Deep Learning Approaches to Malware Detection of Encrypted Traffic in Community Networks

Lucas Carr
CRRLUC003
University of Cape Town

ABSTRACT

The ability to perform intrusion detection on a network is an important component of having reliable and secure networks. The systems which have historically been used for intrusion detection are increasingly ill-equipped to provide this service. This is largely due to the widespread adoption of encryption protocols in networking. Consequently, new approaches towards intrusion detection, which do not rely on the ability to inspect payloads of traffic, are required. Various approaches have been taken to this challenge; however, the most promising are is present in deep learning techniques. There are numerous examples of recent work using deep learning to classify encrypted network traffic - a task closely aligned to detecting malicious traffic. In this paper, we will review these approaches, as well as holding a discussion of the technologies these approaches use. More specifically, this discussion will note an awareness around the computational and data requirements for these approaches. We will go on to identify malware detection in encrypted traffic as a less explored field which faces difficulty due to an absence of high quality data.

Keywords: Malware Detection, Networks, Deep Learning, Autoencoders, Recurrent Neural Networks, LSTMs,

1 INTRODUCTION

The last two decades have witnessed a marked increase in use of networks; applications of which are ubiquitous to areas of modern life. The has been helpful, due to its ability to facilitate transactions and forms of communication without geographic limitations, enabling new connections which would otherwise be impossible. However, adoption of networks as a vehicle to distribute information has added new dimensions of risk: a non-trivial amount of the information which flows through networks is security-critical: passwords and financial transactions are examples of such. There is an adjacent argument argument that individuals are entitled to have their personal data be private - which would extend the list of security-critical information to browser activity, correspondence, and Insert another example here. The advent of malicious software, or malware, as an attempt to leverage this increased risk for gain, is an increasingly common occurrence.

Once malware has infected a host, or network of hosts, it is both expensive, and difficult to remove - the cost of a single malware attack has previously amounted to an excess of a billion dollars. Malware, despite the amount of recent attention it is awarded, is not a novel phenomenon. Historically, networks were able to secure themselves against malware through the use of Network Intrusion Detection Systems (NIDS). NIDS are intended to be able to identify and block malware attempting to enter, or propagate through, a network. This was achieved through the use of a broad range of approaches; specifically, port analysis, deep packet inspection (DPI), or statistical modelling of packet flow [22] [6].

However, more recent standard practice around networking has made it difficult for the aforementioned detection systems to function well. Port-numbers have become a less reliable indicator of an application type; moreover, port-obfuscation is a technique which masks the destination port of some traffic [22]. Similarly, the adoption of dynamic IP addresses have also made it difficult for NIDS, given that NIDS rely on their ability to associate traffic from specific IPs with devices - a difficult task if the IP address changes frequently. Furthermore, much of modern internet traffic undergoes some encryption protocol; notably, in 2017 $\approx$ 75% of analyzed malware made use of encryption [21] . Encrypted packets make conventional DPI inspection impossible, since they require access to the payloads contained in packets - which are now encrypted [17]. It is important to note that these practices are useful, and in many cases the reason for their adoption is that they offer significant security benefits - port obfuscation, for example, can prevent attackers from targeting specific ports for specially created attacks, since their required ports no longer function as expected.

Consequently, novel approaches to intrusion detection, which are capable of working within the previously mentioned network practices, are required. Machine Learning (ML) applications are capable of such; however, many of the existing ML-based intrusion detection models require carefully engineered feature-selection, which is a slow and difficult task [13] [22]. This paper aims to explore the applications of deep learning (DL) to the problem of intrusion detection on encrypted traffic; the motivation behind using DL, is that some current DL methods are quite well suited to automatic feature extraction, extendable to extracting features from encrypted data - reducing the requirement of hand-crafted features [23]. Moreover, DL models are capable of reasoning with a much higher dimensionality than shallow ML counterparts - which enables them to learn more complicated patterns [22].

Given this inability to rely on existing NIDS, such as Snort, community networks are especially vulnerable. Generally defined as being community-run, low-resourced networks, community networks require lightweight tools which can protect them against malicious traffic. It is quite common for discussions of DL to ignore, or gloss over the performance requirements of their applications - usually willing to sacrifice performance for accuracy in results. The following discussion will attempt to balance these two factors in the hope that a middle-ground can be discovered.

The proceeding sections of this paper are organized as follows: Section 2 will discuss concepts which are required for understanding a more detailed scope and direction of this paper - expanding on different types of malware, the difference between flow and payload analysis, etc. Section 3 will discuss existing work tangential to the scope of this paper. Section 4 will hold a thorough discussion of different DL algorithms, providing both insight into how the algorithms work (and why they are potentially useful for this task), as well as existing work which uses these algorithms for similar applications. Significant takeaways from 4, including successful existing work, will be further expanded upon in Section 5. Lastly, the most significant conclusions from this review, as well as a brief plan for future work will be detailed in Section 6

2 BACKGROUND

The following section will introduce a variety of loosely related topics which are essential to understand should one engage with the scope of this paper. This section will not discuss approaches towards addressing the problem of malware detection of encrypted traffic, but rather familiarize the reader with the adjacent ideas.

2.1 Community Networks

Community Networks are defined as networks which are built and managed by a community. These are often decentralized, poorly resourced, and managed without industry-standard expertise [5]. Moreover, the users of these networks typically own devices which, due to a mixture of bandwidth constraints and device age, are not patched with the latest security updates. While community networks are not necessarily a more likely target for malicious traffic, they are less equipped to manage intrusions on their network. This is in part due to their decentralized design - a private business network which detects an intrusion has the ability to remove and clean infected components, this task is much more challenging in a community network. Moreover, community networks are not able to enforce strict guidelines around good cyber-security practices; this, coupled with un-patched security exploits, make them vulnerable to malicious attacks [14].

2.2 Types of Malware

Malware is a fairly broad category, defining software which, without permission, installs itself a computer, or network with some malicious intention [9]. The aims of malware are vast, ranging from theft of personal information, to government-based attacks on foreign adversaries. Just as the aims of malware are broad, so are the methods a piece of malware might use to accomplish its goal [16]. Despite the many forms of malware, certain types garner more attention - typically due to their proclivity to effect a wide audience, or cause significant damage. We will discuss these types of malware, specifically: worms, ransomware, and botnets, as well as introducing significant instances of these malware [16].

Ransomware is malware which, after infecting a host, encrypts the data stored on the system. The user is prevented from accessing their data until - usually - some ransom is paid to the distributors of the malware [2]. Ransomware attacks have grown in popularity, as they are able to not only infect individual users, but also shutdown institutions. WannaCry, also known as Wana Decrypt0r, was a ransomware attack which took place in 2017, infecting millions of systems across the world. The affected systems extended to hospital networks and government sectors [2] [9]. Although WannaCry is considered to be an instance of ransomware, it also into another category of malware, called a worm [24]. A worm is most distinctly a piece of malware which is able to self-replicate, and propagate through a network, infecting other connected machines [16]. Worms can be used to slow down a networks performance, as the worm infects more and more machines, more replications are made and sent out across the network, causing congestion [9]. Another use for worms, as is seen in WannaCry, is to attach a payload to the worm - this payload can be an instance of another piece of malware - usually a virus, or Trojan [16]. Generally, malware attacks use adaptations of known malware instances; this is helpful for detecting malware, as there will be sub-strings in the malware payload which are present in known malware, enabling identification [24]. However, zero-day attacks and polymorphic worms challenge this notion - these, respectively, have no known predecessors, or *morph* as they spread through a network [16] [24].

The final category of malware we will speak about are botnets; a botnet is described as a network of infected computers, bots, which can be taken over and controlled remotely to perform tasks [9]. Primarily, botnets are used to perform attacks on a large scale - for instance, DDoS attacks, or massive email spam [9]. As was seen with ransomware, specifically, WannaCry, a botnet will usually combine various types of malware into an attack - using a Trojan to install the botnet on a system, and a worm to spread to other systems. Gameover Zeus is an example of an adaptation of known malware, a botnet called Zeus, which is peer-to-peer network, where peers share updates with each-other and exchange information around locations to store stolen data [3].

2.3 Payload versus Flow Analysis

As mentioned in Section 1, historically, payload analysis has been performed using methods like DPI. These approaches examine the contents of a packet's payload, looking for pre-defined patterns, or expressions. These pre-defined patterns correspond to known applications, or malware - matching a pattern to a single packet's payload enables the classification of traffic [13]. The practice of using known signatures to classify packets is acceptable when it is reasonable to assume that a large portion of signatures are already known - for benign traffic, this is likely the case. However, zero-day attacks and amorphous malware challenge this assumption - both being examples of traffic which, by design, is difficult/impossible to have an existing signature of [8]. Additionally, as has been mentioned in Section 1, the popularization of encrypted payloads makes it impossible for this type of payload analysis to function - if the payloads are encrypted, they cannot be inspected to see if they contain known signatures. The deep learning approaches which are discussed in the next section do not require the payloads to be inspected for known signatures [8] [7].

Another method of traffic analysis is known as flow-based analysis. Where payload analysis examines individual packets, flow-based analysis will group packets together, in accordance to the network flows they are a part of [23]. A network flow is an object consisting of multiple packets, where all the packets have the same 5-tuple of header information, *{source IP, destination IP, destination port, transport-level protocol}* [1]. Section 1 introduced the concepts of dynamic IP addresses, and port obfuscation which are increasingly being incorporated into network protocols - this has made flow analysis more challenging, since it relies on this information to recognize flows [12]. Moreover, data-sets of encrypted network traffic are usually in the form of raw streams of data - identifying and extracting flows from this data is time consuming process [23]. This factor, coupled with the problems faced by flow-analysis, involving dynamic IPs and port obfuscation, make payload-based classification a more desirable approach to identifying malware.

2.4 Gathering Data

Machine learning is a process for which it is imperative to have high quality data; the essence of ML applications is to use training data to teach a model something. If the training data is of a poor standard - what exactly 'poor' means will be discussed further on - the training process is unlikely to produce a useful model. The previous section looked at flow versus payload analysis; these are the two main approaches which make up the basis of classifying network traffic. Consequently, when using applications of machine learning to classify network traffic, we deal with datasets made up of flow or payload based data [23]. The properties which determine whether a dataset is good are best realised in relation to the task being tackled; with respect to malicious traffic detection, a large dataset containing real-world network traffic might not be suitable as training data due to malicious traffic being substantially less common than benign traffic, resulting in an imbalanced dataset [21]. Wang et al. [21] posit that training a model to detect or classify malicious traffic requires a dataset that is (1) sufficiently large, (2) contains a large variation of different, encrypted malware traffic, and (3) has a good balance between benign and malicious traffic. Unfortunately, there are no obvious datasets which meet these criterion [21].

Assuming some suitable dataset has been found, a pre-processing step is usually required. Pre-processing techniques vary based off the methodology used. For instance, convolutional neural networks require a uniform input size, while packet payload sizes are typically varied. A solution to this is to employ truncation or padding to packet payloads in a dataset [21] [11]. If one were to use an LSTM, this pre-processing step would not be required, since LSTMs accept inputs of variable size [10].

In the literature, datasets would typically consist of raw input data, which was then split into discrete PCAP files, this step would be followed by a process of 'cleaning' - which is where unnecessary information would be removed [22]. If the task were to perform classification based entirely off payloads, information stored in the packet header might be removed in order to prevent the neural network from using this information to learn [13]. As mentioned earlier, the use of a CNN requires inputs of a uniform length. This was achieved through deciding on an appropriate length, Lotfollahi et al. [13]selected 1480 bytes of the payload. Packets with payload sizes less than 1480 would be zero-padded to make up the missing byes [12].

2.5 Evaluating Performance

There are a set of conventional metrics which have been employed in Zhou *et al.* [24] and Zeng *et al.*'s [22] work on malware detection. These metrics are aimed at providing a comprehensive base of coverage of evaluating an algorithm's ability to make classifications; specifically, they are Precision, Recall, F1-score, and Accuracy. Given,

- TP - true positive
- FP - false positive
- TN - true negative
- FN - false negative

We can then define these metrics as:

$$(1)\ \text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

$$(2)\ \text{Precision} = \frac{TP}{TP + FP}$$

$$(3)\ \text{Recall} = \frac{TP}{TP + FN}$$

$$(4)\ \text{F1-score} = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

The accuracy of a model refers to the fraction of classifications it makes which are correct; generally, this is a useful measurement of a models performance. However, a significant limitation of this metric is that it can produce misleading results - in the case of an unbalanced data-set; for example, a model might always correctly classify x type of traffic, and incorrectly classify everything else. If x made up 90% of the data-set, the accuracy of the model would be misleadingly high [12].

The F1-score evaluation metric provides an alternative to this. F1-score is a harmonic mean, calculated using *Precision* and *Recall* - where *Precision* is the measure of what portion of total positive predictions made by the model are true positive, and where *Recall* is the measure of what portion of true positives your model predicted to be positive [18] [12]. Consequently, F1-score is a balanced representation of these two measurements, represented in a single metric, where a 1 indicates a perfect *Precision* and *Recall* score, and a 0 when either *Precision* or *Recall* are 0.

3 RELATED WORKS

There is a significant corpus of work on the area of using DL to classify network traffic; as the task is further specified - for instance, requiring that the traffic be encrypted, or include classification of malicious traffic, the available work is reduced. In Section 4, the discussion includes works related to the relevant DL algorithm - to avoid redundancy, these will not be discussed in this section. Instead, the following will look at works which are not related to

any of the discussion in Section 4, but are relevant to the scope of this paper.

BlindBox, proposed by Sherry et al. [20], is an attempt to perform DPI on encrypted payloads, ultimately, acting as an IDS. The BlindBox software is a middlebox, meaning it exists on a network path and monitors the traffic flowing through it. Middleboxes which came before BlindBox were unable to handle encrypted traffic. Since the payloads were encrypted, the middlebox would either *i)* simply not work, or, using a man-in-the-middle attack, *ii)* the middlebox would decrypt the traffic to access its payload and perform conventional DPI [20]. Implementations of *ii* have been criticized, as they are violations of assumed privacy offered by encryption.

BlindBox works by generating a set of rules, or, encrypted signatures, which are based off known, suspicious keywords [20]. Furthermore, all connections on a network which uses BlindBox, must use a specific BlindBox HTTPS configuration. BlindBox functions by essentially requiring network traffic to use two encryption schemes: the conventional SSL-encryption for the payload, as well as a novel token encryption called DPIEnc, which encrypts *tokens* [20]. When the traffic flows through the network, the payload remains secure, while the BlindBox system compares its set of encrypted signatures against the DPIEnc encrypted tokens - this comparison is a process which is facilitated through a tool called BlindBox Detect [20]. This approach leverages the idea that certain parts of a payload - which can be considered important for threat detection - can be tokenized, and encrypted using an in-built encryption scheme, enabling them to be analyzed using BlindBox. This, strictly speaking, does maintain the integrity of the encrypted payloads; however, the set-up process of BlindBox is substantial, requiring every device which operates on the network to install an compliant HTTPS configuration [20]. Moreover, BlindBox only works on HTTPS traffic [13].

Papadogiannaki et al. [17], recognizing the difficulty NIDS have when handling encrypted traffic, propose an approach to intrusion detection which uses signatures based off payload sizes. This approach attempts to generate a comprehensive language of malicious traffic signatures, where a signature is defined as a sequence of consecutive payload sizes; network flows are inspected to see if they contain these signatures [17]. For example, Papadogiannaki et al. [17] recognized that should a network flow contained a sequence of four packets of respective sizes 22, 976, 48, 16, an intrusion attempt of a Hydra password cracking tool has taken place. An important aspect of this approach is the size of the signature - or, how many payload sizes a signature refers to. If a signature is too short, it is likely to result in false positives; false positives are particularly undesirable for IDS since they undermine the confidence of the system, which can lead to ignoring real intrusion events.

While most of the approaches to intrusion detection in encrypted network traffic discussed in this paper attempt to extract features from the encrypted payload - a challenging process, the work of Papadogiannaki et al. [17] benefits from its simplicity: it does not require any information from the payload, other than the payload size. This also imposes limitations on the efficacy of the approach. Firstly, only known malicious traffic instances can be detected - since detection of malicious traffic requires an existing signature [17]. Moreover, since the only feature used to detect traffic is a sequence of payload sizes - creators of malware could quite easily avoid detection by padding malware - increasing the payload sizes.

4 DISCUSSION OF RELEVANT DEEP LEARNING MODELS

The following section will attempt to engage quite thoroughly with literature concerning four distinct deep learning methodologies: multi-layer perceptrons, convolutional neural networks, stacked autoencoders, and recurrent neural networks. Each method will have its own sub-section, where the initial discussion will be a discussion of the theory behind why the method is used, and what advantages it has over other methodologies. The discussion will then progress to an exploration of existing implementations of the methodology, tailored to tackling problems adjacent to malware and encrypted traffic classification. In sum, each sub-section will have a theoretical breakdown as well as a discussion of efficacy of the model in existing implementations.

4.1 Multilayer Perceptrons

Multilayer perceptrons (MLPs) were one of the earliest forms of neural networks, and as such, express concepts which are fundamental to the more advanced neural network which are discussed further on. The goal of an MLP model is to, as best as possible, emulate some function [10]. An MLP consists of an input layer, one or more *hidden layers*, and an output layer - all of which are fully connected to the next layer. The depth of a model is described as the number of hidden layers.

Aceto et al. [1] implement two different MLPs with one and two hidden layers - from their results, the MLPs performed considerably worse than their DL counterparts, with F1 of ≈ 0.7. However, it should be recognized that the inclusion of MLPs in this discussion is not a suggestion that they are a potentially good methodology to classify encrypted network traffic. Rather, as is proposed in Aceto et al. [1], an MLP can provide a useful baseline performance achievable by a shallow neural network.

4.2 Convolutional Neural Networks

Convolutional neural networks have become one of the most utilized neural network architectures, initially due to their impressive ability to perform pattern recognition on image data [15]. Applications of CNNs have had great success when working with grid-like input data; an image should be thought of as simply a 2x2 matrix, where the value of each element in the matrix corresponds to a pixel. This idea can be extended to RGB images by thinking of each element in the matrix as a vector of three numbers, corresponding to the red, green and blue channels [10]. More recently, there have been successful applications of CNNs with respect to network traffic classification.

The architecture of a CNN can be broken into three clearly distinct stages, or layers. As is typical in most neural networks, there

is an input layer, comprising of nodes which correspond to discrete values of the overall input - for instance, pixels.

Suppose the image contains m neurons in the input layer (pixels), and n neurons in the second layer. If each pixel were connected to each neuron in the second layer - known as a fully connected layer - the network becomes increasingly expensive to train, especially so when given larger inputs [10]. Instead, CNNs implement a sparsely-connected layer, where each input neuron has a limited number of outgoing connections. Significantly, networks which limit the number of outgoing connections each input neuron can have to k, have achieved good results with k being significantly smaller than m [10]. *Figure 1* depicts a sparsely connected network, where neuron, *c3*, is only connected to neurons *m2, m3* and *m4* - these neurons are known as the receptive field of *c3*.

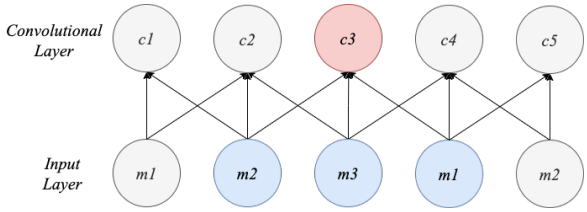

Figure 1: Visualization of sparsely connected layers

In the convolutional layer, an $n \times n$ kernel will traverse through the input layer, producing an activation map. If the input to a CNN is assumed to be a two dimensional matrix, with a convolutional layer containing a 3×3 kernel, each element of the input will go through a convolution. Once the kernel has traversed the entirety of the input, a corresponding feature map would have been produced. [15]

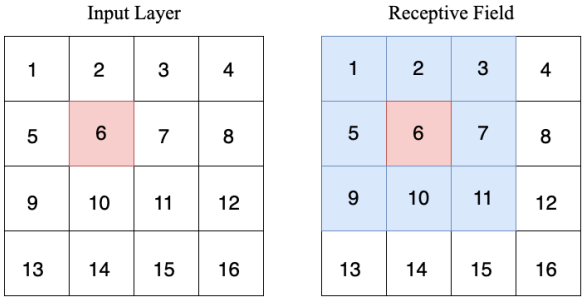

Figure 2: Visualization of the receptive field of a neuron

A feature map can be thought of as a representation of the activated areas of some input data, after applying a convolution operation [15]. Which areas are activated depend on the activation function, as well as the make-up of the kernel, in shallower layers, feature maps generally represent small concepts, or relations, in data; for instance, a map of horizontal edges. As we go deeper into the network, feature maps begin to represent more complete objects - it is important to note that these are not necessarily objects which, if we could see them, would make sense, so it is not useful to think of feature maps as necessarily representing blocks of information we use to identify an image [15] [10]

Figure 2 demonstrates a 2-Dimensional input - due to their applications in image processing, CNNs are often given two dimensional grid-like input (2D-CNNs). In the case of images, it seems intuitively correct to use 2D-CNNs, given that elements in an image are spatially related to one another in two dimensions. Network traffic, however, is sequential, and, unlike images, do not require a two dimensional representation [22].

One approach to using CNNs to process with network traffic, is to convert the stream of data stored in a packet into a two dimensional structure - essentially, forming transforming it into an image. However, Lotfollahi et al. [13] implemented a 1D-CNN as method of classifying encrypted network traffic. It was argued that a 1D-CNN is more suitable than a 2D-CNN given its ability to better capture the relationship between adjacent bytes, finding notable features in the data stream. The first convolutional layer had a filter size of four, a stride of three, and 200 channels. The second layer had a filter size of five, a stride of one, and 200 channels. Their network then added a final softmax classification layer - obtaining an F1-score of 0.98 [13].

Zhou et al. [24] discussed approaches to worm detection using DL methods. They implemented a 1D-CNN, arguing that, although CNNs typically have two dimensional matrices as input, network traffic is naturally one dimensional; on top of that, using a one dimensional vector as input data would result in decreased storage space, and faster training times [24]. In their discussion, Zhou et al. experimented with three different CNN architectures, with one, two and three convolutional layers, each convolutional layer would be followed by a pooling layer. The final pooling layer was flattened into a one-dimensional vector, and connected to the fully connected layer. A final softmax layer is added for classification. The CNNs - which performed a binary classification of whether network traffic was a worm or not - achieved F1-scores of 0.92, 0.921, and 0.924, respectively.

Noticeable in both Lotfollahi et al. [13] and Zhou et al. [24] is the use of a pooling layer after each convolutional layer. Pooling is a technique many CNNs implement as a method of downsampling the output of a convolutional layer [1]. A pooling layer will implement some sort of pooling function; most commonly, max pooling is used - an approach which splits the data into local areas, and reduces each segment to its maximum value [10]. The benefits of pooling are get smaller feature maps, discarding unimportant information and speeding up training. Additionally, dimensionality reduction caused by pooling effectively increases the receptive field of neurons in the layer, which helps the model generalize better, as it learns invariant features [1].

4.3 Autoencoders

Autoencoders (AEs) are an example of an unsupervised learning algorithm which, given some input, attempt to reconstruct this input. An AE can be bifurcated into an encoder function, which transforms some input into a latent vector, *or code*. And a decoder function, which reconstructs the input from the encoding [10]. This

explanation of AEs, as a tool which essentially are an approximation of an identity function, is not particularly interesting. However, AEs are not designed to recreate their input exactly, instead, the output is intended to be an approximation of the input - with loss being minimized.

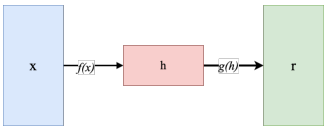

Figure 3: Input x, encoded to h, decoded to r

The significant aspect of AEs appears when looking at the encoding process. The latent vector, h, can be thought of as having a much lower dimensionality than the input, x. This encourages the encoding function to learn only the seminal features of the input data, in order to construct an approximation of it. This results not only in dimensionality reduction, but also automatic, seminal feature identification. The learning process uses a loss function which minimizes the reconstruction error - which is defined as the divergence between $g(h)$ and x [10]. If h is given too large a dimensionality, the algorithm fails to extract only the seminal features of the input, as the only learning motivation is minimizing the reconstruction error.

Sparse autoencoders are an attempt to address this issue; they function similarly to traditional AEs, but with an additional sparsity penalty, $\Omega(h)$. Sparsity loosely relates to neurons being 'inactive' more than not. The algorithm now learns by minimizing both the reconstruction error, and the sparsity penalty [10].

Another variation of AEs is known as a stacked autoencoder (SAE) - these are essentially a sequence of AEs, where the output from one AE is used as input into the next one in sequence. Lotfollahi et al. implemented a five-layer SAE, where layers were made up of 400, 300, 200, 100 and 50 neurons, respectively, with a final softmax layer added for classification [13]. This decreasing size of each layer is suggests that the capacity of the encoder was reduced, in an attempt to encourage it to only learn the seminal features of its input data. There was, however, no mention of enforcing sparsity among the layers of the network - consequently, it was assumed that there was no inclusion of sparse AEs within this network. The results showed that their SAE was successfully able to identify the related application of a packet - achieving a weighted average F1 score of 0.92 [13]. Notably, the model used encrypted payloads of packets as training data - a divergence from the norm of traffic classification, which uses traffic flows.

One of the risks of reducing the dimensionality of the hidden layer of an AE - which is not necessarily present in Lotfollahi et al. - is that the dimensionality is reduced too significantly, preventing the model from encoding the necessary features into the hidden layer(s). Using a sparse autoencoder mitigates this risk, since it is not physically reducing the dimensions of the input - rather it discourages unnecessary activation of neurons in a balancing act between reducing reconstruction error, and enforcing sparsity [10].

Zeng et al. [22] included an SAE in their three-model classifier, DeepFullRange. Their SAE had two layers; the first layer had 900 input neurons, fully connected to encoder of 1000 neurons, this encoding was then connected to a second encoder of 1500 neurons. Finally, a softmax layer was added for classification. Interestingly, this approach appears to misaligned with the earlier discussion on limiting the capacity of encoders to encourage the model to only learn the important features of the input data. There paper offered little justification of their SAE design; one explanation for why their encoders had such a high capacity might be that they applied a dropout on each layer - however, dropouts are intended to act as a prevention against overfitting, and not as a tool to encourage sparsity. It is perhaps worth noting that the SAE in this paper was the model which performed the worst (relative to a CNN and LSTM) [22].

4.4 LSTMs / Recurrent Neural networks

A recurrent neural network (RNN) is a particular type of neural network, geared towards handling sequential input data, which implements feedback loops in its hidden layer(s) [19]. These loops enable information which has flown through the network to persist, accessible to future iterations of the forward-pass [10]. In other words, an RNN *remembers* the information which flows through it, and uses this memory to affect future data. The motivation for this design is the idea that past - in terms of prior elements in a sequence - information is likely to be relevant to understanding current information [10]. This notion becomes quite apparent if we think about how we might predict the next character, given the sequence *a, b, c, d, e* - we would look at the sequence, recognize that it is the alphabet, and fill in what should follow after '*e*'. This is an example of recent memory being useful for reasoning. However, consider a paragraph of text, starting with *"I am going on holiday and don't have accommodation*, several unrelated sentences, and ending with *"I need to book my ...."* , we should might recognize that the next word should probably be *accommodation* [10]. The principle is the same as the previous example: past information is useful for current reasoning. Only the useful piece of past information is, temporarily, much further away. This notion of temporal relationships between elements in a sequence with large intervals is an area where, due to their use of gradient descent, RNNs struggle [4] [19].

A solution to this problem, sometimes referred to as 'vanishing gradient problem', is to use Long Short-Term Memory (LSTM) . These are a variant on traditional RNNs which introduce the idea of a *memory block* [19]. The memory block is responsible for storing past information, and has an internal state. Where an RNN node uses an input value combined with a recurrent connection as input to its activation function, an LSTM cell will additionally include the internal state in its calculations. The state will then be updated, according the output of the activation function. How much influence the state has on the cell, how much it is updated by the output of the cell, and what persists in the memory block, is determined by parameters called gates - specifically, an input gate, and output gate and forget gate [19]. It is this control of the flow of information, offered by the gates, which enable it to avoid the vanishing gradient

problem.

An application of LSTMs to classify network traffic was done by Lim *et al.* [12], where two types of networks were evaluated: a multi-layer LSTM and a combinatory network architecture, comprising of a CNN + LSTM. This approach aimed to use payloads of traffic from a single flow, transformed into image data, as training data; the justification for transformation was that historically, deep learning models have had success classifying image data. Both the 3-layer LSTM and CNN+LSTM would take an input of a dataset of image-converted payloads from a unique flow, where experiments were run using different flow sizes, and different payload sizes. The results indicated that the both multi-layer LSTM and the CNN+LSTM benefited from larger flows, and larger payloads; the multi-layer LSTM outperformed the CNN+LSTM in every instance, with their best F1-scores being 0.99575 and 0.9886, respectively. However, it was noted that as the payload and flow sizes increased, the difference in F1-score decreased. It was concluded that addition of a CNN - used for feature extraction - was unnecessary, and that the unchanged payloads, used in the multi-layer LSTM, were more suitable suitable for classification using an LSTM. However, it should be acknowledged that these experiments operated on unencrypted payload data - the introduction of of encrypted payloads may necessitate the use of the feature extraction capabilities of a CNN.

In their paper, *Deep-Full-Range*, Zeng et al. [22] also implemented a multi-layer LSTM - alongside a CNN and SAE, which are discussed in 4.2 and 4.3, respectively. The LSTM was a three-layered model, where each layer had 256 LSTM cells and made use of dropout - a process whereby random components of a network are *dropped* during a training instance, in order to combat over-fitting [10]. The only F1-score provided, 0.9987, is an average across the three algorithms discussed in the paper, which makes it difficult to draw conclusions about the performance of the LSTM. However, Zeng et al. discussed the procedures they took to ensure that their dataset was balanced, and provided the accuracy ratings for their LSTMs with L1 and L2 regularization applied - which were 99.22% and 97.33%, respectively. This is useful, as it affirms that there is merit to the notion of using LSTMs to classify encrypted network traffic - and, by extension, detect malware in encrypted network traffic.

5 DISCUSSION

There exists a substantial amount of work which engage with ideas around using DL techniques to perform network classification. We considered network classification to be a broad term, which included more specific fields like: encrypted traffic classification, malicious traffic identification, and classification of malicious traffic which is encrypted. These areas - bar the last one - each have a significant amount of work to draw on. Specifically, the success of approaches seen in Zeng et al. [22] and Aceto et al. [1] - which are both applications of DL to classify encrypted network traffic - serve as an indication that DL is well suited to overcome the challenge posed by encrypted data on classification problems.

The initial method introduced was a simple MLP - as described in Section 4.1, the inclusion of MLPs was not out of an expectation that they might perform this task well; but rather to produce a baseline of what a shallow network might be capable of achieving. This simple model would then offer a starting point for comparisons with more elaborate methodologies. Of these, CNNs appeared to be the most widely used approach. However, engaging with their implementation produced the insight that many of these CNNs diverged from the conventional approach - which is to use a grid-like two-dimensional matrix as an input. Instead, CNNs used for traffic classification implemented a one dimensional vector [13] [24] [22]. This decision being justified for the reasons that: a one dimensional vector better represented the input data of a packets payload, and the argued performance benefits, found in Zhou et al. [24]. LSTMs were introduced as an answer to the vanishing gradient problem which exists in more conventional recurrent networks; Zeng et al, [22] implemented an LSTM which classified encrypted traffic, alongside a CNN and SAE - the LSTM was marginally less effective than its counterparts. However, Lim et al, [12] proposed an architecture consisting of a CNN which feeds into an LSTM with encouraging results. SAEs were another, less common approach which generally produced

There was, unfortunately, a noticeable lack of work dealing specifically with the field of intrusion detection in encrypted network traffic. This is slightly concerning, given that this is essentially the aim of this work; however, the noted success of DL applications to classify benign encrypted network traffic, as well as the success of DL applications to identify unencrypted malicious traffic, suggests that the reason for an absence of literature in this field is not necessarily due to it being intrinsically difficult problem to tackle. Rather, as was discussed in Section 2.4, it is more likely an indication of an absence of high quality training data [21]. Moreover, the shortcomings of the approaches discussed in Section 3 suggest that there remains a requirement for a effective way to perform intrusion detection on encrypted network traffic.

Given this insight, and reconciling the notion that, in order to gather good quality training data, different datasets may need to merged, a decision was made to focus more attention onto payload-based approaches, over flow or mixed approaches. In the case where reliable, clean training data is scarce, flow-based analysis is considerably more challenging, as it requires more pre-processing steps to create adequate training data.

6 CONCLUSION

There has always been a need for networks to be secure and reliable; frequently, they are used as a method of communicating sensitive information. Historical techniques to ensure this, like traditional NIDS, are quickly becoming ineffective at fulfilling this role. This is due to the changes in network protocols - specifically, encryption -,which alter the data a NIDS is able to process. Consequently, more effective, novel solutions are required.

In this review, an initial attempt was made to introduce topics peripheral to the core problem. The idea was that understanding these areas first, would be beneficial to one's engagement with the main discussion of this paper. This periphery discussion introduced the concept of community networks. Addressing how, due to certain defining traits - such as their generally low-resourced infrastructure - community networks may be considered more vulnerable to malicious attacks. Proceeding this, an exploration of the different types of malware, their respective strengths, and how multiple instances of malware might be used in conjunction with one another. This was not an exhaustive discussion of the types of malware, but rather sought to provide a reasonably comprehensive introduction into important aspects of malware.

The peripheral discussion was followed by a section which aimed to introduce related works - importantly, the papers discussed in this section were related works that did not make use of DL methodologies, since these would be discussed in tandem with an analysis of the DL methodologies. It was recognized that the approaches presented in this section were ultimately too limited to be considered viable forms of intrusion detection on encrypted traffic.

A promising approach to this area of research is found in the use of DL methodologies. An exploration of four distinct approaches - MLPs, CNNs, SAEs, and LSTM - was done, which involved an attempt to explain how each of these approaches functioned, as well as detailing existing experimentation using these technologies. From this, it was found that CNNs and LSTMs appeared to be the most promising technologies for the task of classifying encrypted traffic. Moreover, a variation of these approaches which sought to combine CNNs and LSTMs into a single architecture yielded promising results.

It was noted, however, that the literature discussed in this paper largely focused on either detecting unencrypted malware traffic, or, classifying encrypted traffic without a focus on malware. Although this is unfortunate, it was not interpreted as a sign that *i)* the research topic was unimportant, or that *ii)* the topic was too challenging to produce seminal contributions. Rather, a possible explanation for the scarcity of literature on this exact topic was that the required datasets are rare.

Moving forward, the most significant hurdle will be to find, or create a suitable dataset for the task. After this, a decision will be made around whether to use payload or flow as an input to the DL methods; this decision will determine the subsequent preprocessing steps on the dataset. Once this is done, and the dataset has been processed, cleaned and labelled, we will begin experimenting with variations DL algorithms, using the findings from this review.

REFERENCES

[1] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. 2019. Mobile Encrypted Traffic Classification Using Deep Learning: Experimental Evaluation, Lessons Learned, and Challenges. *IEEE Transactions on Network and Service Management* 16, 2 (2019), 445–458. https://doi.org/10.1109/TNSM.2019.2899085

[2] Maxat Akbanov, Vassilios G Vassilakis, and Michael D Logothetis. 2019. WannaCry ransomware: Analysis of infection, persistence, recovery prevention and propagation mechanisms. *Journal of Telecommunications and Information Technology* 1 (2019), 113–124.

[3] Dennis Andriesse, Christian Rossow, Brett Stone-Gross, Daniel Plohmann, and Herbert Bos. 2013. Highly resilient peer-to-peer botnets are here: An analysis of Gameover Zeus. In *2013 8th International Conference on Malicious and Unwanted Software: "The Americas" (MALWARE)*. 116–123. https://doi.org/10.1109/MALWARE.2013.6703693

[4] Y. Bengio, P. Simard, and P. Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* 5, 2 (1994), 157–166. https://doi.org/10.1109/72.279181

[5] Bart Braem, Chris Blondia, Christoph Barz, Henning Rogge, Felix Freitag, Leandro Navarro, Joseph Bonicioli, Stavros Papathanasiou, Pau Escrich, Roger Baig Viñas, Aaron L. Kaplan, Axel Neumann, Ivan Vilata i Balaguer, Blaine Tatum, and Malcolm Matson. 2013. A Case for Research with and on Community Networks. 43, 3 (jul 2013), 68–73. https://doi.org/10.1145/2500098.2500108

[6] Qiumei Cheng, Chunming Wu, Haifeng Zhou, Dezhang Kong, Dong Zhang, Junchi Xing, and Wei Ruan. 2021. Machine learning based malicious payload identification in software-defined networking. *Journal of Network and Computer Applications* 192 (2021), 103186.

[7] Qiumei Cheng, Chunming Wu, Haifeng Zhou, Dezhang Kong, Dong Zhang, Junchi Xing, and Wei Ruan. 2021. Machine learning based malicious payload identification in software-defined networking. *Journal of Network and Computer Applications* 192 (2021), 103186.

[8] Ronald Cheng and Gavin Watson. 2018. *D2pi: Identifying malware through deep packet inspection with deep learning*. Technical Report. Tech. Rep.

[9] Harjeevan Gill. Malware: Types, Analysis and Classifications. (????).

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[11] Hyun-Kyo Lim, Ju-Bong Kim, Joo-Seong Heo, Kwihoon Kim, Yong-Geun Hong, and Youn-Hee Han. 2019. Packet-based Network Traffic Classification Using Deep Learning. In *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. 046–051. https://doi.org/10.1109/ICAIIC.2019.8669045

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

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

[14] Panagiota Micholia, Merkouris Karaliopoulos, Iordanis Koutsopoulos, Leandro Navarro, Roger Baig Vias, Dimitris Boucas, Maria Michalis, and Panayotis Antoniadis. 2018. Community Networks and Sustainability: A Survey of Perceptions, Practices, and Proposed Solutions. *IEEE Communications Surveys Tutorials* 20, 4 (2018), 3581–3606. https://doi.org/10.1109/COMST.2018.2817686

[15] Keiron O'Shea and Ryan Nash. 2015. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458* (2015).

[16] Nagababu Pachhala, S. Jothilakshmi, and Bhanu Prakash Battula. 2021. A Comprehensive Survey on Identification of Malware Types and Malware Classification Using Machine Learning Techniques. In *2021 2nd International Conference on Smart Electronics and Communication (ICOSEC)*. 1207–1214. https://doi.org/10.1109/ICOSEC51865.2021.9591763

[17] Eva Papadogiannaki, Giorgos Tsirantonakis, and Sotiris Ioannidis. 2022. Network Intrusion Detection in Encrypted Traffic. In *2022 IEEE Conference on Dependable and Secure Computing (DSC)*. 1–8. https://doi.org/10.1109/DSC54232.2022.9888942

[18] David M. W. Powers. 2020. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *CoRR* abs/2010.16061 (2020). arXiv:2010.16061 https://arxiv.org/abs/2010.16061

[19] Hasim Sak, Andrew W. Senior, and Françoise Beaufays. 2014. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *INTERSPEECH*. 338–342.

[20] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. 2015. Blindbox: Deep packet inspection over encrypted traffic. In *Proceedings of the 2015 ACM conference on special interest group on data communication*. 213–226.

[21] Zihao Wang, Kar Wai Fok, and Vrizlynn LL Thing. 2022. Machine learning for encrypted malicious traffic detection: Approaches, datasets and comparative study. *Computers & Security* 113 (2022), 102542.

[22] Yi Zeng, Huaxi Gu, Wenting Wei, and Yantao Guo. 2019. $Deep-Full-Range$: A Deep Learning Based Network Encrypted Traffic Classification and Intrusion Detection Framework. *IEEE Access* 7 (2019), 45182–45190. https://doi.org/10.1109/ACCESS.2019.2908225

[23] Weiping Zheng, Jianhao Zhong, Qizhi Zhang, and Gansen Zhao. 2022. MTT: an efficient model for encrypted network traffic classification using multi-task transformer. *Applied Intelligence* 52, 9 (2022), 10741–10756.

[24] Hanxun Zhou, Yeshuai Hu, Xinlin Yang, Hong Pan, Wei Guo, and Cliff C. Zou. 2020. A Worm Detection System Based on Deep Learning. *IEEE Access* 8 (2020), 205444–205454. https://doi.org/10.1109/ACCESS.2020.3023434