



# Community Network Traffic Classification Using Two-Dimensional Convolutional Neural Networks

Shane Weisz<sup>(✉)</sup> and Josiah Chavula

Computer Science Department, University of Cape Town, Cape Town, South Africa  
wszsha001@myuct.ac.za, jchavula@cs.uct.ac.za

**Abstract.** Network traffic classification plays an important role in quality of service engineering. In recent years, it has become apparent that deep learning techniques are 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 weakens their suitability in low-resource community networks. This paper explores the computational efficiency and accuracy of two-dimensional convolutional neural networks (2D-CNNs) deep learning models for packet-based classification of traffic in a community network. We find that 2D-CNNs 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 2D-CNNs 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 2D-CNNs, we can improve their prediction speed whilst maintaining higher accuracy than other simpler models.

**Keywords:** Network traffic classification · Convolutional neural networks · Deep learning · Community networks

## 1 Introduction

Network traffic classification—the task of categorizing network traffic into different classes—has many important applications in traffic engineering. One of these applications is providing networks with smoother quality of service (QoS) by assigning different priorities to different applications' flows based on their classification. For example, applications involving video and voice traffic rely on fast packet transmission, whereas speed requirements are not as important for text services like email applications [15]. This is especially prevalent in today's coronavirus-afflicted world, due to the increased digitization of the workplace and emphasis on online/video communication. Effective QoS could be of particular value in low-resource community networks, where, for example, learners

in low-resource communities face particularly strenuous circumstances to adapt to online curriculum. QoS services, guided by the traffic classification task, offer the potential to prioritize those educational applications used by school students to ensure as seamless a learning experience as possible. This is one example that provides evidence of the potential positive impact towards which work on the traffic classification task can strive.

Classic approaches that have historically been used for traffic classification include port-based methods, payload-based methods such as deep packet inspection (DPI), and then classical machine learning techniques such as random forests or k-nearest-neighbours algorithms. However, these approaches have each been shown to have respective weaknesses in classifying modern network traffic. Classification methods that rely on port numbers are no longer reliable since many applications do not use standard ports. Additionally, some applications use a technique known as port obfuscation to disguise their traffic by using well-known port numbers [26]. DPI techniques require substantial time and computational resources to derive, update and maintain the rules and patterns used to identify application signatures [4]—and this task has been made more difficult by the encryption of traffic [8]. Additionally, a disadvantage of classical machine learning approaches is that to be most effective, they tend to rely on human-engineered features derived from data flows—which limits their generalizability [22]. In light of the problems that these traditional approaches face with modern network traffic, deep learning approaches have recently been explored in order to improve on the performance of these methods. Many such studies have reported excellent results, illustrating that deep learning models offer strong potential for successful, accurate and generalizable approaches to traffic classification.

For our purposes, we require a lightweight deep learning approach to traffic classification that can be utilized by community networks in low resource environments. This would require such a model to balance the trade-off between classification speed and timeliness on the one hand (to be suitable for real-time classification), and computational efficiency in terms of low memory resource usage on the other; all whilst meeting acceptable accuracy performance requirements.

To this end, we explore the effectiveness of two-dimensional convolutional neural networks (2D-CNNs) for the traffic classification task within the context of community networks. 2D-CNNs have been shown to demonstrate success in the packet-based classification task, as a result of their ability to learn spatial patterns in the packet data [15, 24, 25]. Additionally, their characteristics of sparse interactions and parameter sharing enhance their ability to meet computational resource-usage constraints and hence their suitability to the needs of community networks. The above factors guided the choice to investigate 2D-CNNs for our use case.

Our study aims to evaluate the effectiveness of 2D-CNNs for the packet-based classification task, compared to simpler MLP and SVM classifiers. To this end, we aim to evaluate these models' suitability for the context of community networks, by aiming for the highest accuracy, lowest computational resource requirements,

and fastest prediction speed (to be suitable for real-time applications). In order to achieve these aims, our experiments are designed to answer the following research questions:

1. What impact does the use of 2D-CNN deep learning models have on classification accuracy compared to the simpler MLP and SVM models given the same computational requirements?
2. Are the classification models fast enough for real-time classification (in terms of the time taken to classify packets)—and how do the 2D-CNN, MLP and SVM models compare in terms of prediction speed (given the same memory and processing power resources)?
3. To what extent can reducing the number of bytes used as input features to the model increase the prediction speed of 2D-CNN classifiers, and at what cost to the accuracy?

This paper makes the following contributions:

1. empirical evaluation of 2D-CNN deep learning models on classification accuracy for traffic classification in the context of computational constraints.
2. empirical evaluation of 2D-CNN, MLP and SVM models for *real-time* classification given memory and processor constraints.
3. empirical evaluation of the impact of reducing the proportion of a network packet’s payload used as model input on the prediction speed and accuracy of 2D-CNN classifiers.

## 2 Background

### 2.1 Community Networks

Community networks refer to network systems that are built, deployed and managed by local geographical communities (often with the help of non-profit organizations) to support their community by facilitating easier connectivity, communication and access to online services [18]. Technically speaking, these network infrastructures are typically distributed, decentralized low-resource systems that use low-cost hardware and wireless technologies to connect network nodes [2].

Community networks aim to help close the digital divide by providing cheaper connectivity in typically rural areas or developing regions that otherwise would struggle to obtain affordable and reliable internet access [18].

The community network constraint of low-cost and low-specification hardware is pertinent to the traffic classification task. Classification models deployed on routers in such a network may have use low-specification processors and limited memory. As a result, these constraints form a key basis upon which we evaluate and compare the classification approaches considered in this paper.

## 2.2 Multi-layer Perceptrons

The *multi-layer perceptron (MLP)* model is the most basic neural network architecture, a non-linear model used for supervised learning [7]. As the simplest neural network structure, MLPs will be useful as a deep learning baseline to which to compare the 2D-CNNs.

## 2.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) are one of the most popular deep learning architectures that have been applied in the traffic classification field, despite traditionally being applied to recognizing patterns in image data [20]. CNNs are designed for processing data stored in a grid-like structure, uncovering local spatial patterns within the data [10].

In particular, two-dimensional CNNs (2D-CNNs) require input data to be stored in a 2D grid-like format and make use of 2D  $k \times k$  filters to uncover patterns in the data. As such, for packet-based traffic classification, the bytes of each packet's payload is reshaped ('imaged') into a 2D image to be used as model input (the bytes values can then be considered as image pixels). In this way, the traffic classification task can be likened to image classification. Moreover, both the sparsity of information to be found in the packet data, and the noisiness of the data in terms of variability amongst packets, provide justification for the suitability of 2D-CNNs for the packet classification task.

## 2.4 Support Vector Machines

*Support vector machines (SVMs)* are a traditional machine learning classification framework suitable for high-dimensional data. For our purposes, the SVM model provides a useful baseline for comparison to the neural networks, as a representative lightweight traditional machine learning classification model applicable to high dimensional packet payload data.

## 3 Related Work

Due to its many important applications, various approaches to traffic classification have been studied extensively in literature. In recent years, much of the research into traffic classification has been around applying deep learning techniques—CNNs in particular – to overcome the difficulties of encryption of traffic. However, many such approaches perform flow-based classification based on inter-packet features, which is less suited to the real-time classification task necessary for QoS (since this would require packets to first be identified as part of a particular flow before they can be classified). As such, the key works that are particularly relevant to our study are rather those that have used CNNs for packet-based classification (that is, classification based solely on each individual packet).

One such study is *deep packet* [17], which uses 1D-CNNs for packet-based classification. The paper explains that due to spatial dependencies between bytes in the packet data, their 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. Their results testify to this end, with their 1D-CNN model attaining a highly impressive F1 score of 0.95, outperforming all prior similar works in literature that perform classification on the same public dataset. A study on *DataNet* [24] applied 2D-CNNs to the same public dataset, attaining an even higher F1 score of 0.98. These examples demonstrate how deep learning networks—and CNNs in particular—can learn valuable representations from the raw high dimensional data that comes from individual packets.

Other studies [15, 25] have applied 2D-CNNs to the packet classification task include those done by for malware classification. However, these studies all make use of separate cleaned public datasets, which makes comparing results across different studies difficult. The conclusions drawn from these studies, however, are still valuable; and in each case 2D-CNNs demonstrate strong success in terms of classification performance. These papers thus provide further support for exploring 2D-CNNs as an approach to the packet classification task for community networks. Whilst packet-based classification using 2D-CNNs has been shown to be successful on particular public datasets, studies have not considered the context of computational resource utilization or classification speed constraints. A previous study on traffic classification in community networks [9] evaluated computational efficiency and accuracy of Long Short-Term Memory (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 Design and Implementation

### 4.1 Overview of Preprocessing Pipeline

In order to compare and evaluate machine learning models for any supervised learning task, appropriate labelled training and testing datasets (often in the form of CSV files) are required. For network traffic classification in particular, traffic data is usually captured in PCAP file format, which must then be pre-processed to construct appropriately formatted datasets needed by the models. Since we are performing packet-based classification, this preprocessing involves labelling the packets, extracting the raw bytes of their payload to be used as features, and then transforming the features such that they are in the appropriate format for our models.

To this end, we scripted a robust pipeline that takes as input a set of PCAP files as input, and produces as output ‘train.csv’, ‘val.csv’ and ‘test.csv’ files that are ready for model building, training, evaluation and testing. This scripting process was implemented using a combination of Python scripts for ease of data

manipulation, and bash scripts for automation, sequencing and file manipulation. The code base was implemented with detailed easy-to-follow documentation that outlines the prerequisites and steps to take to apply the pipeline on a new system, given an arbitrary set of PCAP files as network traffic data. (This will be made available as open-source software).

The end-to-end preprocessing pipeline applied to our study can be summarized in the following diagram, and is expanded upon in the subsections thereafter:

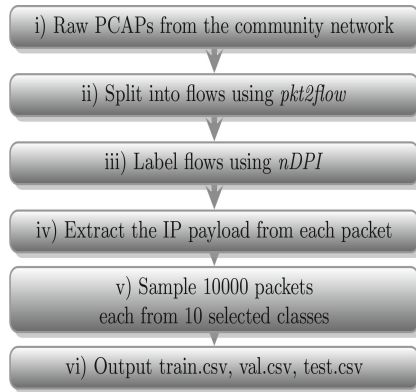


Diagram 1.1: Preprocessing pipeline

## 4.2 Dataset

Training and testing classification models to evaluate suitability for low-resource community networks requires access to community network traffic data. To this end, we have used a dataset that was collected from a community network in South Africa. The dataset consists of numerous raw PCAP files that were collected at the gateway of the network, capturing all traffic flowing between the network and the Internet from February 2019 onwards. The PCAP files were copied to a data repository at university, through which we accessed the data. However, it should also be noted that any arbitrary set of PCAPs could be used as input to this stage of the pipeline, and the rest of the preprocessing would be applied in the same manner. (Name of community network and university withheld for blind-review purpose).

## 4.3 Labelling

Since traffic classification is a supervised learning task, each packet needs to have a corresponding label to facilitate the learning and testing process. Our study performs classification by application, and as such, examples of these labels are Facebook and YouTube. A popular approach used in the literature for labelling the data (when the labels are not recorded at the time of data collection, as applicable to our data) is to use deep packet inspection (DPI) packages which use a database of application signatures to identify different classes from traffic traces

[22]. Another study [3] performed an independent study of different DPI tools, evaluating their traffic classification accuracy, and showed that the open-source tool *nDPI*<sup>1</sup> attains a very high labelling accuracy. This approach is adopted for a traffic-classification study [16], which used the *nDPI* tool—which handles encrypted traffic—to label their dataset. These factors guided our decision to use *nDPI*.

To make the labelling process cleaner, we first use the open-source package *pkt2flow*<sup>2</sup> to split the packets contained in the set of PCAP files into individual flows (with a new PCAP file for each flow). Thereafter, the flows are labelled using *nDPI*—such that each packet associated with a given flow is assigned that flow’s label. The output of this stage of preprocessing is a CSV file containing, for each flow from the original set of raw PCAPs, the flow’s PCAP file name and the application label that applies to each packet in the flow.

#### 4.4 Extracting the IP Payload

The next stage of the preprocessing pipeline involves extracting the (up to) 1480 bytes of each packet’s IP payload to be used as features as input for the classification models. 1480 bytes is the maximum size of a packet’s IP payload, since the maximum transmission unit size over the internet is typically 1500 bytes [6] – with a minimum of 20 bytes used for the IP header, meaning the remaining at most 1480 bytes correspond to the payload. As discussed in the related work section in Sect. 3, using the raw payload as model features has been shown to produce high accuracy classification in classifying both encrypted and unencrypted traffic.

To this end, the Python package *scapy* was used for processing the packet data. Packets that do not contain a payload (such as TCP handshake messages) are discarded. Transport-layer header bytes are masked to increase the generalizability of our models (due to the unreliability of using port numbers [26]). Packets with payloads less than 1480 bytes are zero-padded to ensure that all feature vectors are of the same length, as required by the classification models. The output of this stage is then a CSV file with a row for each packet containing its application label and the 1480 bytes of its payload.

#### 4.5 Sampling Balanced Classes

The next preprocessing stage involves sampling from the packets dataset to produce a balanced dataset. The emphasis on balanced classes was motivated by literature showing that studies that do not account for a class imbalance (e.g. having many YouTube packets but few Facebook packets [16]), do not perform

<sup>1</sup> *nDPI* is a deep packet inspection traffic classification module. It is available at: <https://github.com/ntop/nDPI>.

<sup>2</sup> *pkt2flow* is a simple utility that classifies packets into flows. It takes single PCAP files 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>.

as well when classifying some of the underrepresented classes, since models tend to skew their predictions in favour of the majority classes. To solve this problem, we use a method called under-sampling [15, 17], whereby classes containing more packets than needed are sampled from to extract an equal number of packets for each label class. To this end, we sample 10000 packets each from 10 selected classes that are displayed in the Table 1 below.

**Table 1.** List of applications in the dataset

Application label	Application type
YouTube	Video
Facebook	Social media
GoogleServices	Phone background services
Instagram	Images
WhatsApp	Instant messaging
BitTorrent	Torrent files
TeamViewer	Remote desktop
Gmail	Email
WindowsUpdate	Desktop OS updates
PlayStore	Mobile app store

The choice of 10000 packets per class is guided by the general size of datasets typically used in the literature. The choice of which 10 classes was guided by considering the most popular classes in the community network (and hence which would be most useful to the community network) whilst also ensuring a representative spread across different application types to ensure the classifiers are evaluated on a diverse range of applications.

Lastly, we create training, validation, and testing datasets as the final output of the preprocessing pipeline by randomly sampling from the 100000 observations in an 60-16-24 split, whilst preserving class proportions in each dataset. This split choice is popular in the literature, for example in the study by [17]. The training set is used for training the models, whilst the validation data is used as an estimate of how the models perform on the test set so as to guide the hyperparameter tuning process. Finally, the unseen test data yields an unbiased indication of each model’s out-of-sample performance and ability to generalize to new data.

## 5 Experimental Methodology

### 5.1 Evaluation Metrics

**Accuracy** is one of the most popular metrics used for evaluation of classifiers, indicating the proportion of correct classifications relative to the total number of



predictions made. The **F1 score** is another popular metric in the literature used for evaluating models, since it gives a better indication of a model’s performance on datasets that are unbalanced. However, since we took care to preprocess our dataset to ensure equal number of samples from each application class, for our study there is no reason to consider F1 score over and above accuracy. Accuracy is thus used as our key metric for evaluating the performance of the classification models under consideration.

We also need a metric for evaluating **classification speed**, as a means of assessing how models perform in meeting the real-time classification constraint. For this purpose, we use the average number of packets that a given model can classify per second. The reason for this choice of metric, as opposed to the average time taken to made a prediction, is that this allows for easier evaluation of whether models are suitable for real-time classification. The internet link capacity for the community network in questions is 10mbps, and the average number of bytes per packet in our dataset, including the IP header, is 992.57. We therefore assume that the network processes approximately 10 000 packets per second on average, and this is thus be used as a reference point to compare a given model’s prediction speed in packets per second, as an indication of how suitable it is for real-time classification.

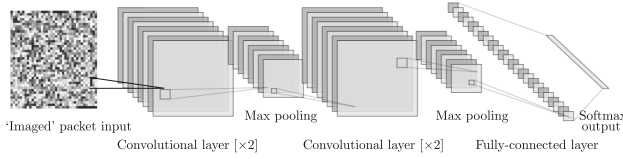
Finally, we use a given model’s **number of parameters** as a measure of its computational resource utilization—since the number of parameters of a model is the clearest determinant of a model’s complexity. The number of parameters gives a proportional indication of a model’s size and associated memory usage (typically parameters are stored as 32 bit floats, and thus 4B are needed for each parameter). The storage to define the model architecture is trivial compared to that needed for its parameters, such that the number of parameters of a model is the main determinant of its memory usage. To this end, low-parameter models are thus more desirable than high-parameter models for low-resource environments since it implies lower memory usage requirements.

## 5.2 Architectures

For the 2D-CNN networks, we consider two key architectures—a ‘shallow’ network with just one convolutional layer, and a ‘deep’ network with four convolutional layers. This allows us to compare whether the additional complexity associated with a deeper CNN can be justified.

The shallow CNN consists of the input layer, a convolutional layer followed by a max pooling layer, and then a fully-connected layer feeding into a 10-way softmax output layer. The more complex deep CNN, however, is made up of two sets of two convolutional then max pooling layers, followed again by a fully connected layer feeding into a 10-neuron softmax output layer. The architecture is depicted in Fig. 1 below.

For both networks, the filters in the convolutional layers are  $3 \times 3$  in size, with “same” padding (meaning each layer’s input is zero-padded in such a way as to preserve its spatial dimensions in its outputs). Such filters with small receptive fields have been shown to have strong success for CNNs [23], and also reduce the



**Fig. 1.** Deep 2D-CNN network architecture

number of parameters per filter which is desirable for reducing computational requirements. The number of filters in the convolutional layers and sizes of the fully-connected layer are varied as outlined in the experiment design in Sect. 5.3 below. Max pooling has been chosen for the pooling layers due to the strong success it has demonstrated with CNNs [11, 14, 23].

All hidden layers make use of Rectified Linear Unit (ReLU) [14] activations to introduce non-linearity. The Adam optimization algorithm has been chosen due its computational efficiency and subsequent reduction in training time, as well as its success in practice compared to other optimization methods [13]. To reduce model variance and prevent overfitting in the fully-connected layers, we use both dropout and l2-regularization. The learning rate and dropout rate hyperparameter-tuning is discussed in the experiment design in Sect. 5.3 below. Note that the 2D-CNN requires the input packets' features to first be reshaped ('imaged') into an  $N \times M$  matrix. For example, when the full 1480 bytes of the IP payload are used as model input, we reshape the bytes into a  $40 \times 37$  matrix. The byte values (ranging from 0 to 255) are also scaled to be between 0 and 1 in order to facilitate faster training, and the labels are one-hot encoded as required by the model for multi-class classification.

For our baseline MLP models, against which the 2D-CNNs is compared, we again consider two main architectures - a 'shallow' network with one hidden layer, and a deeper network with three hidden layers. The shallow network provides a benchmark to allow us to evaluate the performance benefit offered by deep learning compared to 'shallow' learning. Like for the CNNs, we make use of the ReLU activation function and the Adam optimizer. Dropout and l2-regularization are used to avoid overfitting. The number of neurons in each hidden layer, and the hyperparameter tuning of the learning rate and dropout rate are discussed in the experiment design in Sect. 5.3 below.

Lastly, an SVM model is also be considered as a baseline model to which to compare the neural networks. Since we expect the SVM classifier to be lightweight in terms of memory usage, and have a fast classification speed, the neural network architectures needs to show superior accuracy to justify their additional complexity. Because the number of packets in our dataset is large relative to the number of features (64 0000 packets in the training set compared to 1480 features), we use a linear kernel for our SVM classifier [19].

### 5.3 Experiment Design

We make use of two experiments in order to investigate the research questions. Experiment 1 involves comparing the models' accuracy and prediction speed across a varying number of parameters, with reference to the first two Sect. 1. Experiment 2 varies the number of bytes of each packet's payload used as model input, and evaluates the effect on the deep 2D-CNN's accuracy and prediction speeds.

**Experiment 1 – Comparing Accuracy and Prediction Speeds Against Number of Parameters:** For each  $p \in \{2^{11} = 2048, 2^{13} = 8192, \dots, 2^{21} = 2097152\}$  where  $p$  is the number of parameters of the model, we train a shallow MLP, a deep MLP, a shallow 2D-CNN, and a deep 2D-CNN (using the above-described architecture), each with approximately  $p$  parameters. The number of parameters are considered on an exponential scale in order to more comprehensively cover the sample space. Defining the number of model parameters is done by varying the number of filters in each convolutional layer and the size of the fully-connected layer in the CNNs, and varying the number of neurons in the hidden layers in the MLPs. For example, the deep 2D-CNNs for each given number of parameters are constructed by letting the number of filters for the convolutional layers and the size of the fully connected layer be  $f$  for each  $f \in \{4, 8, 16, 32, 64, 128\}$ . The specific configurations for each model are outlined in the Appendix.

For each model with a given number of parameters, we perform a grid-search hyperparameter tuning process. To this end, the grid search involves tuning the learning rate—since the learning rate is widely regarded as the most important hyperparameter to tune for neural networks [10]—as well as the dropout rate to identify what level of dropout is desirable to reduce overfitting for a particular configuration. The learning rates considered are  $\{0.01, 0.005, 0.001, 0.0005, 0.0001\}$  and dropout rates of  $\{0.05, 0.1, 0.2, 0.5\}$ . Each model with one of the combinations of these hyperparameter options is trained for a maximum of 50 epochs through the training data, with *early stopping* used to halt training if the validation accuracy has not improved over the last 5 epochs. Thereafter, the trained model is selected as the model at the number of epochs that attained the highest validation accuracy. Then the hyperparameter combination that results in the highest accuracy on the validation set is chosen as the optimal configuration for the given model architecture and number of parameters.

An SVM classifier is also trained to serve as a lightweight baseline comparison, with the SVM having just 14810 parameters (for each class, there is a parameter for each byte feature from the 1480 bytes in the IP payload, plus a bias term).

We then evaluate each of the chosen models' out-of-sample performance by computing their accuracy on the test set, and then evaluate their prediction speed by calculating the average number of packets predicted per second. This is done by choosing a sample of 10000 packets from the test set, and averaging

the time taken to classify the sample over 25 trials. The number of packets in the sample is then divided by the average time taken in seconds, to produce an estimate of the average number of packets that can be classified per second, along with an estimate of the standard error to aid in error analysis.

The experiments are performed on a low-resource virtual machine instance with only a single core in order to simulate deployment in a low-resource environment (see Sect. 5.4 for hardware specifications). As a result, the results pertaining to prediction speeds are made with reference to this specific hardware environment. However, the comparative results and inferences drawn can be extrapolated and extended to different hardware environments as needed.

**Experiment 2 – Effect of Input Size on 2D-CNN Accuracy and Prediction Speed:** This experiment is designed to determine the effect that decreasing the number of payload bytes used as features for the 2D-CNN has on the prediction time and accuracy. For each  $k^2 \in \{8^2 = 64, 16^2 = 256, 24^2 = 576, 32^2 = 1024\}$ , we train a deep 2D-CNN using the first  $k^2$  bytes of the payload as model input for each packet (reshaped into a  $k \times k$  image). The number of bytes being chosen on a quadratic scale is appropriate due to the 2D-nature of the input required for 2D-CNNs. The model architecture is that of the deep 2D-CNN described in Sect. 5.2 above, with four convolutional layers, two max-pooling layers, and a fully-connected layer feeding into a softmax output layer. Each convolutional layer consists of 32 filters, with the fully connected layer consisting also of 32 neurons. For each input size  $k$ , the corresponding model configuration is chosen according to a grid-search across the hyperparameter space. The hyperparameters considered in the search are again the learning rate chosen from  $\{0.01, 0.005, 0.001, 0.0005, 0.0001\}$ , and dropout rate from  $\{0.05, 0.1, 0.2, 0.5\}$ .

As in Experiment 1, each selected model’s accuracy on the test set is computed as an indication of its out-of-sample performance, and its prediction speed is estimated by calculating the average number of packets predicted per second (along with the associated standard error). These results are used to draw inferences about the relationship between the size of the input and the prediction speed and accuracy of the 2D-CNN.

## 5.4 Software and Hardware Environment

The CNN and MLP networks were implemented in Python through the Keras Sequential API [5] with a Tensorflow 2.0 backend [1]. Keras was chosen as our deep learning framework because of its ease of use and modularity for model building, without reducing flexibility [12]. This allowed for rapid model development, and enabled more time to be spent on experimentation. The SVM classifiers were implemented in Python using the scikit-learn API [21]. Similar to the rationale for choosing Keras for developing the neural networks, scikit-learn was chosen for its simplicity and efficiency in building machine learning models, including SVMs, for multi-class classification.

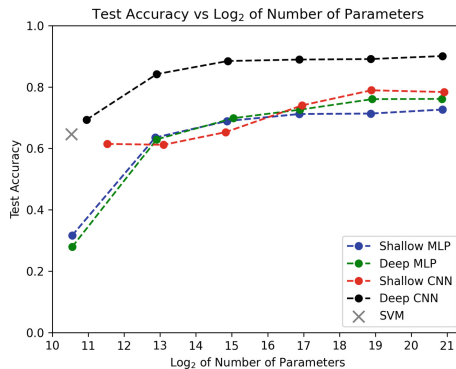
To simulate a low-resource environment, the model testing and evaluation for the experiments were performed on a single-core Intel Xeon 2.50 GHz processor with 3.75 GB RAM. The models were trained using GPUs via Google Colaboratory.

## 6 Results and Discussion

In this section we present and discuss our findings from the experiments conducted. We explore the relationships between both accuracy and prediction speed with number of parameters across the different classification models, and compare the classifiers based on these evaluation metrics. We then reduce and vary the number of bytes used as input to the 2D-CNNs, and explore to what extent the prediction time can be reduced and at what cost to the model accuracy. Full results are provided in the Appendix.

### 6.1 Accuracy Results

Figure 2 compares the deep 2D-CNN, shallow 2D-CNN, deep MLP, shallow MLP, and SVM models on the basis of their accuracy on the test set containing 24000 packets, for each given number of parameters. The test accuracy indicates each model’s out-of-sample performance and, thus, is indicative of the model’s ability to generalize to unseen data. As a result, as per convention in the machine learning field, the accuracy results displayed in the plot are considered a reliable measure of model out-of-sample performance without conducting statistical error analysis.



**Fig. 2.** Accuracy results against number of parameters for the MLP, 2D-CNN and SVM classifiers

It is clear that test accuracy increases as the number of parameters increases across the models. This matches our intuitive expectations, as increasing the

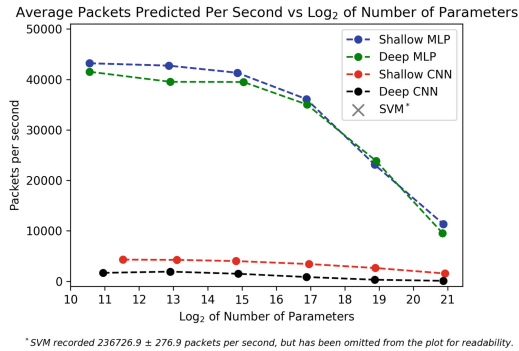
number of parameters improves the flexibility of the models to fit patterns in the data. However, added flexibility in machine learning models has the potential to result in overfitting to the training data, causing out-of-sample performance to worsen as models become more complex. This effect is not observed in our results, however, with the most likely explanation being the multiple measures we took to prevent increasing complexity from causing overfitting (using dropout and l2-regularization). Notably, however, the rate of increase in test accuracy decreases for each of the models as the number of parameters increases. For example, the 2D-CNN test accuracy can be seen to plateau from a  $\log_2$  parameters of 15. An inference that can be drawn from this observation is that it would likely be unnecessary to fit larger models with  $\log_2$  parameters greater than 21—since at this point, the added model size appears empirically to only yield a marginal improvement in performance. Notably though, models with  $\log_2$  parameters of 21 only require approximately 2MB of storage for the parameters—which is unlikely to pose memory problems on a router in production. However, due to our preference for low-parameter models for low-resource networks, we still tend to favour models at the beginning of plateaus in accuracy if increasing model size does not result in a significant improvement in performance.

Now, comparing the models' accuracy, we first note that the SVM classifier attained a test accuracy of 64.6%, and, being the smallest model in terms of number of parameters, provides a baseline accuracy to which to compare the other models. Comparing the neural networks, it is immediately apparent that the deep 2D-CNN model performs significantly better on the unseen test data than all of the other models across the range of number of parameters—with the largest deep 2D-CNN attaining a test accuracy of 90.1%. Thus, if we were to decide on the best model based solely on performance in terms of accuracy, we would choose this deep 2D-CNN model with  $\log_2$  number of parameters of 21. We can also conclude that the use of 2D-CNN deep learning models has a significant impact on attaining higher classification accuracy compared to the simpler MLP and SVM models for a given model size, in answer to our first research question.

The deep 2D-CNN outperforms the shallow 2D-CNN model for any given number of parameters, which shows the benefit that added convolutional layers can offer in terms of fitting more complex patterns in the data. The shallow MLP, deep MLP, and shallow CNN perform relatively similarly across the range of number of parameters. When the number of parameters is sufficiently large, though, we note that the shallow 2D-CNN does outperform the MLP models. This provides evidence that the more sophisticated 2D-CNN models offer better out-of-sample performance than the baseline MLP and SVM models for the traffic classification task given sufficient model complexity. We also observe that the deep MLP model attains a higher test accuracy than the shallow MLP model once the  $\log_2$  of parameters is larger than 15, thus showing the benefit of 'deep' learning when given sufficient model flexibility.

## 6.2 Prediction Speed Results

In Fig. 3 we compare the prediction speed of the various models in terms of average number of packets per second that can be predicted by each model, for each given model size. The associated standard errors for each average packets per second estimate for each model are relatively small (most less than 1%) and hence are omitted from the plot, but they are included in the results in the Appendix. The small standard errors can be attributed to the results being computed as an average taken over 25 trials, thereby testifying to the reliability of the observed results.



**Fig. 3.** Prediction speed (in packets per second) against number of parameters for the MLP, 2D-CNN and SVM classifiers.

The general trend observed amongst the models is the greater the number of parameters, the fewer average packets per second that can be classified. This matches our expectations, since having more parameters corresponds to larger weight matrices being involved in the matrix multiplications performed when making predictions for the MLPs, in addition to more convolution operations (since additional filters are responsible for increasing the number of parameters in the CNN convolutional layers) when the CNNs make predictions. These consequently result in longer computational time required to make predictions.

The lightweight SVM classifier predicts on average 236726.9 packets per second, which is substantially faster than all other models considered—in fact, this is more than five times more packets than the fastest MLP model considered. However, as discussed in the experiment design, 10000 packets per second can be considered an approximate benchmark for a model’s ability to process packets in time for real-time classification for a 10mbps network. Thus the SVM’s fast prediction speed is not necessary for our low-resource purposes, but could offer value to a very high capacity network that supports a very large traffic volume.

Comparing the neural networks, we observe that the MLP classifiers are significantly faster in terms of prediction speed than the 2D-CNNs for any given fixed number of parameters. The reason for this observation is that for each

filter in a convolutional layer in a CNN (and similarly for the pooling layers), the convolution operation using the filter must be applied to every element of the preceding layer’s output. Therefore the parameters for each filter are used repeatedly in multiple computations for a given layer, which is responsible for the high computation time required. This is in contrast to fully-connected layers in an MLP where each layer’s weight matrix is only used once in making a prediction for a packet. This observation then also explains why the shallow 2D-CNN attains a faster prediction speed than the deep 2D-CNN for a given number of parameters, due to fewer convolution operations needing to be applied.

The MLP models are relatively similar in terms of prediction speed, with the shallow MLP generally able to predict slightly more packets per second than the deep MLP. This could be attributed to fewer computational overheads involved in performing a large matrix multiplication for the shallow MLP compared to three smaller matrix multiplications for the deep MLP (with three hidden layers) for a fixed number of parameters.

We now consider the neural networks’ suitability for real-time classification, keeping in mind our benchmark of 10000 packets per second. The deep 2D-CNN, although achieving the highest accuracy amongst the classifiers across the entire range of parameters, is the slowest in terms of prediction speed and does not appear suitable for real-time classification regardless of the model size—the maximum packets per second even amongst the small deep 2D-CNN models is just 1931.4. The results of the next experiment in Sect. 6.3 below explore to what extent this prediction speed can be improved but high accuracy maintained by reducing the size of the input supplied to the 2D-CNNs.

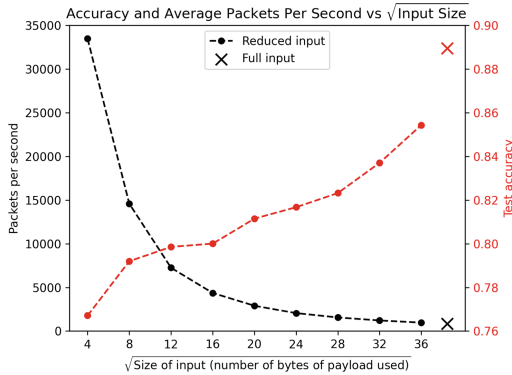
The shallow 2D-CNNs only attained a higher accuracy than the MLPs from  $\log_2$  parameters of 17 onwards—however, in this range the shallow 2D-CNN recorded a maximum of 3435.2 packets per second, which is again likely too slow to be suitable for real-time classification relative to our benchmark. In contrast, both MLP models seem to be suitable for real-time classification across the range of parameters considered, since on average they are able to predict more than 10000 packets per second.

### 6.3 The Effect of Reducing Input Size

Figure 4 plots the test accuracy and average packets predicted per second for the deep 2D-CNN model across varying input sizes (that is, using only the first  $n$  bytes of the packet payload for some  $n$ ). The standard errors associated with the average prediction time estimates were again small and thus have been omitted from the plot for readability purposes, but can be seen along with the full results in the Appendix.

The general trends we empirically observe match our expectations. As we reduce the number of bytes used as model input, the test accuracies decrease, as a result of losing the information contained in the latter bytes of the payload. However, reducing the number of input bytes does result in an increasing average number of packets that can be predicted per second (since fewer convolutions need to be performed in the convolutional layers, and fewer pooling





**Fig. 4.** Accuracy and prediction speed (in packets per second) against root input size for the deep 2D-CNN

operations in the pooling layers) - thus improving the model’s suitability for real-time classification.

We now consider whether this approach of reducing the input size can enable the 2D-CNN model to be suitable for real-time classification but still maintain high accuracy. Recall that the deep 2D-CNN model using the full payload with this given model complexity ( $\log_2$  number of parameters of 17) attained an accuracy of 88.95%, but an average prediction speed of only 851.3 packets per second. This meant that this model was not suitable for real-time classification with reference to our benchmark of 10000 packets per second. However, we notice now that when the  $\sqrt{\text{input size}}$  is 8 or less (i.e. using only the first 64 bytes or fewer), the average packets per second is greater than 10000, thus meeting our benchmark for real-time classification. The associated test accuracy is 79.2%, which is superior to the MLP and SVM models that were deemed suitable candidates for real-time classification. For faster networks, we could use even fewer input bytes (just the first 16 bytes) to get a faster prediction speed of 33472.0 packets per second, and 76.7% accuracy, which still exceeds the highest observed MLP accuracy.

This shows that, by reducing input size, the 2D-CNN model can be suitable for real-time classification and still outperform the other classification models in terms of accuracy. It is thus clear that reducing the number of bytes can significantly improve the prediction speed of the 2D-CNN without the cost to its accuracy decreasing its predictive performance advantage over the simpler models.

## 7 Conclusions and Future Work

The experiments demonstrated that 2D-CNN models are indeed superior to the baseline model candidates of SVM and MLPs when the basis of comparison is solely classification accuracy, given the same computational resources (in terms

of memory allocation, based on the number of parameters). In answer to our first research question, we can thereby conclude that 2D-CNNs do have a significant impact on classification accuracy compared to the simpler models. Notably, the largest 2D-CNN model successfully attained an accuracy of 90.1% on the test set, indicative of excellent out-of-sample performance. We observed that the neural network architectures attained higher test accuracies than the SVM traditional machine learning model, which is evidence of the added predictive power that these deep learning architectures offer over a traditional machine learning approach for traffic classification. The benefits of ‘deep’ learning over ‘shallow’ learning were also highlighted by the fact that both the deep CNNs and deep MLPs outperformed the shallow CNNs and shallow MLPs respectively.

Our second research question was set up to investigate whether the classifiers are fast enough for real-time classification, and how the models compare on this basis. To this end, we noted that despite clearly offering the strongest predictive power on out-of-sample data, the 2D-CNNs models were significantly slower than the other models in terms of prediction time, which weakens their suitability for real-time classification. In comparison, the MLP and SVM models predicted packets at a much faster rate, which is certainly more appropriate for use in real-time.

However, we observed that by reducing the proportion of the payload used as model input to just 64 bytes, the prediction speed of the deep 2D-CNN model can be improved to 14581.5 packets per second, whilst maintaining an accuracy of 79.2%. This exceeds the 10000 packets per second benchmark for real-time classification on a 10mbps network, and still offers superior out-of-sample performance relative to the other models considered. This then provides an answer to our third research question, demonstrating that by reducing the input size, the prediction speed of 2D-CNNs can be significantly improved, without the cost to its accuracy detracting from its accuracy advantage over the baseline models. As a result, we would recommend this 2D-CNN approach as the most suitable for use in real-time in community networks.

In conclusion, 2D-CNNs have been shown to be excellent candidates for the real-time packet-based traffic classification task in low-resource community network environments. As a result, through its application in QoS provisioning (amongst other areas), traffic classification using 2D-CNNs has the potential to offer significant value to the members of community networks and make important contributions towards closing the digital divide.

For future work, we will explore options for model simplification, such as dimensionality reduction using Stacked Auto-Encoders, as well as exploring hybrid architectures that combine CNNs with recurrent neural networks (RNNs) to learn both spatial and temporal patterns in the datasets.

## Appendix: Supplementary Information

### Experiment 1 2D-CNN Model Configurations

Model	Number of Parameters	Filters Per Layer	Dense Layer Size
Deep-CNN	1978	4	4
Deep-CNN	7690	8	8
Deep-CNN	30346	16	16
Deep-CNN	120586	32	32
Deep-CNN	480778	64	64
Deep-CNN	1920010	128	128
Shallow-CNN	2952	4	2
Shallow-CNN	8763	8	3
Shallow-CNN	29025	16	5
Shallow-CNN	127171	32	11
Shallow-CNN	484721	64	21
Shallow-CNN	1983203	128	43

### Experiment 1 MLP Model Configurations:

Model	Number of Parameters	Hidden Layer Sizes
Deep-MLP	1505	(1,1,1)
Deep-MLP	7685	(5,10,10)
Deep-MLP	33730	(20,50,50)
Deep-MLP	121880	(70,100,100)
Deep-MLP	492910	(200,350,350)
Deep-MLP	1892260	(450,900,900)
Shallow-MLP	1501	(1)
Shallow-MLP	7465	(5)
Shallow-MLP	29830	(20)
Shallow-MLP	120781	(81)
Shallow-MLP	480112	(322)
Shallow-MLP	1920418	(1288)

### Experiment 1 Results

Model	Number of Parameters	Test Accuracy	Average Packets Per Second (PPS)	PPS Standard Error
Shallow-MLP	1501	0.316	43197.145	208.174
Shallow-MLP	7465	0.635	42723.336	365.578
Shallow-MLP	29830	0.689	41314.968	259.977
Shallow-MLP	120781	0.712	36097.037	223.702
Shallow-MLP	480112	0.713	23107.187	79.578
Shallow-MLP	1920418	0.727	11352.556	43.226
Deep-MLP	1505	0.280	41511.741	392.387
Deep-MLP	7685	0.629	39529.375	874.219
Deep-MLP	33730	0.698	39491.803	210.787
Deep-MLP	121880	0.726	35031.978	209.343
Deep-MLP	492910	0.760	23859.578	137.279
Deep-MLP	1892260	0.761	9556.860	40.998
Shallow-CNN	2952	0.615	4312.595	6.199
Shallow-CNN	8763	0.612	4257.496	43.852
Shallow-CNN	29025	0.653	4021.531	7.449
Shallow-CNN	127171	0.740	3435.223	22.398
Shallow-CNN	484721	0.789	2653.293	5.637
Shallow-CNN	1983203	0.784	1570.599	26.536
Deep-CNN	1978	0.693	1688.317	1.469
Deep-CNN	7690	0.842	1931.446	2.370
Deep-CNN	30346	0.885	1504.579	1.190
Deep-CNN	120586	0.890	851.302	1.335
Deep-CNN	480778	0.891	312.899	0.425
Deep-CNN	1920010	0.901	97.090	0.062
SVM	1481	0.646	236726.872	276.890

### Experiment 2 Results

Number of Bytes of Input	Test Accuracy	Average Packets Per Second (PPS)	PPS Standard Error
16	0.767	33472.017	95.797
64	0.792	14581.469	36.040
144	0.799	7251.381	20.446
256	0.800	4364.418	5.552
400	0.812	2904.555	5.229
576	0.817	2055.019	2.907
784	0.823	1557.008	2.056
1024	0.837	1214.787	1.448
1296	0.854	967.467	1.167

## References

1. Abadi, M., et al.: Tensorflow: a system for large-scale machine learning. In: 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 2016), pp. 265–283 (2016)
2. Braem, B., et al.: A case for research with and on community networks. SIGCOMM Comput. Commun. Rev. **43**(3), 68–73 (2013). <https://doi.org/10.1145/2500098.2500108>
3. Bujlow, T., Carela-Español, V., Barlet-Ros, P.: Independent comparison of popular DPI tools for traffic classification. Comput. Netw. **76**, 75–89 (2015)
4. 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)
5. Chollet, F., et al.: Keras (2015). <https://keras.io>. Accessed 16 Sept 2020
6. CloudFlare: (2020). <https://www.cloudflare.com/learning/network-layer/what-is-mtu/>. Accessed 15 Sept 2020
7. Cross, S.S., Harrison, R.F., Kennedy, R.L.: Introduction to neural networks. The Lancet **346**(8982), 1075–1079 (1995)
8. Dainotti, A., Pescapé, A., Claffy, K.C.: Issues and future directions in traffic classification. IEEE Network **26**(1), 35–40 (2012)
9. Dicks, M., Chavula, J.: Deep learning traffic classification in resource-constrained community networks. In: 2021 IEEE AFRICON, pp. 1–7 (2021). <https://doi.org/10.1109/AFRICON51333.2021.9570875>
10. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
11. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and 0.5 MB model size. arXiv preprint [arXiv:1602.07360](https://arxiv.org/abs/1602.07360) (2016)
12. Keras (2020). [https://keras.io/why\\_keras/](https://keras.io/why_keras/). Accessed 16 Sept 2020
13. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint [arXiv:1412.6980](https://arxiv.org/abs/1412.6980) (2014)
14. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
15. Lim, H.K., Kim, J.B., Heo, J.S., Kim, K., Hong, Y.G., Han, Y.H.: Packet-based network traffic classification using deep learning. In: 2019 International Conference on Artificial Intelligence in Information and Communication (ICAIC), pp. 046–051. IEEE (2019)
16. 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)
17. Lotfollahi, M., Siavoshani, M.J., Zade, R.S.H., Saberian, M.: Deep packet: a novel approach for encrypted traffic classification using deep learning. Soft. Comput. **24**(3), 1999–2012 (2020)
18. Micholia, P., et al.: Community networks and sustainability: a survey of perceptions, practices, and proposed solutions. IEEE Commun. Surv. Tutor. **20**(4), 3581–3606 (2018)
19. Ng, A.: CS229 lecture notes. CS229 Lecture Notes **1**(1), 1–3 (2000)
20. O’Shea, K., Nash, R.: An introduction to convolutional neural networks. arXiv preprint [arXiv:1511.08458](https://arxiv.org/abs/1511.08458) (2015)

21. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. *J. Mach. Learn. Res.* **12**, 2825–2830 (2011)
22. Rezaei, S., Liu, X.: Deep learning for encrypted traffic classification: an overview. *IEEE Commun. Mag.* **57**(5), 76–81 (2019)
23. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint [arXiv:1409.1556](https://arxiv.org/abs/1409.1556) (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., Zeng, X., Ye, X., Sheng, Y.: Malware traffic classification using convolutional neural network for representation learning. In: 2017 International Conference on Information Networking (ICOIN), pp. 712–717. IEEE (2017)
26. Zhang, J., Chen, X., Xiang, Y., Zhou, W., Wu, J.: Robust network traffic classification. *IEEE/ACM Trans. Netw.* **23**(4), 1257–1270 (2014)