Deep Learning Traffic Classification in Resource-Constrained Community Networks

Matthew Dicks* and Josiah Chavula†
Department of Computer Science, University of Cape Town
South Africa
Email: *dckmat0041@myuct.ac.za, †jchavula@cs.uct.ac.za

2021 IEEE AFRICON | 978-1-6654-1984-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/AFRICON51333.2021.9570875

***Abstract*—Community networks are infrastructures that are run by the citizens for the citizens. These networks are often run with limited resources compared to traditional Internet Service Providers. For such networks, careful traffic classification can play an important role in improving quality of service. Deep learning techniques have been shown to be effective for this classification task, especially since classical approaches struggle to deal with encrypted traffic. However, deep learning models often tend to be computationally expensive, which limits their suitability for low-resource community networks. This paper explores the computational efficiency and accuracy of Long Short-Term Memory (LSTM) and Multi-Layer Perceptron (MLP) deep learning models for packet-based classification of traffic in a community network. We find 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. The improvement in accuracy offered by the LSTM has a tradeoff of slower prediction speed, which weakens their relative suitability for use in real-time applications. However, we observe that by reducing the size of the input supplied to the LSTMs, we can improve their prediction speed whilst maintaining higher accuracy than other simpler models.**

***Index Terms*—Network traffic classification, deep learning, community networks**

I. Introduction

Community networks are infrastructures that are run by the citizens for the citizens, where communities build, operate and own open IP-based networks [1]. These 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. Community networks often provide services using diverse and volatile resources [1]. For community networks, traffic classification can be a vital tool for achieving advanced network management, especially in the context of Quality of Service (QoS) control, pricing, resource usage planning as well as malware and intrusion detection. QoS engineering works by prioritizing traffic of some applications over others depending on the network's requirements [2]. For a QoS system to prioritize certain traffic, it needs to be able to classify traffic into appropriate classes. In addition, traffic classification provides a way for networks managers to determine how applications utilize different resources in the network, and this allows them to take more informed actions that can improve QoS in a community network.

Previous traffic classification methods have 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 [3]. 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 [4], 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.* [5] notes that these hand-crafted features rapidly become outdated due to the evolution and mix of internet traffic. Manual feature extraction is also an expensive method because experts have to be hired and the hand picking procedure is subject to human error.

The key objective of this paper is to evaluate performance of deep learning for real-time traffic classification in a resource constrained community network. In particular, we compare the deep learning architectures Long Short-Term Memory (LSTM) and Multi-Layer Perceptron (MLP), to a traditional machine learning model, the Support Vector Machine (SVM), given the same resource constraints. We also compare the complex LSTM to the simpler MLP, assuming the same computational constraints. To achieve these objectives, we build a data pipeline that takes a set of pcap files to create a dataset that can be used to train and test deep learning models. We build and evaluate three model types – SVM, MLP and LSTM. This paper therefore makes the following contributions:

1) empirical evaluation of accuracy for LSTM and MLP deep learning models in the context of memory constraints.
2) empirical evaluation of LSTM, MLP and SVM models for *real-time* classification given memory.
3) empirical evaluation of the impact of reducing the proportion of packets' payload used as input on the prediction speed and accuracy of LSTM and MLP classifiers.

II. Related Work

A. Flow Classification

More often than not, the objects of classification are flows [5]. In the literature, there are four prominent approaches for collecting data from a flow [6], [5], [7], [8], [9]. The first approach is to take raw data, in the form of bytes, from some of the packets in the flow [6]. 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 [8]. The third approach uses time series data like packet sizes, packets directions and inter-arrival times from individual packets [4]. 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.* [4] notes that this may not be suitable for fast real time classification.

Aceto et al. [5] found that using raw data was better than using hand picked time series features and flow statistics [5]. 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.

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 [9], [8]. *Lopez-Martin et al.* [9] 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 [9]. 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.* [8]. 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 [8]. 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.

B. 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 [10], [11]. 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 [4], it can therefore learn from the raw data of a packet.

Lotfollahi et al. [3] 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 [3]. 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.* [11] used the same data as *Lotfollahi et al.* [3] but disregarded the IP header.

This paper aims to determine whether deep learning can be used to perform 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, whereby the first few packets of a flow are used for classification. For online classification, there is also no way to tell apriori how many packets are in a flow and, therefore, there is no way of determining how long to wait for all packets to arrive before a flow can be classified. For this reason, we explore packet-based classification instead of flow classification.

III. Methodology

A. Obtaining Network Traffic Data

We use a dataset from the iNethi Ocean View community network in South Africa [12]. The data consists of numerous PCAP files collected at the gateway of the community network, capturing all traffic flowing between the network and the Internet since February 2019. The PCAP files have been copied to a data repository at our university, through which we access the data.

B. Preprocessing

1) Flow extraction: Each PCAP file comprises numerous traffic flows. Flow extraction is the process of splitting up a PCAP file into smaller PCAP files that contain a single flow.

We use a utility called *pkt2flow*[1] to classify packets into flows. Using the *pkt2flow* tool, we break each PCAP file into smaller files where each file contains a single flow. Working with these smaller files makes it easier to uniformly label the packets in a file that contains only a single flow.

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 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 receives that flow's label.

3) Feature extraction and transformation: The features used as input into the models are the bytes extracted from the IP payload of each packet. This method was chosen because it enables the use of both encrypted and non-encrypted 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 higher prediction accuracy [11], [10].

We extract the IP payload bytes from packets using a Python open source library called *scapy*. The extracted bytes are transformed into an appropriate format for each model. The number of bytes in the IP payload is variable, with the maximum number of bytes being 1480. To have a uniform input, we pad zeros to all packets of length less than 1480. The data was normalized to increase the learning algorithms' stability and to decrease the training time. We mask the first 20 bytes of the 1480 byte vector to ensure that models only learn the general patterns found in the IP payload, and not the information relating to the port numbers used in the network.

The SVM and the MLP models requires feature vector in the form of dimensional normalized byte stream appropriate. Furthermore, the LSTM model needs the byte stream to be broken up into multiple time steps. This means that the one dimensional 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. One 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. Due to the vanishing and exploding gradients problem [13], it is difficult for LSTM models to learn such long sequences. To solve this problem the feature vector was split 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.

C. The Datasets

Once the preprocessing stage was complete, the final dataset consisted of ten classes, each with ten thousand observations. Table I, summarises the dataset.

[1]https://github.com/caesar0301/pkt2flow.

TABLE I
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**

This dataset is further subdivided into a training set, a validation set and a test set. These subsets were created by randomly sampling from the original dataset to ensure that the distributions in the three datasets remain as similar as possible. 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 the test set.

The validation dataset 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, such that if there is 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 (described in Section III-D).

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.

D. Hyperparameter Tuning

Hyperparameters are values that are set before the training of the models. 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 changes in network topology as a result of 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 (whereby an arbitrary large number of training epochs is specified and training stops once the model performance stops improving) 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.

E. Performance Evaluation

For our use case of a low-resource network, our 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. Thus, 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, with the breakdown as (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). This will allow the models to be compared at the highest and lowest memory requirements. For example, if a network can only have a model with 100 000 parameters due to memory constraints, then we want to know which architecture will perform the best with 100 000 parameters. 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 allow us to see which type has the best accuracy under these constraints. This will also indicate whether the deep learning models can support real time classification.

The SVM model has a constant number of parameters. Our evaluation used 14800 parameters and 10 classes. Therefore, 10 csv lines needed to be fit, each line with 1480 parameters. Therefore, we report the test accuracy and the average time it takes to classify a packet only in the single best performing SVM.

1) Metrics used for predictive performance: The models are evaluated based on two characteristics, predictive performance, and computational efficiency. The metric used for predictive performance is accuracy, and the formula is as follows:

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

2) Metrics used for computational efficiency: We measure the computational efficiency of the algorithms in two ways – the time it takes to make a prediction, and the amount of memory needed to make a prediction. Since our use-case for the algorithms is to produce real-time classifications, the speed of the predictions is key. Therefore, we use 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. As the number of parameters increase, the amount of memory needed to make a prediction will increase, since 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.

F. Implementation of Models

We built the deep learning networks using Python with Tensorflow's implementation of the Keras API. Keras was chosen because it supports the use of input pipelines built in Tensorflow, which decreases the memory requirements when training the models. We implement the SVM using Python's scikit-learn library, which includes the ability to make multi-class classifications. This is important because of the need to classify traffic into multiple application classes. We run LSTM and the MLP models in Google's Colab. 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.

IV. Experimental Results

As stated in Section III-E, the number of parameters for each architecture was varied, and the test accuracy as well as the average time to make a prediction was calculated at each parameter level.

A. Accuracy and Speed of Deep Learning Models – LSTM vs MLP

The key research objective was to compare the accuracy of LSTM and MLP, given the different levels of resource constraints and model complexity. Figures 1 and 2 present results for this objective. In both plots, the points represent the accuracy and the packets classified per second for each parameter level that was described in Section III-E.

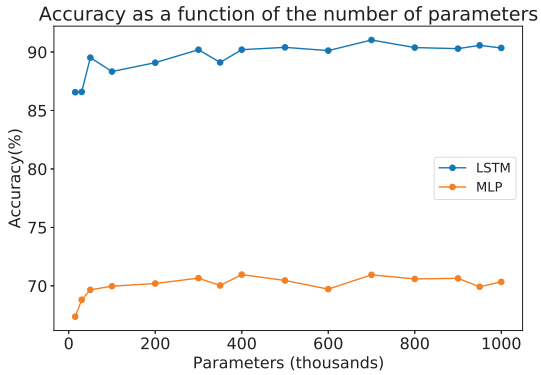

Fig. 1. LSTM vs MLP

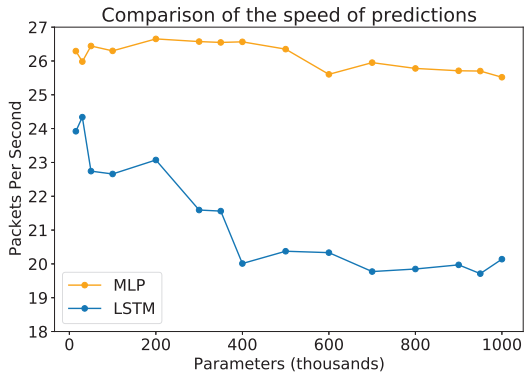

Fig. 2. LSTM vs MLP (Speed comparison); Prediction speed measured as packets per second

Figure 1 shows 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%. While there is some noise in the accuracy levels (due to stochastic nature of the optimization process for neural networks), the difference between these two models indicates that the LSTM performs better for all levels of memory constraints. This can be attributed to the LSTM being more suited to learning sequential information, and the byte stream of a packet is inherently sequential. There also is 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 2 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. 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 evident that the MLP can classify more packets per 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. The can be attributed to the structure of LSTM, which limits the amount of parallelization and reduces the speed of its forward pass. Figure 2 also shows that as the number of parameters increases, there is a decrease in the number of packets classified per second. However, the MLP's decrease is linear, while the LSTM decreases at a much faster rate. This is probably due to the increase in the number of calculations done per sequential step in the LSTM. It is also 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.

B. Deep Learning vs SVM – Accuracy and Speed

Our other research object was to determine the extent to which the deep learning model outperforms a traditional machine learning model, the SVM. Figure 3 compares the SVM with the LSTM.

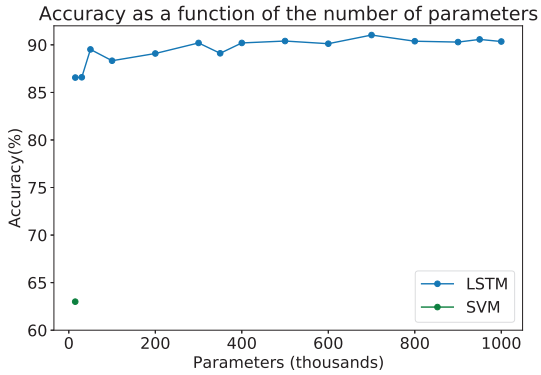

Fig. 3. LSTM vs SVM

The SVM used a constant number of features, and the test accuracy presented in Figure 3 was for the single best performing SVM model. The accuracy obtained on the test set by the SVM was 63%, which is much less than what LSTM achieves at any parameter level. This shows that under any memory constraint, the LSTM will perform better than the SVM. The prediction speed of the SVM shows that it can classify approximately 3846 packets per second. A trade-off needs 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 4 compares all three models. LSTM provides best performance over all the parameter levels, and MLP also outperforms the SVM on all parameter levels.

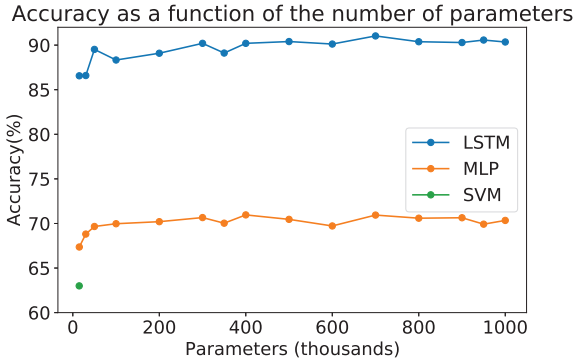

Fig. 4. LSTM vs MLP vs SVM

V. Discussion

The results obtained in this study show that deep learning models achieved a higher classification accuracy than the machine learning models, which was also shown in previous work [5], [7]. Furthermore, the LSTM architecture achieved the best or the three models, achieving accuracy above 90%. This model architecture has shown good performance in previous traffic classification studies [5], [8], [9], where the LSTM had an accuracy of greater than 90%. Sequence models were shown to perform well on packet data, which corroborates evidence found by *Lotfollahi et. al.* [10] and *Chen et. al.* [11].

Figures 5 and 6 summarize the results found for each of the deep learning models. In both figures, the blue line represents the accuracy obtained over the different parameter levels and the 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 some 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. It would thus be advantageous to have smaller powerful models to run in a low-resourced community network. Such models would take up less storage space and 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, the more the operations that need to be done in sequence, which slows down the prediction speed.

Although the accuracy of the LSTM model has a general trend, within this trend, there is some noise. This randomness is caused by the stochastic nature of the optimization algorithm that tries to find the parameter set that minimizes the loss function. However, noises are below 1%. An interesting observation can be made about the prediction speed for the LSTM model (Figures 5): 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. Thus, if the network needs more packets to be classified per second but still requires 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.

The MLP also exhibits some noise as the models get larger,

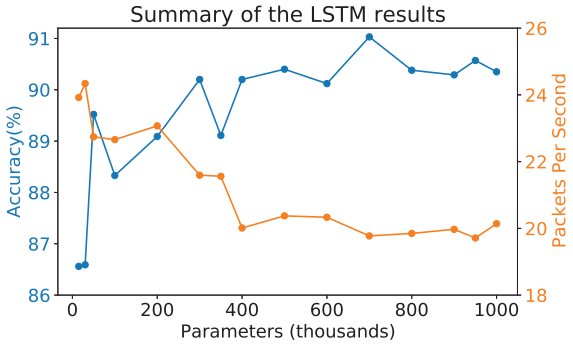

Fig. 5. Accuracy vs Speed

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.

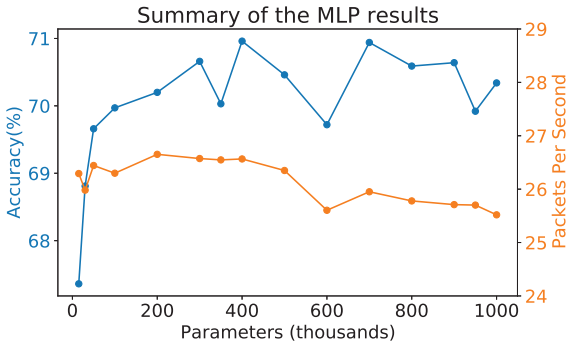

Fig. 6. Accuracy vs Speed

Figure 7 and Figure 8 show the relationship between the batch size and the prediction per second. The data points in the plots represent the average number of packets classified per second, for a given batch size. In Figure 7, 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 8, both models had 700 000 parameters. These plots indicate 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. Figure 8 also shows that the LSTM is slower than the MLP, but this difference is not very significant even when the LSTM is almost twice the size of the MLP.

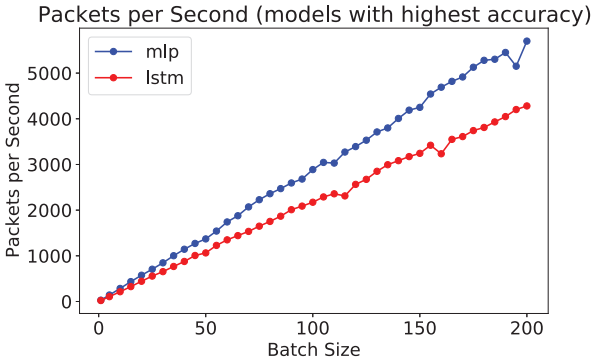

Fig. 7. Number of packets classified per second by the best MLP and LSTM models plotted as a function of batch size.

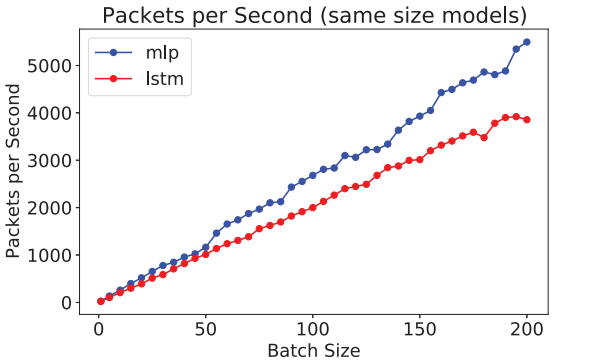

Fig. 8. Number of packets classified per second by the same size MLP and LSTM models plotted as a function of batch size.

VII. Future Work

There are several avenues that present valuable opportunities to extend the work presented in this paper. Firstly, it could be useful to test the trained models on a different community networks' labelled data, to investigate the models' ability to generalize to unseen data of potentially different distribution. Additionally, it could be beneficial to include further classes and to identify whether adequate performance can be maintained as the number of classes increase. A further avenue could be to explore the application of other deep learning architectures to packet-based classification tasks. For example, this could involve applying Stacked Auto-Encoders for dimensionality reduction, or using hybrid architectures that combine CNNs with recurrent neural networks (RNNs) with the aim of learning both spatial and temporal patterns in the data.

VI. Conclusions

This paper has presented a preprocessing pipeline and three models that could be used for traffic classification. 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 shows the deep learning architectures, the MLP and LSTM, should be able to perform real-time classification, which is vital for improving QoS engineering in low-resource networks. The LSTM architecture attained the highest classification accuracy of 91%, and significantly outperformed the MLP across all memory constraints. These results showed that LSTM 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. 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.

VIII. Disclaimer

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

References

[1] B. Braem, C. Blondia, C. Barz, H. Rogge, F. Freitag, L. Navarro, J. Bonicioli, S. Papathanasiou, P. Escrich, R. Baig Viñas, *et al.*, "A case for research with and on community networks," 2013.

[2] A. O. Adedayo and B. Twala, "Qos functionality in software defined network," in *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 693–699, 2017.

[3] M. Lotfollahi, R. S. H. Zade, M. J. Siavoshani, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," 2017.

[4] S. Rezaei and X. Liu, "Deep learning for encrypted traffic classification: An overview," *IEEE communications magazine*, vol. 57, no. 5, pp. 76–81, 2019.

[5] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile encrypted traffic classification using deep learning," in *2018 Network Traffic Measurement and Analysis Conference (TMA)*, pp. 1–8, IEEE, 2018.

[6] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "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.

[7] Z. Chen, K. He, J. Li, and Y. Geng, "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] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye, Y. Huang, and M. Zhu, "Hast-ids: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection," *IEEE Access*, vol. 6, pp. 1792–1806, 2017.

[9] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for internet of things," *IEEE Access*, vol. 5, pp. 18042–18050, 2017.

[10] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, 2020.

[11] P. Wang, F. Ye, X. Chen, and Y. Qian, "Datanet: Deep learning based encrypted network traffic classification in sdn home gateway," *IEEE Access*, vol. 6, pp. 55380–55391, 2018.

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

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.