# Traffic Classification Using Deep Learning with Sampled Packets

Gil Knafo 318590593
Shoham Danino 204287635
Aviv Pelach 208387407

Submitted as final project report for Data Streaming Algorithms course, IDC, 2020

## 1 Introduction

In this project we decided to focus on traffic classification, because we think that network traffic is a classic example for data streams, and we find this subject very interesting. The specific problem we chose is packet classification to applications (Facebook, Skype etc). The approach we chose to classify the traffic is deep learning, and especially convolutional neural networks.
The packets in our data-set are very big, and every stream consists a large number of packets, so the training of the network and the classification may be time and space consuming. In our project, we are intend to examine the effect of packet sampling and dimensionality reduction on the accuracy of the prediction, and by that, reduce time and space complexity.

### 1.1 Related Works

Traffic classification is a known problem that has many forms. For example: malware detection, app classification, traffic type classification etc. As mentioned above we chose to focus on associating network flows with the application that generated them. Traffic classification with deep learning is being used in the last few years. Especially, it has been used to detect malwares[3] and to classify traffic[2].

In all of the known approaches to classify traffic with deep learning that we found in our literature review the models predict the labels based on the full packet. But, at the same time, other literature addressed the issue of packet sampling as an effective method to network monitoring[5].

In this project we are about to combine these two approaches in order to classify traffic faster and with less space requirements, based on sampled packets.

# 2 Solution

## 2.1 General approach

In this project we'll examine two sampling approaches:

1. Systematic sampling - packets are sampled in a deterministic fashion by selecting the first x bits in the packet, or every other (1500/x)th bits in the packet.

2. Random sampling - packets are sampled at random (reservoir sampling). Each bit is sampled independently at a rate p = 1/x.

(x is the length of the wanted sample).

Those two approaches will be measured in relation to the original implementation of the article (each packet is 1500 bits length - trimmed or padded with zeros).

For each approach, we will evaluate the neural network's performance for various number of x's (ratio). It should be noted, that the network will be evaluated according to accuracy ratio: (true labeled/all) * 100.

In addition to sampling approaches, we will also examine the performances of the neural network on packets with less dimensions (after applying dimensionality reduction algorithms).

## 2.2 Design

Our code is written in python and we used Google Colab platform for an efficient GPU-based implementation. We also used PyCharm in several parts of the data preprocessing.
Our code includes 4 major parts: data preprocessing, data sampling, dimensionality reduction and neural network experiments.

### 2.2.1 Preprocessing

Before training the neural network, we had to prepare the network traffic data so that it can be fed into the network properly. In the pre-processing phase, we performed the following:

1. Removed Ethernet header.

2. Padding traffic with UDP header with zeros to the length of 20 bytes.

3. Masked the IP in the IP header.

4. Removed irrelevant packets such as packets with no payload or DNS packets.

5. Converted the raw packet into a bytes vector.
6. Truncated the vector of size more than 1500, pad zeros for the
7. byte vector less than 1500.
8. Normalised the bytes vector by dividing each element by 255.

In our work we assure that the NN is not using irrelevant features to perform classification. This implementation is based on online implementation[4] based on DeepPacket article[2].

#### 2.2.2 Sampling

Implementation of the sampling methods we explored above:

1. Original function (from the original article[2]) - use the full packet (1500 bits). If the packet is a bit longer we'll trim it, and if it's shorter we'll pad it with zeros. Note - this function is implemented within the preprocessing part.
2. Trimming function - cut the end of the packet. This function gets the desired length of the sample as input.
3. 'Every other x' function - chooses every other x-th (as input) elements of the packet (e.g. for x=2, the function will choose only even indexed elements).
4. Reservoir sampling function - random sampling. This function gets the desired length of the sample as input.

| Sampling Method | Systematic/Random | Packet's Used elements | Length |
|---|---|---|---|
| **original** | systematic | start/all | 1500 |
| **trimming** | systematic | start | x |
| **'every other'** | systematic | all | x |
| **reservoir** | random | all | x |

Table 1: Sampling Methods Comparison

#### 2.2.3 Dimensionality Reduction

We have also explored incorporating dimensionality reduction process with our dataset before training. As we have discussed in the lectures, dimensionality reduction is an important and relevant tool for using less storage and tackle the "curse of dimensionality" while still get good results with our predictive model. The two approaches we have decided to implement and explore were Johnson-Lindenstrauss and Principal Components Analysis.

#### 2.2.4 Neural Network

Our neural network is based on Lotfollahi's, Siavoshani's, Hossein Zade's and Saberian's architecture [2] and published online implementations[4].
This network contains two 1-dimensional convolution layers followed by dropout (rate = 0.05) average pooling (kernel size = 2) and Relu activation function; and five fully connected linear layers followed by dropout (rate = 0.05) and Relu activation function. The final layer is like a softmax classifier with 10 neurons (number of app labels).

This part of the project took us long than we expected. Our original plan was to use online published implementation of the original article, and then focus on the sampling stage and on the parts of the project that are in the scope of our course. But we found out it wasn't as easy as we thought at first, because we had to adjust the neural network to our dataset and make it work successfully, and that was not an easy task. We struggled with this part for a long time and we didn't manage to get to everything we have planned. In result, we chose to focus only on packets sampling and dimensionality reduction (rather stream sampling etc.).

# 3 Experimental results

## 3.1 Dataset

We used "ISCX VPN-nonVPN" traffic dataset (link), that consists of captured traffic of different applications (e.g., Skype, Facebook, etc.) in pcap format files[1], the same dataset that was used in DeepPacket article[2]. In this dataset, the packets are seperated into different pcap files labeled according to the application generated the packets, 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.[1] 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[2] and it has labels such as Twitter, Google, Facebook, etc. In our project, we relates only to the application label.

[1] A VPN is a private overlay network among distributed sites which operates by tunneling traffic over public communication networks (e.g., the Internet)
[2] Tor is a free, open source software developed for anonymous communications

This dataset is very large, and therefore the training time of the network was too long (and RAM consuming). Out solution was to randomly sample the dataset and work with the smaller sampled dataset.

## 3.2 Sampling Experiments

As mentioned earlier, we examined each sampling method (4 overall - original; trimming; 'every other'; random) with various packet's length (see below), in order to explore the sample ratio effect on the accuracy of the model. We also compared the training time of the network, since one of the advantages of the sampling is time complexity reduction.

We chose 6 sample ratio: 1% of the original packet (x=15); 3% (x=45); 5% (x=75); 10% (x=150); 20% (x=300); 50% (x=750).

The model was trained (for 5 epochs) and evaluated against the independent test set (random 20% of the data) that was extracted from the dataset. The test set was the same within all experiments in order to create fair comparison. The evaluation is based on accuracy metric ((correct classified packets/all classified packets)*100).

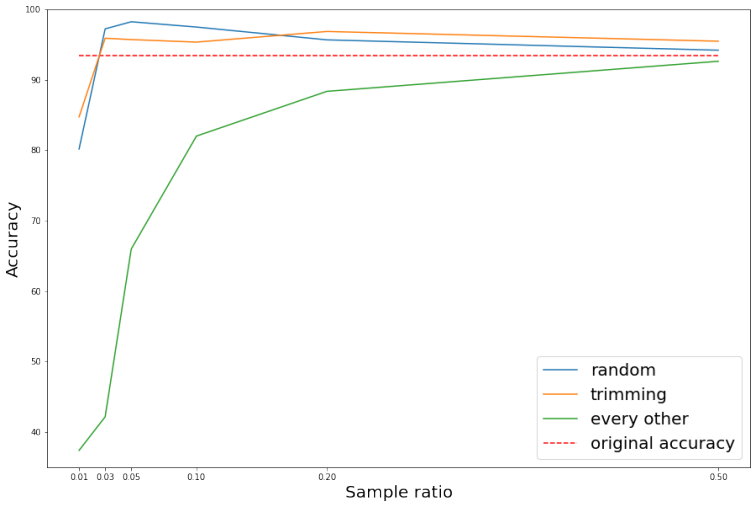


Figure 1: Sample ratio effect on accuracy

As we can see, we have improved the accuracy of the original algorithm (from the article) and our sampled packets leads to better accuracy than original packets (most of the time)!

Two main insights:

1. It seems like the significant information of the packet is at it's beginning, as we can see from the good performance of the trimming.
2. We assume that in general, random sampling is better than systematic, as we can conclude from a comparison between random and 'every other'. Trimming is a special case, further to 1. In addition, as the sample ratio decreases, random sampling performs better also than trimming.

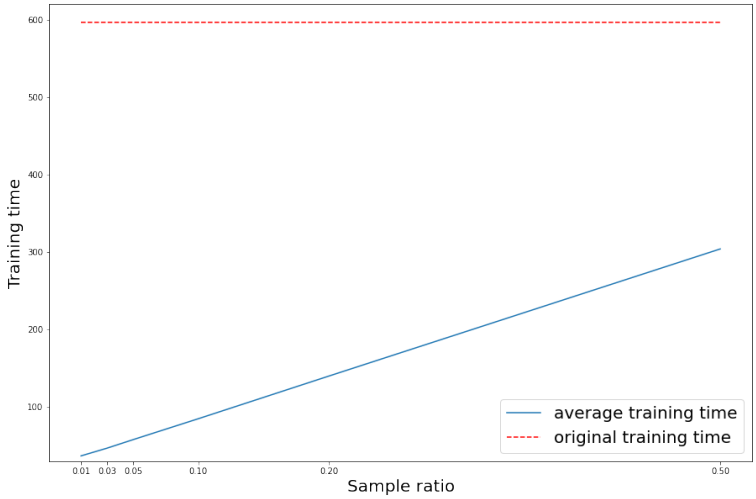


Figure 2: Sample ratio effect on training time

Training time is a major aspect when dealing with data streams. As expected, the smaller the sample, the faster the training time (linear relation).

### 3.3 Dimensionality Reduction

We used two dimensionality reduction algorithms: JL and PCA.
First, we used Johnson-Lindenstrauss method that preserve the Euclidean distances between data points using random projections. We chose 7 different epsilon-values and calculated the number of dimensions required to preserve the Euclidean distances (with error of delta). Then, we used random projections as described in JL lemma to reduce our 1500 dimensions dataset to the relevant number of dimensions.

In addition, we have explored the performance of the model under dimensionality reduction with Principal Components Analysis. We chose the same number of dimensions (principal components) as in Johnson-Lindenstrauss approach in order to observe the differences.

| | Opt.1 | Opt.2 | Opt.3 | Opt.4 | Opt.5 | Opt.6 | Opt.7 |
|---|---|---|---|---|---|---|---|
| **Epsilon** | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
| **Dimensions** | 180 | 195 | 223 | 270 | 351 | 498 | 812 |

Table 2: Epsilon-Dimensions Correlations From 1500 Original Features Using Johnson-Lindenstrauss Approach

We can see in figure 3 below that with Johnson-Lindenstrauss, using more dimensions most of the time help the model to train better and to predict better, however the accuracy results are not as good as the original or sampled data. In contrast, we can notice that when using PCA, adding more components actually decreases performance.

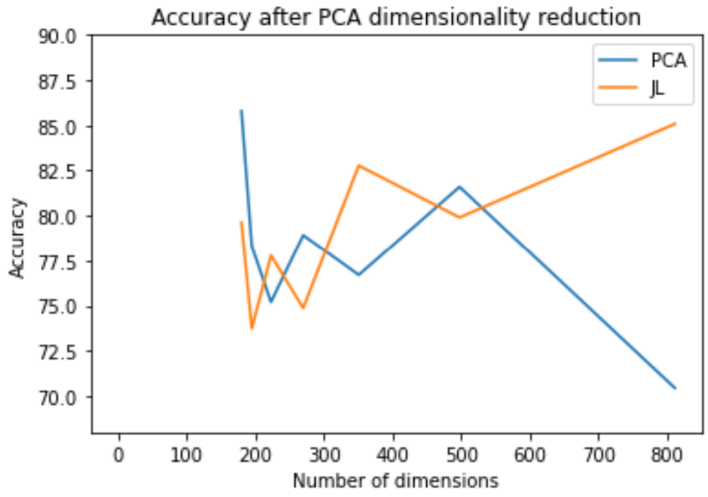


Figure 3: PCA vs JL dimensionality reduction

# 4 Discussion

The main insight of this project, is that packet classification using deep learning is not harmed, and even improves, when using sampled packets. We were surprised to find out that using sampled packets improve the method presented in the original article; especially random sampling, but even the trimming method.

When we look at the reasons for that we assume that the packets generated by same applications are very similar to each other at the beginning (the headers) but immediately afterwards in the fourth layer (after the IP) and in the data section they are different, what "confuses" the network. We assume that

when using full size packets the network overfit the train set, what leads to lower accuracy on the test set. Moreover, most of the packets in the data-set were smaller than 1500 and we (based on original article) padded them with zeros to create a uniform size, because the CNN requires a fixed size input. That is another reason for the good results of the trimming method (especially for a higher sample ratio.)

In addition, we found out that dimensionality reduction might also be a relevant tool for packet classification. The number of dimensions required is important to determine the technique for reducing the dimensions. However, dimensionality reduction didn't achieve results as good as the original or sampled data.

The compressing method (sampling or dimensionality reduction) and the ratio (sample ratio or number of dimensions) should be determined, in our opinion, mainly by accuracy (which can be different for every dataset), but training time should also be taken into account when dealing with data streams.

**Future work** - We explored the effect of packet sampling on the accuracy with specific neural network and specific data-set. We assume that architecture tuning of the neural network (or even different networks for different data sets and sample ratios) can improve the performance. In addition, we recommend trying to test our method using a network that does not require a fixed input size, like RNN and not CNN as we examined in the article. Moreover, we assume that the results will be similar with other packet data-sets (because of the similar structure of packets), but it can still be interesting to explore this.

# 5 Code

Preprocessing notebook is here
Experiments' notebook (sampling, dimensionality reduction and NN) is here

# References


[1] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Mamun, and Ali Ghorbani. Characterization of encrypted and vpn traffic using time-related features. 02 2016.

[2] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning, 07 2018.

[3] Gonzalo Marín, Pedro Casas, and Germán Capdehourat. Deepmal – deep learning models for malware traffic detection and classification, 03 2020.

[4] MunHou. Pytorch implementation of deep packet: A novel approach for encrypted traffic classification using deep learning.

[5] Davide Tammaro, Silvio Valenti, Dario Rossi, and Antonio Pescapè. Exploiting packet-sampling measurements for traffic characterization and classification. *International Journal of Network Management*, 22, 11 2012.