# Website Fingerprinting and Traffic Labeling with Deep Neural Networks

Till Beemelmanns[1] and Adam Weckle[2]

***Abstract*— Our paper proposes that deep learning methods can be applied to differentiating between different types of traffic and file transactions. We utilize both a CNN and a Bidirectional-LSTM neural network architecture to test this hypothesis. We also make a comparison with a Bidirectional LSTM to a CNN with website fingerprinting.**

## I. INTRODUCTION

In our project we will investigate the capabilities of Deep Neural Networks for Website Fingerprinting and traffic labeling based on The Onion Router (Tor) network traffic. Users of the Tor network expect total anonymity while they are visiting websites and using online services. However, various researchers have proven that the network traffic that is exchanged between the user and the network can be used for Website Fingerprinting (WF) attacks. A possible eavesdropper between host and Tor entry node could capture encrypted routing information and the specific traffic pattern. The timing and the sizes of network packages can create a unique fingerprint for a specific website or set of websites. A possible attacker could use this pattern in order to determine which website Tor users are accessing. This method of attack was presented in various studies [1]–[5]. Based on these works, we want to improve the attack mechanism by using a different deep learning model and by predicting what type of file transaction the user is performing on these websites. This result is predicted to be more widely applicable to open-world applications.

[1]T. Beemelmanns is with the Department of Computational Engineering Science, RWTH Aachen University, Germany till.beemelmanns@rwth-aachen.de

[2]A. Weckle is with the Department of Engineering and Computer Science, Michigan State University, USA wecklead@msu.edu

## II. PROJECT MOTIVATION

### A. OUR APPROACH

Our proposed approach to this problem is to utilize the ideas demonstrated by Rimmer et al [6], and use both their tested models and a Bidirectional LSTM to analyze network traffic. However, rather than trying to separate data into the specific websites they could be from, we will simply try to determine what type of file transaction the user may be involved in at the moment. We expect this to have more significant open-world applications since it doesn't rely as much on getting previous traffic traces from a chosen website. In the absence of an easy way to collect this data, it was all done manually with a small dataset and may not actually be representative of an open-world scenario.

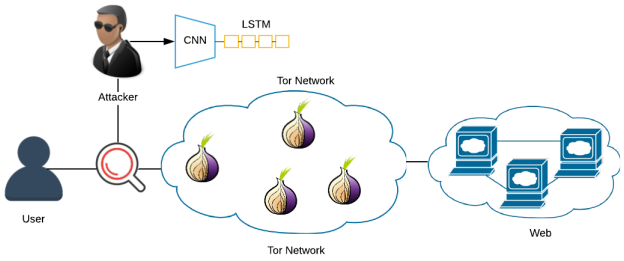


Fig. 1. The user utilizes a Tor network to access a website. The listener observes the encrypted messages between the user and Tor network, and uses Deep Learning to analyze it.

This supervised learning technique proposes a Bi-directional network that takes traffic sequences as an input and is trained to map the input to the desired output labels. Since [6] has shown that it is possible to process encrypted traffic streams with CNNs, although CNNs were designed to process image data, we belief that it is possible to train and evaluate this data on an advanced BI-LSTM architecture. This enables us to repeat this process to discriminate between different types of file transactions, including: compressed file upload, video

streaming, standard browsing/searching and messaging. The distinction between these transaction types can be important for realizing if a website guess is accurate or for open-world scenarios where a guess is less plausible. Additional motivations can include tracking of illegal uploads/downloads and criminal investigations.

## III. BACKGROUND SURVEY

### A. MACHINE LEARNING APPROACHES

Several studies have shown that Website Fingerprinting attacks on Tor are feasible with various Machine Learning and Deep Learning algorithms.

The first approaches to analyzing Tor network traffic were data mining algorithms presented by Herrmann et al [7]. The authors of this publication have shown that their model was able to determine which websites were accessed by the user. They deployed a Naive Bayes classifier along with a word frequency feature. This approach achieved an average accuracy of 3%.

A more sophisticated approach was proposed by Panchenko et al. [5]. Additional handcrafted traffic features such as bytes/packets transmitted, along with the use of Support Vector Machines. This approach resulted in an accuracy of about 50%.

Substantial improvements on this classification problem have been archived by Cai et al [8]. These researchers proposed a SVM with a custom handcrafted kernel function based on the Damerau-Levenshtein distance which measures additional drops and retransmissions of TCP packets. Cai et al. claimed that they reached an accuracy of 86% fingerprinting Tor traffic and an accuracy of more than 91% for SSH tunnel traffic. However, their approach was later criticized in [9] for using a strict closed world and unrealistic ideal conditions and using small datasets. Further important factors such as multi-tab browsing, internet connection and page load interruptions were not considered.

The following three works were recently published and use commonly generic classifiers in combination with handcrafted feature extractors.

Wang-kNN [10] is an attack with 90-95% accuracy, based on the k-Nearest Neighbors classifier. It has nearly 4000 traffic features, a majority of which are from packet lengths.

CUMUL [4] is an attack with 90-93% accuracy, based on an SVM with a Radial Basis Function kernel. It has 104 features derived from packet lengths and incoming packet patterns.

k-Fingerprinting [3] is an attack with similar accuracy to CUMUL, based on Random Forests. It has 175 features, including timing features like packets per second.

### B. DEEP LEARNING APPROACHES

With the recent boom of Deep Learning (DL) algorithms in the past 5 years, the traditional approach (cf. [3]–[5], [7], [9], [10]) using manual feature engineering along with generic, partially customized classifiers has become less important. Deep learning algorithms such as Feed Forward Neural Networks (FFNs), Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are designed to jointly train feature extraction and classification, which can lead to a more powerful overall design. Hence, profound manual engineering of features is in most cases not necessary any more, but dataset size, dataset quality, model training and hyperparameter tuning are the new challenges for a successful DL approach.

There is some previous research that applied DL to WF or similar problems [1], [2], [6]. We will now introduce and discuss these publications.

A similar problem was tackled by Wang [2]. The researcher was able to classify protocols and application traffic using a Stacked Auto-Encoder (SAE), essentially equivalent to a multilayer perceptron. This is particularly interesting for the application of discovering whether the user is uploading or downloading a specific type of data to a website. The author of this publication claimed that they were able to determine the traffic of e.g. Gmail, BitTorrent or Apple's iTune. Hence, it seems possible to use DL for traffic identification. However, their data analysis was not performed on encrypted data streams and they used a relatively simple neural network architecture.

Abe and Goto [1] improved the model of [2] by developing a Stacked Denoising Autoencoder (SDAE). The SDAE employs a denoising regularization technique that improved the model's prediction robustness. Further, the researchers used a bigger dataset compared to the previous approach. As in the previous paper, the researcher used a comparable small neural network and they did not investigate upstream and downstream behavior.

Rimmer et. al. [6] applied the approaches of Stacked Denoising Autoencoder (SDAE), CNN, and

LSTM to the problem of website fingerprinting and found they matched or surpassed the most recent machine learning approaches. In particular, they use large scale datasets with varying sizes and captured under dynamic changes of web content over time. They have proved that the size of the datasets is essential for the net's performance. It was stated that SDAE, CNN and LSTM have a comparable performance, but each of them have strengths and weaknesses regarding WF. While CNN is the fasted network it has a higher risk of overfitting. LSTMs have the better generalization capabilities but they are constrained to shorter traffic sequences due to backpropagation problems. It was not investigated whether it is possible to used a bi-directional LSTM model consisting of bi-directional RNN layer. We belief that this could improve the model's performance as TCP traffic has a highly complex structure where packages depend highly on past and future messages. As in the previous papers, the user's behavior on the various websites was not investigated. We will use this publication as the main basis for our research. In particular we are going to use the published benchmark datasets and the preprocessing methods as posted on https://github.com/DistriNet/DLWF.

## IV. ARTIFICIAL NEURAL NETWORKS

Artificial Neural Networks (ANNs) are a class of nature-inspired computational systems. An ANN consist of a large collection of *artificial neurons* that are connected with each other through weighted directed edges that imitate the *synapses* and *neurons* of a biological brain [11]. The weights of the edges represent the strength of the synapses between the neurons [11]. The computational complexity and the memory storage of a single neuron is limited. However, the connection of hundred thousand of artificial neurons arranged in a network compound can achieve remarkable artificial intelligence like performance.

Figure 2 depicts such an artificial neuron.

Training of a big network is done by backpropagation and a stochastic gradient decent algorithm. A *loss function* $\mathcal{L}(\hat{y}, y)$ defines net's error of the current prediction and this error is consequently backpropagated to every single weight of the ANN. On this basis, an optimization gradient decent algorithm is used in order to manipulate the weights in such a way that error of the ANN with respect to the error function is minimized.

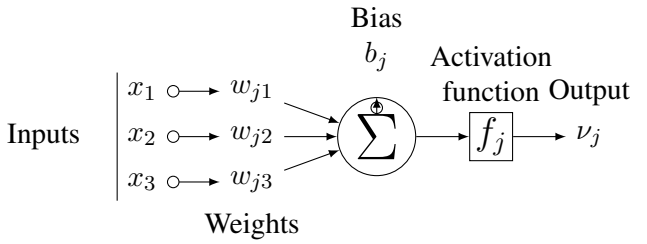


Fig. 2. The output of the artificial neuron $j$ is a weighted sum of its inputs which is put through an activation function. Figure adapted from [12].

There exists a variety of possibilities to arrange and structure artificial neurons and the information flow between them. In this way it is possible to differentiate between several types of ANNs. These types have typically strong different properties and show their best performance in a specialized task. For example, *Convolutional Neural Networks* (CNNs) achieved state of the art performance in computer visions tasks [13], where *Recurrent Neural Networks* (RNNs) were able to model time series data seen in natural language processing [14]. In this thesis, both last mentioned ANNs are being applied on the learning problem. In the following, both types will be introduced briefly.

### A. Convolutional Neural Networks

Convolutional Neural Networks are a special subversion of Feed Forward Neural Networks (FFNs). That means that the neurons are ordered in vertical, straight, forward layers where each unit of one layer is connected to every node in the following layer. This class of architecture uses special layers that compute convolutions over the input tensor. That allows the net to extract *local features* [6] from the input and create out of those features a generic classifier. The reception field is usually very small in comparison to the input dimension but several stacked convolutional layers in combination with non-linear activation functions can lead to a very powerful image classifier. Pooling layers between the convolutions allows the model to be persistent against spatial shifts. The last layers of a CNN classifier are usually fully connected dense hidden-layers. These allow the model to combine different extracted features into a classification using a softmax output layer.

### B. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a subclass of Deep Neural Networks (DNNs). RNNs have the unique feature that edges transfer data from the previous time step to the current state. These loops, called recurrent edges, allow the information to persist over time and enable complex, time-dependent sequence modelling. In contrast to Feed Foward Neural Networks (FFNs) RNNS do not require a fixed dimensionality of the input and the output.

Figure 3 visualizes a very simple recurrent cell. $x_t$ denotes the current input of that cell, $h_t$ is the hidden state at time step $t$ and $\hat{y}_t$ represents the output of the softmax classification function $\sigma$.

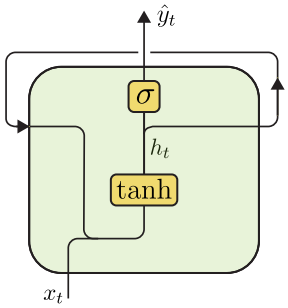


Fig. 3. Simple recurrent cell at time step $t$. Figure adapted from Olah [15].

Bidirectional Recurrent Neural Networks (BRNNs) extend a simple RNN by adding a second hidden layer that has also a recurrent connection but to the future time step. Such a cell is depicted in Figure 4. The BRNN cell a time $t$ receives information both from the past time step $t-1$ and the future one $t+1$. Hence, it is more likely that this architecture is able to learn complex pattern that rely on bidirectional temporal behavior such as in in Natural Language Processing. We assume that this methodology could also apply for TCP traffic. Payload Packets, ACKS and re-transmissions have a strong complex temporal behavior. Further, the timing of those packets play also an important role if we consider Website Fingerprinting over a Tor network.

Long Short Term Memory networks also called LSTMs where introduced by Hochreiter & Schmidhuber [17] in the late 90's and are known for their good performance in storing and remembering information for a long period of time outperforming standard RNN. In recent literature, LSTMs reached state-of-the-art performance in various sequence learning tasks. Up to date, the application of LSTMs in speech processing and natural language translation appears to be the most favoured research area [14] [18]. Further applications are chatbots [19] and image caption generation [20].

The mathematical representation of a usual LSTM Cell is defined by the Equations 1 - 6.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}C_{t-1} + b_i) \quad (1)$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}C_{t-1} + b_f) \quad (2)$$
$$\tilde{C}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (3)$$
$$C_t = f_t \otimes C_{t-1} + i_t \otimes \tilde{C}_t \quad (4)$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}C_t + b_o) \quad (5)$$
$$h_t = o_t \otimes \tanh(C_t). \quad (6)$$

Exactly these equations are explicitly depicted in Figure 5. This representation clearly visualizes how the information flow between the different time instances happens. $h_t$ is the current hidden state and $C_t$ denotes the cell state at time t. Thank to a multiplicative gates approach, information are able to persists over a long time without encountering the problem of vanishing gradients. A single LSTM cell consists of the following gates:

- Input gate $i_t$: Controls the input according to the current input $x_t$ and $h_{t-1}$. The resulting vector is added to the cell state.
- Forget gate $f_t$: Controls what information should be deleted from the cell state based on the fused information of input and hidden state.
- Output gate $o_t$: Regulates what is stored in the next cell state and future hidden state.

Overall, the LSTM is able to preserve gradient information over long sequences thank to the multiplicative gates approach. In this thesis we will use a LSTM network in a bidirectional fashion. In this case, all equations use both directions. That means that the cell at time step $t$ receives information both from the past time step $t-1$ and the future one $t+1$. This kind of approach should have better performance than just a recurrent approach especially for data characteristics with bidirectional behavior.

## V. DATASETS

### A. DATASET FOR WEBSITE FINGERPRINTING

For the evaluation of the new network architecture we used the proposed dataset in [6]. They are publicity

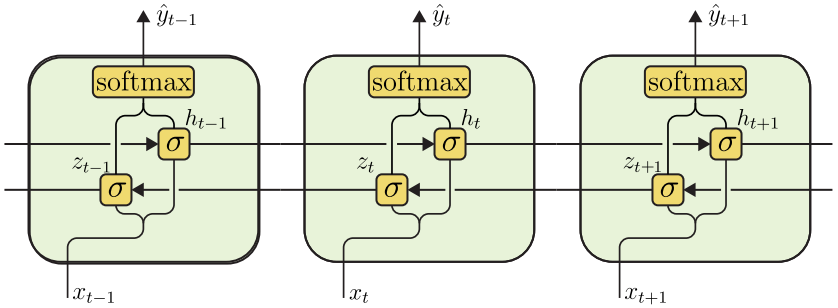


Fig. 4. Bidirectional recurrent neural network unfolded along time step $t-1$, $t$ and $t+1$. Figure adapted from Lipton et al. [16].

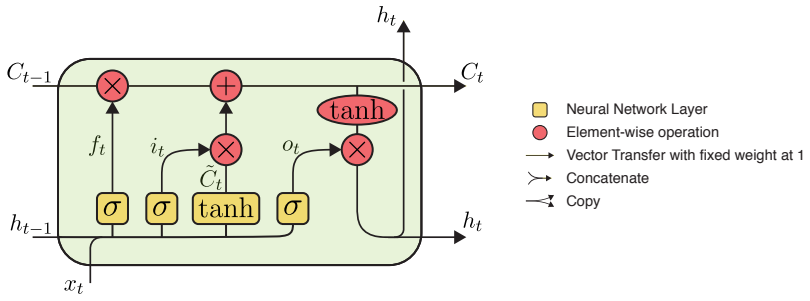


Fig. 5. Long Short Term Memory Cell with, explicit representation similar to Olah [15].

available on https://github.com/DistriNet/DLWF. The authors of [6] collected traffic traces of the 1,200 most popular websites according to a popular website ranking service. Hereby, each website was visited up to 3,000 times. Parallel virtual machines were used to gather this massive set of raw Tor traffic. All in all 3.6 million page visits have been performed. In this thesis only a small subset of this huge dataset could be investigated, due to limited computational resources. The closed world dataset $CW_{100}$ contains 2,500 traffic trace of the top 100 websites. Each traffic trace contains the first packages that were send in order to request the data from the specific website and the capturing finished 10 seconds after the website was successfully loaded. Further, this dataset is called *closed world* since it does not contain any noise traffic from other browsing or streaming activities. In an open world scenario it is more likely that a Tow user browses on several websites simultaneously and on an unpredictable range of different websites.

### B. DATA COLLECTION FOR TRAFFIC LABELING

We endeavored to collect data of five different types: *file upload*, *file download*, *video streaming*, *chatroom traffic*, and *general browsing*. All communications were carried out exclusively using Wi-Fi. To begin, we installed Tor Browser and Wireshark on both a Windows and a Mac operating system. We selected arbitrary files we had prepared beforehand and uploaded them to some standard file-sharing or storage websites, including vimeo.com, github.com, and onedrive.live.com. We also downloaded files from these websites. Video stream was performed by just watching random videos one after another. Chatroom traffic was manually simulated by the authors of this thesis. By doing so, random text messages and pictures were exchanged. We tried to simulate a chat conversation as realistic as possible. All of these use cases were conducted by using the encrypted and anonymized tor connection. The corresponding traffic was captured using Wireshark and the traffic was saved as `.pcap` files. We also tried to collect data for the use case *video chatting*, but the Tor browser prohibits any usage of cameras and microphones due to security reasons. In comparison of the datasets collected by [6], our dataset is much smaller and has less page visits or samples per category. Further, it is less strictly standardized as in [6], since the dataset was collected manually and not with a bot-like system. The used services were chosen upon personal favorites. Nevertheless, we believe that enough and high quality website traces could be captured so that this dataset qualifies for representative results.

In the following a overview over the different categories and the used services is given. We used a `trace length` of 250 to compute these values.

- **Video Streaming:** 1582 Traces
  - Youtube: 9 Sessions
  - Vimeo: 5 Sessions
- **Upload:** 1496 Traces
  - Vimeo: 2 Sessions
  - Github: 4 Sessions
  - Onedrive: 1 Session
- **Browsing:** 671 Traces
  - Google: 4 Sessions
  - DuckDuckGo: 4 Sessions
  - Bing: 3 Sessions
  - Yahoo: 3 Sessions
- **Messaging:** 335 Traces
  - Skype Online: 2 Sessions
  - Slack: 2 Sessions
  - Discord: 2 Sessions

The complete dataset measures about 1.3 Gigabytes.

## VI. Data Processing

In our manual data collection process, we used Wireshark which saves `pcap` and `pcapng` files containing all raw tcp packages. In order to extract the important information of the raw traffic data, we used the same data pipeline as by Rimmer et al. [6]. This pipeline removes any payload from the data since it is encrypted and does not have any value for the classification task. As described in [6], the extracted information are the time stamp, the direction and the size of each TCP package. Furthermore, Tor cells inside each TCP package is identified. The final extracted information, which is called *traffic trace*, is a sequence of negative ones and positive ones that denote incoming and outgoing Tor cells. Hereby $+1$ denotes outgoing traffic from the client to the Tor entry gate and $-1$ denotes incoming traffic. As in [6], we did not fix the Tor entry guard node, since we collected the data in several sessions using different WiFis. Further, the Tor network changed frequently by hopping servers. By doing so, we hope that our dataset contains more general traffic patterns relying on the different use cases and not on the specific service that was used. That means, that the attack scenario should work on any victim that has potentially different Tor entry nodes.

## VII. Training

### A. Compute Cluster

The training of the Artificial Neural Network requires the above mentioned large training data sets and optimization method that change thousands of weight simultaneously. Hence, a lot of computational resources were used during the development of this thesis. In particular, we used the HPCC GPU Cluster so we were able to perform the training and the evaluation of many different configurations simultaneously. The used nodes are *Nvidia K80 Kepler Cards* with each 256 Gigabytes of ram. We used the slurm batch system to submit and manage our GPU jobs. In order to use those resource, several bash scripts needed to be defined.

The framework that we extended rely on the top end deep learning library *Keras* [21] and we used *Tensorflow* [22] as backend. A virtual environment was set on the cluster in order to install all required Python libraries. Other libraries such as *CUDA*, *cuDNN* and *OpenMPI* could be loaded from the cluster.

Every architecture was trained with different hyperparameters. The usage of the Hyperparameters is defined in section VII-C.

### B. Model

The neural network architectures were trained in a supervised fashion. For each traffic trace we know the corresponding label. Given the input sequences and its labels we want to minimize the classification error of the model

$$E = -\frac{1}{N}\sum_{i}^{N}(p_i \log_2 p_i). \tag{7}$$

This function is denoted by the loss function of the model where $p_i$ is the probability for the predicted class i with N different classes. The weights of the neural networks were optimized in order to fit such classification and by using the `RMSprop` optimizer. The labels are encoded as an one-hot encoding scheme.

In order to simulate randomized traffic, we took the whole dataset and randomly shuffled all samples. Then 70% of the data were use to train the network.

15% of the data was used for validation and another 15% of the data was used to compute the final score of the model.

### C. HYPERPARAMETERS

Choosing the right hyper parameters is often based on trial and error and leads to very high computational costs. We tried to find the best parameters to fit the model to the data. Table 6 below gives an overview over the chosen parameters that we have used for the training of the specific models. The number of training epochs for the traffic labeling resulted from a sensitivity analysis.

| Hyperparameters | CNN-LSTM | BI-LSTM |
|---|---|---|
| Optimizer | RMSProp | RMSProp |
| learning rate | 0.001 | 0.001 |
| batch size | 16 | 16 |
| training epochs | 5 | 20 |
| layers | 7 | 2 |
| dropout | 0.25 | 0.2 |
| hidden units | 32 | 16 |

Fig. 6. Overview of the used training hyperparameters

For the Website Finger printing problem we used a trace length of 2500 using the CNN-LSTM model. For th BI-LSTM model we used a trace length of 150. As in [6], we could observe the problem that the LSTM is constrained in backpropagation. By that means, it was very difficult to train the BI-LSTM with much longer traffic traces without having a very slow convergence of the overall model.

## VIII. RESULTS

### A. WEBSITE FINGERPRINTING

Our BI-LSTM architecture was not significantly more useful than the CNN architecture. It took more time to run an epoch, even with fewer layers. It also was less accurate even when the same number of epochs was assumed. As shown in Figure 7 the performance of LSTM and BI-LSTM are quite similar on the $CW_{100}$ dataset. It is possible that some error occurred as a result of insufficient data, but even with double the epochs the LSTM architecture falls short.

| Architecture | CNN | LSTM | BI-LSTM |
|---|---|---|---|
| $CW_{100}$ | 96,26 % | 94,02 % | 94,28 % |
| Traffic Labeling | 86 % | - | 83,66 % |

Fig. 7. Test accuracy over the different datasets. The accuracies reflect the maximal achieved score using this specific model. The values for LSTM and CNN on $CW_{100}$ are taken from [6], but we could also achieve almost identical results while rerunning the mode.

### B. TRAFFIC LABELING

Our results as shown in Figure 8 indicate that a CNN continues to be the most accurate method of deep learning on traffic traces for our purposes, also maintaining the lowest loss. This is consistent with the previous study done by [6]. In addition, our results imply that we can estimate with over 80% accuracy which type of traffic a user is engaging in. Considering that we had a very small data set and only ran the CNN for 10 epochs, this measure of accuracy is very significant. Also consider the information in Figure 10. The precision for most transaction types is around 75%, and 95% in upload. The recall is excellent on video streaming and uploading but rather bad on messaging and browsing. The general reliability of determining whether the user is uploading a file or the user is viewing a video can thus be considered fairly accurate. Some of the traffic traces were uploading or streaming from the same websites. Therefore, this result is independent of the website indicating patterns presumably used in [6]. It is reasonable to predict that this measure of accuracy would apply to open-world scenarios as well. The traffic analysis is also predicted to get significantly more difficult if a user were to engage in accessing multiple websites at once.

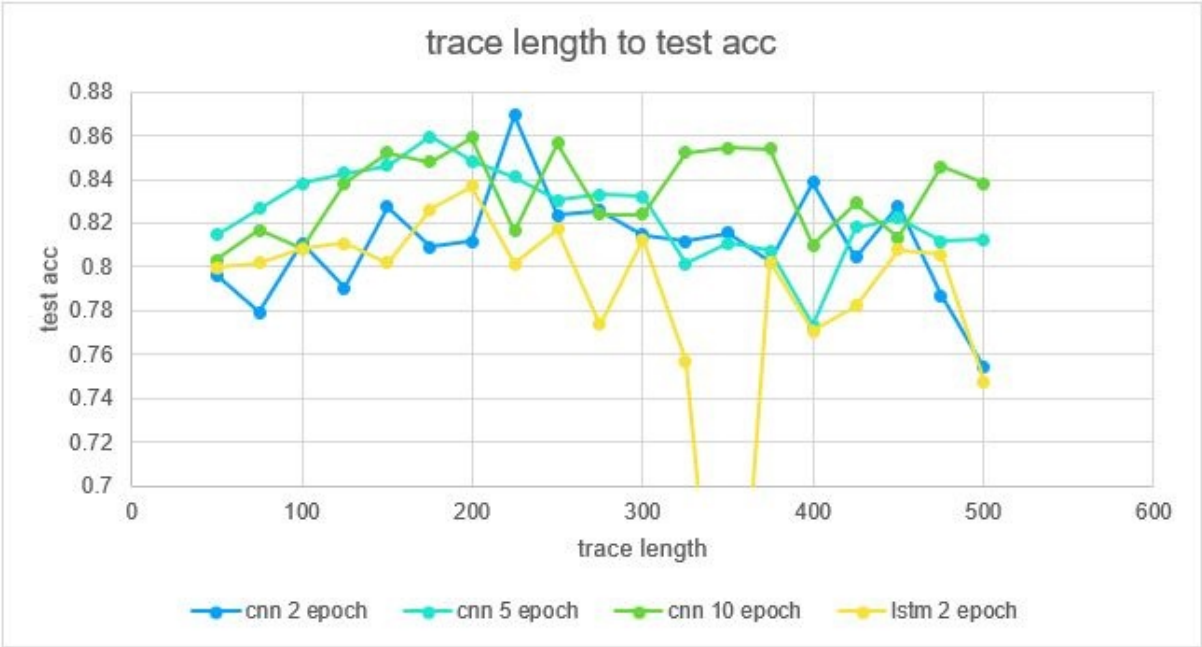


Fig. 8. Traffic trace length vs. test accuracy


Fig. 9. Traffic trace length to test loss

## IX. CONCLUSIONS

In this thesis, we analyzed a Tor network traffic with different neural network architectures in order to determine what type of file transaction the user is doing and what website they may be browsing. We extended Website Fingerprinting concepts from [6] in order to label traffic traces. We could successfully determine the type of traffic occurring using the same CNN-LSTM approach they used. This type of analysis allows a listener to estimate what type of transaction is occurring with high precision even if the website estimation has lower precision due to a realistic open-world scenario.

| | Precision | Recall | Support |
|---|---|---|---|
| Messaging | 0.74 | 0.44 | 52 |
| Video Streaming | 0.76 | 0.98 | 242 |
| Upload | 0.95 | 1.00 | 227 |
| Browsing | 0.75 | 0.26 | 92 |
| Micro Avg | 0.83 | 0.83 | 613 |

Fig. 10. The classification accuracy over the four different traffic classes.

For future work we propose the following ideas in order to refine the model or the extend the general approach.

- Use a bigger dataset and variety
- Add more use cases, E.g. distinguish between downloading and video streaming
- Add traffic noise, E.g. video streaming and parallel browsing

## REFERENCES


[1] K. Abe and S. Goto, "Fingerprinting attack on tor anonymity using deep learning," *Proceedings of the Asia-Pacific Advanced Network*, vol. 42, pp. 15–20, 2016.

[2] Z. Wang, "The applications of deep learning on traffic identification," *BlackHat USA*, 2015.

[3] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016, pp. 1187–1203. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/hayes

[4] A. Panchenko, F. Lanze, A. Zinnen, M. Henze, J. Pennekamp, K. Wehrle, and T. Engel, "Fingerprinting at internet scale," 2015.

[5] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, ser. WPES '11. New York, NY, USA: ACM, 2011, pp. 103–114. [Online]. Available: http://doi.acm.org/10.1145/2046556.2046570

[6] V. Rimmer, D. Preuveneers, M. Juárez, T. van Goethem, and W. Joosen, "Automated feature extraction for website fingerprinting through deep learning," *CoRR*, vol. abs/1708.06376, 2017. [Online]. Available: http://arxiv.org/abs/1708.06376

[7] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *CCSW*, 2009.

[8] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: Website fingerprinting attacks and defenses," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 605–616. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382260

[9] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt, "A critical evaluation of website fingerprinting attacks," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: ACM, 2014, pp. 263–274. [Online]. Available: http://doi.acm.org/10.1145/2660267.2660368

[10] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, 2014, pp. 143–157. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/wang_tao

[11] I. Basheer and M. Hajmeer, "Artificial neural networks: fundamentals, computing, design, and application," *Journal of microbiological methods*, vol. 43, no. 1, pp. 3–31, 2000.

[12] C. Gershenson, "Artificial neural networks for beginners," *CoRR*, vol. cs.NE/0308031, 2003. [Online]. Available: http://arxiv.org/abs/cs.NE/0308031

[13] P. Sermanet and Y. LeCun, "Traffic sign recognition with multi-scale convolutional networks," in *Neural Networks (IJCNN), The 2011 International Joint Conference on*. IEEE, 2011, pp. 2809–2813.

[14] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014. [Online]. Available: http://arxiv.org/abs/1409.3215

[15] C. Olah, "Understanding lstm networks," http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015, accessed 20-December-2016.
[16] Z. C. Lipton, "A critical review of recurrent neural networks for sequence learning," *CoRR*, vol. abs/1506.00019, 2015. [Online]. Available: http://arxiv.org/abs/1506.00019
[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8.
[18] A. Graves, "Generating sequences with recurrent neural networks," *CoRR*, vol. abs/1308.0850, 2013. [Online]. Available: http://arxiv.org/abs/1308.0850
[19] O. Vinyals and Q. V. Le, "A neural conversational model," *CoRR*, vol. abs/1506.05869, 2015. [Online]. Available: http://arxiv.org/abs/1506.05869
[20] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," *CoRR*, vol. abs/1411.4555, 2014. [Online]. Available: http://arxiv.org/abs/1411.4555
[21] F. Chollet *et al.*, "Keras," https://keras.io, 2015.
[22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

# APPENDIX

## A. CNN-LSTM Architecture

The CNN-LSTM Architecture is shown in figure 11. The convolutional layers have a kernel size of `5x5` and a stride of 1. Further, a `ReLu` activation function was used. A dropout of 0.25 was applied on the first convolutional layer. The Max-Pooling layers used a pool size of `4x4`. The LSTM layer had a unit size of 128. Finally the *softmax* classification layer had a output size of 100 for the fingerprinting classification and 4 for the traffic labeling scenario. All in all the architecture consists of 7 layers. All weights of these layers a trained together in a fused fashion.

### B. Bi-LSTM-Architecture

The Bi-Directional LSTM Architecture is show in Figure 12. It consists of two LSTM Layers; each of them has a bidirectional architecture and they both have 64 nodes. A `hard sigmoid` activation function inside these cells was deployed for a higher computational efficiency. Further both LSTM cell have a dropout coupled layer with a dropout probability of $p = 0.2$. The last layer is a softmax classification layer and it has according to the classification problem 100 output nodes, respectively 4 output nodes.

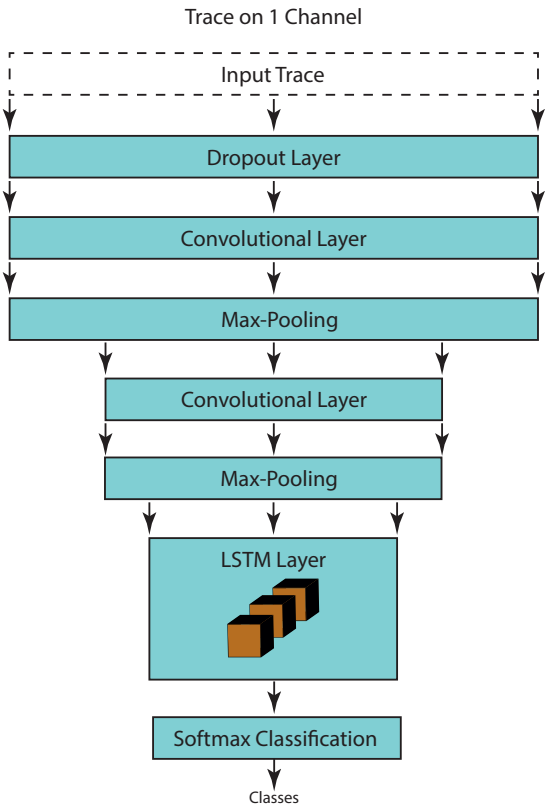


Fig. 11. The CNN-LSTM Architecture depicted as a stack of layers.

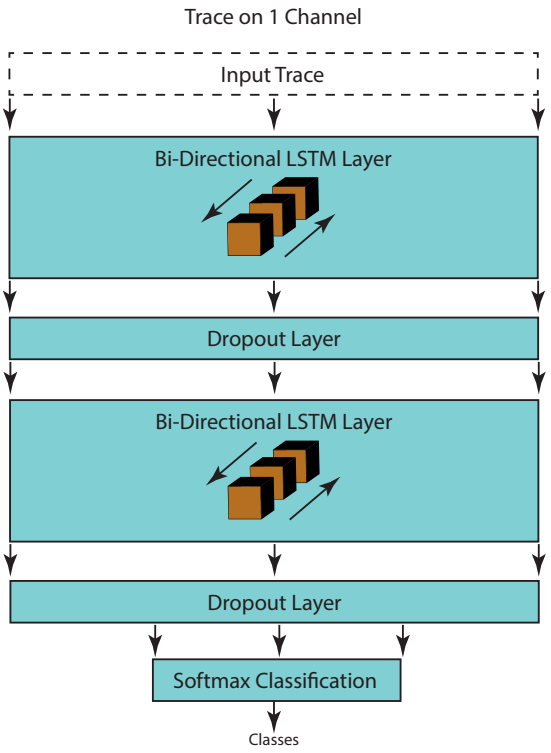


Fig. 12. The CNN-LSTM Architecture depicted as a stack of layers.