UNIVERSITY OF CAPE TOWN

Using Deep Learning to Classify Community Network Traffic

by

Chiratidzo Matowe

supervised by

Dr. Josiah Chavula

A thesis submitted in partial fulfillment for the degree of Master in Computer Science

in the

Faculty of Science

Department of Computer Science

February 2022

Declaration

"Education is all a matter of building bridges."

Ralph Ellison

Acknowledgements

Firstly, I am grateful to God, my Maker who made it possible by granting me the free gift of life and good health to pull through tough and good times. I felt encouraged through reading his Word and through prayer. His presence in my life was instrumental in bringing my work to completion.

I would like to appreciate my supervisor, Dr. Josiah Chavula who is also the co-author of this research. He guided me through it all and never grew weary of correcting my mistakes. He cheered me on, and was present in the conferences and meetings at whichI had to present. He made it possible to supervise me amidst the Covid-19 pandemic.

Many thanks go to my Dear loving family whose immense support carried me through the times I felt like giving up. They ensured I was comfortable and consistently checked up on me. It was not easy to carry out my research in a foreign land. Many times, I called them letting them know that I was ready to throw in the towel, but they encouraged me through it all.

Big thanks are also due to my friends who had a big hand too, in seeing me through this journey. Special mention to Theophilus Mathema. Their academic support geared me up and reassured me whenever I had doubts.

Finally, I would like to appreciate the Department of Computer Science as a whole and the University of Cape Town as a whole. Dealing with students during the pandemic was so hard, but they made it possible by supplying students' basic needs and stood with me through the mental strain by providing counselling services. . . .

Abstract

Traffic classification is an important aspect of network management. This aspect improves the quality of service, traffic engineering, bandwidth management and internet security. Traffic classification methods continue to evolve due to the ever-changing dynamics of modern computer networks and the traffic they generate. Numerous studies on traffic classification make use of the Machine Learning (ML) and single Deep Learning (DL) models. ML classification models are effective to a certain degree. However, studies have shown they record low prediction and accuracy scores. In contrast, the proliferation of various deep learning techniques has recorded higher accuracy in traffic classification. The Deep Learning models have been successful in identifying encrypted network traffic. Furthermore, DL learns new features without the need to do much feature engineering compared to ML or Traditional methods. Traditional methods are inefficient in meeting the demands of ever-changing requirements of networks and network applications. Traditional methods are unfeasible and costly to maintain as they need constant updates to maintain their accuracy. In this study, we carry out a comparative analysis by adopting an ML model (Support Vector Machine) against the DL Models (Convolutional Neural Networks (CNN), Gated Recurrent Unit (GRU) and a hybrid model: CNNGRU to classify encrypted internet traffic collected from a community network. In this study, we performed a comparative analysis by adopting an ML model (Support vector machine) Machine against DL models (Convolutional Neural networks (CNN), Gated Recurrent Unit (GRU) and a hybrid model: CNNGRU) and to classify encrypted internet traffic that was collected from a community network. The results show that DL models tend to generalise better with the dataset in comparison to ML. Among the Deep Learning models, the hybrid model peform better than the other models in terms of accuracy score. However, the model that had the best accuracy rate was not necessarily the one that took the shortest time when it came to prediction speed considering that it was more complex. Support vector machines outperformed the deep learning models in terms of prediction speed.

Contents

Acknowledgements **iii**

Abstract **iv**

List of Figures **x**

List of Tables **xii**

Abbreviations **xiii**

1 Introduction **1**

1.1 Problem Description and Motivation of Study 3

1.2 Aim 4

1.3 Research Objectives 4

1.4 Contribution 5

2 Literature Review **7**

2.1 Background 7

2.2 Network traffic Classification Approaches 8

2.2.1 Rule or Port-Based Method 8
2.2.2 Payload-based Method 9
2.2.3 Correlation or Statistical Classification 10
2.3 Packet vs Flow-based Classification 11
2.4 Internet traffic 12
2.5 Features used in the classification of network traffic data 13
2.5.1 Time Series 14
2.5.2 Statistical Features 14
2.5.3 Header Features 15
2.5.4 Byte/Packet Features - Based on Payload Data 15
2.6 Machine Learning 15
2.6.1 Support Vector Machine 16
2.7 Artificial Neural Networks and Deep Learning 16
2.7.1 Multi-Layer Perceptron 18
2.7.2 Convolutional Neural Network 18
2.7.3 Recurrent Neural Network 19
2.8 Related Work 20
2.8.1 Network Traffic Classification Studies using Machine Learning Algorithms 21
2.8.2 Network Traffic Classification Studies implemented using Deep Learning Algorithms 22
2.9 Conclusion 27

3 Methodology **29**

3.1 Data Collection 29

3.2 Data Preprocessing 30

3.2.1 Extraction of Data 30

3.2.2 Data Labelling 31

3.2.3 Payload Extraction 31

3.2.4 Data Cleaning 32

3.2.5 Final Structure of Data 32

3.3 Selected Classes for Experiment 32

3.4 Experimental Design 34

3.4.1 Experimental Environment 36

3.5 Adopted models used for Experiments 36

3.5.1 SVM's Architectural Implementation 36

3.6 CNN Architectural Implementation 37

3.7 GRU Architectural Implementation 38

3.8 CNNGRU Architectural Implementation 39

3.9 Experiment 1: Predicting the Best Model for classifying the network applications 40

3.10 Experiment 2 Setup: Predicting how the rate of change of data size affects the accuracy rate 41

3.11 Experiment 3 Setup:Predicting classification rate when model parameters are varied 41

3.12 Evaluation Metrics 42

3.13 Conclusion 44

4 Analysis of Results 45

4.1 Experiment 1: Maximum Accuracy prediction and evaluation of the experimental study 45

4.1.1 Confusion matrix results for SVM, CNN, GRU and CNNGRU . . 46

4.1.2 Tradeoff of Precision, Recall and F1 Score for each Model 49

4.2 Experiment 2: Accuracy prediction across different packet sizes 52

4.3 Experiment 3: Packet Prediction speed (in packets per second) against packet sizes. 55

4.3.1 Per Class Relationship between Prediction Rate, Trainable Parameters and Test Accuracy 55

4.4 Summary of Results 57

5 Discussion 58

5.1 Classification Performance of SVM against Deep Learning Models 58

5.1.1 Evaluation of the Confusion Matrix Outcome 59

5.2 Classification Performance of Deep Learning Models by varying byte data input shapes 59

5.3 Analysis of Prediction Performance of the Deep Learning Models 60

5.3.1 Evaluating Class Performance 61

5.4 Conclusion 61

6 Conclusion and Recommendations 63

6.1 Limitations and Challenges 64

6.2 Future Work 65

A An Appendix **66**

Bibliography **69**

List of Figures

2.1 The structure or format of a network packet. Source [1] 13

2.2 SVM Architecture, showing classification outcome from training two classes. Source [2] 17

2.3 Structure of a Convolutional Neural Network with 2 layers. Source [2], [3] 20

2.4 Structure of a Recurrent Neural Network. Source [4] 21

3.1 Traffic Classification Model Overview. 30

3.2 Images of classes extracted from raw packet data. 34

4.1 SVM, CNN, GRU and CNNGRU test accuracy results 46

4.2 Confusion matrix for SVM Algorithm. 48

4.3 Confusion matrix for CNN Algorithm. 49

4.4 Confusion matrix for GRU Algorithm. 50

4.5 Confusion matrix for CNNGRU Algorithm. 51

4.6 SVM, CNN, GRU, CNNGRU highest percentage accuracy achieved for different packet sizes. 53

4.7 Accuracy Distribution by each Model against Packet size. 54

4.8 Relationship displaying Packets Size for each model against Prediction speed and Test Accuracy. 56

4.9 Relationship displaying Packets per Second for each class against packet sizes with Test Accuracy of 70 %. 56

4.10 Relationship displaying Trainable Parameters for each class against packet sizes. 57

A.1 Class Distribution. 66

A.2 Summary of Gridsearch of CNNGRU Algorithm. 67

A.3 Summary of Gridsearch for GRU Algorithm. 67

A.4 Summary of Gridsearch for CNN Algorithm. 68

A.5 Summary of Gridsearch for SVM Algorithm. 68

List of Tables

2.1 Assigned port numbers by IANA [1]. 9

2.2 Assigned port numbers by IANA [5]. 10

2.3 Summary of Papers using Deep learning to classify network traffic data. . 27

3.1 Grid Search parameters for SVM Model. 37

3.2 Grid Search parameters for CNN Model. 38

3.3 Grid Search parameters for GRU Model. 39

3.4 Grid Search parameters for CNNGRU Model. 40

3.5 Overview of all Experiments. 44

4.1 Accuracy, Precision, Recall and F1 Score for SVM Model. 50

4.2 Accuracy, Precision, Recall and F1 Score for CNN Model. 51

4.3 Accuracy, Precision, Recall and F1 Score for GRU Model. 52

4.4 Accuracy, Precision, Recall and F1 Score for CNNGRU Model. 52

Abbreviations

DL	**D**eep **L**earning
ML	**M**achine **L**earning
ISP	**I**nternet **S**ervice **P**rovider
QoS	**Q**uality **o**f **S**ervice
TE	**T**raffic **E**ngineering
PCap	**P**acket **C**apture
DPI	**D**eep **P**acket **I**nspection
TCP	**T**ransmission **C**ontrol **P**rotocol
UDP	**U**ser **D**atagram **P**rotocol
SSH	**S**ecure **S**hell
SSL	**S**ecure **S**ockets **L**ayer
HTTP	**H**yperText **T**ransfer **P**rotocol
FTP	**F**ile **T**ransfer **P**rotocol
HTTPS	**H**yperText **T**ransfer **P**rotocol **S**ecure
Voip	**V**oice **O**verIP
SSL	**S**ecure **S**ockets **L**ayer
SVM	**S**upport **V**ector **M**achine
CNN	**C**onvolutional **N**eural **N**etwork
RNN	**R**ecurrent **N**eural **N**etwork
GRU	**G**ated **R**eccurent **N**eural **N**etwork
CNNGRU	**C**onvolutional **N**eural **N**etwork and **G**ated **R**eccurent **N**eural **N**etwork

Chapter 1

Introduction

Community networks are low-resourced networks built as a technological revolution that is a solution for providing access to the internet in rural areas/townships [6], [7]. For instance, resources such as bandwidth are usually significantly lower compared to traditional networks because they are established by individuals coming together in their local community to establish a network infrastructure by deploying an access point to link their network to the wider internet[8], [9]. The Ocean View community network runs on a 10Mbps link. Therefore, implementing a traffic classifier that can cater for the low amount of traffic volumes influenced by the bandwidth size is an important component of network capacity management. Additionally, knowing which applications network packets in a traffic flow belong to, will help in the prioritisation of the correct traffic [6]. For instance, transferring video and audio content requires high bandwidth for it to be transported across the network efficiently. Therefore, video and audio traffic require rapid packet transfer unlike text and email traffic, in which a low-resource network such as a community network will benefit from such an implementation.

Traffic classification is a technique considered important in network systems. Through this technique, Internet Service Providers (ISP) can manage the overall performance of a network system by implementing a mechanism that differentiates network traffic data according to application types. Different application types have different Quality of Service requirements such as bandwidth, loss, jitter, delay and best-effort options [10]. Therefore, network systems require adequate analysis, administration, and monitoring to meet different application demands and requirements. In computer networks, traffic classification is defined as associating an application to a traffic flow based on features extracted from a network flow [11]. It is an automated process that divides network traffic into flows that belong to respective application classes based on the characteristics of each application. Therefore, the fundamentals of classification known as fine-grained classification or application identification are based on user-centric/end-host applications.

The evolution of computer networks has necessitated the development of newer and more effective ways for analysing how network systems operate along with the huge amounts of internet traffic data packets flowing through them. Therefore, mechanisms including traffic analysis and traffic classification amongst others assist in achieving network administration functions such as Quality of Service(QoS), Traffic Engineering (TE), security and billing [10]. However, it is a difficult undertaking considering how current encryption techniques obfuscate the process [12]. Secondly, many network applications are continuously deployed on the network, and traditional methods are insufficient in identifying applications because of their low prediction and accuracy[13], [14]. As a result, deep-learning techniques are a viable method for automating traffic classification. Deep learning is a cutting-edge approach utilised in fields such as pattern recognition, natural language processing, and network security [14]. As computer networks evolve along with network transmission mechanisms used to transport network traffic data, it calls for advanced and effective traffic classification methods to overcome limitations of former methods (port-based, deep-packet inspection and statistical methods) of traffic classification. Accordingly, different traffic classification approaches have been proposed.

Recent studies on traffic classification implemented by, [15], [16], [17], [18], amongst others, implemented ways to automate the classification of network traffic using deep learning. The popular techniques used in these studies are Artificial Neural Networks (ANN), Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN). Recently, researchers have implemented hybrid models such as CNNLSTM and CNNGRU [13]. The hybrid model combines CNN for determining local characteristics and GRU architecture for capturing local features from both old and new data. Currently, hybrid models show potential in attaining high accuracy and prediction in classifications tasks. Hassan et al. [19] and Sarhangian et al.[13] have shown the benefits of using hybrid models in comparison to single models for a real world dataset collected from the Installation Support Centre of Expertise (ISCX) [20]. Hybrid models are more robust and achieve better predictive performance than a single model. CNNs are well-known for their excellent feature extraction capabilities [2], whereas RNNs are well-known for their ability to capture long-term dependencies [2], which can be useful when dealing with sequential data like network traffic.

1.1 Problem Description and Motivation of Study

Traditional classification methods namely: port-based, deep-packet inspection, and statistical methods implemented in the past have proven to work: however, they are inefficient and ineffective to identify encrypted network traffic data and also make it difficult for networks to apply class-based Quality of Service (QoS) and Traffic Engineering (TE) guarantees [10], [21]. Nowadays, with the trend that a variety of applications are developed, and also that the network is more complex and secure, internet traffic is primarily encrypted to ensure the safe delivery of network packets. It is vital to enhance methods of traffic classification to meet modern-day requirements of networks. When achieved successfully, it becomes easy to shape and manage a network, according to which network class requires a certain quality or fast delivery [22]. For instance, it is easier to treat a network and allocate resources based on individual network application requirements. When this is achieved, it enables a foundation of both traffic engineering and quality of service.

Network functions such as QoS and TE are reliant on a good automated classification process which is difficult to achieve with traditional methods[10]. Establishing consistent and efficient network traffic classifiers to provide real-time traffic identification is a critical unresolved topic, since the internet imposes stringent accuracy, latency, and classification speed requirements. Moreover, traditional methods require an expert to select features that distinguish applications[17]. That selection process is cumbersome and can result in human error.

In an attempt to address the classification task for QoS and TE, we implemented a packet-based classification approach considering that it would be faster to identify packets that belong to a flow in comparison to considering an entire flow. We chose to focus on packet-based network traffic classification, which requires that traffic gets classified using individual packets in near real-time. Our choice was considered from the perspective of emulating a real-time environment. Moreover, to achieve Quality of Service in a network gateway, it is more beneficial for a traffic classifier to be configured to classify individual packets into their respective application classes [16], [15].

However, considering accuracy alone was not sufficient for a low-resource environment such as a community network. In this study, we assess the performance of the three suggested deep-learning models (CNN, GRU, and CNNGRU) in terms of accuracy and prediction speed to see which one is best for the prediction task. We also took into account how much resources each application requires which is essential for rendering good quality of service for community network users. .

In addition, classification of network traffic is a useful tool for enhancing QoS and classification of network traffic is an integral and vital procedure for the community network, which has driven for the following: selecting the best suitable optimum deep learning technique that complements a community network's environment and has a good trade-off between accuracy, computational complexity, and prediction.

1.2 Aim

The study aimed to evaluate a packet-based classification framework, where we comparatively evaluated machine learning against deep learning techniques to classify community network traffic according to end-host applications. In this study, the goal was to develop a granular (fine-grained) classification task that can handle both encrypted and non-encrypted traffic data. This research further assessed the prediction rate in terms of packets per second together with varying packet sizes for each deep learning technique using packet features for classifying internet traffic applications generated in a community network. Furthermore, selecting the suitable model is highly reliant on the model's ability to correctly classify applications. Careful consideration was also required to find a model with minimal complexity and predicts more packets, while having a high accuracy score. This was in consideration of how community networks are low resourced. Each model's capacity to function efficiently in a community network with limited resources has yet to be determined.

1.3 Research Objectives

The following sub-objectives were formulated in order to address the main aim:

1. Investigate the effectiveness of existing deep learning models (CNN, GRU and CNNGRU) against an existing machine learning model (Support Vector Machines) using a packet-based classification approach for accurately classifying network traffic collected from a community network.

2. Test to what extent would varying different packet sizes would impact on the classification accuracy score of the deep learning models.

3. Investigate the effect of the deep learning model's performance in terms of prediction rate (packets predicted per second (pps)) for classifying network traffic.

1.4 Contribution

Although different techniques have evolved and have been applied on classification tasks, many studies on classification are based on private datasets, and there is no single agreed-upon solution to solve the classification problem. Currently, little has been done in evaluating the suitability of a hybrid or single DL model for automated classification that suits low-resourced environments like community networks. This study aims to fill this gap by adopting a packet-based classification method that employs deep learning to classify encrypted traffic using raw community network traffic. This demonstrated the value of a DL network traffic classification model that could be used as an automated system for class-based predictions in a community network environment. The DL model could be extended to predict network applications in real-time, thus a potential contribution to ensuring efficient traffic engineering and quality of service. Another important aspect of the proposed study is the investigation and insight into which of the present DL approaches is best for the classification study. The result shows how well the DL approaches described in section 1.3 may be used to make classification predictions using a dataset provided by the researcher. This research also looks at how Gridsearch may be used to improve model performance as a parameter-tweaking tool. It also provides a better understanding of how the amount of data needed to train a deep learning model affects its performance.

This section outlines the structure of the rest of the thesis and provides a brief introduction to the content of each chapter. The structure is as follows:

Chapter 2: Literature Review
Background theory that outlines some of the theories needed to gain a better understanding of the network traffic domain. In addition, we review experimental work done by other researchers which outlines the theoretical foundation from which we constructed a solution to the posed task.

Chapter 3: Methodology
Methodology which has the explanation on the Dataset which describes how the dataset came to be, as well as describes some of the tools and practices behind the generation of the dataset, coupled with Experiments which describes the different Experiments, describing the different experiments made in the process of classifying network traffic, as well as interpreting network decisions.

Chapter 4: Analysis of Results
Analysis reflects on explaining the detailed findings from the experiments done in Chapter 3.

Chapter 5: Discussion
Discussion reflects on some of the findings and decisions made along the way. It was based on evaluating and explaining the results we found plus how it was related to the literature and research questions.

Chapter 6: Conclusions and Recommendations
Conclusion outlines the key findings as well as providing an overall conclusion to the project and suggests some interesting directions for further work.

Chapter 2

Literature Review

This chapter gives a detailed account of the state of the art comparative analysis, open problems and justification of this research in the classification of network traffic. We present detailed information regarding the types of classification, and the criteria used in the different types of classification. Experimental tasks are pivotal in helping us determine which features are important in network traffic classification as every dataset is distinct. We also look into studies with similar methodologies for network traffic classsification. While this chapter looks at the classification techniques, Chapters 3 and 4 provides methodology and experimental results respectively through data analysis and modelling.

2.1 Background

Classification of network traffic is divided into two sub-classes. These are coarse-grained classification and fine-grained classification, [11]. The coarse-grained classification also termed traffic characterisation identifies or clusters traffic flows into several classes based on a protocol family or rough traffic type or based on groups that have similar patterns such as attack (worm and virus attacks), P2P (e.g. BitTorrent, Kazaa), bulk transfer (e.g File Transfer Protocol-FTP), Voice-Over IP (e.g. Skype), and so on. Fine-grained classification, also termed 'application identification', is focused on finer details such as identifying the exact application or service name traversing in a network (e.g. Spotify, Hangouts, e.t.c.). In the case of a community network, focusing on fine-grained classification is important as the fundamentals of traffic classification are based on user-centric applications, which give a more convenient management of a community network.

The development of a network traffic classifier, is dependent on the classification goal and also on the network traffic dataset. For example, quality of service provisioning, billing system customisation, resource usage planning, traffic engineering, intrusion detection, and malware detection can be some of the ultimate goals for classification. For community networks, network traffic classification is critical for traffic engineering (TE) and providing quality of service. For example, QoS in a community network will prioritise certain applications over others to guarantee a certain level of performance and will ensure resource prioritisation mechanisms for the prioritised applications. The aim is therefore to mark each traffic class with a traffic flow or session. Within a flow is the message carried in the form of packets.

2.2 Network traffic Classification Approaches

Various approaches or methods have been employed to achieve classification of network traffic. The need to better understand existing and new applications, as well as the evaluation of the impact of these applications on peering agreements and/or return on investment if a peer-to-peer project has been undertaken, are all reasons for changing techniques [23]. Finally, application-based services should be considered, such as securing multimedia service transmission.

2.2.1 Rule or Port-Based Method

The port-based technique consists of the study of the used communication ports in the Transport Control Protocol (TCP) or User Datagram Protocol (UDP) header and its connection with well-known TCP/UDP port numbers, which is then mapped to the associated application. The Internet Assigned Numbers Authority (IANA) is responsible for allocating port numbers [24]. Port-based is the oldest and most common method for traffic classification, which made it easy to identify applications or services[22], [25], [26]. The method also yielded fast classification because port numbers are easy to access [27]. However, internet or network communications advancements and the development of applications are inevitable. Network operators are required to know application port numbers or specific rules before implementation of a classification framework. With the increasing variety of network applications which use random ports (port-obfuscation), the port-based technique cannot identify all applications, and traffic encryption techniques caused this method to be unreliable. According to Moore et al. [28], only 30 -70% of applications are detectable using the port-matching method. In addition, the port-matching method is inefficient in the classification of large traffic flows, such as the use of the same port by web services [22], [29]. To address these issues, an extension of the port-based method, termed deep-packet inspection (DPI) or payload-based, was invented. Table 2.1 shows examples of the common port numbers assigned by IANA

Table 2.1: Assigned port numbers by IANA [1].

Port Number	**Application**
161	SNMP
123	NTP
110	POP3
80	HTTP
22	SSH
23	Telnet
21	FTP

2.2.2 Payload-based Method

Unlike port-based methods that rely fully on port numbers to identify applications, the payload-based method classifies traffic data using the payload of a packet (the actual data sent over the network) and compares the data to a known signature of protocols. DPI can detect applications or services regardless of the port used by an application. With the knowledge that every application has a unique signature that is associated with it, this technique relies on searching for the known signature [30]. Therefore, DPI hinge on the fact that it is rule-based and relies on hard-coded rules [31], [32]. In a study by Sen et al. [31], DPI was heralded as a solution to traffic classification but it has its shortcomings. The shortcomings of using DPI as a traffic classification method is linked to its computational complexity and high costs. Also, it essential to keep track of application semantics, meaning manual checks are a mandatory requirement. This is, however, a challenge, given that networks and network applications always evolve (meaning applications change frequently and hence application signatures are constantly updated). The drawback of this method lies in privacy [33]. Network users' data privacy ought to be protected. The method is subject to delay as it requires a lot of processing power [30], and it cannot access encrypted payloads [34]. For this reason, statistical methods which rely on statistical features such as packet sizes and lengths, packet inter-arrival time etc circumvent these problems since they are based on payload-independent parameters. Solutions based on DPI are commercialised and in practical use, for instance, the tool nDPI [35] and Paessler [36]. Table 2.2 below shows P2P applications that are known to use camouflage techniques and use common port numbers to traverse the network.

TABLE 2.2: Assigned port numbers by IANA [5].

Transport Protocol	**String**	**P2P Protocol**
TCP	"Get hash:"	SNMP
TCP	"GNUT" "GIV"	Gnutella
TCP	"0x13Bit"3	BitTorrent
TCP	"Get /.hash"	FastTrack
TCP/UDP	0xe319010000	eDonkey

2.2.3 Correlation or Statistical Classification

This technique relies on highly correlated features such as those discussed in section 2.5. In order, to fulfill the statistical requirements, several steps ought to be carried out to extract useful feature information. After these steps, an ML model is then trained on the extracted features. Packets are selected and grouped according to their similar statistical characteristics, which mostly stem from a network flow. Because ML methods tend to rely on human-engineered features, this tends to limit their generasibilty [37]. The approach was found to be much faster than DPI. It often uses Machine learning (ML) algorithms such as Support Vector Machines (SVM), Decision Trees (DT), Random Forests (RF), etc. Features of traffic flow can be divided into two classes according to their observation levels: flow-level features and packet-level features. The flow-level features are usually calculated after the flow has completed, such as number of packets, flow duration, mean packet size of a flow. On the contrary, the packet-level features can be obtained by early stage of flow, such as packet length, inter-arrival time and the packet direction of first few packets of flow. Low DPI scalability to high bandwidth is another point posed in favor of statistical classification. However, a study done by Cascarano et al. [38], undermines this belief. They reach a conclusion in their study that it is mostly on the traffic mixture analysed through comparing Support Vector machines(SVM) and DPI.

2.3 Packet vs Flow-based Classification

The domain of traffic classification simplifies classification by distinguishing network traffic into either flows or packets. When dealing with the classification of network traffic, researchers distinguish between flows and packets so that classification is more streamlined.

Flow-based classification is associated with the extraction of network data into traffic flows or biflows (sessions). A flow is a unidirectional packet-stream from one source to destination defined by a 5-tuple identifier (source IP, source port, destination IP, destination port, and transport-level protocol), whereas, a biflow is a bidirectional packet-stream from one source to destination that is also defined by a 5-tuple identifier. Several studies have used flows for classification [21],[16], [39], [40], [14], [41]. To ensure successful extraction of flows, several techniques are employed. The first technique method is to extract raw data in the form of bytes from some of the flow's packets [41]. In another technique, raw byte data is collected from individual packets that belong to a flow [39]. Now data that flows in a network can be viewed as individual packets or individual flows. The last two techniques are concerned with the statistics of either packet streams or flow streams. The third technique is concerned with time series features such as packet sizes, packet lengths, packet inter-arrival times, and so on [32]. The last technique deals with flows statistics, requiring more packets that belong to a flow to avoid too much variance [32]. Flow statistics include packet maximum and minimum inter-arrival times, mean, standard deviation, etc. In a survey done by Velan et al. [42], many studies used flow-based classification in comparison to packet packet-based because of the commonality of ML.

Packet-based classification, on the other hand, is associated with further preprocessing raw packets into granular entities of network traffic data known as packets. Studies such as those done by Lim et al. [15], Lotfollahi et al. [17], Wang et al.[43] and Gupta et al., [44], prove that packet-based classification is achievable, even though it is difficult to implement. Algorithms such as Multi-Layer perceptrons, Convolutional Neural Networks and Recurrent Neural Networks have been frequently implemented in packet-based classification and have achieved great results. Among the three, CNNs are commonly used even though they are well-known for object detection and image recognition. Their popularity stems from their ability to learn spatial patterns within data. In addition, unlike flow-based classification which has been the norm, few researchers have used packet-based classification. Classification using flows is better suited for offline classification because computing statistical features within the raw packets take considerable time.

With the current trend of network operations relying on real-time operations which are more ideal when the application classifier is deployed within the community network gateway, classification using packets will yield better results given that recent studies have shown classification to be possible with just a few packets. Detailed experimental researches linked to packet-based classification are recorded in the Related Work section. Flow-based classification is reliant on statistical information from packet flows whereas packet-based classification relies on raw payload data in the form of bytes/packets. In packet-based classification we only consider the payload which consists of the individual packets. Excluding header information and time-series information is a good strategy that helps retain only packet features of a flow. In this way, relevant comprehensive byte/packet features can be learnt.

2.4 Internet traffic

The evolution of the internet over the years has led to the emergence of different network applications. The internet/network is simply a connector to these applications. The applications range from multimedia streaming, file-transfer services, remote login, electronic commerce, to early text-based services [45]. Therefore, the data flowing through a network at a given point in time is internet or network data traffic. A packet is the smallest unit of network traffic data carried over the internet. Below is a diagram that shows what is contained within a packet.

The packet is split into a header, data or payload and the trailer, each with the characteristics contained in it. It is where features for classification are obtained. However, to ensure safe delivery of network traffic data requires techniques such as encryption protocols because the internet is insecure and does not preserve integrity and confidentiality. Encryption protocols are used to protect packets in transmission from source to destination. It is a form of data encoding in such a way that requires reading or decoding the encrypted data. Encryption is implemented at layer 4(transport layer) or layer 7(application layer). Layer 4 encryption protocols are Transport Layer Security (TLS) and QUIC. TLS 1.3 is latest version after TLS 1.2 that has been adopted in common browsers. Layer 7 encryption protocols include Secure Shell (SSH), Secure Sockets layer (SSL), Hyper Text Protocol Secure (HTTPS), Message Security Protocol (MSP), and so on [1]. The two types of encryption ensure that the payload is encrypted.

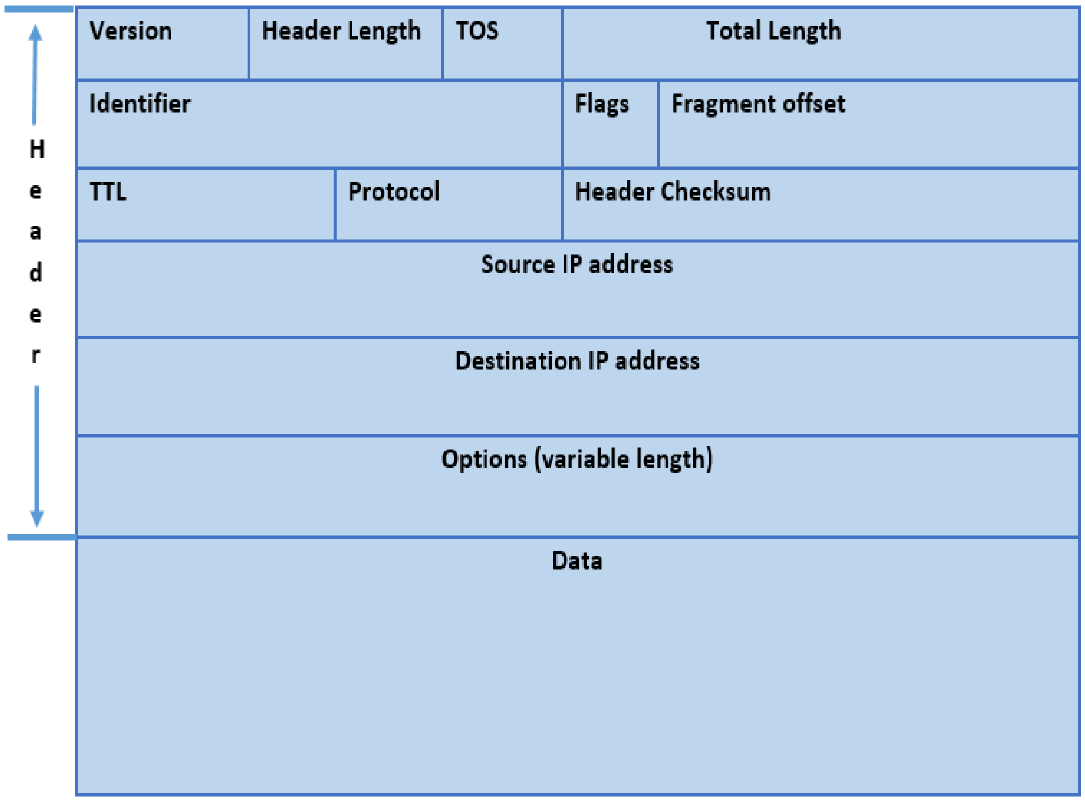


FIGURE 2.1: The structure or format of a network packet. Source [1]

.

2.5 Features used in the classification of network traffic data

As mentioned in the previous section, classification is reliant on suitable features to output accurate results. A classification model or framework may use one or more selected features for classification. In this section, we shall further expound on each feature and its relevance to classification. The features present influence the choice of models. Studies have used a combination of features to ensure accurate classification. For instance, a majority of studies have implemented classification using statistical features. Several studies have used a combination of payload and header features [14], [39],[17], [41]. The studies also show that ML and MLP do not record satisfactory results in comparison to LSTM and CNN. Studies done by Chen et al. [46] and Lopez et al. [40] use time series and header features to classify network traffic using DL models. It requires a relatively low complexity in comparison to using payload and header as it requires a minimum number of packets [40], [37] to ensure accurate classification. Using times series features does not affect traffic datasets even for encrypted traffic [32].

2.5.1 Time Series

Time series or time-related features are dependent on bidirectional flows between source and destination [32], [20]. A flow is defined by several features associated with a flow based on packet/flow inter-arrival time, flow duration, flow bytes per second and so on. The time series feature has been used [20] to distinguish VPN/non-VPN traffic, TCP or UDP flows. In particular, to distinguish a UDP/TCP flow, it is important to note that upon link teardown (FIN packet), TCP flows are typically terminated while UDP flows are terminated by a flow time-out [20]. In the case of distinguishing VPN/non-VPN traffic, the two classes can be distinguished with the flow time outs [20]. Using time-series features is most suitable in a low-computational, offline setting since it results in low computational overheads. It has been shown that these series features are useful in achieving good accuracy [20], [40], [37].

2.5.2 Statistical Features

A dataset collected from network traffic has features in it and each feature is described by a set of statistics and a class that defines the application. Numerous features can be deduced from a flow of traffic such as minimum/maximum packet inter-arrival time for all packets of the flow, the total number of packets sent between client and server, packet lengths and sizes and many more [47]. Using statistical features to classify traffic is most feasible if done offline as it can be limiting since it requires the entire flow. In addition, statistical features for some packets may not be present, as applications that use different protocols and OS platforms coupled with user behaviour can influence the output of these features. For instance, TCP is a connection-oriented protocol and therefore, SYN, ACK and SYNACK packets are sent between source and destination to establish a connection. These packets are not present in a UDP connection. Therefore, these features available in TCP which are useful for classification are not present in UDP data. Time-series and statistical features are not limited as they can be used for both encrypted and non-encrypted traffic [32], [21]. It can be easy to distinguish between encrypted and non-encrypted traffic considering transmission times differs for both classes are different since non-encrypted traffic takes less time to transmit over a network [20].

2.5.3 Header Features

A network packet contains a header and within that header is useful information that is relevant for the classification of traffic. A typical IPv4 packet has 20 bytes of data. Studies done by domain experts have varied what they deemed important. In some studies, such as that done by Lotfollahi et al. [17], the entire packet was used whereas in some studies they just selected the protocol, packet length and protocol, which can be deduced from header information.

2.5.4 Byte/Packet Features - Based on Payload Data

Payload is the data carried in transmission with a packet; the payload is the data excluding the header information. Network payload data is byte format, of which the standard packet has 1480 bytes allowed for maximum transmission over a network, whereas with the header included in a standard packet has a total of 1500 bytes. Recent studies by Rezaei et al. [32] and Lim et al. [15], extract payload data as packets or bytes for classification.

2.6 Machine Learning

Machine learning is a process that automatically learns from data through algorithm training [48]. It is part of artificial intelligence. The genius behind machine learning is in its ability to learn hidden patterns in data. It becomes accurate overtime without being specifically tailored to do so. It is divided into supervised, semi-supervised, unsupervised and reinforcement learning [2], [48]. Supervised ML requires labelled input data paired with desired output, whereas unsupervised and reinforcement learning use unlabelled data. Supervised is more commonly used for classification, regression and ensembling. It is adopted because of its accuracy however it can be time consuming to train ML models and computationally expensive. Semi-supervised uses labelled and unlabelled data. Reinforcement learning just like unsupervised learning does not require labelled data. It uses an action-reward mechanism whereby, a reward is given when a goal is accomplished through certain actions.Unsupervised learning is commonly adopted for clustering, anomaly detection, association mining and dimensionality reduction.

Random Forests, Support Vector Machines, K-Nearest Neighbours, Decision Trees, amongst many others, are popular machine learning algorithms. In this study we adopted supervised ML and DL. SVM model is discussed in detail below.

2.6.1 Support Vector Machine

Support vector machines are supervised mathematical models. Studies such as those done by Fan et al. [49], Cao et al. [50], Chen et al. [46] and Li et al. [51] used SVM's. They are commonly used models because of their ease of implementation and memory efficiency. SVM's can be used for regression, classification and outlier detection. When used for classification, a hyperplane is used to separate data points. A hyperplane is defined by the equation: w * x + b = 0. Though SVM's were initially developed for binary classification, they have also been popularly used in multi-classification. In multi-classification, the same theory is applied as in binary classification. Numerous binary classification examples, commonly referred to as one-vs-one or one-vs-many situations, are used to break down the multi-classification. The Scikit-learn library has the multi-classification option available with one-vs-may set as default. The fundamental essence of classification with an SVM is most readily explained for the simple situation in which there are two linearly separable groups in n-dimensional space. Using the training results, xi, yi, i=1,. . . ..r, yi e1,-1, in the n-dimensional space, with the goal of developing a classifier that generalises accurately. The goal is to find a hyperplane that connects data points to their potential classes in an n-dimensional space. The hyperplane should be as far away from the data points, nonetheless, data points adjacent to the hyperplane are referred to as "support vectors." Support vector machines use a hyper-plane or a boundary between two or more data classes that maximises the margin between the two or more classes. Hyper-planes are decision boundaries. An SVM algorithm looks for a hyperplane in an n-dimensional space with the amount of features to achieve classification.

2.7 Artificial Neural Networks and Deep Learning

Artificial Neural Networks (ANNs) are mathematical structures that are modelled after the brain's biological learning system [52], [2]. Several studies have shown that they are among the most powerful algorithms for modelling dynamic real-world relationships [10], [19] and [53]. Neural networks are made up of an input layer, a hidden layer and an output layer. Within each layer are neurons and connections from one layer to another.

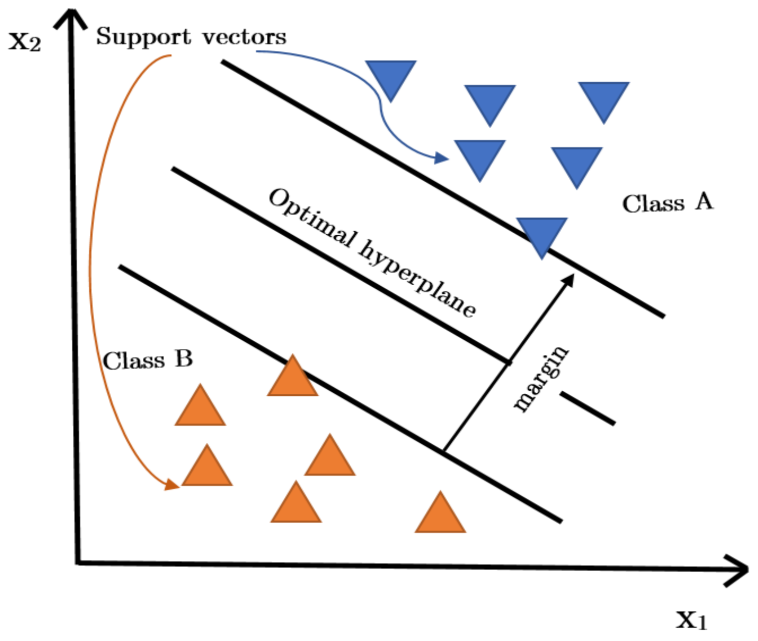


FIGURE 2.2: SVM Architecture, showing classification outcome from training two classes. Source [2]

Each neuron, has a parameter associated with it and after obtaining one or more inputs, it generates an output. These outputs are then moved on to the next layer of neurons, which use them as inputs to their own functions and generate further outputs. The outputs are then transferred on to the next layer of neurons, and so on, until all of the neurons have been considered and the terminal neurons have obtained their feedback. The model's final outcome is then output by those terminal neurons. Furthermore, because of their ability to adapt, they can be used for clustering applications, classification models, and regression models. Deep Learning (DL) is a subset of machine learning. However, the ease of this technique relies on the ability of DL algorithms to automatically learn through training. DL can also be categorised as supervised or unsupervised. With multiple applications, and traffic encryption techniques evolving used to ensure safe transmission of data, DL is more suitable and desirable in comparison to ML. It can process large amounts of data and can quickly learn patterns within the data. DL algorithms try to learn multiple levels of representations by using a hierarchy of multiple layers. ANNs learn in three fashions which are supervised, unsupervised and reinforcement learning. Supervised learning is the simplest of all as it requires labelled inputs. The labelled instances are then used to extract generalisable rules that can be extended to situations that are not classified. In unsupervised learning, the input data is unlabelled. The method instead infers rules and functions from the provided data as well as the network's performance. Deep Learning models are an extension of artificial neural networks however, they are more complex as they more hidden layers. The structure of deeper models is such that it can compute more data in comparison to a simple ANN and require less feature engineering.

2.7.1 Multi-Layer Perceptron

Multi-Layer perceptrons, have a wide range of application in the area of classification regardless of their low accuracy because of their low complexity [32]. Multi-Layer perceptrons can range from one hidden layer to several hidden layers, depending on the complexity of design of the network, having an input layer, hidden layer(s) and an output layer [54]. MLP's have the most basic structure of a neural network. A dataset forms the input to a network, with the output set to be the number of applications/services intended to be predicted and each successive network node calculates its output by applying an activation function that multiplies the linked weight to the sum of the outputs of the previous layer and adds its bias value [4]. The model implements backpropagation to learn again through a process of several iterations. The model training usually requires the network to learn by some version of the gradient descent optimisation algorithm, using gradients determined by the weight parameter values of the backpropagation algorithm that minimise a loss function that captures the discrepancies between the expected output and the output of ground-truth. Categorical cross-entropy is used in the instance of multi-classification. MLPs are common because of this and form the basis for all other deep learning models.

2.7.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are widely regarded as one of the most effective methods for image processing and learning [52]. They have shown state-of-the-art results on classification, segmentation, detection, and retrieval-related tasks [4], [3]. In classification, CNNs process an input and generate a class or a probability that the input is that class ('this input has a 45% chance of being class Google'). The CNN uses 2D convolutional layers to combine learnt information with input data. As a result, a CNNetwork is well suited in the analysis of 2D input. In this study, we used a 2D tensor input to match the 2D layer structure of CNN. Although CNNs can still process data in the form 1D, or 3D inputs [3]. CNNs have a hierarchical feature extraction ability. Hierarchical organisation of CNNs emulates how the neocortex of the human brain extracts features from the underlying data. A typical CNN design consists of convolution and pooling layers alternated with one or more fully linked layers at the end.

Basic Building Blocks of a CNN

- **Convolution Layer:** is the first layer where features are extracted from an input image. It is a mathematical operation that takes two inputs such as an image matrix and a filter or kernel [4]. Decisions to consider when convolving are padding and striding parameters. Convolution of an image with different filters can perform operations such as edge detection, blurring, sharpening etc.

- **Activation Function:** This is a decision function that helps in learning a complex pattern and is used to inculcate nonlinear combination of features. The process of training can be accelerated by choosing the right function [4], [2]. Examples of functions are sigmoid, tanh, maxout, ReLU. However, ReLU and its variants are most preferred as it performs better in most cases.

- **Pooling Layer:** Pooling reduces the dimensionality of each feature map but retains important information [2]. It also reduces the number of parameters and computations in the network. Examples are max pooling, average pooling and sum pooling.

- **Fully Connected layer:** The Fully Connected layer is a traditional Multi Layer Perceptron. The term "Fully Connected" implies that every neuron in the previous layer is connected to every neuron on the next layer. The purpose of the Fully Connected layer is to use these output (high level features) from the previous layer for classifying the input image into various classes based on the training dataset [3].

To improve CNN performance, several learning phases, as well as regulatory units such as batch normalization, data augmentation, and dropout, are generally included.

2.7.3 Recurrent Neural Network

Artificial neural networks containing feedback connections are known as Recurrent Neural Networks (RNNs) [55]. RNNs are used when analysing sequential data or time series data [3]. RNNs are distinguished from ANN's because of how they learn previous information from the last hidden layer along with the time-stamp [2], [3]. RNNs include long-short term memory (LSTM) and gated recurrent units (GRU). RNNs train differently from CNNs with how they retain information during training. Information retention is done through the use of gates. For instance, LSTMs use input gate (which determines which data is kept in long-term memory), output gate (which generates new short term memory

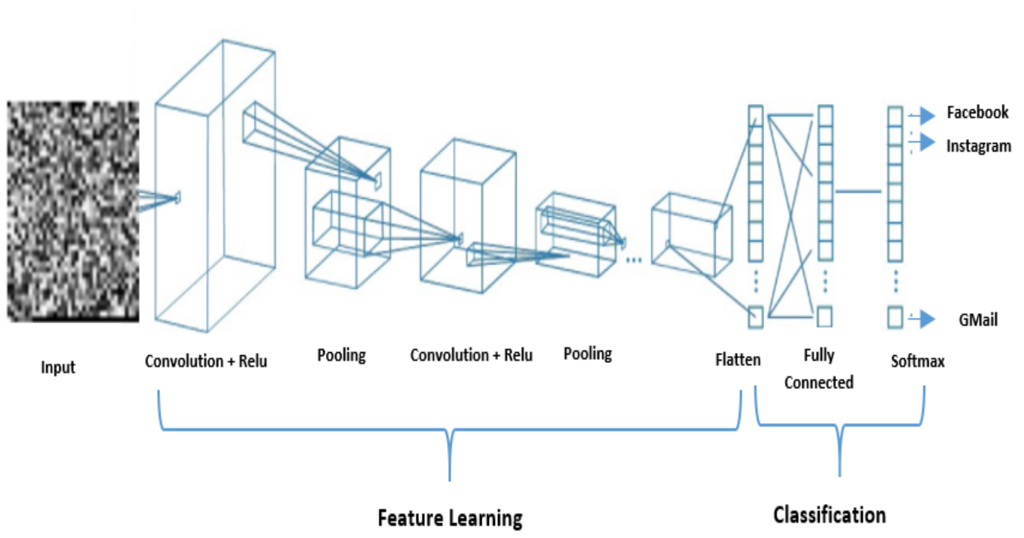


Figure 2.3: Structure of a Convolutional Neural Network with 2 layers. Source [2], [3]

that will be sent on to the cell in the next time stamp) and forget gate (which decides on what is information is to be retained or discarded) to retain information [2]. GRUs similarly use an update gate (which calculates how much of the old data needs to be sent through to the next state) and reset gates (which determine how much of the old data needs to be cut-off) [2]. Sequential data is quite long, and training RNNs on sequential data may result in the vanishing gradient problem. Memory cells are governed by gating units, which control how the memory cell memorises, erases, and exposes information. GRUs use gates, reset and update gates. The gates determine information that should be passed to the output [3]. For instance, the update gate determines what information should be passed into the next sequence whereas, the reset gates decides which information from the past should be left out or forgotten [2].

RNNs' ability to analyse sequential data makes them useful in the application of network traffic classification because network data is sequential in nature.

2.8 Related Work

In this section, our explanation and analogy concentrates primarily on studies that focused on classification using deep learning techniques.

The domain of network traffic classification has few public datasets and numerous frameworks agreed upon as it is nearly impossible for a single dataset to cover all classes of

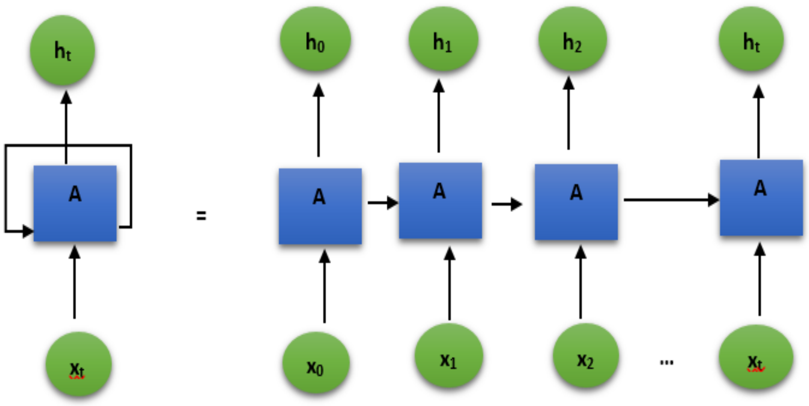


FIGURE 2.4: Structure of a Recurrent Neural Network. Source [4]

traffic. As a result, most published research works use their own datasets. In addition, there is no widely accepted method of data collection, which now leads to a disparity in collection methods causing distinct distributions and data features.

2.8.1 Network Traffic Classification Studies using Machine Learning Algorithms

A number of experiments have been implemented using machine learning. Our work used an ML algorithm as a baseline, hence we also looked into the literature of traffic classification using ML models.

Zhon Fan and Ran Liu [49] experimented with traffic classification using a statistical approach of bidirectional flow traffic with a real research facility dataset containing 248 statistical features extracted from the flow. The dataset with 10 classes: WWW, Mail, Bulk, Services, P2P, Database, Multimedia, Attack,Interactive and Games. The dataset was tested with Support vector machine (SVM) and K-means (KM) clustering. The results of the experiment showed that SVM algorithm out-performed KM, and overall classification results were attained with over 95% accuracy. The emphasis of their research was to test the impact of classification in order to ensure application awareness in Software Defined Networks. Application knowledge is important for features such as virtual network resource slicing and rapid routing. The fact that the extraction process used labelled data was more beneficial to SVM in comparison to KM, which is an unsupervised learning approach. Secondly, the classification results showed the possibility of SVM outperforming KM, when subjected to small portion of data, as KM's precision results were poor for large amounts of data. The conclusion means SVM will do well when implemented in an online scenario though KM will do well with new or unknown data.

Another study [56] looks into building a robust network traffic classification (RTC) tool that classifies zero-day applications (refers to new applications, and they are not present in the training set) given that prior traffic classifiers struggle to correctly classify zero-day applications or end up classifying thos applications as predefined classes. Their framework RTC tool was tested against the four state of the art algorithms: one-class SVM, random forest, correlation-based classification and semi-supervised clustering. The RTC outperformed the aformentioned algorithms in classifying zero-day applications.

Numerous experiments have been carried out using several well-known machine learning algorithms, such as Hoeffding adaptive trees, [57], and support vector machines, [58], [50], [59]. Erman et al. [60] takes a distinguished approach by classifying network traffic through using unidirectional statistical features. Some research has shown the possibility of carrying out online classification through using supervised learning techniques by solely using the first few packets, [61], [62], whereas, some researchers suggested classifying a subflow captured at any given moment, considering the first few flow packets could be skipped or disguised [63], [64].

Among most discussed machine-learning techniques, SVMs have drawn significant interest because of their high precision. SVMs achieve an average accuracy of over 95 %, 2.3 % higher than the best performance of other machine learning methods on the same data sets, as shown by the experiments done by Este et al. [59].

2.8.2 Network Traffic Classification Studies implemented using Deep Learning Algorithms

Throughout the recent years, early internet traffic classification has received attention due to the fact that more and more network applications are encrypted. Deep learning, a subset of machine learning, has the ability to extract features from raw encrypted data by utilising neural networks as discussed earlier in section 2.7. Unlike machine learning, which requires humans to extract the relevant features from data, then predict. Deep learning is distinct because it skips manual feature extraction and uses a neural network to do so. In this section, we discussed studies that utilise deep learning techniques to achieve high traffic accuracy scores. We reviewed studies of interest based on the extraction mechanism, Deep learning algorithms used and the features extracted. We used a table to summarise the discussion in this section.

In a paper by Lim et al.[15], they showed how a DL framework can classify network traffic using a packet-based approach. Training is done using 2-Dimensional Convolutional Neural networks (CNN) and Residual network (Resnets) with a 3 -Layer convolutional group. Their work focuses on using payload data and application layer headers as the features for modelling algorithms. So the dataset with 80000 entries which translates to 8 classes with 10000 random packets per class was converted into 4 pixel image datasets. The dataset contained 8 classes of encrypted and non-encrypted traffic. They transformed the network traffic data to image data in such a way that input totalled packet lengths of size 36, 64, 256 and 1024. We drew insight of transforming network traffic to appear as image pixels, thus our work applied the similar concept of using payload data and using different payload sizes, aiming to classify packets using the first 36 packets of payload data and finding the right payload size that does the best prediction. Precision was measured using the F1-score. The CNN model initially had a better F1-score in comparison to Resnets when fed with a small dataset; however, it was not the case when the dataset grew exponentially. They demonstrated the effectiveness of the two models; however, in our study the use of Resnets may be inapplicable considering how structurally complex they are. Training our data on Resnet may be computationally expensive which is infeasible for real-time classification.

With most researches relying on DL algorithms to use labelled data, Rezaei et al. [37], however, prove how to use a 1D-CNN model trained with unlabelled data to train a small labelled dataset. Their work is an example of using transfer learning, particularly if training data is insufficient. Unlike in Lim at al.'s study [15], Reazaei et al. [37] combines statistical features and time-series features to retrain a public traffic networks datasets known as ISCX dataset [20], which produces 80% accuracy. Accuracy results for their own dataset of the first 20 flows was 68%. The downside of using their method was that it required substantial statistical and time-series features, which meant that they overlooked the fact that data collection method is of paramount importance, as it affects the features. Their work proved the possibility of using transfer learning to achieve classification of traffic using an unlabelled dataset. With the knowledge that DL trains relatively better on large datasets in comparison to small datasets, their work show the significance of using transfer learning to make training of small datasets possible. However, the only disadvantage is that such a setup requires initial and target problems to be similar.

Aceto et al. [14] show the use of Multi-layer perceptron (MLP), Convolutional Neural Network (CNN) and Long short-term memory (LSTM). They confirm the successful application of packet features as input to a model just like in Lim et al.'s [15] study test traffic classification with mobile traffic data. They make their work distinct by adding header features together with payload data. They use the first 784 bytes on payload data. Their mobile traffic dataset is trained with MLP, 1D and 2D CNN, LSTM and SAE. The experiments achieve good accuracy results.

Wang et al [21] point out the precision of Artificial neural network(ANN) and Stacked-auto encoders (SAE) to identify traffic. Their work confirm the use of unlabelled data just like, Rezaei et al. [37]. However, they test their dataset with a different set of algorithms. They used a dataset collected from an internal enterprise network with 0.3 million records of 1000 bytes of flow sessions of network traffic, with 58 classes of protocols after data cleaning. ANN merge information from layers below them and SAE's do not require labels for training and remove redundant information and reduce dimensionality reduction. This experiment achieved good levels of accuracy among all application classes. Precision results among the 25 classes showed over 90 % accuracy with some protocols achieving 100% accuracy. The experiment proving to be successful, their work was left to be tested with P2P applications.

In a study done by Vu et al. [65], they proposed handling imbalanced datasets in order to classify network traffic data. They use a public dataset by [53]. Their research conducted classifying SSH form non-SSH traffic through using 22 statistical features[66]. The flows from the dataset had 35454 SSH flows and 678396 non-SSH flows. In order to balance the dataset, AC-GANs was used to generate synthesized data to balance the dataset. The original dataset with imbalanced classes was combined with the synthesized data to produce a new augmented dataset. Six experimental tests are conducted with SVM, DT and RF as baselines, then SVM and AC-GAN, DT and AC-GAN and RF and AC-GAN on the augmented dataset. RF and AC-GAN produce overall higher results across three metrics which are the accuracy, F1 score and Area under ROC curve. The results are 99.89%, 95.43% and 95.65% respectively. Experimental results show that the 3 supervised learning algorithms (SVM, DT and RF) performed better on the synthesized dat produced by AC-GANs than by a publication done by Vu et al. [67].

Sarhangian et al. [13], show the effectiveness of classifying real-world traffic data using hybrid models. Their data is collected from Installation Support Center of Expertise (ISCX) network, provided by [20]. Implementation is done using CNNLSTM and CN-NGRU as hybrid models for binary and multi-classifcation. The results for binary classification is 99.23% and 93.23% respectively. The result for multi-classification was 67.16%.

Their work shows a trade off between complexity and accuracy. Hybrid models are more complex than single models but achieve high accuracy scores. Our work adapts their CNNGRU hyrbid model because of how CNNs have the ability to learn local attributes and GRU learn local features from former and new data.

In contrast, to using Deep Learning for classification. Some researchers have also focused on using Deep Learning in detecting intrusions, in Software-Defined Networks and many more. For instance, Wang et al. [39], [16] shows the effectiveness of using different Convolutional Neural Networks for malware detection. In their first study [39], they employ 2D-CNN to detect malware in network traffic. They prove that classification cannot only be used to categorise traffic but can be used to detect intrusion in networks. They are able to achieve classification with 20 different classes dataset. Because their study works with a CNN which is an image based model they convert their packet data by transforming the first 784 bytes into a 28x28 grey image, similar to the MNIST dataset. Their architecture is similar to Le-Net [68]. The accuracy scores are high. Their experiments are setup in two ways whereby in the first scenario, a binary classifier is used to classify if traffic is malware or not followed by a two 10 class classifiers. In the second scenario, a single 20 class classifier is setup, which classifies traffic at once. The same architecture is shared amongst the three models except the number of classes 2, 10, 20 respectively in output layer. Their setup achieves an accuracy higher than 99%. However Wang et al [39], instead used a 1D-CNN algorithm. They follow the first setup of using the first 784 bytes of traffic data's flow or session. To ensure an end-to-end encrypted model requires raw features rather than flow features. They use a dataset by Draper et al. [20], which is a class of 12(namely: six VPN and six non-VPN ISCX) dataset. The dataset contains six classes of Email, Chat, Streaming, file-transfer, P2P, VoIP which are VPN and six non-VPN of the same classes. Amounting to a total of twelve classes. Their work was split into 4 experimental proofs namely: classifying VPN from non-VPN traffic, classifying individual classes on non-VPN traffic, classifying individual classes of VPN traffic and lastly classifying all classes belonging to the dataset. The precision of the first three experiments is as follows: 99.9%, 85.8%, , 94.9% respectively. Something worth noting in their work was that dataset imbalance contributes to the result disparity.

Hassan et al. [19], demonstrates the efficiency of a hybrid model termed a CNN-WDLSTM learning model for intrusion detection. For feature extraction, a CNN algorithm is utilised, and WDLSTM is used to identify relationships between those features. The hybrid model outperforms standard singe models such as GRU, CNN, LSTM by reaching a 97% accuracy score. Network traffic classification studies are not only limited to identification of applications alone but can be extended to intrusion detection and anomaly detection.

Kim et al.[69], suggested merging CNN and LSTM models to detect anomalies on web traffic data. This was initiated in order to obtain more complex features. The results show an accuracy of 98.6% and a recall of 89.7%. However, the downside was that identification of anomalies for real data was delayed because of how the pre-processed data uses a sliding window.

In a different study by Lim et al. [18], they prove the effectiveness of a packet-based classification framework for an SDN environment. Their work implements hybrid learning. They test an LSTM model against CNNLSTM hybrid. The LSTM model recorded high accuracy scores compared to CNNLSTM. The experiment used the same dataset and concept from their previous study [15]. We followed the same proof of concept but chose to use only packet data. RNNs have the ability to learn temporal patterns between individual packets. The choice to use GRU was how GRUs learn long sequences just like LSTMs but instead GRUs train better without the exploding or vanishing gradient problem [3].

There are several deep learning algorithms used to classify network traffic that were discussed, including CNN, LSTM, MLP, among many others. The most commonly applied algorithm is the CNN with the hybrid models showing high classification scores. This application is extended in this study to employ a packet-based framework to achieve good results. We conduct our research with similar models to demonstrate the effectiveness of adopting single-DL vs hybrid-DL vs ML techniques. Although the use-case of traffic classification in this thesis is using deep learning techniques compared to using SVM as a machine learning technique baseline, it was important to show how different network traffic features can be applied to deep learning techniques as presented in the scenarios above.

The papers shown in the table summarise deep learning techniques in section 2.8.2 used in different traffic classification studies. The above studies were of interest because different setups used in the experiments resulted in successful outcomes. For instance, classification studies employ different techniques because of the different datasets used and the features utilised for each study. Note that in some of the studies the features were computed based on statistical information for a traffic flow, or for a packet stream or using packet payload information. In some instances, combining the features was found useful in achieving high levels of classification accuracy. Using the payload as a feature for our study was of interest because identifying correlations between succeeding bytes in a payload is critical in identifying patterns in encrypted traffic. On the other hand, reviewing papers of interest in the field of traffic classification was quite a challenge because of how two studies relate primarily because of how they use similar models though use different

TABLE 2.3: Summary of Papers using Deep learning to classify network traffic data.

Summary of Papers reviewed			
CATEGORY	**ALGORITHM TYPE**	**FEATURES**	**AUTHORS**
App identification	CNN, Resnet	Header + Payload	Lim et al.[15]
App identification	CNN, CNN + LSTM	Payload	Lim et al.[18]
App identification	CNN and SAE	Header + Payload	Lotfollahi et al.[17]
Traffic identification	ANN and SAE	Header + Payload	Wang et al.[21]
Intrusion detection	CNN	Header + Payload	Wang et al.[16]
App classification	CNN, LSTM MLP and SAE	Header + Payload	Aceto et al.[14]
Mixed-Type classification	CNN and LSTM	Header + Time-series	Lopez et al.[40]
App identification	CNN	Time-series	Rezaei et al.[37]
Traffic indentification	AC-GAN	Statistical	Vu et al.[65]
App identification	RKHS and CNN	Time series	Chen et al.[46]
Traffic identification	Autoencoder	Header+Statistical	Hochst et al.[70]
Traffic identification	CNNLSTM and CNNGRU	Statistical	Sarhangian et al.[13]h
Anomaly detection	CNNLSTM	Time series and statistical	Kim et al.[69]
Anomaly detection	CNNWDLSTM	Time series and statistical	Hassan et al.[19]

datasets and feature extraction methods. Public datasets for network traffic classification are substantially less developed than those for image processing-related projects, which employ benchmark datasets like ImageNet [71] and Cifar10 [72] and have well-organised training and testing data collections with pre-defined goals. In addition to the problem of scant to non-existent dataset uniformity, the specific research objectives of practically every study are distinctive. Even research work that choose to compare QoS-related goals just in general have very varied definitions of the classes of relevance [73].

2.9 Conclusion

Researchers have contributed to the area of traffic classification using very different approaches. Classifying network traffic constitutes one of the most challenging research topics. Over the years, the solutions for classifying traffic have become more complex, as a proportional reaction to the evolution of the networks.

In the studies reviewed, we noted that Sarhangian et al., [13], Raezai et al. [37] and Lim et al. [15], [18] are the key researches we would like to focus on. We found that for previous studies, experiments conducted using machine learning where limited mostly to statistical information regarding network information and mostly flow data was useful in achieving that goal. Secondly, to achieve classification flow statistics have to be collected offline. the resultant dataset is feature engineered by experts to retrieve information necessary for classification. Hence, such a method is limiting when implementing a live classifier in a network.

With deep learning, raw features as input to our models is useful in gauging if the full payload dataset is useful for training and classifying applications. Different studies focused on using different payload sizes to test the ideal payload size for accurate application prediction. Using packet features rather than flow features was a form of emulating a real-time environment.

Chapter 3

Methodology

This chapter gives details of the steps we took from unstructured network traffic data to refined structured data to efficiently train DL algorithms. We give a detailed explanation of each of the refinement steps such as data collection, preprocessing, labelling and cleaning. We also discuss each of the DL methods used and describe the process of parameter-tuning using grid-search.

3.1 Data Collection

Network data can be accessed from the gateway, or from either the client or server side of the network. The data used in our research was collected from a community network's gateway(router). The community network is situated in Ocean View, South Africa. Since all network packets transit via the gateway before being sent to various destinations, we captured real/live network packets at the gateway using wireshark. Wireshark is a popular tool used to facilitate the smooth capturing of packets flowing through any network. Wireshark is an open source tool commonly used for packet filtering. The collected data was saved in Pcap format. Network traffic data in Pcap format is unidentifiable, hence, tools such as nDPI are useful in reading the Pcap files. We accessed the data by transferring the Pcap files to a data archive to University of Cape Town's server. Pcap is an abbreviation for packet capture and is an application programming interface for capturing live network packet data from OSI model layer 2 to layer 7 [74]. It is also known as libpcap on Unix systems or winpcap on Windows. Network analyzers such as Wireshark or tcpdump are used to create a **.pcapfile**.

3.2 Data Preprocessing

Preprocessing steps included data extraction, labelling, cleaning and exploratory analysis. The details of each procedure is explained in the following sections. Below is a diagram detailing the steps followed:

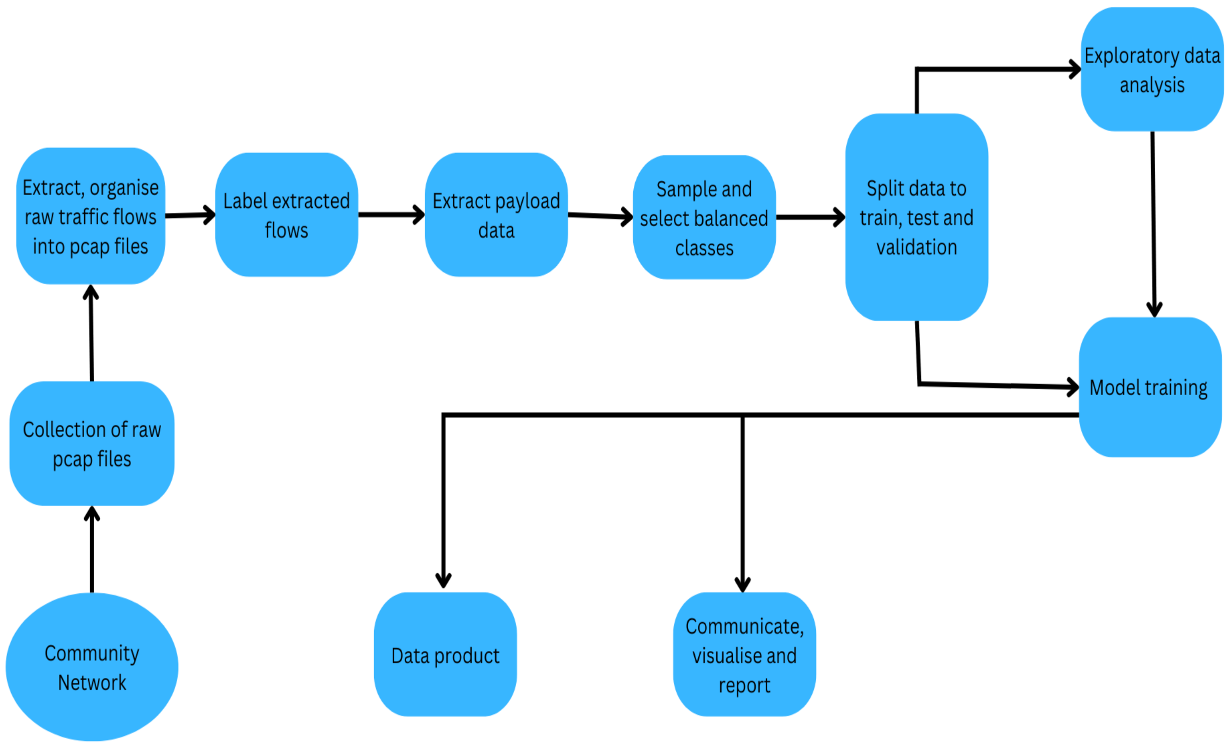

Community Network
Collection of raw pcap files
Extract, organise raw traffic flows into pcap files
Label extracted flows
Extract payload data
Sample and select balanced classes
Split data to train, test and validation
Exploratory data analysis
Model training
Data product
Communicate, visualise and report

FIGURE 3.1: Traffic Classification Model Overview.

3.2.1 Extraction of Data

As previously explained in section 3.1, we collected raw network data from a community network. The data collection was the first important step to establish a network dataset that enabled further experimentation. Next on, extraction of data requires constructing a robust preprocessing pipeline required preprocessing raw network packet data, then organising into flows as discussed in Section 2.1 particularly by protocol (UDP and TCP flows). The extraction process was pivotal in transforming raw network traffic in the form of pcap files to quality data. This is so, because the structure of network traffic as discussed in section 2.4 is collected in its rawest format and contains a lot of information which is irrelevant for machine learning training. Most DL models are used in pattern recognition. In addition, the aim of only extracting payload data is to have the models fit well with unseen network traffic packets. Excluding header information with a consistent structure and values helps to enhance generalisation performance. Training a model with a high degree of generalization can fit the entire data sample space well. Similarly, an example of pattern recognition is the derived network traffic flows that encapsulate payload data. We used a tool called pkt2flow, which streamlines pcap files into flows by selecting the source and destination ip address and the source and destination port [12]. The tools extracted TCP, UDP, non UDP and non-TCP flows. The pcap files produced by pkt2flow were stored in the respective folders to separate TCP from UDP and non TCP/UDP flows.

3.2.2 Data Labelling

This step took the pkt2flow output as input and produced a file with a label for each flow. To train a model, it is a requirement to convert the dataset into a workable format suitable for modelling, such as a csv file. In that regard, we used a tool called nDPI [35]. The tool labelled all flows that belonged to an application/class/service in our dataset. nDPI uses an engine that examines application-layer signatures within a packet stream thereby identifying and labelling application classes. Even though it is an accurate reliable tool, constantly learning up-to-date application signatures is a complex task, given that applications are continuously updated.

3.2.3 Payload Extraction

A more fined-grained approach would be to classify individual packets. A number of network traffic classification studies yield good results and is more plausible when solely working with raw data [17], [43]. We therefore extracted the payload data and masked the header features by using a python library called Scapy [75]. The size of the payload extracted for experimentation was guided by [17]. A standard packet has 1500 bytes with 20 bytes belonging to the header, which means the 1480 bytes are solely payload. As the packets come as a byte string, we decided to interpret the individual bytes as integers in the range [0 - 255]. This way we got a vector of integers in the same length as the original packet. The output csv files produced by nDPI and Scapy were concatenated, to produce a unified csv file.

3.2.4 Data Cleaning

The experiments were hinged on dealing with equal examples for each class in order to predict results. Therefore, the data cleaning process was crucial in the delivery of a clean-balanced dataset. In order to ensure we had a balanced dataset we randomly sampled a given amount of packets from each class. In general we did not want to remove data, given the quality was good, since the general consensus within machine learning tasks is more data results in better results. So we chose 10000 packets per class as a benchmark. The baseline was to work with 10 000 packets for each class. With 13 selected applications or services. The choice to have 13 classes was to vary the dataset size,as previous studies had worked with a maximum of 10 classes. Our main focus was to classify applications and services, excluding protocols was part of the cleaning procedure. Working with 10 000 packets instances per class was necessary as deep learning models require a lot of training examples. Secondly, neural networks are complex structures, a lot of training data is needed to achieve good prediction scores.

3.2.5 Final Structure of Data

The nature of our final corpus/dataset contained rows and columns, where the rows were the instances and the columns were the features. The last column which was the actual label (class/application), is the feature that the models would predict.

3.3 Selected Classes for Experiment

The final dataset as aforementioned had 13 balanced classes. To achieve a balanced dataset, we implemented under sampling [17], a method that gathers abundant packets for the popular classes in order to equalise the classes in the dataset. We opted for this method rather than training on an imbalanced dataset whereby some classes have an uneven distribution of packets because experiments that use imbalanced datasets in the past did not execute well [40]. The selected classes were constituting applications and services that use encryption techniques to transmit data over the network. We chose to include different services and applications because it was important to investigate whether or not it was possible to distinguish between them. To obtain diversity within the data we chose streaming services, text/messaging services, social media services, file-sharing applications and mobile background services. Below is the list of classes with the protocols along with use over the networks:

- **YouTube:** Streaming data from YouTube will be labelled youtube. It is categorised as a video application type. The stream is served via the QUIC 23 protocol over UDP port 443 using QUIC crypto [76], but only if the user is using the Google Chrome browser. If the user is using another browser such as Mozilla Firefox YouTube will stream using HTTPS with TLS v1.2. YouTube use a HTML5 video player to buffer and assemble fragments.
- **Amazon:** Identified as regular web traffic and so it uses HTTPS/TLS/SSL protocol to encrypt traffic.
- **GoogleServices:** It is a background services application type, mostly because of mobile users' activities. Any data the user sends to the Google Front-End is encrypted in transit with Transport Layer Security (TLS) or QUIC, or SSL.

  [77]. The user's request is sent as an HTTPS request.
- **Instagram**: It is messaging platform where mostly users send pictures. When newtork users requests data with instagram it will use SSL/TLS over port 443 [24] to encrypt requests from Instagram servers and will send data over the same encrypted data stream.
- **WhatsApp:** It is a commonly used chat service platform. Uses end-to-end encryption to secure transmission of packets. However, some users use WhatsApp Web, of which any info sent over the web uses HTTPS.
- **NetFlix**: This class's video streams are served using HTTPS and TLS v 1.2 as encryption protocols [78].
- **BitTorrent:** A platform for peer-to-peer file sharing or messaging. Uses HTTP protocol.
- **Teamviewer:** Teamviever is an application that can be used for various purposes such as remote control, desktop sharing, online meetings, web conferencing and file transfer between computer users. The application implements TLS protocol for all standard connections [79]. However, it sometimes uses UDP as a way to connect.
- **GMail:** It is an a emailing or messaging service, that runs on TLS. Google workspace supports TLS 1.0 through to 1.3.

- **PlayStore:** Uses TLS as an encryption protocol. It is an application used on mobile devices.
- **Messenger:**A messaging platform implemented by Facebook. Uses the same protocol as Facebook.
- **Pinterest:** It is an image-sharing and social media service. Uses SSL/TLS and HTTPS.
- **Ookla:** It is a web-service for testing the internet's speed. Implements HTTPS.

Below is an image analysis of the raw packet byte data extracted from the application layer. The procedure transformed the aforementioned classes into images.DL algorithms use images to perform classification, through analysing an image and converting it into thematic maps. The images for all the 13 classes are shown below:

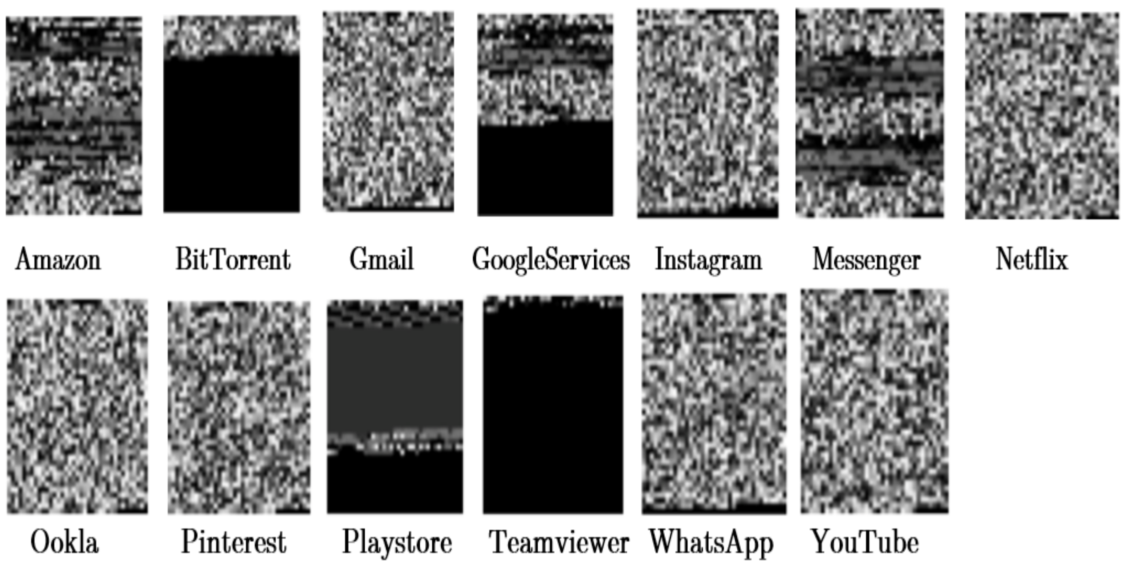


FIGURE 3.2: Images of classes extracted from raw packet data.

More detailed work on model implementation and grid-search parameter tuning used are discussed in the following section.

3.4 Experimental Design

The payload can be either a TCP or UDP payload, and it can be encrypted using SS-L/TLS or can be encrypted at the application layer. Hence, in order to find the most efficient machine learning algorithms in terms of accuracy and efficiency in classifying network payloads, it was necessary to vary model parameters and using grid search in determining the best values for accurate classification. Several factors can affect the efficiency of each of the proposed ML algorithms, including:

1. **Input data for the DL algorithms.** In this study, we used varying tensors as input to the models. This was so, considering the different requirements for each model. Models such as SVM's required single-dimensional (1D) tensor and the CNN and Resnet required two-dimensional(2D) tensors. This was done in order to balance the format of our input data. In particular, standard size contained in a payload packet of 1480 bytes was taken as input of vector length of 1480, whereas, the neural network models required data to be stored as 2D tensors. As such, the same payload length of 1480 bytes was reshaped into a 2D tensor of 40 x 37. Reshaping of an image into a 2D tensor was advantageous as it allowed us to keep the spatial meaning of our data. Reshaping was made possible through scikit-learn's open cv resize method.

2. **Choice of parameters**. A model's parameter is a configuration variable that is intrinsic to the model and whose value can be calculated based on the data provided. When making predictions, the model needs them. Parameter values determine the model's ability to solve the problem. These values are based on data that has been estimated or learned and therefore, saved as a component of the learned model. Parameter values differ from the weight values of a model, to the learning rate, to the epochs, batch size and so on. The specific details and values for each model's parameters are detailed in the following sections.

3. **Amount of training, testing and validation data.** Supervised learning algorithms tend to approximate better when trained on large amounts of data and variance of the models tend to decrease meaning we avoid overfitting. In the event, that a model is trained on little training data, it produces a weak estimate. In this research, the dataset was split in such a way that we had ample training samples, into train, validation, test sets into 64, 16, 20 respectively. With 10 000 datapoints for each application class, the dataset had a total of 130 000 datapoints. The split resulted in 83 200 datapoints for the train set, 22 400 datapoints for the test set and 33 600 datapoints for the test set.

4. **Method of parameter tuning.** We implemented the common approach to searching for the ideal parameters which is known as the grid search [80]. We used a grid search on a subspace of hyperparameters. Our choice to use grid search was to

setup the experiments in such a way that we test every combination of parameter values in comparison to using a random search that would test randomised values from a statistical distribution. The scikit-Learn Grid Search CV module in Python was used to tune the grid search parameters.

Numerical representation of data is the best format and the common way to train models because machine learning models cannot directly work with categorical data, in order to represent our data in the required format, our y-label, which represented the 14 categorical classes in our dataset was represented in a more expressive way, through the process of one-hot encoding.

3.4.1 Experimental Environment

Our work was made possible with software and tools such as scikit-learn, Tensorflow and Keras. The work was implemented on two separate platforms to execute the preprocessing of the network traffic data and the training of models. The preprocessing was done on a Linux machine running Ubuntu 18.04 with 8GB of RAM and Intel processor. The training of the models was done on Amazon Web services (AWS). The data was trained using an NVIDIA GeForce GPU given that we were working with large volumes of data. We used Tensorflow 2.4.1 and Keras 2.3.0 with Python 3.7.

3.5 Adopted models used for Experiments

The following section gives a detailed analysis of the models used for the experiments we carried out. The experiments in this study were designed to determine the most effective DL algorithm in terms of performance accuracy (proposed in section 1.3.) for predicting network traffic classes with respect to a community network dataset.

3.5.1 SVM's Architectural Implementation

The model was implemented in Python and using scikit-learn machine library. The choice to use this library was based on its robustness as it provides a selection of efficient tools for classification. The standard input was 1480 tensors as mentioned in the Experimental design section. Unlike the other models, SVM's architecture had constant parameters because of its constant features. An SVM with linear kernel was used for prediction. An SVM with linear kernel usually have an important parameter such as C that influences performance. Using C as a model parameter assisted in controlling the error, by avoiding mis-classifying training examples. Given that our dataset had a lot of features, using a linear kernel was sufficient because it only searches for the C parameter. For the kernel function, we opted to use a linear kernel function in comparison to other non-linear kernel options because of how they significantly cost more resources to train and offer little to no improvement in predicting performance. To determine the optimal accuracy for the different input packet sizes, grid search was used, see Table 3.1.

Table 3.1: Grid Search parameters for SVM Model.

Parameter boundaries for SVM	
c	1
kernel	Linear
Packet size	36, 64, 256, 576, 1024, 1480

3.6 CNN Architectural Implementation

In our study we built a 2D CNN model, which has 3 simple layers. The model has 3 convolutional layers, activation layers, maxpooling layer and a fully connected layer. The input shape to the model was the a 40x37 image which is meant to suit the 2D requirements of the CNN model. The convolution operation which is a mathematical operation requires filters to be applied to the input array [39]. This is done so that the input array's features are identified. The convolution operation results in shrinkage of the input array, though the most critical characteristics are retained. The result is a feature map. The model was designed to learn features from raw traffic automatically. The features are learnt layer by layer and high-level features are input to the softmax layer. It directly learns from the non-linear relationship from raw traffic and outputs the label. Features learnt are automatically classified through the softmax layer. The structure of the layers of the 3-layer Convnet model is such that the first layer had 32 filters with a filter size of 2 x 2. Rectified linear unit (Relu) was used as the activation function. A maxpooling layer of size 2 x 2, that halves the image dimensions. Subsequently, we stacked two additional layers. With the the second layer having 64 filters and the third layer using 128. The activation layer is an extension of the convolution layer, and, after every convolution operation, the feature maps were passed through non-linear activation functions such as the (ReLU). CNN's activation functions allow it to approximate nearly any nonlinear function. After activation functions have been applied, the output feature maps were fed to the pooling layers. The pooling layers then combined the output feature maps to create a final output feature map. Pooling layers are used to reduce the number of parameters that need to be processed when visualising a feature map. Combining similar features together allows layers to be pooled together semantically. After iterating through multiple levels of convolution, activation, and aggregation, the final result is computed at the fully connected layer of the network. The fully connected layer uses these features to make predictions about the problem. CNNs are similar to most Neural Networks in that they are trained using backpropagation and gradient descent. Our choice to implement the max pooling as our pooling layer was inspired with the studies done by [81], [82]. Their work demonstrated a great success. In addition, in order to implement non-linearity, they used a Rectified Linear Unit (Relu) which influenced our choice of Relu activation function. Adam is the commonly used optimizer as most datasets are significantly large. It is known to reduce training time and has shown great success in a lot of researches [83].

TABLE 3.2: Grid Search parameters for CNN Model.

Parameter boundaries for CNN	
Learning rate	0.01, 0.001 , 0.0005
Batch size	32, 64, 128
Epochs	50
Dropout	0.2, 0.3
Packet size	36, 64, 256, 576, 1024, 1480
Convolution layers	1, 2
Optimizer	Adam
Activation function	Relu

3.7 GRU Architectural Implementation

GRU was implemented with 3 dense layers, with 16 neurons for each layer. The final layer in the structure of the model is connected to a fully connected layer, which multiplies the outputs from the other layers and then adds a bias vector. The outcome is sent to the softmax layer, which is normalises outputs into a probability distribution of the final labels. The input shape for the model was designed to be a 1D tensor of 1480. The models were built up to provide repeatable results, therefore all of the tests began with a fixed random seed. To test the GRU model we chose the Adam optimizer, and Tanh activation function since research has shown that they are very likely to provide positive outcomes.

Kernel regularizer, L2 is applied to apply penalties on layer parameters or layer activity and an early stopping rate with a patience of 2 was used to avoid overfitting. Other parameters were selected with the grid search parameter tuner as shown in Table 3.3.

Table 3.3: Grid Search parameters for GRU Model.

Parameter boundaries for GRU	
Learning rate	0.01, 0.001
Batch size	32, 64
Epochs	50
Packet size	36, 64, 256, 576, 1024, 1480
Dropout	0.2, 0.3
GRU layers	1, 2, 3
Optimizer	Adam
Activation function	Relu

3.8 CNNGRU Architectural Implementation

To obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone, we built a hybrid model which was a combination of CNN and GRU. The ensemble model consists of 2 fully connected CNN layers, a flatten operation before adding a final GRU layer. For each CNN layer contains a pooling operation and convolution operation. The CNN layers are used for spatial features extraction from the features, and then fed into a GRU layer. But before, getting connected to the final GRU layer, the convolution and pooling operation on the last CNN layer converts the high-dimensional data to one-dimensional data which is connected to a fully connected layer. Pooling layer is employed to model the acquired feature maps and convert them into features to a more abstract form. GRUs use gated recurrent units which are connected to the fully connected layer. We used a fixed pooling size for the CNN and CNNGRU. The padding for the model was fixed to be the same for a small input shape of 36 otherwise it was set to valid. The models were set up to produce consistent results, so a fixed seed was used [52]. We used the Adam optimizer and Tanh activation function for all of our experiments because studies have shown them to be very likely to produce good results. Other parameters were selected using the grid search parameter tuner. See Table for more information. The grid search tuner found the best parameters for a specific class, which were then used to make a classification prediction for that class. We used the Adam optimizer and the Relu activation function in all experiments, as it is known to achieve accurate results in most hybrid model prediction studies [37]. Other parameters were selected with the grid search parameter tuner.

TABLE 3.4: Grid Search parameters for CNNGRU Model.

Parameter boundaries for CNNGRU	
Learning rate	0.01, 0.001
Batch size	32, 64, 128
Epochs	50
Packet size	36, 64, 256, 576, 1024, 1480
Dropout	0.5 ,0.7
Number of neurons	16, 16, 16
Convolution layers	2
GRU layers	1
Optimizer	Adam
Activation function	ReLU and Tanh

3.9 Experiment 1: Predicting the Best Model for classifying the network applications

The first experiment compared how Support Vector Machines, CNN, GRU and CNNGRU predicted network applications into respective classes. The three models were fed with the standard input data of 1480 bytes, with 10 000 network packet samples represented application class being the standard number. The specifications and details of the models were discussed earlier in the previous sections. The experiments were carried out using different input feature sizes. After pre-processing, there were 130 000 instances across all 1480 features available for training, validation and testing. This was divided in the ratio 64%:/16%:/20% for training, validation and testing respectively. Each algorithm was trained and tested with a specific set of parameters. These parameters are specified in the previous sections of architectural implementations for each model. Accuracy, precision and recall are used as a performance metric.

3.10 Experiment 2 Setup: Predicting how the rate of change of data size affects the accuracy rate

The second experiment was implemented to evaluate how each model performed when increasing the number of packet bytes used as features. We varied the payload feature shapes across all the models to check their performance and recorded the accuracy. According to [15], they varied packet features to 36, 64, 576, 1024 and 1480. Along with the different packet streams, with different parameters as a combination we used parameter tuning to train the adopted algorithms. We evaluated outputs of the best parameters based on a given performance metric. We used accuracy as the performance metric for this study. The Grid search parameter tuning was implemented with the python Scikit-Learn Grid Search CV package. The grid search tuning was implemented for each model. The input data used for the grid search tuner was the standard sizes across all models with the last column as the labelled output. See Table 3.1 -3.4 for the list of parameters tuned for each model. SVMs were one of the ML algorithms that was proposed for this study and are described in Chapter 2. Details are included in section 3.5.1 and Table 3.1 has parameter details used for gridsearch. SVM was implemented using scikit-learn. The best parameters selected by the grid search tuner for different input sizes was selected. CNNs, GRU and CNNGRU were also proposed for the study. All DL models were implemented with the Keras Deep-Learning Library and TensorFlow backend. The models were configured to make reproducible results, thus a fixed random seed was set for all experiments. We used the Adam optimizer and activation functions (Tanh and ReLU) for all our experiments because they have been known to achieve accurate results for most CNN prediction studies [37]. Other parameters were selected with the grid search parameter tuner. See Table 3.2 - 3.4 for more details. The best parameters selected by the grid search tuner for a specific input shape or feature size. Before training, the data pre-processing technique explained in section 3.2 was implemented.

3.11 Experiment 3 Setup:Predicting classification rate when model parameters are varied

Including prediction speed and evaluating model complexity as part of the experiments was fundamental to test the efficiency of this simulation if it where to be implemented in a live setting. The calculation of the prediction speed in packets per second used the formula below:

$$Packetspersecond = \frac{1}{Averagetimetaken} \tag{3.1}$$

To clearly assess the effect of the prediction performance, we evaluated the prediction speed against the different packet sizes across the models. Moreover, the number of parameters is an exponential scale that comprehensively covers a sample area. If you increase the number of weights, neurons, or batch size, your model will be more complex by having more parameters. The parameters are discrete or continuous values that can be adjusted in order to obtain the optimum model performance. For each model with a given number of parameters, we randomly vary some of those parameters in order to find the best fit. To this end, the grid search involves tuning the learning rate specified in the Tables 3.1 - 3.4 since learning rate is widely regarded as the most important hyperparameter to tune for neural networks [52] as well as the dropout rate to identify what level of dropout is desirable to reduce over-fitting for a particular configuration [52]. When the highest validation score is attained, we select that model as the optimum. All the models are trained using 50 epochs and with an early stopping rate to prevent overfitting.

3.12 Evaluation Metrics

1. Accuracy: Firstly, an evaluation metric for indicating classification success of a given model is needed. One of the most used metrics for evaluating classifiers is accuracy, which measures the proportion of packets correctly identified to belong to a certain class to the total packets predicted in an experiment. Therefore, Accuracy was a key metric used in performance evaluation of the models.

$$Accuratepredictions = \frac{numberofcorrectpredictions}{Totalpredictions} \tag{3.2}$$

The equation above can also be translated as:

$$Accuratepredictions = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.3}$$

- TP (True Positive) is expressed as a proportion of packets that were accurately identified as being in a certain class.

- TN (True Negative) is defined as the proportion of packets that were accurately identified as not falling under a certain class.
- FP (False Positive) is calculated as a percentage of packets that were mistakenly assigned to a certain class.
- FN (False Negative) is defined as the percentage of packets that are accurately identified as belonging to a certain class but are actually classed as not belonging to that class.

2. Confusion matrix A confusion matrix is a N x N matrix, where N is the number of classes. In our study, we had a 13 x 13 matrix. We used a confusion matrix as a way to assess how well the model predicted each individual class in our dataset. It is a way of assessing the correct (actual) predictions versus the incorrect (predicted) outcomes. Additionally, we can calculate the precision, recall and F1-score using the True positive, True negative, False positive and False negative previously defined in equation (3.3). Precision is calculated as the proportion of accurately identified traffic flows belonging to a class out of all traffic flows identified as belonging to a class. Recall is calculated as a percentage of all packet streams properly identified as being in a class out of all packet streams in a class. The F1 Score is a harmonic mean of both precision and recall and gives equal weighting to both precision and recall.

$$Precision = \frac{TP}{TP + FP} \tag{3.4}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.5}$$

$$F1Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.6}$$

3. Packets classified per second: We also used time as a metric for evaluating classification speed, this was done to find a means of assessing how models perform in the event they are used in a real-time setup. For this purpose, we used time taken for prediction done by a model. Equation 3.1 summarises this evaluation metric.

Table 3.5: Overview of all Experiments.

Summarised experiment evaluation criteria				
Experiment Name	Experiment description	Method of Parameter Tuning	Parameters Tuned	Research Question answered
Experiment1	Classification with full payload size	Grid Search	refer to Table 3.2 -3.5	refer to Chapter1, Section1.4, Que1
Experiment2	Classification with different payload sizes	Grid Search	refer to Table 3.2 -3.5	refer to Chapter1, Section1.4, Que2
Experiment3	Prediction speed comparison	Grid Search	refer to Section 3.16	refer to Chapter1, Section1.4, Que3

3.13 Conclusion

In this study, we used one ML algorithms (SVM) and three DL algorithms(CNN, GRU and CNNGRU) to classify community network traffic. The approach that was used to design our models was similar to what have been done in previous traffic classification studies for different networks with different traffic volumes and classes however, there were some slight peculiarities. For instance, in designing our solution we considered factoring in model predictive performance calculated as packets per second. We found incorporating such a solution as a relevant way to evaluate the most efficient model that can classify traffic within a low-bandwidth environment.

Chapter 4

Analysis of Results

This chapter presents the experiment outcomes from the preceding chapter's experimental setup. The sections below describe the key findings to determine the solutions to the research questions in Chapter 1, Section 1.3. The results explained from section 4.1 to 4.3 used the final structured dataset with 83 200 instances for the train set, 26 000 instances for the test set and 20 800 instances for the validation set. The fact that we worked on implementing a streamlined preprocessing pipeline that produced 130 000 rows of structured data from the unstructured pcap data. The structured data could be used as input for our models which was a good contributor for this project. Hence, building our own dataset to see how it was possible to test literature on the possibilities of classifying network traffic data, in such away that we could distinguish classes of traffic flows

4.1 Experiment 1: Maximum Accuracy prediction and evaluation of the experimental study

Our first research objective from Chapter 1 in section 1.3 was to: Investigate the effectiveness of existing deep-learning models (CNN, GRU and CNNGRU) against an existing machine-learning model (Support Vector Machines) using a packet-based classification approach for accurately classifying network traffic collected from a community network. We conducted an experiment that could test the accuracy prediction of Machine Learning against deep learning. We adopted and tested ML and DL models. The effectiveness is given by accuracy which is the number of correct predictions over the total class * 100 %. The results displayed are from the test set of 26 000 samples. The test set indicates the evaluation performance of the models and how it generalises well with new or unseen data. From this experiment, we discovered that the best performing DL model amongst them all was the hybrid model, CNNGRU, followed by GRU, then CNN. According to Figure 4.1 shown, we can depict that Deep Learning models could generalise well with the traffic data in comparison to SVM. The accuracy percentages were as follows: CNNGRU had a value of 91%, with GRU following at 87%. CNN had an accuracy prediction of 85%. Lastly, SVM had an accuracy of 51%. The difference in accuracy values across all deep learning models was marginal, with a 2% difference. However, the major difference was between SVM and other Deep learning models.

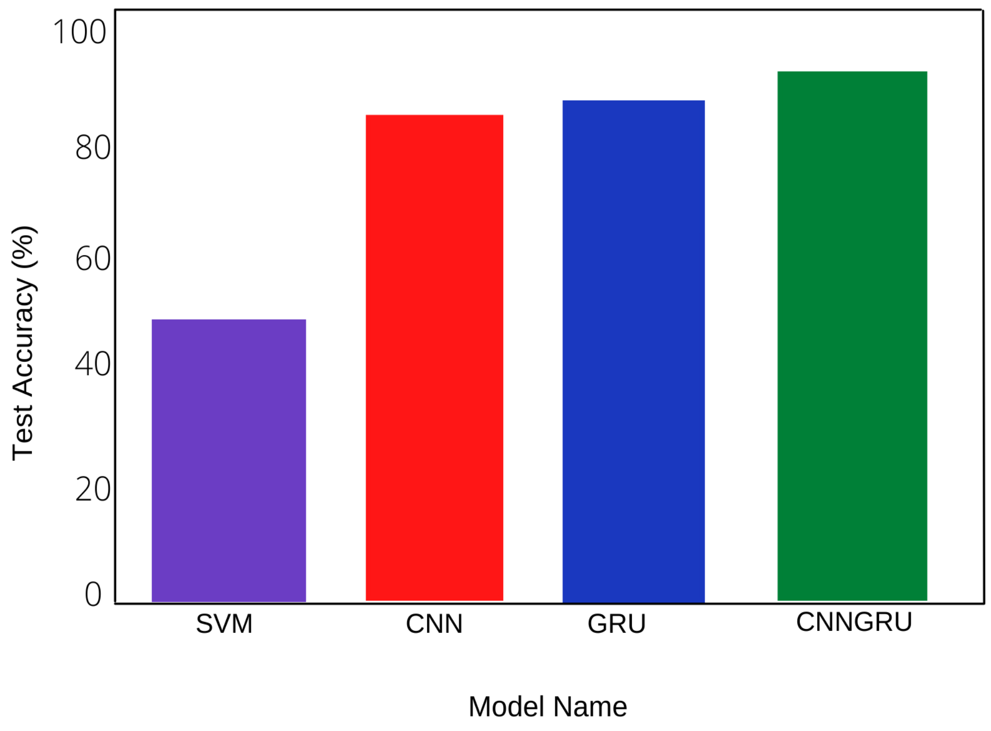


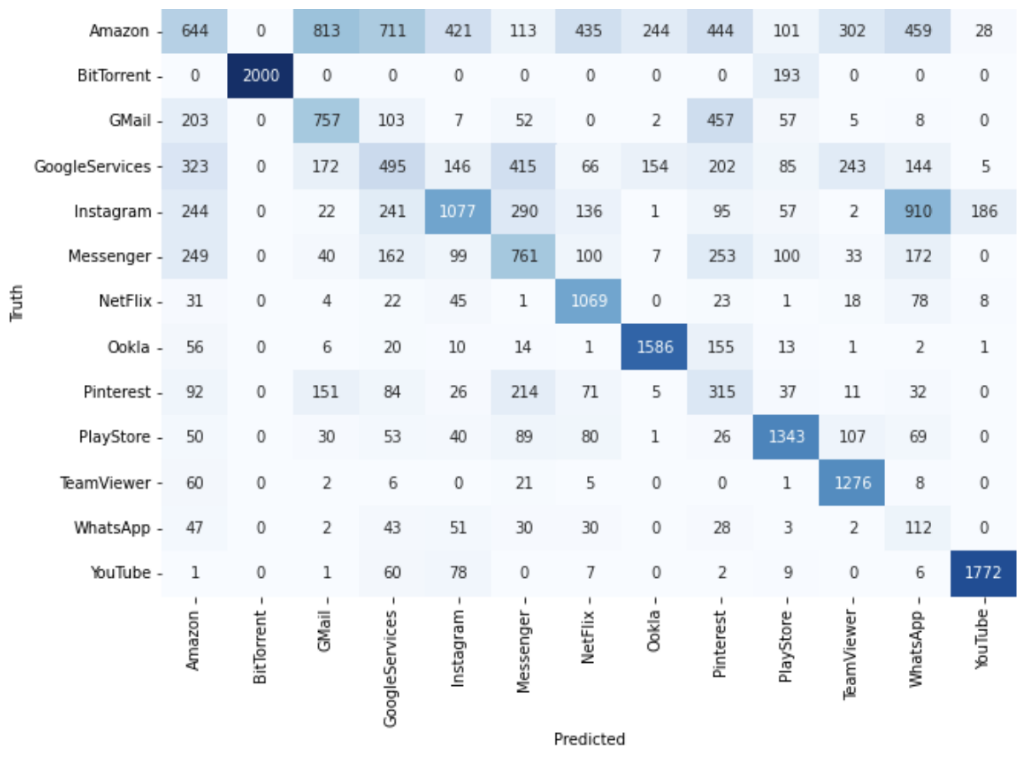
FIGURE 4.1: SVM, CNN, GRU and CNNGRU test accuracy results

4.1.1 Confusion matrix results for SVM, CNN, GRU and CNNGRU

Following the results above in Figure 4.1, we further explored the confusion matrix results for each of the models. Figure 4.2 -4.5 describes each of the outcomes. The results shown are for the best model for each model type. The matrix shows the correctly classified packets/bytes belonging to a class against the total bytes belonging to each class. The values of the correctly predicted classes is known as the true positive values and these are represented by the high values in the diagonals, where a predicted application class on the y-axis coincides with the same application class name on the x-axis, with a lighter blue showing a low true positive value and the darker blue showing a high true positive value. Figures 4.2 to 4.5 represent the outcome classification prediction of SVM, CNN, GRU and CNNGRU respectively. We found that the general trend across all model prediction was that certain classes were accurately classified in comparison to some classes. This means that when a model predicted a class as X, the predicted outcome was truly class X. The outcome for each class prediction is represented in the diagonal of the matrix. The general trend observed over each confusion matrix outcome is such that BitTorrent, Youtube, Teamviewer, WhatsApp, Ookla, Instagram and Netflix were the applications with predictions recorded with over 1800 instances true positive values. Across all the different DL models, BitTorrent was consistently classified with a high true positive rate.

The graphical representation of SVM in Figure 4.2, shows the true positive prediction for each of the 13 application classes. BitTorrent, OOkla, PlayStore, Teamviewer, and YouTube had high true positive predictions.The misclassified results are represented by the values in the corresponding rows and the values in the corresponding columns. For instance, the values in the corresponding row of Amazon are the False positives packet instances of Amazon. The false positive values for a class are represented by the sum of the values in the corresponding row. In the instance of Amazon the false negative value summed up to 4 129 false positive instances of Amazon. There were 1412 total false negatives instances misclassified instances of Amazon. The True negative values are represented by all the values except the ones in the corresponding rows and columns.

Figure 4.3 shows the CNN confusion matrix shows the true positive prediction, false positive, false negative and true negative for each of the 13 application classes. The graphical representation of CNN shows the true positive prediction for each of the 13 application classes. BitTorrent, GMail, Messenger, Netflix, Ookla, PlayStore, Teamviewer, WhatsApp and YouTube had high true positive predictions.The misclassified results are represented by the values in the corresponding rows and the values in the corresponding columns. For instance, the values in the corresponding row of PInterest are the False positive values of Interest. The false negative values for a class are represented by the sum of the values in the corresponding column which summed up to 209 false positive misclassified instances of Pinterest. There were 725 false negative instances misclassified



Truth \ Predicted	Amazon	BitTorrent	GMail	GoogleServices	Instagram	Messenger	NetFlix	Ookla	Pinterest	PlayStore	TeamViewer	WhatsApp	YouTube
Amazon	644	0	813	711	421	113	435	244	444	101	302	459	28
BitTorrent	0	2000	0	0	0	0	0	0	0	193	0	0	0
GMail	203	0	757	103	7	52	0	2	457	57	5	8	0
GoogleServices	323	0	172	495	146	415	66	154	202	85	243	144	5
Instagram	244	0	22	241	1077	290	136	1	95	57	2	910	186
Messenger	249	0	40	162	99	761	100	7	253	100	33	172	0
NetFlix	31	0	4	22	45	1	1069	0	23	1	18	78	8
Ookla	56	0	6	20	10	14	1	1586	155	13	1	2	1
Pinterest	92	0	151	84	26	214	71	5	315	37	11	32	0
PlayStore	50	0	30	53	40	89	80	1	26	1343	107	69	0
TeamViewer	60	0	2	6	0	21	5	0	0	1	1276	8	0
WhatsApp	47	0	2	43	51	30	30	0	28	3	2	112	0
YouTube	1	0	1	60	78	0	7	0	2	9	0	6	1772

FIGURE 4.2: Confusion matrix for SVM Algorithm.

instances of Amazon. The True negative values for PInterset are represented by all the values except the ones in the corresponding rows, columns and the True positive value.

Figure 4.4 shows GRU confusion matrix has high true positive predictions for BitTorrent: 2000, GMail: 1792, Messenger: 1829, Instagram: 1604, Netflix: 1869, Ookla: 1983, PlayStore: 1562, Teamviewer: 1981, WhatsApp: 1723 and YouTube: 1976 had high true positive predictions. The misclassified results are represented by the values in the corresponding rows and the values in the corresponding columns. For instance, the values in the corresponding row of Instagram are the False negative values of Instagram. The sum of false negatives are 1326 for lnstagram. There were 388 false positive instances misclassified instances of Instagram. The True negative values for Instagram are represented by all the values except the ones in the corresponding rows, columns and the True positive value.

Figure 4.5 shows the graphical representation of CNNGRU, showing the true positive prediction for each of the 13 application classes. BitTorrent: 2000, GMail: 1713, Instagram: 1892, Messenger: 1749, Netflix: 1948, Ookla: 1984, PlayStore: 1721, Teamviewer: 1699, WhatsApp: 1840 and YouTube: 1991 had high true positive predictions.The misclassified results are represented by the values in the corresponding rows and the values in the corresponding columns excluding the true positive value. For instance, Google Services had

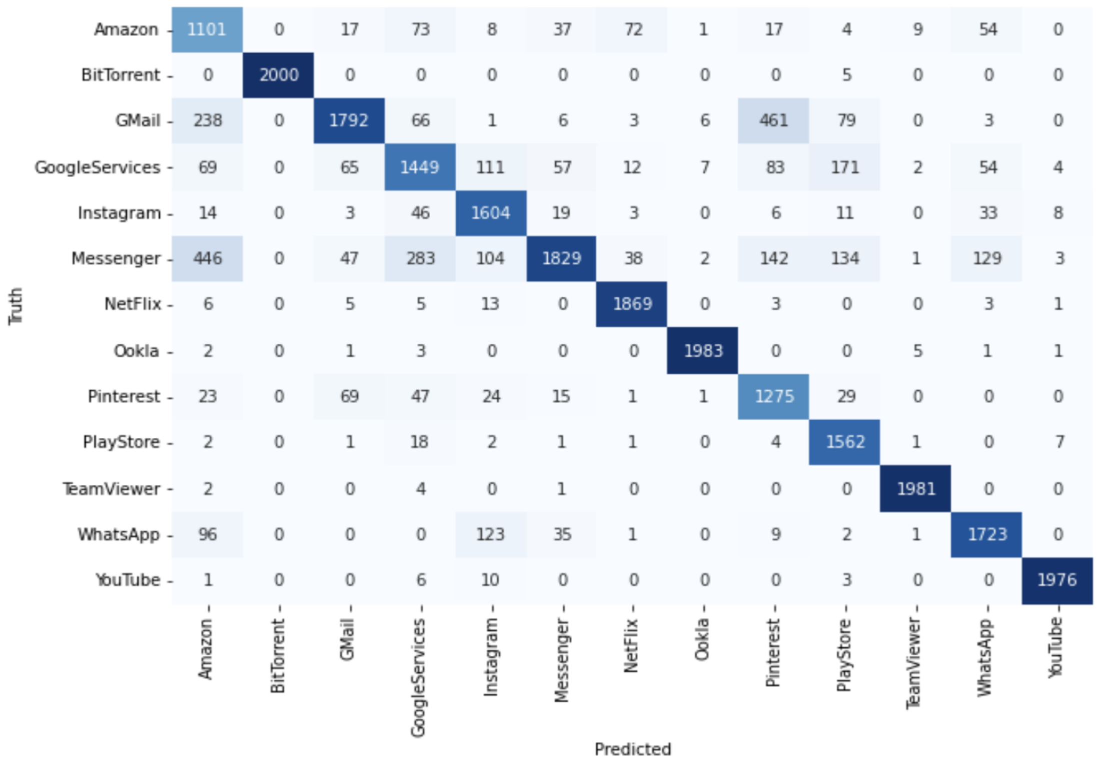


FIGURE 4.3: Confusion matrix for CNN Algorithm.

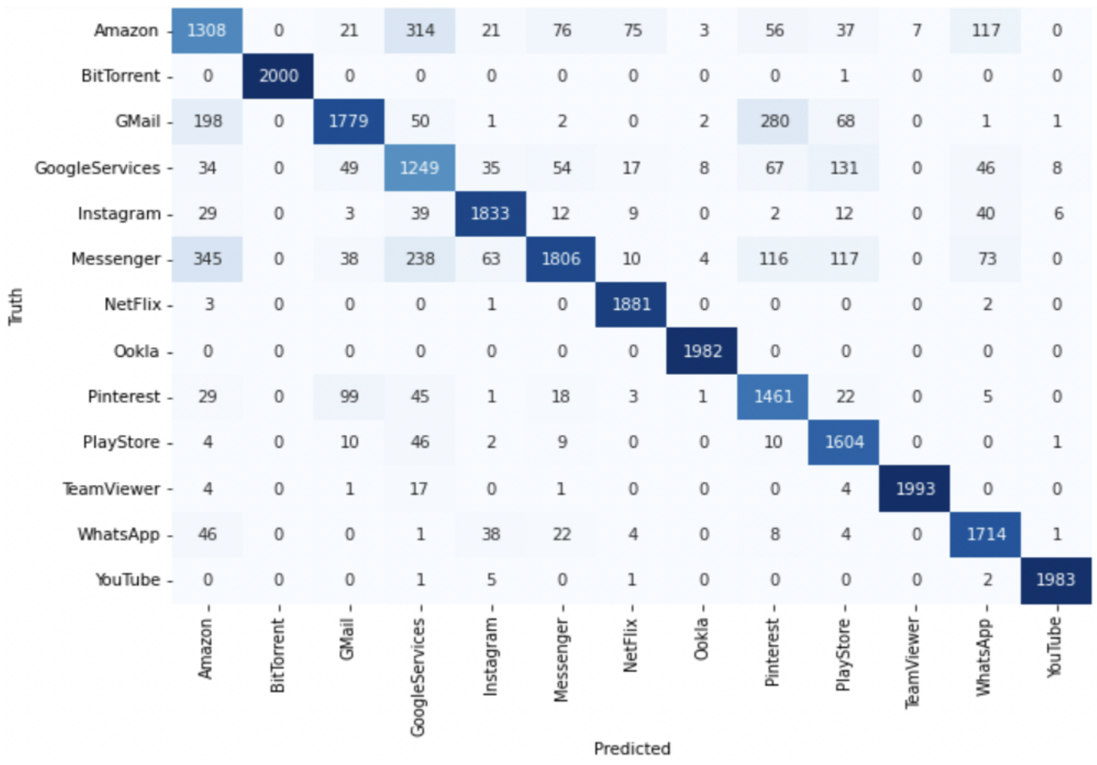
288 false negative misclassified instances of GoogleServices. There were 430 total false positive instances misclassified instances of GoogleServices. The True negative values are represented by all the values except the ones in the corresponding rows and columns.

4.1.2 Tradeoff of Precision, Recall and F1 Score for each Model

Following the results from Figure 4.2 to 4.5. We further inferred the precision, recall and F1 Scores for each class. Table 4.1 shows precision, recall and F1 Scores results that belong to SVM per for each of the 13 classes. The trend shows that the precision scores were generally poor in comparison to the recall scores, meaning the SVM model was efficient in identifying traffic classes as belonging to a class out of all packet streams in a class.

Table 4.2 shows precision, recall and F1 Scores results that belong to the CNN per for each of the 13 classes.

Table 4.3 shows precision, recall and F1 Scores results that belong to the GRU per for each of the 13 classes.



Truth \ Predicted	Amazon	BitTorrent	GMail	GoogleServices	Instagram	Messenger	NetFlix	Ookla	Pinterest	PlayStore	TeamViewer	WhatsApp	YouTube
Amazon	1308	0	21	314	21	76	75	3	56	37	7	117	0
BitTorrent	0	2000	0	0	0	0	0	0	0	1	0	0	0
GMail	198	0	1779	50	1	2	0	2	280	68	0	1	1
GoogleServices	34	0	49	1249	35	54	17	8	67	131	0	46	8
Instagram	29	0	3	39	1833	12	9	0	2	12	0	40	6
Messenger	345	0	38	238	63	1806	10	4	116	117	0	73	0
NetFlix	3	0	0	0	1	0	1881	0	0	0	0	2	0
Ookla	0	0	0	0	0	0	0	1982	0	0	0	0	0
Pinterest	29	0	99	45	1	18	3	1	1461	22	0	5	0
PlayStore	4	0	10	46	2	9	0	0	10	1604	0	0	1
TeamViewer	4	0	1	17	0	1	0	0	0	4	1993	0	0
WhatsApp	46	0	0	1	38	22	4	0	8	4	0	1714	1
YouTube	0	0	0	1	5	0	1	0	0	0	0	2	1983

FIGURE 4.4: Confusion matrix for GRU Algorithm.

TABLE 4.1: Accuracy, Precision, Recall and F1 Score for SVM Model.

Class	**Precision(%)**	**Recall (%)**	**F1 Score (%)**
Amazon	13.7	32.2	19.2
BitTorrent	91.2	100	95.4
Gmail	35.7	37.9	37.8
GoogleServices	20.2	24.8	22.3
Instagram	33	53.9	40.9
Messenger	38.6	38.1	38.3
NetFlix	82.2	53.5	64.8
Ookla	85	79.8	82.3
Pinterest	30.3	15.8	20.7
Playstore	71.1	67.2	68.6
Teamviewer	92.5	63.8	75.5
WhatsApp	27.1	5.6	9.3
YouTube	91.5	88.6	90

Table 4.4 shows precision, recall and F1 Scores results that belong to CNNGRU model per for each of the 13 classes.

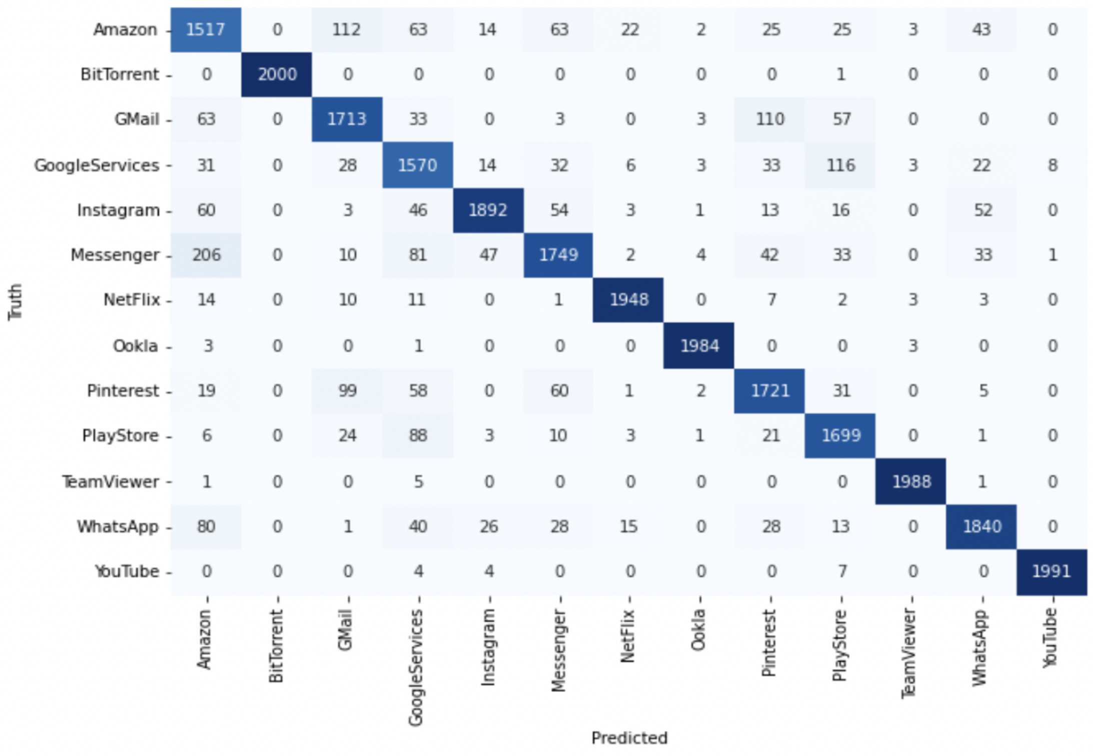


FIGURE 4.5: Confusion matrix for CNNGRU Algorithm.

TABLE 4.2: Accuracy, Precision, Recall and F1 Score for CNN Model.

Class	**Precision(%)**	**Recall(%)**	**F1 Score(%)**
Amazon	79	55.1	64.9
BitTorrent	99.8	100	99
Gmail	67.5	89.6	77
GoogleServices	69.5	72.3	70.9
Instagram	91.8	80.2	85.6
Messenger	60	91.5	72.5
NetFlix	98.19	93.5	92.3
Ookla	99.3	99.2	99.2
Pinterest	86	63.8	73.25
Playstore	97.7	78.1	86.8
Teamviewer	99.6	99.1	99.3
WhatsApp	86.6	86.2	86.4
YouTube	99	98.8	98.9

Table 4.3: Accuracy, Precision, Recall and F1 Score for GRU Model.

Class	**Precision(%)**	**Recall (%)**	**F1 Score (%)**
Amazon	39.1	65.4	48.9
BitTorrent	99.9	100	99.5
Gmail	76.7	89	82.4
GoogleServices	73.6	62.5	67.6
Instagram	92.3	91.7	92
Messenger	64.3	90.3	75.1
NetFlix	99.7	94.1	96.8
Ookla	100	99.1	99.5
Pinterest	86.8	73.1	79.4
Playstore	95.1	80.2	87
Teamviewer	98.6	99.7	99.1
WhatsApp	88.9	85.7	87.3
YouTube	99.5	99.2	99.3

Table 4.4: Accuracy, Precision, Recall and F1 Score for CNNGRU Model.

Class	**Precision(%)**	**Recall (%)**	**F1 Score (%)**
Amazon	80.3	75.9	78
BitTorrent	99.9	100	99.9
Gmail	86.4	85.7	86
GoogleServices	84.1	78.5	81.2
Instagram	88.4	94.6	91.4
Messenger	79.2	87.5	83.1
NetFlix	97.4	97.4	97.4
Ookla	99.6	99.2	93.4
Pinterest	86.2	86.1	86.2
Playstore	91.5	85	88.1
Teamviewer	99.6	99.4	99.5
WhatsApp	88.7	92	90.3
YouTube	99.3	99.6	99.4

4.2 Experiment 2: Accuracy prediction across different packet sizes

In order to answer the research question in Chapter 1, section 1.3 which states that: Test to what extent would varying different packet sizes would have an affect on the classification score of the deep learning models. We conducted a grid search experiment across the different packet lengths/input shapes of 36, 64, 256, 576, 1024 and 1480. The grid search tested the combinations of the hyperparameter values of interest regarding the input shapes. Comparing the results obtained, we observed that each model performed differently across the range of input shapes, shown by the highest accuracy that each model class attained for each input shape. Figures 4.6 and 4.7 show a barplot and boxplot showing the classification accuracy's for each of the input shapes. The boxplot shows the accuracies attained per each packet size. For CNN, the accuracy values from packet sizes: 256 to 1024 ranged from 82% to 84.23%. For GRU, the accuracy values from packet sizes: 256 to 1024 ranged from 83% to 85.32%. For CNNGRU, the accuracy values for packet sizes: 256 to 1024 ranged from 83% to 90.22%.The following values were recorded for the different packet input sizes from 36, to 1480 for the SVM model was 42% to 51% respectively. For CNN, the accuracy values ranged from 75% to 85%. For GRU, the accuracy values ranged from 77% to 87%. Lastly, CNNGRU ranged from 79.2 to 91%. Lastly, SVM had accuracy values from 42% to 51%.

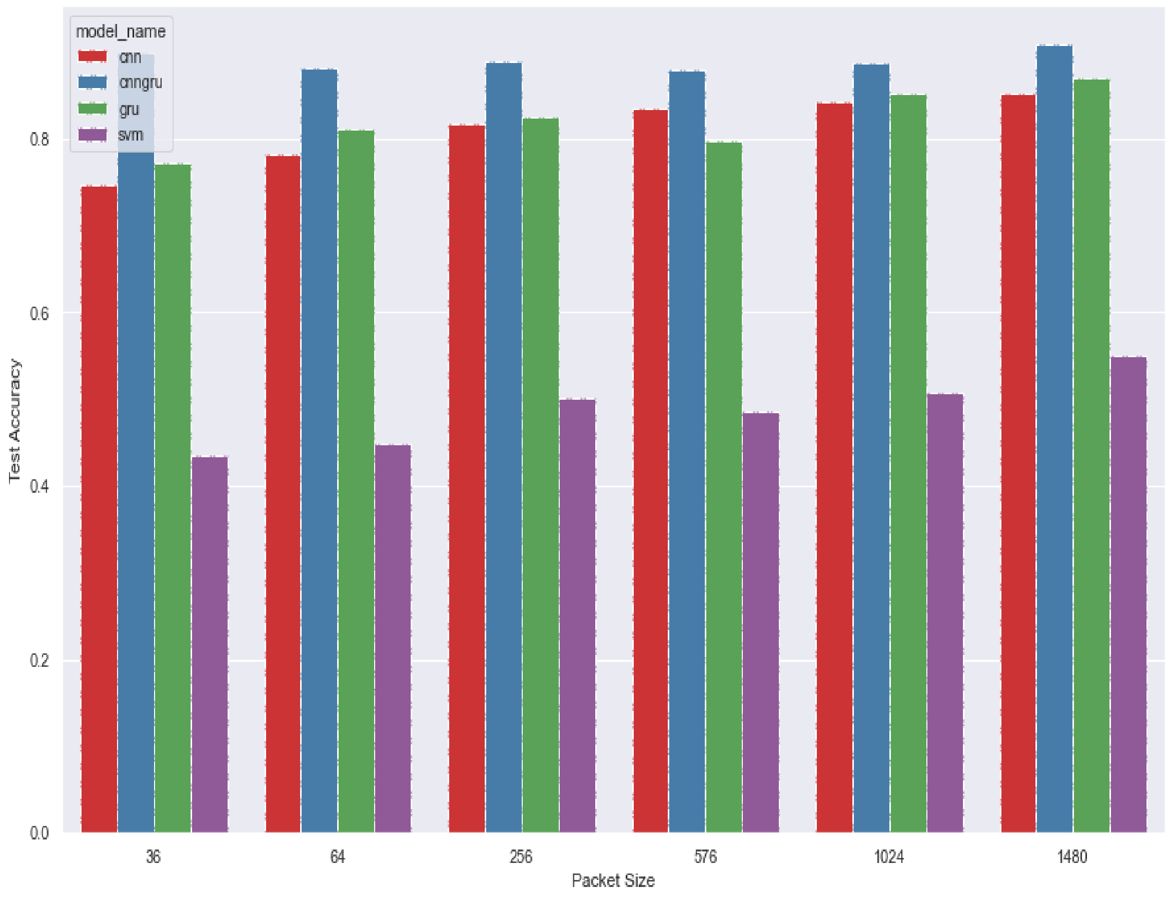


FIGURE 4.6: SVM, CNN, GRU, CNNGRU highest percentage accuracy achieved for different packet sizes.

Although in Figure 4.6, we represented the maximum accuracies achieved across the different models, in Figure 4.7, we observe the distribution of the test accuracy for each model is vastly different. SVM had the lowest test accuracy and the least distribution which is seen with the significantly low accuracy scores in comparison to the rest of the models. The CNN model's prediction is more dependable because there was less variability across all packet sizes. The median accuracy values were close to 73% for the first four input shapes. The median values drops for the packet size of 576 though the maximum test accuracy is maintained. The GRU outcome can be seen to have a fluctuating median value from as low as 54% to 83%. The test accuracy values are not as evenly distributed across each packet size. The CNNGRU has evenly distributed test accuracy scores for the first three packet sizes. The test accuracy scores for the last packet sizes are unevenly distributed. Overall, the median scores across each packet size for the different models is vastly different. With CNN and CNNGRU recording consistent test accuracy values for the 25% highest accurate values.

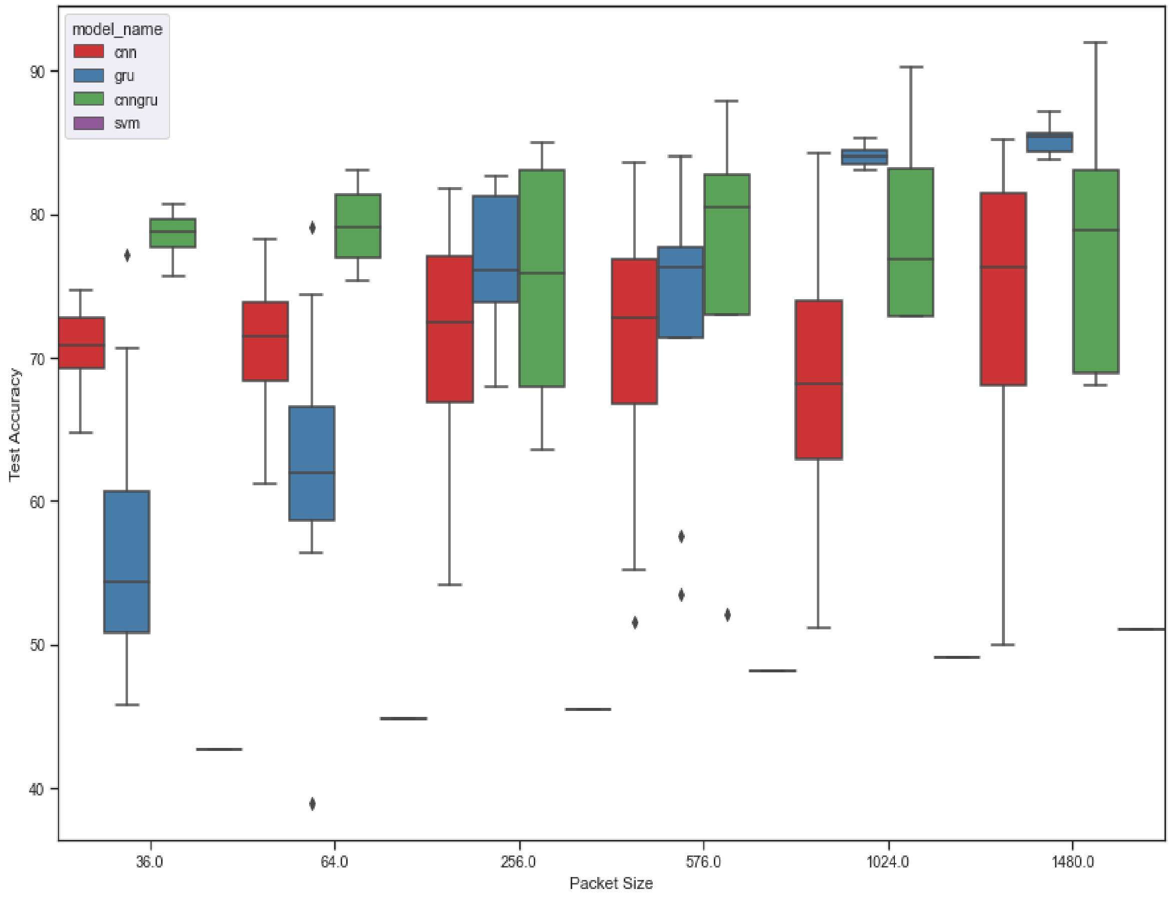


FIGURE 4.7: Accuracy Distribution by each Model against Packet size.

4.3 Experiment 3: Packet Prediction speed (in packets per second) against packet sizes.

Investigate the effect of the deep learning model's performance in terms of prediction rate (packets predicted per second (pps)) and complexity for classifying network traffic? To predict the performance of the models, we used the concept of *packets per second* as described in section 3.12 of Chapter 3. We measured the complexity of the models using the trainable parameters of each model. In Figure 4.8, the general trend that can be observed is that prediction speed of the models decreases as the packet sizes increases for the 3 deep learning models. Determining the number of packets predicted is affected by the number of parameters each model requires and the size of the input. As the model's input shape changes, the number of parameters that make up a model increases contributing to the complexity of the model. The more complex a model becomes, the more time it takes to predict. In addition to packet size, parameters such as batch size are correlated with the performance of deep learning models. For instance, when predicting classes, applying larger values for batch sizes translates to larger weight matrices associated with algorithm matrix multiplications. This matches our expectations. Moreover, for the CNN and CNNGRU, filter sizes plays a role in increasing parameters and results in more convolution operations. Though the models compete in their accuracy prediction. The results of their prediction speeds were marginally different. This could be attributed to the difference in the model architecture. CNN had the greatest number of packets per second prediction with results ranging from 25 000 to the least one being 10 000. GRU followed with packets predicted ranging from 12 000 to 2100. CNNGRU had the least packets predicted per second ranging from 4 500 to 3 000.

4.3.1 Per Class Relationship between Prediction Rate, Trainable Parameters and Test Accuracy

In Figure 4.9 - 4.11 we further analysed the relationship of the Prediction speed, trainable parameters and Test Accuracy for each class. Information based on each class may be useful for ensuring QoS for a low-resource environment. Figure 4.9 shows an inverse relationship of the prediction speed represented by packets per second against the packet size. However, there is no much variation across all the 13 classes. The prediction speed drops significantly after the 200 packet size.

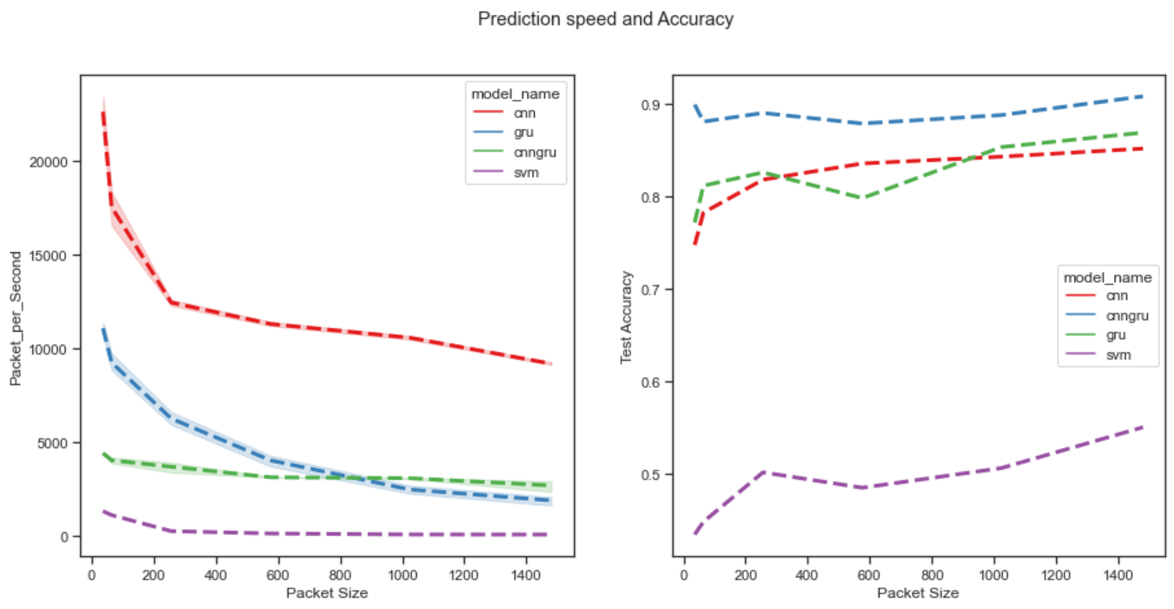


FIGURE 4.8: Relationship displaying Packets Size for each model against Prediction speed and Test Accuracy.

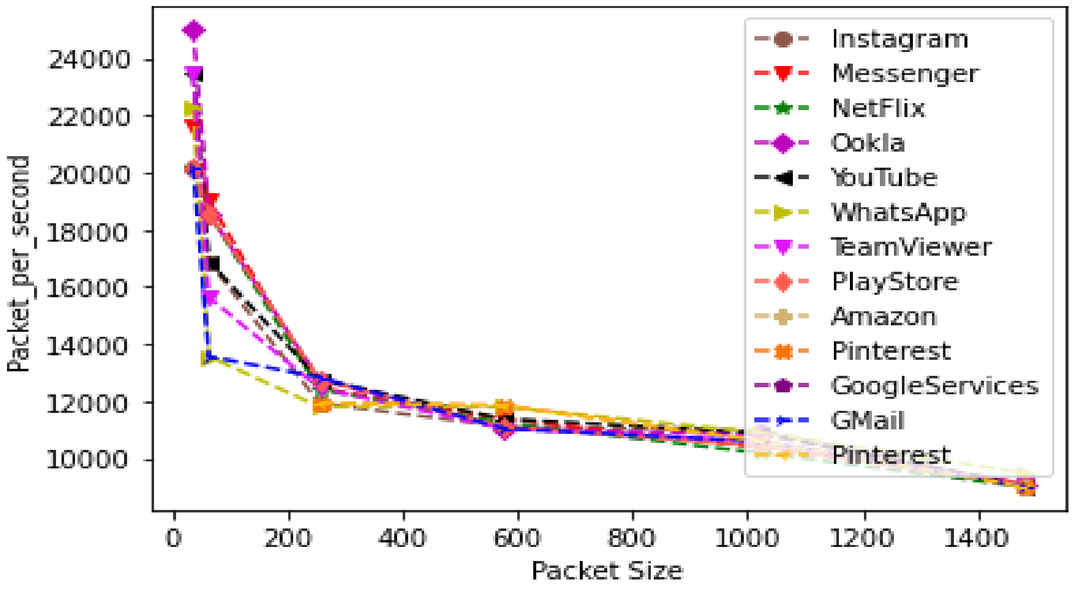


FIGURE 4.9: Relationship displaying Packets per Second for each class against packet sizes with Test Accuracy of 70 %.

Figure 4.10 also shows an exponential relationship of the packet size and the trainable parameters. As we train across a higher number of packets, parameters increase, this shows the relationship between accuracy being a function of increasing parameters. However, it not simple to distinguish classes in this scenario, as they follows a similar trend.

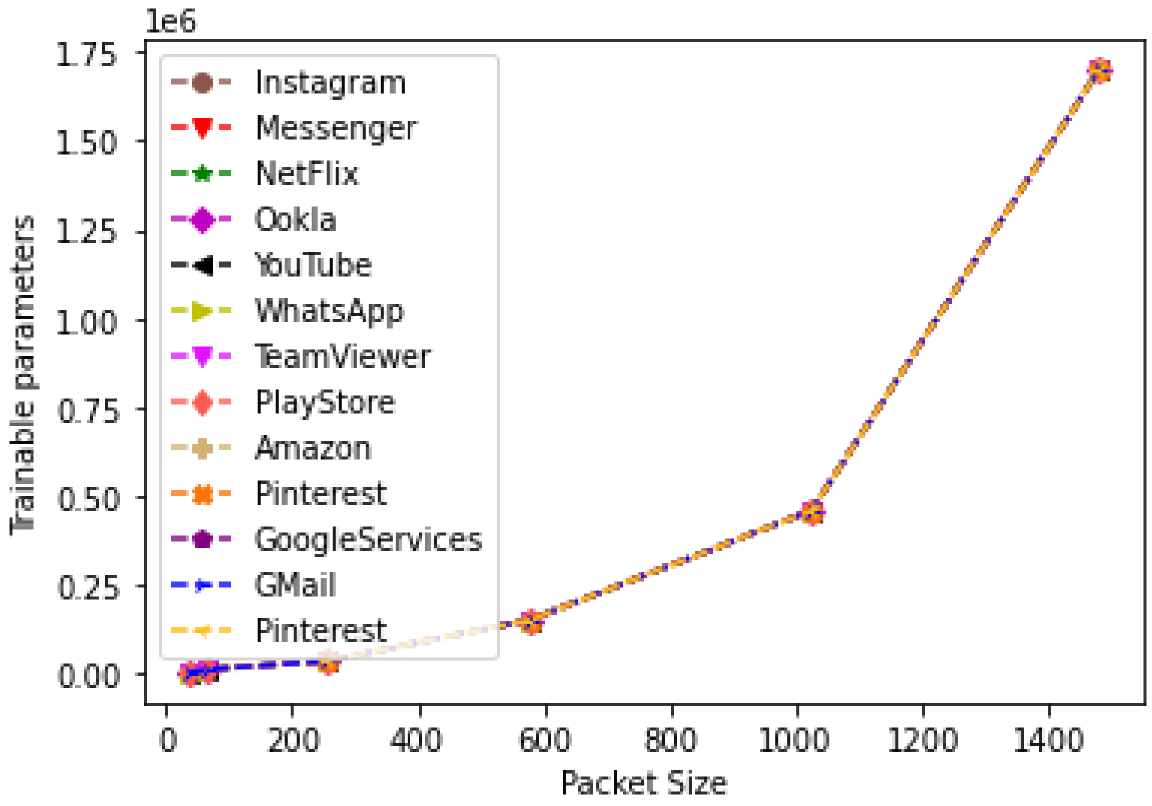


FIGURE 4.10: Relationship displaying Trainable Parameters for each class against packet sizes.

4.4 Summary of Results

The information in this section provides a detailed account of all the experiments carried out for this study. This information helped us to better understand our research goals. The results from section 4.1 can be used to make an inference as to which of the three DL models performs best with respect to the network traffic dataset collected from the community network. The results in section 4.2 can be used to make deductions about the influence of packet size on the performance of an ML model. The observations in section 4.3 can be used to infer the effect that the packet size has on prediction speed, and the effect of packet size on class identification. The results of this study can help you choose the best model type and packet size for accuracy.

Chapter 5

Discussion

This chapter provides a thorough discussion of the performance of each approach (SVM, CNN, GRU and CNNGRU) in relation to the experiments in Chapter 4. We present a detailed analysis of how each method performs in different scenarios elaborated in Chapter 4, section 4.1 to 4.3. Furthermore, we correlate this result with previous studies that used Deep Learning (DL) and conventional ML techniques to classify network traffic, and show their impact on the research goals in section 1.3.

5.1 Classification Performance of SVM against Deep Learning Models

The findings presented in section 4.1 of Chapter 4 reveal that all three DL models (CNN, GRU, and CNNGRU) are suitable for classifying community network data compared to the baseline ML model (SVM). Therefore, this validates the assumption we had in the first chapter in section 1.3, for research objective 1. DL models are more suitable and effective for classifying network traffic data [14], [10]. The test accuracy represents each model's out-of-sample performance and, as a result, its ability to generalise to new data. With SVM achieving an accuracy of only 51%, and the DL models achieving accuracy's of over 80%, with the hybrid model showing significant improvement and attaining the highest accuracy of 91%. We can conclude that hybrid models are ideal and suitable for multi-classification. This matches a previous study that hybrid models generalise well compared to single models [13], [15]. The differences in accuracy results could be attributed to the unique architecture designs. For instance, CNNs are useful in extracting spatial features, on the other hand, GRUs have a lot of gated units and are efficient in learning long sequences through storing crucial information throughout the training phase. Therefore, a hybrid architecture that combines the two results in the best accuracy performance compared to single model architectures in all settings. Additionally, the difference in results between the single CNN and GRU model architectures could be that GRUs perform better due to the nature of network traffic dataset. Network packet bytes are sequential in nature, meaning packets are arranged in consecutive order. This implies that GRUs are more suitable to capture the long-range information than CNNs. Transforming network data from the original 1D to 2D format could potentially contribute to loss in spatial meaning of the network packets. A possible explanation for poor performance of the SVM could be because of the small dimensional space of possible parameters compared to the DL models.

5.1.1 Evaluation of the Confusion Matrix Outcome

To further address our first objective stated above (also in Chapter 1, section 1.3), our Experiment explored confusion matrix results for each model to determine which of the proposed DL techniques performed best given the network traffic dataset. The confusion matrix results displayed in (section 4.1.1, Figure 4.2 -4.5) were a result of highest recorded results after a grid-search was conducted accuracy score for each model done on a test set of 26 000 instances. Using a confusion matrix gave us an in-depth summary of the prediction results for the model performance. We could further infer the precision, recall and F1 Score. Using F1 Score is the logical choice in this classification scenario because it gave an equal weighting of both precision and accuracy. The F1 scores of the SVM were much lower than the scores for the DL models. We observed that application classes such as BitTorrent, Teamviewer, YouTube, NetFlix and WhatsApp had consistently significantly high F1 scores. This could mean that there is more useful information for the classes that have high scores.

5.2 Classification Performance of Deep Learning Models by varying byte data input shapes

To further investigate the performance of the different models, we modified the input byte data and trained the models on the byte data stated in Chapter 4. This was due to the fact, different classes or applications have different byte sequences. In order to assess which input size would be appropriate for each class it was important to train on different input sizes. The results as explained and shown in Chapter 4 (Section 4.2). The trend on the accuracy is such that as we move towards the higher input packet sizes the accuracy value gradually increases. The accuracy increase is attributed to an increase in parameters of the Deep Learning models. For instance, parameters such as batch size, filter size and layers contribute to an increase in parameters thus contributing to model complexity. The highest accuracy was achieved with the entire packet size of 1480 across all models, showing that if accuracy is the main issue, the whole payload should be used. However, the results in the section 5.3 demonstrated that taking into account fewer packet sizes might result in faster prediction speed, suggesting that adopting a small packet size could be useful in a community network context. The best accuracy possible for models with input sizes ranging from 256 to 576 bytes does not appear to be significantly different, implying that bytes 256 to 1024 do not add substantial information. The results were of varying values particularly for the ensemble model. However, across the other 3 models, CNN, GRU and CNNGRU, the results were consistent across all input shapes. We infer that it maybe unnecessary to fit a full payload of 1480 packets. Although the full payload attained the highest accuracy because that is where the useful features that distinguish classes are positioned. We can infer that there is an inverse relationship between packet size and accuracy. A full payload guarantees a high accuracy score, at the cost of low prediction speed and compute resource. It is possible to design a robust classifier that classifies network traffic data based on the first few packets.

5.3 Analysis of Prediction Performance of the Deep Learning Models

Experiment 3 from Chapter 4, section 4.3 was carried out to assess the prediction speed of each model, taking into account that considering classification accuracy alone was not sufficient to select the ideal model but the speed at which the model could predict the packets using the test set was critical to be considered for the community network. Carrying out this experiment thoroughly answered the Third Research Question from Chapter 1, section 1.3. The results shown in Figure 4.8 - 4.11, show a great disparity in prediction speed across the four models. For instance, CNN has more packets/bytes predicted per second in comparison to GRU, and CNNGRU. Although it is evident that models with the full packet sizes have smaller packets per second values, though when evaluating the accuracy, packets-size input, does not appear to contribute a substantial difference in the accuracy achieved. The higher packets per seconds scores achieved by the 36-byte models could be attributed to the few computations needed because the penultimate and last packets in the packet stream distinguishes one class for the other. Therefore, models fully learn when fed with full packet sizes. Deducing results from Figure 4.8, it is evident that hybrid models are more complex in comparison to single DL models. However, there is a trade-off between accuracy and prediction speed. There is a trade-off between prediction speed and accuracy. Choosing a solution that is suitable for a low-resource environment is dependent on what matters, i.e. accuracy or prediction speed. If accuracy alone were of importance, then choosing a more complex model such as a CNNGRU is great fit, which means less complex models such as the CNN and GRU are unsuitable in such a scenario. However, if prediction speed alone is of importance, a less complex model such as a CNN is suitable, with the knowledge that the community network has a bandwidth of 10Mbps which can be translated as the measure of network traffic flow per second. Therefore, our gateway can process a threshold of 10 0000 packets per second. Therefore, we can infer from the results in Section 4.3 that implementing a CNN-classifier for a 10Mbps link could be ideal, considering that CNN model had the fastest prediction speed.

5.3.1 Evaluating Class Performance

By analysing further the prediction speed and trainable parameters with respect to each class, we could infer and distinguish application classes. The results displayed were for classes over a threshold accuracy of 70%. This analysis was pertinent in evaluating prediction speed and accuracy information distinct to each class. This information could present itself is useful for resource planning and QoS provisioning. For example, we can traffic engineer application classes across the community network based on the prediction speed requirements, although, results for each application class was not clearly distinguishable.

5.4 Conclusion

In this study, our choice to differentiate these Deep Learning models was to assess the suitability of the models on our dataset and environment. Therefore, we considered Convolutional Neural Networks, Gated Recurrent Unites and CNNGRU as a hybrid. In Chapter 3, we explained in detail the architectural implementations of the models. In all scenarios, to clearly determine the fitting model, we used the same number of three fully connected layers across all models. In reference to section 2.8.2 of our Literature, CNNs have been demonstrated to be outstanding contenders for the real-time packet-based traffic classification task. Traffic classification using CNNs therefore has the potential to provide meaningful benefits to a community network setting and make significant contributions to bridging the digital divide through its application in QoS provisioning.

Chapter 6

Conclusion and Recommendations

In this study we outlined the effectiveness of ML vs single DL models vs a Hybrid DL model in classifying community network traffic. Inspired by this, we transformed raw packet data into an nxn tensor grey image. Because CNNs are popularly used in image recognition, we transformed the raw packet data into a nxn tensor grey image. In the case of the GRU and CNNGRU, we used the same concept as CNN, but the difference was that the raw packet data was a 1-D nxn tensor whereas with the CNN, it was a 2D-tensor. The GRU model requires sequential data which the raw packet features in our data resembled, hence we did not see the need to transform the input. SVMs, on the other hand were used because they are a conventional ML method. We adopted them in the study because of how they are regarded for their ability to perform multi-classification. This motivated the direction of our research to evaluate the effectiveness of these methods. Despite, the capabilities of DL models to achieve high classification accuracy scores, accuracy alone was not sufficient to form a basis for choosing the ideal model in this study's context. We used different evaluation criteria to assess the chosen DL models. Different important parameters were taken into consideration in order to evaluate the suitable ML/DL model for our environment. Our choice to evaluate different models with different architectures was intended to ascertain the most suitable model for classification. Performance metrics such as prediction time and prediction speed in the form of packets predicted per second (pps), derived by calculating the time a model took to predict packets in a given sample. Using this metric was a measure of determining the speed at which a model processes packets.

Considering real-time classification is important to address QoS through network traffic prioritisation. In addition, we used different packet feature sizes in order to assess the model's ability to correctly classify application classes when packet features were varied.

For instance, training a model on a small number of packet sizes would likely guarantee a good accuracy score. In addition, a model's complexity, is critical given that implementing a real-time classifier in a low-resourced network such as a community network could affect speed and performance. A model's trainable parameters were a determining factor used to determine the complexity. Even though, DL models achieve high precision, recall, F1-scores and accuracy scores they are comparatively more complex because of the matrix calculations through the neural connections characterised by weights and biases. For that sake, we adopted a 3-layer architecture across all the DL models, in order to have a fair basis of comparison. The complexity and performance of a DL model is dependent on different hyper-parameter settings and values, therefore, instead of using default hyper-parameter settings and values, a grid search was implemented. When all four models are compared to one another, using a single hyper-parameter method may decrease performance estimation bias. The motivation to use our very own dataset was inspired by applying a solution that is unique to a low-resource environment such as a community network.

Our study's motivation followed a packet-based classification approach by Lim et. al [15], as we tested the effectiveness of machine learning vs Deep Learning models. Secondly, we implemented a comparative analysis of single DL models versus a DL hybrid model approach for a community network dataset. We adopted 3-layer networks because we endeavour to implement a light-weight solution that could necessitate a model that struck a compromise. For instance, classification speed and timeliness on the one hand needs to suit real-time classification, and computational efficiency in terms of memory resource utilisation on the other, all while fulfilling acceptable accuracy performance standards. Thus our approach to apply deep learning techniques to accomplish classification was pivotal, to attain high accuracy scores.

6.1 Limitations and Challenges

We evaluated the performance of the DL model using a supervised learning approach. The process required a data generation process demanding, which was a large part of the problem. Several studies have demonstrated the suitability of deep learning techniques as a viable solution in classifying encrypted traffic data in comparison to classical/traditional approaches [8], [9], [84]. However, the challenge still lies in how computationally expensive it is to implement Deep Learning. Therefore, careful consideration has to be done in order to evaluate the applicability of the correct model in a live environment, the cost associated with maintaining such an implementation and evaluating how adding more application classes affects performance and eventually, how it can be used in an actual community network traffic classifier.

Deep Learning demonstrates to be a practical and beneficial approach to solving traffic classification studies. Numerous ground-breaking algorithms have been presented recently as a result of the field's steady growth in research. However, by conducting a detailed literature review of prior work, we evaluated that there exists different trends that put into question the true rate of progress in the field of network traffic classification. Trends such as the lack of one standard dataset to test the different hypotheses can be a major drawback to the progress of a true evaluation procedure that is applicable in any setting. Furthermore, the failure to adequately evaluate and compare novel classification methods is another main issue since a particular problem is solved by a different classification mechanisms.

6.2 Future Work

For Future work, the hardware aspect may need to be further researched in detail to suit the community network environment. Therefore, this would mean deploying a classifier on the community network gateway in a simulated live environment in order to assess how often the classifier would classify new classes or applications: in addition, to assess the feasibility of this work and how often the classifier requires retraining. A couple of network operators and service providers in South Africa other than community networks may be of benefit from such studies.

Exploring more application classes and testing this concept with different algorithms and simulating the same set of experiments to gain a broad perspective of model performance, prediction speed and accuracy.

A standardised performance evaluation for network traffic classifiers is of relevance such that, if widely adopted, would greatly improve the validity and credibility of future research, make replication and reproducibility easier, as well as improve the ability of the field to accurately gauge the rate of progress over time by being able to make sound comparisons across different works.

The findings of this study are promising, and there are a seemingly unlimited number of opportunities for exploration in derivative works.

Appendix A

An Appendix

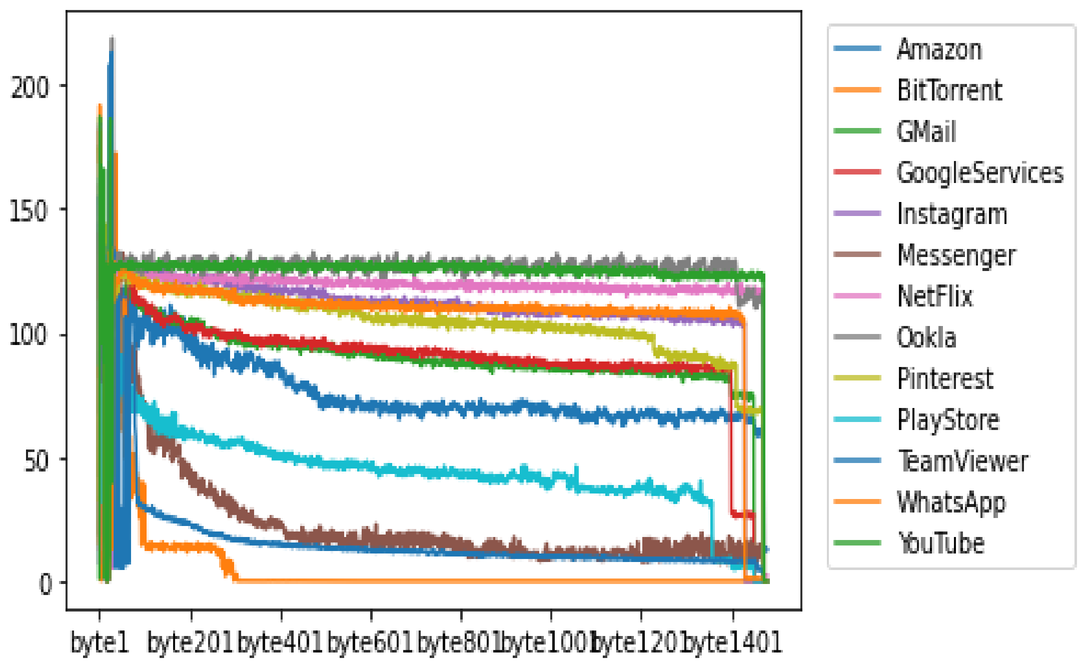


FIGURE A.1: Class Distribution.

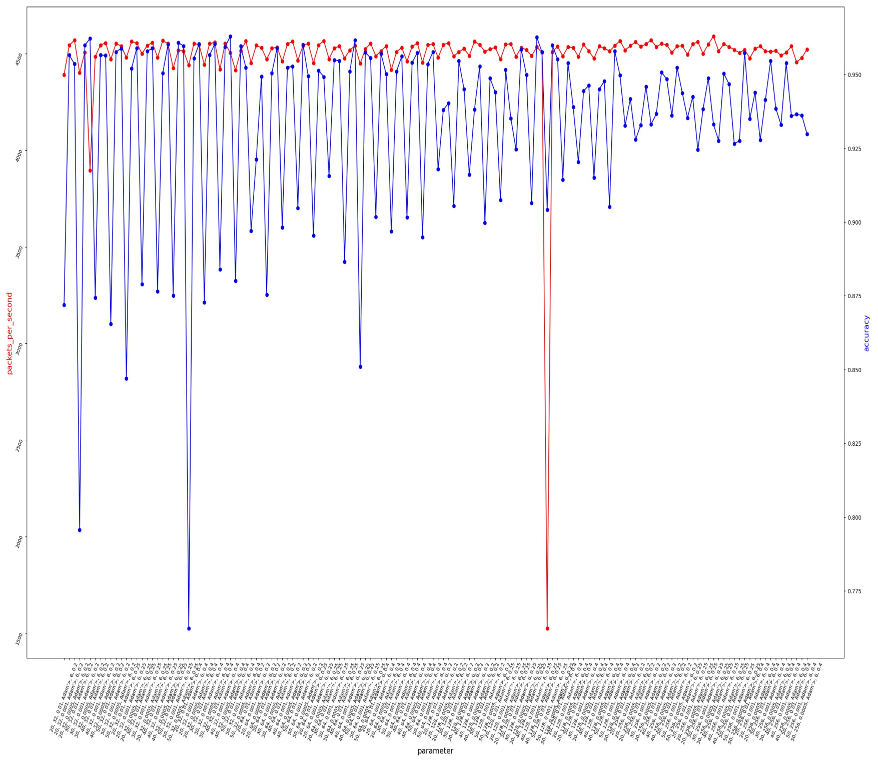


FIGURE A.2: Summary of Gridsearch of CNNGRU Algorithm.

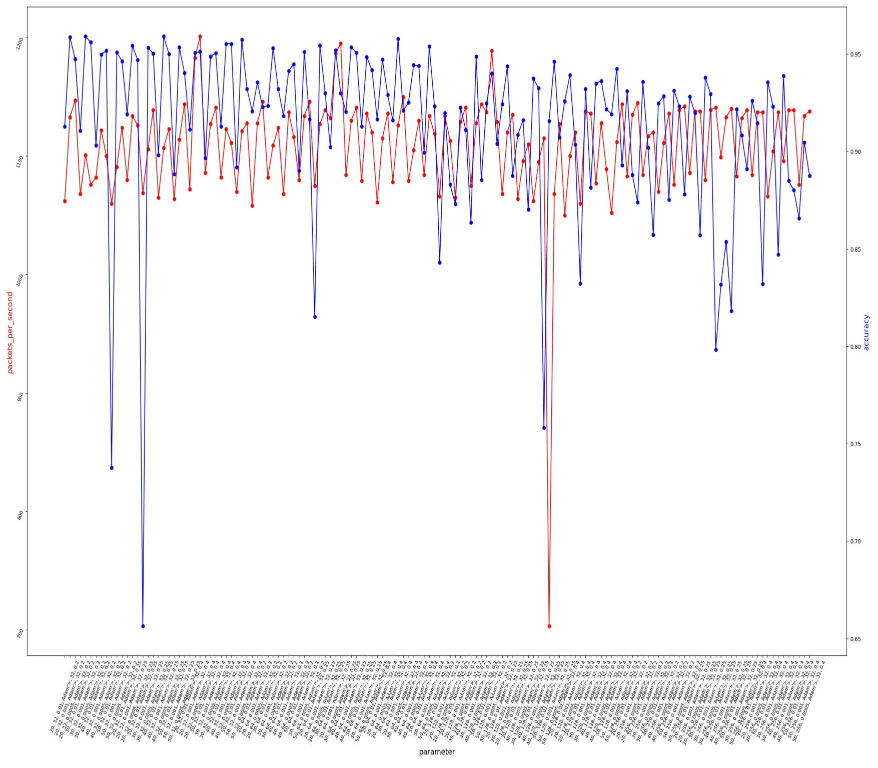


FIGURE A.3: Summary of Gridsearch for GRU Algorithm.

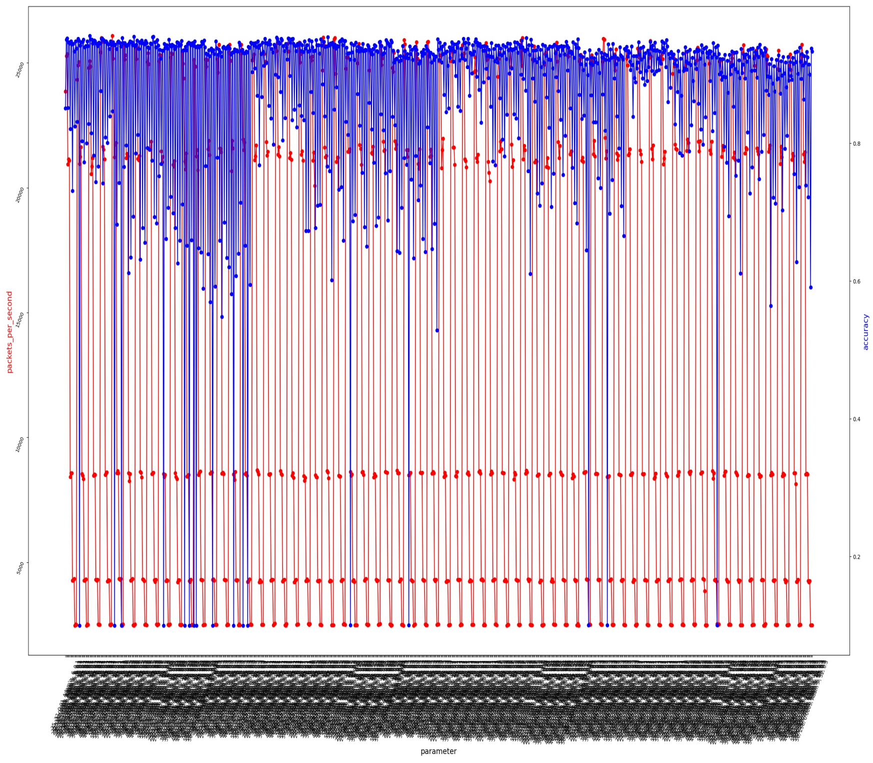


FIGURE A.4: Summary of Gridsearch for CNN Algorithm.

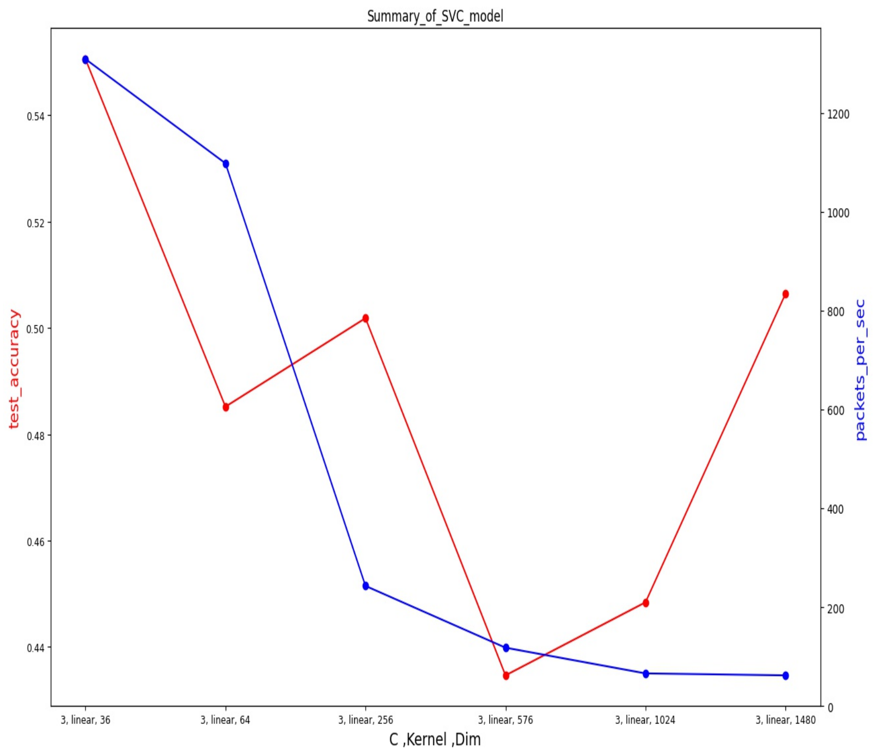


FIGURE A.5: Summary of Gridsearch for SVM Algorithm.

Bibliography

[1] Larry L Peterson and Bruce S Davie. *Computer networks: a systems approach.* Elsevier, 2007.

[2] Russell Stuart, Norvig Peter, et al. Artificial intelligence: a modern approach, 2003.

[3] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems.* " O'Reilly Media, Inc.", 2019.

[4] Chris Albon. *Machine learning with python cookbook: Practical solutions from preprocessing to deep learning.* " O'Reilly Media, Inc.", 2018.

[5] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and KC Claffy. Transport layer identification of p2p traffic. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134, 2004.

[6] Panagiota Micholia, Merkouris Karaliopoulos, Iordanis Koutsopoulos, Leandro Navarro, Roger Baig Vias, Dimitris Boucas, Maria Michalis, and Panayotis Antoniadis. Community networks and sustainability: a survey of perceptions, practices, and proposed solutions. *IEEE Communications Surveys & Tutorials*, 20(4):3581–3606, 2018.

[7] Bart Braem, Chris Blondia, Christoph Barz, Henning Rogge, Felix Freitag, Leandro Navarro, Joseph Bonicioli, Stavros Papathanasiou, Pau Escrich, Roger Baig Vinas, et al. A case for research with and on community networks, 2013.

[8] Jonathan Tooke and Josiah Chavula. Resource-constrained real-time network traffic classification using one-dimensional convolutional neural networks. 2021.

[9] Matthew Dicks and Josiah Chavula. Deep learning traffic classification in resource-constrained community networks. In *2021 IEEE AFRICON*, pages 1–7. IEEE, 2021.

[10] Adebayo Oluwaseun Adedayo and Bhekisipho Twala. Qos functionality in software defined network. In *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 693–699. IEEE, 2017.

[11] Jinghua Yan and Jing Yuan. A survey of traffic classification in software defined networks. In *2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN)*, pages 200–206. IEEE, 2018.

[12] Chang Liu, Longtao He, Gang Xiong, Zigang Cao, and Zhen Li. Fs-net: A flow sequence network for encrypted traffic classification. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 1171–1179. IEEE, 2019.

[13] Farnaz Sarhangian, Rasha Kashef, and Muhammad Jaseemuddin. Efficient traffic classification using hybrid deep learning. In *2021 IEEE International Systems Conference (SysCon)*, pages 1–8. IEEE, 2021.

[14] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. Mobile encrypted traffic classification using deep learning. In *2018 Network Traffic Measurement and Analysis Conference (TMA)*, pages 1–8. IEEE, 2018.

[15] Hyun-Kyo Lim, Ju-Bong Kim, Joo-Seong Heo, Kwihoon Kim, Yong-Geun Hong, and Youn-Hee Han. Packet-based network traffic classification using deep learning. In *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 046–051. IEEE, 2019.

[16] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng. Malware traffic classification using convolutional neural network for representation learning. In *2017 International Conference on Information Networking (ICOIN)*, pages 712–717. IEEE, 2017.

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

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

[19] Mohammad Mehedi Hassan, Abdu Gumaei, Ahmed Alsanad, Majed Alrubaian, and Giancarlo Fortino. A hybrid deep learning model for efficient intrusion detection in big data environment. *Information Sciences*, 513:386–396, 2020.

[20] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. Characterization of encrypted and vpn traffic using time-related. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, pages 407–414, 2016.

[21] Zhanyi Wang. The applications of deep learning on traffic identification. *BlackHat USA*, 24(11):1–10, 2015.

[22] Pu Wang, Shih-Chun Lin, and Min Luo. A framework for qos-aware traffic classification using semi-supervised machine learning in sdns. In *2016 IEEE international conference on services computing (SCC)*, pages 760–765. IEEE, 2016.

[23] Haiyong Xie, Y Richard Yang, Arvind Krishnamurthy, Yanbin Grace Liu, and Abraham Silberschatz. P4p: Provider portal for applications. *ACM SIGCOMM Computer Communication Review*, 38(4):351–362, 2008.

[24] IANA. Service name and transport protocol port number registry. URL `https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml`.

[25] Walter De Donato, Antonio Pescapé, and Alberto Dainotti. Traffic identification engine: an open platform for traffic classification. *IEEE Network*, 28(2):56–64, 2014.

[26] Alberto Dainotti, Antonio Pescape, and Kimberly C Claffy. Issues and future directions in traffic classification. *IEEE network*, 26(1):35–40, 2012.

[27] Michael Finsterbusch, Chris Richter, Eduardo Rocha, Jean-Alexander Muller, and Klaus Hanssgen. A survey of payload-based traffic classification approaches. *IEEE Communications Surveys & Tutorials*, 16(2):1135–1156, 2013.

[28] Andrew W Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. In *International workshop on passive and active network measurement*, pages 41–54. Springer, 2005.

[29] Muhammad Afaq, Shafqat Ur Rehman, and Wang-Cheol Song. A framework for classification and visualization of elephant flows in sdn-based networks. *Procedia Computer Science*, 65:672–681, 2015.

[30] José Suárez-Varela and Pere Barlet-Ros. Sbar: Sdn flow-based monitoring and application recognition. In *Proceedings of the Symposium on SDN Research*, pages 1–2, 2018.

[31] Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *Proceedings of the 13th international conference on World Wide Web*, pages 512–521, 2004.

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

[33] Douglas C Sicker, Paul Ohm, and Dirk Grunwald. Legal issues surrounding monitoring during network research. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 141–148, 2007.

[34] Thuy TT Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE communications surveys & tutorials*, 10 (4):56–76, 2008.

[35] nDPI. Open and extensible lgplv3 deep packet inspection library. URL https://www.ntop.org/products/deep-packet-inspection/ndpi/.

[36] PAESSLER. Prtg manual: Packet sniffer sensor. URL https://www.paessler.com/manuals/prtg/packet_sniffer_header_sensor.

[37] Shahbaz Rezaei and Xin Liu. How to achieve high classification accuracy with just a few labels: A semi-supervised approach using sampled packets. 2019.

[38] Niccolo Cascarano, Alice Este, Francesco Gringoli, Fulvio Risso, and Luca Salgarelli. An experimental evaluation of the computational cost of a dpi traffic classifier. In *GLOBECOM 2009-2009 IEEE Global Telecommunications Conference*, pages 1–8. IEEE, 2009.

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

[40] Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. Network traffic classifier with convolutional and recurrent neural networks for internet of things. *IEEE Access*, 5:18042–18050, 2017.

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

[42] Petr Velan, Milan Čermák, Pavel Čeleda, and Martin Drašar. A survey of methods for encrypted traffic classification and analysis. *International Journal of Network Management*, 25(5):355–374, 2015.

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

[44] Pankaj Gupta and Nick McKeown. Algorithms for packet classification. *IEEE Network*, 15(2):24–32, 2001.

[45] Carey Williamson. Internet traffic measurement. *IEEE internet computing*, 5(6): 70–74, 2001.

[46] Zhitang Chen, Ke He, Jian Li, and Yanhui 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)*, pages 1271–1276. IEEE, 2017.

[47] Andrew Moore, Denis Zuev, and Michael Crogan. Discriminators for use in flow-based classification. Technical report, 2013.

[48] Tom M Mitchell. Does machine learning really work? *AI magazine*, 18(3):11–11, 1997.

[49] Zhong Fan and Ran Liu. Investigation of machine learning based network traffic classification. In *2017 International Symposium on Wireless Communication Systems (ISWCS)*, pages 1–6. IEEE, 2017.

[50] Jie Cao, Zhiyi Fang, Guannan Qu, Hongyu Sun, and Dan Zhang. An accurate traffic classification model based on support vector machines. *International Journal of Network Management*, 27(1):e1962, 2017.

[51] Bingdong Li, Jeff Springer, George Bebis, and Mehmet Hadi Gunes. A survey of network flow applications. *Journal of Network and Computer Applications*, 36(2): 567–581, 2013.

[52] Tom M Mitchell et al. Machine learning. 1997.

[53] Riyad Alshammari and A Nur Zincir-Heywood. Can encrypted traffic be identified without port numbers, ip addresses and payload inspection? *Computer networks*, 55(6):1326–1350, 2011.

[54] Hassan Ramchoun, Mohammed Amine Janati Idrissi, Youssef Ghanou, and Mohamed Ettaouil. Multilayer perceptron: Architecture optimization and training. *IJIMAI*, 4(1):26–30, 2016.

[55] Wenxiao Jia, Yi Wan, Yanpu Li, Kewei Tan, Wenqing Lei, Yiying Hu, Zhao Ma, Xiang Li, and Guotong Xie. Integrating multiple data sources and learning models to predict infectious diseases in china. *AMIA Summits on Translational Science Proceedings*, 2019:680, 2019.

[56] Jun Zhang, Xiao Chen, Yang Xiang, Wanlei Zhou, and Jie Wu. Robust network traffic classification. *IEEE/ACM transactions on networking*, 23(4):1257–1270, 2014.

[57] Valentín Carela-Español, Pere Barlet-Ros, Albert Bifet, and Kensuke Fukuda. A streaming flow-based technique for traffic classification applied to 12+ 1 years of internet traffic. *Telecommunication Systems*, 63(2):191–204, 2016.

[58] Guanglu Sun, Teng Chen, Yangyang Su, and Chenglong Li. Internet traffic classification based on incremental support vector machines. *Mobile Networks and Applications*, 23(4):789–796, 2018.

[59] Alice Este, Francesco Gringoli, and Luca Salgarelli. Support vector machines for tcp traffic classification. *Computer Networks*, 53(14):2476–2490, 2009.

[60] Jeffrey Erman, Anirban Mahanti, Martin Arlitt, Ira Cohen, and Carey Williamson. Semi-supervised network traffic classification. In *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 369–370, 2007.

[61] Béla Hullár, Sándor Laki, and Andras Gyorgy. Early identification of peer-to-peer traffic. In *2011 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2011.

[62] Laurent Bernaille and Renata Teixeira. Early recognition of encrypted applications. In *International Conference on Passive and Active Network Measurement*, pages 165–175. Springer, 2007.

[63] Thuy TT Nguyen, Grenville Armitage, Philip Branch, and Sebastian Zander. Timely and continuous machine-learning-based classification for interactive ip traffic. *IEEE/ACM Transactions On Networking*, 20(6):1880–1894, 2012.

[64] Thuy TT Nguyen and Grenville Armitage. Training on multiple sub-flows to optimise the use of machine learning classifiers in real-world ip networks. In *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, pages 369–376. IEEE, 2006.

[65] Ly Vu, Cong Thanh Bui, and Quang Uy Nguyen. A deep learning based method for handling imbalanced problem in network traffic classification. In *Proceedings of the Eighth International Symposium on Information and Communication Technology*, pages 333–339, 2017.

[66] Gabriel Paulino Siqueira Junior, Jose Everardo Bessa Maia, Raimir Holanda, and Jose Neuman de Sousa. P2p traffic identification using cluster analysis. In *2007 First international global information infrastructure symposium*, pages 128–133. IEEE, 2007.

[67] Ly Vu, Dong Van Tra, and Quang Uy Nguyen. Learning from imbalanced data for encrypted traffic identification problem. In *Proceedings of the Seventh Symposium on Information and Communication Technology*, pages 147–152, 2016.

[68] Yann LeCun, Lawrence D Jackel, Corinna Cortes, et al. Learning algorithms for classification: A comparison on handwritten digit recognition.

[69] Tae-Young Kim and Sung-Bae Cho. Web traffic anomaly detection using c-lstm neural networks. *Expert Systems with Applications*, 106:66–76, 2018.

[70] Jonas Höchst, Lars Baumgärtner, Matthias Hollick, and Bernd Freisleben. Unsupervised traffic flow classification using a neural autoencoder. In *2017 IEEE 42Nd Conference on local computer networks (LCN)*, pages 523–526. IEEE, 2017.

[71] Princeton University Stanford Vision Lab, Stanford University. Imagenet. URL https://www.image-net.org.

[72] Canadian Institute for Advanced Research. The cifar-10 dataset. URL https://www.cs.toronto.edu/~kriz/cifar.html.

[73] Raouf Boutaba, Mohammad A Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano, and Oscar M Caicedo. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):1–99, 2018.

[74] CompariTech. Pcap: Packet capture, what it is what you need to know. URL https://www.comparitech.com/net-admin/pcap-guide/.

[75] Philippe Biondi and the Scapy community. Scapy: Packet crafting for python2 and python3. URL https://scapy.net//.

[76] Adam Langley and Wan-Teh Chang. Quic crypto (2013). *URL https://www. chromium. org/quic*, 2014.

[77] Google Cloud. Encryption in transit in google cloud. URL https://cloud.google.com/security/encryption-in-transit#:~:text=We%20ensure%20the%20integrity%20and,encryption%20at%20the%20network%20layer.

[78] Randall Stewart and Scott Long. Improving high-bandwidth tls in the freebsd kernel. *FreeBSD Journal*, pages 8–13, 2016.

[79] TeamViewer Trust Center. Teamviewer trust center. URL https://www.teamviewer.com/en/trust-center/security/#:~:text=TeamViewer%20traffic%20is%20secured%20using,completely%20safe%20by%20today's%20standards..

[80] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[81] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[82] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[83] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[84] Shane Weisz and Josiah Chavula. Community network traffic classification using two-dimensional convolutional neural networks. 2021.