghz per core. As found in some community network servers.

B Architectures and Hyperparameters Considered

For each model constructed an item is selected from the parameters provided in the lists below.

B.1 MLP

- Number of layers: [1,2,3]
- Number of nodes per layer: [8, 16, 32, 64, 128, 256, 512, 1024], but does not select a number of nodes for each layer that is greater than the input length
- Activation function: [relu, selu, sigmoid, swish]
- Dropout from input layer: [0, 0, 0.05, 0.1, 0.15, 0.2]
- Dropout between hidden layers: [0, 0, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5]
- Dropout to the output layer: [0, 0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5]
- Learning rate: [0.002, 0.0015, 0.001, 0.001, 0.0009, 0.0008]

B.2 1D-CNN

- Number of convolutional layers: [1,2]
- Number of filters: [20, 50, 75, 100, 125, 200]
- Kernel size: [2, 3, 4, 5, 6]
- Stride length: [1,2,3]
- Padding: [valid, same]
- Max pooling size: [2, 4, 6, 8]
- Activation function: [relu, selu, sigmoid, swish]
- Learning rate: [0.002, 0.0015, 0.001, 0.001, 0.0009, 0.0008]
- Number of hidden layers: [1,2]
- Number of hidden nodes: [30, 50, 80, 100, 120, 200, 300]
- Dropout to hidden nodes: [0, 0, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5]
- Dropout to output layer: [0, 0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5]

B.3 Other

– Batch size: [64, 128, 256, 512, 1024]

C Notable Model Parameters

C.1 MLPs

MLP_1460_127 (highest accuracy MLP)

– Number of layers: 3
– Number of nodes per layer: [32, 32, 256]
– Activation function: relu
– Dropout from input layer: 0
– Dropout between hidden layers: 0
– Dropout to the output layer: 0.35
– Learning rate: 0.002
– Batch size: 128
– Number of bytes as input: 1460

MLP_365_136

– Number of layers: 3
– Number of nodes per layer: [16, 64, 32]
– Activation function: selu
– Dropout from input layer: 0.05
– Dropout between hidden layers: 0
– Dropout to the output layer: 0.1
– Learning rate: 0.0015
– Batch size: 64
– Number of bytes as input: 365

C.2 1D-CNNs

CNN_1460_177 (highest accuracy CNN)

– Number of convolutional layers: 2
– Number of filters: [100, 20]
– Kernel size: [5, 6]
– Stride length: 2
– Padding: same
– Max pooling size: 6
– Activation function: relu
– Learning rate: 0.001
– Number of hidden layers: 1
– Number of hidden nodes: 50
– Dropout to hidden nodes: 0.3
– Dropout to output layer: 0.5
– Batch size: 64
– Number of bytes as input: 1460

CNN_1460_197

- Number of convolutional layers: 2
- Number of filters: [20, 20]
- Kernel size: [5, 4]
- Stride length: 1
- Padding: same
- Max pooling size: 2
- Activation function: relu
- Learning rate: 0.002
- Number of hidden layers: 1
- Number of hidden nodes: 100
- Dropout to hidden nodes: 0.45
- Dropout to output layer: 0.1
- Batch size: 256
- Number of bytes as input: 1460