Encrypted Web Traffic Classification Using Deep Learning

by

Iman Akbari

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2021

Author's Declaration

This thesis consists of material all of which I have authored or co-authored. See Statement of Contributions included in the thesis.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

Chapters 1 to 5 borrow content and figures from the paper: "A Look Behind the Curtain: Traffic Classification in an Increasingly Encrypted Web" [7].

Abstract

Traffic classification is essential in network management for operations ranging from capacity planning, performance monitoring, volumetry, and resource provisioning, to anomaly detection and security. Recently, it has become increasingly challenging with the widespread adoption of encryption in the Internet, e.g., as a de-facto in HTTP/2 and QUIC protocols. In the current state of encrypted traffic classification using Deep Learning (DL), we identify fundamental issues in the way it is typically approached. For instance, although complex DL models with millions of parameters are being used, these models implement a relatively simple logic based on certain header fields of the TLS handshake, limiting model robustness to future versions of encrypted protocols. Furthermore, encrypted traffic is often treated as any other raw input for DL, while crucial domain-specific considerations exist that are commonly ignored. In this thesis, we design a novel feature engineering approach that generalizes well for encrypted web protocols, and develop a neural network architecture based on Stacked Long Short-Term Memory (LSTM) layers and Convolutional Neural Networks (CNN) that works very well with our feature design. We evaluate our approach on a real-world traffic dataset from a major ISP and Mobile Network Operator. We achieve an accuracy of 95% in service-level classification with less raw traffic and smaller number of parameters, out-performing a state-of-the-art method by nearly 50% fewer false classifications. We show that our DL model generalizes for different classification objectives and encrypted web protocols. We also evaluate our approach on a public QUIC dataset with finer and application-level granularity in labeling, achieving an overall accuracy of 99%.

Acknowledgements

I would like to thank my supervisor Professor Raouf Boutaba for his unceasing help, support, and kindness that enabled me in this project and our other research projects.

I would also like to thank Professor Mohammad Ali Salahuddin for his continuous help in my thesis and publications, Professor Noura Limam for her suggestions and help, our colleagues in Orange Labs for providing the datasets used in this thesis.

My most sincere and heartfelt thanks to my parents and sister for their love and support that enabled me to push further and achieve more in my life and during my education in Waterloo.

Dedication

To the 176 innocent souls who lost their lives in the attack on flight PS752.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Traffic classification (TC) is quintessential for network operators to perform a wide range of network operation and management activities. This includes capacity planning, security and intrusion detection, quality of service (QoS), performance monitoring, volumetry, and resource provisioning, to name a few. For example, an enterprise network administrator or Internet service provider (ISP) may want to prioritize traffic for business critical services, identify unknown traffic for anomaly detection, or perform workload characterization for designing efficient resource management schemes to satisfy performance and resource requirements of diverse applications. Depending on the context, misclassification on a large scale may result in failure to deliver QoS guarantees, incur operational expenses, security breaches or even disruption in services.

Formally, traffic classification can be defined as the categorization of network traffic units (e.g., packets, flows, bi-flows, sessions, etc.) according to an arbitrary set of parameters or into a set of pre-defined classes [17]. In the past, traffic classification was mostly performed based on TCP/UDP *port numbers*. However, with the growing complexity of network protocol stacks, the widespread use of web applications, and more sophisticated use-cases like P2P networks, port number-based classification quickly became insufficient. *Deep Packet Inspection* (DPI) has also been used as an alternative approach for traffic classification. Nonetheless, encryption has introduced severe obstacles for classification via DPI as well.

To preserve the privacy of Internet end-users, encrypted communication between clients and servers is becoming the norm. Most prominent web-based services are now running over Hypertext Transfer Protocol Secure (HTTPS). Furthermore, to improve security and quality of experience (QoE) for end-users, new protocols (e.g., HTTP/2 [14] and QUIC [36])

have emerged, which overcome various limitations of HTTP/1.1. These new protocols are already in large-scale use by many Internet services and supported by all major clients. For example, based on an analysis of our real-world mobile traffic dataset (cf. Section 4.2.1), around 32% of all HTTPS sessions with identifiable HTTP version use HTTP/2 as their underlying protocol. This suggests a high adoption rate of HTTP/2 in the Internet. However, HTTP/2 features, such as payload encryption, multiplexing and concurrency, resource prioritization, and server push, add to the complexity of traffic classification. Moreover, in HTTP/2 and QUIC services, the use of encryption is a de-facto standard.

Machine Learning (ML) and Deep Learning (DL) have undoubtedly become evermore influential with their application extending over a wide range of domains, including traffic classification. Over the past two decades, a large body of literature has surfaced to harness the power of ML for different traffic classification objectives, such as *service-level* classification (i.e., classification into coarse-grained service categories, e.g., video streaming, chat and webmail), *application-level* classification (i.e., fine-grained classes, e.g., YouTube, Gmail and WhatsApp), QoS and QoE prediction (e.g., video quality inference), and malicious traffic detection (e.g., intrusion detection systems).

1.1 Motivation

Despite the growing academic and industrial interest in ML-based traffic classification, there are still various functional limitations that need to be addressed for its real-world, practical usage.

For instance, numerous works (e.g., [70, 44, 31]) in traffic classification pick their labels somehow arbitrarily, which are often inconsistent in granularity. For example, authors in [44] use labels such as Network Time Protocol (NTP), a protocol, and YouTube, an application, at the same time. This allows for gerrymandering the dataset to report higher accuracies, and at the same time it is not conducive to real-world traffic analysis use-cases. Some other works (e.g., [74, 48, 45, 68]) use datasets with a mixed set of protocols that are often easily distinguishable using header signatures. Realistically, there is no practical interest in DL-based traffic classification, when the classification can be achieved with high accuracy via deterministic non-ML approaches (e.g., header matching). We address this issue by using a dataset of real-world traffic (cf. Section 4.2.1) with consistent labeling at the service and application levels. We have made the dataset publicly available.

Some of the works in traffic classification, on the other hand, leverage clever techniques to guide the models based on expert domain-specific knowledge. For example, authors in

[19] use the position of non-MTU-sized packets to finger-print content inside an HTTP/2 session. While these solutions might show improved performance in the short-term, small variations in the protocol can jeopardize their entire approach. Thus, future research should move towards generic data-driven methods that can adapt to different protocols and different traffic classification objectives, by simply changing the training data. We discuss this issue further in Section 3.1.

More importantly, extensions such as the Server Name Indication (SNI) in Transport Layer Security (TLS) can essentially reveal the server's identity, allowing for trivial classification of many traffic flows based on the server name. It has been showcased [48] that current DL models trained for traffic classification rely heavily on these extensions, and the model performance degrades if they are removed. This has severe implications for the state-of-the-art methods. For one, expensive and complex models are being used to learn a relatively simple logic, similar to that of a server name to label look-up table, that can be implemented deterministically. We argue that the true power of DL is exploited when the models actually learn about the *nature* of traffic in different classes, and identify complex patterns to distinguish between them. We discuss this issue in more details in Section 2.5.2, and address it by occluding these extensions from the training data and eliminate the model's reliance on them, while leveraging a carefully crafted feature engineering approach (cf. Section 3.1 & Section 3.5).

Effective and efficient application of Machine Learning to traffic classification relies on considering the significance of encryption and adapting our methods to encrypted traffic, especially encrypted *web* traffic due to its ubiquity. The issues raised here compel us to study the existing approaches in the context of encrypted web protocols and to design traffic classification solutions that can adapt to their evolution.

1.2 Contributions

In this thesis, we leverage DL for service classification (e.g., video streaming, social media, web mail) with a focus on new encrypted web protocols, i.e., HTTP/2 and QUIC, and overcome the above limitations. Unlike many works in this area, we focus exclusively on encrypted web traffic, and explore the challenges of unleashing the full potential of DL to find complex patterns in traffic that are innate to each traffic class. We occlude parts of the input that may allow the DL model to learn a lazy and unsophisticated logic, and instigate how encrypted traffic should be treated differently from general raw ML input, e.g., images. We also place emphasis on a well-generalizable feature set that can be utilized by a diverse variety of future works in encrypted web traffic classification.

Our main contributions are as follows:

- We propose a novel feature engineering approach for encrypted traffic classification that focuses on protocol-agnostic aspects of the encrypted web traffic. We leverage standard flow statistics, the traffic shape with respect to packet size, inter-arrival time (IAT), and direction, along with raw bytes from the TLS handshake packets. This is in contrast to most DL approaches for traffic classification, where the full raw traffic is fed to the DL model. We justify why the proposed feature set is a better fit for classification of encrypted traffic.

- We develop a neural network architecture based on Convolutional Neural Network (CNN) and Stacked Long Short-Term Memory (LSTM) layers that is highly effective in making use of the extracted features for distinguishing between different traffic classes. We exploit the full potential of DL in identifying and correlating useful traffic traits, while being lighter in the number of trainable parameters and less likely to overfit.

- We use a real-world mobile traffic dataset from an ISP, and demonstrate that our approach has an edge over the state-of-the-art in service classification of encrypted web traffic. We achieve an average accuracy of over 95% for classification exclusively over HTTPS (i.e., HTTP/1.1 and HTTP/2 over TLS), outperforming [48] by a significant margin of $\sim$50% fewer false classifications. We have made the corresponding pre-processed dataset available to the public.

- We showcase that our DL model generalizes for a finer classification granularity, i.e., application classification. We also show that our model adapts to a different encrypted web protocol, i.e., QUIC, by simply changing the training data. We achieve an accuracy of 97% in application-level classification and an accuracy of 99% on a public QUIC dataset [49].

1.3 Thesis Organization

The rest of the thesis is organized as follows: In Chapter 2, we provide a brief background on the HTTP/2 and QUIC protocols, and the challenges they bring to traffic classification, as well as typical feature categories used by traffic classification approaches. We then review the literature that inspired our work. We delineate our methodology, feature design and neural network architecture in Chapter 3. In Chapter 3, we also discuss data processing and

our DL model training. We present the evaluation of our approach in Chapter 4, providing insights into what affects its performance and its advantage over existing methods. We conclude in Chapter 5 with a brief summary of our work and instigate future research directions.

Chapter 2

Background

Traffic classification has attracted significant attention in the past two decades. In this section, we start by briefly introducing the HTTP/2 and QUIC protocols and the challenges they bring to traffic classification. We provide some context on Deep Learning algorithms in general with a focus on the ones used in our work and the literature inspiring our work, and shed light on how different approaches in traffic classification compare. We explain the broad categories of solutions for traffic classification, followed by a review of the related works that have inspired this thesis.

2.1 New Web Protocols: An Overview

2.1.1 HTTP/2

The HTTP protocol has become a fundamental component in most web-based services. Based on Google's SPDY protocol [13], HTTP/2 is the successor of HTTP/1.1 and was released in May 2015 (cf. RFC 7540 [12]). It has since been widely adopted by the biggest providers on the Internet.

In HTTP/2, the smallest unit of communication is a frame, which consists of a header and a sequence of bytes depending on the frame type. A sequence of frames in an HTTP/2 connection is called a stream. A connection may contain concurrently open streams, and these streams may be established or terminated by either endpoint. The connection initiator assigns the stream an unsigned 31-bit integer in order to identify them, which is usually an even number for connections initiated by the server and odd for those initiated

by clients. The identifier is 0 if it is a connection control message, and 1 if it is an HTTP/1 connection upgraded to HTTP/2.

This allows for header compression, as well as prioritization of objects by the web-browser, to avoid the head-of-line (HOL) blocking. It offers performance improvements and addresses various limitations in HTTP/1.1 with features including synchronous communication (i.e., pipelining), multiplexing of streams, server push, and binary message framing. Furthermore, HTTP/2 uses encryption as a de-facto standard.

By providing the features described above, HTTP/2 achieves its main design goals, which is to offer reduced latency, better bandwidth, and improved QoE, while maintaining continuity with previous versions and allowing for easy integration into existing clients and servers. However, HTTP/2's multiplexing, compression, and encryption, introduce new challenges for traffic classification [18].

2.1.2 QUIC

QUIC is a transport-layer protocol, recently proposed by Google. It is already in use by most Google services, supported by most modern web-browsers, and submitted to IETF [35] for standardization. QUIC provides an alternative to TCP and brings the key exchange and encryption to the transport layer. By providing multiplexing and pushing congestion control to the user space, QUIC provides enhanced performance compared to HTTP over TCP and works hand-in-hand with HTTP/2 to address HOL blocking.[1] The significance of QUIC for traffic classification is two-fold: First, it reinforces the fact that future web communications will use more encryption, i.e., a larger proportion of each session's information will be encrypted. Second, similar to HTTP/2, enhancements such as multiplexing add complexity to traffic classification.

2.2 Deep Learning Architectures

Deep Learning [41] has become an unprecedented phenomenon in computer science over the past decade. By showing great promise in areas such of computer vision [63], speech recognition [25], natural language processing [72], and sequential decision-making [46], it has gained recognition throughout the entire scientific community and its application have

[1] HTTP-over-QUIC is often referred to as HTTP/3, as the protocol is aware of the semantics used in HTTP/2 and binds them to the wire protocol. This name is used to underline the integration and separate it from the general QUIC protocol.

expanded over a vast range of areas in the scientific literature. As a subset of machine learning, deep learning is generally based on artificial neural networks (ANN) that mimic the inner workings of the human brain using multiple layers of linear and non-linear transformations. Broadly speaking, each layer in a deep neural network is capable of processing the input information at a more abstract level than the last. The parameters of these transformations are set based on a labeled or unlabeled set of data-points during a process called *training*. During the neural network model's training, each data-point (or batch of data-points to be more accurate) contributes to an update to the parameters of the entire neural network, which is typically calculated based on the value of a global *objective function* via an *optimization algorithm*. Most of the prominent optimization algorithms are in one way or another related to gradient descent [51]. Hardware accelerations and software optimizations for these algorithms over the past few years have led to revolutionary advancements in machine learning.

There are many types and families of neural networks that have been applied for different use-cases. Although a comprehensive study of their varieties, use-cases and structures is outside the scope of this thesis, we hereby go through some of the most basic types of neural networks that are widely used in traffic classification and ML in general.

2.2.1 Convolutional Neural Networks

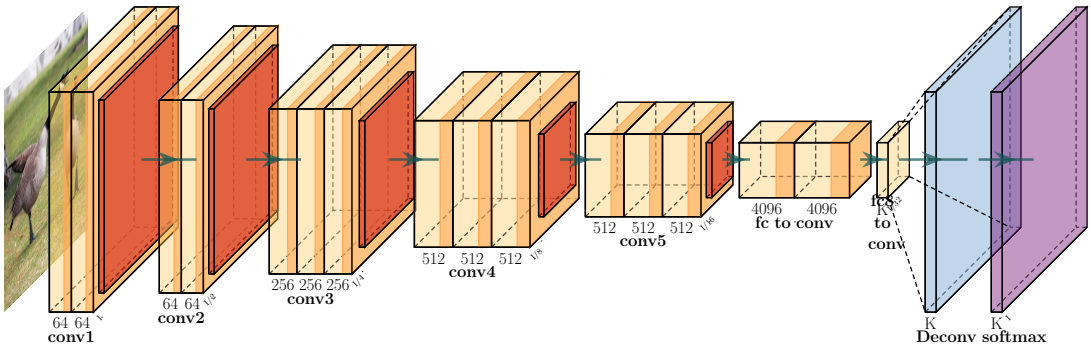


Figure 2.1: A deep convolutional neural network is capable of detecting more complex patterns at each subsequent layer, up to the final layer which decides the output of the network.

Convolutional Neural Network (CNN) is a class of neural network models with the

assumption of spatial invariance built into its architecture. Compared to a dense neural network, the invariance assumption massively cuts down the number of parameters in the model. A convolutional layer has a set of *filters*, which are convolved against the inputs to produce the outputs. Each filter has very few parameters compared to the number of inputs and outputs. Commonly, convolutional layers are interleaved with pooling layers, which reduces the dimension of the output by down-sampling.

Intuitively, each CNN layer is capable of detecting a number of patterns that might occur at any location in a multi-dimensional input. The deeper a layer is, the more abstract the patterns detected by it can get. This is why, for instance, CNNs are widely used in computer vision use-cases where the initial layers can detect more rudimentary structures and arrangements such as curves, lines, shapes occuring at arbitrary locations in the image. Going deeper into the network, the internal layers can gradually learn more intricate patterns such as particular symbols or objects. Figure 2.1 depicts a deep neural network which can process an input image.

2.2.2 Long Short-term Memory

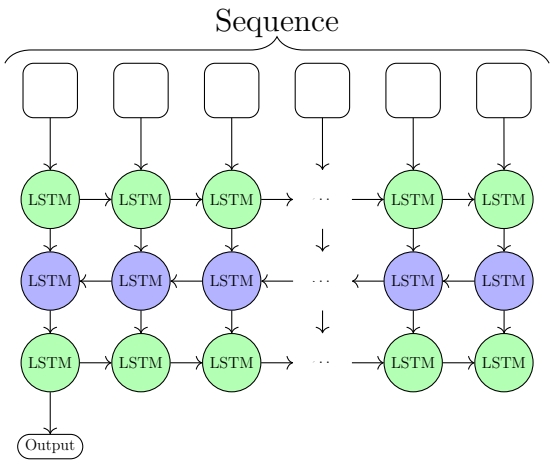


Figure 2.2: Part of our proposed DL model uses three stacked LSTMs to process the flow information

A Long Short-term Memory (LSTM) unit is a recurrent neural network, which is capable of remembering features from earlier positions in a sequence. LSTM is a common element in neural networks for natural language processing (NLP) and sequence processing in general.

Multiple LSTM units can be stacked together to improve the information flow between positions in the input sequence. An example of three stacked LSTMs is shown in Figure 2.2.

An LSTM unit has an internal state that can be altered based on the input sequence it has seen at each time-step. The output of the unit is a function of both its internal state and the input, making the neural network model extremely powerful in processing sequential data.

2.2.3 Autoencoders and Generative Models

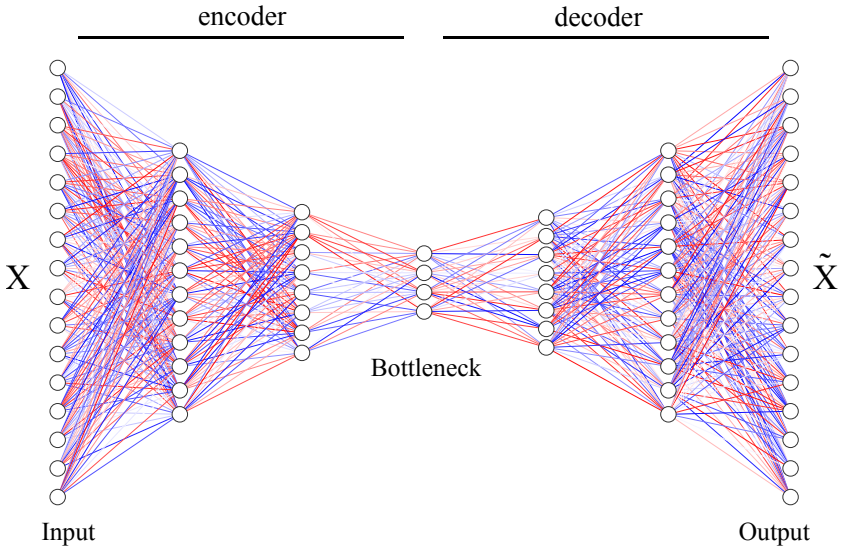


Figure 2.3: An autoencoder architecture capable of extracting a 4-dimensional encoding out of a 16-dimensional input by trying to reproduce its input (X) as accurately as possible in the output ($\tilde{X}$).

Autoencoders are a family of unsupervised neural network models which are capable of learning a *representation* of the input variables. They are used for dimensionality reduction, de-noising, capturing more abstract semantics from the input, compression, disentangling features, etc. The core idea in autoencoders is to recreate the input as accurately as possible at the model's output layer. If the size of the model's middle layers gradually decrease up to a certain point, this creates a *bottleneck* in the smallest internal layer, forcing the model to learn an efficient *encoding* of the input in a smaller *latent space*. After the training, the model can be split into two halves around the bottleneck, where the first half (closer to input) can be thought of an "encoder" and the second half (closer to output) as a "decoder"

(cf. Figure 2.3). Stacked Autoencoders (SAE) [62] are a subset of autoencoders that are created from multiple layers of sparse autoencoders where the output of each hidden layer is fed to the input of the subsequent hidden layer.

A Generative Adversarial Network (GAN) [30] is another unsupervised artificial neural network model which can be used to generate data-points *similar* to those of a given unlabeled dataset. GANs are based on two competing neural network models: a *generator* which receives some randomness (noise) and outputs a data-point, and a *detector* which tries to distinguish the data-points synthesized by the generator from those of the actual dataset. By iteratively training the two models against each other, we end up with a generator model whose outputs are difficult to distinguish from the input dataset. GANs are widely used in *data augmentation*.

2.3 Data Collection

In the application of ML to network management, the scarcity of labeled data is a well-known problem. While the research in security management is suffering the most from this obstacle, service and application classification is no exception. Figure 2.4 depicts our categorization of TC approaches. When it comes to data collection, there are generally two ways to procure training and validation datasets:

- **Synthetic:** Typically, synthetic data is generated using automated agents. The primary benefit is the ease of labeling, as we always know the nature of traffic at its source. Furthermore, the labeling can be done at a very fine granularity, as the cryptographic keys and the traffic payloads are available. For instance, the objective of a classification task can pertain to the actual words a user searches on a website or application (e.g., [38]). Publicly available and labeled datasets are difficult to find for such specific use-cases, and the use of synthetic data is a far more viable option. However, the evident disadvantages of using synthetic data include its failure to capture the true distribution of traffic in real-world networks, which affect the model's generalization and performance in practice.
- **Real-world:** This pertains to data recorded from real networks with real users. There are many considerations and obstacles in using real-world network traces, including labeling, and privacy and security concerns. The latter is crucial, especially when the data or even the model is to be partially or fully released. Recently, a hot topic in ML and cybersecurity communities is the emergence of highly effective ML privacy attacks, such as model inversion attacks [28]. It has been shown that neural networks are far

more likely to leak information about their training data, than one might first imagine. Therefore, even if only the model is to be released to external entities, there are still privacy and security risks involved. On the other hand, labeling real-world data is often an obstacle, as real-world users are unlikely to announce the purpose of their traffic, unless instructed to do so. Furthermore, in some applications of ML to traffic classification, such as intrusion detection, the labeling may even defeat its very purpose. This is primarily because, if there exists a deterministic method to flag attacks in real-world data, the ML model would likely not be needed.

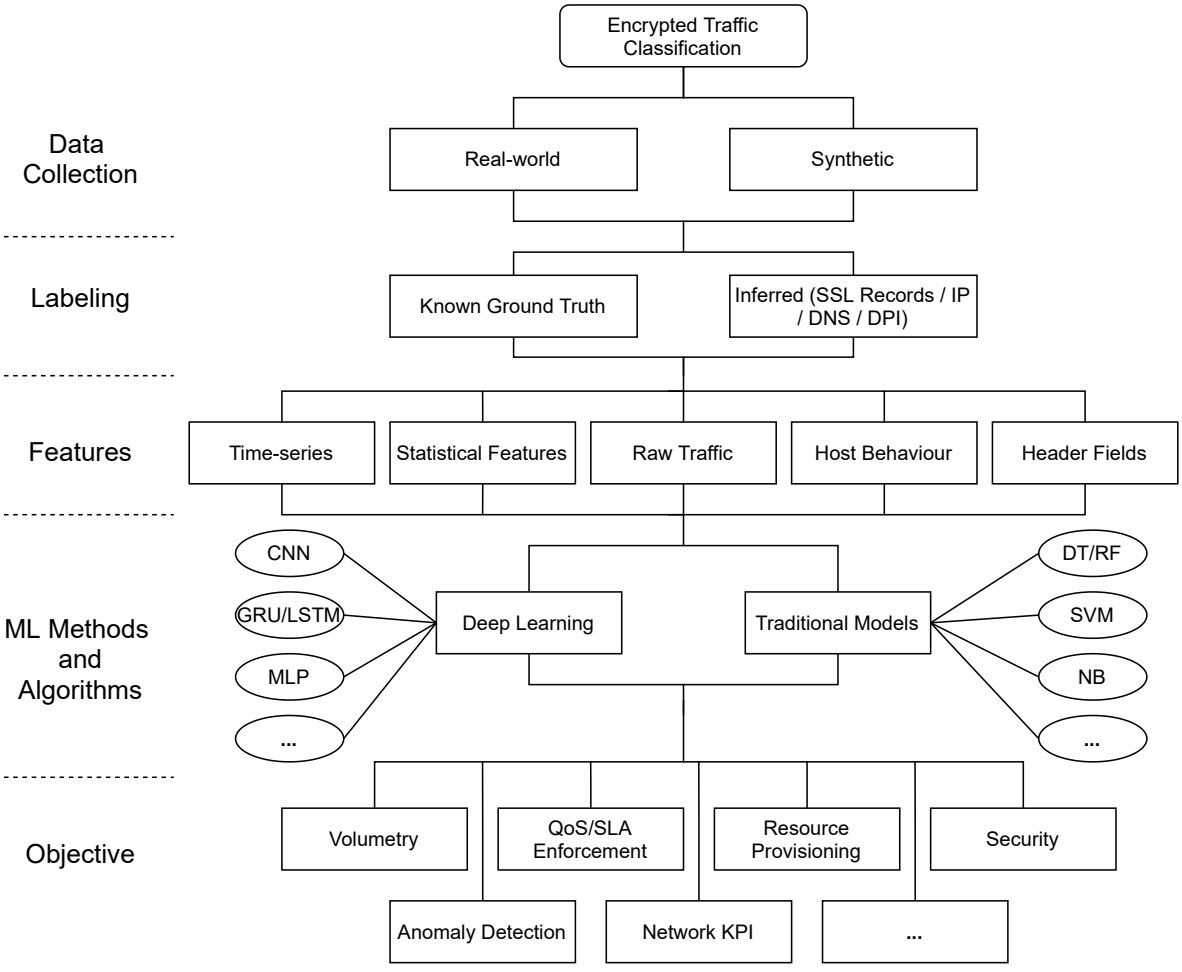


Figure 2.4: Our categorization of traffic classification works based on their data, labeling, features, ML algorithms, and objectives.

Labeling synthetic datasets is generally trivial, as the nature of each connection is usually known when it is being generated. For real-world datasets, however, the labels are either:

- **Known:** When the traffic is captured or generated precisely for the purpose of creating a training dataset, the data is labeled at its origin. This includes the scenario where users are hired to use a network and corresponding traces are recorded, and when the labels are known via other means, e.g., recording email traffic at the mail server's ports.
- **Inferred:** Generally, unlabeled data is much easier to come by for all ML tasks, including TC. It is rather trivial to capture a large corpora of network traces and logs in large-scale networks. As also explored in this work, in limited cases, the data can be automatically labeled with a certain degree of accuracy. However, it is crucial to evaluate if a definitive automated labeling exists. Understandably, this would usually defeat the purpose of developing an ML-based classifier, unless the classifier is either computationally cheaper (which is seldom the case with large capable models), or when the learned information is extrapolated, i.e., the data is partially labeled, and the learned patterns and features can be generalized to the entire data.

In some cases, no labeling is performed and the model training is unsupervised. However, this is generally irrelevant to TC, unless when performing anomaly detection, e.g., in IDS. In addition, semi-supervised approaches are also viable, where a small labeled dataset is used alongside a large unlabeled one, to train a more capable model than the one trained only on the small dataset. For example, clustering data points followed by labeling based on a few known entries. A more elaborate example is when good representations of the traffic packets or flows are learned using Autoencoders [66], or modeling the traffic as sentences of a language and learning a word embedding [55] in NLP.

Currently, one of the most popular public labeled dataset for service classification is UNB's ISCX VPN-nonVPN 2016 [26]. Ariel University has also released a dataset of mobile application traffic that is labeled by the browser, website, OS, and QoE [47], and a similar one for videos [27]. Rezaei et al. [49] have released a small dataset of QUIC traffic from both human users and synthetic generators. It is worth noting that numerous prominent works in TC make use of private datasets, which is an obstacle for confidently reproducing their results. We hope that our dataset (cf. Section 4.2.1) can also help future research in this area.

ML algorithms for TC can be informally categorized into traditional (e.g., Random Forest (RF), C4.5 Decision Tree, Naive Bayes (NB), Logistic Regression (LR), and Support Vector Machines (SVM)) or deep learning (DL) (e.g., Convolutional Neural Networks

(CNN), Stacked Auto-encoders (SAE), Long Short-term Memory (LSTM), Gated Recurrent Units (GRU), and Multi-layer Perceptron (MLP)) methods. One might expect DL to replace traditional algorithms in TC, as it has done for computer vision, speech processing, NLP and many other areas. However, some fast traditional algorithms, such as RF, are notorious for achieving very high accuracy in TC [37]. Furthermore, due to hardware limitations, these algorithms are more likely to be put into production for *real-time* use-cases. The recent surge in DL and its prospects has nonetheless sparked a lot of effort in applying similar techniques to network traffic. A large proportion of these DL techniques are borrowed from other more well-known areas, such as computer vision, but there seems to be a gap in applying domain-specific knowledge and the system's point-of-view to the task. We will further discuss these issues in Section 2.5.

In terms of the objective, our work in this thesis is mostly concerned with service- and application-level classification, despite having some implications for the other categories as well. The features used for the model's input are discussed in Sections 2.4 and 3.1. However, it is important to note that in the TC literature most related to our work, the host behaviour features (cf. Figure 2.4) are often irrelevant because: (i) they require a high degree of visibility into the network, e.g., complete visibility over all hosts, including the external ones, throughout a long timespan, and (ii) for connection-level or flow-level classification, they are not the most becoming as they are fit for classifying at a level where hosts remain generally consistent; A single host may be running many different applications and services, thus features describing the overall host behaviour (e.g., number of connections, number of peers, average time between connections, average activity time, etc. similar to what used by [53] or graph features in [3]) are not very relevant in this context. This is in contrast to use-cases like cybersecurity where classification at host level (e.g., benign or malicious) is often desired. The header fields features (e.g., fields and records of headers in protocols like TCP, HTTP, TLS, etc.) were quite popular in early works in TC. However, in encrypted traffic many header fields are not available in plain-text. Moreover, as raw traffic features can capture entire headers and also statistical features can to some extent reflect things like no. TCP flags in a flow, "header fields" as a separate category of features has started to lose interest. This is why these two feature categories are not reflected in our categorization in Section 2.4 and moving forward.

2.4 Related Works

We can broadly categorize the typical features employed in the relevant traffic classification literature for modeling traffic into the following groups:

1. **Flow Statistics:** A standard flow-meter, such as CICFlowMeter [40], yields the mean, standard deviation, minimum, maximum, of packet lengths, inter-arrival times, TCP flag counts, flow durations, number of packets, number of bytes, etc. These statistics constitute a feature vector for each flow and have been employed since the early traffic classification literature.

2. **Raw Bytes:** The actual flow bytes from packet headers and payloads, which have grown in popularity with the advent of DL. Their appeal is leveraging data in the rawest form, as done in more conventional applications of DL such as computer vision.

3. **Time-series:** Following a fixed-size, packet-level feature through all packets in a flow can yield a dynamic-sized time-series feature representing the flow. For example, the sizes of the packets in a flow is a valid time-series feature.

The effort in traffic classification using ML started in the early 2000s with the application of ML models in order to distinguish between different protocols (e.g., DNS, SMTP, HTTP, etc.) or different categories of services and use-cases (e.g., database, mail, P2P, multimedia, etc.). Most of the earlier works in the area use feature type 1 with lightweight ML models,such as Naive Bayes (NB), Bayesian Networks (BN), and Decision Trees (e.g., C4.5) as well as ensemble learning methods (e.g., RF, XGBoost [23], etc.). Williams et al. [69] mark one of the first works in this area by comparing the effectiveness of C4.5, BN, NB, and Naive Bayes Tree (NBT) for classifying FTP, SMTP, DNS, HTTP, and the multiplayer game Half-life, all of which were readily detectable based on port numbers. In their comparison, they showed that the C4.5 decision tree achieved the best results between the examined models, with above 90% accuracy. While these results seem promising, one must note that the objective of the model was something that could be done deterministically via signature matching.

Soon, the attention in academia shifted towards more challenging TC tasks, such as classification of encrypted traffic e.g., Skype. The industrial and academic interest in this subject was justified due to the fact that unlike the former efforts, there are often no trivial non-ML solutions for traffic classification in their context, without DPI. For instance, applications like Skype do not necessarily operate on a fixed port number. Alshammari et al. [8] tackled this problem using a similar approach with statistical features (i.e., feature type 1) and more elaborate algorithms such as SVM, C4.5, and AdaBoost. In their evaluations, they trained binary classifiers for SSH and Skype traffic using four different model types and were able to achieve less than 3% false-positive rate and above 97% detection rate using C4.5. While showcasing the effectiveness of ML for classification of encrypted traffic is very interesting, it is important to note that the domain of their work still remains

detecting a particular protocol, as opposed to the service type or use-case, which is often more interesting for *volumetry*[2].

Schatzmann et al. [53] took on a more challenging variety of TC, which is the identification of a certain use-case, namely webmail, over an encrypted protocol (HTTPS). However, their focus is mostly on classifying *endpoints* as opposed to flows or packets themselves. They make use of features pertaining to host behaviour (e.g., periodicity of connections) and network view (e.g., proximity to POP/IMAP servers) to distinguish HTTPS webmail servers. As discussed earlier in Section 2.3, although this is a fairly creative approach, it does require a holistic view of the network including both the sources and destinations, which is often not available. Also, it is hard to generalize these strategies to all other services since they are very specific to mail servers.

Deep learning, created new opportunities in TC by allowing incorporation of fine-grained features as opposed to using statistics and aggregations alone (feature type 1). Lotfollahi et al. [45] for instance, proposed using raw packet bytes (with certain adjustments in pre-processing) as the feature vector to convolutional neural networks (CNN), and stacked autoencoders (SAE). On the ISCX'16 VPN-nonVPN dataset, which is a corpus of mixed-protocol traffic, they were able to achieve 97% accuracy using these kinds of neural networks. The resulting model should be thought of as an automated fingerprinting approach, as it tries to classify single packets and not the flows. However, we have found that when it comes to classifying different services on top of HTTPS for example, its performance completely deteriorates. The capabilities of Multilayer Perceptron (MLP), Stacked Autoencoders (SAE), Convolutional Neural Networks (CNN), and Long Short-term Memory (LSTM) for TC have been explored extensively in the literature [74, 66, 67, 68, 24, 50, 32]. Aceto et al. [4, 44] perform an extensive evaluation of a number of these methods for classification of different mobile apps' traffic on their own proprietary dataset. In their assessment, they argue that there is no silver bullet when it comes to the choice of a neural network model for TC, however, 1D-CNN and LSTM networks seem to be a good fit for the job due to the sequential nature of network traffic.

While the ISCX dataset has been widely used in the traffic classification literature, due to its mixed-protocol nature, it is a much more straightforward classification task than classifying homogeneous fully encrypted traffic, as the model can learn an easy header-matching logic. For instance, the traffic in the mail class includes SMTP and POP3 traffic, while the file transfer class includes SFTP and FTPS traffic, which are easy to distinguish based on their headers. Thus, even models receiving packets individually and outside the context of the flow (e.g., [45]) can have good accuracies. In contrast, in TLS traffic

[2]We define *volumetry* as measuring the volume of different categories of traffic in a given network trace.

classification, the performance of such models degrades, as there is simply not enough information in a single encrypted packet alone.

On the other hand, it is clear that many deep learning TC solutions in the literature are inspired by standard neural network methods used in computer vision. For instance, as reflected in [45] and [5], CNNs have shown great promise in classifying network traffic. This is probably due to the fact that they can detect patterns (e.g., certain values for TLS records) that might occur at different offsets in the packet.

Chen et al. [24] for instance, have taken this analogy between traffic and images a step further in their method called Seq2Img. The authors use two-dimensional CNNs, which are commonly used for image processing, in order to perform traffic classification on a dataset involving 5 applications: Instagram, Skype, Facebook, Wechat and YouTube. The raw flow information in their work includes the sequence of packet sizes, inter-arrival times and directions, while *Reproducing Kernel Hilbert Space (RKHS)* is leveraged to create a 2-D 6-channel "image" of the raw training entries. As for the number of packets extracted from each flow, the authors have chosen to use the first 10 packets of the flow (skipping the first 3 packets) via empirical experiments, nevertheless they have indicated that the accuracy of the model can be increased by using more packets. The 2-D feature tensor is later fed into a simple CNN with two convolutional layers, two max pooling layers and three dense layers with categorical cross-entropy as the loss function, which is a conventional set-up. The results have been compared against previous literature making use of SVM, MLP, NB, and DT which has outperformed the best previous results with an accuracy of 99.84%, i.e., a 3% lead against SVM. As also mentioned in Section 1, we argue that the equivalence between raw traffic and raw images is incorrect, or rather incomplete when it comes to encrypted traffic. We discuss this issue in more detail in Section 3.1.

Liu et al. [42] explore the utility of representation layers and an "embedding layer" over the traffic flows in an initial unsupervised step. The authors use a stack of bi-directional Gated Recurrent Unit (GRU) layers connected to an encoder network. Autoencoder-based models have also been explored in other literature [5, 45]. However, it is difficult to identify a clear advantage in Autoencoder architectures over state-of-the-art CNN models in their evaluations. Therefore, we partly adopt the use of stacked recurrent units in our models, while proposing a lighter feature engineering that is more apt for general encrypted web traffic classification. On the other hand, we argue that the authors' analogy between the initial encoding layer to embedding layers in NLP (i.e., used to motivate their approach) is inaccurate. The NLP embedding layers capture the semantics of how words of a corpus appear in correlation to each other and complex linguistic semantics, while encoding layers find compressed representations of individual flows. The exact transfer of NLP methodology to traffic classification has been done in [55], but the research in this

area is in its infancy, and has not received much attention due to the restrictiveness of such methods.

Wang et al. [67] combine CNN and LSTM layers for learning both spacial and temporal features of the traffic, with raw one-hot encoded bytes (i.e., feature type 2) making up the input to the model. Although the objective of their training is intrusion detection, the effectiveness of their method is still significant to service classification. The authors show that though they might be outperformed by rival methods in detecting specific attacks, they are the most consistent in detecting attacks across all categories, with a 99% accuracy on the ISCXIDS2012 dataset [56]. Lopez-Martin et al. [44] use a similar approach for service classification, on a time-series features (i.e., feature type 3), and achieve 96% accuracy. However, the classes used in their learning are essentially protocols and inconsistent in granularity, a problem discussed in Section 2.5.1.

Two of the remarkable works that have greatly influenced our efforts are those of Anderson et al. [9] and Schuster et al. [54]. The former, although neither using DL nor focusing on service classification, conveys important insights about feature engineering. In their work, Anderson et al. [9] augment the typical features used in classic ML techniques (i.e., feature type 1), with: (1) values of certain fields from the TLS handshake headers, picked by a domain expert, and (2) packet length for the first fifty packets. Their evaluation shows a clear advantage of including these additional features, as the results of all models (i.e., Support Vector Machine, Linear Regression, Logistic Regression, Random Forest, Decision Trees, MLP) show higher accuracy at 0.001% False Discovery Rate (FDR)[3]. More recently [10], the authors propose a TLS client fingerprinting approach based on similar hand-picked features from the ClientHello message, and contextual information such as IP address, port number, and autonomous system. A weighted Naïve Bayes (NB) model is used for classification. While the approach is sound for client software fingerprinting, it is in contrast to our work which focuses on identifying regularities in traffic patterns rather than particular servers. Their approach relies on exclusively hand-picked features and data from all relevant clients. Moreover, while we appreciate the focus on simpler algorithms when they perform well, in our experiments lightweight traditional models, especially NB, perform very poorly for service classification. Lastly, in many use-cases of service classification (e.g., volumetry), *early* detection, which is an advantage of the approach in [10], is not a high priority.

Schuster et al. [54] also bring the importance of traffic shape to spotlight, showing that even the actual traffic content can be finger-printed based on traffic shape. The authors

[3]Accuracy at .001% FDR means the accuracy of the model, when it is only allowed one false positive for every 100,000 true positives.

model the traffic of Netflix, YouTube, and other streaming platforms over Dynamic Adaptive Streaming over HTTP (DASH) protocol as a time-series of down/up/all bytes-per-second, down/up/all packets-per-second, and down/up/all average packet length. Using these features and a standard CNN, they identify the video being streamed with an accuracy of over 98%, on a dataset that constitutes up to a 100 titles. The authors show that by analyzing the "burstiness" of traffic at different points in time, the actual title being watched under encryption can be determined with high accuracy.

Bronzino et al. [20] explore classification and regression models for inferring important QoS metrics of encrypted video traffic. The authors make use of traditional ML models such as Linear Regression, Ridge Regression, Support Vector Regression, Decision Tree, and Random Forest (RF) regressors, as well as RF classifiers. They leverage a carefully crafted set of statistical features (i.e., feature type 1) to predict the target metrics, effectively achieving 93% precision for detecting video resolution. While this approach is effective in predicting particular metrics (i.e., playback startup delay and resolution), the feature engineering is tailored for a specific task and does not generalize to various traffic classification objectives.

Rezaei et al. [49, 48] leverage CNN and CNN-LSTM architectures with certain adjustments to achieve high performance. Their focus, like ours, is on encrypted web traffic such as HTTPS. However, their dataset also contains non-encrypted traffic. In [48], for their CNN-LSTM model, the authors model the traffic sessions as a series of flows. From each flow, the first six packets are fed raw to the flow-level model (i.e., feature type 2). Their dataset is comprised of real-world mobile traffic, labeled with applications and 45% of the flows use SSL or TLS. The authors report an overall accuracy of 94.22%, but the accuracy for HTTPS alone is 75.43%. In their post-hoc analysis, the authors identify the importance of different parts of the TLS headers to their model by masking different portions of the input and evaluating its impact on the model accuracy. They find that the model does in fact heavily fit to cipher info and the SNI field, to the point that the accuracy of the model drops to 38% when SNI records are occluded. We discussed this as one of the pitfalls of applying DL to traffic classification in Section 1. Due to its high relevance, we use [48] for comparison of our model to the state-of-the-art.

In [49], Rezaei et al. propose a semi-supervised approach to address scarcity of labeled data. In pre-training, the convolutional part of the CNN-LSTM model is trained on unlabeled data, with statistical features of traffic (i.e., feature type 1) used as the target of training. In the subsequent supervised step, the authors connect the pre-trained CNN to the LSTM part and perform the training on labeled data. They demonstrate the effectiveness of their approach on both a small QUIC dataset they have made public, and the Ariel dataset [47] for HTTPS with an accuracy of 96% and 84%, respectively. The authors show

that without the pre-training step, their accuracy is lower.

2.5 Considerations

In this section, we provide some key insights and future directions that we gathered through the course of an extensive literature survey on TC, a summary of which was provided in Section 2.4. Our goal is to identify some of the aspects in which the state of TC has room for improvement. Some of these points were also briefly mentioned in Section 1.

2.5.1 Traffic Classification Pitfalls

TC is a relatively well-explored field of study, which is only expected to grow in popularity, partly because it is one of the most fitting areas of computer networking that can adopt recent advancements in ML. However, not all research in this field is geared towards practical usage. The inclination to report high accuracy and outperforming the state-of-the-art, has compelled numerous researchers to ignore some of the pragmatic aspects of traffic classification. We identify the following recurring issues as a few examples:

- **Inconsistent Labeling:** From a real-world, industrial usage point-of-view, there should be a clear objective to TC. Numerous works (e.g., [70, 44, 31]) in the area pick their labels rather arbitrarily, which is often not even consistent in granularity. For example, in some works [44] we have seen classes such as NTP and YouTube, a protocol and a specific application, used at the same time. For industrial significance, it is important that the labeling has a consistent level of granularity e.g., service, application, QoS level, etc. In this work, we follow a high-level categorization of the network usage. However, we believe there is room for standardization of these classes and the field's taxonomy.

- **Mixed or Non-encrypted Protocols:** Numerous works (e.g., [74, 66, 45, 68]) still make use of datasets that have a mixed set of protocols, which are easily distinguishable using header signatures. Realistically, there is no interest from the industry in DL-based TC, in use-cases where TC can be deterministically performed using existing commodity network devices. This shifts our attention to homogeneous datasets that contain a single encrypted protocol, where the services cannot be trivially identified. With the ossification of the Internet, HTTP is becoming the backbone of web communications, wherein new web protocols have made encryption the norm. Therefore, research efforts

must shift towards classifying standard web traffic, and away from using computationally expensive models used for performing tasks that either have deterministic solutions or can be achieved via simple ML solutions with high accuracy (i.e., traditional models and statistical features). Even very recently, the datasets used by many researchers include non-encrypted HTTP traffic too (e.g., [48]), which can result in highly misleading performance measurements and hide a model's shortcomings in encrypted traffic classification.

- **Ad-hoc Protocol-dependant Techniques:** Certain works in TC have proposed clever techniques that essentially guide the ML model based on the expert's domain-specific knowledge. For instance, Brissaud et al. [19] propose the use of non-Maximum Transmission Unit (MTU) sized packets for detecting the end of a data stream inside the HTTP/2 session. Although these works show good performance with relatively simple models in certain contexts, we argue that these achievements are short-lived, as the next version of the same protocol can easily jeopardize their approach. Therefore, research in TC should focus on fully automated methods that unleash the full potential of DL, and have high applicability to most encrypted protocols. We will discuss this further in Section 2.5.3.

2.5.2 Canary Features

In this thesis, "canary features" is a term we use for features that essentially *give away* the service labels. For future TC research, the use of such features should be avoided, as they direct the model to learn an oversimplified logic. For instance, the IP address and/or the Server Name Indicator (SNI) field can essentially give away the remote server's identity, and make it easy for the ML model to identify the underlying service. Given this information, there is little need for training an expensive ML model, as it can facilitate deterministic identification of services in real-time. A motivation behind encrypted traffic classification is the identification of services from applications, services and websites that have never been seen before. Therefore, the main focus should be on learning the actual traffic patterns that can characterize a certain service, as opposed to finding ways to directly infer the domain name or other identifiers of the server from the metadata.

In this work, we propose a pre-processing step where the canary fields, namely the certificate cipher record and the SNI field are masked. This is the application layer equivalent of masking the IP addresses, which is a standard in most works on TC. This forces the ML model to learn characterization of traffic patterns or the TLS protocol implementation,

instead of finding lazy ways to search the metadata for the server's identification, such as its domain name or certificate.

There is a tendency to oversell the state-of-the-art's success in providing real-world solutions for TC. Not masking the canary fields discussed above, is one major pitfall of the existing works (e.g., [58, 45, 44, 68, 4, 66, 47, 33]). Authors in [48] showcase the inadequacy of ML models when the SNI or the certificates are hidden, which otherwise promise over 90% accuracy in general TLS classification. Nevertheless, most works on TC continue to leverage canary features in their input. This is similar to having port numbers included in a model's input, where the task is to identify SMTP, SSH, HTTP, etc. One must note that the black-box nature of DL models, make it difficult to interpret under-the-hood operations and logic of the model.

Questionable datasets, especially those that are synthetically generated, are another primary issue. Data generation is often achieved via automated agents that are developed with Selenium [21] or other headless browser libraries and do not deviate from their instructions on visiting a certain website. Hence, the generated data is prone to be highly homogeneous, making service classification much easier than the real-world task. Our emphasis in using real-world TLS datasets is partly attributed to this fact. Another issue is that these datasets will also often include non-encrypted traffic, which is discussed in Section 2.5.1.

There are other less significant issues with the practicality of some of the works in TC. For instance, the assumption that the input flows to the model include a very large chunk of the entire user session, rules out the prospect of using the resulting ML model in real-time use-cases, even if the performance issues are ignored. This entails that the input needs to be minutes long before the ML model decides on a verdict, making the model unfit for real-time use-cases. For instance, the LSTM models in [67, 44, 50] need a discussion on how the cut-off threshold for the flows (*a.k.a.* Active Timeout) affects model accuracy.

2.5.3 Generalizable Methods

As discussed in Section 2.5.1, the constantly evolving nature of application-level encryption protocols calls for methods that are able to easily generalize to new protocols. We argue that the research in TC should lead to solutions that require little or no modifications to their core concepts and design, in order to be applied to different encrypted protocols. Adapting to a new protocol or an update to the existing ones should be possible through re-training on new data, with minor changes to the pre-processing and training procedure

itself. This way, the research in TC can have more continuity and the evolution of encrypted protocols would not require drastic changes to the way our methods work.

Although different protocols may leak different amounts of information, whether in their metadata fields or in other ways, it is crucial to find aspects of the traffic whose *availability* does not radically differ from one protocol version to the other. One such aspect is the traffic shape. Indeed, the traffic shape may exhibit noise and variations due to a large range of factors. Nevertheless, it will always remain an important indicator of the underlying traffic and unlike features like the SNI or header fields, it is very unlikely to be *unavailable* or obscured. It is conceivable that obfuscation protocols may deliberately try to mask the traffic shape. However, that requires a significant amount of redundancy, and is also very unlikely to be commonly adopted for general Internet use-cases in the near future. We will revisit this concept in Section 3.1.

Chapter 3

Design

In this chapter, we discuss our traffic engineering strategy that unifies the three categories of traffic features explained in Section 2.4 and is highly suitable for encrypted traffic classification. We also present our deep neural network model based on Stacked LSTM and convolutional layers, which is designed to work with our feature engineering and realize its potential.

3.1 Feature Engineering

Most works in DL-based traffic classification feed raw traffic bytes to the neural network model (cf. Section 2.4). Indeed, DL models are powerful enough to extract meaningful features from raw input on their own, provided a sufficiently large dataset. As discussed in Section 2.4, the notion of leveraging raw traffic bytes as model input is inspired from more conventional domains of DL, such as computer vision. Some works in traffic classification, such as Seq2Img [24], have gone as far as modeling the traffic as a two-dimensional image. However, as with the adoption of DL in any new domain, there are important considerations in traffic classification based on domain-specific knowledge of the task and the nature of the data.

An important distinction between network traffic and images is encryption, which is becoming the norm rather than the exception in ordinary web usage. A traffic flow or packet is often almost completely encrypted, except for the initial handshake and some of the header fields that are transmitted in plain-text. Therefore, in the computer vision analogy, a traffic flow is like an image that is completely obfuscated except for a small

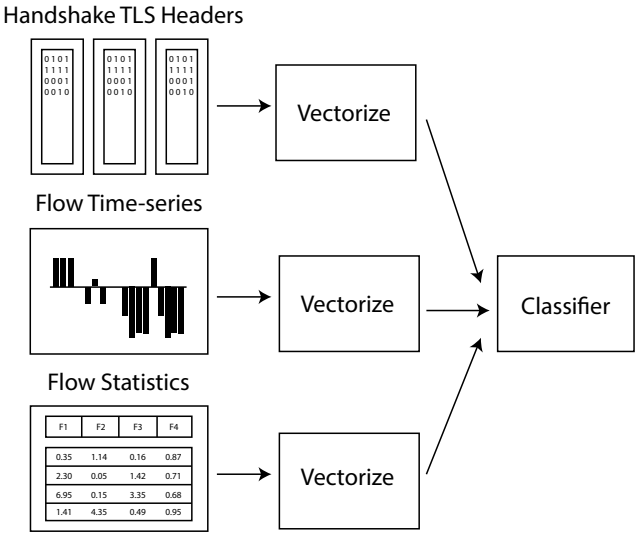


Figure 3.1: TLS headers from the handshake, flow time-series, and standard flow statistics as the DL model input.

area in it. Any effort to consume the encrypted portions of the traffic as the classification model input is essentially an attack on established encryption algorithms, such as Advanced Encryption Standard (AES), which is unrealistic.

Furthermore, it is crucial to consider what the DL model is exactly learning during training. For example, in an insightful post-hoc analysis, Rezaei et al. [48] show that the accuracy of their DL model completely degrades when the SNI field or TLS cipher info is masked. This implies that typical neural network models trained on raw traffic basically implement a look-up table, which predicts a class based on the server's identity exposed by certain TLS extensions. We referred to the parts of the traffic that expose the server's identity as *canary features* (cf. Section 2.5.2). There are three major drawbacks in relying on canary features: (i) An expensive deep neural network is used for implementing a relatively simple logic, which can be performed deterministically with a very low computational overhead. (ii) The performance of the DL model is highly dependent on seeing large amounts of traffic from all relevant servers in a service category (e.g., traffic from all video streaming platforms) in training. In other words, the model is not really learning anything about the nature of video flows in general. (iii) The availability of these identifiers of the server (e.g., plain-text SNI field) in-the-clear is crucial for the utility of the DL approach. If the SNI field becomes outdated or encrypted in the future versions of TLS, which is not unlikely with the advent of Encrypted SNI, the entire DL method can lose its effectiveness.

Our input to the DL model combines all three types of features, described in Section 2.4. As summarized in Figure 3.1, it is comprised of: (i) handshake header bytes, (ii) flow time-series, and (iii) flow statistics.

First, we include raw bytes from the handshake in our input to the model. However, we remove the canary features such as SNI and cipher info in our pre-processing, to diminish the model's reliance on that information. Also, due to our focus on encrypted protocols, we assume that L5-7 payloads contain very little information as they are expected to be encrypted. Therefore, there is no utility in including entire packets in the DL model input and the aforementioned payloads only create more ways for the model to overfit. Besides, packets other than the handshake packets (i.e., ClientHello and ServerHello messages) are redundant and expose virtually no meaningful information to the model. Thus, the raw traffic data for our DL model input, is truncated after the TLS headers of the handshake packets.

Secondly, we steer our DL model's focus on traffic aspects that are hardly affected by encryption. While the TLS records and extensions evolve over time and new encrypted protocols emerge with radically different characteristics, the traffic shape would always be available regardless of the underlying protocol. These kinds of traffic features can be quite effective for different traffic classification objectives, such as detecting a video being streamed using DASH [54] or distinguishing between malware and non-malware traffic [9]. We hypothesize that the traffic flow time-series of packet sizes, directions, and inter-arrival times (IATs) contain useful information for service and application classification as well. We discuss the flow time-series in more detail in Section 3.2.

Lastly, traditional flow statistics measured by standard flow-meters can also assist the model in traffic classification. Examples of traditional features include mean, standard-deviation, and median of packet sizes, number of different TCP flags, duration of the flow, etc. These features have been used for a variety traffic classification tasks for over two decades, and continue to be a simple yet powerful tool for distinguishing between different classes of traffic. This also allows for our overarching methodology to generalize for works such as [20], where a set of features are picked by domain experts for a particular traffic classification or regression task. Recall that Rezaei et al. [49] propose a semi-supervised traffic classification approach based on flow statistics, where the neuron weights from the unsupervised pre-training are used as a starting point for supervised learning. Their CNN model is initially trained on a large unlabeled dataset to compute standard statistical features. Subsequently, the pre-trained CNN is attached to an LSTM layer with time dimension over the different flows of a session. The new composite model is then trained on a smaller labeled dataset. While we appreciate the shift towards semi-supervised learning and the use of unlabeled datasets, steering the DL model towards statistical aggregations

is not ideal to harness the full potential of DL. Their approach guides the DL model to extract meaningful statistical insights from the traffic. However, if the weights learnt in pre-training are expected to stay relatively constant, one can deterministically compute the statistical features used in pre-training and offer them as the DL model's input. Moreover, DL is effective in detecting much more nuanced patterns in traffic, i.e., patterns that cannot be easily aggregated into high-level scalar metrics of the flow.

Note that the combination of handshake features and flow time-series was first proposed by [11] to detect malicious traffic. Aside from the use of statistical features as a third input, another key distinction between our approach and [11] is the use of DL to extract useful features of the handshake, while they require a domain expert to cherry-pick them for the TLS protocol. Though our research is focused on encrypted web protocols and mostly revolves around TLS, our feature engineering methodology is *protocol-agnostic*. Regardless of a protocol's implementation details, it is expected to have a negotiation or handshake segment, while the rest of the traffic would be fully encrypted. This segment's bytes will make up the only raw inputs to the model. The flow time-series, i.e., traffic shape and IATs, as well as flow statistics will always be available in IP traffic. Therefore, the model will simply have to be retrained and specialized for new protocol versions as they evolve, but our overarching feature engineering methodology will still be applicable.

3.2 Flow Time-series

Previously, we discussed the drawbacks of relying only on raw traffic bytes for traffic classification, which include learning over-simplified logic and poor transferability to future use-cases. Indeed, the goal of employing highly intricate DL models for traffic classification is to find distinctive and complex patterns pertaining to the nature of a traffic class. Therefore, we divert our attention to the aspects of traffic that are not used to identify a known server, but are rather innate to the service class itself.

One advantage of relying on the flow time-series features (cf. Section 3.1), is their relative independence from the implementation details of the protocol. While different extensions may be added to and removed from the protocol making it more difficult to classify traffic, characteristics such as the traffic shape and IAT of the packets will always be present, unless a protocol is intentionally designed to obfuscate such information. Though this is a possibility, as discussed in Section 2.5.3, designing such obfuscation strategies would have a negative impact on bandwidth, latency, and QoS, as it entails sending redundant traffic or delaying packets in order to manipulate the time-series. Therefore,

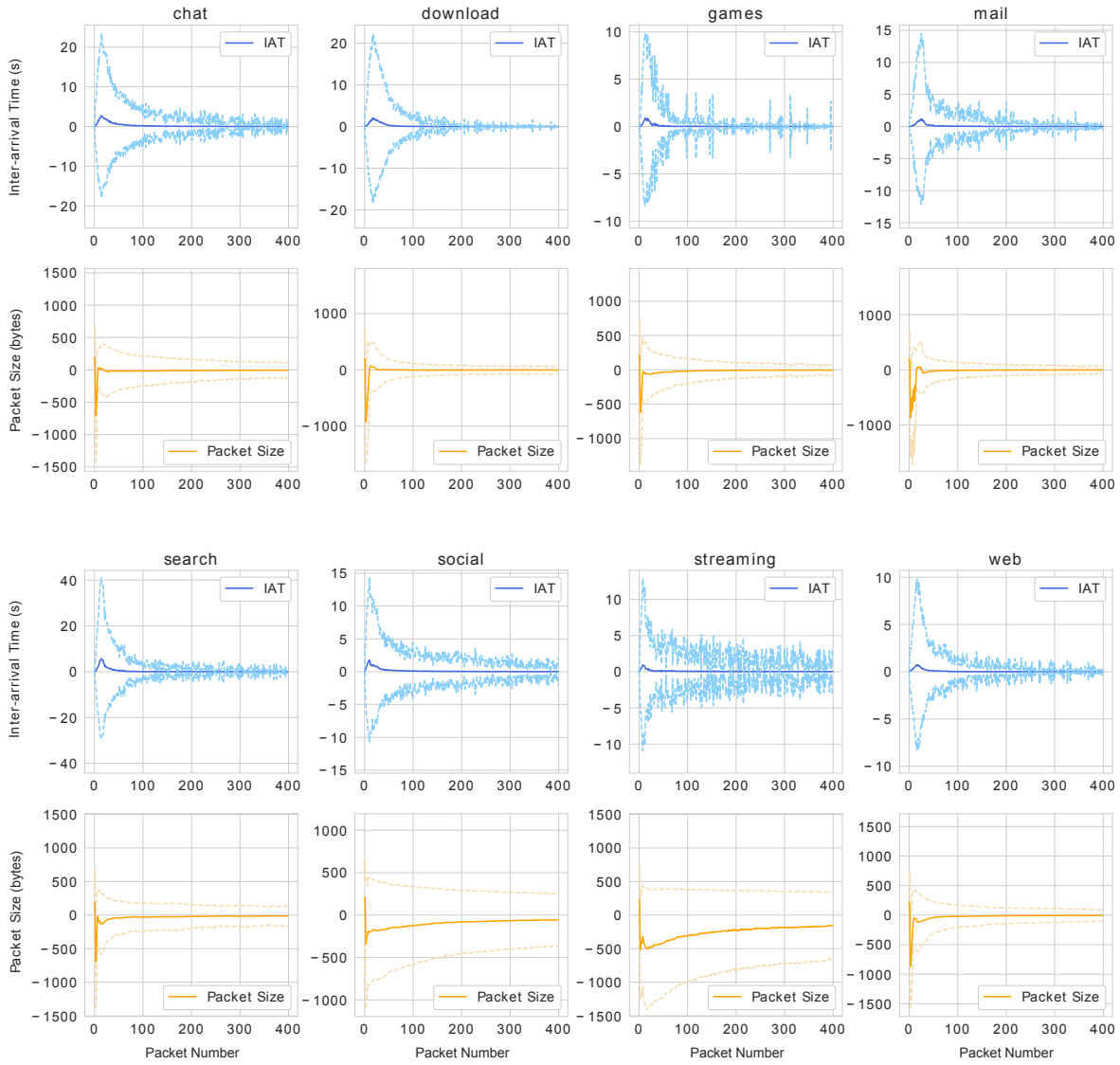


Figure 3.2: The average of flow time-series extracted from the Orange'20 dataset. The blue lines follow the IATs of the packets in the traffic flows. The orange lines show the size of the packets sent in each direction, where negative values indicate packets from flow's destination to its origin. The dashed lines delineate error bars of one standard deviation for each graph.

it is unrealistic that there would be enough motivation for introducing such measures in ubiquitous web protocols.

Figure 3.2 shows the difference of flow time-series between various classes in our dataset (cf. Section 4.2.1), where the time-series of packet IATs and sizes are averaged for each class and the error is plotted with dashed lines. At a glance, there are subtle but visible differences between the traffic shapes of different classes. Note that the packet size time-series has a positive value when the packet goes from the flow's origin to its destination, and negative otherwise. In other words, we multiply the packet size by -1 when it's going from flow destination to flow origin. As revealed later in our experiments, deep neural networks (e.g., CNN and LSTM layers) are highly effective in detecting these nuanced patterns for each class, and traffic classification can be enhanced by including the time-series in the input. By combining these features with raw bytes, we can create a powerful feature set that can be used for learning the nature of traffic, as well as identifying useful parts of the secure protocol's headers for identifying applications.

3.3 Model Architecture

Our neural network architecture reflects the structure of the features presented in Section 3.1. As shown in Figure 3.3, our neural network model separately processes the flow time-series, the handshake TLS headers, and the standard flow statistics as input. Each of the three inputs is fed to a separate set of neural network layers, and the output of those layers is later concatenated and passed through additional fully-connected layers to produce the final prediction.

The raw handshake bytes are fed to a deep one-dimensional CNN with max-pooling layers in between. The structure of these layers is quite standard and a one-dimensional equivalent of commonly used computer vision models, which has proven effective on network traffic [45, 4, 48]. We only feed the first C bytes of up to three ClientHello and ServerHello packets from the flow to the model. In our evaluations in Section 4.4, it is proven that there is no real disadvantage in omitting the rest of the traffic, which has strong implications for future research in this area. The value for C in our experiments is 600 bytes, which is picked through a hyper-parameter search described in Section 4.4.1.

The flow time-series has three channels: (i) IAT, (ii) size, and (iii) direction. In our experiments, we found that a stacked LSTM architecture preceded by a dense layer, is extremely effective in processing the flow time-series features, while one-dimensional CNNs are also viable (cf. Section 4.4). In our implementation, we use a stack of three LSTM

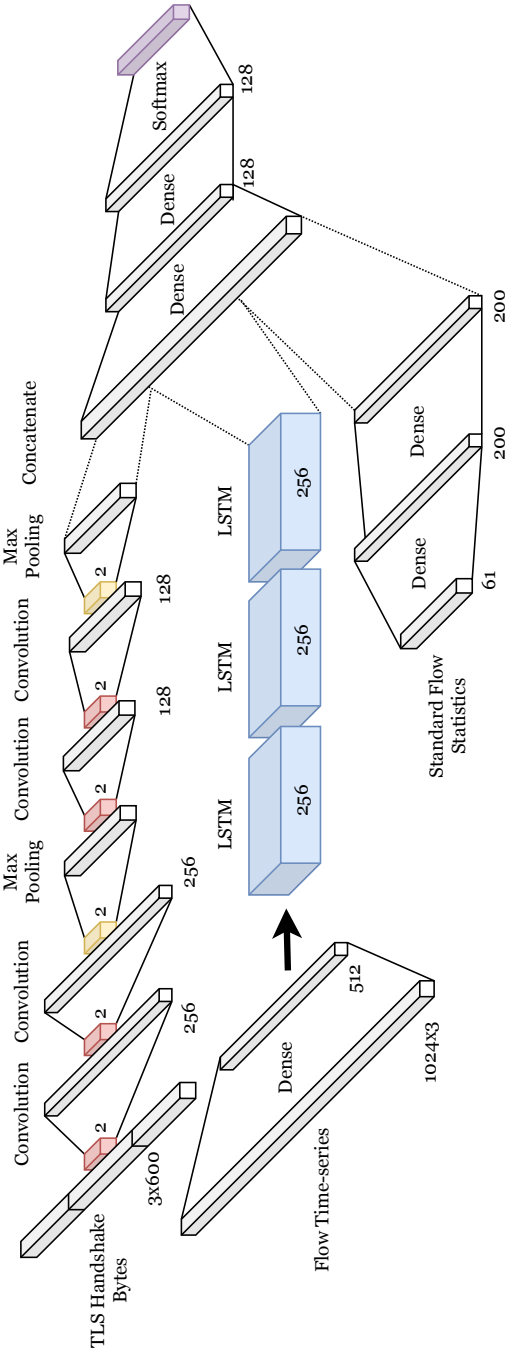


Figure 3.3: Tripartite neural network architecture

layers going through the flow time-series in both directions (cf. Section 2.2). We also include flow statistics extracted using CICFlowMeter [40]. Since these features do not have a natural ordering or sequentiality, a fully-connected network is used to ingest them.

One of the major advantages of our feature engineering is the ability to include information about a large number of packets without substantially increasing the model size. In a classic raw input approach, it is normal to include the first b bytes of the first k packets of a flow to the model. The size of the input grows linearly by increasing k, which can create a super-linear increase in the number of model parameters and quickly lead to overfitting. In contrast, our model limits the raw traffic to the handshake packets, and uses a lightweight representation with only three channels for the other packets of the flow. This allows the model's scope to grow and consider hundreds of packets without a significant impact on its complexity. The outputs from these three parts (i.e., flow time-series and statistics, and TLS headers) are concatenated and passed through multiple additional dense layers, which yields the output of the network as a softmax layer.

Our early experiments showed that the models are highly prone to overfitting. This is not surprising considering the fact that the number of parameters in the model to be trained is in the order of millions. It is not uncommon for traffic classification models to be trained on datasets which have an order of magnitude less entries than the number of trainable parameters in the model. To overcome the problem of overfitting, we use very high drop-outs (i.e., up to 50% at some layers), especially in the final dense layers. As showcased in Section 4.4, our feature engineering itself has a tremendous effect in lowering the chance of overfitting when compared to the conventional raw traffic input.

3.4 Labeling based on Server Names

Pre-processing is a very important step in traffic classification, which is sometimes overlooked despite the fact that it can significantly impact the accuracy of the model. A primary challenge in the application of ML to network systems is the scarcity of labeled real-world packet traces (cf. Section 2.3). Due to privacy concerns, administrators are reluctant to publish real-world network data to the public. An alternative is to use synthetic datasets. However, they often fail to capture the true distribution and patterns of real-world network traffic, which significantly affects model generalization and makes their evaluations questionable. It is also generally difficult to label traffic at its origin, as it requires users to actively participate in the process and log their activities.

In contrast, raw unlabeled network traces are available in abundance to the service providers. We leverage the SNI field of TLS records to extract the server names. A

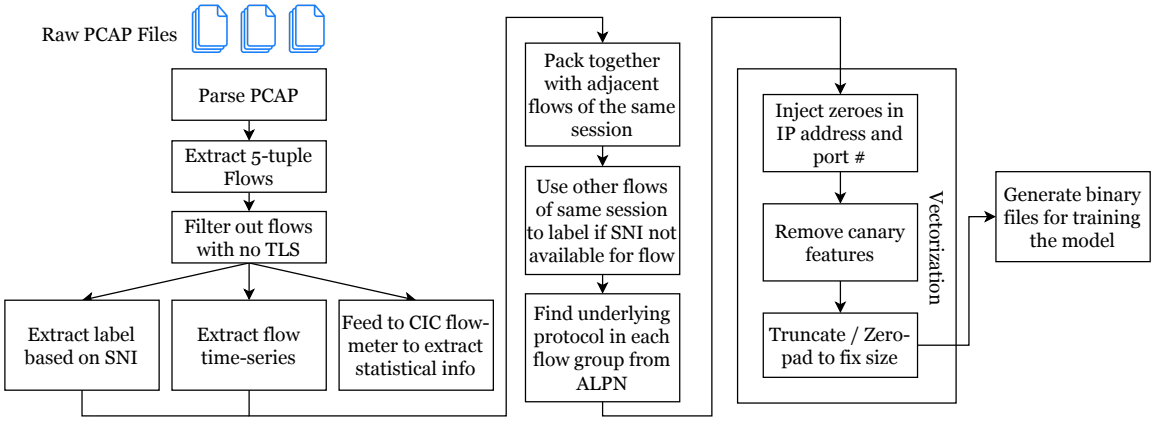


Figure 3.4: Overview of the pre-processing steps

look-up table is used to match server names to their respective classes. Each domain name from the SNI field is matched against a set of regular expressions that are either carefully handpicked (i.e., by monitoring the traffic of prominent websites and mobile apps, e.g., Netflix, YouTube, and AppStore), or gathered from a dataset of categorized domain names, such as the Blacklists UT1 dataset [1]. For certain providers (e.g., Google and Facebook) extra care must be taken, as similar sub-domains may be shared between multiple service categories. The granularity of classification can be set by simply modifying the look-up table. For instance, based on whether we would like to perform application-level or service-level classification, we can change how server names are mapped to labels.

3.5 Pre-processing

We design our pre-processing to be performed in a distributed fashion using Apache Spark [73]. As shown in Figure 3.4, the following steps are taken in the pre-processing of the data:

1. Extract flows (i.e., 5-tuples of src/dst IP/port and protocol) via standard flow-meters like YAF [34].

2. Filter flows with TLS packets, as we are particularly interested in encrypted web traffic.

3. Extract basic flow information, such as the flow start and end times, packet count, byte count, flow time-series, etc.

4. Extract statistical features for the flow using CICFlowMeter [40], and store it as metadata.

5. Extract SNI domain name and assign label based on a look-up table.

6. Group flows from the same TLS session ID together. If TLS session ID does not exist, time proximity and NAT-aware source and destination IP & port numbers are used.

7. For each unlabeled flow f, check other flows in the same session as f, and use their label for f. Often in multi-flow TLS sessions, only the first flow contains the SNI record.

8. Vectorize the flow into a time-series of binary information as follows: (i) mask IP addresses by injecting zeroes even if they are already randomized, (ii) remove TLS cipher information, (iii) mask the SNI record, and (iv) truncate to MTU size or zero-pad the packet bytes—ensure fixed vector size.

9. Write raw traffic bytes to binary files, with each entry having an array of vectorized bytes from up to three handshake packets. Include flow statistics and a time-series of maximum length 1024 with the three channels for packet sizes, directions (± 1), and IATs for each entry as well.

3.6 Training strategy

In addition to the model itself, the training strategy (i.e., the optimization process) has an impact on the end result as well. In our approach, we use an Adam optimizer [39] that performs stochastic gradient descent based on adaptive estimates of lower-order moments. We have learned that one effective strategy is to manually lower the learning rate as the training progresses.

Since the learning rate decides the granularity of the parameter search, there is an inherent trade-off in the choice for its value. Small values for the learning rate can lead to slow training and over-exploration of local optima in the loss function's optimization. Excessively high values for the learning rate can prevent enough granularity in the parameter search and lead to low accuracy in the final model, as we leap over optimal configurations.

By manually reducing the learning rate after every few epochs, we can balance this trade-off. Our strategy is to start from a default learning rate (e.g., 10^{-3}) and decimate the learning rate every 10 to 20 epochs when the overall validation loss starts to flatline. This way, we essentially increase the resolution of the search as the training progresses. The optimizer will start with making large leaps in the parameters and gradually decrease the step size in order to learn a highly optimal set of weights for the DL model.

Due to the imbalance in the number of dataset entries for each class, we also employ an upsampling strategy, which increases the impact and weight of entries from smaller classes during training, proportionate to the size of the classes. Although quite simple, the upsampling strategy considerably enhances the model's overall performance.

3.7 Software Implementation

In the interest of transparency and reproducibility, not only the dataset used in this project has been made publicly available (after certain transformations to address user privacy concerns), we have also designed, implemented, and released a full software API for data pre-processing, model training, and model evaluation code-named *DeepTraffic*.[1] In this chapter, we explain the design concepts and some of the software engineering decisions made throughout the development of the DeepTraffic library.

Our software API, consists of an end-to-end ML pipeline that is implemented in Python. In high-level, the library performs three key operations: *pre-processing*, *training*, and *volumetry*. The overall structure of the API is shown in Figure 3.6.

At the first stage, raw packet traces will be fed into the pre-processor module as PCAP files. The module processes the PCAP files by splitting them into flows, extracting basic statistical information, enriching them with metadata, and writing the results as binary files to an arbitrary output directory. The structure of the binary files is hidden from the user and the dataset can be conveniently loaded as a whole using the Python library.

Upon loading the dataset, the library can dynamically apply the labeling look-up table at run-time. Hence, there is no need to repeat pre-processing when the labeling is changed (e.g., going from service-level to application-level labels). The training module can take in the dataset along with the model specifications, and fit a deep neural network model to it. Although there are many configurations that can affect the model, the library is designed to conveniently work with zero configurations, if needed. If the user wishes to

[1] Code base is released and maintained at https://bit.ly/uw-orange-codebase

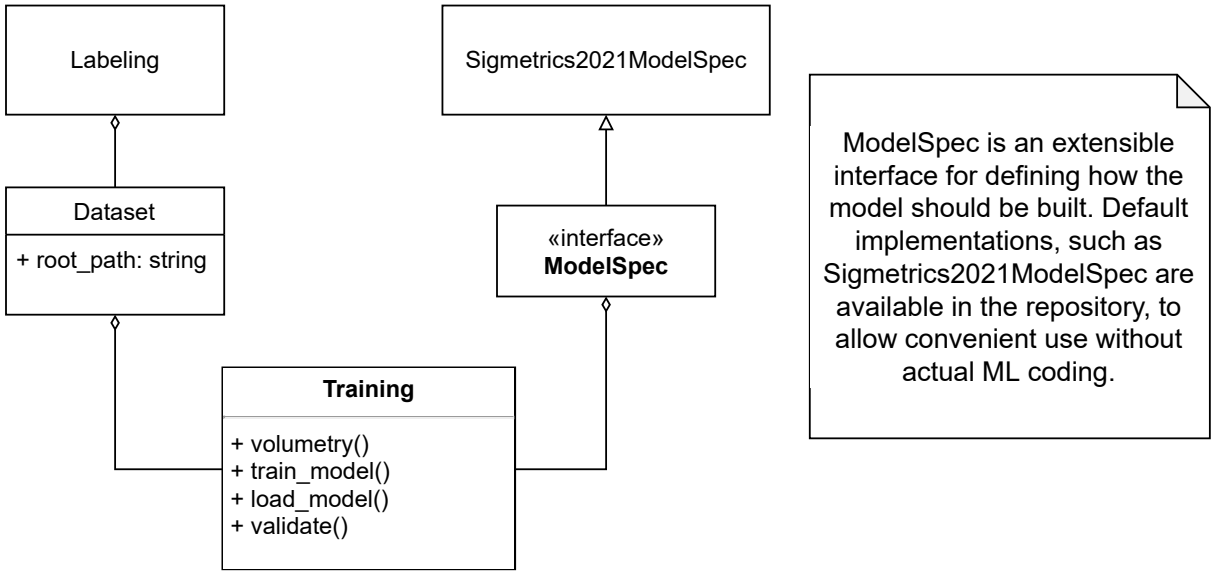


Figure 3.5: UML Class Diagram depicting the object-oriented structure of the user-facing classes in the DeepTraffic library

customize certain aspects of training, they can do so by using the user-friendly YAML [16] configuration. Our repository has an in-depth documentation in this regard. The object-oriented structure of the API is designed with extensibility in mind, to support customization and adaptation to all user needs. Figure 3.5 depicts the class structure of the library.

Lastly, the training module is also capable of invoking the *volumetry* API to classify potentially unlabeled data. Volumetry is defined as the act of measuring the amount of traffic from each class in terms of bytes, packets, or number of flows, given a sample network trace as the input. In this case, the trained neural network model is used as a predictor for identifying each flow's assigned label. The output of the `Training` object's `volumetry` method is the number of packets, flows, and bytes of each class in an arbitrary input traffic dataset, along with the ML model's average confidence for its predictions (cf. Appendix .6).

It is important to note that this software API is a Python/Linux library, which is based on NVIDIA CUDA [52], Tensorflow [2], Apache Spark, and standard flow-meters.[2] The use of these well-established technologies ensure that the code base is horizontally scalable

[2]The in-depth technical details of how the library should be used and how its environment should be set-up on Linux systems is also available in the code repository.

and resilient.

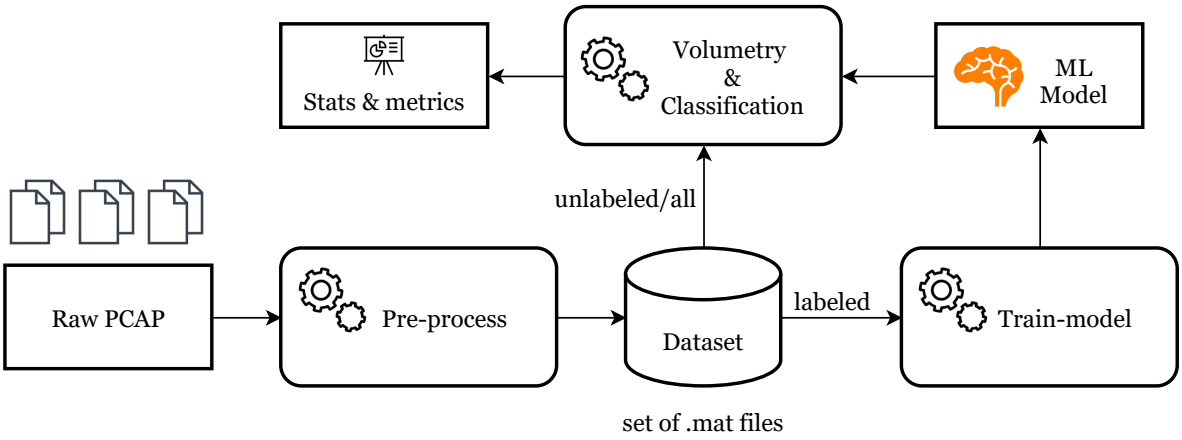


Figure 3.6: The overall structure of software API

Chapter 4

Evaluation

For showcasing the effectiveness of our method proposed in Chapter 3, we evaluate our model on a real-world dataset collected by our industry partner. In this chapter, we begin with describing the software, hardware, and datasets used in our evaluation. We then demonstrate the model's performance in service-level classification and compare it against the state-of-the-art. We also outline important insights about the model's behaviour and its effectiveness over different protocols and classification granularities.

4.1 System Specification

We conduct our experiments on a machine with one NVIDIA Tesla P40 GPU 24GB GRAM, 56 Intel(R) Xeon(R) Gold 5120 2.20GHz CPUs and 376 GBs of RAM. The software stack for training and pre-processing includes Centos 7, CUDA 10.1, Tensorflow 1.15 and PySpark 2.4.4. The code base is developed with Keras [29] for convenience.

4.2 Dataset Description

4.2.1 Orange'20 Dataset

The primary dataset used in our work is provided by Orange S.A., a major ISP in Europe. The dataset was collected on July 11, 2019 for about 80 minutes, from the ISP's mobile network. For privacy concerns, the IP addresses are masked and the packet payloads are

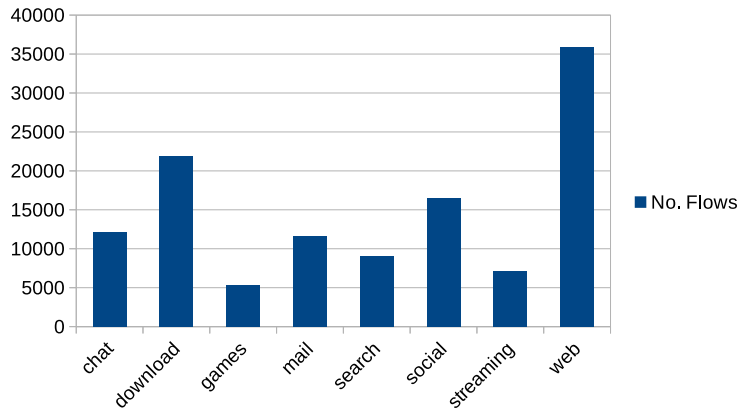


Figure 4.1: Breakdown of the approximately-labeled portion of the ISP network dataset per class

removed with the exception of TLS headers. The entire dataset has more than 800K unlabeled flows, where about ∼300K are TLS flows and of interest to us. We use the SNI field to label the TLS flows (cf. Section 3.4) with the following service categories: (i) chat, (ii) download, (iii) games, (iv) mail, (v) search, (vi) social, (vii) streaming, and (viii) web. A total of 119,565 out of the 343,228 TLS flows are labeled, using our approximate labeling scheme, with both manually picked URLs and the UT1 dataset [1]. The distribution of service categories in the labeled dataset is shown in Figure 4.1. In the spirit of transparency and openness in research, we have released the pre-processed dataset (cf. Section 3.5) with certain adjustments for privacy compliance to the public.[1]

4.2.2 UC Davis QUIC Dataset

The QUIC dataset has been recently released by the authors in [49]. It comprises of 3,637 flows, classified into Google Docs, Google Drive, Google Music, Google Search and YouTube. This is natural as Google is currently the primary advocate for the QUIC protocol's adoption in the industry. The dataset is relatively balanced, with no class being twice as large as the others. Furthermore, it is partly generated by human users and partly via automated agents.

[1]The dataset is available for download at http://bit.ly/UW-Orange-2020

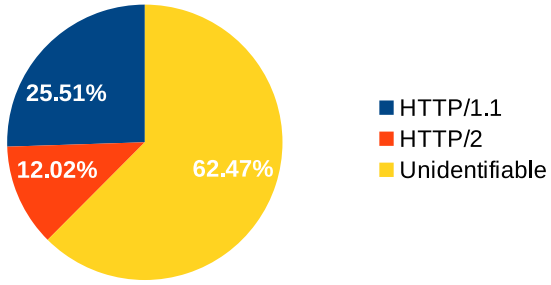


Figure 4.2: The percentage of TLS sessions by the overlying protocol. Note that not all TLS sessions can be identified, as the ALPN and NPN records may not be available (cf. Section 4.4.2).

4.3 Evaluation Methodology

Evidently, the accuracy of a traffic classification model by itself can not be thought of as a global KPI for a model's strength, as the model performance can significantly vary from one dataset to another. Hence, the significance of these results can only be verified when compared against existing models in the literature.

For comparison, we implement the CNN and CNN-LSTM models proposed by Rezaei et al. [48], which have shown good performance in application-level traffic classification. The authors model the input as a series of flows, which can be thought of as a *user session*. The CNN model operates on the *flow* level, i.e., the first 256 bytes of the first 6 packets of a single flow in the series are fed to a deep CNN. The CNN model is very similar to the header part of our model (cf. Figure 3.3). On the other hand, the CNN-LSTM model receives the session (i.e., a time-series of flows) as its input, and essentially makes the CNN model time-distributed over the flows of each session and labels the entire session. Similar architectures have been employed over the years for encrypted traffic classification [5, 68], making it an ideal baseline to compare our model against.

4.4 TLS Classification at Service-level

We start by evaluating the performance of our feature engineering approach and DL model architecture on the Orange'20 dataset. The data is pre-processed according to Section 3.5

and comprises of TLS flows only. We compare our DL model against the state-of-the-art CNN and CNN-LSTM architectures, showing a clear advantage and asserting our contributions.

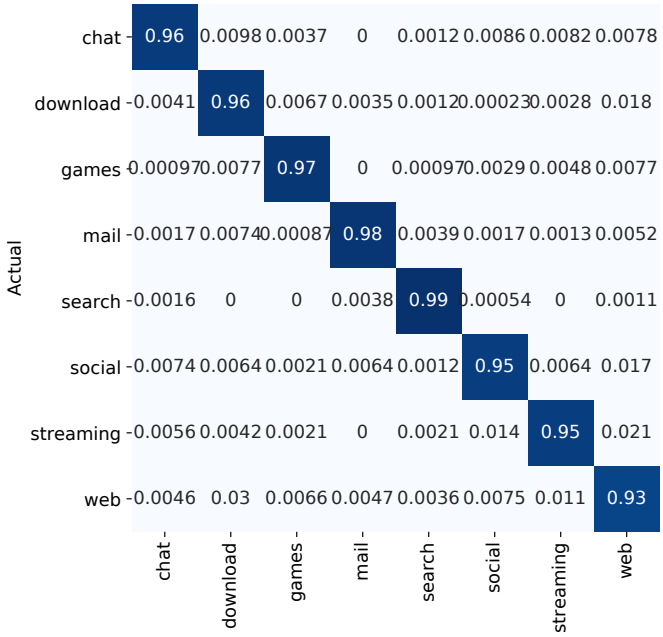
Our model is trained using the Adam optimizer for 40 epochs, with 20% of the dataset used for validation. The learning rate is set to 0.001 at first and reduced every 10 epochs. The results of the experiment are shown in Figure 4.3, with an overall accuracy and weighted average F1-score of 95.56% and 95.58%, respectively. Figure 4.3b shows the per-class precision, recall, and F1-score of our model. The F1-score is over 94% for all classes, which implies very good stability despite the highly imbalanced classes in the dataset. This can be attributed to the upsampling strategy employed during training. As a baseline, a C4.5 model is trained on statistical flow features. C4.5, among other DT-based algorithms, is a popular choice in traditional traffic classification [18, 61, 8] but only achieves an accuracy of 81.39%, as shown in Table 4.1. We attribute the disadvantage of the traditional approach in part to its sole reliance on high-level statistics and not being able to make distinctions based on more fine-grained details of the traffic shape.

The advantage of our model is clear when compared against the UCDavis CNN model, as shown in Table 4.1. When evaluated on the Orange'20 dataset, the UCDavis CNN model in [48] achieves an accuracy of 91.09% after 20 epochs, which is 4.5% lower than our three-part model in Figure 3.3. This is a significant gain in performance with 50.39% reduction in false classifications.

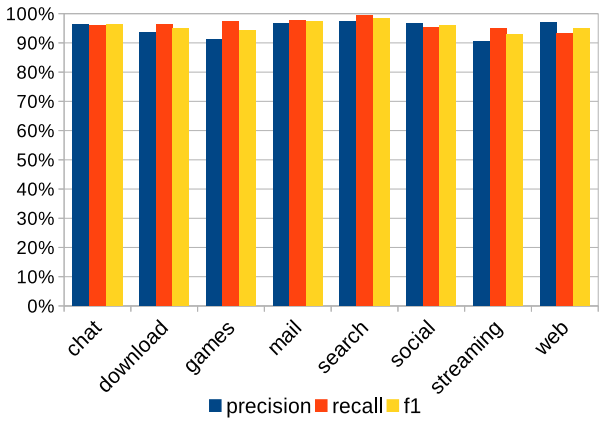
In Figure 4.4, we highlight the training progress to reason about this performance gain. The figure shows validation loss, training loss and accuracy at the end of each training epoch. Evidently, the UCDavis CNN model quickly overfits to the dataset. This is primarily due to a larger raw traffic input, only a part of which is actually useful to the model. After 12 epochs, the training accuracy is perfect, while the validation performance fails to improve. In contrast, the validation and training accuracies converge very well in the case of our model[2], as our feature engineering only provides the useful information for encrypted flows (i.e., handshake and flow shape). The implications of these results are significant. Despite having access to twice the number of packets, the competitor UCDavis CNN model is far less effective, as all meaningful information lies in the handshake. Exposing a larger chunk of the raw traffic to the UCDavis CNN model, simply confuses the model and provides more ways to overfit.

Table 4.1 depicts the performance of other variations of our model. We replace the

[2]The validation accuracy reported is actually higher than training accuracy. This is due to the *high dropout* strategy discussed in Section 3.3. During training, the dropouts are active, which in the case of very high dropout values can cause the training accuracy to suffer.

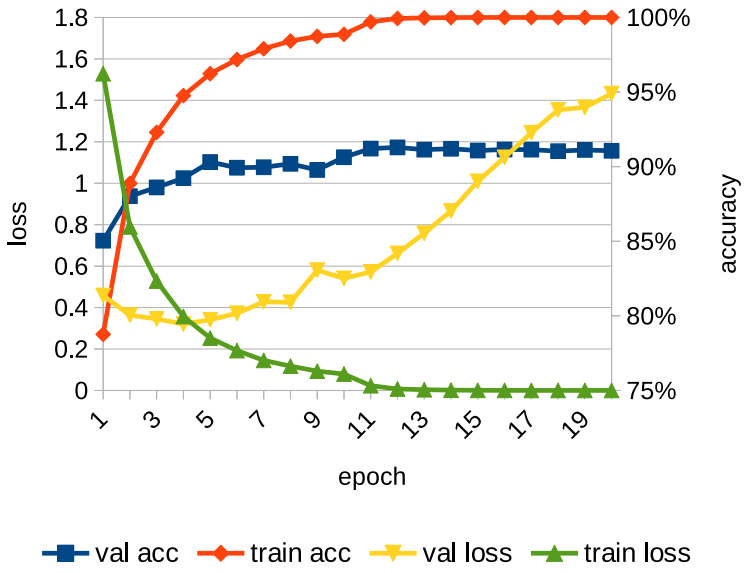


(a)

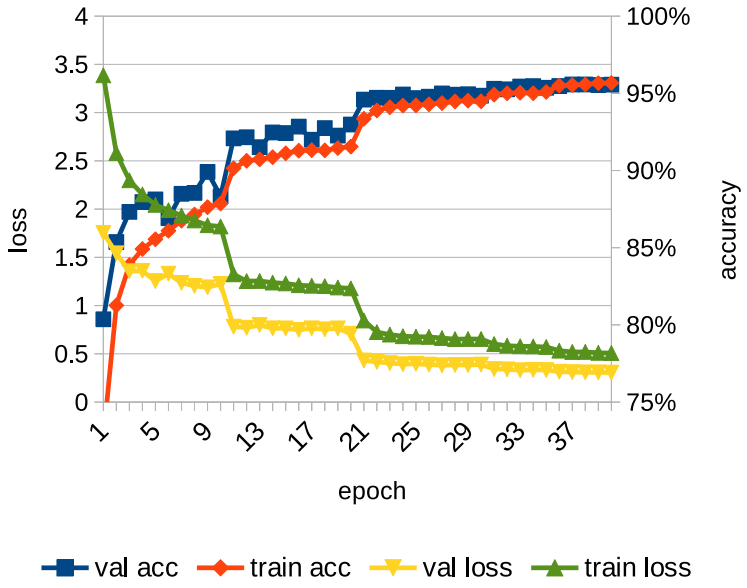


(b)

Figure 4.3: Confusion matrix (a) and per-class precision, recall, and F1-score (b) for our model's evaluation on the Orange'20 dataset. Our classifier consistently achieves +94% F1-score across all classes.



(a) UCDavis CNN model



(b) Our model

Figure 4.4: The progression of training and validation accuracy, and loss during the training of the UCDavis CNN model [48] and our model

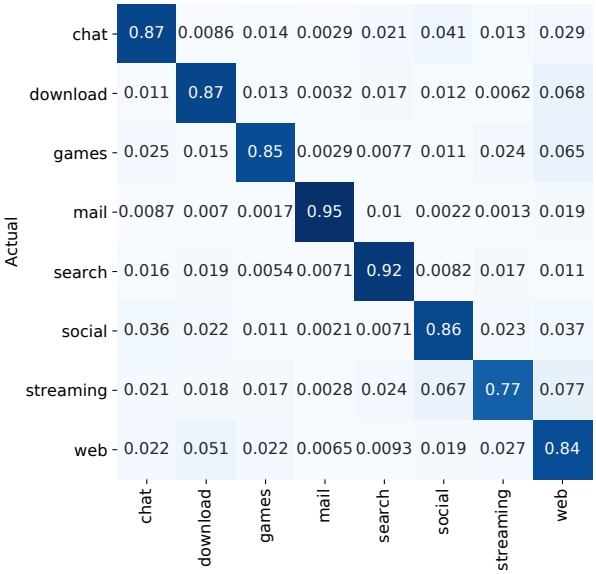
	W Avg precision (%)	W Avg recall (%)	W Avg F1-score (%)	Accuracy (%)	Epoch Time (s)
Full Model (SLSTM)	95.62	95.56	95.57	95.56	2584
Full Model (CNN)	94.54	94.42	94.37	94.43	232
Flow-only Model (SLSTM)	86.71	86.51	86.56	86.51	1814
Flow-only Model (CNN)	76.77	73.17	73.76	73.17	211
UCDavis CNN [48]	91.09	91.06	91.04	91.05	168
UCDavis CNN-LSTM [48]	89.74	89.72	89.73	89.72	245
Traditional Baseline (C4.5)	81.56	81.39	81.41	81.39	18*

Table 4.1: Performance comparison of TLS flow classification models (*the training time reported for the C4.5 model is for the entire training)

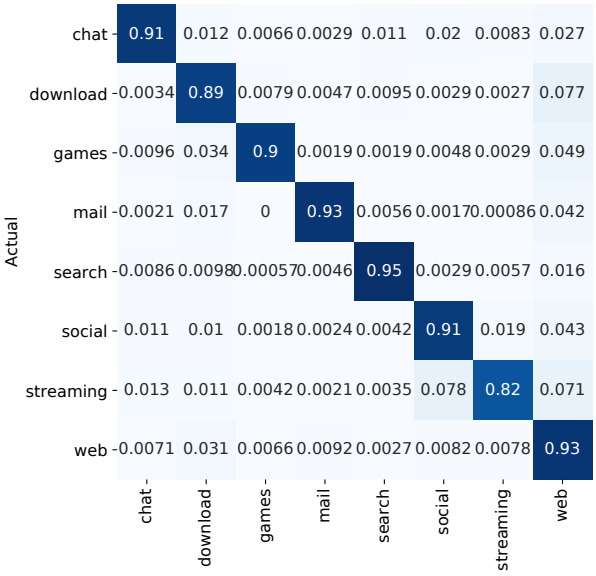
Stacked LSTM (SLSTM) layers in the original model (cf. Section 3.3) with a deep one-dimensional CNN (cf. Appendix .2), which is also a reasonable network for consuming a one-dimensional time-series. Though inferior to Stacked LSTM, the accuracy and F1-score of the resultant model is 94.43% and 94.37%, respectively. Nevertheless, it has a clear advantage over the UCDavis CNN model, which only processes raw traffic, with 37.77% less false classifications. The advantage of employing CNNs on the flow time-series side of the model, however, is in higher training speed (232 vs. 2,584 seconds/epoch), which is close to our competitor model despite being much more accurate. LSTM networks are notorious for being computationally expensive to train, and Stacked LSTM layers are even more so. Nevertheless, as model training is often a one-time investment and with the rapid advancement of computational hardware, this is a reasonable cost for higher classification accuracy.

We attribute the superior performance of our model to the flow aspect of our feature engineering and the Stacked LSTM layers. In fact, the flow time-series, despite being a simple feature set to model the traffic, is quite effective by itself. In Table 4.1, we also showcase results of our model variation with all the other inputs and their corresponding layers removed, except the flow time-series. We refer to these models as *"Flow-only"*. In this case, the Stacked LSTM and deep one-dimensional CNN architectures achieve an accuracy of 86.51% and 73.17%, respectively. Although being inferior in performance to models that also include raw traffic as input, it is important to note that the flow time-series features will always be available regardless of how the encrypted protocols evolve. These features enable a model to learn about the nature of traffic categories themselves, rather than fingerprinting a particular set of servers. Therefore, for all future research, we instigate the use of these flow features as a baseline for evaluations.

It is important to note that the UCDavis CNN model depicts a high misclassification between the streaming and social classes, where mutual providers such as Facebook and



(a) Flow-only Stacked LSTM



(b) UCDavis CNN

Figure 4.5: Performance of UCDavis CNN model [48] and our Flow-only Stacked LSTM model

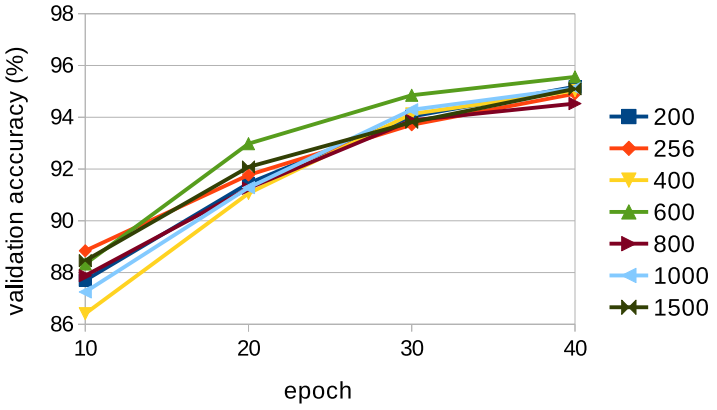
Twitter exist, as shown in Figure 4.5. This is a reoccurring issue with DL models for traffic classification that only rely on raw TLS bytes and are more tuned towards identifying a *server* rather than a *service*. The Flow-only model, despite having access to very simplistic traffic features, makes fewer misclassifications between these classes, and when used together with the handshake bytes in our full model, is able to alleviate the difficulties in distinguishing between the two classes.

A rather surprising result in Table 4.1 is the performance of the UCDavis CNN-LSTM model [48]. As mentioned in Section 4.3, the UCDavis CNN-LSTM model is time-distributed over the flows of a "session", and theoretically has access to more information in comparison to our model that processes flows individually. However, the model's access to full traffic bytes does not work in its interest, and though having more parameters and capacity than the UCDavis CNN model, it overfits more severely to the data achieving a slightly less accuracy of around 90%. In fact, similar to the UCDavis CNN model, it quickly rises and achieves perfect training accuracy at around the seventeenth epoch, but fails to increase the validation accuracy any further.

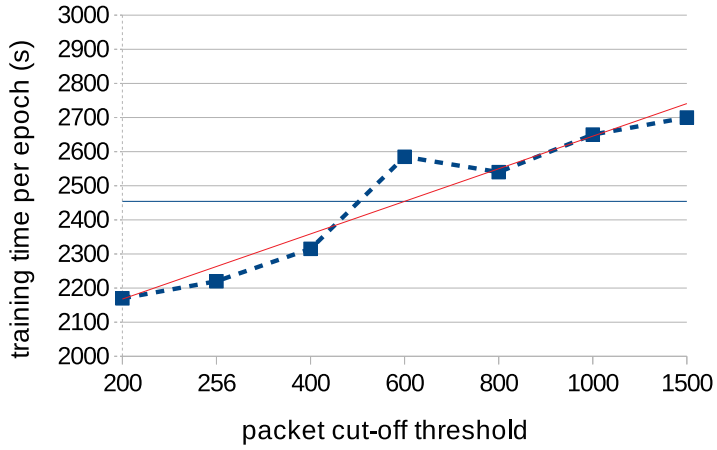
4.4.1 Packet cut-off

We discussed in Section 3.5 that when the traffic bytes are pre-processed, we only truncate them to MTU for the sake of having fixed sizes. However, when loading the data for training, we additionally truncate the packets of the handshake to a cut-off of C bytes. Figure 4.6a shows the validation accuracy of the models with different values for the hyper-parameter C, at each epoch of training.

There is clearly a trade-off when increasing the value of C. Exposing a larger portion of the packet headers can give the model more information to work with. However, it also increases the capacity and size of the model, making it more likely to overfit to the training data. Based on the results, we found $C = 600$ to be a good value for the parameter. The final accuracy varies between 94.53% and 95.56% for all the models we considered. Note that the input is zero-padded at the end if the packet's header is smaller than C. Therefore, given that not all packets have useful information in the first C bytes, it is not surprising that a middle value for C works better on average in the aforementioned trade-off. Training time per epoch is displayed in Figure 4.6b. As expected, the training time increases almost linearly with C, since the number of parameters on the raw bytes side of the model increases.



(a) Accuracy per cut-off value



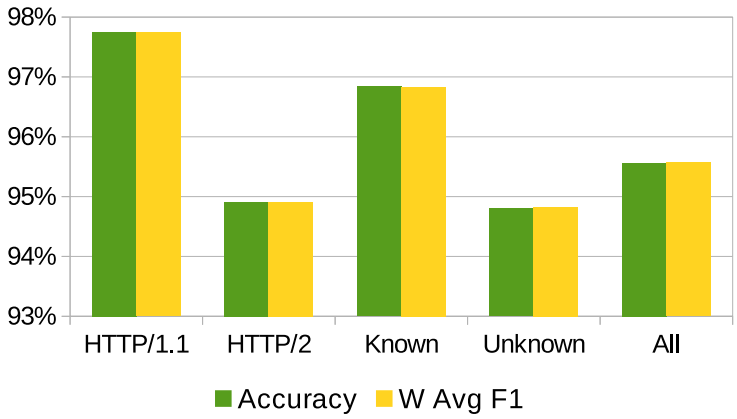
(b) Epoch time

Figure 4.6: Our model's performance for different values of packet truncation cut-off (C)

4.4.2 HTTP Versions

The final insight here pertains to the HTTP version and how it affects the performance of our model. In Sections 1 and 2, we mentioned that HTTP/2 brings new features to the web, but at the same time complicates traffic classification. We evaluated our model's performance on different subsets of the validation set, based on the HTTP version. Not all flows captured in the Orange'20 dataset have the Next Protocol Negotiation (NPN) or ALPN records available to identify the protocol used over TLS. The "known" and "unknown" in Figure 4.7 elude to this fact.

As expected, the performance of the model on HTTP/1.1 is higher than HTTP/2,



(a)

	W Avg precision (%)	W Avg recall (%)	W Avg F1-score (%)	Accuracy (%)	% of dataset
HTTP/1.1	97.79	97.75	97.74	97.75	25.51
HTTP/2	95.12	94.91	94.91	94.91	12.02
Known	96.88	96.84	96.83	96.84	37.53
Unknown	94.94	94.80	94.83	94.80	62.47
All	95.62	95.56	95.57	95.56	100.00

(b)

Figure 4.7: The impact of different HTTP versions on the performance of our model. Note that not all TLS sessions can be identified as the ALPN and NPN records may not be available. The flows without next protocol information are denoted as *"unknown"*.

i.e., 97.75% vs. 94.91%, due to the latter being a more complex protocol with features such as multiplexing, which make traffic classification more difficult. More importantly, the flows captured without the ALPN/NPN records are also generally harder to classify than the rest. This is due to the fact that there is a higher likelihood that these flows are captured from the middle of a session. Hence, they contain less information in their beginning for the model to leverage. It should also be noted that there is a disparity between the number of HTTP/2 and HTTP/1.1 flows in the training set. This reinforces the model's better performance on HTTP/1.1.

4.5 TLS Classification at Application-level

Many works in traffic classification (e.g., [5, 48]) focus on application-level classification, which is at a finer granularity than our labels in Section 4.4. While counter-intuitive, application-level classification is often an easier task, especially when model is closed-world (i.e., dataset entries strictly belong to one of the n known applications) or canary features are not occluded. If the DL model is trained with the objective of a look-up table for identifying servers themselves (cf. Section 3.1), application-level classification is generally easier for the model, as it does not need to learn what behaviors are *shared* between different applications of the same service category.

In order to evaluate our model in application-level classification, we identified 19 fine-grained labels that have enough representative flows for the training to be consistent. The distribution of these labels that make up for 82,776 flow entries of the dataset (i.e., $\sim$ 70%) is shown in Figure 4.9. Figure 4.8 depicts the result of employing our model by simply modifying the last softmax layer to accommodate 19 fine-grained classes instead of the 8 service categories. The overall accuracy of the model is 97.08%, as shown in Table 4.2. Despite having more classes, the accuracy of the model is higher than service-level classification, due to the raw bytes part of the model being extremely effective in fingerprinting specific servers. One side effect is that the different services from the same provider (e.g., Facebook video and Facebook social) have higher cross misclassifications, as evident in Figure 4.8.

W Avg precision (%)	W Avg recall (%)	W Avg F1-score (%)	Accuracy (%)
97.24	97.08	97.11	97.08

Table 4.2: Model performance in application-level classification

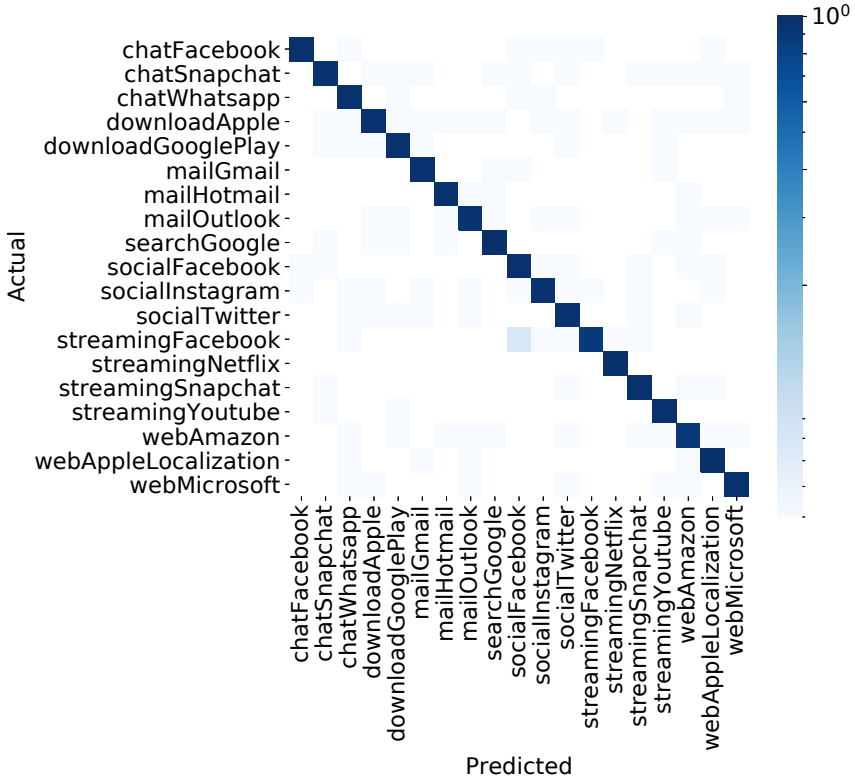


Figure 4.8: Application-level model accuracy

A key take-away from this experiment is that a good feature engineering approach for encrypted traffic classification can be adapted to different classification tasks, as it is good in capturing the "nature of traffic". It will be an interesting part of future research to leverage the feature engineering presented in Section 3.1 in areas other than service- and application-level classification, such as QoS classification and security.

4.6 QUIC Classification

We also evaluated our model on the UC Davis QUIC Dataset (cf. Section 4.2.2), which only includes the traffic shape time-series and not the actual network traces. In order to adapt to the dataset structure, we modified our approach by only activating the flow time-series part of the model and conducted the training for 20 epochs.

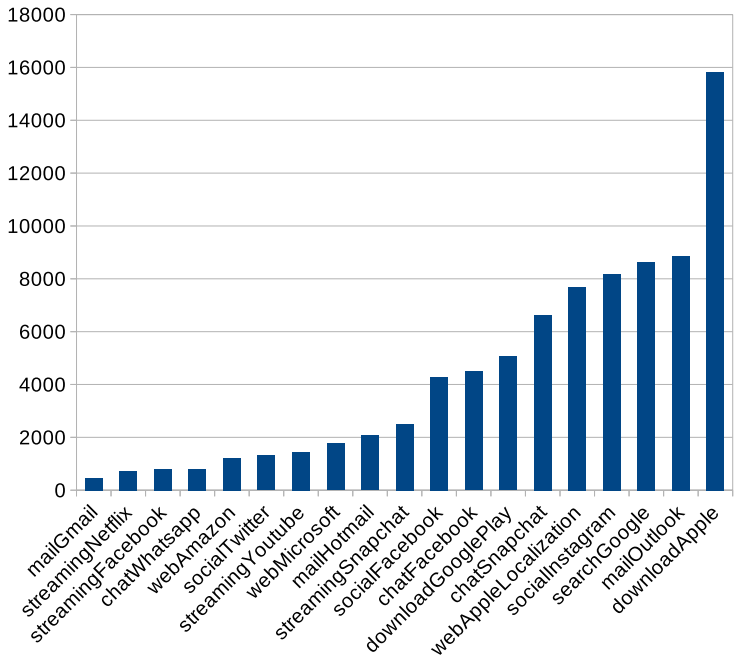
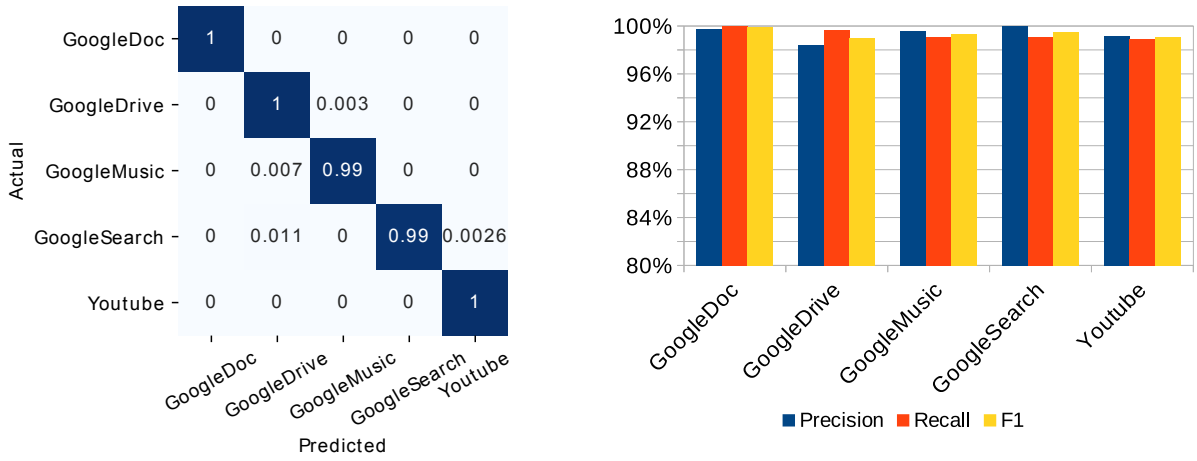


Figure 4.9: Label distribution

In evaluation, our model achieve a high validation accuracy (i.e., 99.37%), which is higher than the best one reported by the authors for their CNN model in [49] (i.e., ∼98%), regardless of whether their semi-supervised scheme (i.e., pre-training on unlabeled data first) is carried out or not. Figure 4.10a shows the result of the classification, which re-affirms that our proposed feature engineering is indeed a good indicator of the traffic class, and can adapt well to different encrypted web protocols. Our model achieves high accuracies despite the fact that QUIC is a more challenging protocol with a larger encrypted portion. These results also validate the utility of the Stacked LSTM architecture used in the flow time-series part of our model.



(a) Confusion matrix

(b) Per-class precision, recall, and F1-score

Figure 4.10: Model performance on the UCDavis QUIC traffic dataset

Chapter 5

Conclusion & Future Work

5.1 Conclusion

Traffic classification has become increasingly challenging with the widespread adoption of encryption in the Internet. Moreover, encrypted protocols are bound to evolve, rendering protocol-specific approaches futile in the future. In this thesis, we propose a DL approach for encrypted traffic classification that focuses on rather protocol-agnostic aspects of the encrypted web traffic as well. Our proposed feature set comprises of (i) a time-series of packet size, direction and inter-arrival times, (ii) flow statistics, and (iii) raw bytes from only the TLS handshake. We also propose a DL model architecture which is based on CNN and Stacked LSTM layers. We show that raw HTTPS traffic, apart from the TLS handshake part, does not contribute to the DL model's overall performance, but rather adds to its complexity and increases overfitting.

To ensure our DL model's robustness to future versions of TLS, we also obfuscate parts of the TLS handshake (e.g., SNI field and cipher info) that give away the server's identity. We argue that the scientific literature in TC often ignores what is actually being learned by the deep models which can result in expensive models being used for learning a simple logic, like that of a domain-name to label look-up table. Instead, we focus on the traffic shape and timing of packets, which show high potential in learning the complex nature of traffic among different classes. We show that our DL model generalizes for different classification objectives, i.e., service and application-level classification, and adapts to different encrypted web protocols (i.e., HTTP/2 and QUIC) by simply changing the training data. We evaluate our approach for service-level classification on a real-world mobile traffic dataset from an ISP, and show that by leveraging less raw traffic and a smaller number of parameters,

our model outclasses a state-of-the-art approach [48, 49]. To ensure reproducibility of our results, we have released the pre-processed dataset to the public (cf. Section 4.2.1).

In our evaluations, we compared our work to one of the prominent existing deep neural network methods. In our HTTP/2 dataset, our model achieves a 95.56% accuracy as compared to our competitor's 91.05% and the traditional C4.5 model's 81.39%, establishing a clear advantage against both of them with near 50% and 76% fewer false classifications respectively. We showed that our proposed Stacked LSTM architecture has a very good performance for processing traffic shape features. We also demonstrated that our method is applicable to the QUIC protocol, achieving 99% accuracy on the UCDavis QUIC dataset, as well as different classification granularity, which was shown in our application-level classification experiment with 97% overall accuracy. In the interest of transparency and openness in research, we have also publicly released our code-base as a Python library.

5.2 Future Work

Despite the surge in traffic classification publications since 2016, there are still many avenues for future research in this area which remain to be explored.

Evolution of ML Models: Some recent works [65, 64] have leveraged generative models for addressing the class imbalance problem in deep traffic classification. These data augmentation approaches can facilitate DL models and should be explored more in the future. Furthermore, Attention Networks [60] have gained popularity in the past few years, especially in the NLP literature. Recently, they have also been leveraged in traffic classification [71, 43]. Despite our initial experiments showing no real advantage in replacing LSTM layers in our architecture with Attention Networks, it is worth to further experiment with the combination of recurrent and convolutional layers with Attention Networks.

Transfer Learning: It is important to investigate domain adaptation and transfer learning [15], which are important yet often overlooked in traffic classification literature, and were not in the scope of our work. For instance, it will be interesting to study how well a model trained under certain network conditions (e.g., subnet, packet drop rate, latency and QoS level) would work under different conditions. Similarly, measuring the accuracy of a model through time and studying how well it translates to completely different sets of applications and servers should be explored in depth. Another important open problem is re-purposing models for different traffic classification tasks. For instance, a service classification model trained on millions of entries might be transferable to malware

detection using relatively few data points, by reusing the weights of its initial and middle layers.

Privacy and Security in ML: The robustness of traffic classification models against adversarial attacks and privacy leakage is of utmost importance and remains an open research question for future work. The effectiveness of black-box and white-box model inversion or data inference attacks is still an open question which will grow in importance over the next few years.

Federated Learning: Traffic classification generally suffers from a data scarcity problem (cf. Section 2.4), which is in large part due to privacy concerns. By leveraging federated and collaborative learning, multiple institutions can train a mutual model on all the (potentially privacy-critical) data available to them collectively, without actually sharing the data with one another. This is particularly interesting in use-cases related to cybersecurity. These efforts can be conducted in parallel to the former category, privacy in traffic classification, to study the privacy risk of such collaborative learning arrangements as well.

Detection of New Categories: Another aspect of TC as a ML problem is the *detection of new classes*. With the constantly evolving landscape of web services and applications, automatic detection of new emerging traffic categories can be very useful for network management. To achieve that, a wide range of strategies can be adopted. For instance, detection can be performed based on observing relatively large quantities of new data-points with low prediction confidence, or falling on decision boundaries of the model. Few Shot Learning (FSL) and meta-learning [57] methods can also be leveraged for updating TC models using small datasets, without retraining from scratch.

Self-supervised Learning: Labeling data is one of the primary challenges in almost all traffic classification problems. Evidently, the automated approximate labeling approach described in Section 3.4 has its limitations and might become increasingly difficult with new updates to the TLS protocol, such as the Encrypted SNI (ESNI) [22]. Thus, unsupervised and semi-supervised methods are crucial for the future of TC research. One possible avenue for research in this area is the application of Self-supervised Learning (SSL) to TC. The advantages of self-learning strategies have been demonstrated in representation learning and computer vision, especially in problems dealing with sequential data (e.g., [59]). The sequential nature of network traffic make these strategies a good candidate for addressing the data scarcity problem in TC.

References

[1] Université Toulouse 1. Blacklists UT1. http://dsi.ut-capitole.fr/blacklists/index_en.php, 2020. [Online; Accessed 01-October-2020].

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[3] Abbas Abou Daya, Mohammad A Salahuddin, Noura Limam, and Raouf Boutaba. Botchase: Graph-based bot detection using machine learning. *IEEE Transactions on Network and Service Management*, 17(1):15–29, 2020.

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

[5] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges. *IEEE Transactions on Network and Service Management*, 16(2):445–458, 2019.

[6] Iman Akbari. DeepTraffic. https://bit.ly/uw-orange-codebase, 2021. [Online; GitHub Repository].

[7] Iman Akbari, Mohammad A Salahuddin, Leni Ven, Noura Limam, Raouf Boutaba, Bertrand Mathieu, Stephanie Moteau, and Stephane Tuffin. A look behind the curtain: Traffic classification in an increasingly encrypted web. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(1):1–26, 2021.

[8] Riyad Alshammari and A Nur Zincir-Heywood. Machine learning based encrypted traffic classification: Identifying ssh and skype. In *IEEE symposium on computational intelligence for security and defense applications*, pages 1–8, 2009.

[9] Blake Anderson and David McGrew. Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1723–1732, 2017.

[10] Blake Anderson and David McGrew. Accurate tls fingerprinting using destination context and knowledge bases. *arXiv preprint arXiv:2009.01939*, 2020.

[11] Blake Anderson, Subharthi Paul, and David McGrew. Deciphering malware's use of tls (without decryption). *Springer Journal of Computer Virology and Hacking Techniques*, 14(3):195–211, 2018.

[12] M. Belshe, R. Peon, and M. Thomson. Hypertext transfer protocol version 2 (http/2). *RFC*, (7540), 2015.

[13] Mike Belshe and Roberto Peon. SPDY Protocol. Technical report, Network Working Group, Aug 2012.

[14] Mike Belshe, Roberto Peon, and Martin Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). IETF RFC 7540, IETF, May 2015.

[15] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2010.

[16] Oren Ben-Kiki, Clark Evans, and Brian Ingerson. Yaml ain't markup language (yaml™) version 1.1. *Working Draft 2008-05*, 11, 2009.

[17] David L. Black, Zheng Wang, Mark A. Carlson, Walter Weiss, Elwyn B. Davies, and Steven L. Blake. An Architecture for Differentiated Services. RFC 2475, December 1998.

[18] 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. *Springer Journal of Internet Services and Applications*, 9(1):16, 2018.

[19] Pierre-Olivier Brissaud, Jérôme Francçis, Isabelle Chrisment, Thibault Cholez, and Olivier Bettan. Transparent and service-agnostic monitoring of encrypted web traffic. *IEEE Transactions on Network and Service Management*, 16(3):842–856, 2019.

[20] Francesco Bronzino, Paul Schmitt, Sara Ayoubi, Guilherme Martins, Renata Teixeira, and Nick Feamster. Inferring streaming video quality from encrypted traffic: Practical models and deployment experience. *ACM on Measurement and Analysis of Computing Systems (SIGMETRICS)*, 3(3):1–25, 2019.

[21] Andreas Bruns, Andreas Kornstadt, and Dennis Wichmann. Web application tests with selenium. *IEEE software*, 26(5):88–91, 2009.

[22] Zimo Chai, Amirhossein Ghafari, and Amir Houmansadr. On the importance of encrypted-sni ({ESNI}) to censorship circumvention. In *9th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 19)*, 2019.

[23] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[24] 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 *IEEE International Conference on Big Data (Big Data)*, pages 1271–1276, 2017.

[25] Li Deng, Geoffrey Hinton, and Brian Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8599–8603. IEEE, 2013.

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

[27] Ran Dubin, Amit Dvir, Ofir Pele, and Ofer Hadar. I know what you saw last minute—encrypted http adaptive video streaming title classification. *IEEE transactions on information forensics and security*, 12(12):3039–3049, 2017.

[28] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015.

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

[30] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[31] Ramin Hasibi, Matin Shokri, and Mehdi Dehghan. Augmentation scheme for dealing with imbalanced network traffic classification using deep learning. *arXiv preprint arXiv:1901.00204*, 2019.

[32] Jonas Höchst, Lars Baumgärtner, Matthias Hollick, and Bernd Freisleben. Unsupervised traffic flow classification using a neural autoencoder. In *IEEE Conference on Local Computer Networks (LCN)*, pages 523–526, 2017.

[33] Auwal Sani Iliyasu and Huifang Deng. Semi-supervised encrypted traffic classification with deep convolutional generative adversarial networks. *IEEE Access*, 8:118–126, 2019.

[34] Christopher M Inacio and Brian Trammell. Yaf: yet another flowmeter. In *Proceedings of LISA10: 24th Large Installation System Administration Conference*, page 107, 2010.

[35] Jana Iyengar and Martin Thomson. Quic: A udp-based multiplexed and secure transport. *Internet Engineering Task Force, Internet-Draft*, 2018.

[36] Janardhan Iyengar and Ian Swett. QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2. Technical report, Network Working Group, Jun 2015.

[37] Li Jun, Zhang Shunyi, Lu Yanqing, and Zhang Zailong. Internet traffic classification using machine learning. In *2007 Second International Conference on Communications and Networking in China*, pages 239–243. IEEE, 2007.

[38] Pierre-Marie Junges, Jérôme François, and Olivier Festor. Passive inference of user actions through iot gateway encrypted traffic analysis. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 7–12. IEEE, 2019.

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

[40] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. Characterization of tor traffic using time based features. In *International Conference on Information Systems Security and Privacy (ICISSP)*, pages 253–262, 2017.

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

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

[43] Xun Liu, Junling You, Yulei Wu, Tong Li, Liangxiong Li, Zheyuan Zhang, and Jingguo Ge. Attention-based bidirectional gru networks for efficient https traffic classification. *Elsevier Information Sciences*, 541:297–315, 2020.

[44] 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.

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

[46] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[47] Jonathan Muehlstein, Yehonatan Zion, Maor Bahumi, Itay Kirshenboim, Ran Dubin, Amit Dvir, and Ofir Pele. Analyzing https encrypted traffic to identify user's operating system, browser and application. In *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–6. IEEE, 2017.

[48] Shahbaz Rezaei, Bryce Kroencke, and Xin Liu. Large-scale mobile app identification using deep learning. *IEEE Access*, 8:348–362, 2019.

[49] Shahbaz Rezaei and Xin Liu. How to achieve high classification accuracy with just a few labels: semi-supervised approach using sampled packets. *arXiv preprint arXiv:1812.09761*, 2018.

[50] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. *arXiv preprint arXiv:1708.06376*, 2017.

[51] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[52] Jason Sanders and Edward Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[53] Dominik Schatzmann, Wolfgang Mühlbauer, Thrasyvoulos Spyropoulos, and Xenofontas Dimitropoulos. Digging into https: Flow-based classification of webmail traffic. In *10th ACM SIGCOMM Conference on Internet Measurement*, pages 322–327, 2010.

[54] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the burst: Remote identification of encrypted video streams. In *USENIX Security Symposium (USENIX Security 17)*, pages 1357–1374, 2017.

[55] Yan Shi, Dezhi Feng, and Subir Biswas. A natural language-inspired multi-label video streaming traffic classification method based on deep neural networks. *arXiv preprint arXiv:1906.02679*, 2019.

[56] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *computers & security*, 31(3):357–374, 2012.

[57] Jake Snell, Kevin Swersky, and Richard S Zemel. Prototypical networks for few-shot learning. *arXiv preprint arXiv:1703.05175*, 2017.

[58] Van Tong, Hai Anh Tran, Sami Souihi, and Abdelhamid Mellouk. A novel quic traffic classifier based on convolutional neural networks. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2018.

[59] Hsiao-Yu Fish Tung, Hsiao-Wei Tung, Ersin Yumer, and Katerina Fragkiadaki. Self-supervised learning of motion capture. *arXiv preprint arXiv:1712.01337*, 2017.

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

[61] 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.

[62] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

[63] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

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

[65] Pan Wang, Shuhang Li, Feng Ye, Zixuan Wang, and Moxuan Zhang. Packetcgan: Exploratory study of class imbalance for encrypted traffic classification using cgan. In *IEEE International Conference on Communications (ICC)*, pages 1–7, 2020.

[66] 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.

[67] 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, 2018.

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

[69] Nigel Williams, Sebastian Zander, and Grenville Armitage. A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. *ACM SIGCOMM Computer Communication Review*, 36(5):5–16, 2006.

[70] Haipeng Yao, Pengcheng Gao, Jingjing Wang, Peiying Zhang, Chunxiao Jiang, and Zhu Han. Capsule network assisted iot traffic classification mechanism for smart cities. *IEEE Internet of Things Journal*, 6(5):7515–7525, 2019.

[71] Haipeng Yao, Chong Liu, Peiying Zhang, Sheng Wu, Chunxiao Jiang, and Shui Yu. Identification of encrypted traffic through attention mechanism based long short term memory. *IEEE Transactions on Big Data*, 2019.

[72] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75, 2018.

[73] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

[74] Zhuang Zou, Jingguo Ge, Hongbo Zheng, Yulei Wu, Chunjing Han, and Zhongjiang Yao. Encrypted traffic classification with a convolutional long short-term memory neural network. In *IEEE International Conference on High Performance Computing and Communications; IEEE International Conference on Smart City; IEEE International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 329–334, 2018.

APPENDICES

.1 Performance Metrics

Each prediction from a binary classifier model can be either a true-positive (TP), false-positive (FP), true-negative (TN), or false-negative (FN). When evaluating a model's performance, the count of each of these four metrics, can help calculate other useful metrics.

Precision denotes what percentage of positive instances from the model, are correct. It is defined as follows:

$$precision = \frac{TP}{TP + FP} \times 100\% \tag{1}$$

Recall indicates what percentage of the instances that have a positive label in ground truth, have indeed been classified as positive by the model:

$$recall = \frac{TP}{TP + FN} \times 100\% \tag{2}$$

However, these two metrics by themselves can not be used to evaluate a model. For instance, a model that always gives a positive prediction will have a perfect recall (i.e., $recall = 100\%$). Similarly, a model that always gives negative predictions, would never have a false-positive and will have a perfect precision (i.e., $precision = 100\%$)[1]. That is why, F1-score (or F1) is defined as a combination of the two metrics:

$$F1 - score = 2 \times \frac{precision \times recall}{precision + recall} \times 100\% \tag{3}$$

[1]Although in this case $p = \frac{0}{0}$ and thus precision is undefined, the idea is that getting near-perfect precision is relatively easy as the model can only give positives when extremely certain about it and hence $\lim_{FP \to 0} p = 1$.

This is called a "harmonic mean" of precision and recall. It is more useful than the arithmetic mean, since if either metric falls to zero, so would the F1-score.

Accuracy is a more familiar metric, which essentially indicates what percentage of all predictions are correct:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% \tag{4}$$

In our evaluations, we use weighted average recall, precision, and F1-score of the models, which is an average of those metrics over all classes and weighted by the class sizes (i.e., number of entries with each class label in the dataset). This is denoted by "W Avg" precision, recall, and F1-score throughout the paper.

.2 Model Details

Table 1: Architecture of the tripartite model with convolutions in the flow side. By default a layer connects to its predecessor.

Type	Shape	Connection
In	$(3, 600)$	Header Input
Reshape	$(1800, 1)$	
Convolution1D	$(1799, 256)$	
ReLU		
Convolution1D	$(1799, 256)$	
ReLU		
MaxPooling1D	$(899, 256)$	
Convolution1D	$(898, 128)$	
ReLU		
Convolution1D	$(897, 128)$	
ReLU		
MaxPooling1D	$(448, 128)$	
Flatten	(57344)	
In	(61)	Flow Meter Input
Dense	(200)	
BatchNorm		
LeakyReLU		
Dropout(0.2)		
Dense	(200)	
BatchNorm		
LeakyReLU		
Dropout(0.5)		
In	$(1024, 3)$	Flow Input
Convolution1D	$(1024, 128)$	
BatchNorm		
ELU		
Convolution1D	$(1024, 128)$	
BatchNorm		
ELU		
MaxPooling1D	$(512, 128)$	
Convolution1D	$(512, 64)$	
BatchNorm		
ELU		
Convolution1D	$(512, 64)$	
BatchNorm		
ELU		
MaxPooling1D	$(256, 64)$	
Flatten	(16384)	
Concatenate	(73928)	
Dense	(128)	
LeakyReLU		
Dropout(0.5)		
Dense	(128)	
LeakyReLU		
Dropout(0.5)		
Dense	(8)	
Softmax		

Table 2: Architecture of the tripartite model with LSTM's in the flow side. By default a layer connects to its predecessor.

Type	Shape	Connection
In	$(3, 600)$	Header Input
Reshape	$(1800, 1)$	
Convolution1D	$(1799, 256)$	
ReLU		
Convolution1D	$(1799, 256)$	
ReLU		
MaxPooling1D	$(899, 256)$	
Convolution1D	$(898, 128)$	
ReLU		
Convolution1D	$(897, 128)$	
ReLU		
MaxPooling1D	$(448, 128)$	
Flatten	(57344)	
In	(61)	Flow Meter Input
Dense	(200)	
BatchNorm		
LeakyReLU		
Dropout(0.2)		
Dense	(200)	
BatchNorm		
LeakyReLU		
Dropout(0.5)		
In	$(1024, 3)$	Flow Input
Dense(Distributed)	$(1024, 512)$	
BatchNorm		
LeakyReLU		
LSTM(Forward)	$(1024, 256)$	
LSTM(Backward)	$(1024, 256)$	
LSTM(Forward)	(256)	
Dropout(0.5)		
Concatenate	(57800)	
Dense	(128)	
LeakyReLU		
Dropout(0.5)		
Dense	(128)	
LeakyReLU		
Dropout(0.5)		
Dense	(8)	
Softmax		

.3 Flow-based statistics from CICFlowMeter

- Duration of the flow in Microsecond
- Duration of the flow in Microsecond
- Total packets in the forward direction
- Total packets in the backward direction
- Total size of packet in forward direction
- Total size of packet in backward direction
- Minimum size of packet in forward direction
- Maximum size of packet in forward direction
- Mean size of packet in forward direction
- Standard deviation size of packet in forward direction
- Minimum size of packet in backward direction
- Maximum size of packet in backward direction
- Mean size of packet in backward direction
- Standard deviation size of packet in backward direction
- Number of flow packets per second
- Number of flow bytes per second
- Mean time between two packets sent in the flow
- Standard deviation time between two packets sent in the flow
- Maximum time between two packets sent in the flow
- Minimum time between two packets sent in the flow
- Minimum time between two packets sent in the forward direction
- Maximum time between two packets sent in the forward direction
- Mean time between two packets sent in the forward direction
- Standard deviation time between two packets sent in the forward direction
- Total time between two packets sent in the forward direction
- Minimum time between two packets sent in the backward direction
- Maximum time between two packets sent in the backward direction
- Mean time between two packets sent in the backward direction
- Standard deviation time between two packets sent in the backward direction
- Total time between two packets sent in the backward direction
- Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
- Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)

- Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
- Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
- Total bytes used for headers in the forward direction
- Total bytes used for headers in the backward direction
- Number of forward packets per second
- Number of backward packets per second
- Minimum length of a packet
- Maximum length of a packet
- Mean length of a packet
- Standard deviation length of a packet
- Variance length of a packet
- Number of packets with FIN
- Number of packets with SYN
- Number of packets with RST
- Number of packets with PUSH
- Number of packets with ACK
- Number of packets with URG
- Number of packets with CWE
- Number of packets with ECE
- Download and upload ratio
- Average size of packet
- Average size observed in the forward direction
- Average number of bytes bulk rate in the forward direction
- Length of header for forward packet
- Average number of bytes bulk rate in the forward direction
- Average number of packets bulk rate in the forward direction
- Average number of bulk rate in the forward direction
- Average number of bytes bulk rate in the backward direction
- Average number of packets bulk rate in the backward direction
- Average number of bulk rate in the backward direction
- The average number of packets in a sub flow in the forward direction
- The average number of bytes in a sub flow in the forward direction
- The average number of packets in a sub flow in the backward direction
- The average number of bytes in a sub flow in the backward direction
- The total number of bytes sent in initial window in the forward direction
- The total number of bytes sent in initial window in the backward direction
- Count of packets with at least 1 byte of TCP data payload in the forward direction

- Minimum segment size observed in the forward direction
- Minimum time a flow was active before becoming idle
- Mean time a flow was active before becoming idle
- Maximum time a flow was active before becoming idle
- Standard deviation time a flow was active before becoming idle
- Minimum time a flow was idle before becoming active
- Mean time a flow was idle before becoming active
- Maximum time a flow was idle before becoming active
- Standard deviation time a flow was idle before becoming active
- Total packets in the forward direction
- Total packets in the backward direction
- Total size of packet in forward direction
- Total size of packet in backward direction
- Minimum size of packet in forward direction
- Minimum size of packet in backward direction
- Maximum size of packet in forward direction
- Maximum size of packet in backward direction
- Mean size of packet in forward direction
- Mean size of packet in backward direction
- Standard deviation size of packet in forward direction
- Standard deviation size of packet in backward direction
- Total time between two packets sent in the forward direction
- Total time between two packets sent in the backward direction
- Minimum time between two packets sent in the forward direction
- Minimum time between two packets sent in the backward direction
- Maximum time between two packets sent in the forward direction
- Maximum time between two packets sent in the backward direction
- Mean time between two packets sent in the forward direction
- Mean time between two packets sent in the backward direction
- Standard deviation time between two packets sent in the forward direction
- Standard deviation time between two packets sent in the backward direction
- Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
- Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)

- Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
- Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
- Total bytes used for headers in the forward direction
- Total bytes used for headers in the backward direction
- Number of forward packets per second
- Number of backward packets per second
- Number of flow packets per second
- Number of flow bytes per second
- Minimum length of a flow
- Maximum length of a flow
- Mean length of a flow
- Standard deviation length of a flow
- Minimum inter-arrival time of packet
- Maximum inter-arrival time of packet
- Mean inter-arrival time of packet
- Standard deviation inter-arrival time of packet
- Number of packets with FIN
- Number of packets with SYN
- Number of packets with RST
- Number of packets with PUSH
- Number of packets with ACK
- Number of packets with URG
- Number of packets with CWE
- Number of packets with ECE
- Download and upload ratio
- Average size of packet
- Average size observed in the forward direction
- Average number of bytes bulk rate in the forward direction
- Average number of packets bulk rate in the forward direction
- Average number of bulk rate in the forward direction
- Average size observed in the backward direction
- Average number of bytes bulk rate in the backward direction
- Average number of packets bulk rate in the backward direction
- Average number of bulk rate in the backward direction
- The average number of packets in a sub flow in the forward direction
- The average number of bytes in a sub flow in the forward direction
- The average number of packets in a sub flow in the backward direction
- The average number of bytes in a sub flow in the backward direction

- Minimum time a flow was active before becoming idle
- Mean time a flow was active before becoming idle
- Maximum time a flow was active before becoming idle
- Standard deviation time a flow was active before becoming idle
- Minimum time a flow was idle before becoming active
- Mean time a flow was idle before becoming active
- Maximum time a flow was idle before becoming active
- Standard deviation time a flow was idle before becoming active
- The total number of bytes sent in initial window in the forward direction
- The total number of bytes sent in initial window in the backward direction
- Count of packets with at least 1 byte of TCP data payload in the forward direction
- Minimum segment size observed in the forward direction

.4 Acronyms

Acronym	Meaning
AES	Advanced Encryption Standard
ALPN	Application-Layer Protocol Negotiation
CNN	Convolutional Neural Network
DASH	Dynamic Adaptive Streaming over HTTP
DL	Deep Learning
DNS	Domain Name System
DT	Decision Tree
GRU	Gated Recurrent Unit
HTTPS	Hypertext Transfer Protocol Secure
IAT	Inter-arrival Time
IDS	Intrusion Detection System
ISP	Internet Service Provider
LSTM	Long Short-term Memory
ML	Machine Learning
MLP	Multi-layer Perceptron
MTU	Maximum Transmission Unit
NAT	Network address translation
NB	Naïve Bayes
NDA	Non-disclosure Agreement
NLP	Natural Language Processing
PCAP	Packet Capture
QUIC	Quick UDP Internet Connections
QoE	Quality of Experience
QoS	Quality of Service
RF	Random Forest
SAE	Stacked Auto-encoder
SMTP	Simple Mail Transfer Protocol
SNI	Server Name Indication
STD	Standard Deviation
SVM	Support Vector Machine
TC	Traffic Classification
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
VPN	Virtual Private Network
VoIP	Voice over Internet Protocol

.5 DeepTraffic Software API Code Examples

The software API developed based on our work in this thesis is called DeepTraffic, the logical structure of which is described in 3.7. The full documentations of the code base are available in its Git repository. [6] Here, we provide some code examples of how the library can be used for pre-processing network traces, training models and using trained models for volumetry.

Our library's implementation is based on Tensorflow 2.1, CUDA 10.1, TensorRT 6.0.1.5, and CuDNN 7.6.5. DeepTraffic uses a number of dependencies for which we have simplified its environment's set-up on Ubuntu 18.04 LTS and later by developing automated installation scripts. Listing 1 shows how these scripts are used.

```
# Installs YAF and dependencies
./install-deps.sh

# Installs Python distribution used for Deep Traffic
./install-py.sh

# Download the data used for automated labeling
./install-data.sh

# activate environment and set required environment variables
. activate-env.sh
```

Listing 1: Set-up the environment for DeepTraffic

.5.1 Pre-processing

The pre-processing takes packet traces (PCAP files) as input, and stores the result of the steps described in Section 3.5 in a special folder structure with binary files in a given output directory.

The entire pipeline can be executed conveniently using a single command as shown in Listing 2.

```
# Pre-process the PCAP file at input location and store the resulting
# directories in the output path
./pcap_to_dataset.sh /path/to/pcap-file -o /path/to/output-dir
```

Listing 2: Converting an input PCAP file to pre-processed dataset

.5.2 Usage

The core functionalities of DeepTraffic are implemented in its Python library, which is able to perform all the complex tasks involved with loading the data, performing training on GPU or CPUs, analyzing the results, loading and storing models, etc. with only a few lines of code. It is also designed to work with zero configurations as well.

Listing 3 shows a full ML pipeline in Python, starting from the point of loading the dataset stored by the command (cf. Listing 2), which evidently is the first step of any data pipeline in DeepTraffic. A model and a *Training* object (cf. Figure 3.5) are then instantiated and the latter is used to conduct the training. The model is then validated with visual plots and is used to perform volumetry on the same dataset.

```
import deeptraffic.train
import deeptraffic.model
import deeptraffic.dataset

# load data-set with service-level labeling
dataset = deeptraffic.dataset.DataSet("/path/to/dataset", labeling="service")

# create the model described in Section 3.3 with all three feature categories
model = deeptraffic.model.Sigmetrics2021ModelSpec(dataset, features="fha")

# train the model
training = deeptraffic.train.Training(model, dataset)
training.train_model("/path/to/output.model")

# model validation (with PyPlot visualization of confusion matrix)
training.validate(plot=True)

```

```
# volumetry on same dataset (with PyPlot visualization)
training.volumetry(dataset=dataset, plot=True)
```

Listing 3: A full ML pipeline from loading the dataset to computing the volumetry in DeepTraffic

Configuration

DeepTraffic is designed with simplicity and convenience in mind. However, the fine-grained details of the model's training and usage can all be modified and tuned according to the user's need, if necessary. The configurations are done through a YAML file called `conf.yml`. Table 3 shows the available configurations.

Configuration Name	Default	Description
`cores`	64	number of Spark executors
`spark.driver.memory`	8g	Spark driver memory
`spark.executor.memory`	4g	Spark executors memory
`preproc.cfm.replace_nan`	0	whether should replace NaN in CFM results and if so, with what value
`preproc.cfm.replace_pos_inf`	2	whether should replace +inf in CFM results and if so, with what value
`preproc.cfm.replace_neg_inf`	-2	whether should replace -inf in CFM results and if so, with what value
`model.packet_cutoff`	600	cut-off threshold for TLS headers in bytes
`model.max_flow_size`	1024	maximum flow packets included in training
`model.num_headers`	3	maximum handshake packets included in training
`model.activation`	relu	neuron activation function
`model.sigmetrics.header_channel_size`	256	no. channels in each layer of the model in the side processing raw traffic

model.sigmetrics.flow_channel_size	512	no. channels in each layer of the model in the side processing traffic shape time-series
model.sigmetrics.aux_channel_size	200	no. channels in each layer of the model in the side processing statistical features
model.sigmetrics.dense_channel_size	128	no. channels in each layer of the model in the part after concatenation of the three parts (fully-connected layers)
model.sigmetrics.recurrent_units	256	no. channels in the model's SLSTM part
training.validation_set_ratio	5	ratio of validation set to training set
training.models_dir	./Models	directory for storing model weights backup at checkpoints
training.epochs	10	how many epochs to train for
training.learning_rate_checkpoint	3	how many epochs between decimations of the learning rate
training.learning_rate_discount	0.1	co-efficient for discounting learning-rate at checkpoints
training.initial_learning_rate	0.001	initial learning rate in training
labeling.domains_path		path to directory of domains data used for labeling relative to the execution path

Table 3: The available configurations for the DeepTraffic library, their default value and their functionality's description.

.6 Confidence

In the absence of labels for input data, the typical metrics, such as accuracy and F1-score, can not be calculated for evaluation. However, there are other metrics that can help provide insights into the performance of the model. One such metric is how "confident" the model is in its predictions, on average. Classification models typically calculate a probability distribution over the available labels. For each input, the label with the highest probability, as evaluated by the model, will be chosen as the model's prediction. The value of that probability can be thought of as the model's *confidence* in the prediction i.e., for inputs that are quite similar to the examples seen by the model in training this probability will be higher, and for the inputs that "confuse" the model the probability will be lower.

More formally, confidence is the average of the maximum value in model's output layer. When evaluating a model f_θ on a set of sample inputs called S, confidence can be defined as follows:

$$\text{confidence} = \frac{1}{|S|} \sum_{x \in S} \max f_\theta(x), \tag{5}$$

assuming $f_\theta(u) \in \mathbb{R}^{|L|}$ returns a probability distribution over the labels set L for each input (i.e., $\forall u \in S : \sum_{1 \leq i \leq |L|} f(u) \cdot e_i = 1$)[2]

[2] e_i is the conventional linear algebra notation for the i^{th} standard basis (*a.k.a.* natural basis)