DL - An Article Implementation
Deep Packet: A Novel Approach For Encrypted Traffic Classification Using Deep Learning

Shohamd Danino 2042087635
Yehonatan Moshkovitz 314767674

final project report for the Deep Learning course, IDC
March 2021

1 Introduction

Network traffic classification is an important task in modern communication networks. Due to the rapid growth of high throughput traffic demands, to properly manage network resources, it is vital to recognize different types of applications utilizing network resources. Consequently, accurate traffic classification has become one of the prerequisites for advanced network management tasks such as providing appropriate Quality-of Service (QoS), anomaly detection, pricing, etc. Traffic classification has attracted a lot of interests in both academia and industrial activities related to network management. As an example of the importance of network traffic classification, one can think of the asymmetric architecture of today's network access links, which has been designed based on the assumption that clients download more than what they upload.
In this project we decided to focus on traffic classification, we find this subject very interesting because we both specialize in our work in network security and we meet this kind of problems on a daily basis. The specific problem we chose is packet classification to applications (Facebook, Skype etc). The approach we chose to classify the traffic is convolutional neural networks, which we learned in our course, and staked auto-encoder, which we didn't study in the course and we wanted to experiment with building this type of network.

1.1 Related Works

Network traffic classification is divided into three main methods: port-based, payload inspection and machine learning. Traffic classification via port number is the oldest and the most well-known method, this method uses extract the port number from the TCP/UDP headers which is assumed to be associated

with a particular application.
Port-based classification is known to be among the simplest and fastest method for network traffic identification. How ever, the pervasiveness of port obfuscation, network address translation (NAT), port forwarding, protocol embedding and random ports assignments have significantly reduced the accuracy of this approach.Payload inspection are based on the analysis of information available in the application layer payload of packets. Most of the payload inspection methods, use predefined patterns like regular expressions as signatures for each protocol. The derived patterns are then used to distinguish protocols form each other.
The need for updating patterns whenever a new protocol is released,and user privacy issues are among the most important drawbacks of this approach. Statistical and machine learning approach have a base biased assumption that the underlying traffic for each application has some statistical features which are almost unique to each application.Two of the most important papers that have been published on "ISCX VPN-nonVPN" traffic dataset are based on machine learning methods. the first one used time-related features such as the duration of the flow, flow bytes per second, forward and backward inter-arrival time, etc. to characterize the network traffic using k-nearest neighbor (kNN) and decision tree algorithms[1].
They achieved approximately 92% recall, characterizing six major classes of traffic including Web browsing, email, chat, streaming, file transfer and VoIP using the C4.5 algorithm. They also achieved approximately 88% recall using the C4.5 algorithm on the same dataset which is tunneled through VPN. the second manually selected 111 flow features and achieved 94% of accuracy for 14 class of applications using kNN algorithm [2]. The main drawback of all these approaches is that the feature extraction and feature selection phases are essentially done with the assistance of an expert. Hence, it makes these approaches time-consuming, expensive and prone to human mistakes. Moreover, note that for the case of using kNN classifiers it is known that, when used for prediction, the execution time of this algorithms is a major concern.

2 Solution

2.1 General approach

In this work, we developing a framework that comprises of two deep learning methods, namely, convolutional NN and stacked autoencoder NN, for both "application identification" and "traffic characterization" tasks. Before training the NNs, we perform a pre-processing phase on the dataset (detailed below). At

[1]Gil GD, Lashkari AH, Mamun M, Ghorbani AA (2016) Characterization of encrypted and vpn traffic using time-related features. In: Proceedings of the 2nd International Conference on Information Systems

[2]Yamansavascilar B, Guvensan MA, Yavuz AG, Karsligil ME (2017) Application identification via network traffic classification. In: Computing, Networking and Communications (ICNC), 2017 International Conference on, IEEE, pp 843-848

the test phase, a pre-trained neural network corresponding to the type of classification, application identification or traffic characterization, is used to predict the class of traffic the packet belongs to.

2.2 Data-set

we use "ISCX VPN-nonVPN" traffic dataset, that consists of captured traffic of different applications in pcap format files. In this dataset, the captured packets are separated into different pcap files labeled according to the application produced the packets (e.g., Skype, Hangouts, etc.) and the particular activity the application was engaged during the capture session (e.g., voice call, chat, file transfer, or video call).

The dataset also contains packets captured over Virtual Private Network (VPN) sessions. [3] Tunneling IP packets, guaranteeing secure remote access to servers and services, is the most prominent aspect of VPNs. Similar to regular (non-VPN) traffic, VPN traffic is captured for different applications, such as Skype, while performing different activities, like voice call, video call, and chat. Furthermore, this dataset contains captured traffic of Tor software. This traffic is presumably generated while using Tor browser [4] and it has labels such as Twitter, Google, Facebook, etc.

2.3 Pre-processing

Before training the NNs, we have to prepare the network traffic data so that it can be fed into NNs properly. The "ISCX VPN-nonVPN" dataset is captured at the data-link layer. Thus it includes the Ethernet header. The data-link header contains information regarding the physical link, such as Media Access Control (MAC) address, which is essential for forwarding the frames in the network, but it is uninformative for either the application identification or traffic characterization tasks.

Hence, in the pre-processing phase, the Ethernet header is removed first. Transport layer segments, specifically Transmission Control Protocol (TCP) or User Datagram Protocol (UDP), vary in header length. The former typically bears a header of 20 bytes length while the latter has an 8 bytes header. To make the transport layer segments uniform, we inject zeros to the end of UDP segment's headers to make them equal length with TCP headers. The packets are then transformed from bits to bytes which helps to reduce the input size of the NNs. Since the dataset is captured in a real-world emulation, it contains some irrelevant packets which are not of our interest and should be discarded. In particular, the dataset includes some TCP segments with either SYN, ACK, or FIN flags set to one and containing no payload. These segments are needed for three-way hand-shaking procedure while establishing a connection or finishing one, but they carry no information regarding the application generated them,

[3] A VPN is a private overlay network among distributed sites which operates by tunneling traffic over public communication networks (e.g., the Internet)

[4] Tor is a free, open source software developed for anonymous communications

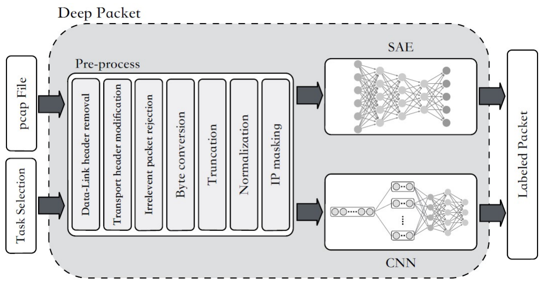

Figure 1: General illustration

thus can be discarded safely. Furthermore, there are some Domain Name Service (DNS) segments in the dataset. These segments are used for hostname resolution, namely, translating URLs to IP addresses. These segments are not relevant to either application identification or traffic characterization, hence can be omitted from the dataset. packet length varies a lot through the dataset, while employing NNs necessitates using a fixed-size input. Hence, truncation at a fixed length or zero-padding are required inevitably.
To find the fixed length for truncation, we inspected the packets length's statistics. Our investigation revealed that approximately 96% of packets have a payload length of less than 1480 bytes. This observation is not far from our expectation, as most of the computer networks are constrained by Maximum Transmission Unit (MTU) size of 1500 bytes. Hence, we keep the IP header and the first 1480 bytes of each IP packet which results in a 1500 bytes vector as the input for our proposed NNs. Packets with IP payload less than 1480 bytes are zero-padded at the end. To obtain a better performance, all the packet bytes are divided by 255, the maximum value for a byte, so that all the input values are in the range [0 to 1].
Furthermore, since there is the possibility that the NN attempts to learn classifying the packets using their IP addresses, as the dataset is captured using a limited number of hosts and servers, we decided to prevent this over-fitting by masking the IP addresses in the IP header. In this matter, we assure that the NN is not using irrelevant features to perform classification.

3 Design

Provide some general information about your code, platform, how long it took you to train it, technical challenges you had, etc. You are encouraged to use figures and tables as much as possible.
Our code provides two separate ways to predict what is the application and witch type is the traffic in a single packet. after the prepossessing, we used 2 NN's - the CNN designed especially for this problem, and Stacked Auto-Encoder.
The CNN designed especially for this problem is a minimal illustration of the

second proposed scheme, based on one-dimensional CNN. The model consists of two consecutive convolutional layers, followed by a pooling layer. Then the two-dimensional tensor is squashed into a one-dimensional vector, and fed into a three-layered network of fully connected neurons which also employ dropout technique to avoid over-fitting.
The stacked auto-encoder is built of 4 hidden layers. we take the 1500 features we got out of the reprocessing procedure, encode them to 300 features, and after that to 60 features. after this, we decode then back to 300 and 1500 features and than we compress them into the number of classes we want to divide to - 17 in application case and 12 in traffic case. We use linear transformations with the activation function ReLU for the decode and the encode. We have used Cross Entropy Loss for the training module. With this loss function, we use softmax, like the article describes.
It is important to mention that we have used our networks on a smaller data set because we had less computing power and we anticipated our result to not be as good as the article presents - we have used 8 percent of the training and the testing data. On the Application classification problem, the training time of the CNN designed especially for this problem was 290 seconds, and the training time of the Stacked Auto-Encoder was 16 seconds. On the Traffic classification problem, the training time of the CNN designed especially for this problem was 375 seconds, and the training time of the Stacked Auto-Encoder was 25 seconds. Overall, the Stacked Auto-Encoder was over 15 times faster than the CNN.

4 Experimental results

In our experiment we successfully classified the Traffic and the Application of the packets with success rate of over 90% in both Networks, and with both problems (Traffic and Application).
As we anticipated, our results were a bit worse than the results the article presents due to the fact we have used smaller data set. In the article they describe 98% accuracy in the Application classify problem and 94 in the Traffic classify problem with the CNN that was built for this problem; and 96% accuracy in the Application classify problem and 92 in the Traffic classify problem with the stacked auto-encoder.
In our results we have reached an accuracy of 91% in the Application classify problem and an accuracy of 92% in the Traffic classify problem with the CNN that was built for this problem; and we have reached an accuracy of 94% in the Application classify problem and an accuracy of 95% in the Traffic classify problem with stacked auto-encoder.

There ate two main things that draw our eyes about this results:
1. We got better results with the Traffic classify problem, when in the article they showed better results for the Application classify problem.
2. We got better result with the Stacked Auto-Encoder, compared to the CNN, when in the article they got better results with the CNN.

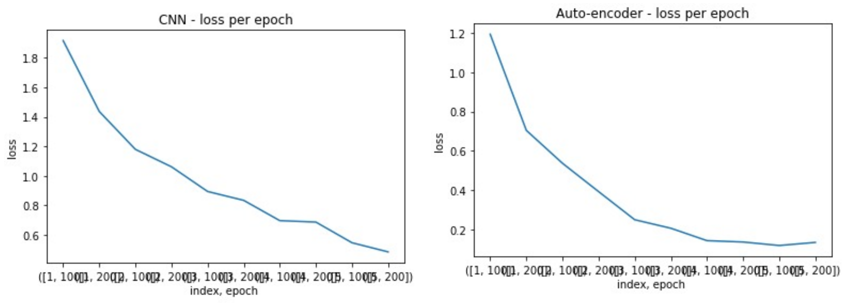

Figure 2: Application classification loss

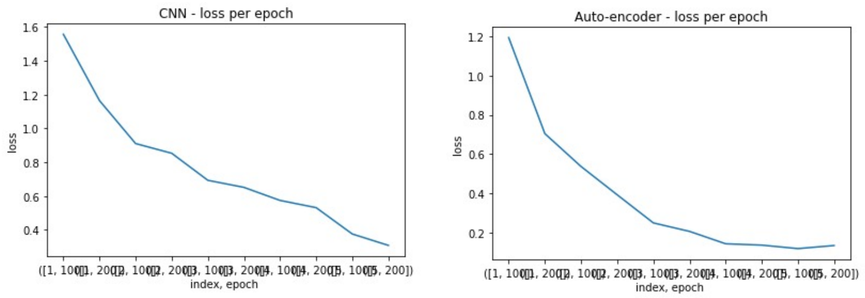

Figure 3: Traffic classification loss

We have to remember that we used a sample of the whole data, and that this sample can be not corresponding with the original problem. One thing that can make sense is that we have achieved better result in a problem of classify less categories (12 and not 17) on a smaller data set. Since we have only 12 categories in the traffic problem, we needed less data to appropriately classify and select the right category. We do not have a specific simple explanation for the fact we had better results in the Stacked Auto-Encoder, and we will touch this matter in the discussion part.

5 Discussion

Working on the packet classification problem was not an easy task. We had to understand and recreate the pre-processing procedure on the packets, and than to create two different NN's. The CNN that was built for this problem and described in the article (we had to understand the architecture and implement it ourselves), and we had to create a generic Stacked Auto-Encoder, based on the picture and few wards in the article, and further reading on this topic. We have accomplish to create a CNN that got us great results that was close to what the article presents, even though we have used a smaller data set.

The first thing that we can safely say from our research is that the Stacked Auto-Encoder architecture will provide faster solution for both classification problems (in our code the training was more than 15 times faster); and If we want fast training (for some online use or other reasons) we should use the SAE and not the CNN.
Moreover, we think that even with the limitation of the smaller data set, we can have an interesting hypothesis and suggest a further research in this topic. It is possible that in smaller data sets, the Stacked Auto-Encoder is a better way to classify the Application and Traffic of packets than the CNN that is built especially for this problem and that presented in the article. We can see in the loss functions (on figures 2,3), that the SAE loss in both problems is already asymptotic, and in the last epoch the loss did not improved, but the CNN loss was still improving even at end of the fifth epoch. We can assume from this that the SAE learn faster than the CNN. We recommend to check this hypothesis, by checking this two networks on several times on a smaller data set (similar to the one we have used witch was 8% of the size that the writers of the article have used).

6 Code

https://colab.research.google.com/drive/1e$_xJo_I3j-0jt_JhH9IaKzcnF3RTdVW7?usp = sharing$