ANALYTICS OVER ENCRYPTED TRAFFIC AND DEFENSES

by

Thilini N Dahanayaka

A thesis in fulfilment of the requirements for the degree of
Doctor of Philosophy

THE UNIVERSITY OF SYDNEY

School of Computer Science
Faculty of Engineering

January 2023

Declaration of Authorship

I, Thilini Dahanayaka, declare that this thesis titled 'Analytics over Encrypted Traffic and Defenses' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Abstract

UNIVERSITY OF SYDNEY

Faculty of Engineering
School of Computer Science

Doctor of Philosophy

by Thilini N Dahanayaka

Encrypted traffic flows have been known to leak information about their underlying content through statistical properties such as packet lengths and timing. While traffic fingerprinting attacks exploit such information leaks and threaten user privacy by disclosing website visits, videos streamed, and user activity on messaging platforms, they can also be helpful in network management and intelligence services. Most recent and best-performing such attacks are based on deep learning models. In this thesis, we identify multiple limitations in the currently available attacks and defenses against them. First, these deep learning models do not provide any insights into their decision-making process. Second, most attacks that have achieved very high accuracies are still limited by unrealistic assumptions that affect their practicality. For example, most attacks assume a closed world setting and focus on traffic classification after event completion. Finally, current state-of-the-art defenses still incur high overheads to provide reasonable privacy, which limits their applicability in real-world applications.

In order to address these limitations, we first propose an inline traffic fingerprinting attack based on variable-length sequence modeling to facilitate real-time analytics. Next, we attempt to understand the inner workings of deep learning-based attacks with the dual goals of further improving attacks and designing efficient defenses against such attacks. Then, based on the observations from this analysis, we propose two novel defenses against traffic fingerprinting attacks that provide privacy under more realistic constraints and at lower bandwidth overheads. Finally, we propose a robust framework for open set classification that targets network traffic with this added advantage of being more suitable for deployment in resource-constrained in-network devices.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor Dr. Suranga Seneviratne for giving me the opportunity to pursue my Ph.D. under his supervision and for his immense support and guidance throughout the entirety of my Ph.D. candidature. I am incredibly grateful to him for always being available to discuss challenges and taking the time to provide me with timely feedback, finding necessary resources, connecting me with other researchers, and recommending research and other opportunities that helped build up my research career.

I extend my sincere gratitude to everyone who worked with me during my Ph.D. candidature. My special thank goes to Dr. Guillaume Jourjon for his invaluable support and guidance during research collaborations as well as the thesis preparation. I am also grateful to Prof. Aruna Seneviratne, Dr. Kanchana Thilakarathna, and Dr. Harini Kolamunna for their support and guidance during my candidature and research collaborations.

I would also like to thank my colleagues Jiawei Zhao, Chamara Madarasingha, Kwon Nung Choi, Alex Vaskevich, Yi Huang, Yasod Ginige, David Kennedy, and Zhiyi Wang for their support and contributions in joint research work.

Last but not least, I would like to thank my family, my parents, grandparents, and my sister, for their unconditional support and encouragement toward the completion of my Ph.D. I would also like to thank Mrs. Yasawathie Gurusinghe for her immense support since the beginning of my Ph.D. degree.

Contents

List of Figures

List of Tables

Abbreviations

CNN	Convolutional Neural Network
CPU	Central Processing Unit
DASH	Dynamic Adaptive Streaming over HTTP
DNS	Domain Name Sydtem
DoH	DNS over HTTPS
DoT	DNS over TLS
DPI	Deep Packet Inspection
E2EE	End to End Encryption
GAN	Generative Adversarial Networks
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IoT	Internet of Things
IP	Internet Protocol
ISP	Internet Service Provider
LSTM	Long Short Term Memory
MAC	Media Access Control
ML	Machine Learning
MLP	Multi Layer Perceptron
NAT	Network Address Translation
OS	Operating System
RFC	Request For Comments
RNN	Recurrent Neural Network
SDAE	Stacked Denoising Autoencoder
SSH	Secure Shell
SSL	Secure Sockets Layer

SVM	Suppport Vector Machine
TA	Traffic Analysis
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VoIP	Voice over IP
WF	Website Fingerprinting
WWW	World Wide Web

List of Publications

Conferences

1. **T. Dahanayaka**, G. Jourjon, S. Seneviratne, "Understanding traffic fingerprinting cnns", In Proceedings of IEEE 45th Conference on Local Computer Networks (LCN) 2020.

2. A. Vaskevich, **T. Dahanayaka**, G. Jourjon and S. Seneviratne, "CGAN-based Network Traffic Morphing as a Defense Against Video Streaming Fingerprinting," 2021 IEEE 20th International Symposium on Network Computing and Applications (NCA), 2021.

3. **T. Dahanayaka**, Z. Wang, G. Jourjon, S. Seneviratne, "Inline Traffic Analysis Attacks on DNS over HTTPs", Accepted to IEEE 47th Conference on Local Computer Networks (LCN) 2022.

4. **T. Dahanayaka**, Y. Ginige, Y Huang, G. Jourjon, S. Seneviratne, "Robust Open Set Classification for Encrypted Traffic Fingerprinting", Currently under review for The ACM ASIACCS 2023.

Journals

5. H. Kolamunna, **T. Dahanayaka**, J. Li, S. Seneviratne, K. Thilakarathna, A. Zomaya, and A. Seneviratne, "DronePrint: Acoustic Signatures for Open-Set Drone Detection and Identification with Online Data", In Proceedings of ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT) 2021.

6. **T. Dahanayaka**, G. Jourjon, S. Seneviratne, "Dissecting Traffic Fingerprinting CNNs with Filter Activations", The International Journal of Computer and Telecommunications Networking 2022.

Posters

7. K. N. Choi, **T. Dahanayaka**, D. Kennedy, K. Thilakarathna, S. Seneviratne, S. Kanhere, P. Mohapatra, "Poster Abstract: Passive Activity Classification of Smart Homes through Wireless Packet Sniffing", In Proceedings of The International Conference on Information Processing in Sensor Networks (IPSN) 2020.

8. H. Kolamunna, J. Li, **T. Dahanayaka**, S. Seneviratne, K. Thilakarathna, A. Zomaya, and A. Seneviratne, "Poster: AcousticPrint: Acoustic Signature based Open Set Drone Identification", In Proceedings of 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec) 2020.

Publications without Proceedings

9. "What exactly do traffic fingerprinting CNNs See? Improved defenses against traffic fingerprinting", In N2Women Workshop 2021 at ACM SIGCOMM'21.

10. 'DronePrint: Acoustic Signatures for Open-set Drone Detection and Identification with Online Data', In ADSTAR Summit 2022.

Authorship Attribution Statement

We present the authorship attribution for the published works from the previous section (List of Publications) included in the thesis chapters.

- Chapter 3 of this thesis is published as [3] in the Publications List.
  In [3], I performed the initial data analysis, designed the attack models, carried out the experiments and wrote the drafts of the conference paper.

- Chapter 4 of this thesis comprises content published as [1] and [6] in the Publications List.
  In both publications, I designed the experiments, carried out the experiments and drafted the conference/ journal paper. The chapter includes all content from [1] and Sections 1-6 from [6].

- Chapter 5 of this thesis comprises content published as [6] in the Publications List.
  I designed the defenses, carried out the experiments and drafted the journal paper from which Section 7 is included in this chapter.

- Chapter 6 of this thesis is published as [4] in the Publications List.
  I designed the framework, carried out all the experiments except for the experiments related to quantization and drafted the journal paper.

I certify that the aforementioned authorship attribution statements are correct, and I have received permission from the other authors to include the published materials. Also, being the lead author, by convention in my research field, I have made the major contribution to the publications. Also, I am the corresponding author for the publications used in this thesis.

Thilini N Dahanayaka: Signature: Date:

As the supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

Dr Suranga Seneviratne: Signature: Date:

Chapter 1

Introduction

The Internet is a network of networks that interconnects computer networks operated by public, private, academic, business and government entities all over the world, encompassing a wide range of services including the *World Wide Web (WWW)*, email and file sharing applications. Initially developed in the 1960s by the United States Department of Defense to enable time sharing between their computers, the Internet has been growing fast, surpassing the expectations of the original designers of the underlying Internet architecture. With the introduction of WWW in 1994, the Internet began its exponential growth and has now become the key component of all communications worldwide and a vital part of our day-to-day lives. By July 2021, the number of Internet users worldwide reached 4.9 billion [1], which implies that by then, over two-thirds of the global population was already connected to the Internet. With the contribution of the exponentially increasing number of mobile device users and Internet of Things (IoT) device deployments, and the expansion of social media usage, we can expect the Internet to continue expanding at a rapid rate.

With the growth in variety and complexity of Internet traffic, service providers and network administrators faced an increasing need to understand the type of traffic (or the applications that generated the traffic) in their networks, which is known as *Traffic Classification*. Thus traffic classification facilitates optimizing network engineering, including application-based service differentiation and content-sensitive pricing, and network management tasks such as traffic shaping, policy routing, and packet filtering. Furthermore, traffic classification is also essential for detecting new forms of malicious traffic patterns

which threaten legitimate services on network links. Finally, in contrast to entities aiming for mere network engineering and management tasks, traffic classification can be exploited by intelligence services to monitor and censor the Internet activities of the general public, which raises major privacy concerns.

1.1 Traditional Methods of Traffic Classification

Traffic classification has been studied and carried out since the early 90s. The most straightforward and therefore, most frequently used method of traffic classification used to use transport layer (TCP and UDP) port numbers to map traffic to applications that generated them. This method was based on the intuition that specific application types are bound to specific port numbers and gave reasonably accurate results in the past. However, with time, applications began using unpredictable ports because some new applications have no IANA (Internet Assigned Numbers Authority) registered ports and hence use already registered (to other applications), randomly selected, or user-defined ports. In some cases, applications use unpredictable ports with the specific intention of bypassing traffic classification. As a result, the traditional method of using port numbers gradually became ineffective. Another approach for traffic classification was to use the destination IP addresses. For example, if the destination IP address of a packet is a well-known IP of a video content provider like *Youtube*, it can be inferred that the network packet corresponds to video streaming. However, due to deployment of Network Address Translations (NAT) caused by IPv4 address exhaustion, this approach is also no longer effective.

As a result, researchers focused more on payload-based methods such as *Deep Packet Inspection (DPI)*, which inspects packet content to identify the associated application or perform more complicated analysis tasks. Compared to using transport layer port numbers, DPI poses significantly higher privacy risks as the actual payload/content of network packets which are mostly transmitted in plaintext, are exposed to third parties. For example, sensitive information that is transmitted via the Internet, including banking details, health information, and other financial details, could fall into the hands of adversaries who can perform DPI with minimum resources, such as simple packet sniffing tools. In response to growing privacy concerns of general Internet users, *End-to-End Encryption (E2EE)* which ensures that only the sender and the intended receiver

with keys can access the plain text, was introduced to ensure the confidentiality of Internet communications. E2EE was first introduced with SSL v1.0 (*Secure Socket Layer*) and went through various improvements until it reached TLS v1.3 (*Transport Layer Security*) which is currently in use (Figure 1.3). By mid-2021, more than 80% of global Internet traffic was reported to be end-to-end encrypted [2]. Since DPI depends on being able to access payload content, the increased deployment of E2EE rendered DPI less usable. As a result, researchers faced a new challenge to devise novel techniques to understand traffic types on networks using encrypted traffic.

On the other hand, *side-channel leaks* and related attacks have been studied for decades, with documented attacks dating back to 1943 [3]. Side-channel information leaks can be discussed under many contexts, such as information leaks through electromagnetic signals, shared memory/registers/files between processes and CPU usage metrics, and have been used in a wide range of tasks, including breaking cryptographic systems [4] or inferring keystrokes in SSH [5]. However, in this work, we use the term *side-channel leaks* in the context of encrypted communications only, where such information can be leveraged by a network eavesdropper to infer valuable information from encrypted traffic. To understand side-channel information leaks, we first pay attention to Figure 1.1.

FIGURE 1.1: Gift wrapped bike [6]

Figure 1.1 illustrates a gift, well wrapped using opaque wrapping paper covering the entire object. However, regardless of the fact that the entire object is covered by wrapping paper, anyone looking at it can easily guess that the gift is a bicycle. The reason is that the crucial information about the object, such as its specific shape and size that can hint at the object inside, is still visible. The same scenario can be seen with encrypted traffic as well. Even though encryption 'hides' the plaintext content of network packets by converting them to unintelligible ciphertexts, features such as packet lengths and timing of

the encrypted packets can reveal information about its underlying content. More specifically, side-channel information of encrypted traffic flows, such as packet lengths and timings, can reveal valuable information about encrypted traffic. Attacks that leverage side-channel information for traffic classification over end-to-end encrypted traffic have been used since the late 1990s [7].

Since the vast majority of current Internet traffic is end-to-end encrypted, and the industry can be seen to be moving towards 100% deployment of E2EE for most communications, from this point onward, this work will only cover traffic classification over encrypted traffic using side-channel information leaks. Furthermore, as side-channel information can be used to infer various types of information from different types of network traffic types, and not only for mere traffic classification, in the subsequent sections, we will refer to attacks that leverage side-channel leaks of encrypted traffic as ***Traffic Analysis (TA)*** attacks.

1.2 Traffic Analysis Attacks

We define the term *Traffic Analysis attack* as the scenario where an attacker passively eavesdrops on an encrypted channel and makes inferences about the user's Internet activities without any decryption. An example scenario of such an attack is illustrated in Figure 1.2. Here we first consider **Attacker 1** i.e., the network administration of a local network of an institution (Company K) where users connect to the Internet through the institutional network. Here, **Attacker 1** is interested in knowing whether its users visit restricted websites. First, the attacker will identify a set of websites frequently visited by its users (including restricted websites) and capture his own network traffic (encrypted) when visiting each of the websites in the identified list. Next, the attacker will pre-process (normalizing and feature extraction) the data and use the processed network traces labeled with the corresponding website to train a machine learning-based traffic classifier to label the network traces correctly. Then, the attacker can passively capture network traffic from its users, pre-process and feed the processed network traces to the trained traffic classifier to correctly identify the website visited by the user. If the user visits one of the restricted websites, the administration can take appropriate action and otherwise, allow the user to browse uninterrupted.

It should be noted that the above scenario is just one example and that the point of eavesdropping and the purpose of the attack could be varied. For example, an attacker who is interested in a specific user in Company K can either eavesdrop on the network layer (same as **Attacker 1** if he is someone with access, such as a network admin) or simply do wifi packet sniffing. A similar attack can be carried out on an intermediate link as well. Even though such an attacker would find it difficult to target a single user, the attacker (ISP-level) can still target an enterprise network or a particular geographical area. For example, **Attacker 2** in Figure 1.2, an intelligence service agent eavesdropping on the network link outside Company K can devise a similar attack as the local admin of Company K and will be able to map specific website visits with the originating enterprise network. Accordingly, in the example scenario, intelligence services can identify that traffic originating from Company Q is harmless, while traffic from Company K includes a connection to a blacklisted website. Moreover, the target of the TA attack can be varied too. For example, if the attacker is interested in the user's streaming activity instead of their website visits, the attacker can isolate the video streaming traffic (using IP addresses of content providers such as Youtube and Netflix) and follow an approach similar to that of the scenario in Figure 1.2 to identify the specific video streamed by the user.

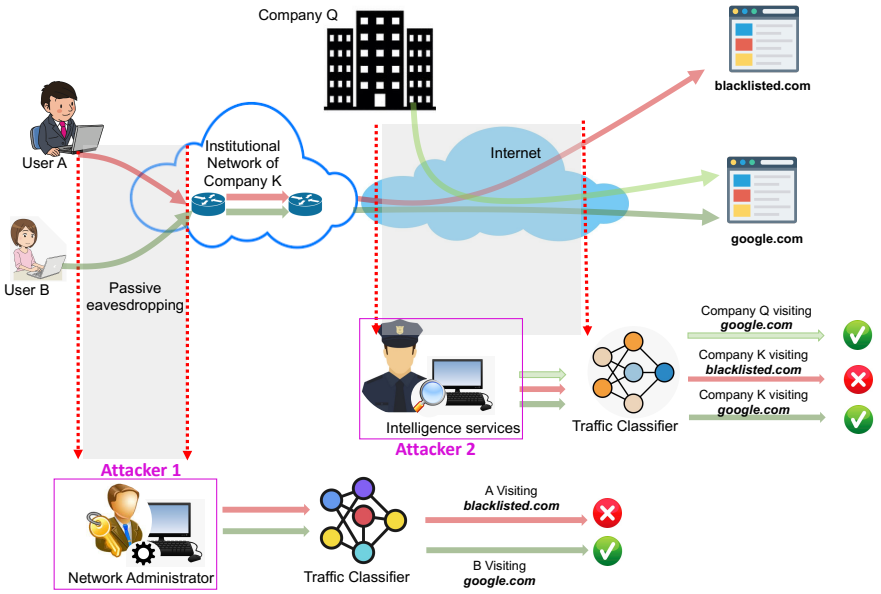


FIGURE 1.2: Threat model

Most early works on TA attacks were based on traditional machine learning techniques such as SVMs, Random forests, Bayesian models, k-Nearest Neighbor method and etc.

For example, for website fingerprinting over Tor, SVMs [8], k-Nearest Neighbor classifiers [9], Random Forest [10] and Bayesian models [11] were used in early works while a Hidden Markov model was used for uncovering spoken phrases in VoIP conversations [12]. While these traditional machine learning models gave reasonable results with relatively lower training costs, especially for a relatively small number of target classes, they suffered from drawbacks such as lower accuracies and the dependence on manually extracted features which require expertise on the behavior of specific traffic types. With the rapid growth of Internet traffic, a growing amount of training data became available for training TA attacks, and inspired by the success of deep learning techniques in other domains like computer vision and natural language processing since 2012, researchers explored the feasibility of using deep learning models for TA attacks. In most cases, deep learning models do not require manually engineered features, and they have the ability to achieve very high accuracies even in the presence of a very large number of classes, given an adequate amount of training data.

The first attempt at using deep learning for TA was made by Wang [13], where a Stacked Denoising Autoencoder (SDAE) was used for network protocol recognition on encrypted traffic. Since then, multiple works have used deep learning methods such as SDAE, Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) and Multi Layer Perceptron (MLPs) for various TA tasks such as website fingerprinting [14, 15], video stream fingerprinting [16, 17] and voice command fingerprinting on smart speakers [18], recording over 90% classification accuracy.

A summary of the evolution of traffic classification and traffic analysis attacks, along with events with a significant effect on such attacks, is given in Figure 1.3.

Even though TA attacks that leveraged deep learning methods achieved very high accuracies, they still face challenges that limit their practical applicability. For example, most TA attacks presented in prior work assumed that samples from all classes the classifier might encounter after deployment is available at training, which is not realistic in the context of TA attacks due to the large number of target classes available. On the other hand, the improvements in TA attacks increase the possibility of these attacks being exploited by malicious parties to undermine privacy guarantees provided by E2EE. Therefore designing low-cost defenses that provide adequate security against

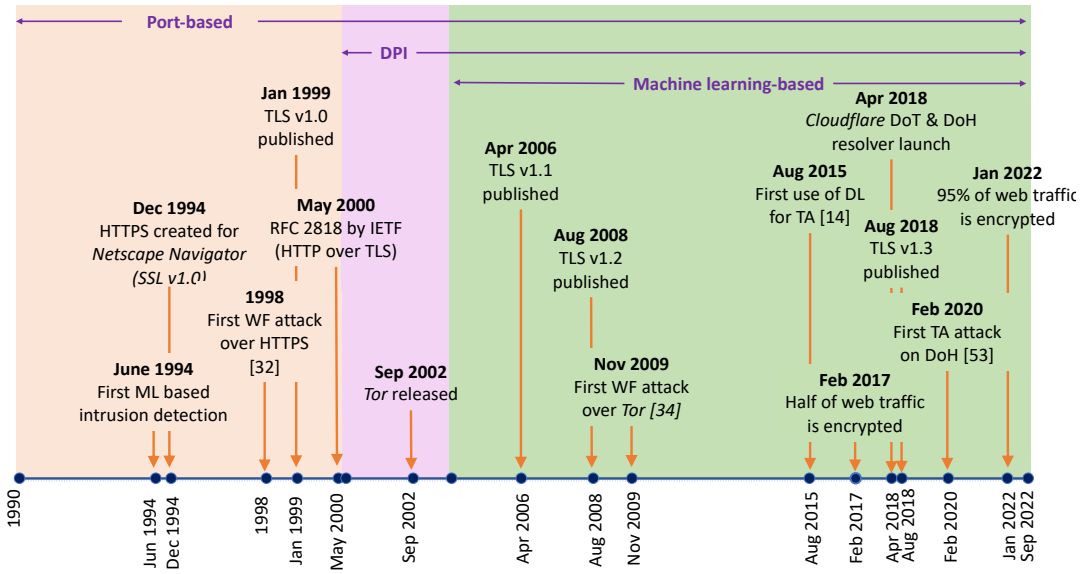


FIGURE 1.3: Evolution of traffic analysis

TA attacks while imposing minimum effect on the functionality of Internet-based applications is of significant importance. Towards this end, having a clear understanding of the behavior of deep learning-based classifiers that are, in general considered black-box attacks could be instrumental. The following sections will address these issues related to deep learning-based TA attacks. First, Section 1.3 discusses defenses against TA attacks, and then Section 1.4 explains how concepts in explainable AI can contribute towards developing better TA attacks as well as defenses. Finally, Section 1.5 presents how current TA attacks can overcome the limitations of closed set assumption with open set classification.

1.3 Defenses against Traffic Analysis Attacks

As discussed earlier, traffic analysis attacks aim to infer important information from encrypted traffic flows. While such attacks are useful for various network management and malware detection tasks, they also pose a significant threat to the privacy of Internet users, even in the presence of E2EE, as information about the underlying content of encrypted traffic is exposed to third parties. For instance, Chen et al. [19] showed that very detailed personal information such as illnesses and family income can be inferred by simply observing the HTTPS traffic flows, which can cause significant security risks to the victim. Accordingly, government intelligence agencies as well as third parties with

malicious intents towards the users can use TA attacks to undermine privacy guarantees provided by E2EE and hence, multiple research has explored possible mitigation techniques against such attacks.

TA attacks are made possible by the side-channel information leaks of encrypted traffic flows, such as packet lengths and timing, and therefore, the main goal of a defense against such attacks would be to obfuscate the side-channel information of encrypted traffic flows. If we refer back to Figure 1.1 where we explained side-channel information using the analogy of a gift-wrapped bike, a mitigation technique there will try to hide the specific shape and size of the wrapped bike. In this case, a naive way of hiding the shape of the bike would be to put the bike inside a large box and then wrap that box. Even though this approach completely hides the details of the bike from an observer, it should be highlighted that it would incur a significantly high cost (i.e., we now need a large box and much more wrapping paper than required to wrap the bike only). Similarly, a defense against TA attacks aims to obfuscate the lengths and timing information of encrypted packets and therefore, will either add dummy data into packets or split existing packets before encryption to hide packet size and introduce delays for packet transmissions to obfuscate timing information. Similar to the analogy of hiding the details of the bike by putting it inside a box, using padding and packet delays incur additional costs. More specifically, introducing dummy data into packet streams incurs additional data overheads while time delays incorporated into packet streams cause timing overheads which can result in severe degradation of user experience for some applications like streaming services. Accordingly, a defense against TA attacks would have to leverage either one or both of the above approaches but would have to explore efficient approaches of doing so without becoming impractical to be implemented in real-world scenarios.

One of the earliest works against TA attacks was BuFLO [20], which pads/fragments all packets to a fixed length and sends packets at fixed intervals, which incurs significant data and timing overheads. Later research [21, 22] attempted to improve the data and timing overheads of BuFLO but did not succeed in reaching a level where these defenses can be practically implemented in real-world scenarios. WTF-PAD [23], a defense against website fingerprinting over Tor which provided reasonable privacy with tolerable data overheads and zero timing overheads using adaptive padding, was chosen

as a candidate defense and yet, was later proven ineffective against a recent deep CNN-based attack *Deep Fingerprinting* [14]. More recent works such as FRONT [24], which can be considered as the current state-of-the-art defense against website fingerprinting over `Tor`, based its defense on specific traffic patterns of web surfing and focused on adding more padding at the beginning of a trace thereby reducing the data overhead to an acceptable level. It should be noted that a vast majority of defenses against TA attacks focus on website fingerprinting over Tor, and defenses against other types of TA attacks are scarce. Zhang et al. [25] explored the possibility of using statistical privacy to defend against such video stream fingerprinting attacks but their methods incur significant data overheads as well as timing overheads which affects the user experience of video streaming and are less adaptable for DASH streaming.

Accordingly, it should be noted that further research is needed to defend Internet traffic against TA attacks while incurring reasonable overheads that would not affect the deployability of such defenses in real-world applications. And we propose that having a better understanding of the deep learning-based classifiers that are frequently used in TA attacks would lay a strong foundation towards this objective.

1.4 Explainable Artificial Intelligence

Even though deep learning models have achieved near-perfect results in various tasks, very little is known about the inner workings of such models. Deep learning models in general, are neither explainable nor interpretable and are often referred to as black-box models as they provide no information about the reasoning behind their predictions. It should be noted that even though a deep learning model gives near-perfect results for a given task on a specific dataset, not knowing how the classifier arrives at its decision or what exactly the model is learning can be unsafe in the long-term deploying of such models in fast-changing environments with novel test data appearing quite frequently. For example, in [26], DeGrace et al. discovered that most deep learning-based classifiers used to detect COVID-19 using chest radiographs fail to learn the true underlying pathology reflecting the presence of COVID-19 and instead learn 'shortcuts' based on spurious associations between the presence or absence of COVID-19 and radiographic features that reflect variations in image acquisition. At the same time, not knowing the

actual decision-making process of a model can cause distrust towards deploying such models in real-world, safety-critical applications.

As a solution, researchers started exploring techniques that could help humans understand the inner workings of deep learning models and their decision-making process. This area of research became increasingly popular within the computer vision research community, and one such attempt was Grad-CAM [27] which uses the gradients of a given target label flowing into the final convolutional layer of the model to produce a coarse localization map that highlights the important regions in the image for predicting that label. In Figure 1.4 we show the results of using Grad-CAM[1] on an *Xception* [28] model trained on the *Imagenet* [2] dataset to visualize the pixels of an input image that are most influential in predicting a specific class label. When Figure 1.4a was fed into the model, the top two predictions were *German shepherd* and *Rottweiler* with class scores of 0.35 and 0.14 respectively. We then used Grad-CAM to visualize which pixels that are responsible for the class score for *German shepherd*, and as shown in Figure 1.4b, the face of the German Shepard dog is, in fact most responsible. Similarly, when used to visualize the same for the class *Rottweiler*, the result is Figure 1.4c where the Rottweiler dog's face is shown to be responsible for the class score for class *Rottweiler*. This visualization is helpful in verifying that the model has correctly learned to focus on the relevant area of an image to generate class probabilities.

(A) Original image (B) Result for class *German shepherd* (C) Result for class *Rottweiler*

FIGURE 1.4: Sample result from Grad-CAM

This area of research became increasingly popular within the computer vision research community, with researchers proposing multiple techniques such as *Gradient Ascent* [29], *Deconvolution* [30] and *Occlusion experiments* [30] to visualize the patterns learned by various filters from different depths of deep CNNs used in computer vision tasks. For

[1] `https://keras.io/examples/vision/grad_cam/`
[2] Imagenet: `https://www.image-net.org/`
Pre-trained model: `https://keras.io/api/applications/xception/`

example, using the above methods, prior work discovered that with most 2D CNNs used in computer vision tasks, the filters from initial layers focus on very simple patterns, such as the orientation of lines, while filters of deeper layers learn complex patterns [29].

A close study of recent TA attacks reveals that for most TA attack types, CNNs perform best compared to other deep learning and traditional machine learning models [15, 16, 18]. When considering RNNs and CNNs, this observation can be counter-intuitive as CNNs were mainly designed for image classification, whereas, for time series data such as network traffic flows, RNNs were shown to be more successful in other domains such as speech recognition and speaker identification [31, 32]. Accordingly, a better understanding of the decision-making procedure of CNNs used for TA attacks would help improve current TA attacks as well as develop efficient defenses against such attacks. To the best of our knowledge, such analysis has not been done on TA attacks, and we highlight the need for more research on to this area.

1.5 Open Set Classification

Although deep learning models have shown near-perfect results with classification tasks in most domains, a majority of those models were evaluated in a *closed set* setting. A closed set setting refers to a scenario where the classification model works under the assumption that all possible target classes were available during training and aims to classify each test sample into one of the classes seen during training. However, this closed set scenario is not realistic due to a few reasons. First, given the large number of classes present, it is not possible to include samples from all those classes at training time which would increase the resource consumption of the model. Additionally, in most scenarios, we are not interested in all possible classes but a subset of them, and trying to train a model for all possible classes would be wasteful. For example, if we consider website fingerprinting, we would not be interested in fingerprinting all possible websites available on the world wide web but rather in identifying a subset of websites we are interested in. Secondly, in the fast-evolving world, new classes would be introduced continuously, and hence it is not possible to represent all possible classes during training time. For instance, new websites are introduced every day, and it is expected that the training set will not contain samples from websites introduced after model training. Due to the above reasons, once deployed, classification models will encounter samples from classes

not seen during training. In such a situation, the closed set classifier would assign those samples from unseen classes to one of the classes it has seen during training, which could be problematic.

First, let's consider an example situation where we have a deep CNN network trained to detect specific bird species with very high accuracy. We are only interested in detecting the bird species *Bluejay* (A), *Magpie* (B) and *Ostrich* (C). Once deployed in the natural environment, the model will encounter samples outside these three bird species as illustrated in Figure 1.5. For example, the model will be presented with airplanes, clouds, trees and animals in addition to the above three bird species. A model trained in the closed set setting would classify any of these samples as one of the three targeted bird species, which undermines the original purpose of the classifier. For the model to achieve its expected goal, it should have the capacity to correctly detect any of the three bird species while rejecting any inputs from other classes like trees, clouds or dogs.

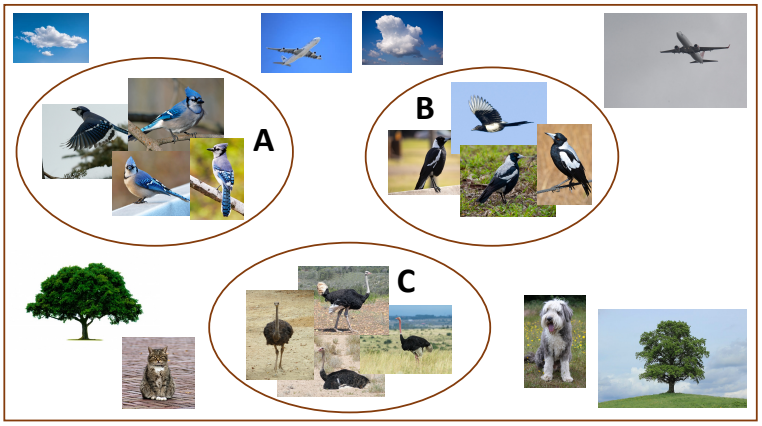


FIGURE 1.5: Open set example

A more realistic scenario would be the open set scenario where the goal of the classifier is to correctly classify samples from classes it has been trained on while effectively rejecting any sample from previously unseen classes (open set). The most naive way of open set classification would be to treat all open set classes as a single class (background class) and train the classifier with an extra class using a few samples of the open set during training. This method is based on the intuition that all open set samples have similar characteristics and are different from any known class. However, this assumption is not always true since an unseen class could have characteristics very much similar to one of the known classes and significantly different from the samples used as the open set during training. Hence, this method does not always work and has the additional disadvantage of requiring a comprehensive subset of the open set for training.

Another approach would be to assume that a classifier would give lower confidences for samples from classes not seen during training and reject samples with lower prediction (softmax) scores as open set samples. This method is called *Softmax thresholding*. However, default model training of deep learning models has a closed nature and does not encourage a classifier to output higher prediction scores for the correct class. Rather it just needs the prediction score of the correct class to be the highest out of all classes. Accordingly, the assumption used in softmax thresholding might not be applicable to all situations.

Since the above two naive methods have considerable drawbacks, researchers have faced the novel challenge of developing better open set classifiers. The first key work on open set classification with deep learning models; OpenMax [33], was proposed for the computer vision domain. Afterward, several other methods [34, 35] were also proposed by the computer vision community. However, to the best of our knowledge, all previous work on network traffic domain that handled the open set did not explore beyond the two naive methods, background class method and softmax thresholding [14, 15, 18]. Furthermore, there is no prior work studying the effectiveness of using open set methods proposed in the computer vision domain on network traffic. Hence we emphasize the need to study those methods with network traffic data and customize them to suit the behavior of network traffic better. At the same time, open set classification in general can be considered to still be in the early stages, and further research is required to improve the open set classification results.

1.6 Structure of the Thesis

The rest of this thesis is structured as follows.

Chapter 2 presents related work under the categories: *Traffic Analysis Attacks against End-to-End Encrypted Traffic*, *Understanding CNNs* and *Open set Classification*.

Chapter 3 first shows the feasibility of using DoH traffic for website fingerprinting attacks and then presents a more practical *Traffic Analysis Attacks on DoH Traffic* under realistic assumptions such as inline traffic classification, open set classification and

working with network traces without explicitly knowing the start of the event.

Chapter 4 identifies the lack of prior work on understanding 1D convolutional networks (which were observed to be the most successful for many TA tasks) and then explores several approaches to understand the inner workings of traffic fingerprinting CNNs using six different network traffic datasets. Finally, based on the observations from the above experiments, it proposes two novel defenses against traffic analysis attacks on encrypted network traffic.

Chapter 5 highlights the lack of prior work exploring the open set scenario in TA attacks and evaluates the efficiency of using several key open set classification methods; two naive methods and four methods proposed for computer vision domain, on encrypted network traffic. This work also proposes a novel open set classification method leveraging signatures built from neuron activations in deep learning models.

Chapter 7 concludes the thesis by summarizing the results and discussing possible future directions of research.

Chapter 2

Related Work

With the exponential growth of internet communications, entities such as ISPs and network administrators face an increasing requirement of understanding the behavior of network traffic in their networks in order to perform various important tasks such as network optimization and management, and application/content based service differentiation. On the other hand, intelligence communities also need a way to understand behavior and underlying patterns of network traffic, which reflects the behavior of internet users. Hence, *Traffic Classification* which refers to identifying type of traffic on networks, emerged in the mid 90's. In Chapter 1 we discussed how the techniques of traffic classification evolved through the years, starting from port-based methods and deep packets inspection until the machine learning/deep learning based current *Traffic Analysis* (TA) attacks. As discussed in Chapter 1, our research mainly focuses on TA attacks on encrypted traffic and defenses as a significant portion of current internet traffic is end-to-end-encrypted. More specifically, our research attempts to get a better understanding of the behavior of TA attacks and improve TA attacks and defenses to better suit real world applications. Accordingly, in this section, we discuss in detail about key relevant previous research under the following sub topics.

- Traffic analysis attacks on encrypted traffic
- Defenses against TA attacks on encrypted traffic
- Open set classification over deep learning models
- Understanding the behavior of deep leaning models

2.1 Traffic Analysis Attacks

It has been known for a while that side-channel information such as packet lengths and timings of encrypted traffic can leak important information about the underlying content such as websites requested [7, 36] or sensitive user data such as health conditions, annual income and investment details entered via web forms [19]. In this section, we present previous work that introduces traffic analysis attacks targeting to uncover specific details from encrypted traffic such as website visits, videos streamed and voice traffic.

2.1.1 Website Fingerprinting (WF)

Website fingerprinting refers to traffic analysis attacks that infer the websites visited by a user, especially over encrypted traffic flows that are passively observed. A WF attack can be considered as a supervised classification problem where each traffic trace is considered as an instance to be assigned a label from the set of labels comprised of the website domain names under a setting similar to that shown in Figure 1.2. It should be noted that for most direct website visits (that do not use anonymizing approaches like Tor or tunneling), the destination IP address itself can be used to identify the website without resorting to traffic analysis. Therefore, most WF attacks presented after the release of the anonymity network (*Tor*) focus on WF attacks over Tor (Figure 2.1), which uses encryption (HTTPS) on top of providing anonymity. According to Figure 2.1 the attacker eavesdropping on a user that uses Tor sees the IP address of the entry guard as the destination IP and therefore, can not identify the website using destination IP only. Therefore the attacker needs to leverage traffic analysis attacks to fingerprint websites over Tor. These attacks can be broadly categorized into two types based on whether the inference was done using entire HTTPS communications or just using the encrypted DNS communications aiming to resolve requested websites.

2.1.1.1 HTTPS Traffic

One of the first studies on WF attacks was done in 1996, when Wagner et al. revealed that when a web browser connects to a web server via an encrypted transport channel such as SSL, the GET request containing the URL leaks the length of the URL requested, and along with the length of the HTML data received from the server, this

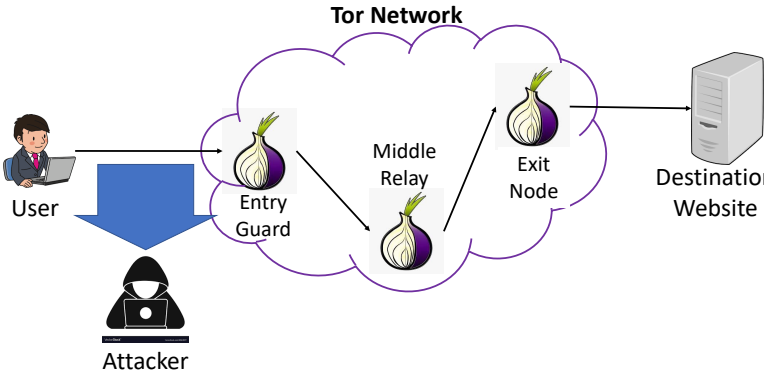


FIGURE 2.1: WF attack over Tor network

leakage allows an eavesdropper to identify the website accessed by the user [7]. Andrew Hintz [37] presented a successful implementation of a WF attack on *Safe Web*, which is an encryption-based web proxy.

WF over Tor with traditional ML: Even though Tor has been one of the most popular tools of web anonymity since its release in 2002, WF potential on Tor was not studied until 2009. The first evaluation of a WF attack on Tor was performed by Herrmann et al. [38] using a naïve Bayes classifier with the frequency distribution of packet lengths as the feature set. Their attack was not effective and achieved a classification accuracy of only 3%. Their failure was due to Tor's use of fixed-sized (512 bytes) data units called *tor cells* to hide the unique packet lengths. In 2011, Panchenko et al. [8] devised an improved WF attack using an SVM and additional features that exploited burstiness of traffic (rather than packet length) and achieved a classification accuracy of 55%. After their work, multiple works incrementally improved WF attacks using better feature sets.

In 2014, Wang et al. proposed a novel attack using a k-nearest neighbor (k-NN) [39] classifier, which achieved a significant improvement in classification accuracy compared to previous work. For their model, the authors used a large, diverse set of features that included packet ordering, packet concentration, the number of incoming and outgoing cells and bursts in packets, which was believed to have contributed to the success of their model. First, the authors formed a distance metric using a combination of weighted features to measure the similarity between traces of two websites. The weights which indicate the importance of each feature were learned from traffic instances in the training set. Then, the authors used the learned weights to compute distance and identify nearest

neighbors. This k-NN model achieved an accuracy of 91% in a closed world setting with 100 websites.

Two years later, Panchenko et al. proposed CUMUL [40], a model that used a Support Vector Machine (SVM) with a Radial Basis Function kernel and achieved an overall accuracy of 92% in a closed world setting of 100 websites. The authors derived the feature set for the SVM using the cumulative sum of packet lengths computed by adding the lengths of outgoing packets and subtracting the lengths of incoming packets. Regardless of the high success rate of the attack, the authors pointed out that WF attacks cannot scale when applied in a real-world setting because an adversary has to train the classifier on a large fraction of all websites to achieve high accuracies. It is important to note that the performance of the above SVM was evaluated on a novel dataset which was the most realistic dataset at the time that represented the actual websites browsed in the internet via Tor. The authors used trend links on Twitter, trends on Google, and censored sites in China to collect the above dataset. Moreover, Panchenko et al. were the first to distinguish *website fingerprinting* from *webpage fingerprinting* and evaluated the fingerprintability of both single webpages and complete websites.

In the same year, another WF model k-FP, that achieved a similar performance was proposed by Hayes and Danezis [41]. k-FP used a Random Forest Classifier to encode a new representation of the monitored sites using a combination of features from prior work and timing information such as the number of packets per second. The output from the Random Forest Classifier was then fed into a k-NN classifier for the actual classification. It should be noted that k-FP is the first WF model that considered timing features. As a part of their study, Hayes and Danezis analyzed and ranked the importance of the features used in their model. The rankings revealed that the number of packets in a sequence leaks more information about the identity of the webpage compared to complex features like packet ordering and inter-packet arrival time.

Drawbacks of using traditional ML for WF: As discussed above, the success of the above WF models largely depended on manually selecting better feature sets, and the reason for this is because all above methods use traditional machine learning techniques where manual feature engineering is an important step. Feature engineering is a process that finds a representation of raw data that projects characteristics that are

most relevant to the learning problem. Several previous work [20, 42] have also proved that feature engineering is more important in traditional ML models than the choice of specific machine learning algorithms. Accordingly, all the above work on WF focused on feature engineering in order to compose and select the most relevant features for website identification, making the above attacks highly sensitive to changes in traffic patterns that affect the selected features. As a result, most countermeasures in the Tor network that focused on concealing specific features were sufficient to defend against such attacks. The vulnerability of the above attacks to minor changes in traffic patterns, along with costly manual feature engineering rendered such WF attacks less realistic.

WF over Tor with deep learning: As a solution to the weaknesses of traditional ML techniques, researchers explored the potential of using *Deep Learning* (DL) for WF attacks. The principle motivation for their choice came from the recent success of DL techniques in other domains such as image recognition and speech recognition [43, 44], and the capability of DL to automatically extract and learn effective features that are not easily discovered or understandable by humans as opposed to manual feature engineering used with traditional ML models.

The first work to apply DL for WF attacks was done by Abe and Goto [45] in 2016. The authors developed the attack based on *Stacked Denoising Autoencoders* (SDAE) and trained the model on raw packet directions, denoted by a sequence of 1's (outgoing packets) and -1's (incoming packets). The performance of the SDAE model was evaluated on Wang-kNN's dataset, and the model achieved an 88% accuracy in the close world setting. The low performance of the model compared to the state-of-the-art traditional models was attributed to the limited number of training examples used in the experiment, even though DL models are known to require large datasets for training. In 2017, Rimmer et al. proposed using deep learning to devise a website fingerprinting attack with the intention of leveraging the capability of deep learning techniques to exploit several layers of non-linear mathematical data transformations for automatic hierarchical feature extraction and selection [15]. Since the main reason for the low performance of the above SDAE model was the lack of sufficient training data, the authors first focused on generating a larger dataset. Their dataset which was the largest WF dataset gathered to date, comprised of traces for 900 websites with 2500 traces per website for the closed world setting, 400,000 unknown websites, and 200 monitored websites

in the open world setting. The authors evaluated the performance of traditional WF attacks on their new dataset and identified that the CUMUL attack was the state-of-the-art among them. Next, Rimmer et al. implemented three deep learning models, namely SDAE, *Convolutional Neural Network* (CNN) and *Long-Short Term Memory* (LSTM), and compared their performance with each other and against CUMUL. The results of the experiment revealed that all the models using the deep learning approach were more robust against web content changes than CUMUL, while achieving a similar performance of approximately 95%. Specifically, LSTM model was observed to be twice as robust as CUMUL against changing web content, and SDAE showed better results on a large close world dataset than CUMUL. The authors also identified that among the three DL models, SDAE had the best overall performance while CNN was the fastest to train (fewer learnable parameters) though it had a higher risk of overfitting. LSTM was the slowest but showed the best generalization capabilities. However, the LSTM model's constraint in backpropagation had adverse effects on the model's performance on long traffic traces.

In 2018, Sirinam et al. developed a WF attack named *Deep Fingerprinting* [14] using a CNN model which performed better than [15]. Compared to [15] their DF model used a more sophisticated variant of CNN with more convolutional layers, better protection against overfitting, hyperparameters that vary with the depth of each layer, activation functions tailored to their input function, and a two-layered fully-connected classification network. The authors evaluated the performance of Deep Fingerprinting against a dataset of 100 websites with 1000 traces per site and achieved an accuracy of 98% in the closed world setting. Hence, DF can be considered the state-of-the-art classifier in website fingerprinting.

More recently, Sirinam et a.l [46] explored the use of N-shot learning for website fingerprinting, with the main objective of reducing the amount of training data required. Their work used a triplet network with the CNN from [14] as the base model and achieved up to 95% accuracy using only 20 traces per website. Furthermore, they compared the performance of their model against a transfer learning scenario and showed that N-shot learning is more efficient at handling less training data compared to transfer learning.

TABLE 2.1: Summary of WF

Attack	Classifier	Feature set	Number of classes	Accuracy	Open set
k-FP [41] 2016	Random forest	Packet length and timing	100	90.00%	N/A
CUMUL [40] 2016	SVM	Cumulative sum of packet lengths	100	93.00%	N/A
AWF [15] 2018	CNN	Packet direction	900	91.79%	Binary classification with softmax thresholding
DF [14] 2018	CNN	Packet direction	95	98%	Background class
Triplet Fingerprinting [46] 2018	CNN	Packet direction	100	94.5%	Background class
Var-CNN [47] 2018	CNN	Cumulative features of packet size, number and timing	900	99.0%	Background class

2.1.1.2 Encrypted DNS Traffic

Domain Name System (DNS) translates human-readable domain names to numerical IP addresses. The fact that the domain names that the users are requesting to resolve are sent out in plain text has been considered a significant privacy threat on the internet for a long time. As a result, several proposals such as DNS over TLS [48], DNS over QUIC [49], DNS over HTTPS [50] were made to use encryption as a means to ensure the privacy and confidentiality of DNS requests and responses. However, later research discovered that encrypted DNS traffic is also prone to side-channel attacks and as a result, the statistical properties of the DNS packet flows can be used to predict the websites that are being visited by the users with high accuracy.

WF over Tor with DoT traffic: Early work by Shulman [51] investigated side-channel privacy leaks of DoT. However, the work focused on using the destination IP address of the name server as the side-channel. Since name servers often host multiple zones, the author argued that the risks are limited. Nonetheless, Shulman pointed out that the implementation of encryption alone is not adequate for protecting the privacy of DNS. Hence, the use of padding for encrypted DNS traffic flows was introduced and standardized [52, 53]. More recently, several other works [54, 55] attempted more realistic attacks that depend only on the destination IP addresses for website fingerprinting while handling website co-location and the dynamics of domain–IP mappings.

Houser et al. [56] studied website fingerprinting attacks that exploited information leakage of DoT. The authors extracted high-level features (e.g., DNS query or response lengths and time intervals) from DNS flows and calculated summary statistics (e.g., mean and maximum) to train a Random Forest classifier for webpage category identification (e.g., health insurance, dating and gambling) and an AdaBoost classifier to identify individual webpages. The authors tested both padded and un-padded data. The model for identifying individual webpages reported a 2.5% FNR for un-padded data and a 17% FNR for padded data. Nonetheless, this work did not consider the open set traffic classification problem. Also, all the classifiers operated on the full trace length.

WF over Tor with DoH traffic: Siby et al. [57] explored website fingerprinting using DoH traffic flows. The authors used a concatenation of uni-grams and bi-grams of sizes and directions of TLS records and bursts as features to train a Random Forest classifier that achieved an F1 score of 0.898 in a closed set setting. The authors also evaluated their attack on data with EDNS(0) padding and discovered that padding is not as effective in strengthening DoH privacy since they still could achieve an F1 score of 0.71 on padded data. The authors handled the open set scenario by training a binary classifier to make the initial decision of whether a given trace belongs to a monitored (known) or un-monitored (open set) set and achieved an F1 score of 0.7.

2.1.2 Video Streaming Fingerprinting

According to Cisco's Visual Networking Index, online video traffic would account for 80% of all web traffic by 2019. Adaptive Bitrate Streaming (ABS) based on HTTP is gradually becoming the major market of video streaming due to its advantages of flexibility and infrastructure-friendly property. By splitting videos into segments of multiple quality levels (bitrates), ABS enables a smart client-driven bitrate adaptation. Dynamic Adaptive Streaming over HTTP (DASH) is a representative implementation of ABS that has been an international standard since 2011 and is widely used by leading companies of video streaming such as YouTube, Netflix, Amazon, and Vimeo. A video in DASH is first encoded into multiple copies of different quality levels with respect to bit rate and resolution using Variable Bit-Rate (VBR) encoding, and each copy of a video is split into segments of a fixed length of playback time and stored in web servers. When streaming a

video, a client will request a video segment in a certain quality level for playback depending on the current network condition. It is observed that such a mechanism in DASH results in distinct traffic patterns due to segment-based transmission and segment size variation of VBR, which can be used to devise a side-channel video identification attack. A video fingerprinting attack can be considered as a supervised classification problem where each traffic trace (corresponding to streaming a given video) is considered as an instance to be assigned a label from the set of labels that comprises of the video titles under a setting similar to that shown in Figure 1.2. The popularity of video streaming intensifies the seriousness of such traffic-based information leakages.

Pre-computed fingerprints for video streaming: A. Reed and B. Klimkowski performed one of the first comprehensive studies of side-channel information leaks in video streaming services [58]. The authors presented an attack that is capable of identifying the Netflix video being streamed over a secure WPA2 wireless connection (802.11n) with high accuracy in less than five minutes. The attack was based on the observation that the combination of DASH and VBR produces sequences of video segment sizes that are unique for each video. The authors used a script to extract the metadata URLs for each quality level of a video from the HTTP GET requests sent by the Silverlight player and downloaded all metadata to calculate the segment sizes of each video. The results were then used to create a database of fingerprints for a video's multiple encodings. Finally, the authors used an algorithm implementing a 6d tree to match captured network traffic to video segments in their fingerprint database and achieved an accuracy over 90%. In [59], A. Reed and M. Kranch improved the above attack by A. Reed and B. Klimkowski by fully automating the process of creating fingerprints to generate a larger collection of Netflix fingerprints (fingerprints for 42027 Netflix videos) which was then used to conduct a robust assessment of the attack. Using a similar approach to [58] on the new dataset, the authors were able to identify HTTPS-protected Netflix videos using IP/TCP headers obtained from passive capture of network traffic in less than 2.5 minutes into the video and achieve an identification accuracy over 99%.

In order to explore whether YouTube video streams playing over HTTPS-protected connections are vulnerable to the same kind of attacks illustrated in [58] and [59], Melcher Stikkelorum developed an algorithm named WARP (WARP Arrangement Rewarding Pipeline) [60]. The algorithm used state machines to match video segments and reward

well-ordered segments. When evaluated on a small dataset, the algorithm reached 80% accuracy on the detection of videos in a playlist and 88.9% for separately played videos. However, the author recognized the requirement to perform more experiments on larger datasets to determine the performance of WARP on larger datasets.

Video stream fingerprinting with deep learning: The first application of Deep Learning in the context of identifying videos streamed over encrypted channels was performed by Schuster et al. [17]. The authors observed that most of the previous work [58, 59] generated many false positives and also were not robust to noise in the network. As a solution to the above issues and to be able to process lower-level features and construct a more complex model to characterize the network traces of a given video, the authors proposed using a CNN. The authors defined the problem as a supervised learning problem where each training instance is a traffic measurement of streaming a video, labeled with its correct class, which is the identity of the video. The CNN model used comprised three convolution layers, max pooling, and two dense layers and was trained using an Adam optimizer on batches of 64 samples using categorical cross-entropy as the error function. In order to evaluate the performance of the model, a novel dataset was also created. The authors collected traffic traces on Netflix, YouTube, Amazon, and Vimeo and, from each captured TCP flow, extracted the time series of down/up/all bytes per second (BPS), down/up/all packet per second (PPS), and down/up/all average packet length(PLEN). Using the CNN model described above, the authors trained separate classifiers for each video streaming service and each feature type. The results revealed that all classifiers achieved accuracies above 96% with very low false positive rates (when fed with the trace of a previously unknown video, the classifiers identified that the video was not seen before with high accuracy). Since it is not practical for an adversary to collect traces to train the model from the same local network as the victim, the authors also evaluated the performance of their classifiers on test data which were collected by streaming videos via a local network different from the network used to collect the training set, achieving an accuracy of 98%. This result confirmed that the model was robust to changes in the network conditions.

In 2018, Li et al. investigated the effectiveness of using traffic analysis on wifi traffic (encrypted at both network and MAC layer) to identify video streaming content [16]. Most works prior to their research had used network layer traffic and hence had access to

metadata such as IP addresses of source and destination, which can be used to identify separate traffic flows accurately and conveniently. Accordingly, one of the major challenges for the authors was the absence of statistical properties of traffic flow and other metadata. Despite these challenges, the attack model developed in the research achieved a similar performance (above 97% accuracy) while using fewer resources compared to the CNN model [17], which was the state of the art for traffic analysis attacks on video streams using IP traffic. The authors implemented three neural network architectures; CNN(same as state of the art for comparison), RNN and MLP, and measured their performance with respect to classification accuracy and resource consumption (time and computational power). The data set used for the experiment was obtained by capturing wifi traffic from a laptop that repeatedly streamed the same ten videos. Using MAC layer parameters such as frame size, frame type, duration, MAC source and destination, the captured packets were first filtered to separate data frames, and later the separated data frames were grouped as uplink, downlink and combinational. The features generated from this dataset included the number of packets in a dataframe, the number of bytes in a dataframe, min, max, average, and variance of packet size. The results of the experiment revealed that all three models performed approximately the same with respect to accuracy. Specifically, the MLP model was faster to train and used much less resources compared to the other two models. However, the models developed do not apply to a real-time TA attack since the dataset used corresponded to the first three minutes of streaming a video and a real-time attack would require performing the identification using traffic corresponding to a much smaller interval of the video. Furthermore, they assumed a scenario where the streaming videos do not contain advertisements, which is a deviation from the real-life scenario. Having advertisements changes the traffic patterns of streaming a video and hence affects the performance of the models.

2.1.3 Voice Traffic Fingerprinting

Since the early 2000's Voice over IP (VoIP) has become increasingly attractive as an alternative to traditional phone services provided by Public Switched Telphone Networks (PSTN) due to low cost and richer features. With applications like *Skype*, *WhatsApp*, *Viber* and *Telegram* seeing exponential growth in recent years, encryption is used to

ensure confidentiality of the content of voice traffic.

Traffic analysis on VoIP: In 2007, Wright et al. [61] showed that VoIP packets are compressed with variable bit rate (VBR) encoding schemes to save bandwidth before being encrypted with a length-preserving stream cipher which can be used to determine the language of the encrypted conversation. A year later, Wright et al. [62] used a profile-hidden Markov model trained with speaker and phrase-independent data to detect the presence of specific phrases in VoIP calls with an average recall and precision of 50% and 51% respectively, for a wide variety of phonetically rich phrases spoken by a diverse group of speakers. In 2010, Wright et al. [12] extended their previous work to show how noise, dictionary size, gender, and audio quality affect the performance of the techniques proposed in [62].

Zhu and Fu [63] proposed a method that extracts application-level features from encrypted traffic flows, which is then used in a class of passive traffic analysis attacks on Skype VoIP calls. The extracted features are then used to train a Hidden Markov Model (HMM) that can detect spoken phrases or individual speakers.

Fingerprinting voice commands on smart speakers: Smart speakers, such as Amazon Echo, Google Home and Apple HomePod, are being increasingly adopted in households all over the world. If revealed to an outsider, a user's interactions with a smart speaker can expose private and personal information about them and their lifestyle. Wang et al. [18] observed that the content of a response from the corresponding server to voice commands given via smart speakers is correlated with the voice command and therefore can be leveraged to identify specific voice commands despite encryption. More specifically, they proposed a CNN based attack that can identify specific voice commands given to a smart speaker with over 90% accuracy by simply eavesdropping on a victim's WiFi network to capture encrypted traffic.

2.1.4 Traffic Analysis Attacks on Messaging Apps

During the past decade, instant messaging services have become the most dominant form of communication in the world with over tens of millions of messages, photos and videos

exchanged each day. For example, Whatsapp which owns the largest market share in the world for instant messaging, handled 55 billion messages, 4.5 billion photos, and one billion videos per day by 2017 [64]. Despite most of the instant messaging services using E2EE to ensure the privacy of their customers, the side-channel information leak of the encrypted traffic flows of such instant messaging services still pose a threat to user privacy.

Coull et al. were one of the earliest research groups to explore the potential of traffic analysis attacks on encrypted instant messaging services [65]. Focusing on the encrypted traffic of Apple iMessage, the authors investigated the possibility for an eavesdropper to learn information about user actions, the language of messages, the length of the message and the operating system running on the device of the user by passively monitoring streaming iMessage traffic to or from Apple servers. The authors used a binomial naïve Bayes classifier (from Weka machine learning library) that used packet length and direction as features and identified the OS of the observed device with 100% accuracy. The authors also observed that the stable and deterministic nature of packet lengths corresponding to most user actions in Apple iMessage (start, stop, text, attachment (image), and read) renders the use of probabilistic classifiers unnecessary to identify user actions and therefore used a hash-based lookup table to identify the user action with approximately 99% accuracy. Furthermore, the authors were also able to identify the language of the message using the Weka multinomial naïve Bayes classifier and achieve an accuracy of 93%. The distinct traffic features of the unique mix of character sets used to encode each language exposed by the corresponding encrypted traffic flows were identified as the reason for the success of the classifier. Moreover, the authors performed similar experiments on other instant messaging services (such as Whatsapp, Viber, and Telegram) and discovered that these services were also vulnerable to traffic analysis attacks regardless of using E2EE.

While most TA attacks proposed in previous work recorded near-perfect results, we have observed that they were under assumptions that are less realistic for real-world applications. Therefore our research aims to improve the attacks by introducing concepts such as open set classification and inline traffic classification which makes them more practical for real-world scenarios.

2.2 Open Set Classification over Deep Learning models

In Section 1.5 we discussed how deep learning models do not recognize samples from classes not seen during training (unknwons or open set) and would simply treat them as one of the known classes (closed set or classes seen during training). In real-world applications it is not possible to train a model on all possible classes and therefore, it is important that deep learning models be able to detect open set samples. Open set classification corresponds to a multi-class classifier that can correctly classify samples from known classes (closed set) while also effectively detecting and rejecting open set samples. Open set classification is a key requirement for TA attacks as well, because the Internet is a fast-evolving environment with a large number of classes and an attacker is mostly interested in a small subset of it. While we note that there has been prior work done on open set classification over traditional machine learning methods such as SVMs [66, 67], since the majority of recent TA attacks are based on deep learning models, we will focus only on methods that support deep learning models. We will first discuss the few works that attempted to handle open set classification on network traffic using naive methods and then present key open set classification techniques developed for the computer vision domain and discuss their applicability to encrypted network traffic.

Naive methods: One of the naive methods of open set classification is *softmax thresholding*. The majority of current deep neural networks use softmax activation in the last layer to obtain a probability vector representing the confidence of a given sample belonging to each known class. Softmax thresholding-based open set classification is based on the intuition that any model trained on a set of classes will output a very low softmax score (confidence) for samples from the open set. More specifically, this method simply uses a threshold over the softmax probability score of the predicted class and rejects samples with a probability lower than the threshold as open set samples. Additionally, the attacker may use a small dataset from known unknowns to decide the threshold value to obtain a preferred balance between closed set accuracy and open set accuracy. Nonetheless, since the softmax activation skews the output probabilities to favor the class with the highest probability and the training procedure does not explicitly push the model to output low confidences for open set samples, this method cannot be

expected to always work. For example, it is known that neural networks have very high confidence even if they make a wrong prediction [33]. Rimmer et al. [15] is one example work where softmax thresholding was used in traffic classification to handle the open set.

Background class method is another naive method that assumes that a subset of the open set (i.e., samples from known-unknown classes) are available during model training, and are combined to form a single *known-unknown* or background class. Given an input, the classifier learns either to put it into one of the known classes or to the background class, essentially making an n class classification problem into an $n + 1$ class classification problem. The background class method is based on the strong assumption that all the samples from unknown classes will have similar characteristics as the samples from known-unknown classes. This may not necessarily be true all the time. There might be samples of unknown-unknown classes that are closer to the known classes than the combined representation of known-unknown classes. Works in traffic classification such as [14, 68] used this method to tackle the open set problem.

Ensemble learning: Wang et al. [69] proposed using ensemble learning on top of softmax thresholding as a way of handling open set traffic classification for website fingerprinting. More specifically, the authors assume that combining the outputs from multiple model instances that would have learned different sets of features can help the overall model generalize better towards unknown data as opposed to a single model. Accordingly, the authors use a threshold on the averaged output from N different model instances to build an open set website fingerprinting attack. During model training, a dynamic learning rate is used to separate model locations in the loss function by immediately increasing the learning rate at fresh starts, so that the learning point displaces by a large distance resulting a new model having different set of parameters and learning different set of features. Additionally, authors use squeeze and excitation layers as an attention technique to weight features according to their effectiveness to the final result and increase the robustness of the model and separable convolution layers to reduce the computational cost.

Even though work in traffic classification has mostly used the above naive methods, other domains such as computer vision have developed more advanced open set classification

methods, and we will next discuss a few key works. We first discuss methods that do not require open set samples for training.

Distance-based methods: One of the first works that explored open set with deep neural networks proposed *OpenMax* [33]. The key idea behind OpenMax (and its variants [70, 71]) is to leverage the fact that the penultimate layer output of deep neural networks is a representation of relationships between classes in the closed set and open set samples will have anomalous behaviors. OpenMax uses Extreme Value Theory (EVT) modeling on the distance between a given sample and the mean of its predicted class in the space represented by the prenultimate layer to identify open set samples. Since it only depends on the distance between a sample and the means of known classes, it has the added advantage that it does not require samples from the open set during model training at all. Later Webb [72] explored the possibility of using OpenMax for traffic classification and required modifications to the method to adapt it to network traffic.

In 2021, Miller et al. proposed *Class Anchor Clustering* [34] which uses a novel loss function to force samples from the same known class to cluster closer to pre-defined class anchors in the space denoted by the output vector from the classifier model. To achieve this goal, the model replaces the softmax layer with a novel layer that calculates distances between a given sample and each of the anchors of known classes so that the loss function can reflect how far each sample is from its actual class anchor and how close it is to anchors of other classes. Similar to OpenMax, class anchor clustering does not need known unknowns during training time. To the best of our knowledge, this method has not been tried on traffic classification data.

Reconstruction loss-based methods: In 2019, Oza et al. proposed *Class Conditioned Auto-Encoder for Open-set Recognition (C2AE)* [73], which used a combination of reconstruction loss from a class-conditioned autoencoder and EVT modeling on the reconstruction loss to distinguish between closed and open set samples. This was based on the intuition that the autoencoder will do an accurate reconstruction of closed set samples while it will not do a good job in reconstructing open set samples. The reconstruction loss of the autoencoder model is defined such that the decoder is forced to perfectly reconstruct the original input when conditioned on the label matching the class identity of the input, while poorly reconstructing the original input when conditioned

on a label that does not match the class identity of the input. Later, EVT modeling is used to decide the threshold on the reconstruction loss which would be used to reject open set samples.

In the absence of open set samples (known unknowns) for training, using generative models to generate known-unknown samples and use them during training to improve the differentiation between the closed set and open set samples is another approach to handling the open set. We next discuss methods that followed this approach to model the open set.

Generative methods: Lee et al. [74] proposed a joint confidence method that used known unknown samples generated by a Generative Adversarial Network (GAN) [75] to restrain the overconfidence problem of a classifier [33, 76, 77], which is a known artifact of softmax activation. Instead, this method implicitly calibrates the softmax score through model training with a joint confidence loss. This method introduced known-unknown samples generated by a GAN into training and jointly trains the classifier and the GAN model with an integrated loss (i.e., the joint confidence loss) that includes cross-entropy loss, original GAN loss, and Kullback–Leibler (KL) loss between the softmax output and the uniform distribution. The cross-entropy loss item is the basic item of a classifier. The GAN loss is utilized to generate the most effective known-unknown samples. The KL loss forces the open set samples to be less confident. Again, to the best of our knowledge, this has not been adapted to traffic classification data.

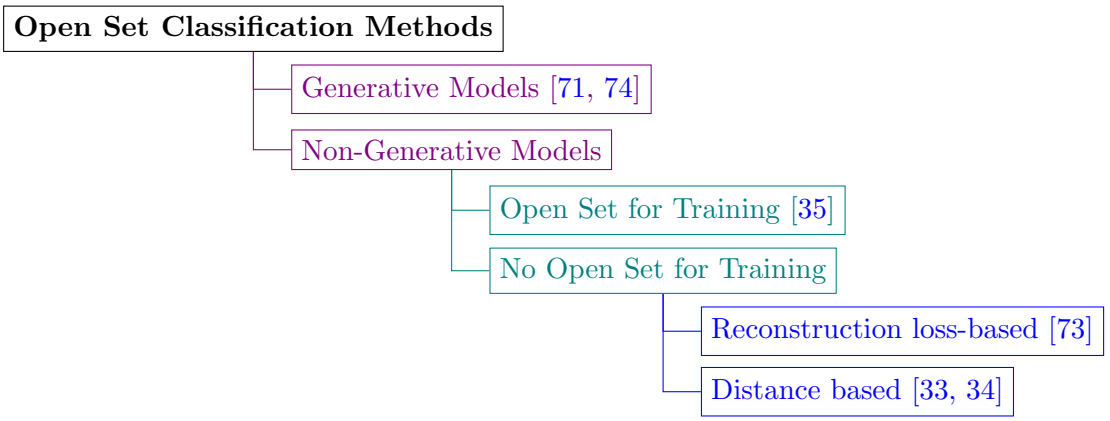
In 2017, Ge et al. proposed *G-OpenMax* [71], which extended OpenMax by using Generative Adversarial Networks (GANs) to synthesize samples from the open set. More specifically, the authors used samples generated from mixture distributions of known classes in the latent space of the GAN, which leads to plausible representation with respect to the known class domain. Such explicit representation of unknown classes enables the classifier to locate the decision margin with the knowledge of both known and unknown samples, thus resulting in better results.

The literature on open set classification also includes methods that operate under the assumption that samples from a subset of unknown classes (open set) which are representative of the entire open set are available during model training. The background

class method we discussed earlier was a naive method that followed this approach.

Methods using the open set for training : Dhamija et al. [35] proposed using *Entropic Open-Set Loss* which aims to produce equal softmax scores for all classes for open set samples. The authors had also observed that open set samples generally have lower feature magnitudes and proposed *Objectosphere Loss*, which includes Entropic Open-Set Loss while additionally forcing open set samples to have lower feature magnitudes. To this end, the authors assumed that a small subset of the open set (known unknowns) is available at model training and used Objectoshpere loss during model training to effectively identify open set samples.

In the diagram below, we visualize a summary of key open set methods we have discussed in this section.



As discussed previously, to the best of our knowledge, all work that handled open set in the network traffic domain depended on naive methods while all novel methods of open set classification were proposed targeting computer vision domain. In our research, we explore the adaptability of such methods to the network traffic datasets and evaluate their performance using several encrypted network traffic datasets.

2.3 Understanding CNNs

As discussed before, despite their state-of-the-art performance, deep learning models are often considered black-box models as the model provides very little information

regarding their decision-making procedure. Understanding this decision-making process of a deep learning model is very important due to many reasons such as verifying the model is doing what it is supposed to be doing before being deployed in mission-critical applications, identifying possible weaknesses in the model that can be exploited by adversaries, and so on. When studying previous literature on understanding the inner working of deep learning models, we noted that most work had been done related to 2D CNNs, which are extensively used in computer vision applications. In this section, we present a few key works that attempted to understand the behavior of deep learning models.

2.3.1 Retrieving inputs that maximally activate a neuron

Erhan et al. [29] proposed that a maximal activation of a neuron is caused by the pattern of the input it is most sensitive to. Accordingly, the authors suggested that finding and visualizing inputs from the dataset that causes maximal activation of a given neuron of a CNN can help understand what patterns the unit is looking for. Girshick et al. [78] followed this approach and fed a large set of images to an AlexNet and used the inputs that corresponded to the highest activations of a selected filter to visualize the features in the images that the filter is sensitive to. For example, some filters in the 5th pooling layer were identified to be sensitive to concepts such as people or material properties. Nonetheless, this method requires a manual inspection to identify common patterns in selected inputs.

2.3.2 Gradient-based approaches

To overcome manual inspection, Erhan et al. [29] proposed that rather than using existing inputs from the dataset, gradient ascent can be used to generate an input that maximizes the activation of a given unit of a CNN. This method feeds a random noisy sample into the network and the gradient vector of the selected unit is back-propagated back to the input image. This process is repeated to generate an input sample that maximizes the activation. Simonyan et al. [79] used gradient ascent to visualize class models (input image which maximizes the score for a selected class) learned by CNNs. Zeialer et al. [30] and Dosovitskiy et al. [80] demonstrated similar ideas using Deconvolutions.

2.3.3 Occluding parts of the input

Zeiler et al. [30] suggested that if covering up a certain portion of an image causes significant changes to the output of a neuron of a deep learning model, we can consider that the neuron is sensitive to the content in the covered part of the image. Based on this idea, Zeiler et al. [30] demonstrated the capability of an image classifier CNN to localize the object of interest inside an image by gradually covering portions of the input image with a grey square and monitoring the resulting class scores. It was observed that the occlusion of the object of interest caused a significant reduction in the class scores of the corresponding object, as well as notable reductions in activations of neurons that were previously identified as sensitive to the occluded object. In Figure 2.2 we demonstrate this idea using a pre-trained VGG19[1] model trained on Imagenet. We iteratively covered 12x12 pixel areas of Figure 2.2a before it was fed to the model and observed the class score for its original label *flamingo*.[2] We show the result in Figure 2.2b as a heat map. We observe in Figure 2.2b that the class score is only affected when sections of the flamingo is covered, which reveals that the decision of the classifier is mostly influenced by the target object in the figure.

(A) Original image (B) Occlusion result

FIGURE 2.2: Sample result for occluding parts of image

We observe that most of the prior work on understanding the underlying behavior of deep learning models was done in the domain of computer vision, focusing on 2D convolutional models. Motivated by these recent work, in this chapter, we methodically dissect several traffic fingerprinting CNN architectures to understand the reasons behind their performance and the patterns they look for. Such analysis will not only shed light on understanding why CNNs are highly effective in traffic fingerprinting but also will be helpful in designing more resilient defenses against traffic fingerprinting.

[1] `https://keras.io/api/applications/vgg/`
[2] `https://github.com/saketd403/Visualizing-and-Understanding-Convolutional-neural-networks`

2.4 Defenses against Traffic Analysis Attacks

As discussed earlier, end-to-end encryption does not conceal distinctive features of encrypted content such as the size, timing and direction of packets, which allows an eavesdropper to infer information about the encrypted content. The feasibility of such attacks led to a body of work proposing mitigation methods. The fundamental idea behind such countermeasures is to hide statistical properties of encrypted traffic such as size, time and directional information of packets, by adding dummy packets and/or packet delays to the traffic flows before encryption. Then the protection provided by the defense can be measured by classification accuracy over defended traces. It should be noted that high classification accuracy of a model trained on defended data would reflect better effectiveness of the defense as such a model assumes that the attacker even has access to defended data, which is highly favorable to the attacker. Two important parameters that define the efficiency of countermeasures are data overhead and timing overhead (latency). The data overhead of a defense refers to the percentage of additional dummy data added to the traffic flow with respect to the amount of data of the original (undefended) message. Similarly, timing overhead can be defined as the percentage of the extra amount of time required to complete the data transmission compared to the original message.

2.4.1 Distribution-based countermeasures

Most of the early work on defenses falls under this category. Dyer et al. proposed BuFLO [20], which pads/fragments all packets to a fixed length and sends packets at fixed intervals, which incurs large data and timing overheads. CS-BuFLO [21] and Tamaraw [22] are two extensions to BuFLO that attempted to improve its data overheads. Another defense Walkie-Talkie [81], requires the browser to communicate with servers in half-duplex mode and adds dummy packets and delays to create confusions between traces. All of these defenses were implemented as defenses against website fingerprinting over `Tor`. Most of these methods still incurred relatively high overheads or had limitations in practical implementation (such as Walkie-Talkie). A more recent work [82] explored how universal adversarial perturbations can be used to defend against website fingerprinting attacks and achieved relatively better security with reasonable data overheads.

2.4.2 Padding countermeasures

More recent attempts at defending against website fingerprinting belong to this category. WTF-PAD [23], is a defense nominated to be implemented on `Tor` network and uses adaptive padding to reduce the possibility of website fingerprinting with reasonable data overheads. However, it was found ineffective against recent attacks like [14]. FRONT [24], which can be considered as the current state-of-the-art defense against website fingerprinting over `Tor`, focuses on adding more padding at the beginning of a trace using a Rayleigh distribution. Compared to defenses against website fingerprinting, defenses against video fingerprinting is scarce. Zhang et al. [25] explored the possibility of using statistical privacy to defend against such attacks. The authors attempted using two ϵ-deferentially private mechanisms, Fourier Perturbation Algorithm (FPA) and d*-private mechanism. However, the FPA method required the entire traffic trace to be known in advance, and the d*-private method needed large overheads to provide adequate privacy, which made it difficult to deploy either of the methods in real-world applications. In a more recent work [18], Wang et al. presented a proof-of-concept defense against voice command fingerprinting in encrypted smart speaker traffic, using a combination of adaptive padding and d*-private mechanisms which reduced the attacker accuracy by 69.89%.

When studying prior work on defenses, we observe that the majority of work has focused on defending against website fingerprinting attacks. Even with such defenses, we see room for further improvement, especially with respect to data overhead. Furthermore, to the best of our knowledge, we only discovered one work that proposed defenses against video stream fingerprinting [25], and as discussed above, it makes unrealistic assumptions that make deployment of such methods in real-world applications less practical. As a result, our research will mainly focus on attempting to bring down overheads of defenses against TA attacks while providing reasonable privacy under more realistic assumptions that do not hinder the deployability of such defenses in real-world applications.

Chapter 3

Traffic Analysis Attacks on DNS-over-HTTPS Traffic

Domain Name System (DNS) translates human-readable domain names to numerical IP addresses. When a user needs to access content from a particular (web) server on the internet using its URL, they first need to convert the human-readable URL into the IP address of the corresponding server. For this purpose, the user needs to communicate with a DNS server, which is a server on the internet that contains the mapping between URLs and IP addresses. Figure 3.1 demonstrates DNS resolving process of a user who needs to access *www.cnn.com*. First, the user's browser checks if it already has a saved entry for the required URL, and since it does not, send a *DNS request* to its corresponding recursive resolver requesting the IP address corresponding to *www.cnn.com*. Then the recursive resolver communicates with several levels of DNS servers such as root name servers and TLD name servers to obtain the IP address corresponding to *www.cnn.com* and sends it as a *DNS response* to the user. Finally, after receiving the corresponding IP address, the user sends an HTTP request to the CNN web server asking for the required content. In our work, we are not interested in the communications between the recursive resolver and other DNS servers. Our work primarily focuses on the traffic between the user and the recursive resolver (indicated in red in Figure 3.1).

Since the security of DNS communications was not a major concern at the time of its conception, DNS requests and responses have always been sent in plain text form. However, with the growth of the Internet and along with it, the interest in security

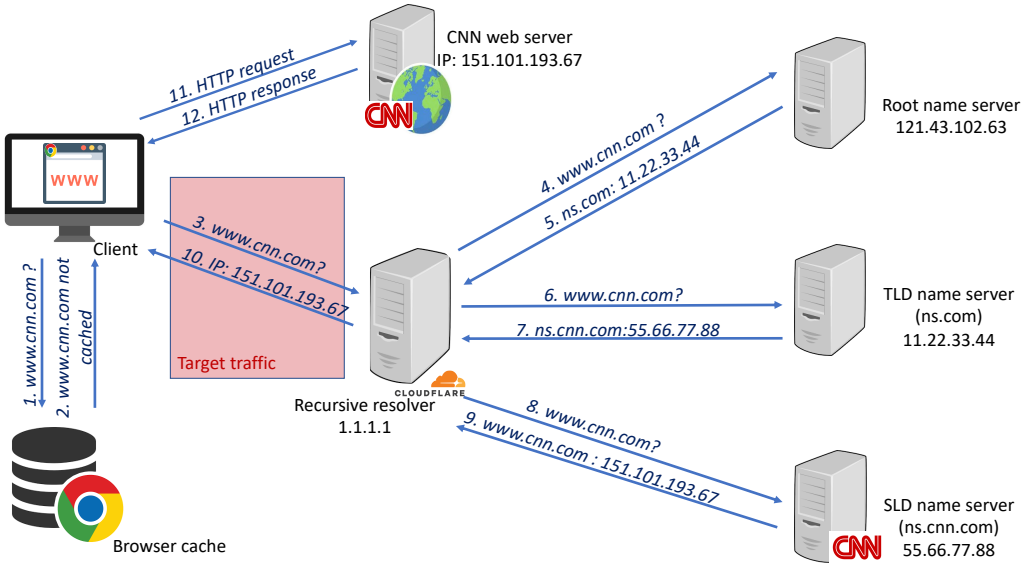


FIGURE 3.1: DNS resolving process

breaches of secure communications, resolving DNS requests over plain text that allows any eavesdropper to see and or edit the content of such communications became a major vulnerability. On one hand, a passive eavesdropper can use the URL requested by a user for a wide variety of applications that aims to monitor/filter Internet communications of that user. For example, a majority of enterprise networks that use filtering/access control to limit users from accessing certain websites, depend on DNS communications sent over plain text. Since the URLs requested by a user within their network are visible to an administrator, they can intervene and block the request from reaching the DNS resolver if the URL corresponds to any content the enterprise wishes to prevent their users from accessing. At the same time, an active eavesdropper can change the URL in the DNS request or the IP address in the DNS response to carry out a spoofing attack where the attacker can force the user to visit a website of attacker's choice. To minimize such vulnerabilities of plaintext DNS messages, several proposals such as DNS over TLS [48], DNS over QUIC [49], DNS over HTTPS [50] were proposed to ensure the privacy and confidentiality of DNS requests and responses. All of these solutions are yet to become mainstream. However, since 2020 DoH is getting popular due to the availability of public DoH resolvers from the likes of Google [83], Cloudflare [84], and Quad9 [85] and to the fact that major web browsers and operating systems started advocating the use of DoH [86–88].

DoH and other encrypted DNS variants: DNS over HTTPS (DoH) is defined

in RFC8484 [50]. As the name implies, in DoH the *stub resolver* (i.e the client) sends DNS requests using HTTPS to `port 443` of the *recursive DNS server*. The server makes the necessary requests on behalf of the client and sends the resolved IP address using the same encrypted channel. The DNS requests are either sent as HTTP `GET` or `POST` messages over TLS in either the DNS wire format or JSON format.

Oblivious DNS over HTTPS (ODoH) defined in RFC 9230 [89] was introduced to improve the privacy of DoH clients. In addition to the encryption provided by DoH, ODoH additionally requires a network proxy between clients and DoH servers which is expected to guarantee that no single entity other than the client, has access to both the DNS messages (plaintext form) and the client's IP address at the same time. From the point of view of an attacker using traffic analysis, ODoH behaves the same way as DoH and does not prevent such attacks. For example, the selected proxy will have access to the client's IP address as well as the encrypted DNS messages which can be used to carry out a traffic analysis attack to identify which URLs are requested by the particular client.

DNS over Transport Layer Security (DoT) [48] is another protocol that encrypts DNS messages. Unlike DoH, DoT has a dedicated port (i.e. `port 853`) which is now considered a drawback since a separate port makes it easier to block DoT requests. From a network management point of view, DoT is considered the preferred choice as it still allows some visibility of the DNS traffic in a local network. Nonetheless, due to the marginal privacy benefits provided by DoH by using `port 443`, it appears that DoH is more likely to be widely used than DoT in the coming years.

DNSSEC (Domain Name System Security Extension) [90] is another related term to DNS security. However, DNSSEC does not involve any encryption and as such, does not guarantee the privacy and confidentiality of DNS messages. Instead, DNSSEC allows the requester of a DNS record to verify whether the response they received was indeed from the authoritative name server responsible for the record using a public key infrastructure-based chain of trust. DNSSEC can be used in conjunction with DoH or DoT.

Apart from the above, there are a number of other proposals to ensure the privacy of DNS requests. Such solutions include DNSCrypt [91], DNSCurve [92], DNS over Datagram Transport Layer Security (DTLS) [93], and DNS over QUIC [49]. However,

as of now, none of these solutions are mainstream. Since we established that DoH would be the mainstream encrypted solution, the main focus of this will be on DoH traffic.

3.1 Traffic Analysis Attacks over Encrypted DNS Traffic

While the wide adoption of DoH is a positive step towards a more private and secure internet, the implications of DoH deployments need to be studied cautiously. First, DoH might give a false sense of privacy and security to Internet users. As discussed in previous sections, many internet protocols based on encrypting traffic flows are known to leak crucial information about online user activities such as videos they may be watching [16, 17], messenger app activities [65, 94], and visited websites [8, 14], through side-channels despite the network packets being end-to-end encrypted. As such, it is necessary to fully understand information leaks associated with DoH. Second, there are several legitimate and useful services that might be disrupted or even become obsolete with the adoption of DoH. For instance, many organizations and individuals are relying on monitoring unencrypted DNS messages for internal network management tasks such as content filtering (e.g., parental control filtering and blocking access to illegal content) and controlling malware distribution, and such solutions will not work if the users move to encrypted DNS (DoH [95]). Furthermore, similar to the use of Tor in illegal activities, there is a possibility that bad actors can leverage the obscurity provided by encrypted DNS to conduct cybercrime. Therefore, it is important to look into methods that can facilitate content filtering and monitoring services when DNS communications are encrypted.

Recent works [56, 57] showed that encrypted DNS traffic is prone to side-channel attacks and as a result, using the statistical properties of the DNS packet flows the websites that are being visited by the users can be predicted with high accuracy. This observation is of significant importance because such attacks undermine the privacy provided by encrypted DNS. It is also important to note that due to smaller packet sizes and shorter trace lengths of DNS communications compared to actual HTTPS traffic, both attacks could achieve high accuracies using light-weight traditional machine learning methods such as Random Forest and Adaboost classifiers but needed an additional feature engineering step. More importantly, both studies conducted the attacks in the form of post-analysis (i.e., assuming the full network trace of a DNS burst is available at the time of the attack). But from a more practical point of view, such as assumption limits

the usability of an attack. For example, a post-analysis type attack can only detect an event after its completion and can't prevent the event from completion which would be of immense help for network administrators for content filtering malware detection. A more realistic surveillance task requires a real-time operation and it is vital to have the ability to progressively make a prediction as the packets arrive which allows the detection of an event while it is happening and facilitates the prevention of such an event from completion as early as possible. This is one of the key focuses of our work. Furthermore, many existing traffic fingerprinting attack demonstrations operate under the closed set assumption. In real-world settings, the attacker's task is far more challenging because the attacker needs to identify the target website visits among a large number of other website visits happening in the background. Moreover, to the best of our knowledge, all previous work assumes that the attacker is aware of when an event starts happening and the attacker has access to the complete packet flow from the very beginning. For example, if we consider WF attacks over DoH, they assume that the attacker knows when the user starts requesting the website and therefore he can capture the complete packet trace from the beginning. However, this assumption is also less realistic and during our work, we also explore the efficiency of traffic analysis attacks when the traces are captured from an arbitrary point rather than the very beginning.

3.2 Experiment Design

This section describes the experiment setup and the design choices, the threat model, and provides a basic characterization of the dataset to explain some of our pre-processing steps.

3.2.1 Cloudflare implementation

We focus on the Cloudflare DoH resolver. We selected it because, at this stage, it appears to be the main DoH resolver in use. For instance, the popular web browser Firefox has started using DoH by default in the US, with rollouts planned for other countries [86].

DoH endpoint: Cloudflare DoH endpoint is hosted at IP addresses `1.1.1.1` and `1.0.0.1`. It is a recursive DNS resolver that resolves DNS requests coming from clients. In addition to DoH, the endpoint also supports unencrypted DNS as well as DoT.

DoH proxy: Many internet browsers and some operating systems already have native DoH support. In those browsers, the users have to simply enable the DoH feature and enter the Cloudflare DNS endpoint address (or any other preferred DoH server). To support endowments where no native DoH support is available, Cloudflare provides DoH clients that run as local DNS proxies; `cloudflared` and `dnscrypt-proxy`.[1] In that case the users can run one of the local proxies at `port 53` and redirect all the DNS queries there. The proxy will accept the unencrypted DNS requests, convert them to DoH, and get them resolved from the Cloudflare DoH endpoint as illustrated in Figure 3.2.

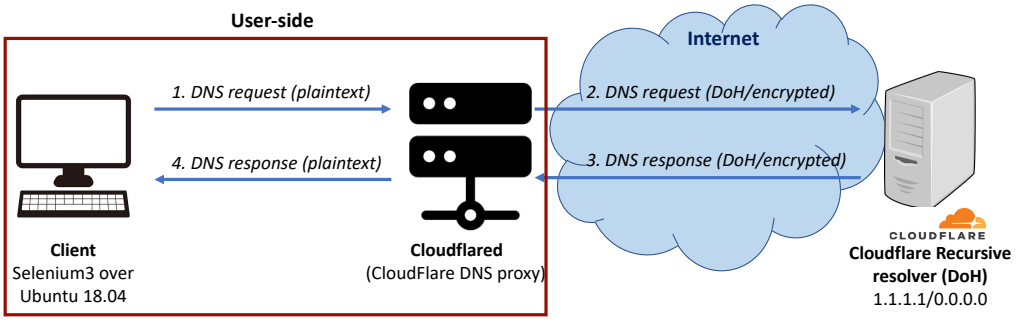


FIGURE 3.2: Cloudflare implementation

3.2.2 Threat model

We assume an attacker who can passively observe encrypted DoH messages between the target users and their DNS resolver. The attacker has a list of target websites and the goal is to take some action if a user is trying to visit a website from the target list.

We illustrate this attack scenario in Figure 3.3. A victim user communicates with a chosen DNS resolver over DoH to access '`google.com`' and the attacker eavesdropping on the encrypted communication is attempting to figure out whether the victim is trying to visit a website in the attacker's target list. If so, the attacker plans to take some action such as terminating the victim's connection.

[1] https://developers.cloudflare.com/1.1.1.1/dns-over-https/cloudflared-proxy/

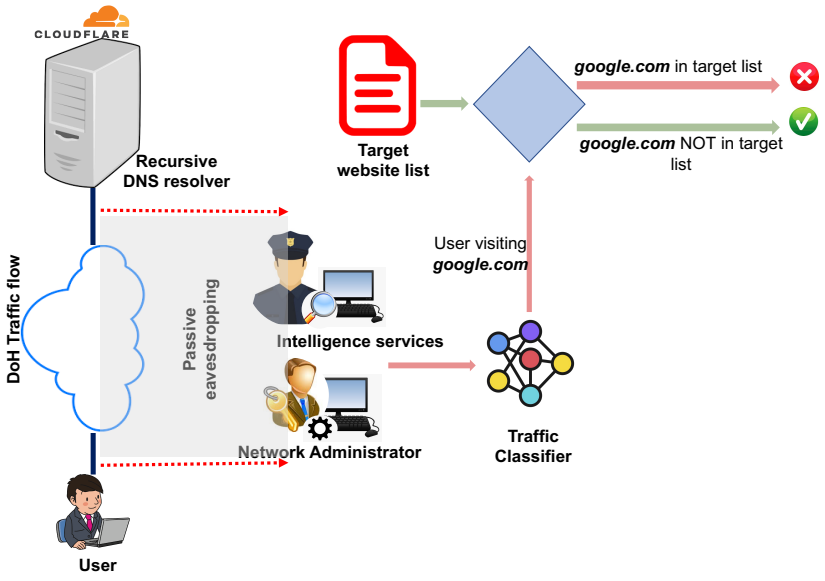


FIGURE 3.3: Threat model

In addition to accomplishing this base task, there are a few other desirable properties the attacker would prefer to have.

1. Identify the target website visits among all other possible website visits made by the user (handling open set)
2. Make the decision as quickly as possible (inline).
3. Since the attacker may not always capture user traffic when they initiate the website visit, be able to make the same decision, yet at arbitrary starting points.
4. Not to miss any target website visits even if that means having false positives from other website visits. (ideally, the attacker would like less false positives as possible)

3.2.3 Data collection

To collect data, we set up Ubuntu 18.04 virtual machines with `cloudflared` DoH proxy that forwards Operating System (OS) DNS queries to Cloudflare public resolvers in DoH format. We use Selenium3[2] to automatically visit a webpage from a list of websites, triggering DNS lookups for URLs associated with the page. Since the Cloudflared proxy is configured to use Cloudflare recursive resolver (at 1.1.1.1 or 0.0.0.0), all the traffic related to the DoH resolving process can be isolated using the resolver's IP address.

[2]https://www.selenium.dev/

In our main data collection (below *closed set* and *open set*), we restarted the browser in between webpage visits and flush the system cache to ensure that the DNS cache and in-memory browser cache do not affect the data collection. During each data collection, we let `cloudflared` proxy run continuously and never restarted it. Using this setup we collected the following datasets.

i) Closed set: In our closed set experiment, we collected traces from the top 201 webpages in Alexa's top million websites list on March 2018 [96]. We ran this experiment for 11 continuous weeks recording more than 1 million traces per website. We use the data collected during the first two weeks for the basic evaluations of our proposed model and use the rest to investigate the effects of changes in DoH traffic over time on traffic analysis.

ii) Open set: We selected 1,500 webpages from the top, middle, and bottom 500 webpages in Alexa's top million websites and removed the top-201 webpages as proposed in [57]. We use this dataset as a background class to simulate more realistic attacks where the target website visits are blended with all other visits. We collected DoH traces for these 1,286 webpages for 11 continuous weeks.

iii) User behavior emulation dataset: In order to investigate the effect of user behavior and its artifacts (such as caching) on DoH traffic fingerprinting, we emulate 25 different users using Daily Pageviews per Visitor Score (DPV_score) provided by the Alexa top websites list [96]. DPV_score refers to the daily unique pageviews per visitor on a given website and since it reflects how frequently a user would visit that website, we use this value to assign probabilities for a given user choosing to visit this website.

Each user has a list of 50 websites along with corresponding probabilities of the user visiting those websites, and the user keeps continuously browsing websites for 2 hours at a time, by picking websites according to the given probability values. The probability assigned to each website is proportional to its DPV_score. To emulate realistic user browsing patterns, we select the 50 websites assigned to each user from three categories.

- Top-10 websites of Alexa's top million websites list, representing websites that are popular among all the users.

TABLE 3.1: Summary of datasets

Dataset	**Number of Classes**	**Total Traces**	**Duration**
closed set	201[3]	1,138,229	11 Weeks
open set	1,286	295,233	11 Weeks
User behavior emulation	99[4]	121,695	9 Weeks

- Ten websites (excluding top 10) with DPV_score > 1, randomly picked based on their probability values to represent the websites that are specific to a user that are visited frequently, yet not as frequent as the top-10 websites.
- Thirty websites (excluding top-10) with DPV_score $= 1$, randomly picked based on their probability values to represent random websites a user may visit.

Accordingly, for the 9 weeks from the second week after the original data collection started, we allocated a separate machine to emulate 25 user behaviors. Each user was allowed to continuously browse websites from his list for 2 hours at a time before that user is suspended and the next user was allowed to browse without flushing the system DNS cache.

In Table 4.1 we show a summary of our datasets before applying any pre-processing steps.

3.2.4 Pre-processing

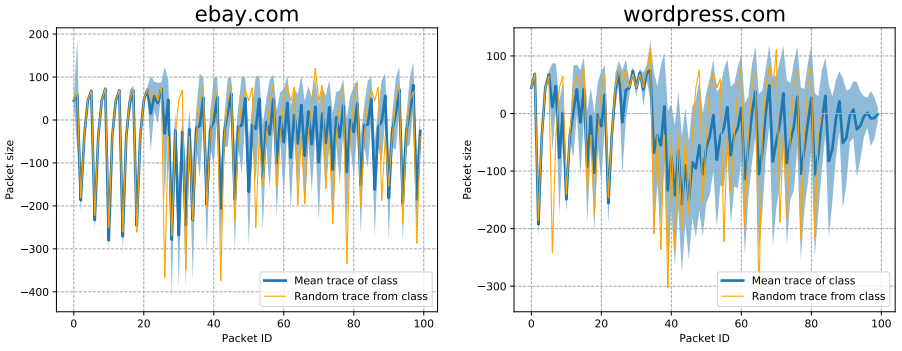
We represent each network trace by a sequence of L integers $< x_1, x_2, ..., x_L >$, where the absolute value of each integer corresponds to the size of the packet (i.e., TCP segment length) and the sign represents the direction (i.e., +'ve for queries and -'ve for responses). The length of the sequence L is the maximum length of a trace we observe in our data, which is 1,015. If a particular website visit had not generated a total of 1,015 packets, we pad the remainder of the sequence with zeros. After we remove unsuccessful page visits resulting in sequences with less than 4 packets, the final dataset contains 5,650 traces per class.

[3]We selected 201 classes to ensure the no. of classes after pre-processing reaches 150.

[4]This dataset does not contain all 201 classes because some classes had very low probabilities of being selected in the user model.

In Figure 3.4, we visualize a data sample and the mean of the class (i.e., the average of all data samples of a class) for two example websites, *ebay.com* and *craigslist.org*. In each figure, the orange line depicts a randomly picked sample trace from the corresponding class while the line in dark blue depicts the average of all traces belonging to the class. The area shaded in light blue corresponds to the +/- standard deviation from the mean.

For clarity, we show only the first 100 packets of traces. According to the figures, the two websites show very different DoH traffic patterns and also it is noticeable that the values of negative points are always much higher than that of positive points. This is expected since DNS responses generally have larger sizes compared to DNS requests.



(A) ebay.com (B) wordpress.com

FIGURE 3.4: Mean traffic trace for two sample websites

Next, we characterize our main dataset, i.e., **closed set** (first 2 weeks), to explain some of the data pre-possessing steps.

3.2.5 Pruning traces

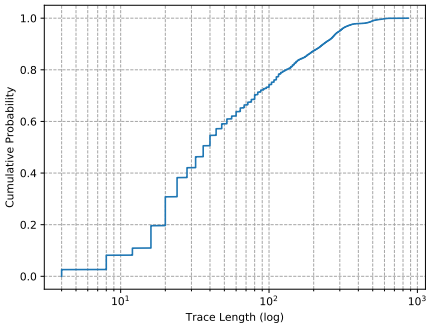


FIGURE 3.5: CDF of trace lengths for all websites

In Figure 3.5, we show the cumulative distribution function (CDF) of the trace lengths. Approximately 80% of the traces are less than length 100. Also, we found the majority

of the samples that have less than or equal to five packets were from 12 websites. Further investigations show that those websites were not fully browsed due to two main reasons.

- **No proper website or the website is not functional** - The top website list contained currently non-functional rogue websites such as `ntd.tv` and `nextoptim.com`.
- **Websites blocked by IP addresses** - Inside the enterprise network we conducted our experiments, adult and pornographic websites were blocked using an SSL proxy. In this setting, even if the first DoH request gets resolved the first HTTPS request is blocked by the SSL proxy, avoiding any further progress in loading the website.

We removed blocked/non-functional sites from our analysis.

3.2.6 Website-wise entropy

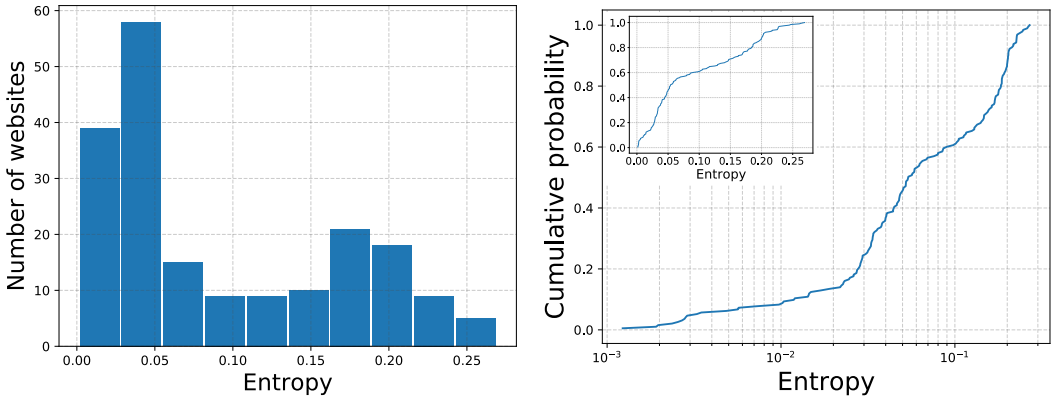
We represent each website with a discrete joint probability distribution $P_S(X, Y)$; where X represents the packet sequence number (0 to 100) and Y represents the packet length binned over $B = 447$ bins. For a given website, $P(X = x_i, Y = y_j)$ represents the probability of packet x_i belonging to bin y_j. Thus, the normalized Shannon entropy of a website, $H(S)$ ($[0, 1]$) can be calculated as Equation 3.1:

In Figure 3.6a and 3.6b, we show the histogram and the CDF of website-wise entropy. As can be seen, the entire probability mass of the entropy values is accumulated closer to zero indicating that there is less variance between different samples of the same website, which shows the feasibility of website fingerprinting over DoH traffic.

$$H(S) = \frac{1}{L * \log_2 B} \sum_{i=0}^{L} \sum_{j=0}^{B} -P(X = x_i, Y = y_j) \log_2[P(X = x_i, Y = y_j)] \tag{3.1}$$

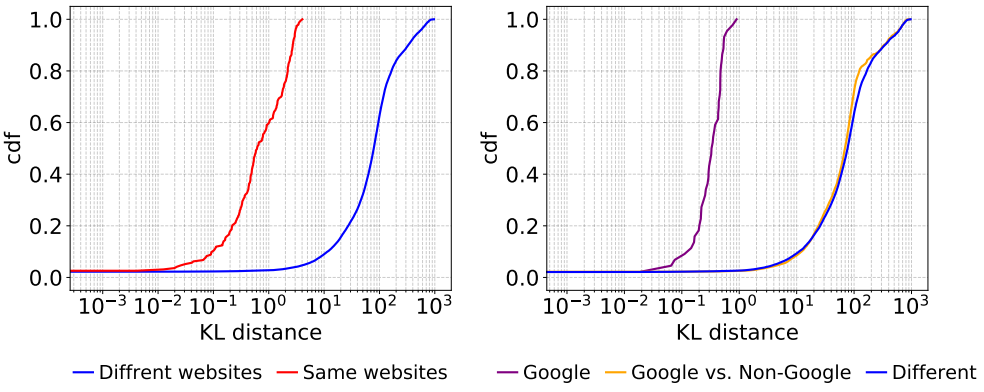
3.2.7 Removing website from same owners

Next, we investigate the distribution differences between websites using Kullback-Leibler divergence (KL divergence), which is a common metric used to compare probability distributions. We calculate the KL divergence between two websites S_1 and S_2 as defined in Equation 3.2.



(A) Histogram of entropy (B) CDF of entropy

FIGURE 3.6: Website-wise entropy distribution



(A) Same and different websites (B) Google vs. other websites

FIGURE 3.7: Empirical CDF of KL-Divergence

$$KL(S_1||S_2) = \sum_{i=0}^{L}\sum_{n=0}^{B} P_{S_1}(X = x_i, Y = y_j) * \log_2 \frac{P_{S_1}(X = x_i, Y = y_j)}{P_{S_2}(X = x_i, Y = y_j)} \tag{3.2}$$

Here, in $P_{S_i}(X, Y)$; X represents the packet sequence number (ranging from 0 to 100) and Y represents the packet length binned over 447 bins. For a given website S_i, $P_{S_i}(X = x_i, Y = y_j)$ represents the probability of packet x_i belonging to bin y_j, which can be calculated empirically. We provide further details of this joint probability distribution in Appendix A where we show the general low entropy of DoH flows.

In Figure 3.7a, we show the cumulative probability distribution of KL divergence between the same website and different websites. To calculate the KL divergence between the same class samples, we calculate two probability distributions by randomly splitting the samples of a single class into two. According to the figure, the KL divergences of the distributions representing the same class are significantly lower compared to the KL divergences between different websites. This indicates that the variance between different

samples of the same class is less than that between different samples of different classes, which can potentially facilitate traffic signatures.

Nonetheless, in Figure 3.7a we notice that there are different website pairs that result in lower KL divergence. Further analysis showed that these were various Google websites. To elaborate this further, in Figure 3.7b, we show the cumulative probability distribution of KL divergence between different URLs of Google domains (Eg: *google.co*, *google.ru*, *google.fr*), between URLs of Google and non-Google domains and different non-Google domain URLs. When comparing the graph in blue and the graph in purple, it can be seen that while only ¡1% of KL divergence values between all non-Google classes are less than 0.1, more than 60% of KL divergence values between different classes of Google domain are less than 0.4. Thus, in our experiments, we drop all other Google domains except for *google.com*. The observation of Google domains showing similar traffic patterns is observed in previous work as well [15, 56].

After dropping eight classes that correspond to non-functional and blocked websites, and dropping 43 other Google domains, the final dataset we use in subsequent analysis contains 150 classes.

3.3 Methodology

Previous attacks on encrypted DNS [56, 57] considered only fixed-sized inputs and operated over the full trace length. In such attacks, the attackers can make useful inferences only after the event has happened. A more realistic attack requirement is to do real-time inferences about the websites a user visits whilst the event is happening and identifying target traffic flows in the mix of all the other DoH flows. This work introduces a novel attack using three key ideas;

i) LSTM-based variable length sequence modeling

ii) A novel meta-recognition-based open set traffic classification method

iii) Imbalanced sampling

3.3.1 Variable length LSTM model

A simple RNN architecture as illustrated in Figure 3.8, takes x_t as input at each time step t, and updates a hidden state variable h_t and produces an output y_t. Here, $h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$ and $y_t = W_{hy}h_t$ where W_{hh}, W_{xh}, and W_{hy} are weight matrices that will be learned during model training. This is repeated over many unrolled time steps to accept time series data. It is important to note that the weight matrices of the RNN layer are shared across all time steps and hence are independent of the length of the input. Thus, an RNN layer can handle inputs of varying lengths as it only needs to repeat the same RNN cell to match the input length. Similarly, the LSTM (Long Short Term Memory) layer [97] we use in this work, is a more complex version of a simple RNN capable of modeling long-term dependencies that also can handle variable length inputs.

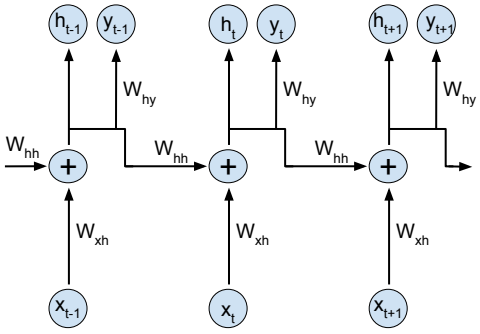


FIGURE 3.8: Simple RNN

Figure 3.9 shows the variable length LSTM model we use to demonstrate DoH fingerprinting attacks. It contains two stacked LSTM layers wrapped by dropout layers. The *length of the input layer, N* (i.e., the number of DoH packets to observe) is not fixed and varies according to the length of the input.

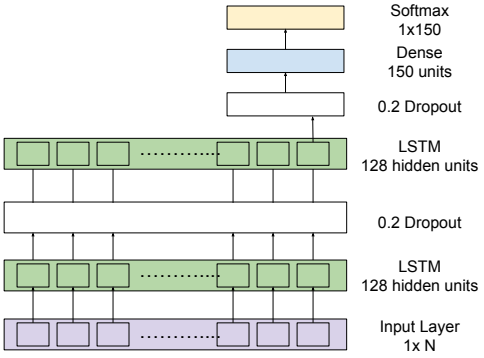


FIGURE 3.9: Variable Length LSTM Model

3.3.2 open set classification

The most frequently used ***closed set classification*** trains a classifier that can assign a given data sample to one of the classes from a known set with some confidence probability. In the context of website fingerprinting, this means the attacker is operating under the strong assumption that all the possible websites a target user may be visiting are known upfront, and samples from all those websites were available during training time. On many occasions, this is not a realistic assumption since the target user can visit any random website while the attacker is interested in knowing whether a user visits a website from a target list (cf. Section 3.2.2). Thus, a more realistic attack requires the ability to handle *unknown classes*.

As discussed in Sections 1.5 and 2.2, ***open set classification*** [33, 98], aims to correctly classify traces from a set of target websites while effectively rejecting traces from websites outside the target set. We explore two existing methods (*Background class* and *OpenMax*) and propose a novel *Jaccard Similarity Index-based method* that offer different trade-offs when it comes to balancing true-positive and false-positive rates and training data requirements.

3.3.2.1 Background class

The background class method handles unknown classes using a single class *background class* to represent all unknown classes together and requires samples from a subset of the open set at training time (i.e. *known unknowns*).

3.3.2.2 OpenMax

Originally proposed by Bendale and Boult [33], for open set image classification. It is built on the intuition that the values from the penultimate layer (the layer before the softmax activation layer) of a deep neural network provide a distribution of how classes are related to each other as opposed to being independent per-class score estimates.

3.3.2.3 Jaccard Similarity Index (JSI) of top-n predictions

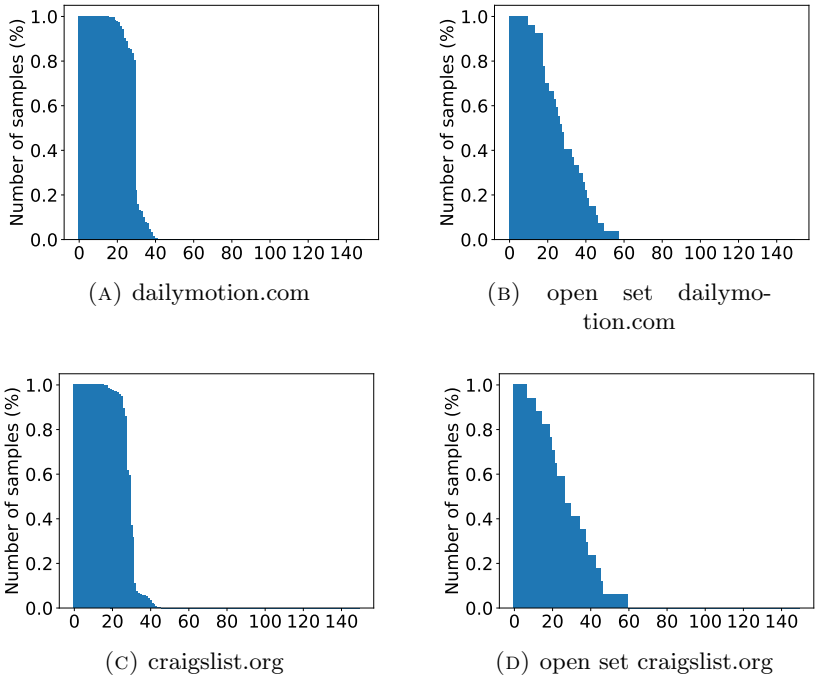
OpenMax has some inherent limitations when it comes to handling negative logit values and open set samples that have revised activation vectors that are much closer to the Mean Activation Vectors (MAVs) of closed set classes. Therefore, we propose a new meta-recognition open set classification solution using the *Jaccard Similarity Index* of the top-K predictions (K classes with highest softmax probabilities) from a closed set classifier. Similar to OpenMax, this approach also does not require samples from the open set during training time.

The intuition behind JSI method is that the top K predictions for a sample from a given class have to be stable (i.e., a given class is always closer to a known set of other classes). For open set samples, this closer set of classes is expected to change and that change could be used to separate out open set samples.

We illustrate this point in Figure 3.10 through some examples. We first used the closed set classifier described in Section 3.3.1 and built the set $S_K^{c_i}$ that consists of the union of top-K predicted classes for each correctly classified sample belonging to class c_i in the training set (we use $K = 30$ in this example). Then for each known class c_i, $\forall s_j \in S_K^{c_i}$, we calculate $Q_{c_i}^{s_j}$ which is the percentage of number of samples from c_i having the class s_j in their top-K predictions. The intuition behind $Q_{c_i}^{s_j}$ is that if class s_j is constantly appearing in the top-K predictions for class c_i, value of $Q_{c_i}^{s_j}$ will be close to 100%. For a given class c_i in the closed set, we hypothesize that there will be a limited number of classes with an $Q_{c_i}^{s_j}$ close to 100%.

In Figure 3.10a and Figure 3.10c we plot the $Q_{c_i}^{s_j}$ values for *dailymotion.com* and *craigslist.org* in descending order. For comparison, the same figures for open set samples classified as *dailymotion.com* and *craigslist.org* are shown in Figure 3.10b and Figure 3.10d, respectively. According to Figures 3.10a and 3.10c, we see that given a class the majority of its samples have a fixed set of classes (approximately 30 classes) present in their $S_K^{c_i}$ as opposed to samples from the open set which are classified as a given closed set class where the distribution is more dispersed (i.e. for each incorrect classified sample from the open set as class c_i has a different set of classes in top-K).

On the other hand, *Jaccard Similarity Index* (JSI) measures the similarity between two sets of data to see which elements are shared between the two sets and which elements



(A) dailymotion.com

(B) open set dailymotion.com

(C) craigslist.org

(D) open set craigslist.org

FIGURE 3.10: Composition of the set of top-30 predictions

are unique to a given set. More specifically, JSI value between two sets A and B is calculated as shown in Equation 3.3.

$$J(A, B) = \frac{A \cap B}{A \cup B} \tag{3.3}$$

Based on our observation of the stability of the top-k predictions of closed set classes, we propose to use the JSI measure to devise a new open set classification method named *JSI method*. Accordingly, the JSI method will use the JSI value between the S_K of an input sample and that of its predicted class to identify open set samples. We refer to the S_K set of closed set class as its *Mode Prediction Vector (MPV)* which is defined as given in Equation 3.4 and calculated as shown in Equation 3.5. Here $MPV_K^{c_i}$ refers to the MPV of class c_i with respect to top-K predictions, $Mo()$ refers to the statistical mode, and J^{c_i} is the number of correctly classified samples from class c_i. We use the *statistical mode* for this calculation instead of using mean or median as we are interested in the frequency of a class appearing in the S_K of a sample rather than the actual numerical value of the class label.

$$MPV_K^{c_i} = \{\{c_i\} \cup \{K-1 \text{ closest class to } c_i\}\} \tag{3.4}$$

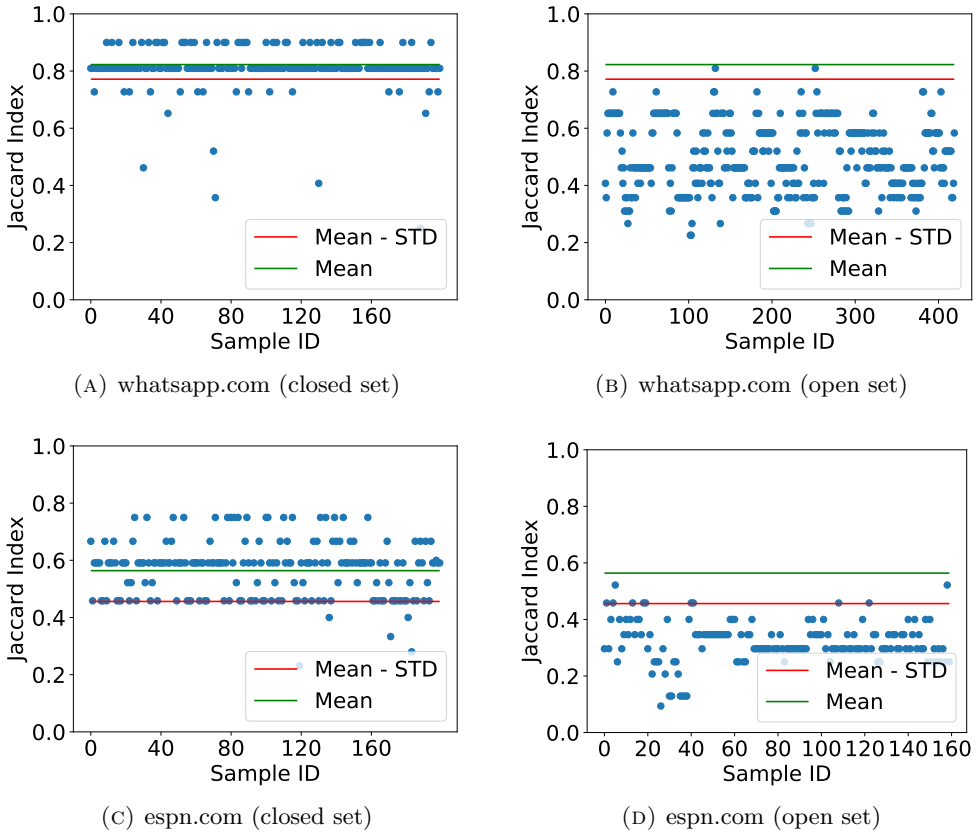
$$MPV_K^{c_i} = \{l_k | l_k = Mo(S_K(x_{c_i}^j)[k]), k \in [1, K], j \in [1, J^{c_i}]\} \tag{3.5}$$

Next, we use an example to demonstrate how thresholding on the JSI value between a sample and the MPV of its predicted class can be used to identify open set samples as described above. In Figure 3.11a and 3.11c we plot JSI values of all samples in the validation set for two example classes; *whatsapp.com* and *espn.com* for $K = 30$. For comparison, in Figure 3.11b and 3.11d we plot the JSI values of the open set samples that are labeled as the corresponding closed set class by the closed set classifier. The horizontal green and red lines mark the *mean* and *mean - standard deviation* of closed set training samples of the corresponding class.

In Figure 3.11a and Figure 3.11b, we observe that the JSI between the MPV of *whatsapp.com* and a sample trace from the same website has a mean value of approximately 0.81 while almost all open set samples labeled as *whatsapp.com* have JSI values below 0.8. Similarly, the mean value of JSI for a sample trace from *espn.com* is approximately 0.58 while almost all open set samples labeled as *espn.com* have JSI values below 0.5. We observed similar trends with other classes as well, suggesting that by using an appropriate threshold value on the JSI value, open set samples can be effectively rejected. After experimenting with different threshold values, we used class-wise *(mean-standard deviation)* of JSI values of the validation set as the open set rejection threshold. As can be seen in both figures, using *mean* as the rejection threshold would still reject a significant number of actual closed set samples compared to *(mean-standard deviation)* as the threshold.

Finally, we summarize the procedure of using JSI method for open set classification as an algorithm. The two main goals during the training phase of JSI method are to identify a set $MPV_K^{c_i}$ for each of the known class and to determine the rejection threshold which represents the extent to which the set S_K of an input sample can vary from $MPV_K^{c_i}$ of its actual predicted class c_i, where S_K is the set of top-K predicted classes of the input sample.

Algorithm 1 describes the training phase of the method given K is the number of top predictions to consider (we use $K = 30$), N is the number of known closed set classes, and J^{c_i} is the number of correctly classified training samples from class c_i.



(A) whatsapp.com (closed set)

(B) whatsapp.com (open set)

(C) espn.com (closed set)

(D) espn.com (open set)

FIGURE 3.11: Distribution of JSI at input length 10

Algorithm 1: JSI method: Training

Require: Softmax layer output from closed set classifier for input x:
$v(x) = v_1(x).....v_N(x)$

1 Define $S_K(x) = [argsort(v(x))][1:K]$

2 For each known class c_i, calculate $S_K(x^j_{c_i})$ for each correctly classified training sample j from class c_i.

3 **for** $c_i = 1.....N$ **do**

4 Compute *Mode Prediction Vector*
$MPV^{c_i}_K = \{l_k | l_k = Mo(S_K(x^j_{c_i})[k]), k \in [1,K], j \in [1, J^{c_i}]\}$

5 For each known class c_i, let $JSI_K(x^j_{c_i}) = \frac{\{MPV^{c_i}_K \cap S_K(x^j_{c_i})\}}{\{MPV^{c_i}_K \cup S_K(x^j_{c_i}\}}$ for all samples from class c_i in the validation set.

6 **for** $c_i = 1.....N$ **do**

7 Compute rejection threshold $JSI\_T^{c_i}_K = mean(JSI_K(x^j_{c_i})) - std(JSI_K(x^j_{c_i}))$

8 **return** $MPV^{c_i}_K$ *and* $JSI\_T^{c_i}_K$

At inference time, the JSI method follows Algorithm 2 where the main goal is to determine whether the deviation of the JSI value of an input is within the predetermined range of the predicted class and if not to reject the sample as coming from the open set. The step given in lines 7 and 8 of Algorithm 2 is optional and used to further optimize the performance of the JSI method on closed set samples by directly accepting

Algorithm 2: JSI method: Inference

Require: Softmax layer output from closed set classifier for input x:
$v(x) = v_1(x).....v_N(x)$

Require: $MPV_K^{c_i}$ and $JSI\_T_K^{c_i}$ for $c_i = 1.....N$

Predicted label $\tilde{c}(x) = argmax(v(x))$

SoftMax probability of $\tilde{c}(x)$, $P(\tilde{c}(x)) = max(v(x))$

$S_K(x) = [argsort(v(x))][1:K]$

$JSI(x) = \frac{\{MPV_K^{\tilde{c}(x)} \cap S_K(x)\}}{\{MPV_K^{\tilde{c}(x)} \cup S_K(x)\}}$

if $JSI(x) \geqslant JSI\_T_K^{\tilde{c}(x)}$ **then**

  Output $\tilde{c}(x)$

else if $P(\tilde{c}(x)) \geqslant \epsilon$ **then**

  Output $\tilde{c}(x)$

else

  Reject x as open set sample

the prediction from the closed set classifier for samples that the closed set classifier is highly confident in labeling (i.e. the Softmax score for the predicted class is very close to 1). In our experiments, ϵ was set as .9999.

3.3.3 Imbalanced sampling

The variable length LSTM model proposed in Section 3.3.1 can be trained and tested on inputs of arbitrary length. We include several instances of the same sample at different selected lengths in to the training set so that the model can learn to identify traffic patterns in traces of different lengths. We feed more samples from shorter lengths to obtain higher accuracies at lower lengths. We noticed that including only a few input lengths from lengths greater than 10 is sufficient to get over 90% test set accuracy for all input lengths greater than 10. Therefore, we include samples with lengths $l \in L$ and $L = \{5, 6, 7, 8, 9, 10, 14, 20, 100\}$ in to the training set. We extract samples at each trace length l from the entire training set to build the final training set of size, $|L| * N * number\_of\_traces\_per\_class$.

3.3.4 Arbitrarily sampled segments

Capturing network traffic traces from beginning to end could be difficult in some situations. Almost all the existing work in traffic fingerprinting assumes that the start of the activity is known in traffic traces [14–16]. Nonetheless, a more realistic attack is to

consider whether an attacker can start from an arbitrary point of the traffic flow, observe packets for some time and make a prediction. Therefore, here we segment our training dataset with a sliding window of length w with $w-1$ overlap. If the trace length is not sufficient to fill a window, the rest of the window is filled with zeros. We tested with different w values as we discuss in Section 3.4.3.

3.4 Results

In this section, we present the results of the inline traffic fingerprinting attacks in both closed set and open set settings. In the following experiments, the closed set dataset comprises of traces from 150 classes with 900 traces from each class and the open set of 1,286 classes with 20 traces from each class (**cf.** *Closed set* and *open set* in Table 4.1). While we had more data samples that could be used to train the closed set LSTM model, we selected these values because increasing training samples had only diminishing returns yet large training times that are inherent to variants of RNNs [99].

3.4.1 Closed set classification

3.4.1.1 Experiment 1: Temporal analysis

Our primary closed set dataset was collected over 11 weeks. In this experiment, we used the data collected within the first two weeks to provide results over multiple training and test set splits based on time, to understand the generalization of the attack better.

First, we divided the total dataset from the first two weeks into three sub-datasets so that data from four consecutive days fall into one sub-dataset and a one-day gap was left between two datasets. For each sub-dataset, a training and test set was chosen such that the training set comprised 120 samples per class randomly picked from the first three days of each dataset, and the test set comprised of 65 samples per class randomly picked from the last day of each dataset.

Next, for each training dataset, we trained our model for 500 epochs using input lengths as explained in Section 3.3.3 and used its own test set as the validation set to choose the model with the lowest validation loss. Finally, we evaluated the trained model on its own test set as well as the other two test sets.

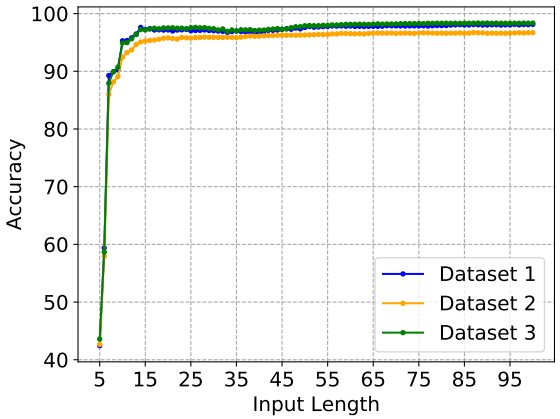


FIGURE 3.12: Overall accuracy at varying input lengths

In Figure 3.12, we show the accuracy of the three models trained on the three datasets, when tested on the combined test set of the other two datasets than its own at all input lengths. We observe that even though the model was trained only with a few input lengths, it yields correct classification at any input length greater than 10, with over 90% accuracy.

We show detailed cross-dataset results in Table 3.2. For brevity, we show results of only three lengths. According to columns 3, 4, and 5 of Table 3.2, all test scenarios of input lengths over 10 achieved over 90% accuracy, and at the full trace length of 100, the accuracy was in the range of ∼96%–99%. Noticeably, observing only the five packets of the trace could still result in ∼42%–44% accuracy. Observing only 10 packets gives an accuracy above 90%, which is close to that of observing the full trace. Finally, there was a gap of at least seven days between the training set of Dataset-1 and the test set of Dataset-3, and yet that scenario resulted in 90%-96% accuracy indicating the short-term invariance of DoH patterns.

Next, we used the same three small datasets but a different splitting strategy. We randomly picked 120 and 65 traces per class from each dataset as training and test sets. We show the results in the last three columns of Table 3.2. They are similar to the previous results, further indicating that there are no major changes to DoH patterns within a few days. Therefore, for all the subsequent experiments, we randomly split the total closed set dataset into training, validation, and test sets, with 500, 200, and 200 traces per class, respectively.

TABLE 3.2: Temporal Analysis

		Trained On					
		Split based on time			Random split		
Input length	**Tested On**	**Dataset 1**	**Dataset 2**	**Dataset 3**	**Dataset 1**	**Dataset 2**	**Dataset 3**
5	**Dataset-1**	42.81%	41.99%	43.32%	39.75%	41.43%	41.34%
	Dataset-2	42.42%	42.45%	43.84%	39.54%	42.30%	40.93%
	Dataset-3	42.43%	42.34%	43.97%	39.94%	42.02%	39.50%
10	**Dataset-1**	96.03%	92.36%	94.41%	94.85%	94.63%	94.84%
	Dataset-2	95.76%	93.04%	95.40%	94.59%	95.74%	93.80%
	Dataset-3	94.80%	91.92%	96.27%	93.67%	94.72%	91.16%
100	**Dataset-1**	99.36%	96.65%	97.96%	99.17%	98.54%	98.91%
	Dataset-2	98.73%	97.16%	98.76%	98.93%	99.32%	98.43%
	Dataset-3	97.52%	96.58%	99.48%	98.17 %	98.48%	99.30%

3.4.1.2 Experiment 2: closed set classifier from full dataset

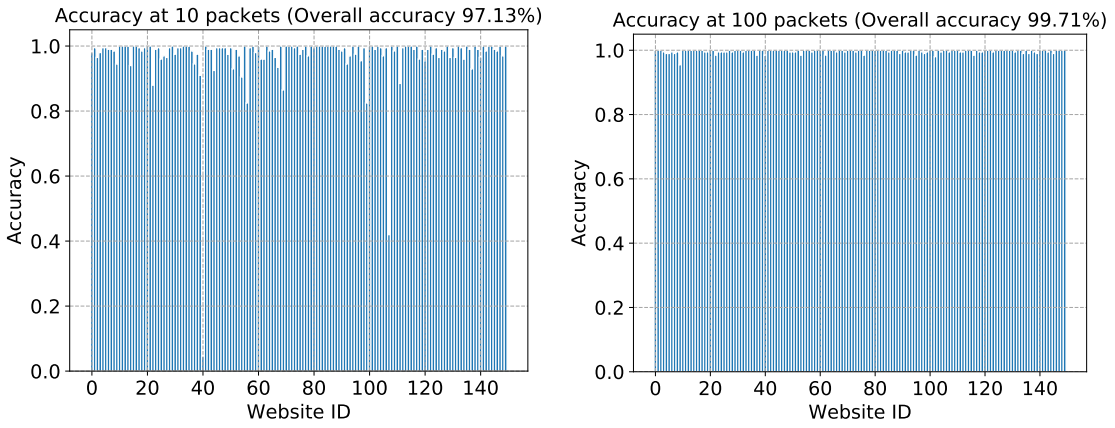
Next, we built a closed set classifier from the above-described randomly split data. First, we trained a variable length LSTM model and select the model having the lowest validation loss. We used the same input lengths as described in Section 3.3.3.

In Figure 3.13 we plot the class-wise test set accuracy at input lengths 10 and 100. Though there are more classes with lower accuracy at input length 10, their performance improves as the input length increases.

We also studied confusion matrices for the model at various input lengths and observed that for all classes that record accuracy below 80% for input lengths higher than 10, the model labels all the incorrectly classified traces into one specific class. Such errors work in favor of the attacker. For example, in Figure 3.14 we report the top-2 (i.e. correct label is within the first two predictions) and top-5 prediction accuracies along with the top-1 prediction accuracy. We observe that even when using just 5 packets, the attack accuracy can be increased to 63% when considering top-2 predictions; considering the top-5 predictions, it further improves to 89%.

3.4.1.3 Experiment 3: Concept drift results

We next investigated the performance of our LSTM model trained in Section 3.4.1.2 on a test set captured at a much later point in time than the original training set (i.e.



(A) Length 10 (B) Length 100

FIGURE 3.13: closed set accuracy of the full dataset at different trace lengths

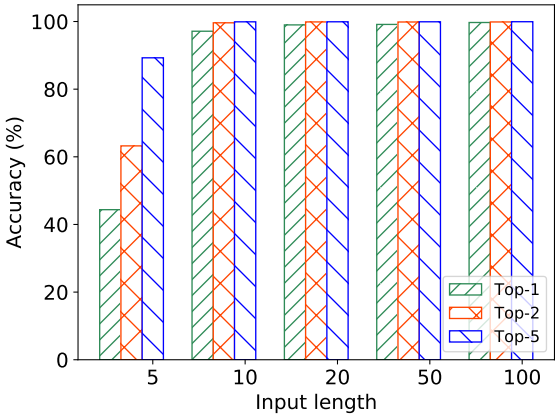


FIGURE 3.14: closed set top-n results

concept drift). We extracted four test sets with 30,000 traces (200 traces per class) in each from the *closed set* dataset, which was collected 2, 4, 6, and 8 weeks after the collection of the original dataset. We show the accuracy of these datasets on the model trained in Section 3.4.1.2 in Table 3.3. When comparing the rest of the columns with the second column (0th-week test set), we observe that there is no significant drop in accuracy even if the test set was collected 8 weeks after the training set.

TABLE 3.3: Accuracy: Concept drift

	Accuracy (%)				
Input length	**0 weeks**	**2 weeks**	**4 weeks**	**6 weeks**	**8 weeks**
5	44.37	43.20	42.92	42.43	42.06
10	97.13	95.59	94.69	93.14	91.44
20	99.00	97.55	96.87	96.20	94.69
50	99.14	97.59	96.73	96.26	94.52
100	99.71	98.50	98.18	97.54	95.94

3.4.2 open set Classification

In this section, we explore the ability of the variable length LSTM attack model to handle samples from unknown classes, using three methods; *Background Class*, *Openmax*, and the proposed *JSI* method as explained in Section 3.3.2.

For all open set experiments, we used the combination of our closed set dataset used in Section 3.4.1.2 and the entire *open set* in Table 4.1. For the background class method only, we added randomly chosen 500 and 200 traces from the open set to the training and validation sets, respectively. Each method was evaluated on a test set that combined the test set from Section 3.4.1.2 and the entire open set. Note that for the background class method, we made sure that open set samples used for training and validation were removed from the test set.

For performance evaluation, we used *closed set accuracy*, *open set accuracy*, and *F_score*. closed set accuracy refers to the percentage number of correctly classified samples from known classes, while open set accuracy refers to the percentage number of samples from unknown classes that are rejected as open set samples. We calculate the F_score under the open set as defined by Júnior et al. [100], where samples from the open set that are correctly identified by the model are not considered as true positives because it does not make sense to consider all unknown classes as a single positive class when the training set does not have samples from all possible unknown classes.

3.4.2.1 Background Class

We trained a variable-length LSTM for 151 classes (150 known classes and the background class) and selected the model with the lowest validation loss. To negate any bias due to random sampling from the *open set*, we repeated this experiment for five different splits from the *open set* for training, validation, and test sets. We report the average accuracy and the standard deviations at several input lengths in Table 3.4.

We notice that the closed set performance of the background class model remained almost the same as that of the closed set model in Section 3.4.1.2 and the performance of the model on the open set is only in ∼20%–58.7%. We further note that the random split of the *open set* does not significantly affect the performance. Finally, we highlight

TABLE 3.4: open set results: Previous Methods

	Background class (LSTM)			OpenMax		
	F_score	**Mean Accuracy (%)**		**F_score**	**Mean Accuracy (%)**	
Input Length		**closed set**	**open set**		**closed set**	**open set**
5	0.42±0.00	42.93± 0.53	20.00± 2.99	0.61	39.68	45.15
10	0.96±0.00	96.75± 0.17	49.522± 1.97	0.97	86.11	61.32
20	0.98±0.00	98.75± 0.06	57.47± 2.65	0.98	87.48	62.93
50	0.99±0.00	99.40± 0.34	58.31± 3.06	0.98	87.52	60.73
100	0.99±0.00	99.72± 0.02	58.67± 2.32	0.98	88.67	59.23

TABLE 3.5: open set results: JSI methods

	JSI without SoftMax thresholding			JSI with SoftMax thresholding		
	F_score	**Mean Accuracy (%)**		**F_score**	**Mean Accuracy (%)**	
Input Length		**closed set**	**open set**		**closed set**	**open set**
5	0.42	41.66	44.11	0.42	41.76	42.26
10	0.81	81.30	80.48	0.89	89.67	61.66
20	0.83	83.00	76.33	0.97	97.44	57.97
50	0.84	83.95	77.83	0.97	97.27	57.51
100	0.85	84.64	75.52	0.98	98.48	56.58

that the model is not exposed to all unknown classes at training time since the maximum number of classes that could be included in the training set is 500 (out of 1,286 classes).

3.4.2.2 OpenMax

Next we used the same variable length LSTM model from Section 3.4.1.2 as the base model for OpenMax (with η=25). We report the test results in Table 3.4. While OpenMax achieves closed set accuracy in the range ∼86.11%–88.67% at all input lengths greater than 10, the open set accuracy remains in ∼59.23%–61.32%. When compared with the background class results, OpenMax underperforms in the closed set but performs slightly better with the open set.

3.4.2.3 JSI method

Next, we use the same LSTM model to evaluate the JSI method. In Table 3.5 we show the results of JSI method on the test set based on Algorithm 2 with and without softmax thresholding. In comparison to OpenMax, even though the JSI method without softmax thresholding has similar performance on the closed set, it has better performance on the open set. JSI method without softmax thresholding has approximately similar performance as the background class method despite not using open set samples during training.

When we compare the performance of JSI method with and without softmax thresholding, softmax thresholding on top of JSI method improves the performance in the closed set with a decrease in the open set results. For example, at input length 20, softmax thresholding increases the closed set accuracy by 17.4% while reducing the open set accuracy by 24.1%.

In summary, our proposed JSI method enables the attacker to filter out traffic traces that are out of the target list without the requirement of having samples of such traces at training time. The attacker can decide whether to use softmax thresholding or not based on the attack requirements. For example, if the attacker does not want to miss any website visit from the target list, but can accommodate some false positives, they can use JSI with softmax thresholding. Otherwise, if the attacker needs a balanced accuracy between the closed set and open set, they can use JSI with softmax thresholding, which gives accuracies in the range of 81%–84% and 75%–80% for closed set and open set, respectively, for trace lengths>10.

3.4.3 Arbitrary starting points

We next evaluate the attacker's ability to fingerprint traffic traces when the start of a DoH traffic flow is unknown by segmenting the closed set dataset using a window size of 20 as described in Section in 3.3.4. We trained an LSTM model on the training set (1,909,239 samples) for 100 epochs and used the validation set (1,275,822 samples) loss to select the best model. When evaluated on the test set (1,280,861 samples), the model achieved an accuracy of 73.16% suggesting that it is indeed feasible to fingerprint websites starting from arbitrary points of a DoH trace. Here, we highlight that the test

set was completely disjoint from the training and validation sets since we do the sample separation before windowing. A similar attack using a window size of 10 achieved an accuracy of only 37.45%. Overall, the accuracy increases with the window length as the input gets closer to the complete trace.

3.4.4 User behavior emulation

To investigate the effect of user behavior (such as DNS caching) on traffic analysis attacks, we collected another dataset *User behavior emulation* dataset (**cf.** Table 4.1).

First we extracted a dataset from the *closed set* dataset (cf. Table 4.1) collected during the same period as *User behavior emulation* data to train a variable length LSTM on the it using the same train:valid:test split and procedure as in section 3.4.1.2. Then we compared the model performance on the test set extracted from closed set vs. *User behavior emulation* dataset as shown in Table 3.6 (no retraining), which shows a slight performance gap between the test sets with the highest performance gap of 7.34% occurring at input length 20.

Next, we check the possibility of training a classifier that can even handle possible caching by incorporating user behavior during training. Thus, we combined the dataset from the closed set and the *User behavior emulation* dataset used above and trained a new variable-length LSTM model. We have reported the accuracy of the test sets from *closed set* and *User behavior emulation* set separately in Table 3.6 (with retraining) which shows that the test sets from both datasets record similar accuracies on the model at all input lengths except 5. The overall result suggests that there is minimal effect on DoH fingerprinting due to the effects of user behavior. The better performance of the test set at length 5 on *User behavior emulation* set can be attributed to the lesser number of classes present in that set. As websites with lower DPV_scores have a lower probability of being visited, the user behavior emulation dataset had only 99 classes compared to the previous 150.

3.4.5 Confusion analysis

Despite the high overall performance, confusion matrices show that a few websites are not clearly distinguishable to the model and result in below 80% class-wise accuracies at

TABLE 3.6: Accuracy: User behavior emulation

		Accuracy at varying input length(%)				
Retrained	**Test-set**	**5**	**10**	**20**	**50**	**100**
No	closed set	43.31	96.67	98.63	99.42	99.53
	User emulation	40.14	90.49	91.39	93.03	93.06
Yes	closed set	42.07	95.45	98.18	98.88	99.42
	User emulation	65.82	94.76	95.87	96.32	96.89

shorter input lengths. For instance, at input length 10, 58% of *deviantart.com* samples gets confused as *tribunnews.com*.

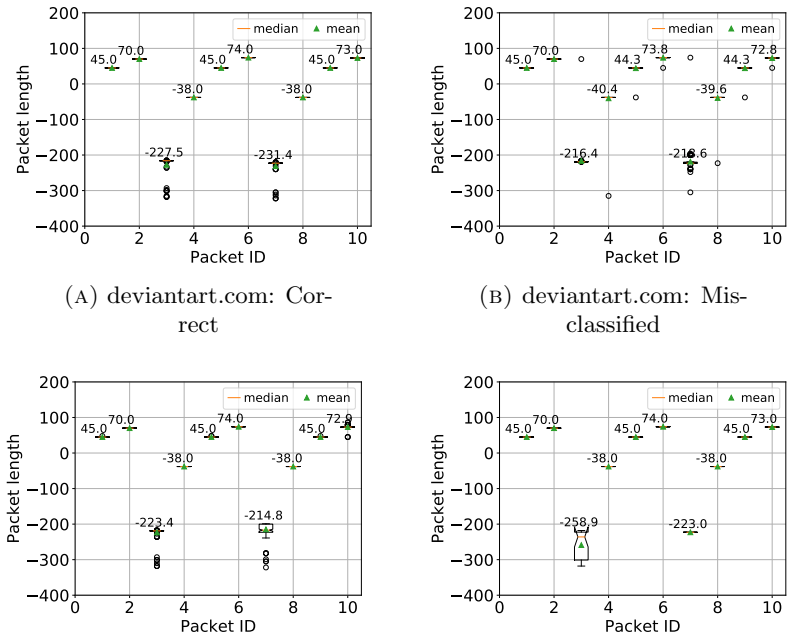
To further understand this, in Figure 3.15 we show the box plots of mean packet lengths at each trace position for correctly and mis-classified samples for *deviantart.com* and *tribunnews.com*.

When comparing the boxplots for correctly classified samples from *deviantart.com* (Figure 3.15a) and *tribunnews.com* (Figure 3.15c), we notice that, except for 3rd and 7th positions, the distributions of all others are similar. Thus, the model would mostly depend on the 3rd and 7th packets to distinguish between the two websites. In Figure 3.15a and Figure 3.15b, we see that the main difference between correctly and mis-classified samples is in 3rd and 7th packet. The mean value of 3rd packet in Figure 3.15b (216.5) is closer to that of Figure 3.15c (223.3) than that of Figure 3.15a (226.7). A similar pattern can be seen with the 7th packet too. We noticed similar trends among other classes with low accuracy.

3.4.6 Defense analysis

Multiple prior works on network traffic fingerprinting [22–24] have suggested using padding as a defense aiming to hide packet lengths. Thus, RFC8467 [53] recommends DNS clients padding queries to the closest multiple of 128 octets and DNS servers padding responses to a multiple of 468 octets.

Next, we evaluated the effectiveness of our model against padding by simulating defended traffic using a subset of our closed set dataset. We trained a variable length LSTM on the defended training set and report its results on the defended test set in Table 3.7 where *overhead* refers to the mean of the percentage amount of extra bytes added per trace. Since prior work [56, 57] has reported differences in padding between DNS resolvers and



(A) deviantart.com: Correct

(B) deviantart.com: Misclassified

(C) tribunnews.com: Correct

(D) tribunnews.com: Misclassified

FIGURE 3.15: Box plots (*deviantart.com* vs. *tribunnews.com*)

clients, we considered three different scenarios; only the queries are padded, only the responses are padded, and both queries and responses are padded.

TABLE 3.7: Defense analysis

Method	**Input length**	**Accuracy (%)**	**Overhead (%)**
Recommended padding (query and response)	10	3.31	213.62
	100	70.57	213.71
Recommended padding (query only)	10	80.03	50.01
	100	97.39	36.66
Recommended padding (response only)	10	76.86	163.58
	100	98.12	176.98

Table 3.7 shows how the recommended padding policy is effective only with both client and server padding. Even then, the model achieves 70% accuracy at input length 100. For all other padding combinations, the model achieves over 75% accuracy.

This result is important as we did not find any straightforward way of enabling padding on the browser side (also observed by [56, 57]), and some DoH resolvers did not pad even if clients requested[57]. Overall, we highlight that current DoH implementations lack padding in their stock configurations and, as such, do not deliver the full privacy benefits of DoH.

3.5 Discussion and Concluding Remarks

In this chapter, our main aim was to propose a practical website fingerprinting attack that enables inline website fingerprinting while handling the open set scenario at the same time. First, we showed that DoH traffic flows of websites have low entropies leaving them vulnerable to traffic analysis. Next, we presented an inline traffic analysis attack against DoH using a variable length LSTM model, which achieves over 96% accuracy by observing only the first 10 packets of a DoH flow. Then we extended our attack with a novel open set classification method which achieves 75%–80% accuracy on both closed set and open set. Then we also presented a more challenging but realistic scenario where the exact starting point of a trace is not known and showed that even when the trace is captured at an arbitrary point, a classifier can still achieve over 70% accuracy. Next, we showed that the effects of user behavior such as caching, can drop the accuracy of the original model, but that effect can easily be negated by retraining using cached traffic traces. Finally, we showed that padding as a defense is not effective as attacks can still achieve ∼70% accuracy.

We next discuss limitations and possible extensions to our work.

Browser and the Operating System - Our data collection process used Chrome as the browser and Ubuntu as the client OS. We speculate that it is possible for the DoH signature of a given website to change between browsers and OSs used when capturing traffic. However, we note that the attacker can include samples from various browser and OS combinations during the training phase to make the attack model independent from the browser/OS setting of the capturing procedure. More importantly, user-specific configurations such as the use of ad-blockers can make the attack more challenging.

Handling concept drift - Though we showed the invariance of DoH patterns over a period of a few weeks, in the long run the DoH patterns can change due to changes in a given web pages' dynamic content (e.g., advertisements). Therefore, the attackers may need to retrain models periodically to capture such changes. For that, the attacker can leverage the ideas of transfer learning [101] or online learning [102].

Open set classification - One of our major contributions is introducing a new open set classification method that enables identifying target websites amid background traffic. We demonstrated this feasibility using the traffic traces of 1,286 websites as the background traffic. While more websites can be added as the background traffic, since internet browsing patterns follow Zip's law [103], we believe our dataset sufficiently represents results closer to the real world. Finally, the area of open set classification is still nascent. As this area develops, we expect the accuracy of the attack to increase further. Moreover, it would be more beneficial to develop an attack that is capable of identifying new classes within rejected open set samples, auto-labeling them, and modifying the classifier to accommodate new classes: i.e., open world recognition [104].

Chapter 4

Dissecting Traffic Fingerprinting CNNs

Over the past decade, deep learning models have achieved remarkable success in a variety of pattern recognition tasks, including image classification [105, 106], text classification [107] and natural language processing [108, 109], surpassing traditional machine learning models. Despite their impressive performance, deep learning models provide very little insight into their inner workings and decision-making process and therefore are commonly referred to as 'black-box' models. As discussed in Chapter 1.4, blindly depending on such models without adequate understanding of how they work or what their decision-making process is based on could be unsafe, especially in mission-critical applications. For instance, it is important to make sure a model has actually learned the exact task it was supposed to learn and that there is no bias in the performance of the model towards the specific nuances of the dataset used to train and test the model. Therefore, various prior works (mostly targeting 2D CNN models used in computer vision tasks) attempted to understand these models using multiple techniques, such as visualizing filters at various layers [30] and identifying inputs or sections of inputs responsible for the decision of a model [27, 29].

At the same time, the recent advances in deep learning motivated researchers in the networking community to leverage them for traffic fingerprinting tasks. Most recent traffic fingerprinting attacks leveraged Multi-Layer Perceptrons (MLPs), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Auto Encoders (AEs)

to achieve very high accuracies surpassing previous traditional machine learning-based attacks and became an increasingly realistic threat to user privacy. Furthermore, some deep learning-based traffic fingerprinting attacks [14, 47] are successful against state-of-the-art defenses. As a result, understanding the behavior of deep learning models used in traffic fingerprinting task has become an important research area. More specifically, a thorough understanding of the deep learning models used in traffic fingerprinting could be helpful in both improving the performance of current attacks as well as in designing better defenses against such attacks.

One of the non-trivial results that have been observed across multiple independent works is the fact that for traffic fingerprinting, CNNs are showing better performances compared to any other deep learning or non-deep learning methods [15–18]. This can be counter-intuitive because CNNs were mainly designed for image classification and for time series data such as network traffic flows, RNNs were proven to be more successful in other domains such as speech recognition and speaker identification [31, 32].

In this chapter, we methodically dissect network traffic fingerprinting CNNs with the objectives of understanding what information from input data has the highest contribution towards their decision, what patterns they learn from input data, and then exploring the possibility of leveraging that information to train traffic fingerprinting CNNs with lesser training data and time. Moreover, the insights obtained from such analysis can also make significant contributions to designing efficient and practical defenses against such attacks. Even though there are some defense mechanisms proposed against traffic fingerprinting, most of them still require significant improvements to come to a level where they can be deployed in real-world scenarios. For example, there has been a body of work on defenses against website fingerprinting attacks [22–24]. Nonetheless, even the state-of-the-art defense [24] incurs around 40% data overhead to provide reasonable security. Most other types of traffic fingerprinting attacks, such as video fingerprinting and voice command fingerprinting, have very few or no defenses proposed against them at all. To the best of our understanding, this is the first work to explore the internal workings of network traffic fingerprinting CNNs as well as 1D CNNs in general, which are less common compared to 2D CNNs used heavily in computer vision applications. Then, based on our findings, we propose two new defense techniques against website fingerprinting and video fingerprinting attacks.

4.1 Dataset and CNN architectures

In this section, we present an overview of three publicly available datasets and associated CNN models that are used in all subsequent experiments.

4.1.1 Datasets under study

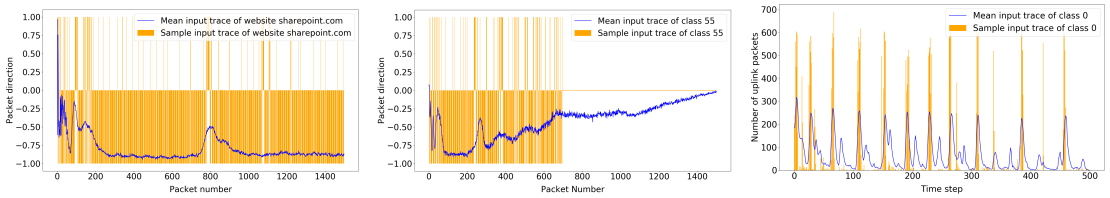
We use three datasets introduced in previous work; *i) Automated website fingerprinting (AWF)* by Rimmer et al. [15], *ii) Deep fingerprinting (DF)* by Sirinam et al. [14], and *iii) Deep content (DC)* by Li et al. [16]. All three datasets are publicly available. Next, we provide an overview of the three datasets.

4.1.1.1 Automated website fingerprinting (AWF) dataset [15]

The AWF dataset contains network traffic traces for visiting homepages of top-200 Alexa websites over `Tor` network. Each site visit is represented by the first 5,000 `Tor` packets in either direction. That is, in this dataset, a data sample is a sequence of +1s (uploads) and -1s (downloads). If a particular homepage visit did not generate a total of 5,000 packets in either direction, the remainder of the sequence was padded with zeros. For each website, there are 2,500 traces making a total dataset size of 500,000 traces. The authors also provided a mapping between the class label and the actual website URL.

4.1.1.2 Deep fingerprinting (DF) dataset [14]

The DF dataset has the same format as AWF dataset. However, the dataset contains traces only for the top-95 Alexa websites, and there are 1,000 traces per website. This dataset does not provide a mapping between the class label and the actual website URL. Therefore, despite having common classes with the AWF dataset, we can not conclusively identify them in the DF dataset, which becomes a limitation in our analysis as we discuss later in Section 4.3.



(A) AWF Dataset (B) DF Dataset (C) DC Dataset

FIGURE 4.1: Example data points and mean traffic trace for a sample class

4.1.1.3 Deep content (DC) dataset [16]

The DC dataset contains traffic traces for streaming the first three minutes of selected YouTube videos, collected at the data-link layer (WiFi). The three-minute interval is binned into 500 time slots (0.36s each), and each time slot is represented by summary statistics of packets observed during that time. Even though the original dataset is comprised of 24 features per trace, the authors observed that the number of packets on uplink traffic of video streaming trained the most accurate model, and hence we only use that feature. Accordingly, this dataset represents a traffic trace as a sequence of 500 integers between 0 and 736, which is the maximum number of non-data packets observed within 0.36 seconds. For each YouTube video, there are 320 traces. Unlike other datasets, the public version of this dataset does not contain the original train and test split. Thus, we randomly split the dataset to build the training, validation, and test sets.

We provide a summary of the datasets in Table 4.1. Figure 4.1, visualizes a data sample and the example *mean class* (e.g., the average of all data samples of a class) of a randomly picked class from each dataset. It should be noted that we use only the first 1,500 packets of a trace from the AWF and DF datasets in our experiments for better comparison between the CNNs with RNNs, as explained later in Section 4.4. In both AWF and DF datasets, most traffic instances have +1s in the initial part, which correspond to `HTTP GET` requests sent to the relevant web servers. The middle and later parts of these traffic traces are mostly -1s which correspond to downloading website content. The traffic traces that require padding to generate 1,500 packets usually bring the value of the average traffic trace of their relevant classes between zero and -1 in the corresponding segments. In contrast to the first two datasets, the traffic traces in DC dataset have periodic patterns that correspond to `DASH` chunk fetching.

While we do a comprehensive study of these three datasets when answering our research questions in Sections 4.2 - 4.4, we check the generalizability of our results using few other publicly available network traffic datasets in Section 4.5.

TABLE 4.1: Summary of datasets

Dataset	**No. of Classes**	**Traces per class**	**Training set size**	**Validation set size**	**Test set size**
AWF [15]	200	2,500	350,000	75,000	75,000
DF [14]	95	1,000	76,000	9,500	9,500
DC [16]	10	320	2,510	50	640

4.1.2 CNN Architectures

The CNN architectures we focus on in this chapter are the architectures proposed in the work corresponding to each dataset used. In all three scenarios, CNN architectures gave the best performance compared to other deep learning and traditional machine learning methods the authors evaluated. Moreover, as the authors of previous work had already tuned the hyperparameters of these models to obtain the best performance, we used the exact same parameters. For AWF and DF datasets, the deep CNN architecture proposed in [14] (i.e., multiple convolutions and pooling layers followed by two dense layers) gave the best performance, and for the DC dataset, the best performing architecture is a CNN with three convolutional layers followed by a max pooling layer and a dense layer. We show these two CNN architectures in Figure 4.2. This results in three test scenarios that will be used throughout this paper as follows.

i DF CNN on AWF dataset ***(AWF model)***

ii DF CNN on DF dataset ***(DF model)***

iii DC CNN on DC dataset ***(DC model)***

While most works on website fingerprinting such as DF and AWF that are based on deep learning models used only the directional information, some recent work [110] used both directional and timing information. Since using timing information may require some manual feature engineering and using only directional information can still achieve over 95% accuracy, in this work, we focus on website fingerprinting CNNs that use directional information only. We note that attacks that use both timing and directional data may need modifications to the architecture and need to be understood separately.

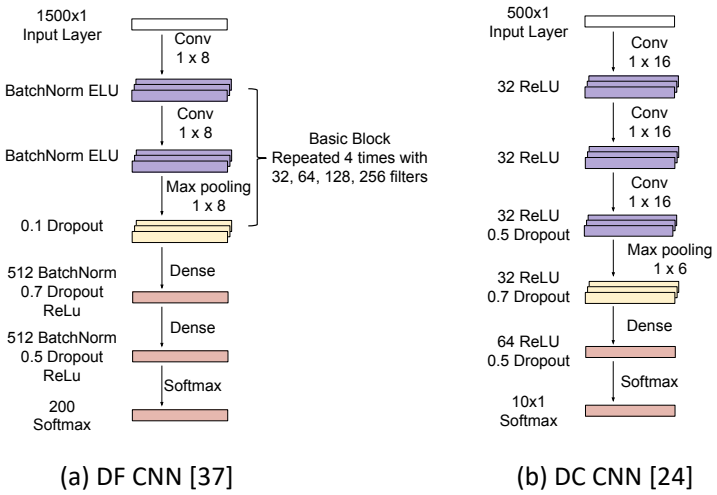


FIGURE 4.2: CNN architectures used

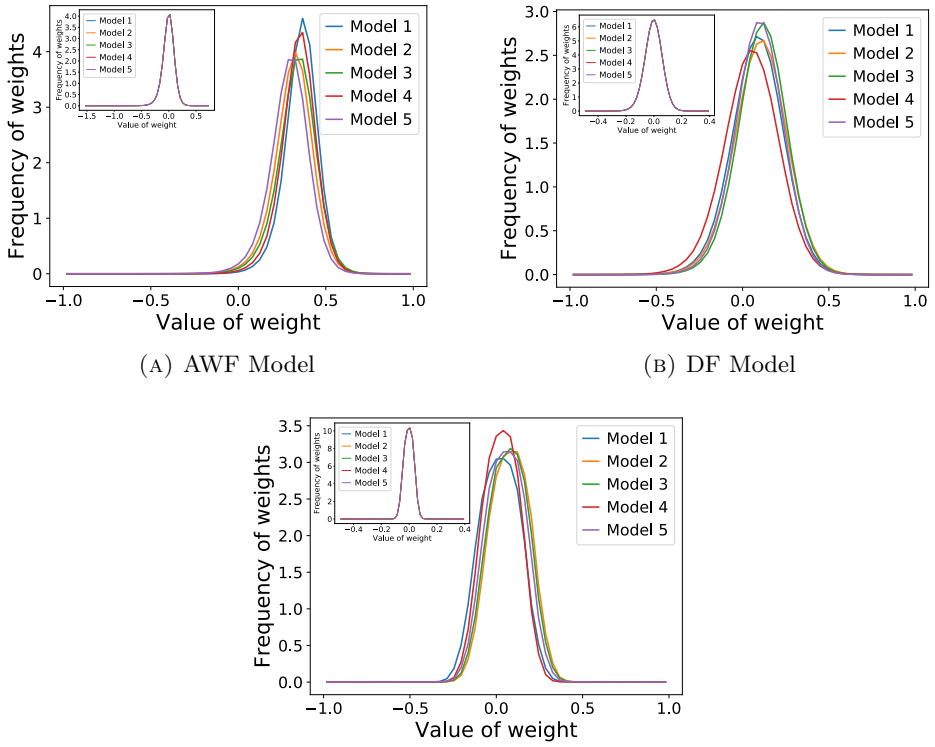
4.2 Impact of Patterns and Lengths of Network Traces

In this section, we methodically study how network traffic patterns activate filters in traffic fingerprinting CNNs with the objective of understanding what patterns they pick from input data and specific parts of the input the filters learn to focus on. As we discuss along the way, this type of analysis not only provides directions to improving traffic fingerprinting attacks but also provides insights on how to develop techniques to bypass deep learning-based traffic fingerprinting tools to design better mitigation techniques that can circumvent traffic fingerprinting. Also, our observations shed light on developing techniques to train traffic fingerprinting CNNs faster and with less training data.

4.2.1 Filter Weight Distributions

First, we present the distribution of all trainable parameters of the three scenarios we consider; DF CNN on AWF dataset, DF CNN on DF dataset, and DC CNN on DC dataset. We trained five model instances for each scenario i.e., five model instances of the same architecture trained on the same dataset and show the distribution of all the trainable parameters (normalized to be in the range between -1 and +1) for each scenario in the form of probability distribution functions (PDFs) in Figure 4.3. We also show the distribution before normalization in the corresponding sub-figures. The figures show that, over different model instances, the distribution of trainable parameters

remains in the same range and shows similar probability distributions. Therefore, for the subsequent analysis, we use one model instance per scenario. Since weight distributions remain similar over multiple trained instances, the results we subsequently derive are general and not specific to one trained model instance.



(A) AWF Model

(B) DF Model

(C) DC Model

FIGURE 4.3: Probability distribution of trainable weights compared over multiple model instances

4.2.2 Visualizing 1D Convolution Filters

Given an input sequence, $x_1, x_2, ..., x_N$ and a 1D convolution filter of size N with weights $w_1, w_2, ...w_N$, the convolution between them is given by $\sum_{i=1}^{i=N} w_i x_i \ + \ b$ where b is the bias term. This is analogous to the cross-correlation function, which measures the similarity between two sequences as a function of the displacement of one relative to the other. Accordingly, the convolution between an input and a filter can be seen as finding sections of the input that match the pattern of the filter.

To demonstrate this idea which is one of the building blocks of our subsequent analysis, in Figure 4.4-(a) and (b), we show an example input (top), an example filter (middle), and the filter output (bottom) for AWF/DF and DC models. In different colors, we

highlight different parts of the input, and positions where the highest filter activation is reached corresponding to that input part, are shown in the same color in the output. For instance, in each figure, when the blue section of the input is convoluted with the filter, the resulting value of the output is also shown in blue. As the figures show, the maximum values in the output correspond to the positions in the input where the sections that exactly match the filter begin.

For instance, in the case of AWF/DF models, i.e., Figure 4.4-(a) where the input can take either -1 or +1 only, for a single convolution between a filter and a section of the input, the corresponding value in the output takes the *maximum possible value* if the sign (positive or negative) of the input is same as that of the weight in the filter for all positions. The output value will be *closer to the maximum possible value* when the sign of the value in the input is the same as that of the weight in the filter for at least the positions where the absolute value of the weight of the filter is high. Similarly, the output will take the *least possible value (largest negative)* when the signs of the values of the input and the filter are exact opposites. As seen in Figure 4.4-(a), when the signs of the filter and input are exact opposites, especially in positions where the absolute value of the weight of the filter is high, the result will be a large negative value. However, the negative values of the filter output have minimum effect on the performance of the model as the non-linearity functions like Rectified Linear Unit (ReLU) and Exponential Linear Unit (ELU) that immediately follow convolution layers shut off these values by making them zero or very close to zero.

In the case of DC model, i.e., Figure 4.4-(b), where the input can only take integer values between 0 and 736 (maximum number of packets observed in the uplink during a 0.36s time window), the output will be closer to the *maximum possible value* if values of input in positions corresponding to large filter weights have larger values and values of input in positions corresponding to negative filter weights have values closer to zero. The opposite scenario will result in large negative values in the output; nonetheless, as explained above, the non-linear functions used in the considered CNN models will eliminate the effects of these values.

This intuition can be helpful in understanding features learned by filters of a single convolutional layer, as one can visualize the weights of a filter of a convolution layer at a given depth and figure out the patterns in the immediate input to that layer the filter

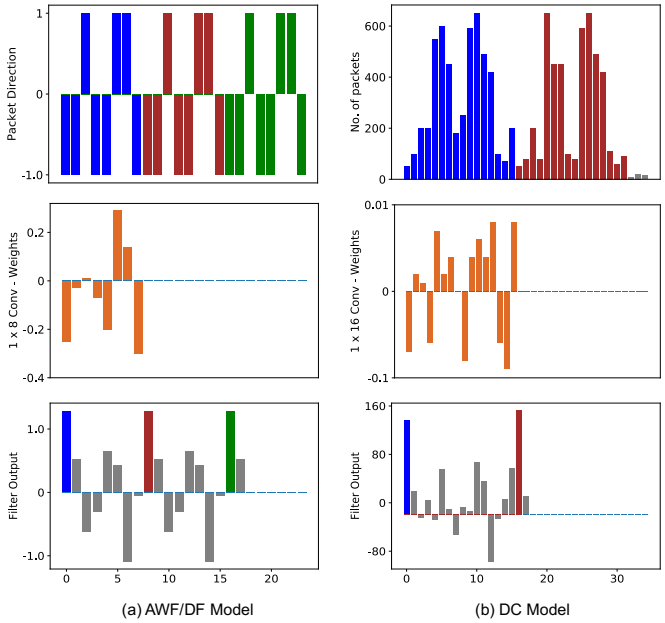


FIGURE 4.4: Pattern matching behavior of 1D convolutions

is sensitive to. However, this method is only suitable for the input convolution layer. The stacked components such as pooling and non-linear activations used in the model, make it infeasible to use this method to visualize the patterns of the original input that maximally activate the filters of subsequent convolutional layers. *To address that, we use two different methods for the AWF/DF and DC datasets that were drawn from the comparable work in understanding image classifying CNNs.*

4.2.2.1 Visualizing DC model filters with Gradient Ascent

As we discussed earlier, the part of an input that maximally activates a given filter (i.e., maximizes the output for the given filter) can be considered as the pattern that the filter is looking for. This idea can be reformed into an optimization problem as given in Equation (4.1), where θ is the set of parameters of the deep neural network and $h_{ij}(\theta, x)$ is the mean activation of the i^{th} filter of the j^{th} layer we are interested in [29]. ρ refers to the modulus of the input x when its elements fall within the range of an element in a valid input to the network. For example, if the input to the network is a sequence of packet sizes, elements of x should be within the valid range for packet size. The aim of this optimization problem is to find an input x^* such that it maximizes the output of the i^{th} filter of the j^{th} layer of the model.

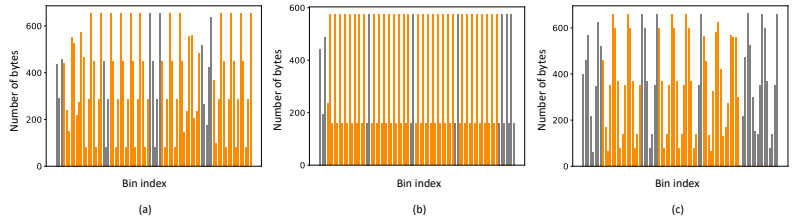


FIGURE 4.5: Gradient ascent learned inputs (Layer 1 DC model)

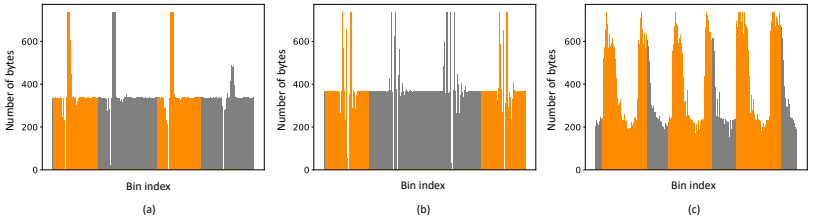


FIGURE 4.6: Gradient ascent learned inputs (Layer 3 DC model)

$$x^* = \underset{x \text{ s.t.} \|x\|=\rho}{\arg\max}\ h_{ij}(\theta, x) \tag{4.1}$$

This optimization problem can be solved using gradient ascent in the input space, i.e., by computing the gradient of $h_{ij}(\theta, x)$ and moving x in the direction of this gradient. This is a non-convex optimization problem, and therefore, gradient ascent will find a local maximum.

Example inputs learned by the gradient ascent method that maximize the mean activation of selected filters of Layer 1 and Layer 3 of DC model are shown in Figure 4.5 and Figure 4.6 respectively, where each figure shows an input of length 500 that gives maximum activation value when convoluted with a given 1x16 filter. The orange-colored sections of each figure correspond to the receptive fields (*i.e., the part of the original input that affects the value of a given position of the output*) of a few selected high activations in the output of the convolution between the given filter and input. The repetitive nature of high activations suggests that video fingerprinting CNNs respond to bursts in their inputs with specific shapes and lengths.

4.2.2.2 Visualizing DF and AWF model filters

Since the traces in DF and AWF datasets are represented with only two discrete values (-1s and 1s), performing gradient ascent on a random input to optimize average filter activation is not feasible. The reason being at each step, gradient ascent will be flipping the sign of the input value without converging. Thus, we propose a simple non-parametric method that uses the input trace with the highest filter activation value from the training set to approximate the features learned by filters of deeper convolution layers. We first feed the entire training set into the model and find the highest activation value from the output of the filter to be visualized. Then we track the input trace corresponding to that activation and, using the receptive field of the filter, identify the exact sequence of values that resulted in the activation value. Since the set of traces used in this method and the set of traces used to train the weights of the model are the same, we can safely assume that the result of this method is in fact, the pattern the filter is looking for. In Figure 4.7 and Figure 4.8 we show the traffic patterns derived using the above method for three selected 1x8 filters from the first and second convolutional layers of both AWF and DF models. Accordingly, it can be seen that all filters look for specific patterns with combinations of +1s and -1s in the input traffic trace, an observation we further elaborate on later.

(A) AWF Model (B) DF Model

FIGURE 4.7: Patterns learned by Layer 1 filters

(A) AWF Model (B) DF Model

FIGURE 4.8: Patterns learned by Layer 2 filters

4.2.3 Impact of Traffic Patterns on Classifiers

Having established the necessary background of 1D convolutions, in the following sections, we try to address a few research questions. The first research question we try to

answer is:

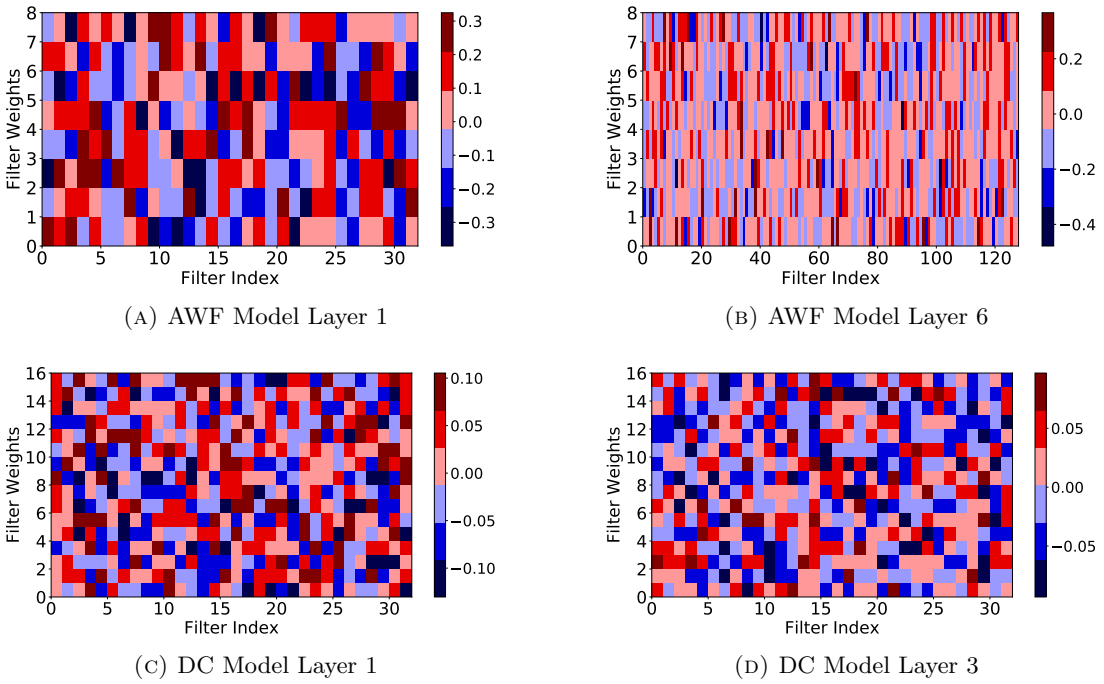
RQ 1*: What type of patterns do traffic fingerprinting CNNs learn?*

To answer this question, we visualize the filter weight distribution of all filters for selected convolutional layers. For all layers except the first, each filter consists of n number of 1x8 sub-filters where n is the number of filters in the previous layer. Hence for clarity, we only visualize the first sub-filter of each filter in layers other than the first layer. Figure 4.9a and Figure 4.9b show that in `Tor` website fingerprinting CNNs, almost all filters look for sequences with a combination of +1s and -1s (uploads and downloads), and not for sequences of all +1s or all -1s (i.e. as highlighted by lack of all "red variant" or all "blue variant" vertical lines in the figure).

In fact, in AWF and DF CNNs, there were only 2 filters out of 32 filters in the first layer that had all positives, and no filter had all negatives. This means traffic fingerprinting CNNs rely on transitions between uploads and downloads of a trace rather than the length of downloads. This observation could be one of the main contributing factors for the resilience of DF CNN model [14] against WTF-PAD [23], a candidate defense against `Tor` website fingerprinting which uses adaptive padding that adds padding only when the channel usage is low.

According to the filter weights shown in Figure 4.9c and Figure 4.9d, the combinations of positive and negative values in the DC filters suggest that video fingerprinting CNNs look for sequences of different number of packets per unit time. This means filters are sensitive to traffic bursts, yet they do not focus only on the envelope of the burst signal but rather consider the finer sub-bursts associated with a major burst. Otherwise, the positive filter values will be available only towards the middle part of the filter.

Thus, based on the learned filter weights, we can conclude that website fingerprinting CNNs learn to focus on the transitions between uploads and downloads within a trace, as opposed to sequences of consecutive uploads/ downloads.



(A) AWF Model Layer 1

(B) AWF Model Layer 6

(C) DC Model Layer 1

(D) DC Model Layer 3

FIGURE 4.9: Filter weights of selected convolution layers

4.2.4 Positions of Traces with Maximum Activations

The next research question we attempt to answer is;

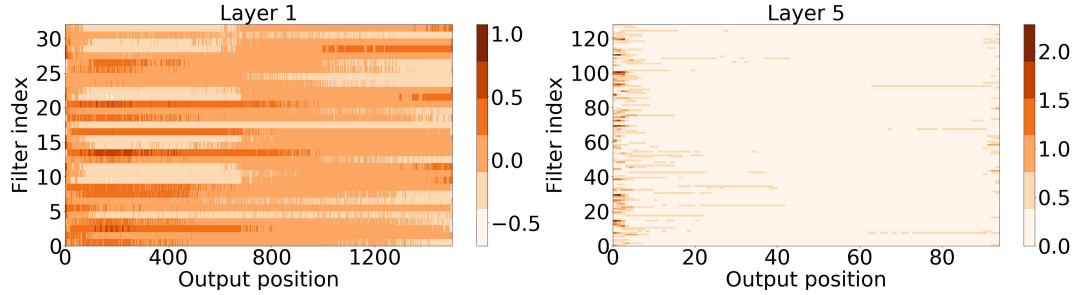
RQ 2*: Is there any part of the trace the traffic fingerprinting CNNs focus more on?*

To answer this question, we took a random sample of inputs from each dataset and calculate the filter output for each filter at each layer for all three CNN architectures we consider.

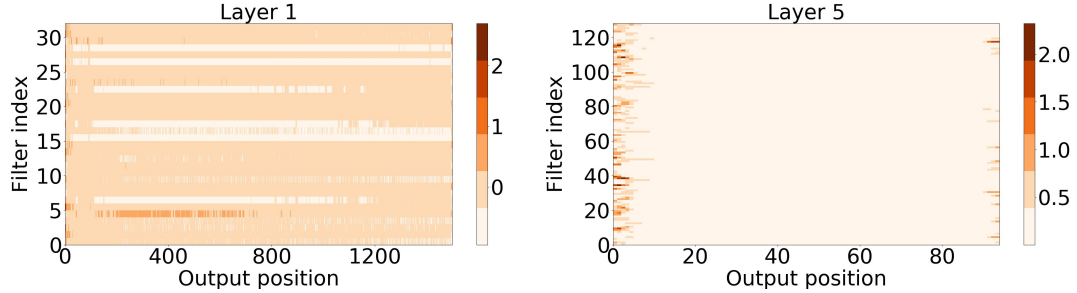
First, we present the analysis for AWF and DF datasets. We studied the mean activation of 500 random input network traces for all filters per each layer of DF and AWF models. In Figure 4.10a and Figure 4.10b, we show the mean filter activation for all filters in Layer 1 and Layer 5 as examples. The figures show that high-filter activations are happening either at the beginning of the trace or at the end of the trace. We show zoomed-in versions of the first 60 inputs of Layer 1 in Figure 4.11a and Figure 4.11b. To further emphasize this fact, in Figure 4.12 we plot the zoomed-in versions of Layer 1 for

two example classes; *twitter.com* and *aliexpress.com*. As can be seen from Figure 4.12, there are significantly high activations in the initial parts of the traces.

As explained in Section 4.1.1, for most traffic traces for website fingerprinting, the initial parts contain a lot of +1s that correspond to `HTTP GET` messages, and therefore for most of the input traffic traces, a high concentration of transitions can be seen in the initial parts. Depending on the weights of the filters where they closely match the patterns of the input, convolution will give very high activations, while the segments of the input that are closer to the exact opposite of the patterns learned by the filter will give very small activation values. Apart from the initial parts of the activation maps, the last few positions are also seen to be having very high/low values. This is due to the zeros added to traffic traces either at the data processing stage to make all traces have a constant length as explained in Section 4.1.1 or during convolution layers to preserve the original length of the input ('same' padding). To further emphasize the above observation, in Figure 4.13 we plot the mean filter output of the entire training set for all filters in each block of AWF model. It can be seen from all the sub-figures that the highest filter activations correspond to either the beginning of the trace or the end of the trace.



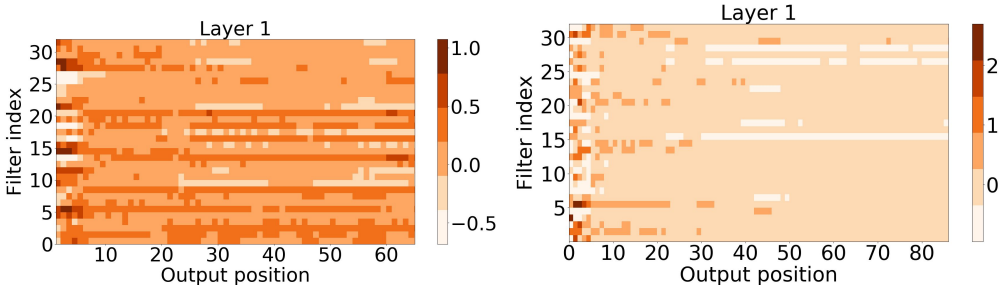
(A) DF Model



(B) AWF Model

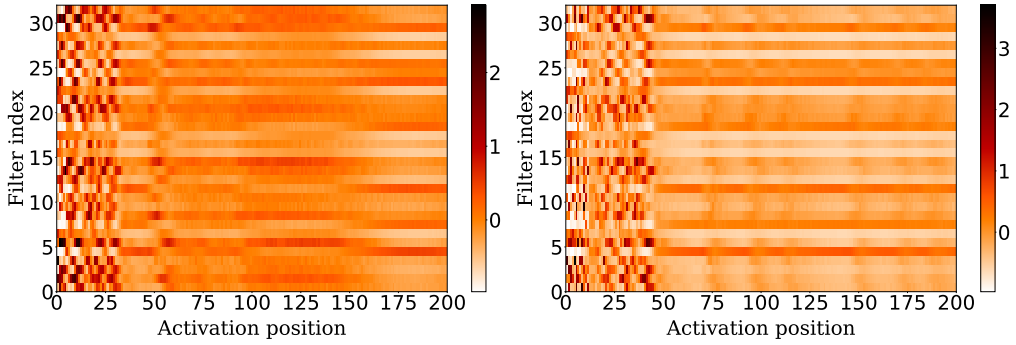
FIGURE 4.10: Activation maps of AWF and DF models

Next, we perform similar experiments with the DC Model. The activation map for 30 random input traces for the first and third convolutional layers of the model are shown in



(A) DF Model (B) AWF Model

FIGURE 4.11: Zoomed-in activation maps of Layer 1



(A) twitter.com (B) aliexpress.com

FIGURE 4.12: Zoomed-in activation maps of Layer 1 for example AWF classes

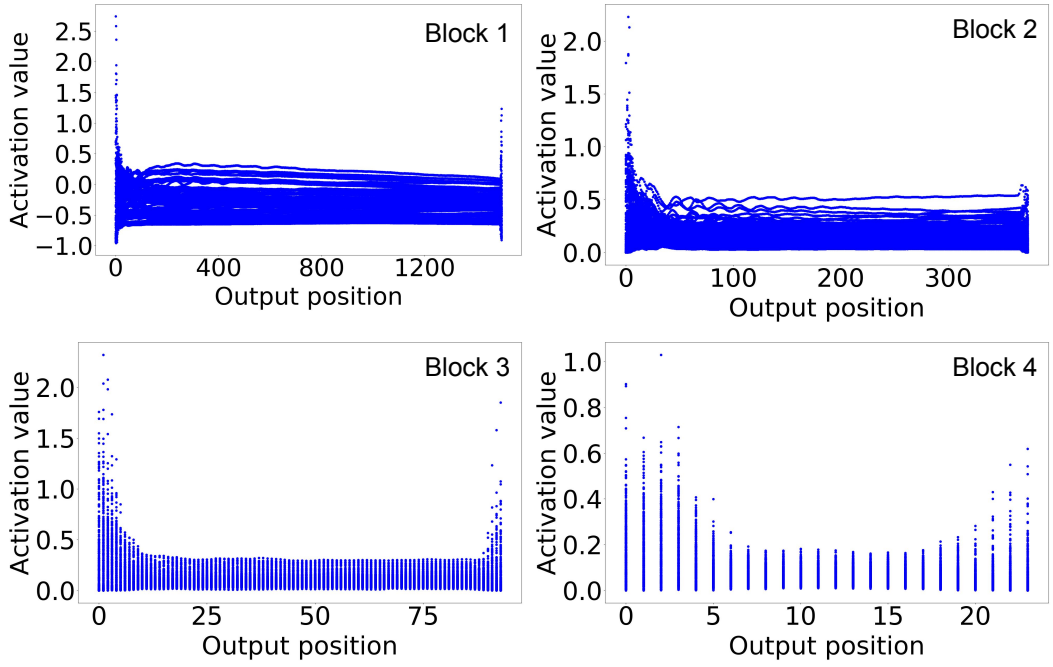


FIGURE 4.13: Block-wise activation maps of AWF model

Figure 4.14. In contrast to the activation maps of AWF and DF models, the activation maps of DC model have high variations along the entire length of the output. The main reason for the above difference is that unlike traces of visiting websites, streaming videos

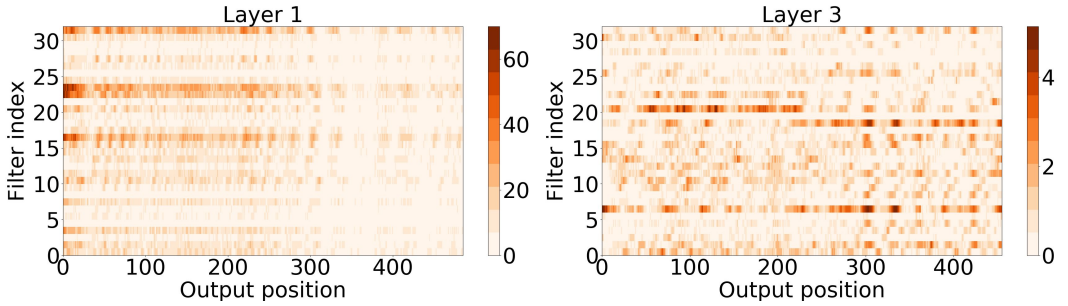


FIGURE 4.14: Activation maps of DC model

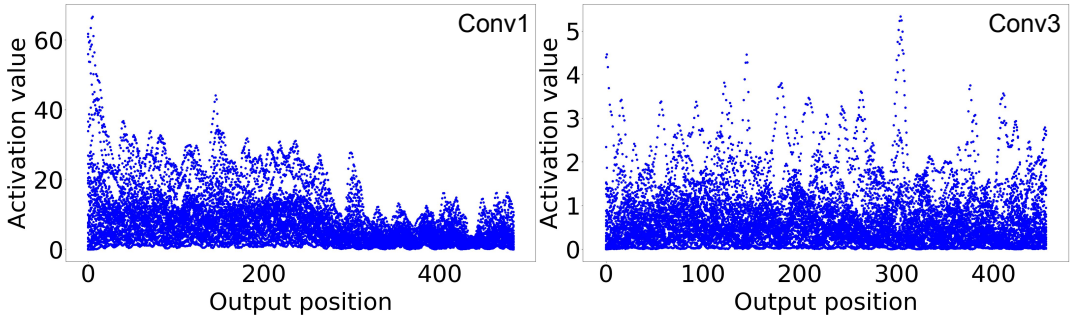


FIGURE 4.15: Layer-wise activation map of DC model

involve periodic requests to the relevant servers and hence contain bursts throughout the input, which causes variations in the output in corresponding positions. Similar to AWF and DF models, depending on the extent to which the filter matches the pattern in the input, the activations can take higher or lower values. The mean filter outputs of the entire training set of the DC model for all filters in Layers 1 and 3, as shown in Figure 4.15, further emphasize the above observation.

In summary, the filter activation analysis corroborates our previous finding; i.e. traffic fingerprinting CNNs focus more on parts of the traces where more `HTTPS` *activities are happening. This means for website fingerprinting, the classifiers place more weight on the initial part of a traffic trace which contains a relatively high concentration of uploads that correspond to initial HTTP GET requests and replies. It should be noted that our observations are comparable with previous work that found that features extracted from the first thirty packets are relatively more important [41] and that the first few incoming packets correspond to the initial unique HTML page that helps discriminate between webpages [8]. In video streaming, there are periodic bursts and accordingly, the CNNs filters focus on the finer shapes of those bursts.*

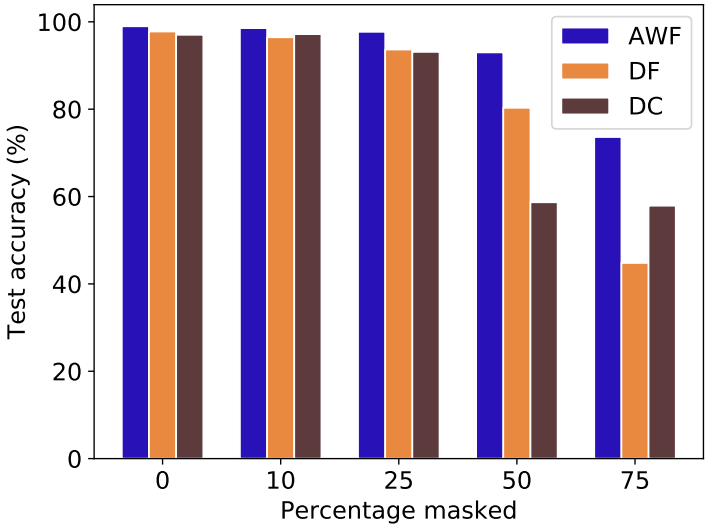


FIGURE 4.16: Test accuracy with masking

4.2.5 Impact of the Trace Length on Classifier

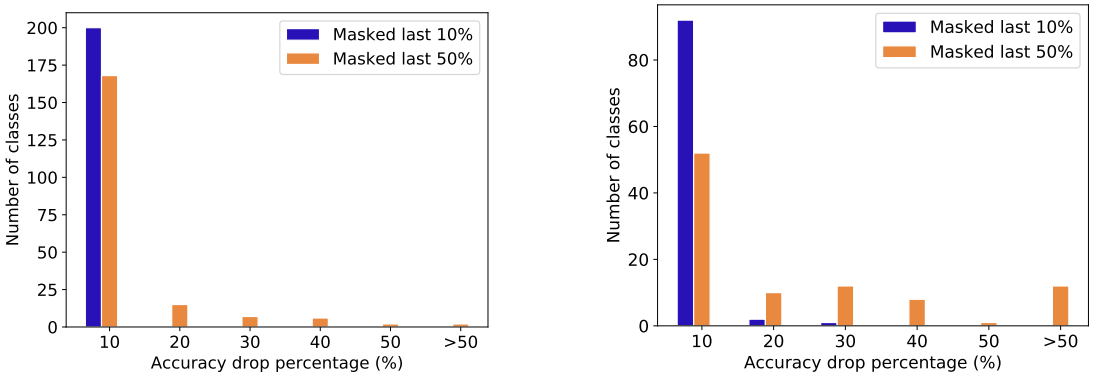
Since we established that for web traffic fingerprinting CNNs, what matters the most is the initial part of the trace, the next research question we attempt to answer is;

RQ 3*: Would the initial trace portion alone be sufficient to make accurate predictions?*

To investigate this, we masked the last 10%, 25%, 50%, and 75% of the inputs of the three test sets with zeros. Then, we used the trained models to predict the class of the input. In Figure 4.16, we show the accuracies we obtained compared to the unmasked (0% masking) accuracies for all three datasets. As the figure shows, website CNNs show strong resilience to masking, corroborating our previous observation that they are more focused on the initial parts of the trace. For example, the AWF CNN and DF CNN models showed 93% and 80% accuracy, respectively even if the input is masked by 50%. The better resilience of AWF model compared to DF model can be attributed to the fact that the DF model is more prone to overfitting due to the less number of classes and fewer traces per class in its dataset compared to the AWF model. DC model as expected, is highly susceptible to masking because it focuses on the periodic transfers of DASH chunks. Nonetheless, even its accuracy does not drop significantly until 25% masking, indicating video content identification can be done with a lesser length than the originally used three-minute interval.

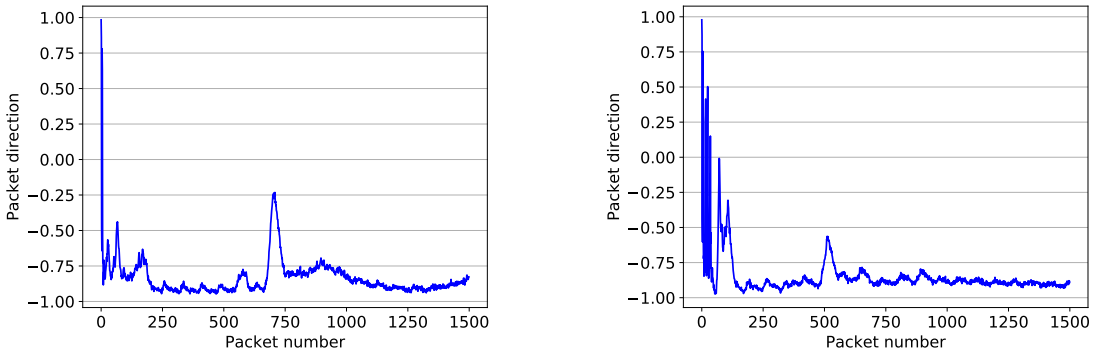
We next analyze the effect of masking on individual classes to understand the accuracy drop further. In Figure 4.17a and Figure 4.17b, we show the number of classes that lost accuracy by different percentages when the last 10% and 50% of the trace is masked in website fingerprinting models. At 50% of masking in AWF dataset, only 8.50% of classes had more than 20% drop in accuracy, and the corresponding number for DF dataset was 16.50%, indicating that the accuracy drop is majorly coming from a limited set of classes and the majority of the classes are not affected by masking at all. Visualizing the average traffic trace for these classes explains the reasoning behind the above observation. For example, the average traffic trace for class 2 of the AWF dataset shown in Figure 4.18a shows a cluster of positive ones around the 700th position that will be ignored by masking which causes the model to classify the instances incorrectly. In contrast, the average traffic trace of class 197, i.e., Figure 4.18b shows that almost all packets after the 600th position are -1s, which provide no additional information and hence masking these packets does not affect the classification accuracy of the class at all.

To conclude, this analysis further reinforces our observations in RQ1 and RQ2 and shows that the initial part of a trace is more important and itself sufficient to achieve a reasonable classification accuracy in network traffic fingerprinting and corroborates similar observations in previous works [8, 41].



(A) AWF Model (B) DF Model

FIGURE 4.17: Class based effect on accuracy from masking



(A) Class 2 (B) Class 197

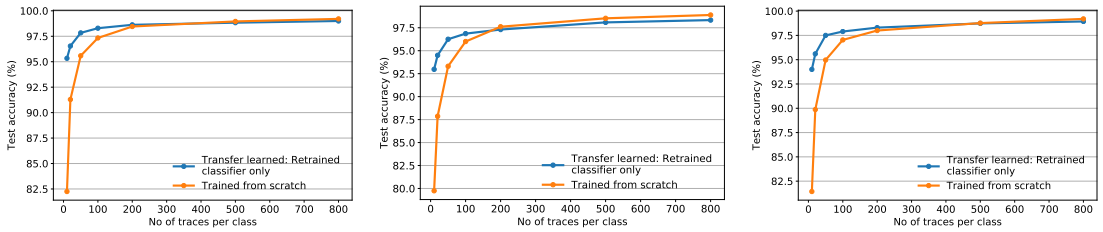
FIGURE 4.18: Mean traffic traces of two example classes affected by masking

4.3 Transfer Learning Capabilities

One of the main reasons CNNs are used heavily in image classification and many other computer vision tasks is their transfer learning capabilities [111]. CNN filters learn to match different patterns related to image classification tasks, and it does not matter which dataset the model is trained on as long as the filters learn common patterns. This allows pre-trained CNNs to be modified to classify new image classes with far less data than usually required to do the original training, as well as fewer training epochs (i.e., the number of passes of the full training dataset, which is one of the main reasons why CNN training takes a significant amount of time). Thus, the next research question we explore is:

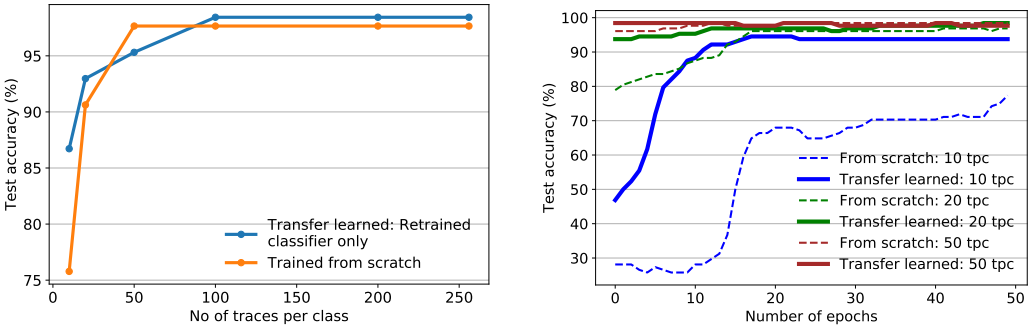
RQ 4*: Do network traffic fingerprinting CNNs have similar transfer learning capabilities, and can we leverage that to optimize the training process?*

To investigate this, we split each dataset into two parts, A and B, such that all instances of one class (i.e., websites or videos) belong to one part only. Next, we pre-train a model instance of the corresponding CNN architecture on dataset A and then keeping the feature extraction part of the model (i.e., convolutional blocks) fixed, fine-tune only the classifier part of the model (i.e., the fully connected layers) on dataset B by progressively increasing the number of traces per class (tpc) to check whether a high accuracy can be achieved with lesser number of samples.



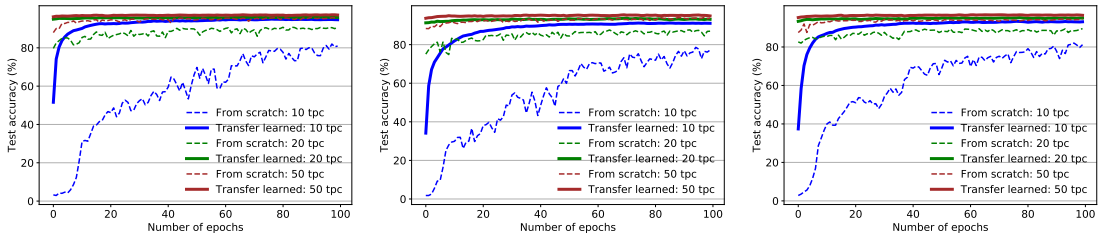
(A) 120:80 (B) 80:120 (C) 100:100

FIGURE 4.19: Test accuracy of transfer learning with varying number of traces per class in training set



(A) Test accuracy (B) Convergence of test accuracy

FIGURE 4.20: Transfer learning on DC dataset.
* *tpc* refers to traces per class



(A) 120:80 (B) 80:120 (C) 100:100

FIGURE 4.21: Convergence of test accuracy with transfer learning.
* *tpc* refers to traces per class

In Figure 4.19, we compare the test set (which was 7.50% of B) accuracy of a model trained from scratch on dataset B, and the transfer learned model explained above for different $A : B$ ratios for the AWF dataset. In all three cases shown in Figure 4.19, the transfer learned model reaches the accuracy plateau with a lesser number of traces per class. If no transfer learning is applied, approximately 150 more traces per class are required to reach the same level of accuracy. Also, transfer learned models start with accuracies over 90% with training data as less as 10 traces per class. The DF dataset showed similar behavior; thus, we have not included the results here. This is

an important observation as it allows to build traffic classifiers with less training data. Also, this observation indicates that the patterns learned by the convolutional blocks are generalizable. Finally, we highlight that we have not included the cross-dataset transfer learning between AWF and DF datasets because we could not identify common classes between them (**cf.** Section 4.1.1). To demonstrate the ability to use a feature extractor trained on one set of classes on a new set of classes, it is important that there is no overlap between datasets A and B.

Then we conduct similar experiments with the DC dataset keeping the ratio between the number of videos in A and B as 3:2. We do not experiment with different $A : B$ ratios in DC dataset as it has only a few classes. In Figure 4.20a, we compare the test set (which was 12.50% of B) accuracy of a model trained from scratch on dataset B vs. the transfer learned model explained above, and it shows that the transfer learned model achieves over 85% accuracy with just 10 traces per class and also records the better performance even with 256 traces per class (max tpc in the dataset). Since the dataset is much smaller compared to AWF and DF datasets, the advantage of transfer learning quickly diminishes.

We next show how transfer learning can help to improve the training time of traffic fingerprinting CNNs. In Figure 4.21 we show the number of epochs (full passes over the training set) till the accuracy plateaued for the three $A : B$ split scenarios we described earlier. In each graph, we compare three cases; 10, 20, and 50 traces per class with and without transfer learning. The advantage of transfer learning is evident when there is a lesser number of traces per class. For example, when there are 10 traces per class, transfer learning reaches the accuracy plateau in less than 40 epochs, whereas training from scratch does not reach the same level even after 100 epochs, and it is most likely to not even reach that level of accuracy. 20 traces per class show a similar tendency, and transfer learned models reach the plateaued level of accuracy in less than 10 epochs, and there is a gap of 10% in accuracy even after 100 epochs. The advantages of transfer learning are diminished at 50 traces per class scenario, and it can be expected there is enough data points to start from scratch. A similar improvement in the number of epochs needed to train DC model is shown in Figure 4.20b.

Thus, we conclude that network traffic fingerprinting CNNs indeed possess transfer learning capabilities analogous to image classifying CNNs and as such, it is possible to fine-tune pre-trained models to accommodate new classes (i.e., websites or videos) with much less labeled data. While it is true that there are other methods for reducing data requirements of traffic fingerprinting attacks (e.g., N-shot learning [46]), we leave such comparisons as out of scope for this work since our objective was only to show the feasibility of transfer learning in traffic fingerprinting CNNs.

4.4 Network Traffic Classification: CNNs vs. RNNs

Since RNNs are known for their capabilities in sequence modeling, they are a natural choice to try when building traffic fingerprinting classifiers. Nonetheless, several works highlighted that in practice, CNNs outperform RNNs, in some cases by a significant margin [15, 16]. For example, as reported by the original work [15], the CNN achieved 96.52% accuracy while the LSTM RNN achieved only 93.17% accuracy on the AWF dataset. Similarly, for the DC dataset, the original work reports that the LSTM RNN model showed 2% less accuracy compared to the CNN model. Finally, the original work corresponding to DF dataset did not report any comparisons between LSTM and CNN models. Thus, we trained an LSTM model and found that there was a 13% gap in accuracy between the CNN (98.3%) and RNN (85.39%) models.

Here, we also highlight a detail that might be helpful in reproducing these results. The LSTM model used by Rimmer et al. [15] had only 150 steps (*LSTM model 150*) compared to the CNN, which accepted all 3,000 packets as the input. The authors reduced the input length to the LSTM model since LSTMs can not handle longer inputs due to the vanishing gradient problem. Since this might not allow a fair comparison between the CNNs and RNNs, for our subsequent analysis, we use 1,500 packets for both types of models for DF and AWF datasets. We note that both models of AWF and DF achieve close to 98% accuracy with input length 1500, and masking the latter parts further had less effect on model performance (Figure 4.16). In the *LSTM model 1,500*, we feed 10 consecutive points from a traffic trace as the input to each step to keep the number of time steps constant. As mentioned in Section 4.1.2, for all the LSTM models used in this section, we conducted hyperparameter tuning.

Furthermore, when evaluating the classification accuracy of traffic traces captured at a much later point in time than the training set (defined as *concept drift* by Rimmer et al. [15]), the CNN architecture surpasses the performance of both models; *LSTM 150* by the original authors and *LSTM 1,500* by us, as shown in Table 4.2. Having established the higher performance of CNNs for traffic fingerprinting tasks, the next research question we attempt to answer is;

RQ 5*: Why CNNs perform better compared to RNNs?*

TABLE 4.2: CNNs vs. LSTMs - Resilience to concept drift

LSTM model	**Number of days between test set and train set capture**				
	3	**10**	**14**	**28**	**42**
CNN	99.80%	97.90%	94.00%	89.00%	87.40%
LSTM_150	92.87%	88.91%	84.01%	77.25%	76.20%
LSTM_1500	93.50%	90.10%	84.80%	76.30%	73.20%

We conduct experiments based on the intuition that the patterns of uploads and downloads of multiple traffic traces of the same class would not be in exactly the same relative positions and can be shifted either left or right by random amounts due to varying network conditions. We hypothesize that the main reason behind the better performance of CNNs is their ability to identify burst patterns in a traffic trace irrespective of the pattern's relative position, analogous to the *translation invariance property* in image classification. To verify this, we evaluate the effect on the accuracy of models on DF dataset when traffic traces of the test set are shifted according to four scenarios;

i) shift right by n steps from initial packet

ii) shift right by n steps from a random position

iii) shift left by n steps from last packet

iv) shift left by n steps from a random position,

where n is a random number from a normal distribution with varying mean and standard deviation 50.

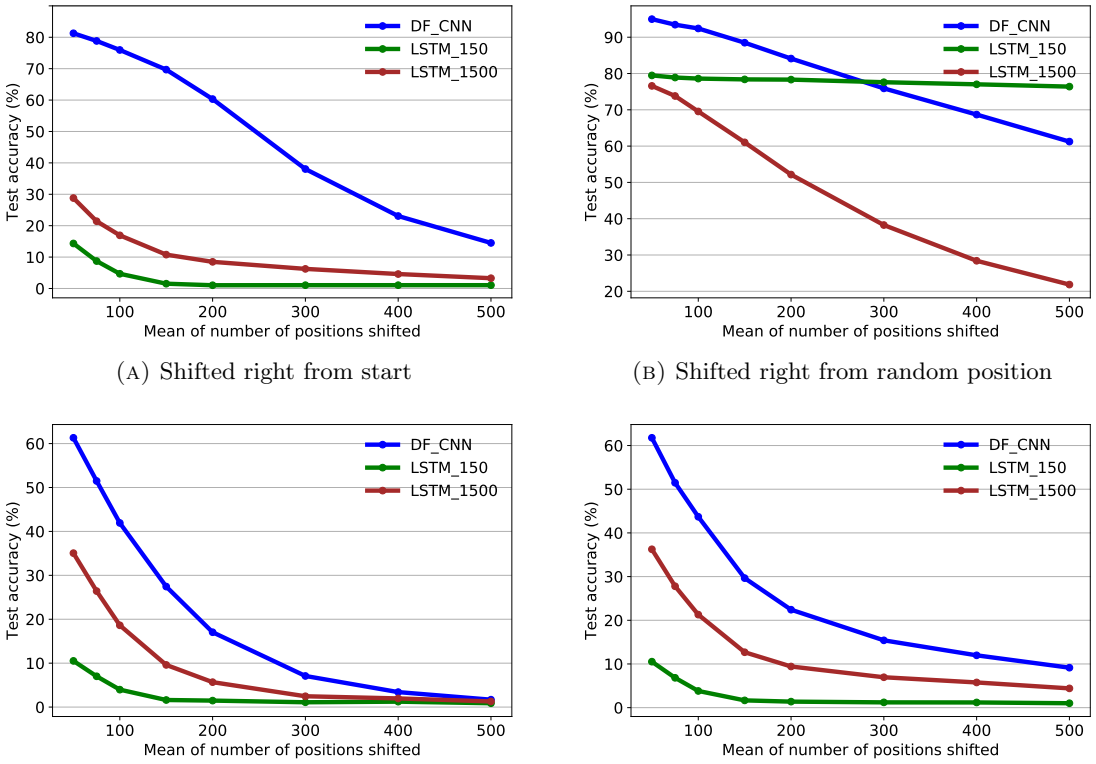
Figure 4.22 confirms this assumption. CNN performs significantly better than the LSTM models in all scenarios. For scenario (i), *i.e.* Figure 4.22a, the CNN model is able to

maintain its accuracy above 70% up to a 10% right shift. In contrast, both LSTM models achieve accuracies below 30% irrespective of the amount of shifting. The slightly better performance of LSTM 1,500 compared to LSTM 150 can be attributed to the longer context available to LSTM 1,500 model. For scenario (ii), *i.e.* Figure 4.22b, the CNN model succeeds in maintaining the accuracy above 85% up to 10% shift, and above 50% irrespective of the amount of shifting. However, the accuracy of the LSTM 1,500 model drastically declines with the amount of shifting and falls below 50% when the amount of shifting exceeds 20%. It should be noted that the accuracy of the LSTM 150 model remains almost constant because when the 1,500-long traffic trace is shifted left from a random position, the possibility of the initial 150 packets being affected is minimal. Furthermore, the accuracy of all three models is considerably better in scenario (ii) compared to scenario (I), as the initial part of the traces which carry more information, is less affected when shifted from a random position. Figure 4.22c and Figure 4.22d correspond to more challenging situations compared to scenarios (i) and (ii) as shifting to the left always drops the initial parts of traces, and hence all models have lower performances compared to the first two scenarios. *One possible way of addressing the above performance gap between RNNs and CNNs would be to train RNN models with augmented data [112] that represents such random shifts in traffic traces.*

Nonetheless, theoretically, it might still be possible to train RNN models that can come close to or even surpass the performance of current CNN models. However, in practice, the effort needed to make that happen appears to be very high. Previous work and our own results show that making a CNN work for traffic fingerprinting requires way less effort and hyperparameter tuning compared to RNNs.

We also compared the performance of CNN and LSTM architectures with Var-CNN [110], which is a model architecture recently used for website fingerprinting using the DF dataset. The model accuracies of CNN, LSTM, and Var-CNN [110] models are 98.30%, 85.39%, and 97.89% respectively showing that CNNs perform best. We note that as the model architectures were evaluated on the DF dataset, which does not contain timing information, Var-CNN was trained in the direction-only mode. The use of both directional and timing information may give different performances for Var-CNN.

To summarize, our analysis shows that compared to RNNs, CNNs are far more resilient



(A) Shifted right from start

(B) Shifted right from random position

(C) Shifted left from end

(D) Shifted left from random position

FIGURE 4.22: Test accuracy after shifting for DF dataset

to shifts in burst patterns of the input data. This makes CNNs a better option for traffic fingerprinting and classification since network traces inherently contain such noise caused by network delays.

4.5 Applicability to Other Datasets

In this section, we investigate the wider applicability of our observations by extending our analysis to three additional datasets. The new datasets include; i) FRONT [24] - a website traffic fingerprinting dataset, ii) SETA [113] - a video traffic fingerprinting dataset, and iii) VT [18] - a smart speaker voice command traffic fingerprinting dataset. We provide a summary of these datasets in Table 4.3.

We define a numerical metric for each research question so that we can compare the validity of our observations across datasets. For example, for RQ1, we need to verify whether a model has a negligible number of filters with all positive or all negative weights.

TABLE 4.3: Summary of additional datasets

Dataset	Traffic Type	Details
FRONT [24]	Website	100 websites (90 traces per class)
SETA [113]	Video	20 Netflix videos (100 traces per class)
VT [18]	Voice commands	98 commands on Google Home (∼1500 traces per class)

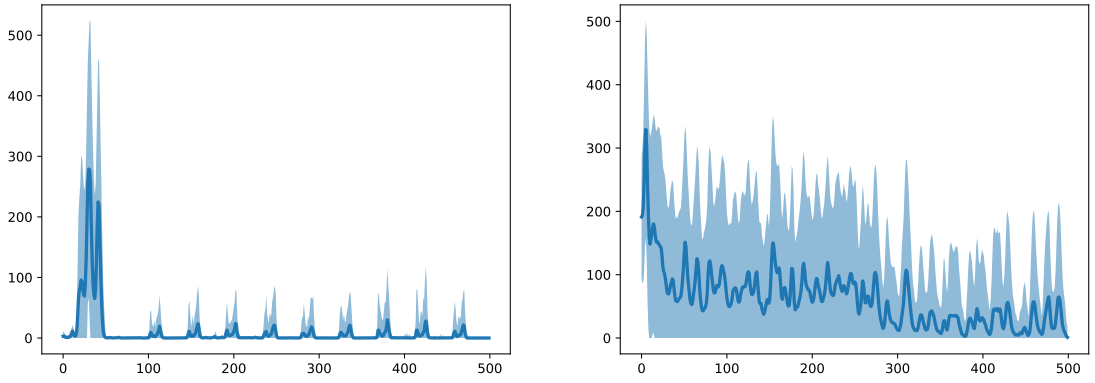
Therefore, for a given model, we calculate the percentage number of filters having all positive or all negative weights.

Similarly, for RQ4 we need to validate whether transfer learning can reduce the number of training samples required to converge to a target accuracy. For this, we calculate the ratio between the number of traces per class needed in the training set for a transfer learned model and for a model trained from scratch to achieve 95% of target accuracy. If the transfer learned model needs a lesser number of traces per class, we can conclude that our observation is valid across all datasets. We present the results of this analysis in Table 4.4 and detail out the findings below.

RQ1 - Under RQ1, our main observation was that traffic fingerprinting CNNs contain only a negligible number of filters with all positive or all negative weights. According to Table 4.4, we see that this observation holds across all datasets as none of the models corresponding to any of the datasets has more than 5% of total filters that have all positive or all negative weights. This strengthens our initial conclusion under RQ1, that for website fingerprinting datasets represented as sequences of uploads and downloads, the classifiers focus more on the transition between uploads and downloads instead of on long sequences of packets in the same direction, while for other datasets represented as sequences of positive integers the classifiers will focus on the fine sub-bursts within bursts of a trace.

RQ2 and RQ3 - Under RQ2 and RQ3 we observed that traffic fingerprinting CNNs give more weight to the initial part of a trace and that the initial portion of a trace alone is sufficient to make an accurate prediction on a model trained on complete traces. Similar to the experiment in Section 4.2.5, we keep increasingly masking the end of a trace with zeros until the accuracy of the masked test set on the original model falls below 5% of the accuracy of the un-masked test set.

Table 4.4 shows that for all datasets except the SETA dataset, approximately 20% (in some cases even 50%) from a trace can be masked from the end before the accuracy on the original model drops by 5%, which implies the significance of the initial part of a trace in the classifier's decision. The reason for the deviation of the behavior of the SETA dataset can be attributed to the fact that its traces have a very high inter-class similarity. In Figure 4.23 we compare the *mean trace* and the *mean±standard deviation* of traces of all classes of SETA and DC datasets. The *mean trace* for each dataset is shown in dark blue while the area shaded in light blue corresponds to the *mean±standard deviation*. When there is high similarity between classes, the model needs more information to differentiate between classes, and as such, the latter parts of the trace matter in the SETA dataset.



(A) SETA (B) DC

FIGURE 4.23: Mean trace (of all classes) for SETA vs DC datasets

RQ4 - In Section 4.3 under RQ4, we observed that traffic fingerprinting CNNs possess transfer learning capabilities that can reduce training data requirements. To check the validity of this observation on other datasets, we split each dataset into two non-overlapping groups such that the ratio between the number of classes in the two groups is 3:2 and observe the training data requirement (in terms of traces per class) when training a model from scratch for the smaller dataset, vs using a model on the larger dataset to transfer learn on the smaller dataset (**cf.** Section 4.3).

According to Table 4.4, for all datasets, the transfer learned model requires much less training data to reach 95% of the original model accuracy compared to training a new model from scratch. Except for FRONT dataset, transfer learning reduces the training

data requirement by half or more.

RQ5 - Finally, under RQ5 we used the AWF dataset to show that random shifts in traces of the test set have much less effect on the accuracy of a CNN model compared to an LSTM model and therefore suggested this as the cause for the better performance of CNNs compared to RNNs on network traffic data. To check if this observation holds for other datasets as well, we perform similar experiments as in Section 4.4 and report the mean amount of shifting needed to drop the accuracy of the un-shifted test set. For clarity, we report the results for a 40% drop for shifting right and 60% for shifting left. Note that shifting left can't handle large numbers because of the relative importance of the initial parts of the trace. However, similar results hold for other percentage drop values as well.

From Table 4.4 we observe that for website fingerprinting and voice command fingerprinting datasets, the corresponding CNN models are significantly more resilient to random shifts than LSTMs. For instance, for DF dataset, traces can be shifted right with a mean of up to 220 before the original accuracy drop by 40%, while the LSTM can only handle a shift with a mean of 10 before reaching the same threshold.

For the DC dataset, when shifted with random amounts with a mean up to 100, even though the accuracy drops below 40% for both models, the dataset maintains a similar trend as the majority, where the CNN is significantly more resilient to shifts compared to the LSTM, except when shifted left from a random position where both models have similar performances.

In contrast to the other datasets, we observe that for the SETA dataset, except for when shifted left from a random position, both CNN and LSTM models have similar performances until shifted with a mean up to 100, and for shifts higher than that, the LSTM performs slightly better. When shifted left from a random position, we observe that the LSTM for the SETA dataset performs significantly better compared to the respective CNN, which contradicts our initial observations in Section 4.4. This again, can be attributed to the high intra-class and inter-class similarity of the SETA dataset compared to others (i.e does not include a significant amount of noisy data).

TABLE 4.4: Applicability of RQs to other datasets - Summary of results

RQ	Criteria	DF	AWF	FRONT	DC	SETA	VT
RQ1	Percentage of filters with all '+' or '-' weights	3.56%	3.50%	3.50%	0.00%	0.00%	4.87%
RQ2/RQ3	Length percentage to mask to drop accuracy by 5%	27.33%	46.67%	18.00%	27%	6%	50.53%
RQ4	Ratio between traces per class needed to reach 95% of original model accuracy (transfer_learned: trained_from_scratch)	2:11	1:3	4:5	1:2	1:2	1:2
RQ5	Ratio between mean amount of shifting needed to drop original accuracy by 40% when shifted right from start, CNN:LSTM	22:1	28:1	5:2	1:1	1:1	13:1
RQ5	Ratio between mean amount of shifting needed to drop original accuracy by 40% when shifted right from random position, CNN:LSTM	22:1	28:1	9:4	1:1	1:1	5:4
RQ5	Ratio between mean amount of shifting needed to drop original accuracy by 60% when shifted left from end, CNN:LSTM	11:8	6:1	1:1	2:1	1:1	11:1
RQ5	Ratio between mean amount of shifting needed to drop original accuracy by 60% when shifted left from a random position, CNN:LSTM	11:8	6:1	13:12	13:20	7:25	9:1

Finally, we also checked whether the above observations would change when they are used in open world settings using an additional open set dataset provided by [14] and training background class models. We report the results in Table 4.5. When compared with Table 4.4, we can see the open set dataset also has a similar behavior showing how CNN behavior is similar in both closed set and open set settings.

TABLE 4.5: Applicability of RQs to open set setting

RQ	Criteria	DF_open
RQ1	Percentage of filters with all '+' or '-' weights	3.48%
RQ2/RQ3	Length percentage to mask to drop accuracy by 5%	48.67%
RQ4	Ratio between traces per class needed to reach 95% of original model accuracy (transfer_learned: trained_from_scratch)	2:5
RQ5	Ratio between the mean amount of shifting needed to drop original accuracy by 40% when shifted right from start, CNN:LSTM	25:6
RQ5	Ratio between the mean amount of shifting needed to drop original accuracy by 40% when shifted right from a random position, CNN:LSTM	25:23
RQ5	Ratio between the mean amount of shifting needed to drop original accuracy by 60% when shifted left from the end, CNN:LSTM	7:3
RQ5	Ratio between the mean amount of shifting needed to drop original accuracy by 60% when shifted left from a random position, CNN:LSTM	6:1

To summarize, we use three additional datasets to show that the observations we made up to Section 4.4 using three datasets are generalizable over most encrypted traffic fingerprinting datasets.

4.6 Discussion and Concluding Remarks

In this chapter, we systematically dissected three traffic fingerprinting CNNs that are trained on three publicly available encrypted network traffic datasets to understand

several key research questions associated with their operation and success over other deep learning methods.

First, we showed that website fingerprinting CNNs focus on parts of a traffic trace with transitions between uploads and downloads instead of sequences of continuous uploads or downloads and give more weight to the initial part of a traffic trace which contains a high concentration of transitions (due to multiple `HTTP GET` messages and replies) which allows using only the first half of the trace to get over 50% accuracy. Similarly, we showed that video fingerprinting CNNs focus on periodic sections of uploads and downloads in traffic traces that correspond to periodic bursts in video streaming. Next, we showed that traffic fingerprinting CNNs show the same transfer learning capabilities as image classifying CNNs, and as such, scaling up traffic fingerprinting CNNs with respect to the number of classes can be done with as less training data as ten traces per class. Then we showed that the resilience of CNNs to random variations in traffic flows and bursts that occur due to varying network conditions is the main contributing factor to their better performance against LSTMs. For instance, we showed that a 10% right shift in input data from the start caused only a 29% drop in accuracy in the CNN model while causing 88% accuracy loss in the *LSTM model 1,500*. This observation provides insights into how traffic fingerprinting RNN training process can be improved by augmenting the data and also opens up ideas for better-suited architectures for traffic fingerprinting by combining CNNs and LSTMs.

Finally, we investigated whether the observations and conclusions we made under several research questions are generalizable over other encrypted traffic datasets by using three additional datasets. According to the results from experiments on additional datasets, we concluded that the observations and conclusions made are valid across the majority of the datasets.

Traffic analysis and defenses have always been an arms race between attackers and defenders, and it is likely to continue in the future with improved attacks defeating state-of-the-art defenses. Nonetheless, our work provides a methodical approach to look into future attacks and propose more efficient defenses. As the observations and conclusions we reach are based on the current datasets used, we acknowledge that network traffic with different behavior compared to the datasets we used may result in different observations and conclusions. But we believe that the framework we proposed can be used

as fundamentals to devise more suitable methods specific to a given model and dataset. Furthermore, as the field of explainable AI [114, 115] further evolves, methods that can be used to analyze the inner working of traffic classifiers can be improved. As of now, there are methods that can explain classifiers other than CNNs such as LSTMs [116], MLPs [117], and AEs [118], and what kinds of insight they provide when it comes to traffic fingerprinting is an interesting research direction to follow.

Chapter 5

Defenses against Traffic Fingerprinting

Previous chapters discussed how traffic fingerprinting attacks are becoming increasingly realistic due to recent advances in machine learning techniques and pose significant threats to the privacy of internet users. Hence, it is of extreme importance to develop efficient defenses against such attacks. It should be noted that while these defenses are expected to reduce the risk of traffic fingerprinting attacks significantly, they should also minimize the effect on the expected functionality of the original applications generating the traffic. More specifically, these defenses should ensure that they do not incur significant bandwidth and timing overheads which would affect the functionality of the applications and render the defenses impractical.

Multiple previous research has attempted to devise defenses against traffic fingerprinting attacks. However, recent attacks have been proven resilient against most of them [14]. In this chapter, we leverage the insights we obtained earlier on the inner working of traffic fingerprinting CNNs in Chapter 4 to propose two novel and efficient defenses, *FRONT-U* against website fingerprinting and *STOMA* against video streaming traffic fingerprinting that provide reasonable privacy at affordable data and timing costs under more realistic assumptions.

We evaluate both our defenses in a realistic setting where the attacker can observe defended traffic and is able to train new classifiers on the observed defended traffic. It should be noted that we simulate defenses by modifying pre-captured traffic (from

previous work) according to the corresponding defense algorithm. Accordingly, we use the accuracy of a model trained on defended traffic as a measure of the privacy provided by the defense. If the test set (defended) accuracy of a model trained on defended traffic is much lower than the accuracy of non-defended traffic on a model trained on non-defended traffic, we consider that the defense provides adequate privacy. We also define data overhead as the percentage of dummy data added compared to the amount of real data of a trace and use the data overhead of a defense as a measure of the cost incurred by the defense. A higher data overhead would imply that the defense needs to transmit more dummy data, thereby increasing the cost of the defense.

5.1 State-of-the-art defenses against traffic fingerprinting

We first present FRONT [24], the current state-of-the-art defense against website fingerprinting attacks, and then we introduce two differential privacy-based defenses against video traffic fingerprinting attacks which are the current state-of-the-art in video stream fingerprinting. We explain these methods in detail because the proposed defenses improve upon these existing methods and use them as baseline defenses to compare with the performance of the proposed methods.

5.1.1 FRONT: Defense against website fingerprinting

Gong et al. [24] observed that some prior website fingerprinting attacks [9, 10] explicitly used trace fronts for the classification task and hence decided to allocate the majority of the padding budget to defend the trace front. We note that this observation corroborates our findings in Chapter 4.2.4. Accordingly, FRONT requires both the server and the client to independently add dummy packets to the outgoing stream of packets, adding more padding to trace fronts to obfuscate actual traffic patterns. At the beginning of each website visit, the client and the server each pick a random number of dummy packets they would add to the channel and then use a Rayleigh distribution to choose the timestamps to insert those packets. The Rayleigh distribution (as given in Equation (5.1)) is selected to ensure that more padding is added to the trace fronts than the rest of the trace. According to Equation (5.1), for a given w value, the probability density function first increases quickly, peaks at w, and then gradually decreases, resulting in a burst of

dummy packets at the beginning of a trace. As a result, FRONT outperformed the then best-performing defense WTF-PAD [23] with approximately the same amount of data overhead.

$$f(t;w) = \begin{cases} \frac{t}{w^2} e^{\frac{-t^2}{2w^2}} & t \geq 0 \\ \\ 0 & t < 0 \end{cases} \tag{5.1}$$

Additionally, to further randomize the padding for each website visit, FRONT requires both client and server to pick the number of dummy packets to add and the specific Rayleigh distribution to choose time stamps from (by picking a random textitw value) before every communication session.

5.1.2 Differential privacy for defense against video stream fingerprinting

With the aim of seeking a *principled* solution to counter video stream fingerprinting attacks, Zhang et al. [25] explored the feasibility of using differential privacy to design a defense. More specifically, the authors aimed to develop ϵ-deferentially private defenses for streaming traffic, which by adding random noise dictated by the parameter ϵ and threshold t, can render two video streams within distance t to be statistically indistinguishable from each other. The authors used two different methods FPA_k and d^*-*privacy* to enforce differential privacy on video streaming traffic.

5.1.2.1 Fourier Perturbation Algorithm

Given a non-defended streaming traffic trace $Q = \{q_i\}_{i\in(0,n)}$ which is a sequence of n integers where each element q_i in the sequence corresponds to the total number of bytes transmitted in a given direction during i^{th} window/bin, FPA methods transforms the entire traffic trace into the frequency domain using Discrete Fourier Transform (DFT) before adding dummy data to each Fourier component and then returning the inverse DFT as the defended trace. It should be noted that this requires the entire traffic trace to be known before the defense can begin. Given that $Lap(\lambda)$ is a random variable drawn from the Laplace distribution with scale λ and location $\mu = 0$, $\Delta_2(\mathbb{Q})$ denotes

the L_2 sensitivity of a set of Qs, $PAD^n(Q)$ denotes padding the sequence Q with zeros up to length n, $DFT(X)$ denotes the Discrete Fourier Transform of X and $IDFT(X)$ denotes the Inverse DFT of X, the FPA method of adding dummy data to streaming traffic is given in Algorithm 3. Here, ϵ is a measure of the amount of dummy data added to an element where lower ϵ would add more dummy data increasing the data overhead and privacy of the defended trace.

Algorithm 3: FPA on streaming traffic [25]

Require: Undefended trace Q of length n, $k \in \mathbb{Z}^+$ and ϵ

Let: $\lambda = \sqrt{k}\Delta_2(\mathbb{Q})/\epsilon$

Calculate $(DFT(Q))$ and keep the first k Fourier coefficients $F[1], ..., F[k]$.

for ($i = 0;\ i < k$) {

  $F[i] = F[i] + Lap(\lambda)$

$\tilde{Q} = IDFT(PAD^n(F[1], ..., F[k]))$

return $\tilde{Q}$

5.1.2.2 d*-private method

Given a non-defended streaming traffic trace $Q = \{q_i\}_{i\in(0,n)}$ which is a sequence of n integers where each element q_i in the sequence corresponds to the total number of bytes transmitted in a given direction during i^{th} window/bin, d* privacy mechanism adds dummy data to each packet in real-time according to a predefined procedure. Given that $D(i) \in \mathbb{N}$ denotes the largest power of two that divides i, the d* privacy mechanism for adding dummy data to a streaming traffic trace is given in Algorithm 4. Here, ϵ is a measure of the amount of dummy data added to an element where lower ϵ would add more dummy data increasing the data overhead and privacy of the defended trace.

5.2 FRONT-U - Defending Web Traffic

Next, we consider defending against website fingerprinting CNNs. In Chapter 4.2, we highlighted that website fingerprinting CNNs focus more on the initial part of a trace compared to the rest of the trace and on the transitions between uploads and downloads instead of continuous periods of uploads or downloads. At the same time, we discussed in Chapter 5.1.1 about FRONT [24], a defense against website fingerprinting, which

Algorithm 4: d* private mechanism for defending streaming traffic [25]

Require: Undefended trace Q of length n and ϵ

Let: $\mathrm{G(i)} = \begin{cases} 0 & i = 1 \\ i/2 & i = D(i) \geq 2 \\ i - D(i) & i < D(i) \end{cases}$

Let: $\mathrm{r}_i \sim \begin{cases} Lap(\frac{1}{\epsilon}) & i = D(i) \\ Lap(\frac{\lfloor \log_2(i) \rfloor}{\epsilon}) & otherwise \end{cases}$

$Q[0] = \tilde{Q}[0] = 0$

for ($i = 1;\ i < n$) {

  $\tilde{Q} = \tilde{Q}[G(i)] + (Q[i] - Q[G(i)] + r_i)$

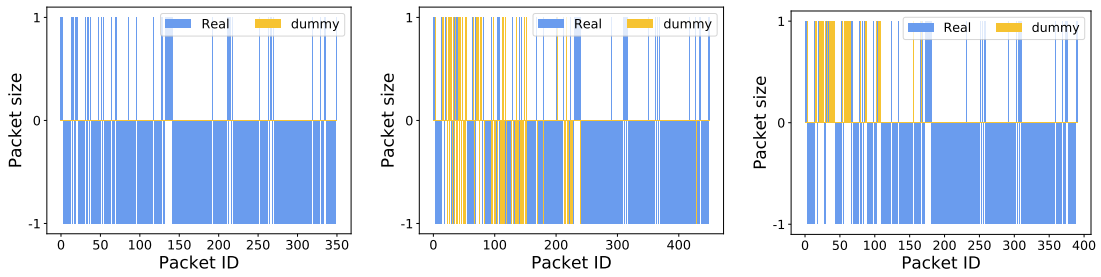
return $\tilde{Q}$

is based on the same idea that the initial part of a trace is more important for the classification.

Although FRONT already leverages our first observation (i.e., trace fronts are more important), it does not leverage our second observation (i.e., website fingerprinting CNNs focus more on the transitions between uploads and downloads). We leverage the second observation to extend FRONT to have less overhead while also removing the constraint that both the server and the client need to participate in the defense.

Accordingly, we propose FRONT-U (U for upload), in which only the client is involved in padding. The main reason behind having only one party doing the padding is that since the attacker is focusing on transitions between uploads and downloads, changing the patterns of only one direction would reduce the ability of the fingerprinting CNN to identify the trace. Since it is common to have fewer uploads compared to downloads when loading a webpage, we decided to pad from the client side only in order to minimize the total amount of padding used. Thus, FRONT-U follows the procedure of FRONT, but with only the client side participating in the defense. Before each website visit, the client would decide on the number of dummy packets to add and then use a Rayleigh distribution to choose the timestamps for those dummy packets, which will be sent out during the website load.

In Figure 5.1, we illustrate a random sample from the FRONT dataset when not defended, when defended with FRONT and when defended with FRONT-U. According to the figure, we see that both FRONT (Figure 5.1b) and FRONT-U (Figure 5.1c)



(A) Non-defended (B) FRONT defended (C) FRONT-U defended

FIGURE 5.1: Defense comparison: Website fingerprinting

have dummy padding added with a higher concentration towards the trace front. Additionally, FRONT-U (Figure 5.1c) adds padding from the client side only resulting in significantly less total data overheads compared to FRONT (Figure 5.1b).

To evaluate the performance of FRONT-U, we simulate the defenses using the same dataset with 101 classes provided by the authors of FRONT in their code repository. This dataset consists of 100 closed set classes and one class representing the open set. The attack models on which we evaluate the defended traces are trained according to the background class method and therefore, we note that our results correspond to the more challenging open set scenario. We note that the implementation of FRONT and FRONT-U requires the time stamps of the original Tor packets so that dummy packets can be inserted at the correct positions. Publicly available versions of both AWF and DF datasets contain only packet sizes and direction, and as a result, those two datasets could not be defended using FRONT and FRONT-U methods. We compare the performance of DF models [14] trained on un-defended, defended using FRONT, and defended using FRONT-U data. We show the results in Table 5.1.

TABLE 5.1: FRONT vs FRONT-U

Method	**Accuracy**	**Data Overhead**
Non-defended	95.42%	N/A
FRONT	72.38%	43.15%
FRONT-U	76.74%	19.8%

In Table 5.1, we see that FRONT-U reduces the data overhead by approximately half while achieving similar protection as the original FRONT. Furthermore, in Table 5.2, we show the performance of FRONT-U against LSTM and AdaBoost classifiers which

demonstrates that even though designed based on the behavior of traffic fingerprinting CNNs, FRONT-U is still effective against other classifiers as well.

TABLE 5.2: FRONT vs FRONT-U against LSTM and AdaBoost

	Accuracy against Classifier		
Method	**CNN**	**LSTM**	**AdaBoost**
Non-defended	95.42%	91.33%	66.33%
FRONT	72.38%	47.06%	48.83%
FRONT-U	76.74%	64.83%	52.61%

5.3 STOMA - Defending Video Streaming Traffic

Next, we consider defenses against video fingerprinting CNNs. In Chapter 4.2, we highlighted that video fingerprinting CNNs focus on the finer sub-bursts associated with a major burst instead of on the envelope of the burst signal. We leverage this insight to devise a defense against such attacks by replacing small windows of a trace with the mean number of bytes in that direction within that window to obfuscate the finer sub-bursts within a major burst. Based on this approach, we introduce *Streaming Traffic Obfuscation with Moving Average (STOMA)* that uses Algorithm 5 to defend streaming traffic.

Given an un-defended trace $x(t)$ of length n with $t = 1, 2, ...n$, the defended trace $\tilde{x}$ will be calculated as given in Algorithm 5. Here, k refers to the window of the trace being averaged at a given time, and s refers to the stride length used to move the averaging window. Note that to calculate the i^{th} element of the defended trace $\tilde{x}[i]$, we only need to observe the next k elements of the non-defended trace, $x[i : i + k]$. Accordingly, increasing the upper bound of the uniform distribution where k is picked from will increase the window length being averaged, which helps obfuscate the bursts better. However, it will in turn, increase the number of elements of the non-defended trace that needs to be known in advance before defending.

In Figure 5.2, we demonstrate the threat model related to the proposed defense. As shown in the figure, both the client side and the server side independently participate in padding their outgoing traffic according to Algorithm 5.

We evaluate STOMA by simulating the defenses on the DC dataset for the feature *Combination Number of Bytes (data)*. We use this feature instead of the number of

Algorithm 5: Streaming Traffic Obfuscation with Moving Average (STOMA) Algorithm

Require: Undefended trace $x(t)$ where $t = 1, 2, ...n$

Given $U(a, b)$ is the uniform distribution between a and b,

Pick k from $U(32, 64)$ and s from $U(2, k)$

for ($i = 0;\ i < (n - k) + 1;\ i = i + s$) {

  $\tilde{x}[i : i + k] = ceil(average(x[i : i + k]));$

if $i + s < n$ **then**

  $\tilde{x}[i + k :] = ceil(average(x[i + k :]))$

return $\tilde{x}$

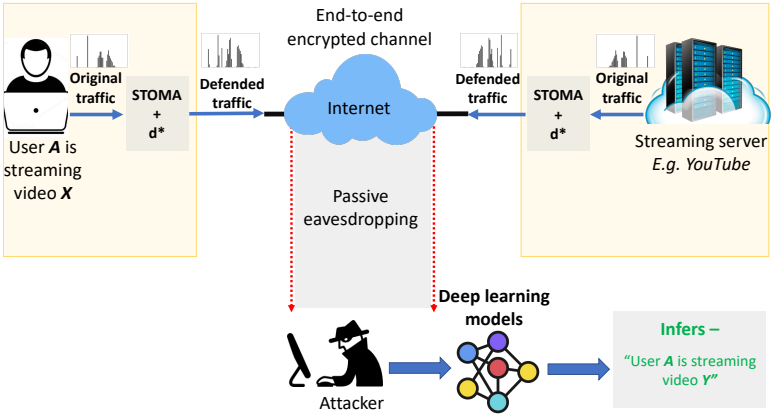


FIGURE 5.2: STOMA threat model

packets in the uplink as used in previous chapters, as it is more practical to alter the number of bytes in both directions than the number of packets in the uplink only. Unlike Tor website visits we considered earlier, which have a fixed packet length, video streaming packets can be padded with dummy bytes.

We compare STOMA's performance with state-of-the-art defenses, d* private, and FPA (Fourier Perturbation Algorithm) methods introduced in [25]. As explained in Chapter 5.1.2.2, the d* private method adds dummy data to each packet in real-time according to Algorithm 4, while FPA method discussed in Chapter 5.1.2.1 assumes the entire trace is known in advance and converts the entire trace to the frequency domain before adding dummy packets as explained in Algorithm 3. Both these methods require a variable ϵ which defines the amount of dummy data to be added. A lower ϵ would imply a larger amount of dummy data and hence more data overhead.

It should be noted that in contrast to assuming that the complete trace is known by the defender in advance as in the FPA method, we make the assumption that only the next t seconds window (n bins $\times$ bin width in seconds) data is known by the defender in

advance. In this particular case, the window size is 24 seconds (64 bins x 0.36s). When it comes to streaming data assuming the prior knowledge of data for only a particular window is more realistic than assuming the prior knowledge of the whole data stream, which can change over time. We show our results in Table 5.3.

TABLE 5.3: Performance of STOMA and other methods

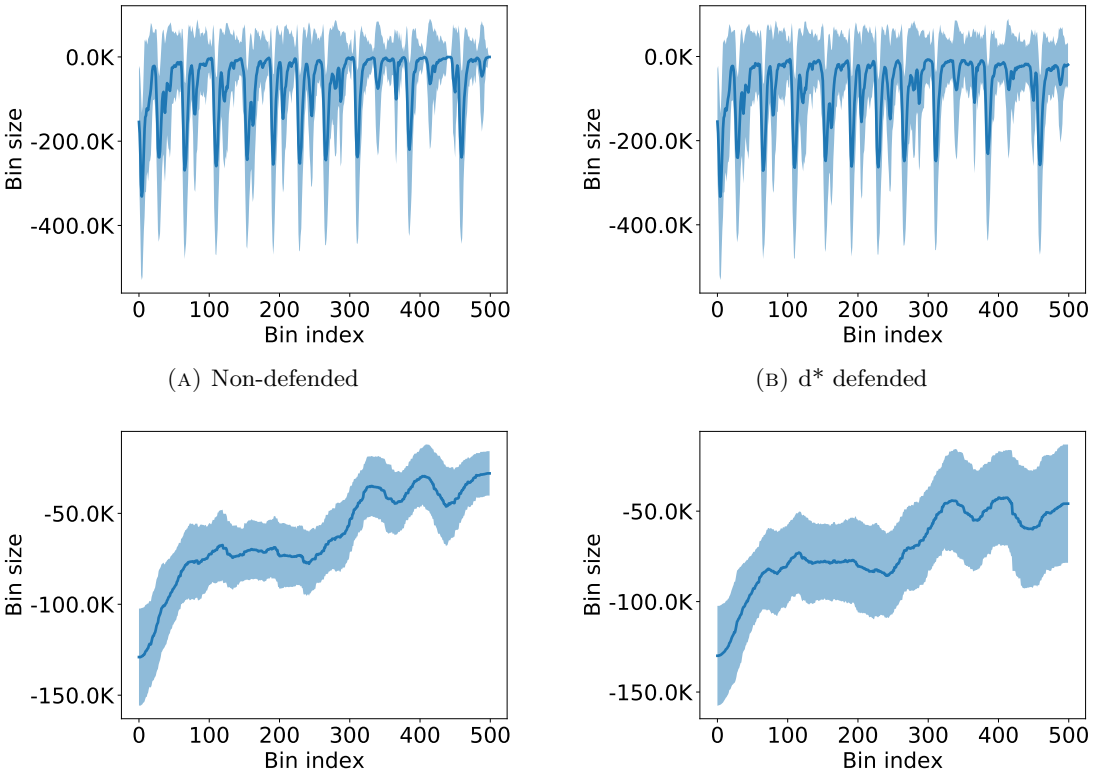
Method	**OH up-link**	**OH down-link**	**Accuracy**
Non-defended	0.00%	0.00%	96.46%
STOMA	0.07%	0.01%	90.75%
d* privacy	763.71%	145.09%	93.63%
STOMA + d* privacy	678.70%	103.62%	52.50%
FPA	21.47%	99.80%	18.31%

According to Table 5.3, we see that even though the d* privacy method and STOMA do not provide reasonable privacy by themselves, the combination of the two methods provides adequate privacy (reduction in classifier accuracy). It should be noted that when the ϵ value in d* private mechanism is increased, the overhead added is increased, resulting in better privacy. Similarly, for STOMA, if the window size (k) used is increased, the privacy provided by the method increases. However, when using these methods in combination, the defense provides sufficient privacy with relatively low overheads. Furthermore, in Table 5.4, we show the performance of STOMA against LSTM and AdaBoost classifiers which demonstrate that even though designed based on the behavior of traffic fingerprinting CNNs, STOMA is still effective against other classifiers as well.

TABLE 5.4: STOMA and other methods against LSTM and AdaBoost classifiers

	Accuracy against Classifier		
Method	**CNN**	**LSTM**	**AdaBoost**
Non-defended	96.46%	94.84%	94.04%
STOMA	90.75%	91.09%	92.63%
d* privacy	93.63%	89.69%	90.91%
STOMA + d* privacy	52.50%	57.66%	65.20%
FPA	18.31%	35.16%	27.59%

In Figure 5.3, we have illustrated the mean class trace for class 0 under four scenarios, non-defended, d* defended, STOMA defended and Combined defended. Accordingly, if we compare the non-defended trace (Figure 5.3a) and d*-private defended trace (Figure 6.4b), we see that even though the d* method (with $\epsilon = 0.0005$) increases the amount of variance between traces of the same class, it fails to hide the burst pattern of



(A) Non-defended

(B) d* defended

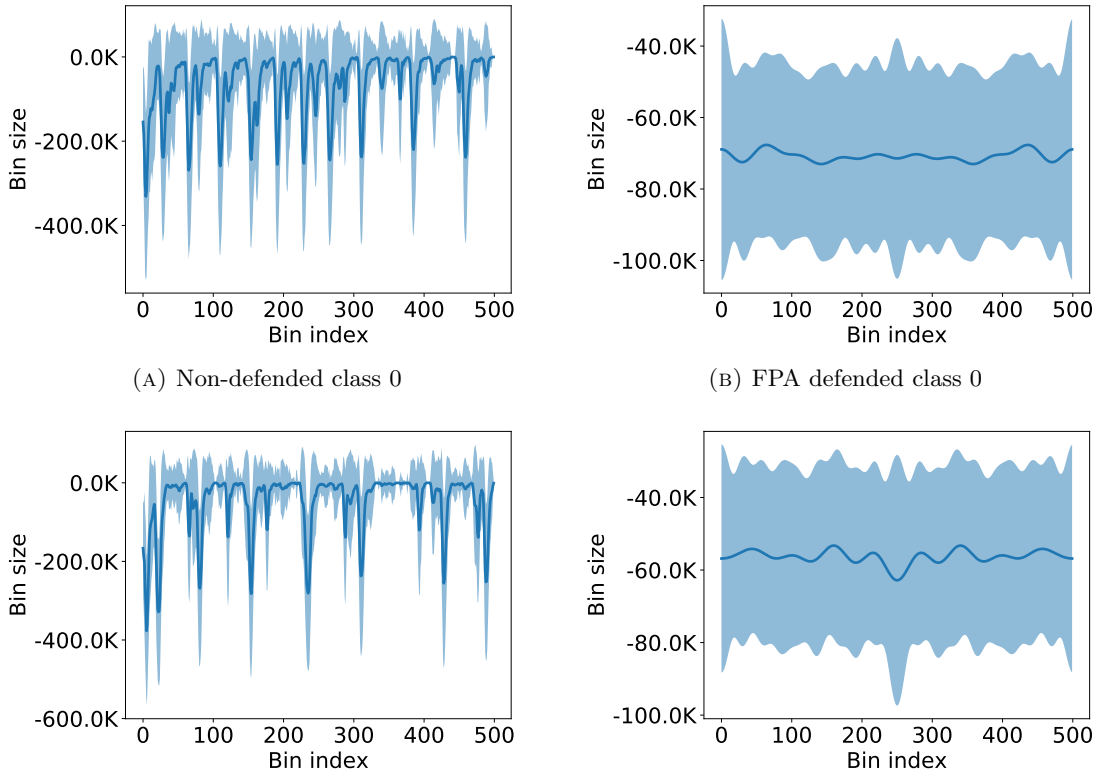
(C) STOMA defended

(D) Combination

FIGURE 5.3: Mean class trace for class 0 for varying defenses

the class. On the other hand, when comparing the non-defended trace (Figure 5.3a) and STOMA defended trace (Figure 5.3c), we see a complete transformation in the burst pattern, but with less variance among traces of the same class. With similar graphs on all other classes, we also observe that even though STOMA is able to change the original burst pattern of the class, the resulting pattern is still unique for each class, which causes higher attack accuracy when trained on defended data. In Figure 5.3d, we see that when the two methods are combined, the original burst pattern of the class is changed, and more variance is seen among traces of the same class, which results in better privacy.

In Figure 5.4, we have shown the mean class trace for class 0 and 5, defended with FPA method (which provides good privacy according to the results in Table 5.3) vs non-defended. According to the figures, we see that the FPA method completely alters the burst pattern of the trace, resulting in an almost flat line. It suggests that if the FPA method is used as a defense, the packet transmission is required to happen at a constant rate using large buffers, which would affect the implementation of the method



(A) Non-defended class 0

(B) FPA defended class 0

(C) Non-defended class 5

(D) FPA defended class 5

FIGURE 5.4: Mean class trace of non-defended vs FPA defended traffic

over DASH. Compared to this, the combination of moving average and d* methods still preserves the bursty nature of the trace, making it relatively easier to be implemented over DASH. Moreover, unlike FPA, STOMA and d* privacy combination does not require the prior knowledge of the full trace but only the traffic pattern of a single window, which is more suitable and realistic for streaming applications.

5.4 Practical Applicability of Proposed Defenses

Finally, we discuss the practical applicability of our proposed defenses compared to state-of-the-art methods.

5.4.1 FRONT-U

As described in Chapter 5.2 in FRONT, both the server and the client decide the number of dummy packets to add and the exact time stamp to add them to the actual trace,

whereas in FRONT-U, this is decided by the client. This happens before the actual communication, and hence padding is done in real-time incurring zero timing overheads (i.e., actual packets are sent out without a delay).

Adding zero delay to the actual packets while significantly reducing the attack accuracy makes both defenses very practical solutions against website fingerprinting. To provide the same level of privacy, FRONT-U requires only almost half of the overhead of FRONT, and as such, more suitable for bandwidth-conscious real-world applications.

Finally, FRONT-U requires padding only from the client side. This provides flexibility to the user to decide whether or not to use the defense and how much padding to be added (which is proportional to the privacy provided) without needing the participation of the server. Overall, these features make FRONT-U a more realistic defense that provides flexibility to the users.

5.4.2 STOMA

In Chapter 5.3 we showed that the combination of STOMA and d* privacy could significantly reduce the attack accuracy as opposed to using either of the methods individually. Although FPA method provides better privacy with a lower data overhead compared to the combination of STOMA and d* privacy method, the FPA method requires the behavior of the entire traffic trace to be known in advance, which is not realistic.

In contrast, our proposed method requires the knowledge of only the next 24 seconds (this is configurable) of the trace, which can be achieved by adding a delay to the actual packets and is more realistic compared to assuming knowledge of the full trace in advances as in FPA. Furthermore, in Figure 5.4 in Chapter 5.3, we showed how FPA requires packet transmission to happen at a constant rate using buffers which affects the implementation of the defense over DASH. Comparatively, our method still preserves the bursty nature of the video streaming transmission, which allows the DASH player to still be able to potentially detect network changes. Indeed, video streaming over DASH makes use of this variable bit rate traffic to optimize the quality of experience for the user by reacting to potential network changes.

To summarize, we leverage the observations from Chapter 4.2 to propose two new defenses against traffic fingerprinting CNNs. We introduce FRONT-U: a defense against website fingerprinting that provides similar privacy as the state-of-the-art defense with half the data overhead, and STOMA: a defense against video fingerprinting that provides reasonable privacy under more realistic assumptions compared to previous work.

5.5 Discussion and Concluding Remarks

In this chapter, we leveraged observations and insights we obtained in Chapter 4, to design efficient defenses against traffic fingerprinting attacks that provide reasonable privacy at affordable costs and compare them with state-of-the-art defenses.

Based on the results from Chapter 4, we proposed FRONT-U, a defense against website fingerprinting CNNs that provides similar privacy as the state-of-the-art with approximately half the data overheads by focusing more on obfuscating initial parts of a trace and transitions between uploads and downloads, and STOMA a defense against video fingerprinting CNNs that provides adequate privacy under more realistic assumptions compared to previous work by obfuscating sub-burst shapes within bursts of streaming traffic. Though we derived the insights that formed the basis of our defenses by dissecting traffic fingerprinting CNNs, we also showed that the defenses we proposed are also capable of mitigating threats posed by other classifiers such as AdaBoost and LSTM.

Traffic analysis and defenses have always been an arms race between attackers and defenders, and it is likely to continue in the future with improved attacks defeating state-of-the-art defenses. Also, future work should look into further reducing the overheads incurred by defenses in order to make them more feasible to be used in real-world applications. In designing novel defenses, another possible avenue to explore is the idea of universal perturbations. Work in computer vision has shown the existence of a universal (image-agnostic) and small perturbation vectors, that cause natural images to be misclassified with high probability across different deep neural networks [119]. If such an input agnostic perturbation that can cause a network trace to be misclassified by an attacking deep neural network exists, it can be used to defend network traffic against fingerprinting attacks in real-time. That is because a pre-calculated perturbation can be used to add dummy packets to the network trace without causing timing delays. This

idea has been explored by a very recent work [82] to defend against website fingerprinting attacks over Tor. Future work can compare its performance, overheads, and effectiveness against multiple deep learning and traditional machine learning-based attack models with other recent defenses like FRONT to identify a more practically applicable defense. Furthermore, future work can also explore whether a similar approach can be used to defend against other types of traffic fingerprinting attacks, such as video fingerprinting attacks.

Chapter 6

Open set Classification for Encrypted Network Traffic

Deep Learning models perform exceptionally well with high accuracies across multiple domains such as image processing and natural language processing. However, they are mostly developed under the closed set assumption where all classes that the model may encounter at inference are represented in the training data. However, when deployed in real-world applications, the closed set assumption becomes invalid because it is not possible to predict all classes the model might encounter in advance and it cannot be guaranteed that training data is available for all known classes. Therefore, using deep learning-based models in practice requires addressing the *open set problem*. That is these models should be able to correctly handle known classes while effectively identifying unknown classes as well.

Traffic fingerprinting models based on deep learning models also face the closed set limitation. However, when deployed in real-world applications, traffic fingerprinting models must be able to adapt to the open set setting where they can separate the traffic flows it can classify *(known-knowns)* from all other traffic flows *(unknown-unknowns)*. A majority of existing work in encrypted traffic fingerprinting either,

(i) focused only on the closed set problem [16, 113]

(ii) addressed the problem by adding a background class or a binary classifier to separate knowns from unknowns [14, 18, 68]

(iii) used thresholding on classifier confidence to filter unknowns [15].

Nonetheless, these methods have their own limitations as discussed in Chapter 2.2. For example, using a background class or binary pre-classification requires samples from background traffic *(known-unknowns)* during training time. There is no guarantee that such available samples at training time will cover all *unknown-unknowns* the classifier may come across in the future. We explain this further in Figure 6.1.

We use the split 5 of the DC dataset and the corresponding background class model trained in Section 6.3, to get the output vector from the Softmax layer for the corresponding test set. Next, we plot the 2-dimensional t-SNE graph [120] for the output values as shown in Figure 6.1. We note that in this specific split, the classes [0, 1, 3, 5] make up the closed set while classes [6, 7, 8, 9] make up the open set. Classes 2 and 4 are used as the known unknowns.

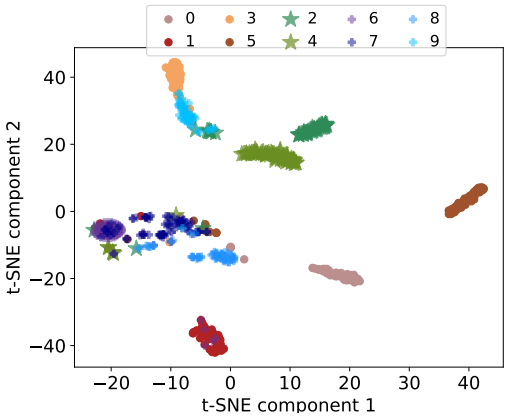


FIGURE 6.1: DC: t-SNE plot for background class method

As can be seen from Figure 6.1, the samples from closed set classes (brownish shades) form four distinct clusters while the samples from the two classes used as known unknowns (greenish shades) are also clustered close together. However, the samples from the open set classes (blueish shades) which ideally should be clustered closer to known-unknowns (i.e, greenish clusters), can be seen clustered either with a specific closed set class (i.e, class 9 clustered close to class 3) or as a separate cluster further away from the known-unknown classes. Notice how none of the open set classes cluster close to the two known-unknown classes (greenish). This confirms how for some datasets, open set samples would have more similarities with closed set classes as opposed to known-unknowns. In such a scenario, the background class method fails.

Another naive approach to open set classification is thresholding on the classifier's confidence, which is based on the idea that the classifier must have high confidence for known classes. However, current deep learning models are known to have high confidence even if they are making a mistake [33]. In Figure 6.2, we demonstrate an example scenario where softmax thresholding does not perform well. Here, we used the second split of the SETA dataset and feed both closed and open set test sets to the corresponding closed set classifier trained in Section 6.3. Then we extract the softmax score for the predicted class for each sample. In Figure 6.2 we show the histogram of these softmax scores drawn to a log scale. Note that the sub-figure shows the histogram for the entire range of softmax scores ([0.0-1.0) while the main figure shows the zoomed-in version to clearly emphasize the section of the figure corresponding to the score in the range [0.995-1.0]. The figure shows how almost all closed set samples have softmax scores greater than 0.999 as expected. However, over 50% of open set samples also have softmax scores in the same range. This makes rejecting open set samples by thresholding on softmax ineffective, especially when sacrificing closed set accuracy is not acceptable.

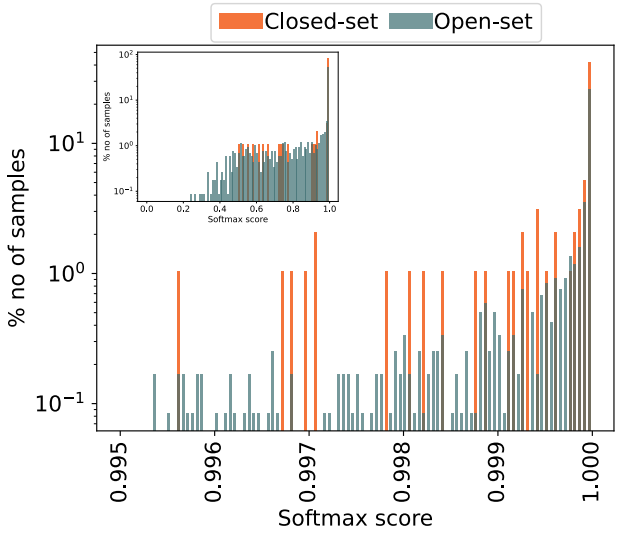


FIGURE 6.2: SETA: Histogram for Softmax scores

As such, encrypted traffic fingerprinting models must be able to handle the *open set* classification more realistically, where the model can correctly identify samples from known classes (i.e., known traffic flows) while effectively rejecting any samples from classes not seen during model training (i.e., background traffic flows).

Another key issue in deploying traffic fingerprinting models in real-world applications is

their high memory and computational power requirements. In practice, these fingerprinting models need to be deployed on *in-network computing devices* such as programmable switches, which have limited memory and computational power [121, 122].

To this end, in this chapter, we propose a novel framework for robust open set classification of encrypted network traffic based on three key ideas. First, we show that a well-regularized underlying closed set classifier improves the performance of an open set classifier. Next, we show that traffic fingerprinting models can be quantized without a significant drop in accuracy. Finally, we propose a novel open set classification method based on *k-logit neighbor distances* with three variants that perform reliably across all datasets considered, with $> 85\%$ closed set accuracy and $> 65\%$ open set accuracy.

6.1 Background

In this chapter, we first detail a typical encrypted traffic fingerprinting scenario and explain the importance of open set classification. Next, we provide a brief overview of the existing open set classification methods that are most commonly used in encrypted traffic fingerprinting. Finally, we introduce five publicly available datasets and their corresponding deep CNN classifier architectures that we use in our experiments.

6.1.1 Open set traffic classification scenario

In Figure 6.3, we illustrate a typical traffic classification use-case where it is essential to have an open set classification component. Assume the law enforcement is trying to identify illegal activities happening over Tor and has the ability to passively observe encrypted Tor traffic in transit (e.g., at a vantage point of an ISP). They have identified a list of websites that are used for illegal activities (i.e., the target list), and they want either to identify users who are browsing those websites or simply to terminate those connections. Other than that, law enforcement does not have any interest in all the other website visits (i.e., websites that are not in the target list) happening over Tor.

To achieve this, first, the law enforcement can collect a dataset of traffic flows by visiting websites in the target set themselves (i.e., the known classes or known knowns) and train a classification model. However, this naive closed set model will have the limitation of

predicting any traffic flow given to it as the input as one of the websites in the target list. To avoid this, the law enforcement can collect samples of other website visits that are not in the target set (i.e., known-unknowns) and add all of these samples as a single class during the training process (i.e, the background class). However, this method may not perform well when unknown unknowns (i.e., websites that were not shown to the classifier during training) are fed as inputs. Alternatively, the law enforcement can reject samples with very low softmax scores (below a threshold) for their predicted classes assuming that the model will have low confidence for unknown samples (i.e., softmax thresholding). However, this method also may not always work well as the classifier was not explicitly trained to output lower softmax scores for unknown samples.

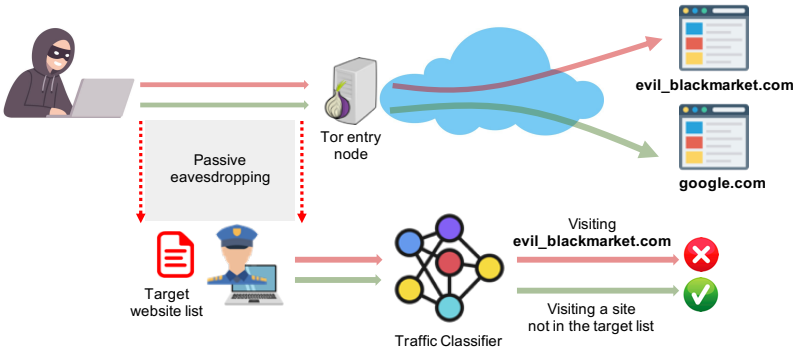


FIGURE 6.3: Traffic classification scenario

6.1.2 Open set classification methods

In Chapter 2.2, we introduced two naive methods; background class and softmax thresholding, OpenMax [33], which is one of the first open set classification methods proposed over deep learning methods and ensemble learning method. As Chapter 2.2 already explains these methods in detail, here, we briefly summarize them.

- **Background class**: Treat the open set as another class and train an $n + 1$ class classifier (n is the number of known classes) using a subset of the open set
- **Softmax thresholding**: Use a predefined threshold on the softmax score of the predicted class to reject samples with low confidence value as open set (threshold can be decided using the validation set to maintain a preferred closed set accuracy)
- **OpenMax**: Use EVT modeling on the distance between a sample and the class mean of the predicted class to refine class scores.

- **Ensemble learning**: Combine the output from multiple simple learner models and use softmax thresholding on the combined output

6.1.3 Datasets

We use five publicly available datasets corresponding to three types of encrypted network traffic as described below.

1. **Website fingerprinting over Tor:** *Automated website fingerprinting (AWF) [15]* and *Deep fingerprinting (DF) [14]* datasets contain network traffic traces for visiting homepages of top-200, and top-95 Alexa websites over `Tor`, respectively. In these datasets, each site visit is represented by the first 5,000 (DF)/3000 (AWF) `Tor` packets in either direction. That is, in these datasets, a data sample is a sequence of +1s (uploads) and -1s (downloads). If a particular homepage visit did not generate a total of the respective number of packets in either direction, the remainder of the sequence was padded with zeros.

2. **Video fingerprinting:** *Deep content dataset (DC)* [16] contains traffic traces for streaming the first three minutes of selected YouTube videos, while *SETA [113] dataset* contains the same for selected Netflix videos. In both datasets, the three-minute interval is binned into 500 time slots (0.36s each), and each time slot is represented by summary statistics of the packets observed during that time. While the original datasets comprised 24 features per trace, the authors of DC observed that the number of uplink packets of video streaming produced the most accurate model, and hence we only use that feature in our work. Therefore, these datasets represent a traffic trace as a sequence of 500 integers.

3. **Voice command fingerprinting:** The *IoT [18] dataset* contains the traffic traces generated by Google Home devices for 98 specific voice commands. It represents a trace by the first 475 packets in either direction and for each packet, a -1 would denote a download and a +1 would denote an upload. If a particular voice command did not generate a total of 475 packets in either direction, the remainder of the sequence was padded with zeros.

   We provide a summary of our datasets in Table 6.1. We split DC, SETA and IoT datasets to create open sets, as discussed below.

TABLE 6.1: Summary of datasets

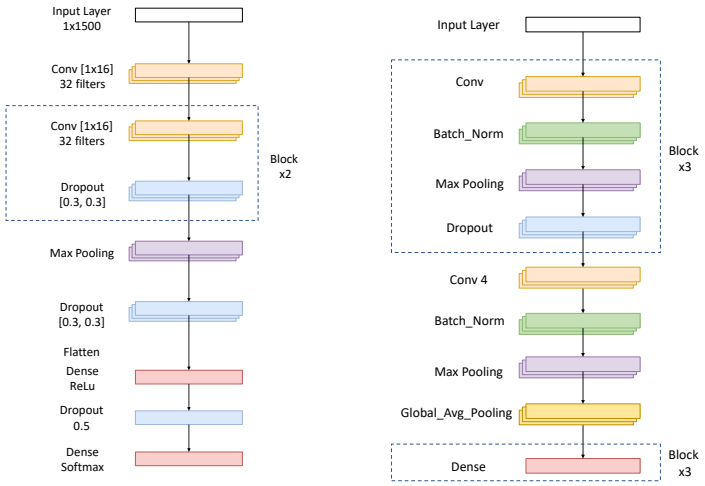
Dataset	Type	Details	Open set
AWF [15]	Website	200 websites (2,500 traces/class)	400,000 classes
DF [14]	Website	95 websites (1,000 traces/class)	40,716 classes
DC [16]	Video	10 YouTube videos (320 traces/class)	N/A*
SETA [113]	Video	20 Netflix videos (100 traces/class)	N/A*
IoT [18]	Voice	98 Google Home comms. (1,500 traces/class)	N/A*

*We split the classes so that 40% of classes are in the closed set.

Data preparation: The original works of AWF and DF considered the open set scenario and hence already has a separate open set. Therefore we did not have to manually split the dataset. In contrast, the original versions of DC, SETA, and IoT datasets did not have open set samples. Therefore, to use these in our open set experiments, we split the original datasets into two parts so that 40% of the original number of classes is used as the closed set while the rest is considered as the open set. To negate any effect on results from specific splits, we use multiple random splits for each dataset and report the average performance. The number of splits depends on the datasets. For DC and SETA datasets which have a smaller number of classes, we created five splits, and for the IoT dataset, we created 10 splits. Later, when we report the results for these datasets, we report the average and standard deviation values of each metric across all splits of a given dataset.

Unless otherwise specified, for all the methods, the training, testing and validation splits from the closed set contain 200, 200, and 100 traces per class, respectively. While more data samples were available in the datasets, the original work as well as subsequent work [15] and Chapter 4 showed that 200 training samples per class are sufficient to train a model with high test accuracy.

For the background class method, which is the only method that requires open set samples during training (i.e., known-unknowns), we separate out 20% of classes as known-unknowns and use 200 and 100 traces from each known-unknown class in the training and validation sets, respectively. We ensured there is no overlap between the classes used for open set during training and testing procedures. The test set comprised all the known and unknown class samples from the original test set.



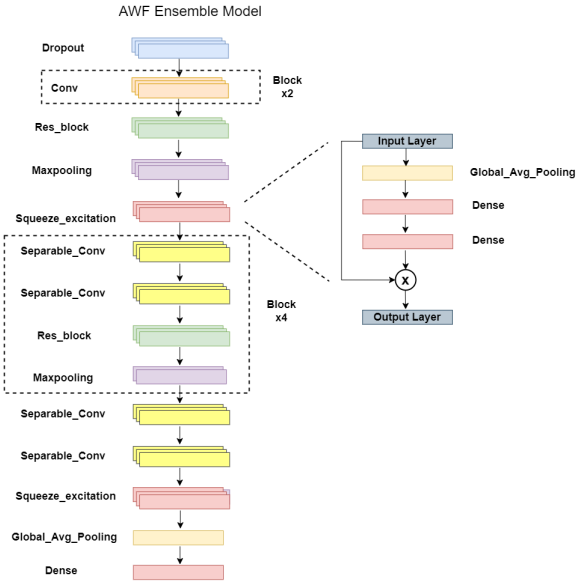
(A) SETA (B) IoT

FIGURE 6.4: Deep learning model architectures

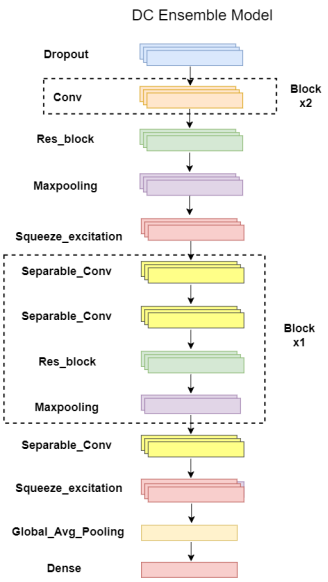
6.1.4 Deep learning models

All of the open set methods we use require an underlying deep neural network. For each dataset except AWF and SETA, the original work proposed the most suitable model architectures and hyperparameters and hence we use those models as it is in our work. For AWF dataset, we use the model proposed for DF [14] since Sirinam et al. [14] showed that DF model is more effective for website fingerprinting. The original work for SETA did not use deep learning models, and hence we use a deep CNN similar to that of DC. The DF model used for AWF and DF datasets were given in Figure 6.3-(a) while the DC model was shown in Figure 6.3-(b). The model architecture used for SETA dataset is shown in Figure 6.4a while that for IoT dataset is shown in Figure 6.4 (refer [18] for parameter values).

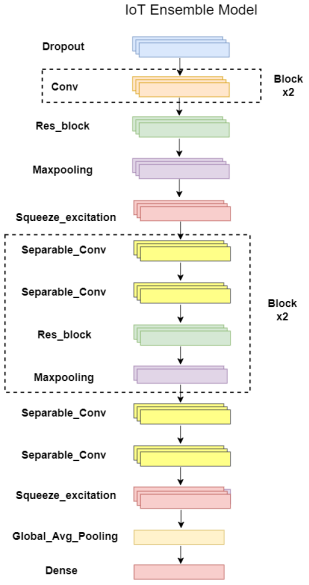
Next, we present the model architectures used for ensemble learning method (**cf. *Chapter 2.2***), corresponding to all five datasets. For the AWF dataset, we use the model proposed in [69], which is illustrated in Figure 6.5a. Since the DF dataset is also a website fingerprinting dataset much similar to AWF we used the same architecture with some hyperparameter changes. For DC, SETA and IoT, the complexity of the model is reduced by reducing the number of repetitive blocks. Figure 6.5b shows the ensemble model architecture for the DC dataset, and SETA uses a similar model with few hyperparameter changes. The ensemble models IoT is shown in Figure 6.5c.



(A) AWF ensemble model



(B) DC ensemble model



(C) IoT ensemble model

FIGURE 6.5: Ensemble model architectures

6.2 Framework for Robust Open Set Traffic Fingerprinting

Our proposed framework for robust open set classification targeted towards encrypted traffic fingerprinting in resource-restricted network devices is based on three key ideas.

1. *Regularized models*
2. *Model Quantization*
3. *k-Logit Neighbor Distance for open set classification*

6.2.1 Regularized model

All open set classification methods we explore in our work use underlying closed set classifiers (a deep learning model). We hypothesize that the performance of an open set classifier depends on how well the underlying deep learning model can identify class boundaries for the known classes. For instance, if the underlying closed set classifier identifies a sub-optimal class boundary or is overfitted to a given dataset causing class boundaries to even accommodate for noise and irregularities, it would cause the subsequent open set classifier to identify even open set samples as samples from a known class. We illustrate such a scenario in Figure 6.6.

Assume two closed set classifiers trained on two classes with *Feature 1* and *Feature 2* referring to the features learned by the classifier (logit layer). Figure 6.6 shows the distribution of Class A in this logit space. Although both Classifier 1 and 2 each have learned a boundary that encompasses all samples from Class A and would give 100% accuracy for Class A in the closed set setting, we see that Classifier 1 has learned a sub-optimal boundary that covers a region that includes only noise samples and no actual samples from Class A. Hence, when used for open set classification, Classifier 1 will label the open set samples that are within the sub-optimal boundary but away from the actual Class A samples as Class A. In contrast, Classifier 2 has learned an optimal boundary that covers only the correct samples from Class A and when used for open set classification can correctly reject all open set samples outside the boundary of Class A. Due to the simplicity of network traffic data that allows a simple model to be easily trained to achieve high accuracies, if proper hyperparameter tuning is not done, there is a likelihood that the resulting model simply learns sub-optimal class boundaries good enough to improve accuracy. For instance, in [16], the authors directly use the CNN model architecture proposed in [17] for traffic captured at the network level, on their traffic captured at data-link layer (WiFi) and still achieve high training and testing accuracies. As we show later, although the DC model performs well, it does not identify optimal class boundaries and therefore tuning the model further improves open set results.

Another possible source of such over-fitting is the number of closed set classes. Out of the five datasets we explore, DC, SETA and IoT did not have a separate open set and therefore we had to split the original dataset to create an open set. By doing so, we

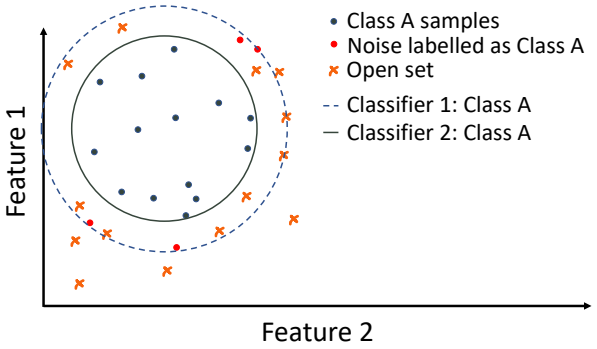


FIGURE 6.6: Optimal class boundary

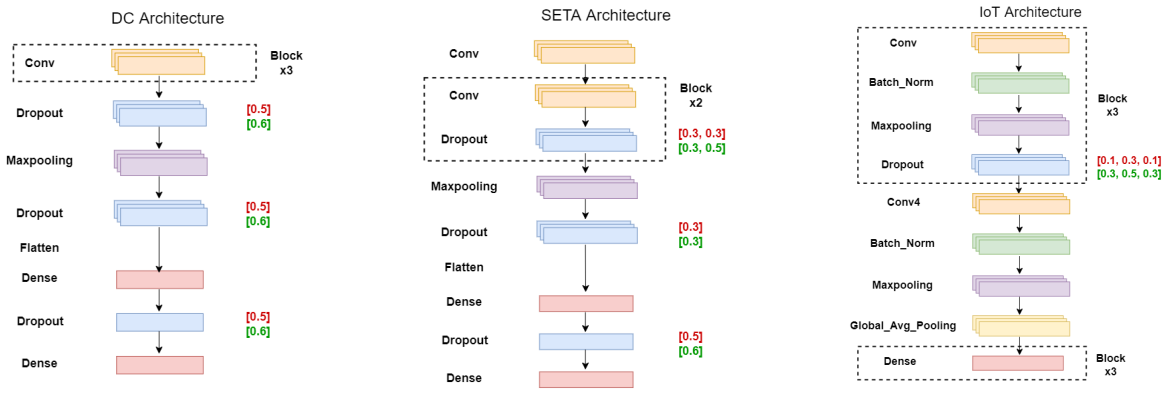
reduce the total number of classes (closed set) the original model architecture was tuned for, and using the exact same model architecture as in the original work on the smaller dataset would result in overfitting.

Accordingly, we hypothesize that if a more robust model (i.e more generalized model that identifies optimal class boundaries without overfitting) is used as the underlying classifier, it will improve the results of any open set classification method. To ensure that the underlying closed set classifier has learned better class boundaries and is not overfitted to a particular dataset, we propose to properly regularize a closed set classifier before using them for open set classification. We use DC, SETA, and IoT datasets to test our hypothesis and regularized the baseline models (models from corresponding original work as described in Chapter 6.1.4) by increasing the dropout rates. Then, we compare the open set classification results for using the baseline model vs the regularized model. We note that the original models used with AWF and DF datasets have undergone thorough hyperparameter tuning to ensure optimal performance and the original datasets are used without splitting in our work as they already contain open sets. Therefore, we do not use those datasets in this experiment.

Figure 6.7 illustrates the regularized models for DC, SETA and IoT datasets where we have denoted the dropout rates to the right of each dropout layer, with the value in red referring to the baseline model parameter and the value in green referring to the value in the regularized model.

6.2.2 Quantization

The second key component in our framework is model quantization. Quantization in general refers to mapping continuous values to a smaller set of discrete finite values.



(A) DC model (B) SETA model (C) IoT model

FIGURE 6.7: Model architectures

Usual neural network weights are real values (continuous infinite and represented as 32-bit or 64-bit floating point numbers) and quantization of neural networks maps these model parameters to 8-bit integer values within the range (-128 to 127). As a result, the model footprint in terms of storage and memory is decreased, and the inference becomes faster due to integer computations, making such models ideal to be deployed in in-network computing devices.

Since there is some information loss when the real-valued neural network weights are converted to integers, model quantization can cause a drop in performance. In our application, in addition to the drop in closed set performance, we also need to consider the effect of open set performance and ensure that the performance drop due to quantization is not significant. While there are many options for model quantization (e.g., quantization-aware training, post-training dynamic range quantization), here we used post-training integer quantization techniques provided by *Tensorflow*[1] to map all values from floating point numbers to `int8` format reducing the model size and perform open set classification using the quantized model as the underlying classification method.

6.2.3 k-Logit Neighbor Distance-based Open Set Classification

The third element of our framework is a novel distance-based open set classification method named k-Logit Neighbor Distance (k-LND) Method with three variants. It is built on the intuition that the output of the logit layer (the layer before softmax

[1] `https://www.tensorflow.org/lite/performance/post_training_integer_quant`

activation) of a deep neural network represents how classes are related to each other as opposed to being independent per-class score estimates.

More specifically, if N is the number of closed set classes, the output of the logit layer is a vector of length N that represents an N dimensional space where samples from the same class would be clustered together around its class center. Here, the class center is the average over the logit layer outputs of correctly classified training samples from the corresponding class, referred to as the Mean Activation Vector (MAV). Hence, if a sample is from the closed set, we expect it to be close to the MAV of its predicted class while being far away from the MAVs of k neighbor classes of the predicted class. In contrast, if a sample is from the open set, we expect it to be distant from the MAV of its predicted class and be relatively closer to the MAVs of k neighbor classes of the predicted class. Based on this intuition, k-LND methods use the distance between a new sample and the MAVs of known classes to identify open set samples.

We further explain this idea using an example in Figure 6.8. Consider a two-class closed set classifier. The output from the logit layer of the closed set classifier would be a vector of length two, with *logit 1* and *logit 2*. Figure 6.8 demonstrates the space spanned by *logit 1* and *logit 2* where the reddish dotted line shows the decision boundary learned by the classifier and C_1 and C_2 represent the MAVs of *Class 1* and *Class 2* respectively. The circle around each MAV represents the cluster around it where a majority of samples ($> 90\%$) from its class fall into. If we define r_i as the radius of the cluster around C_i, k-LND$_1$ (the first k-LND variant) assumes that if a sample is predicted as class i by the closed set classifier and the distance between that sample and C_i is greater than r_i, that sample is from the open set. For instance, consider that point P and point A in Figure 6.8 are both classified as class C_1 by the classifier. If d_{XC_i} defines the distance between a point X and C_i, we see that d_{PC_1} is less than r_1 while d_{AC_1} is greater than r_1. Hence, k-LND$_1$ will label sample P as *Class 1* and sample A as an open set sample. Similarly if both samples Q and B are classified by the closed set classifier as *Class 2*, k-LND$_1$ will label sample Q as *Class 2* since d_{QC_2} is less than r_2 and reject sample B as d_{BC_2} is greater than r_2.

k-LND$_1$ only considers the distance to the MAV of the predicted class to identify open set samples. k-LND$_2$ and k-LND$_3$ improve on the intuition of k-LND$_1$ such that they additionally assume that a sample from a known class would be distant from the MAVs

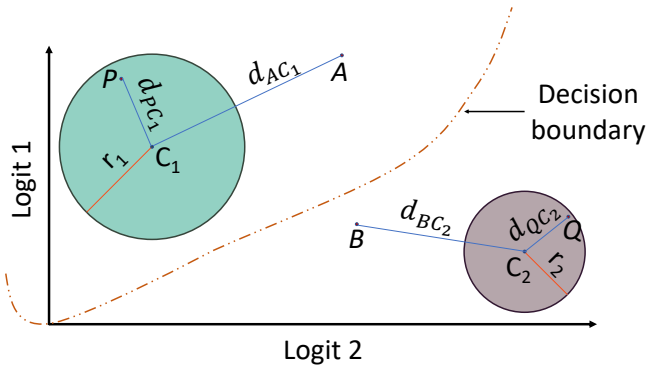


FIGURE 6.8: k-LND method

of the neighbors of its class, in addition to being closer to its own MAV. Accordingly, the three variants of k-LND differ in the way they calculate the distance between a sample and MAVs of closed set classes. In Equations 6.1, 6.2, and 6.3 we show how distances are calculated in k-LND$_1$, k-LND$_2$, and k-LND$_3$ respectively. Here, d_A denotes the Euclidean distance between the sample and the MAV of class A, p denotes the class predicted for the sample by the closed set classifier and k denotes the number of neighbor closed set classes considered. The only difference between k-LND$_2$ and k-LND$_3$ is that they use different methods to incorporate the distances to neighboring MAVs into the final distance metric so that to get a lower distance value, a sample needs to be closer to its own MAV and far away from the MAVs of its neighboring classes.

It should be noted that for datasets where the number of closed set classes is small, k is equal to the number of closed set classes and otherwise k will be less than the total number of closed set classes and would be considered as a hyperparameter of the method. The reason for this decision is that if the logit layer output is of longer length, euclidean length calculations become less effective [123] and the computation times also increase.

$$D_1 = d_p \tag{6.1}$$

$$D_2 = \sum_{i=1}^{k} (\ d_i - d_p) \quad ; i \neq p \tag{6.2}$$

$$D_3 = \frac{d_p}{\sum_{i=1}^{k} d_i} \ ; i \neq p \tag{6.3}$$

In Algorithm 6, we describe the common procedure to follow for all three methods prior to inference to calculate the Mean Class Vectors and corresponding threshold for each closed set class.

Algorithm 6: Before Inference for k-LND

Require: Closed set classifier without Softmax: θ, Number of closed set samples N
Input: Set $[X_i^{setA}, Y_i^{setA}]$, with $setA$ as subset A of dataset
Define: $X_{c_i}^{setA}$: the set of correctly classified samples of class c_i from set A by closed set classifier
Define: $n(X)$: no. of samples in X
Define: $sort(List_A)$: $List_A$ sorted in ascending order

```
for c_i = 1.....N do
    Calculate MAV_{c_i} = Mean(θ(X_{c_i}^{train}))
    distance_{c_i}=[]
    for j = 1.....n(X_{c_i}^{val}) do
        distance_{c_i}.append(D_K(X_{c_ij}^{val}))
    threshold_{c_i} = 90^{th}percentile(distance_{c_i})
return MAV^{c_i} and threshold_{c_i}
```

At inference time, a sample will first be fed to the closed set classifier to get its predicted class and logit layer output. Next, depending on which method is used, the distance values (D_1 or D_2 or D_3) are calculated using its predicted class and MAVs calculated in Algorithm 6. Finally, the calculated distance will be compared with the threshold value of the predicted class from $threshold_{c_i}$, and if the value is less than the threshold value, the predicted label will be accepted and otherwise, it will be rejected as an open set sample.

6.3 Results

Next, we explore the efficiency of the proposed framework. First, we introduce the evaluation metrics used and then analyze the open set performance of the framework when each one of its three key components is added. First, we show how a well-regularized model gives better open set accuracy and then use the regularized method to evaluate the performance of four existing open set methods against the three variants of the novel open set method proposed. Finally, we evaluate the performance of all seven open set methods with a quantized classifier to show how the proposed framework that combines a well-regularized model with its model weights quantized, with the proposed k-LND method, gives the best open set performance.

6.3.1 Evaluation metrics

For performance evaluation, we use *closed set accuracy* which represents the percentage of closed set samples correctly classified into relevant classes, and *open set accuracy* which denotes the percentage open set samples correctly identified. Since we need a single metric that can be used to identify the better-performing model, we also calculate the F1_Score. Since the open set is larger compared to a closed set class which makes the dataset imbalanced, we use the *Micro F1* score which is more suited to handling class imbalance. When calculating *Micro F1*, we consider only the correctly classified closed set samples as True Positives (TP) since the classifier is trained only on closed set data. For True Negatives (TN) and False Positives (FP), open set is considered as another class because samples misclassified from or to the open set class reflect the performance of the classifier. Accordingly, *Micro F1* score is calculated as the given in Equation 6.6 using precision and recall calculated according to Equation 6.4 and Equation 6.5 respectively, where N is the number of known classes. From here onward, *Micro F1 score* is referred to as F_Score. and when comparing two open est classifiers, we consider the classifier with the highest F_Score as the better classifier.

$$Precision_{Micro} = \frac{\sum_{n=1}^{N} TP_i}{\sum_{n=1}^{N} TP_i + FP_i} \quad (6.4) \qquad Recall_{Micro} = \frac{\sum_{n=1}^{N} TP_i}{\sum_{n=1}^{N} TP_i + FN_i} \quad (6.5)$$

$$F\_Score_{Micro} = 2 \times \frac{Precision_{Micro} \times Recall_{Micro}}{Precision_{Micro} + Recall_{Micro}} \quad (6.6)$$

6.3.2 Regularized Models

In Chapter 6.2.1 we explained that for a better open set classification the closed set classifier needs to be well regularized. This is majorly required if the closed set classifier is trained with a subset of the dataset used by the initial work to propose the original model. In such settings, the original model requires to be regularized. To show the effect of regularization, we trained a baseline model (taken from the original work) and a regularized model with high dropout rates for each dataset. Here, note that since AWF and DF datasets have separate open sets, we don't have to consider data splits, and hence the models proposed in the original work are considered the optimized models.

First, we compare the performance of each closed set classifier where we do not consider the open set and report the results in Table 6.2. According to Table 6.2, we see that for any of the three datasets, the performance of the two models is almost the same in the closed set setting.

TABLE 6.2: Closed set classifier performance

Model	**DC**	**SETA**	**IoT**
Baseline	99.08±0.88	98.17±1.70	97.49±0.48
Regularized	99.82±0.85	98.87±1.77	97.33±0.51

Next, we evaluate the performance of the two models in the open set setting. Figure 6.9 shows the percentage increase in F_score due to regularized model and in Table 6.3 we report the closed set and open set accuracy values for open set classification when using the two models. Observing Figure 6.9, we see that across all three datasets and four open set methods, the regularized model achieves a higher F_score compared to the baseline model. For the DC dataset, the highest increase in F_score of 20.69% is observed for the ensemble method while the lowest increase of 2.99% is seen for the OpenMax method. Similarly, for the SETA dataset, the highest increase in F_score of 35.48% corresponds to the ensemble method while the lowest increase of 15.38% is observed with OpenMax. For the IoT dataset, the maximum improvement of F_score of 8.33% is observed for OpenMax while the lowest increase of 3.23% is for the ensemble learning.

According to Table 6.3, in most cases, the regularized model increases the open set accuracy with a less than 1% decrease in the closed set accuracy. For DC dataset, increasing the dropout rates improved the open set accuracy by a maximum of 21.24% (background class) and a minimum of 4.65% (ensemble). However, OpenMax for DC dataset deviates from the above trend where the closed set increases by 5.19% while the open set decreases by 1.14% for the regularized model. Even then, the F1_score of OpenMax for the regularized model is still higher. With SETA dataset, the open set accuracy increases by a minimum of 4.02% (OpenMax) and a maximum of 38.36% (background class) with less than a 1% drop in the closed set when the underlying model is regularized. Similarly for IoT dataset, for a less than 1% drop in the closed set, the open set increases by a maximum of 32.7% (background) and a minimum of 4.63% (ensemble) for the regularized model compared to the baseline model. Based on the results across all datasets and methods, we conclude that our hypothesis that

TABLE 6.3: Effect of regularized model

Dataset	Model	Softmax Thresh.		Background class		OpenMax		Ensemble Learning	
		Closed Acc	**Open Acc**	**Closed Acc**	**Open Acc**	**Closed Acc**	**Open Acc**	**Closed Acc**	**Open Acc**
DC	Base.	90.85±1.9	87.7±3.8	98.70±1.0	36.87±15.9	89.22±2.7	92.16±2.7	84.80±4.0	63.88±22.2
	Reg.	**89.41±2.1**	**89.4±4.1**	**98.69±0.8**	**44.7±11.0**	**93.85±2.1**	**91.11±5.5**	**84.25±4.1**	**66.85±13.3**
SETA	Base.	86.25±4.6	71.59±7.2	96.45±2.4	33.71±11.4	83.75±5.4	79.51±6.1	79.51±1.6	69.16±17.5
	Reg.	**85.41±3.6**	**79.21±9.3**	**95.62±2.7**	**46.64±13.0**	**83.33±4.0**	**82.71±9.8**	**78.8±2.4**	**77.05±19.9**
IOT	Base.	87.55±0.7	59.23±2.8	96.91±0.4	26.21±8.3	87.28±0.7	55.22±6.8	84.78±2.9	68.95±7.3
	Reg.	**87.04±0.6**	**62.35±4.6**	**96.21±0.5**	**34.78±8.6**	**86.86±0.5**	**58.92±8.3**	**84.16±1.5**	**72.14±9.6**

Note that 'Base.' and 'Reg.' refer to the baseline and regularized models respectively.

suggests regularizing the underlying deep learning model improves the results of open set classification is accurate.

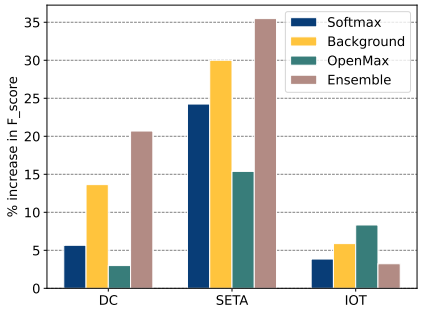


FIGURE 6.9: Effect of regularized model

6.3.3 k-Logit Neighbor Distance Method

We next present the results of evaluating the performance of the three novel open set methods (k-LND$_1$, k-LND$_2$, k-LND$_3$) we propose in Chapter 6.2.3 against four existing methods and report the results in Tables 6.4 and 6.5. Here, DC, SETA and IoT datasets use the regularized method trained before. Figure 6.10 summarizes the F_score values of all seven methods across all five datasets. According to F_score values in Figure 6.10, we see that k-LND$_3$ outperforms all other methods for AWF, SETA, and IoT datasets while background class and OpenMax give the best results for DF and DC datasets respectively.

Furthermore, we observe that k-LND$_1$, k-LND$_2$, and k-LND$_3$ methods shows consistent results across all the datasets. k-LND$_1$ and k-LND$_2$ maintains $> 85\%$ closed set and $> 65\%$ open set accuracy while k-LND$_3$ performs the best and maintains $> 90\%$ closed set and $> 70\%$ open set accuracy regardless of the dataset. In contrast, the performance of other methods fluctuates between datasets. For example, softmax thresholding

TABLE 6.4: Open set method performance - Existing methods

Dataset	Softmax Thresh.		Background class		OpenMax		Enseble Learning	
	Closed Acc	Open Acc	Closed Acc	Open Acc	Closed Acc	Open Acc	Closed Acc	Open Acc
AWF	87.35	88.16	81.60	25.11	87.32	87.93	83.30	85.90
DF	90.69	84.66	**95.20**	**97.40**	88.56	83.99	87.80	69.90
DC	89.41±2.1	89.40±4.1	98.69±0.8	44.70±11.0	**92.15±2.5**	**90.87±5.6**	84.25±4.1	66.85±13.3
SETA	85.41±3.6	79.21±9.3	95.62±2.7	46.64±13.0	83.33±4.0	82.71±9.8	78.80±2.4	77.05±19.9
IOT	87.04±0.6	62.35±4.6	96.21±0.5	34.78±8.1	86.86±0.5	58.92±8.3	84.16±1.5	72.14±9.6

TABLE 6.5: Open set method performance - kLND methods

Dataset	k-LND$_1$		k-LND$_2$		k-LND$_3$	
	Closed Acc	Open Acc	Closed Acc	Open Acc	Closed Acc	Open Acc
AWF	89.37	85.43	89.88	88.12	**97.98**	**89.23**
DF	88.45	83.99	88.29	88.04	97.84	87.21
DC	91.63±1.7	87.78±6.9	94.24±1.4	86.26±7.1	94.51±1.8	86.92±7.3
SETA	85.41±1.6	84.69±9.8	85.21±1.1	85.16±6.7	**95.42±1.7**	**87.84±9.1**
IOT	85.62±0.6	65.92±5.1	85.49±0.9	76.19±2.4	**97.33±0.5**	**74.47±3.5**

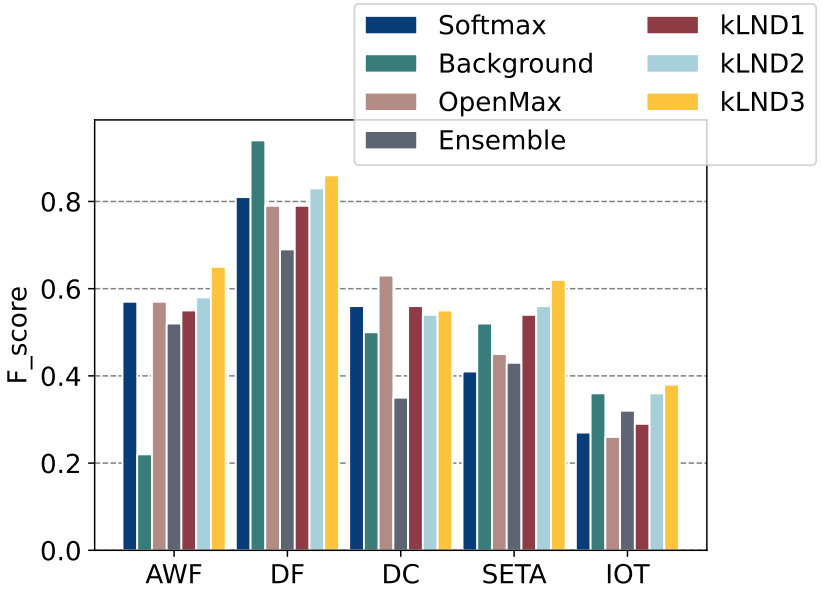


FIGURE 6.10: Default model F_scores

and OpenMax both have relatively lower open set accuracy for IoT compared to other datasets. While the background class method performs very well on DF dataset with $> 95\%$ in both closed and open sets, it performs very poorly in the open set of all other datasets.

In k-LND$_2$ and k-LND$_3$ we calculate the single parameter considering distances to all the class centers (MAV) which embeds the complete information about the output vector placement in the logit space. In other words, they calculate a comparative value from the penultimate layer output and use a threshold to do open set classification. We attribute

the consistently better behavior of novel methods to the fact that in novel methods, the open set samples are identified by being compared with all closed set classes in the logit space as opposed to just using a threshold for a single element (maximum value) or comparing with just the predicted class in the logit space.

To summarize, all three variants of k-LND method perform consistently across all datasets with $> 85\%$ closed set and $> 65\%$ open set accuracy. Moreover, as we show later in Chapter 6.3.4, k-LND methods outperform all other existing methods when model weights are quantized. Additionally, it should be noted that all three variants of k-LND are lightweight compared to the background class that require open set samples for training and OpenMax which needs additional EVT modeling. Hence we highlight that k-LND performs well consistently across all datasets while consuming the least resources.

6.3.4 Quantization

Next, we investigate the effect of quantizing the underlying deep learning model on open set classification. In Table 6.6 and 6.7, we present the results of using quantized model (quantize weights of models used in Chapter 6.3.3)for open set classification. Figure 6.11 summarizes the F_score values of all seven methods across all five datasets which shows that when using a quantized model, the three k-LND methods outperform all other methods across all datasets.

When using the quantized models for AWF dataset, only softmax thresholding and the k-LND methods achieve $> 85\%$ in both closed and open set accuracies. Out of those four methods, k-LND$_1$ method records the best performance with 97.98% and 84.02% as closed and open set accuracy respectively. With the DF dataset, only the novel three methods obtain $> 85\%$ in both closed set and open set accuracy with k-LND$_3$ method achieving the best performance with closed and open set accuracies of 97.9% and 87.2% respectively, when using the quantized models. For DC dataset when using the quantized models, softmax thresholding and all three novel methods obtain $> 85\%$ in both closed set and open set accuracy. Softmax thresholding performs the best with an F_score of 0.75 closely followed by k-LND$_3$. After quantizing the models for SETA dataset, k-LND$_3$ performs the best with 89.6% closed set and 70.7% open set accuracy while none of the other methods achieve $> 85\%$ closed set accuracy with $> 70\%$ open

TABLE 6.6: Quantized model performance - Existing methods

	Softmax Thresh.		**Background class**		**OpenMax**		**Enseble Learning**	
Dataset	**Closed Acc**	**Open Acc**	**Closed Acc**	**Open Acc**	**Closed Acc**	**Open Acc**	**Closed Acc**	**Open Acc**
AWF	89.10	86.90	80.20	73.50	81.90	73.60	85.2	31.8
DF	94.40	78.22	92.50	67.45	94.88	61.75	90.20	58.90
DC	93.72±1.9	90.89±5.0	96.20±2.1	44.10±15.3	89.80±4.7	89.84±6.4	77.22±8.8	59.54±9.4
SETA	81.54±11.7	52.17±20.2	78.75±8.9	34.88±12.2	51.55±14.7	77.62±16.3	69.30±14.26	68.48±21.19
IOT	89.18±5.6	46.70±11.2	91.31±3.4	42.50±8.9	78.09±6.2	18.87±9.8	72.53±5.9	21.30±10.1

TABLE 6.7: Quantized model performance - kLND methods

	k-LND$_1$		**k-LND$_2$**		**k-LND$_3$**	
Dataset	**Closed Acc**	**Open Acc**	**Closed Acc**	**Open Acc**	**Closed Acc**	**Open Acc**
AWF	89.08	86.89	89.88	88.22	**97.98**	**84.02**
DF	87.72	84.20	87.92	87.80	**97.98**	**87.22**
DC	**87.94±3.9**	**93.71±2.4**	93.18±0.9	88.22±5.9	94.21±1.2	87.92±7.3
SETA	70.62±9.1	68.29±14.4	73.33±8.0	73.77±9.1	**89.58±7.1**	**70.74±11.5**
IOT	83.59±1.2	68.86±6.9	83.56±0.9	77.22±1.9	**95.98±0.5**	**76.17±4.8**

set accuracy. Similarly for IoT dataset, k-LND$_3$ performs the best with 95.9% closed set and 76.2% open set accuracy while none of the other methods achieve $> 85\%$ closed set accuracy with $> 70\%$ open set accuracy. If we consider the overall result, we observe that k-LND methods perform best on quantized models achieving the highest F_score for all datasets. More specifically, k-LND$_2$ performs best for DC while for all other datasets, k-LND$_3$ performs the best.

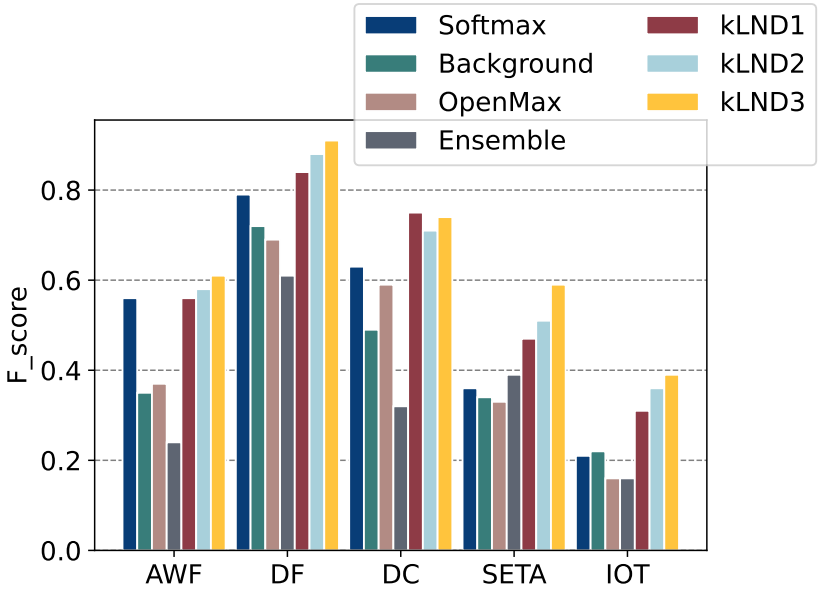


FIGURE 6.11: Quantized model F_scores

Next, we report the model sizes before and after quantization in Table 6.8 where we observe that the quantization of model weights reduces the storage requirement of a

model by a minimum of 60% (SETA) and a maximum of 75.01% (IoT), which shows how quantization helps reduce the model footprint in terms of storage. Additionally, we also compare the model accuracies between the original model before quantization and the quantized model in the closed set setting in Table 6.8. Accordingly, we see that in the closed set setting, quantizing the model has no noticeable effect on the model's accuracy.

TABLE 6.8: Quantized models comparison

	Model size		**closed set accuracy**	
Dataset	**Original model**	**Quantized model**	**Original model**	**Quantized model**
AWF	8.51 Mb	2.14 Mb	98.09	97.32
DF	8.30 Mb	2.08 Mb	97.86	97.02
DC	755 Kb	189 Kb	99.82 ± 0.9	98.69 ± 0.6
SETA	3.41 Mb	1.36 Mb	98.87 ± 1.8	98.12 ± 1.2
IoT	1.02 Mb	261 Kb	97.33 ± 0.5	96.87 ± 0.5

6.3.5 Result Analysis

In Chapter 6.3.4, we showed how the k-LND methods work best when quantizing the underlying deep learning model and we next explore possible reasons for this observation.

As discussed in Chapter 2.2, the background class method treats open set as just another class and uses a subset of open set samples as a single class during training. Because of this, when quantizing the background class, the samples from the open set are also considered in the discrete mapping process. Since the training set from open set consists of a relatively larger number of samples coming from a large number of different classes, the range of continuous values for elements of input samples increases making the discrete mapping process harder for the quantizer as it now has to map a larger range of continuous values to a fixed smaller range. (This mapping procedure is further discussed in [124] and [125].). We attribute the degradation of the performance of the background class method with quantizing, to the large error caused by the mapping function as discussed above.

When comparing softmax thresholding, OpenMax and the three novel methods, the error they encounter at the start from the model output at the logit layer is the same. Softmax thresholding uses this output and performs softmax activation which maps the output vector to another space where the initial error can further propagate. Similarly,

the OpenMax method maps the penultimate layer output to another space first using a Weibull distribution before generating the final probability vector which would cause the error to increase further. Ensemble learning methods that aggregate the result from multiple models in a method similar to softmax thresholding can be expected to face the drawback as softmax thresholding. In contrast, in all three k-LND methods, a single variable is calculated based on euclidean distances between penultimate layer outputs. Additionally, k-LND$_2$ and k-LND$_3$ methods consider the relative distance between a sample and its predicted class center vs. centers of multiple other classes, which can have a negation effect on the initial error resulting in a lower effect on the final results.

We further demonstrate this concept with SETA dataset by calculating percentage between *error_before* and *error_after* as calculated by Equation 6.7 and Equation 6.7 respectively, for each method.

$$error\_before = d(\Theta_{QP}(X), \Theta_{OP}(X)) \tag{6.7}$$

$$error\_after = d(\Theta_{Q\overline{P}}(X), \Theta_{O\overline{P}}(X)) \tag{6.8}$$

Here assume a sample X, deep learning model Θ_O and quantized version of the model Θ_Q, and d is euclidean distance. Also, P and $\overline{P}$ refer to the prenultimate layer output of the final probability vector from the open set classifier respectively. The results are illustrated in Figure 6.12.

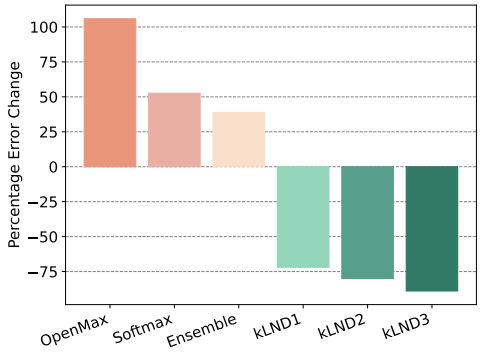


FIGURE 6.12: Percentage of change in error

According to Figure 6.12, the change of error in OpenMax is 105% which shows that the initial error has increased by 105.89% due to Weibull conversion resulting in the high F1_score drop in Table 6.4 for SETA after quantization. Similarly, softmax thresholding

also has a positive change of error of 52.6% which can be seen as the reason for the relatively high F1_score drop after model quantization.

In contrast, the change of error for k-LND_1, k-LND_2 and k-LND_3 are less than zero which shows that it reduces the error in the mapping which results in lesser Micro F1 drops. Accordingly, we can conclude that the three novel open set classification methods we propose are more compatible with model quantization than other existing methods.

6.4 Discussion and Concluding Remarks

In this chapter, we first hypothesized that using a well-regularized classifier as the underlying deep learning model improves the performance of open set classifiers. We used three publicly available datasets with model architectures proposed in the original work and showed that when the original model has not undergone a thorough hyperparameter search or the number of classes in the dataset used for training the original model is changed, regularizing the model architecture improves the performance of open set classifiers that use such models. Next, we showed that quantizing the model weights does not result in a significant drop in the closed set accuracy. Then, we proposed a novel *k-logit neighbor distance* based open set method with three variants and compared their performance with four existing methods using five publicly available encrypted traffic fingerprinting datasets. We also showed how the two most commonly used open set traffic fingerprinting methods; background class and softmax thresholding as well as OpenMax and ensemble learning do not work well across all the datasets. While they performed really well in some datasets they also performed really poorly in other datasets. In contrast, our proposed methods consistently performed well across all datasets with $> 85\%$ closed set accuracy and $> 65\%$ open set accuracy. Finally, we showed when using a quantized model as the underlying closed set classifier for open set classification, our proposed method outperformed all other methods across all datasets maintaining $> 70\%$ closed set accuracy and $> 68\%$ open set accuracy. Overall, our results showed that the framework we proposed supports quantization and the combined contribution from all three key components outperforms all other methods, across all datasets for open set classification proving how the framework is well suited for traffic fingerprinting tasks that need to be carried out in resource-constrained in-network devices such as P4 switches, smart NICs or FPGAs.

Next, we discuss limitations and possible extensions to our work.

Recent open set methods: Open set classification in general is currently a popular research area that is still in its early stages with researchers across multiple domains exploring and proposing novel methods. While we have explored the applicability of a few existing open set classification methods to encrypted network traffic, more recently proposed methods such as *Class Anchor Clustering* [34], *Joint Confidence Method* [74] and *Conditional Gaussian Distribution Learning for Open Set* [126] need further investigation with respect to their applicability to network traffic.

Synthesized open set: The performance of most open set classification methods still has room for improvement. It can be assumed that exposing a model to a good enough representation of open set could in turn improve its ability to efficiently identify open set samples. In our work, the only method that uses open set samples is the background class method which could suffer from lack of training data that comprehensively represents the open set. As a solution, future work can investigate data synthesis with recent generative methods such as Diffusion models [127] to generate samples that better represent the open set that can be used during training.

Other quantization methods: In this work, we only used *post-training integer quantization* as it has very low overheads. However, post-training quantization is considered to have the highest impact on accuracy compared to other quantization modes and hence it would be interesting to see how other quantization modes such as *quantization-aware training*, and *post-training dynamic range quantization* work with traffic fingerprinting models such that where resources are not limited these quantization methods can be utilized.

Chapter 7

Conclusion and Future Work

The primary focus of this thesis was to study the privacy implications of end-to-end encrypted traffic. E2EE is widely considered the gold standard of secure communications, and hence the world is moving fast towards encrypting all Internet communications with the objective of guaranteeing the confidentiality of data in transit and user privacy. As discussed in Chapter 1, side-channel information leaks of encrypted traffic flows allow a passive eavesdropper to infer valuable information about their encrypted content without decryption. While such attacks can be helpful to network administrators and government intelligence services, in the hands of a malicious party, such attacks present a significant threat to the privacy of Internet users. In this work, thesis we first propose new traffic analysis attacks and improve their practical applicability, and then propose defenses against such attacks that are more practical and efficient compared to previous work.

First, we explored the feasibility of performing traffic analysis attacks on encrypted traffic and proposed and evaluated an *inline traffic analysis attack* as described in Chapter 3. Then we dissect deep learning models used for traffic analysis attacks with the objective of understanding their behavior and the decision-making process, which can be leveraged to design better attacks as well as more efficient defenses as described in Chapter 4. Next, we use the insights we obtain from understanding the behavior of traffic analysis models to propose two efficient defenses against traffic analysis attacks named *FRONT-U* and *STOMA* as discussed in Chapter 5. Finally, we explore the open set scenario for encrypted traffic classification, which is critical when using such attacks in real-world

applications. As described in Chapter 6, we proposed a robust framework for open set classification targeting encrypted traffic, which requires relatively fewer resources and is more suited for resource-constrained in-network computing devices.

7.1 Summary and Conclusion

7.1.1 Traffic analysis attacks on DNS-over-HTTPS traffic

In Chapter 3, we observed that DoH traffic flows for websites have low entropies, leaving them vulnerable to traffic analysis attacks. First, we proposed an inline traffic analysis attack to identify websites using DoH traffic by leveraging a variable-length LSTM model. Our proposed method achieved over 96% accuracy by observing only the first 10 packets of a DoH flow. As the next step, we extended the attack to the open set setting with a novel open set classification method *JSI*, which achieves 75%-80% accuracy on both closed set and open set. Next, we explored a more challenging attack scenario where the attacker does not have the knowledge of the exact starting point of a specific DoH flow. We showed that even when a traffic trace is captured at an arbitrary starting point, a classifier can still identify them with over 70% accuracy. This suggests that even if the attacker is not aware of when exactly a victim starts loading a website and starts capturing traffic at some point during the DNS resolving process, the website visited by the victim can still be recognized with 70% accuracy.

Due to the vulnerability of DoH traffic to traffic analysis attacks, Internet Engineering Task Force introduced RFC8467 [53], which recommends specifications to pad DoH traffic as a defense. We evaluated our proposed attack against multiple padding scenarios and showed that the recommended defense is not adequate as the attack can still achieve ∼70% accuracy regardless of the defense.

7.1.2 Dissecting traffic fingerprinting CNNs

We observed that across multiple previous works, CNNs were shown to be the most effective deep-learning model for traffic analysis attacks. To further understand the inner workings of CNN-based traffic analysis attacks, we designed several experiments in Chapter 4. We used activation maps and selective occlusion to show that website

fingerprinting CNNs focus more on sections of a traffic trace that corresponds to transitions between uploads and downloads and give more weight to the initial part of a trace that contains a high concentration of HTTP GET messages and replies. Similarly for video fingerprinting CNNs, we used gradient ascent in addition to activation maps and selective occlusion to show that video fingerprinting CNNs focus more on the periodic sections of uploads and downloads of a trace that overlaps with periodic bursts in video streaming. These observations can serve as a foundation to build defenses against these attacks.

Next, we showed that similar to image classification CNNs, traffic fingerprinting CNNs also have transfer learning capabilities that allow such models to be easily fine-tuned to accommodate more classes.

Finally, we observed that CNNs are more resilient to random variations in traffic flows and bursts that occur due to varying network conditions, and show that this capability allows CNNs to outperform other deep learning models such as LSTMs in traffic analysis attacks. While this result explains the continuous success of CNNs over other model architectures for traffic analysis, it also suggests that just adding delays and padding randomly does not significantly affect a CNN-based classifier's ability to fingerprint a website, which is important to note when designing defenses.

7.1.3 Defenses against traffic fingerprinting

Based on the observations from Chapter 4, in Chapter 5 we propose defenses against traffic analysis attacks that can provide adequate privacy at reasonable data and timing costs, and compare their performance against state-of-the-art defenses.

We first proposed **FRONT-U**, a defense against website fingerprinting attacks where the main aim was to focus more on obfuscating the trace fronts. In Chapter 4, we observed that WF CNNs focus on transitions between uploads and downloads and we leverage this insight to propose that only the client side participates in the defense to reduce data overheads. We show that our defense can reduce the data overhead incurred by 50% and still provide similar privacy as the state-of-the-art defense.

We next proposed **STOMA**, a defense against video stream fingerprinting CNNs. Based on the observations from Chapter 4 that these attacks focus more on the periodic sections

of uploads and downloads of a trace, STOMA targets to obfuscate specific periodic burst patterns by averaging the traffic flow over a small window of time. We show that when combined with d*-privacy method, our defense reduces attack accuracy by 44.40% more at lower data overheads compared to using the d*-privacy method by itself. We also show that compared to other state-of-the-art defenses like FPA method, our defense requires minimal changes to the underlying DASH protocol for video streaming.

7.1.4 Open set classification for encrypted network traffic

Next, in Chapter 6, we proposed a framework for robust open set classification for encrypted traffic fingerprinting, which is more suitable for resource-constrained in-network computing devices compared to existing methods. The framework comprised three components, the first of which was using a well-regularized model as the underlying closed set classifier. Using three publicly available datasets, we showed that having a well-regularized underlying closed set classifier improves open set results irrespective of the specific open set classification method used, with a minimum increase of F_Score value of 3.23%. The second key component of our framework is model quantization. We showed that traffic fingerprinting models could be quantized without a significant drop in accuracy while reducing the memory footprint of classifiers by at least 60%. As the third key component, we proposed 'k-LND'; a novel open set classification method with three variants. Using five publicly available datasets, we showed that k-LND methods perform consistently across all datasets, always maintaining $> 85\%$ closed set and $> 65\%$ open set accuracy. Moreover, when used with quantized models, the k-LND methods always outperformed all other methods, always maintaining $> 85\%$ closed set and $> 65\%$ open set accuracy. Overall, our results showed that our framework supports quantization and the combined contribution from all three key components outperforms all other methods across all datasets for open set classification, proving how the framework is well suited for traffic fingerprinting tasks that need to be carried out in resource-constrained in-network devices such as P4 switches, smart NICs or FPGAs.

7.2 Future Work

The work presented in this thesis contributes to the development of multiple new research directions, and there are possible extensions for some of the presented works. We next discuss these research directions and the possible extensions for each of the main focus areas of the thesis.

7.2.1 Traffic analysis attacks and defenses

Traffic analysis and defenses have always been an arms race between attackers and defenders which would continue into the future. As such, designing novel defenses to counter novel attacks and vice versa needs to be a continuous process.

Traffic analysis attacks:
A key direction future work can focus on is to improve the practical applicability of traffic analysis attacks by relaxing the assumptions previous attacks were developed under. For instance, most current attacks make assumptions such as correctly labeled training data is available, the victim's system specifications are known in advance, traffic patterns of websites have not changed over time, and the devices running the attacks have adequate computing resources.
i) Victim environment-agnostic attacks: Most current attacks assume the attacker knows the specific browser/operating system combination of the victim. However, this may not reflect the real-world situation, and the traffic signatures for the same activity by the same user can be expected to change with the browser/operating system combination [57]. A naive approach that could be investigated further is to explore the effect of adding traffic captures for the same activity under different browser/operating system combinations to the training set. Moreover, user-specific configurations such as the use of ad-blockers or user-targeted advertising can also make the attack more challenging and require further investigation. ***ii) Handling concept drift:*** Website traffic patterns will change with time due to changes in content, layout, and in-page advertisement on a website, and this phenomenon is referred to as *concept drift* [15]. Hence, concept drift can make it difficult for a model to correctly identify a website visit captured at a much later point in time than when the dataset the model was trained

on was collected. Handling such concept drifts and changes to target classes over time is another challenge faced by traffic analysis attacks. Though some attacks proposed in this work and prior work were shown to be resilient against changes in traffic patterns over the period of a few weeks, over longer spans, such changes would have a significant impact on the performance of attack models. Future work can explore ideas of transfer learning [101] or online learning [102] to address concept drift. For instance, transfer learning can be used to extend/change the set of target websites by just fine-tuning the final fully connected layers with a few samples from each class in the new target list, which significantly reduces the training time required for re-training. Similarly, online learning could be leveraged to update a model with novel classes by training the model for a few more epochs with new classes or classes affected by concept drift only while preventing catastrophic forgetting (of previous classes).

iii) Handling lack of/ inaccurate labeled data: Another limitation of current traffic analysis models is that most of them assume traffic captures with accurate labeling are always available and mostly depend on supervised learning approaches. However, in practice, it is possible to encounter situations where labels for captured traffic are not available, or the accuracy of available labels can not be trusted. Therefore, future work needs to explore what semi-supervised or unsupervised learning methods are more suitable for encrypted traffic and leverage them to propose novel attack models.

iv) Deployability on resource-constrained devices: In practice, traffic analysis attacks need to be deployed on networking middle-boxes where computational resources and storage are restricted [121, 122]. Furthermore, the demand for such implementations is expected to grow with the increasing popularity of SDN and programmable switches such as P4 [128]. Future work needs to investigate the feasibility of transferring traffic analysis models to such devices and develop prototypes. For instance, concepts such as model weight quantization could be leveraged to minimize memory requirements in storing trained models on network devices. Moreover, further research is required on how to deploy such attacks and isolate target traffic in real time on commercial networks that handle very large amounts of data. At the same time, it would not be feasible to make predictions for every single traffic flow in a commercial network that handles traffic at Gbps scale and therefore, traffic flow sampling plays an important role in such scenarios. It would be an interesting research area to explore how traffic flow sampling algorithms such as *flow skampling* [129] can be used for real-time traffic analysis in commercial networks.

v) Active side-channel attacks: The main focus of this work was on passive traffic analysis attacks where the attacker simply observes the side-channel information without making any changes to the original traffic. However, recent research has discovered the possibility of active side-channel attacks against encrypted traffic, where the attacker takes active measures such as watermarking the encrypted traffic flows. For example, [130] demonstrated how a mobile subscriber can be identified and localized by sending unsolicited WhatsApp messages, and [131] showed that the Twitch stream watched by a victim could be identified by sending chat messages into a predefined set of Twitch streams. However, active side-channel attacks are still very nascent, and further studies are required to get a better understanding of which online services are vulnerable to such attacks and what types of information can be leaked through such attacks.

Defenses against traffic analysis:

With respect to defenses against traffic analysis, it should be noted that most defenses that have been proposed so far have not been implemented by real-world applications, and the main reason for this is their high overheads that affect the basic functionality and user experience. A key challenge in implementing defenses is that they need to be integrated at the protocol level with minimum changes to the existing Internet infrastructure. For instance, most video streaming applications are built on the DASH protocol, which requires maintaining the bursty nature of video streaming and any defense against video stream fingerprinting should ensure that they do not make significant changes to the bursty nature of video traffic.

Future work could explore other approaches to defenses, such as the use of *universal perturbations*. Previous work in computer vision has shown the existence of universal (image-agnostic) and small perturbation vectors that cause natural images to be misclassified with high probability across different deep neural networks [119]. If such input-agnostic perturbations exist for a traffic fingerprinting model, it can be pre-computed and used to defend network traffic against fingerprinting attacks in real time with fewer data overheads. A recent work [82] explored this idea to defend against website fingerprinting attacks over Tor. Future work can compare its performance, overheads and effectiveness against multiple deep learning and traditional machine learning-based attack models with other recent defenses like FRONT, to identify a more practically applicable defense. Another possible approach is to build a defense based on the idea of *one pixel attacks* [132] where a perturbation in a single pixel can force a misclassification

from an image classification model with minimal cost. Moreover, the above approaches can also be used to defend against other types of encrypted traffic as well.

7.2.2 Open set traffic analysis:

Open set classification in general, is a research area that is still in its early stages and researchers across multiple domains are showing a growing interest in it. While we have explored the applicability of a few existing open set classification methods to encrypted network traffic, more recently proposed methods such as [34, 74, 126] need further investigations.

Furthermore, the performance of most open set classification methods can still be improved. Previously, we discussed how methods that require open set samples for training would not always work because it is difficult to get a training set for the open set which represents the open set comprehensively. Future work can investigate data synthesis approaches to generate samples that represent the open set comprehensively so that the model can be exposed to a better representation of the open set during training. For example, in [133] we used *Conditional GANs (CGAN)* to generate samples similar to a pre-defined class to be used as a reference to altering a traffic trace. Future work could use the same idea and use CGANs to generate open set samples similar to closed set classes, which can be used to train a classifier to identify open set samples better. Additionally, future work could also leverage recent generative methods such as *Diffusion models* [127] for the same purpose.

Finally, *open world recognition* [104] is an improvement over open set classification where an attack model is expected to identify novel classes among samples rejected by open set classification, and updating the target class lists to incorporate such novel classes. More specifically, open world recognition can be seen as the combination of three key components, *1) Open set classification* which correctly classifies known classes while identifying samples from unknown classes, *2) Automated labeling* which identifies novel classes in unknown samples rejected by 1), and *3) Online learning* which can update the original classifier about the novel classes detected without affecting its performance on original known classes. While few prior works [134, 135] in other domains attempted open world recognition, it has not been explored with network traffic data even though

this setting can be beneficial for encrypted traffic classification to accommodate Internet traffic which is a fast-evolving eco-system. We have already discussed a few approaches that have the potential to improve the first component (open set classification). The next challenge for open world recognition lies with the second component (automated labeling), which has to discover novel classes from rejected open set samples without prior knowledge of the number of classes present. One possible approach would be using *Hierarchical Dirichlet Processes* [136], which has been used for similar purposes such as topic discovery in text data [137, 138]. In order to tackle the third component, future work can explore simple approaches like *transfer learning* as well as other methods such as [139] which used a novel loss function that combined traditional cross-entropy loss with *distillation loss* [140] and [141] which used a *dual memory system* inspired by mammalian brains.

Appendix A

Copyrighted Resources

We acknowledge that all images used in this work that do not belong to us were taken from the internet under the Creative Common Licenses. In the Table below, we specify the original sources of images used.

Figure	Sources
Figure 1.5	https://www.flickr.com/photos/berniedup/40116291883
	https://www.flickr.com/photos/antpitta/42962113324
	https://commons.wikimedia.org/wiki/File:Ostrich_male_RWD.jpg
	https://pixabay.com/photos/ostrich-bird-animal-head-nature-4612575/
	https://commons.wikimedia.org/wiki/File:Blue_Jay_%28210140531%29.jpeg
	https://commons.wikimedia.org/wiki/File:Blue_Jay_%28185317371%29.jpeg
	https://commons.wikimedia.org/wiki/File:Cyanocitta_cristata_blue_jay.jpg
	https://pxhere.com/en/photo/1550809
	https://commons.wikimedia.org/wiki/File:Australian_Magpie_%28Gymnorhina_tibicen%29.jpg
	https://www.rawpixel.com/search/magpie%20bird?page=1&sort=curated
	https://www.flickr.com/photos/yeliseev/230788934
	https://www.flickr.com/photos/mikeprince/21442498251
	https://www.publicdomainpictures.net/en/view-image.php?image=35440&picture=dog-sitting
	https://www.flickr.com/photos/stanbury/3930071960
	https://www.flickr.com/photos/horiavarlan/4272009397
	https://www.pexels.com/photo/white-cloud-in-blue-sky-on-sunny-day-4570006/
	https://pxhere.com/en/photo/1150893
	https://www.flickr.com/photos/iansand/31686643558
	https://www.rawpixel.com/search/tree?page=1&sort=curated
	https://www.publicdomainpictures.net/en/view-image.php?image=173260&picture=tree
Figure 2.2	https://www.flickr.com/photos/28656738@N02/5311930838
Figure 1.4a	https://www.rawpixel.com/image/5940985/free-public-domain-cc0-photo

Table A.1: Image sources

Bibliography

[1] Internet usage worldwide, July 2022. URL https://www.statista.com/topics/1145/internet-usage-worldwide/. publisher: https://www.statista.com/topics/1145/internet-usage-worldwide/#dossierKeyfigures.

[2] David Warburton. The 2021 tls telemetry report — f5 labs, 2021. URL https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report.

[3] Ryan Singel. Declassified nsa document reveals the secret history of tempest, Apr 2008. URL https://www.wired.com/2008/04/nsa-releases-se/.

[4] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[5] Dawn Song. Timing analysis of keystrokes and ssh timing attacks. In *Proc. of 10th USENIX Security Symposium, 2001*, 2001.

[6] //www.frogbikesau.com/blog/How-To-Wrap-a-Kids-Bike-for-Christmas?whence=.

[7] David Wagner, Bruce Schneier, et al. Analysis of the ssl 3.0 protocol. In *The Second USENIX Workshop on Electronic Commerce Proceedings*, volume 1, pages 29–40, 1996.

[8] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pages 103–114, 2011.

[9] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 143–157, 2014.

[10] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1187–1203, 2016.

[11] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 31–42, 2009.

[12] Charles V Wright, Lucas Ballard, Scott E Coull, Fabian Monrose, and Gerald M Masson. Uncovering spoken phrases in encrypted voice over ip conversations. *ACM Transactions on Information and System Security (TISSEC)*, 13(4):1–30, 2010.

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

[14] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *CCS*, 2018.

[15] Vera Rimmer, Davy Preuveneers, Marc Juárez, Tom van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. In *25th NDSS*, 2018.

[16] Ying Li, Yi Huang, Richard Xu, Suranga Seneviratne, et al. Deep content: Unveiling video streaming content from encrypted wifi traffic. In *17th IEEE NCA*, 2018.

[17] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the burst: Remote identification of encrypted video streams. In *26th USENIX Security*, pages 1357–1374, 2017.

[18] Chenggang Wang, Sean Kennedy, Haipeng Li, King Hudson, Gowtham Atluri, Xuetao Wei, Wenhai Sun, and Boyang Wang. Fingerprinting encrypted voice traffic on smart speakers with deep learning. In *Proceedings of the 13th ACM*

Conference on Security and Privacy in Wireless and Mobile Networks, WiSec '20, page 254–265. Association for Computing Machinery, 2020. ISBN 9781450380065.

[19] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *2010 IEEE Symposium on Security and Privacy*, pages 191–206. IEEE, 2010.

[20] Kevin P Dyer, Scott E Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *2012 IEEE Symposium on Security and Privacy*, pages 332–346. IEEE, 2012.

[21] Xiang Cai, Rishab Nithyanand, and Rob Johnson. CS-BuFLO: A congestion sensitive website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 121–130. ACM, 2014.

[22] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 227–238. ACM, 2014.

[23] Marc Juarez, Mohsen Imani, Mike Perry, Claudia Diaz, and Matthew Wright. Toward an efficient website fingerprinting defense. In *European Symposium on Research in Computer Security*, pages 27–46. Springer, 2016.

[24] Jiajun Gong and Tao Wang. Zero-delay lightweight defenses against website fingerprinting. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 717–734, 2020.

[25] Xiaokuan Zhang, Jihun Hamm, Michael K Reiter, and Yinqian Zhang. Statistical privacy for streaming traffic. In *Proceedings of the 26th ISOC Symposium on Network and Distributed System Security*, 2019.

[26] Alex J DeGrave, Joseph D Janizek, and Su-In Lee. Ai for radiographic covid-19 detection selects shortcuts over signal. *Nature Machine Intelligence*, 3(7):610–619, 2021.

[27] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep

networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[28] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[29] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

[30] Matthew D Zeiler and Rob Fergus. Visualizing and understanding Convolutional Networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.

[31] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep Recurrent Neural Networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649. IEEE, 2013.

[32] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems*, pages 577–585, 2015.

[33] Abhijit Bendale and Terrance E Boult. Towards open set deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1563–1572, 2016.

[34] Dimity Miller, Niko Sunderhauf, Michael Milford, and Feras Dayoub. Class anchor clustering: A loss for distance-based open set recognition. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3570–3578, 2021.

[35] Akshay Raj Dhamija, Manuel Günther, and Terrance Boult. Reducing network agnostophobia. *Advances in Neural Information Processing Systems*, 31, 2018.

[36] Heyning Cheng and Ron Avnur. Traffic analysis of ssl encrypted web browsing. *Project paper, University of Berkeley*, 1998.

[37] Andrew Hintz. Fingerprinting websites using traffic analysis. In *International workshop on privacy enhancing technologies*, pages 171–178. Springer, 2002.

[38] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 31–42. ACM, 2009.

[39] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 143–157, 2014.

[40] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. Website fingerprinting at internet scale. In *NDSS*, 2016.

[41] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 1187–1203, 2016.

[42] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 263–274. ACM, 2014.

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

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

[45] Kota Abe and Shigeki Goto. Fingerprinting attack on tor anonymity using deep learning. *Proceedings of the Asia-Pacific Advanced Network*, 42:15–20, 2016.

[46] Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, and Matthew Wright. Triplet fingerprinting: More practical and portable website fingerprinting with N-shot learning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1131–1148, 2019.

[47] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. Var-CNN: A Data-Efficient Website Fingerprinting Attack Based on Deep Learning. *Proceedings on Privacy Enhancing Technologies*, 4:292–310, 2019.

[48] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman. Specification for dns over transport layer security (tls). Technical report, 2016.

[49] Christian Huitema, Melinda Shore, Allison Mankin, Sara Dickinson, and Jana Iyengar. Specification of dns over dedicated quic connections. *Internet Engineering Task Force, Internet-Draft draft-huitema-quic-dnsoquic-05*, 2018.

[50] P. Hoffman and P. McManus. Dns queries over https (doh). Technical report, 2018.

[51] Haya Shulman. Pretty bad privacy: Pitfalls of dns encryption. In *Pro. of the 13th Workshop on Privacy in the Electronic Society*, 2014.

[52] A. Mayrhofer. The edns(0) padding option. Technical report, 2016.

[53] A. Mayrhofer. Padding policies for extension mechanisms for dns (edns(0)). Technical report, 2018.

[54] Nguyen Phong Hoang, Arian Akhavan Niaki, Nikita Borisov, Phillipa Gill, and Michalis Polychronakis. Assessing the privacy benefits of domain name encryption. In *Proc. of the 15th ACM Asia CCS*, pages 290–304, 2020.

[55] Nguyen Phong Hoang, Arian Akhavan Niaki, Phillipa Gill, and Michalis Polychronakis. Domain name encryption is not enough: Privacy leakage via IP-based website fingerprinting. *arXiv preprint arXiv:2102.08332*, 2021.

[56] Rebekah Houser, Zhou Li, Chase Cotton, and Haining Wang. An investigation on information leakage of dns over tls. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 123–137, 2019.

[57] Sandra Siby, Marc Juárez, Claudia Díaz, Narseo Vallina-Rodriguez, and Carmela Troncoso. Encrypted DNS -> privacy? A traffic analysis perspective. In *27th NDSS*, 2020.

[58] Andrew Reed and Benjamin Klimkowski. Leaky streams: Identifying variable bitrate dash videos streamed over encrypted 802.11 n connections. In *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1107–1112. IEEE, 2016.

[59] Andrew Reed and Michael Kranch. Identifying https-protected netflix videos in real-time. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 361–368. ACM, 2017.

[60] Melcher Stikkelorum. I know what you watched: Fingerprint attack on youtube video streams. 2017.

[61] Charles V Wright, Lucas Ballard, Fabian Monrose, and Gerald M Masson. Language identification of encrypted voip traffic: Alejandra y roberto or alice and bob? In *USENIX Security Symposium*, volume 3, pages 43–54, 2007.

[62] Charles V Wright, Lucas Ballard, Scott E Coull, Fabian Monrose, and Gerald M Masson. Spot me if you can: Uncovering spoken phrases in encrypted VoIP conversations. In *2008 IEEE Symposium on Security and Privacy*, pages 35–49. IEEE, 2008.

[63] Ye Zhu and Huirong Fu. Traffic analysis attacks on Skype VoIP calls. *Computer Communications*, 34(10):1202–1212, 2011.

[64] Liam Tung. whatsapp: Now one billion people send 55 billion messages per day, Jul 2017. URL `https://www.zdnet.com/article/whatsapp-now-one-billion-people-send-55-billion-messages-per-day/`.

[65] Scott E Coull and Kevin P Dyer. Traffic analysis of encrypted messaging services: Apple iMessage and beyond. *ACM SIGCOMM CCR*, 44(5):5–11, 2014.

[66] Walter J Scheirer, Anderson de Rezende Rocha, Archana Sapkota, and Terrance E Boult. Toward open set recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(7):1757–1772, 2012.

[67] Matthew D Scherreik and Brian D Rigling. Open set recognition for automatic target classification with rejection. *IEEE Transactions on Aerospace and Electronic Systems*, 52(2):632–642, 2016.

[68] Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, and Matthew Wright. Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1131–1148, 2019.

[69] Yanbin Wang, Haitao Xu, Zhenhao Guo, Zhan Qin, and Kui Ren. snwf: Website fingerprinting attack by ensembling the snapshot of deep learning. *IEEE Transactions on Information Forensics and Security*, 17:1214–1226, 2022.

[70] Sridhama Prakhya, Vinodini Venkataram, and Jugal Kalita. Open set text classification using CNNs. In *Proceedings of the 14th International Conference on Natural Language Processing (ICON-2017)*, pages 466–475, Kolkata, India, December 2017. NLP Association of India. URL `https://aclanthology.org/W17-7557`.

[71] ZongYuan Ge, Sergey Demyanov, Zetao Chen, and Rahil Garnavi. Generative openmax for multi-class open set classification. *arXiv preprint arXiv:1707.07418*, 2017.

[72] Darren Webb. Applying softmax classifiers to open set. In Thuc D. Le, Kok-Leong Ong, Yanchang Zhao, Warren H. Jin, Sebastien Wong, Lin Liu, and Graham Williams, editors, *Data Mining*, pages 104–115, Singapore, 2019. Springer Singapore. ISBN 978-981-15-1699-3.

[73] Poojan Oza and Vishal M Patel. C2ae: Class conditioned auto-encoder for open-set recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2307–2316, 2019.

[74] Kimin Lee, Honglak Lee, Kibok Lee, and Jinwoo Shin. Training confidence-calibrated classifiers for detecting out-of-distribution samples. In *International Conference on Learning Representations*, 2018.

[75] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[76] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

[77] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *ICML*, 2017.

[78] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[79] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[80] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[81] Tao Wang and Ian Goldberg. Walkie-Talkie: An efficient defense against passive website fingerprinting attacks. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1375–1390, 2017.

[82] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Defeating DNN-based traffic analysis systems in real-time with blind adversarial perturbations. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[83] Marshall Vale and Alexander Dupuy. Google public DNS over https (doh) supports rfc 8484 standard, June 2019. URL `https://security.googleblog.com/2019/06/google-public-dns-over-https-doh.html`.

[84] What is 1.1.1.1?, November 2019. URL `https://www.cloudflare.com/learning/dns/what-is-1.1.1.1/`.

[85] Doh with quad9 dns servers, October 2018. URL `https://www.quad9.net/doh-quad9-dns-servers/`.

[86] Selena Deckelmann. Firefox continues push to bring dns over https by default for us users, February 2020. URL `https://blog.mozilla.org/blog/2020/02/25/firefox-continues-push-to-bring-dns-over-https-by-default-for-us-users/`.

[87] Juha Saarinen. Doh! microsoft to build dns over https into windows 10, November 2019. URL https://www.itnews.com.au/news/doh-microsoft-to-build-dns-over-https-into-windows-10-534197.

[88] Catalin Cimpanu. Here's how to enable doh in each browser, isps be damned, February 2020. URL https://www.zdnet.com/article/dns-over-https-will-eventually-roll-out-in-all-major-browsers-despite-isp-opposition/.

[89] T. Pauly T. Verma E. Kinnear, P. McManus and C. Wood. Oblivious dns over https. Technical report, 2022.

[90] Donald Eastlake and C Kaufman. Domain name system security extensions. Technical report, rfc 2535, March, 1999.

[91] Frank Denis and Yecheng Fu. Dnscrypt, 2015.

[92] Daniel J Bernstein. Dnscurve: Usable security for dns. *dnscurve. org*, 4, 2009.

[93] Tirumaleswar Reddy, Dan Wing, and Prashanth Patil. DNS over datagram transport layer security (DTLS). *RFC 8094*, 2017.

[94] Kyungwon Park and Hyoungshick Kim. Encryption is not enough: Inferring user activities on KakaoTalk with traffic analysis. In *International Workshop on Information Security Applications*, pages 254–265. Springer, 2015.

[95] Mark Jackson. Google, uk isps and gov battle over encrypted dns and censorship, 2019. URL https://www.ispreview.co.uk/index.php/2019/04/google-uk-isps-and-gov-battle-over-encrypted-dns-and-censorship.html.

[96] The top 500 sites on the web, 2018. URL https://www.alexa.com/topsites.

[97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[98] W. J. Scheirer, A. de Rezende Rocha, A. Sapkota, and T. E. Boult. Toward open set recognition. *IEEE TPAMI*, 35(7):1757–1772, 2013.

[99] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 2013.

[100] Pedro R Mendes Júnior, Roberto M De Souza, Rafael de O Werneck, Bernardo V Stein, Daniel V Pazinato, de Almeida, et al. Nearest neighbors distance ratio open-set classifier. *Machine Learning*, 2017.

[101] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10), 2009.

[102] Steven CH Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. Online learning: A comprehensive survey. *arXiv preprint arXiv:1802.02871*, 2018.

[103] Lada A Adamic and Bernardo A Huberman. Zipf's law and the internet. *Glottometrics*, 3(1):143–150, 2002.

[104] Abhijit Bendale and Terrance Boult. Towards open world recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1893–1902, 2015.

[105] Weijie Chen, Di Xie, Yuan Zhang, and Shiliang Pu. All you need is a few shifts: Designing efficient convolutional neural networks for image classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7241–7250, 2019.

[106] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[107] Ashutosh Adhikari, Achyudh Ram, Raphael Tang, and Jimmy Lin. Docbert: Bert for document classification, 04 2019.

[108] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[109] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[110] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. Var-CNN: A data-efficient website fingerprinting attack based on deep learning. *Proceedings on Privacy Enhancing Technologies*, 2019(4):292–310, 2019.

[111] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: An astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 806–813, 2014.

[112] 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.

[113] Kwon Nung Choi, Achintha Wijesinghe, Chamara Manoj Madarasingha Kattadige, Kanchana Thilakarathna, Suranga Seneviratne, and Guillaume Jourjon. SETA: Scalable encrypted traffic analytics in multi-Gbps networks. In *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, pages 389–392. IEEE, 2020.

[114] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, 2020.

[115] Erico Tjoa and Cuntai Guan. A survey on explainable Artificial Intelligence (XAI): Toward medical XAI. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[116] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding Recurrent Networks. *arXiv preprint arXiv:1506.02078*, 2015.

[117] Marcus Gallagher and Tom Downs. Visualization of learning in Multilayer Perceptron Networks using Principal Component Analysis. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 33(1):28–34, 2003.

[118] Shujian Yu and Jose C Principe. Understanding autoencoders with information theoretic concepts. *Neural Networks*, 117:104–123, 2019.

[119] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1765–1773, 2017.

[120] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[121] Daehyeok Kim, Ankush Jain, Zaoxing Liu, George Amvrosiadis, Damian Hazen, Bradley Settlemyer, and Vyas Sekar. Unleashing in-network computing on scientific workloads. *arXiv preprint arXiv:2009.02457*, 2020.

[122] Yifan Yuan, Omar Alama, Jiawei Fei, Jacob Nelson, Dan R. K. Ports, Amedeo Sapio, Marco Canini, and Nam Sung Kim. Unlocking the power of inline Floating-Point operations on programmable switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 683–700, Renton, WA, April 2022. USENIX Association. ISBN 978-1-939133-27-4. URL `https://www.usenix.org/conference/nsdi22/presentation/yuan`.

[123] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory — ICDT 2001*, pages 420–434, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[124] Kazuki Onishi, Masanori Hashimoto, et al. Memory efficient training using lookup-table-based quantization for neural network. In *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 251–255. IEEE, 2020.

[125] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[126] Xin Sun, Zhenning Yang, Chi Zhang, Keck-Voon Ling, and Guohao Peng. Conditional gaussian distribution learning for open set recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13480–13489, 2020.

[127] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

[128] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, jul 2014. ISSN 0146-4833. doi: 10.1145/2656877.2656890. URL `https://doi.org/10.1145/2656877.2656890`.

[129] Chamara Kattadige, Kwon Nung Choi, Achintha Wijesinghe, Arpit Nama, Kanchana Thilakarathna, Suranga Seneviratne, and Guillaume Jourjon. Seta++: Real-time scalable encrypted traffic analytics in multi-gbps networks. *IEEE Transactions on Network and Service Management*, 18(3):3244–3259, 2021.

[130] Katharina Kohls, David Rupprecht, Thorsten Holz, and Christina Pöpper. Lost traffic encryption: fingerprinting lte/4g traffic on layer two. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, pages 249–260, 2019.

[131] David Hasselquist, Christian Vestlund, Niklas Johansson, and Niklas Carlsson. Twitch chat fingerprinting. 2022.

[132] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019.

[133] Alexander Vaskevich, Thilini Dahanayaka, Guillaume Jourjon, and Suranga Seneviratne. Smaug: Streaming media augmentation using cgans as a defence against video fingerprinting. In *2021 IEEE 20th International Symposium on Network Computing and Applications (NCA)*, pages 1–10. IEEE, 2021.

[134] KJ Joseph, Salman Khan, Fahad Shahbaz Khan, and Vineeth N Balasubramanian. Towards open world object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5830–5840, 2021.

[135] Xiaojie Guo, Amir Alipour-Fanid, Lingfei Wu, Hemant Purohit, Xiang Chen, Kai Zeng, and Liang Zhao. Multi-stage deep classifier cascades for open world recognition. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 179–188, 2019.

[136] Yee Teh, Michael Jordan, Matthew Beal, and David Blei. Hierarchical dirichlet processes. *Machine Learning*, pages 1–30, 12 2006. doi: 10.1198/016214506000000302.

[137] PK Srijith, Mark Hepple, Kalina Bontcheva, and Daniel Preotiuc-Pietro. Sub-story detection in twitter with hierarchical dirichlet processes. *Information Processing & Management*, 53(4):989–1003, 2017.

[138] Zekai J Gao, Yangqiu Song, Shixia Liu, Haixun Wang, Hao Wei, Yang Chen, and Weiwei Cui. Tracking and connecting topics via incremental hierarchical dirichlet processes. In *2011 IEEE 11th International Conference on Data Mining*, pages 1056–1061. IEEE, 2011.

[139] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)*, pages 233–248, 2018.

[140] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

[141] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. *arXiv preprint arXiv:1711.10563*, 2017.