Se Eun Oh*, Saikrishna Sunkam, and Nicholas Hopper

p^1-FP: Extraction, Classification, and Prediction of Website Fingerprints with Deep Learning

Abstract: Recent advances in Deep Neural Network (DNN) architectures have received a great deal of attention due to their ability to outperform state-of-the-art machine learning techniques across a wide range of application, as well as automating the feature engineering process. In this paper, we broadly study the applicability of deep learning to website fingerprinting. First, we show that unsupervised DNNs can generate low-dimensional informative features that improve the performance of state-of-the-art website fingerprinting attacks. Second, when used as classifiers, we show that they can exceed performance of existing attacks across a range of application scenarios, including fingerprinting Tor website traces, fingerprinting search engine queries over Tor, defeating fingerprinting defenses, and fingerprinting TLS-encrypted websites. Finally, we investigate which site-level features of a website influence its fingerprintability by DNNs.

DOI 10.2478/popets-2019-0043
Received 2018-11-30; revised 2019-03-15; accepted 2019-03-16.

1 Introduction

Website Fingerprinting (WF) refers to a network-level traffic analysis attack in which the timing, direction, and volume characteristics of encrypted traffic between a web client and a proxy are used to identify the websites visited by the user, or to attempt to differentiate between visits to *monitored* sites (which may be targeted for censorship) and *unmonitored* ones. First introduced by Hintz in 2002 [15], recent work has focused primarily on the application of WF attacks to perhaps the most widely-used anonymity network, Tor. In this context, the attacker monitors the traffic between a Tor client and the guard relay, extracts features from this traffic, and then attempts to classify this traffic based on these features.

Security researchers have adopted a variety of machine learning algorithms for WF attacks on Tor. Wang *et al.* [37] utilized k-Nearest Neighbors (k-NN) with a new weight learning technique, Panchenko *et al.* [28] proposed a scalable attack with Support Vector Machines (SVMs), and Hayes and Danezis [12] adapted Random Forest classification (k-FP) to be resilient to WF defenses and noise. [2] These classifiers have attained high true positive rates (TPR) and low false positive rates (FPR) at the cost of a great deal of human effort in feature engineering and clever algorithm adaptations. The primary contribution of each of these works was to improve the performance of classifiers by introducing new classifiers or discovering new feature sets.

Recently, Deep Neural Networks (DNNs) have achieved impressive results in diverse research areas such as image recognition [13, 22, 34]. Because DNNs can use multiple layers to learn different levels of abstraction of input data [5] and can fit arbitrary functions with little prior knowledge, they have the potential to automate both types of contribution to the literature on WF attacks.

There have been several efforts to apply DNN to traffic analysis. Rimmer *et al.* [31] revisited WF with deep learning algorithms using an immense dataset, and Sirinam *et al.* [35] further evaluated DNN against recent WF defenses. However, their work did not attempt to isolate the benefits of automated feature engineering from classifier improvements, and they focused only on the classification ability of deep learning for website fingerprints. Even for the classification evaluation, their experiments are primarily based on limited experimental scenarios such as the closed-world and open-world binary classification settings, and using a single dataset.

In this paper, we further explore the applications of DNNs to traffic analysis, including their use in fea-

***Corresponding Author: Se Eun Oh:** University of Minnesota, E-mail: seoh@umn.edu
Saikrishna Sunkam: University of Minnesota, E-mail: sunk0015@umn.edu
Nicholas Hopper: University of Minnesota, E-mail: hoppernj@umn.edu

1 Perceptron

2 In this paper, we refer specifically to these three prior studies as *state-of-the-art* WF attacks in comparison to our work and other concurrent studies applying deep learning classifiers

ture extraction and selection, fingerprinting attacks, and fingerprint prediction. We evaluate their performance at these tasks across diverse experimental scenarios using different datasets, classification settings, and attack and communication scenarios not covered in previous work [31, 35].

Contributions. We summarize our key findings as follows:

Feature Engineering. To isolate the benefits of DNNs as feature extractors versus classifiers, we investigate the use of an autoencoder (AE), an unsupervised learning technique, to extract low-dimensional representations of a dataset, in combination with the classifiers used in state-of-the-art WF attacks. We notice that when AE-generated features are fed into state-of-the-art WF attacks, classifiers become more powerful than when using hand-tailored feature sets from recent WF attacks while requiring less computational cost. In particular, even with feature vectors reduced to as few as 40 dimensions, classifier accuracy is still improved. This suggests that AEs can be a powerful technique for feature engineering in new traffic analysis attacks.

Varying Classification Tasks. We studied the suitability of Multilayer Perceptrons (MLP) and Convolutional Neural Networks (CNN) [3] as p-FP classifiers (we use the names p-FP(M) for MLP and p-FP(C) for CNN) in a wider range of settings than Rimmer *et al.* [31] and Sirinam *et al.* [35]: in addition to identifying top Alexa web sites in the closed-world and open-world binary settings, we also study the performance of these architectures in other WF tasks including open-world multi-class classification, search query (keyword) fingerprinting [26], Onion Service fingerprinting, TLS-encrypted website fingerprinting, and WF against four traffic padding schemes – BuFLO [10], Tamaraw [9], WTF-PAD [18] and Walkie-Talkie [39].

We show that for all of these tasks, DNNs can achieve equivalent or better results to those published in the literature. In particular, p-FP(C) based on CNNs is capable of identifying 100 monitored websites against 40,000 unmonitored websites with 94% true positive rate and 0.009% false positive rate, using 30,000 traces of monitored training data. We also show that BuFLO, WTF-PAD, and Walkie-Talkie are less effective against p-FP than previous classifiers, since we successfully conduct multiclass classification on 100 websites with 15%, 57%, and 49% accuracy, respectively.

Predicting Fingerprintability Say that a website w is *p-Fingerprintable* by classifier c if open-world training with w in the monitored set produces a classifier that correctly identifies at least fraction p of instances of site w. In light of the success of DNN-based classifiers, we revisit the study of Overdorf *et al.* [27] to study the influence on fingerprintability by these classifiers using feature sets that only focus on elements in HTML documents, such as statistics about links and embedded web content, which we call *HTML features*. To the best of our knowledge, this is the first fingerprintability study using DNNs and focusing only on website design-specific features.

We find that several common features are influential for fingerprintability by both DNN and traditional classifiers, and identify ranges for these features that are common to *less*-fingerprintable sites. These results suggest that website designers interested in helping users avoid WF attacks (or onion service designers interested in protecting the location of their servers) can use the features we idenifty to predict the vulnerability of content pages and alter the HTML source code of those pages accordingly.

Reproducibility We provide all source code and datasets used in this paper on github. [4]

2 Related Work

2.1 WF attacks

The importance of feature engineering for Tor WF with cutting edge machine learning algorithms was first highlighted in 2011 by Panchenko *et al.* [29]. Their hand-built features based on traffic volume and timing were powerful enough to have high TPR in an open-world scenario. Building on this, Wang and Goldberg [38] proposed using Tor cell sequences as a new feature set and used SVMs with edit-distance-based metrics to yield classifiers with 95% TPR and 0.2% FPR. The k-NN classifiers with revised weight learning, proposed by Wang *et al.* [37], enabled the attacker to achieve higher TPR than prior WF attacks.

In 2016, Panchenko *et al.* [28] demonstrated more scalable WF attacks with larger background datasets

3 The MLP architecture features several layers of neurons, each fully connected to the layer before it; a CNN convolves its input with several local shared "filter" units, which are locally "pooled" using a common function, before applying one or more fully-connected layers; see Appendix A for further details

4 https://github.com/seeunoh2/pFP

than previous research. They described the *CUMUL* classifier, using SVMs with a new feature set based on cumulative traces, which obtained 96-97% TPR, higher than k-NN classifiers [37]. Hayes and Danezis [12] conducted a thorough analysis of new categorical features and built k-FP classification models based on Random Forests with Hamming distance. Their classifiers were less susceptible to padding-based WF defenses such as BuFLO [10] and Tamaraw [9].

Oh *et al.* [26] extended WF with a new feature set to investigate Keyword Fingerprinting (KF) attacks, which recover user-typed search engine queries over Tor. Their svmRESP classifier used SVMs trained with their RESP feature sets that focus only on the response traffic to capture embedded web objects in search engine results. They achieved 83% TPR and 8% FPR with svmRESP when trying to classify 100 monitored queries against 10k unmonitored queries.

2.2 WF defenses

To defend against these ever-more powerful attacks, researchers have devised methods to confuse the attacker's classifiers while aiming to keep bandwidth and latency overheads reasonable. BuFLO [10] adds dummy packets to fill in timing gaps and further extends the transmission to send packets of fixed length at fixed intervals. Tamaraw [9] improves BuFLO to be a more efficient and effective defense by using different padding intervals for incoming and outgoing packet directions and sending outgoing packets at a lower rate, which reduces the overhead in the common case of infrequent outgoing traffic. WTF-PAD [18] is a lighter-weight defense that focuses on hiding large gaps between traffic bursts by adding fake bursts. WTF-PAD significantly decreases bandwidth overhead and latency compared to BuFLO and Tamaraw while yielding good performance against the k-NN attack. Walkie-Talkie [39] is an efficient WF defense technique based on half-duplex communication, which makes many packet sequences the same, and burst molding, which adds fake cells to mold burst sequences into identical supersequences.

2.3 Related Prior Work

In this section, we review three recent studies of WF that are closely related to our work [27, 31, 35] and discuss the differences between their work and ours.

2.3.1 Automated Website Fingerprinting

In their work on Automated Website Fingerprinting (AWF), Rimmer *et al.* [31] investigated three DNN architectures – LSTM, CNN, and Stacked Denoising Autoencoder (SDAE) – for WF from Tor network traces. Our work differs from theirs in three aspects: we use a different evaluation methodology, evaluate a wider variety of application scenarios, and investigate the utility of autoencoding for feature extraction *independently* of the use of DNNs for classification.

Although we do not conduct a systematic closed-world study, we demonstrate the multinomial classification ability of DNNs in an open-world evaluation. To show the impact of the monitored dataset, we also evaluate p-FP(M) and p-FP(C) using the AWF and Tor Hidden Service (HS) datasets. As opposed to their work, our open-world evaluation follows the methodology of most other recent work on WF [12, 17, 28, 35, 37], in which the adversary trains WF models using both monitored websites and unmonitored websites, allowing better comparison across classifier models.

Application Scenarios. We explore a more diverse set of WF tasks using deep learning, including fingerprinting search query traces over Tor [26], Tor traces defended by recent WF defenses [9, 10, 18, 39], TLS-encrypted (non-Tor) traces, and predicting the fingerprintability of websites (Section 6).

Autoencoder. Rimmer *et al.* use a Stacked Denoising Autoencoder (DAE) architecture for feature extraction, based on the DAE, which is a variant of AEs designed to give better generalization. However, the AWF study did not attempt to isolate the effectiveness of AEs for feature extraction from the effectiveness of DNNs for classification using these features. We address this gap by investigating the use of AE-extracted feature vectors as input to other classification algorithms. In Section 4, we use the low-dimensional feature vectors extracted by an AE to train three state-of-the-art machine learning techniques – SVM, k-NN, and k-FP [12] – and compare the effectiveness of the resulting classifiers with that of CUMUL [28], k-NN [37], and k-FP [12] using their features.

2.3.2 Deep Fingerprinting

Concurrently to our preliminary work, Sirinam *et al.* [35] showed the importance of the details of the CNN architecture by demonstrating that a more tailored "Deep Fingerprinting" (DF) CNN can achieve very high WF performance against Tor traces, even against lightweight defenses such as WTF-PAD and Walkie-Talkie. The CNN architecture they propose has superior performance to the architectures we evaluated for defended Tor traces, but as with the AWF paper, the DF

paper did not explore the same breadth of WF scenarios as our work, nor did it examine the use of DNNs for feature extraction independently of classification. Moreover, their open-world analysis focuses on binary rather than multinomial classification.

In contrast, our experiments consider both binary and multiclass classification; we also explore a more diverse set of DNN models, MLP, CNN, and AE, and evaluate them across diverse fingerprinting scenarios (i.e., search query and TLS-encrypted trace fingerprinting) and datasets (i.e., the AWF and Tor HS datasets).

In addition, we explore different CNN architectures based on 2-2D convolutional layers with Local Response Normalization (LRN) while DF uses 8 1D convolutional layers with batch normalization. It has been shown [20, 32] that LRN handles ReLU [5] neurons, which have unbounded activations, and detects high frequency features with large response properly.

2.3.3 Onionsite Fingerprintability

The "Onionsite Fingerprintability Study" (OFS) of Overdorf *et al.* [27] used both network-level features based on the onion sites' network traces for state-of-the-art WF attacks [12, 28, 37] and site-level features based on HTML files and HTTP headers to determine the best predictors for WF outcomes. Our study has key differences in terms of features considered, methodology, and fingerprintability scoring.

Features and methodology. Overdorf *et al.* studied which of these network-level and site-level features were the most important for WF prediction using the relative difference between inter- and intra-class variance of network-level features and a random forest regressor, respectively.

In contrast, we only use features based on Alexa websites' HTML source code, *design-level* features such as the number of tags and characters in data fields. These features are easy to gather and measure without deploying the site as an onion service and have less variation due to external factors (such as network conditions), which makes the predictions easier to obtain and more stable.

Our goal is to identify which design-level features influence predictability by specific classifiers such as MLPs, CNNs, or k-FP. Thus, our study is in a different setting (open-world vs. closed-world), and it produces features that can be applied directly to a site's HTML source code before deployment.

Fingerprintability Scores. In order to study what makes a website vulnerable to fingerprinting, we must choose some way to assign a label or "fingerprintability score" to a website. The OFS used the F1 score from an ensemble classifier, combining the precision and recall of three classifiers [12, 28, 37].

We use multiple classifiers – MLP, CNN, k-FP, and SVM – independently and compute the score for each of them. We use the *accuracy* of each classifier for a website, calculated as the fraction of correctly identified instances for each website, as the fingerprintability score; this is more appropriate to our goal of predicting the influence of features on the performance of a single classifier as opposed to the OFS goal of measuring the influence of features on a broad range of classifiers.

3 p-FP Overview

In this section, we introduce our adversary model, DNN architectures, metrics to evaluate the performance of our classifiers, hyperparameter tuning to find the optimal parameters to train DNN models, and the datasets used to evaluate our DNN models.

3.1 Threat Model

As in all prior work on website fingerprinting over Tor, we assume a network-level, passive adversary who is only able to monitor network traces, sent and received by users. In situations involving Tor traffic, the adversary is only able to observe network traffic between the client and a Tor entry guard, and does not control any other relays or servers involved in the communication.

Our attacker is interested in two types of classification problems. In *binary classification* the goal is to determine whether a captured trace is from a small list of monitored pages. In *multiclass classification*, the adversary additionally predicts which of these monitored pages was visited.

3.2 DNN Architectures

We discuss general background on DNN models in Appendix A, and details about the selection of the hyperparameters for our networks in Section 3.4.

Experiment setup. We used Tensorflow [1] with the TFLearn [2] front end for the implementation of DNN classifiers. We split the dataset into training, validation, and testing datasets with size ratios 54:6:40, which led to better generalization of the trained models and helped to avoid overfitting. To train our models, we built

5 Rectified Linear Units, a fast, non linear function giving x for positive x and 0 for negative x

five models for each experiment and selected one model yielding the best performance. We then evaluated this model using 20 different iterations, where each iteration consisted of randomly chosen background instances and monitored samples, and each monitored website had the same number of instances.

We used 32 cores and 256GB of memory for all classifiers and the longest job, which trained a CNN consisting of two 2-D convolutional layers and two fully connected layers, was finished within five days. With GPUs, this running time would be considerably reduced.

MLP. The p-FP(M) model (Figure 6a of Appendix A) consists of one input layer allowing a single vector, two hidden fully connected layers, and one output layer with softmax function. We used L2 regularizations for the first two hidden layers and the dropout between those layers to minimize the impact of overfitting. We chose Stochastic Gradient Descent (SGD) as the optimizer and used the categorical cross entropy for the loss function.

CNN. The p-FP(C) model (Figure 6b of Appendix A) is comprised of an input layer accepting input data, two convolutional layers with 128 filters, each followed by a max pooling layer, followed by one hidden fully connected layer and a softmax output layer.

For the input shape, rather than n by n matrix (with original feature vector of size n^2), we constructed a 1 by n matrix, so that the vector of 2,500 features is represented as a 1 x 2500 matrix, which is better suited to WF. We also adjusted the format of filters, using 1 x f rather than f x f filters, because network traffic traces are time series rather than spatial data and we would not expect to find useful patterns in multiple spatial dimensions. We applied L2 regularization for each layer and the categorical cross entropy to compute the loss. Between layers, we used LRN.

AE. An autoencoder (Figure 6c of Appendix A) consists of an encoder and a decoder network, each consisting of two fully connected layers. We varied the number of units in the second hidden layer to get various feature dimensions to be evaluated with SVM, k-NN, and k-FP classifiers. To extract AE-encoded features from target traces, we saved a trained model and retrieved weights of the second hidden layer to derive compressed vectors of those inputs. We used the mean squared error to compute the loss. We used a ratio of 45:5:50 for training, validation, and testing datasets, and extracted AE features based on two testing datasets after two iterations.

3.3 Metrics

Traditional metrics. For the open-world experiments, we used the following metrics, adopted in prior WF work [12, 26, 28].

- True positive rate (TPR): The proportion of positive samples that are predicted as positive.
- False positive rate (FPR): The proportion of negative samples that are mispredicted as positive.
- For both binary and multiclass classification, we trained DNN models using distinct labels for each monitored class plus one additional label for all unmonitored traces. The difference between the settings is whether we count the confusion between monitored classes as a TP. In binary classification, if a monitored trace is assigned *any* of the monitored labels, it is counted as a TP, whereas in the multiclass setting, a monitored trace is only considered a TP if the correct label is predicted. This is consistent with previous WF attack evaluations in the literature [12, 28, 35].
- Bayesian detection rate (BDR): Since reporting accuracy without consideration of the base rate or a prior may have led to bias in early attempts to evaluate WF attacks, we follow the suggestion of Juarez *et al.* [17] and also report the BDR, computed as $P(M|V) = \frac{P(V|M)P(M)}{P(V|M)P(M)+P(V|U)P(U)}$, where V indicates the event that a webpage is classified as monitored, M is the event that the webpage actually belongs to the monitored set, and U is the event that the page belongs to the unmonitored set, i.e. $P(U) = 1-P(M)$. $P(V|M)$ is approximated by TPR and $P(V|U)$ is estimated by FPR. We computed BDR in the same way as prior work [12, 17].
- Within-monitored accuracy (WMacc): For KF experiments, we used within-monitored accuracy as proposed by Oh *et al.* [26] to measure the performance of multiclass classifiers. This is computed by the number of TPs divided by the total number of monitored samples.

Confidence threshold. In our DNNs, the output layer returns vectors with prediction probabilities for all labels. Even though the label with the highest probability is selected as a predicted label, if this probability is low, this indicates the classifier has low confidence in its prediction. To avoid using low-confidence predictions, if the highest probability is less than a confidence threshold, we regard this case as being classified as "others". We also varied this confidence threshold to study how this factor affects the performance of DNNs. In Section 5, for fair comparison to prior WF attacks [31, 35], we use the

Receiver Operating Characteristic (ROC) curve, which is a parametric curve that summarizes the tradeoff between TPR (y-axis) and FPR (x-axis) of a classifier as the confidence threshold varies from 0 to 1.

Top-K analysis. The prediction probability vector returned by the output layer enables us to consider other meaningful labels if their probabilities are high enough to be trustworthy even if they are not the highest. Such labels are candidates for the top k list. We studied the performance of these "top k" predictions as k varied from 1 to 5, where if a correct monitored label appears in the top k list, we count it as a TP.

Because of the possibility that the features of Top Alexa websites have a common bias, we treat the appearance of the unmonitored label in the top k list more importantly. In the open world experiments, if the negative (unmonitored) label is included in the top k list and the actual label is positive (monitored), we always treat this as a false negative (FN) even if the top k list includes a true label. For example, if the top 3 list includes monitored sites M_1 and M_2 with probabilities 0.89 and 0.1, respectively, as well as the unmonitored label U with probability 0.01, then even if the true label is M_1, this outcome is coded as FN. We applied top k analysis to evaluate KF attacks (Section 5.2) and defended traffic analysis (Section 5.4).

3.4 Hyperparameter Tuning

Training DNNs requires selecting many different hyperparameters that can impact the quality of predictions. We used the `hyperopt` library [6] to implement the Tree of Parzen Estimators (TPE) [7], a form of Bayesian optimization, to automate the search for optimal hyperparameters. First, we described the search space, as shown in Table 15 of Appendix B, and constructed DNNs using the `tflearn` library [2]. Then, DNN models were trained and the prediction results were passed to the optimizer, which selected parameter values for the next iteration to minimize the prediction error. We summarize the search space and chosen values for each DNN model used in our experiments in Table 15.

3.5 Datasets

We detail how we collected our datasets and categorize the network traffic traces for different experimental scenarios.

Data Collection. We used a modified version of Tor browser crawler (TBC) [3] and Tor version 0.4.0.8 to collect our datasets. Our modified version of TBC reset the Tor process after each website visit, but otherwise used the default options. We discuss the reasons for this modification and possible impacts on our results in Appendix C. In addition, we used the page source option provided by Selenium and the Beautifulsoup library to extract the HTML source code for each downloaded page.

Website Tor traces (WTT). We collected the WTT dataset with a time gap of one week between batches (We named this dataset WTT-time). Each batch executed one week after the previous batch. For the monitored dataset, we collected 10 different batches where each batch collects 100 traffic instances, HTML source code, and screenshots of each of 100 websites. For the unmonitored dataset, we harvested 10 batches, in which each batch was comprised of 6,000 different Alexa websites ranked between 200 and one million. The collection of this dataset ran from August through December of 2018, which was long enough to capture the impact of dynamic web objects in our collection.

Some instances resulted in abnormal HTML files (for example, due to server errors or network conditions). We noticed that these instances always had HTML files less than 10KB in length, but since some sites also had normal HTML files below this length, we filtered out these failures by manually inspecting the screenshot of any instance with a HTML file of size less than 10KB.

After excluding instances with abnormal capture files or HTML files and sampling uniformly at random from the 10 batches comprising the remaining instances, we ended up with 300 instances each for 100 websites and one instance of each of 50,000 unmonitored websites. Since we failed to extract HTML files for some websites such as netflix since the Beautifulsoup library could not parse their HTML document structure, our final set of 100 monitored websites were selected from the Alexa top 150 sites.

Since Tor encapsulates all data into cells of 512 bytes, we extracted Tor cell sequences from the set $\{1, -1\}$, indicating that the client sent one cell or received one cell, respectively, from each website trace. We determined the optimal feature dimension using hyperparameter tuning, as described in Section 3.4.

To evaluate our models with other datasets, we also used Wang dataset, provided by Wang *et al.* [37], the "Tor HS dataset", shared by Hayes and Danezis [12], and the AWF dataset of Rimmer *et al.* [31].

Keyword Tor traces (KTT). For both monitored and background traces, we used Google search query traffic instances [26], which include 100 instances of each of

100 top-ranked monitored keywords and 80,000 unmonitored keywords.

Website SSL (non Tor) traces (WST). To collect TLS-encrypted traces, we build a normal Firefox web browser crawler using Selenium's web driver after removing the Tor settings in TBC [3]. This set consists of 9,000 instances (90 instances each) of the Alexa top 100 websites for a monitored set and 9,000 Alexa websites excluding monitored sites for an unmonitored set.

4 Feature Analysis

In this section, we demonstrate that DNNs can be used to perform automated, unsupervised feature extraction even for use with traditional classification algorithms.

4.1 Features with Autoencoder

An AE is a widely used unsupervised technique for learning data representations and feature dimensionality reduction, because an encoder network projects the original feature vector into a lower-dimensional representation. This feature compression can both discover useful features and make classifiers more efficient because their training time often depends on the dimension of the input data. Previously, Nasr *et al.* [25] proposed the use of linear projection algorithms from compressed sensing to enable WF [37] and flow correlation attacks [16] with reduced storage and computation cost, at a slight loss in accuracy. In contrast, we use a deep learning framework to do the dimension reduction, which has previously been shown to be an effective methodology to learn structural data representations more efficiently, compared to compressed sensing [24].

As shown in Figure 6c of Appendix A, we trained an encoder and a decoder network and captured feature vectors compressed by the second hidden (blue) layer of the encoder network. We also varied the number of units from 10-100 in this hidden layer to compress the original features into various low-dimensional representations. We evaluated SVM, k-NN, and k-FP classifiers with encoded features and reproduced state-of-the-art WF attacks to compare their performance. This analysis shows that an AE can learn interesting structure about the data while reducing its dimension.

We additionally tested a variational autoencoder (VAE) [19, 30] and extracted encoded features as explained above; however, we failed to extract meaningful traces. With VAE features, we achieved 2% TPR for multiclass classification and 3% TPR for binary classification. While VAEs perform nicely for datasets where the latent space is continuous, allowing random sampling or interpolation to learn variations on data [23], website traffic instances are less likely to have a reasonable local density.

A DAE is another type of AE, adopted by Rimmer *et al.* [31] to prevent overfitting. By using stochastically corrupted input such as adding noise to the input, a DAE avoids learning the identity function. Since we did not experience significant overfitting with an AE and our generated features make the standard classifiers more effective (Table 1), we leave evaluation using a DAE as future work although we acknowledge that it is another powerful method for feature engineering.

Table 1. The best performance after 20 iterations (A(n): n AE features, TRT: Training time, DC: Distance Computation time, m: minutes). We empirically selected n yielding the best result.

Data	Wang		WTT-time			
Metrics	TPR	FPR	TPR	FPR	TRT	DC
k-NN	90.2	10.3	91.7	26.6	7.3m	-
A(80)+*k*-NN	97.9	2.1	93.7	14.9	1.1m	-
k-FP	88	0.5	90.1	5.4	39m	82m
A(100)+*k*-FP	95.9	1.4	91.3	7.8	1.2m	32m
CUMUL	96.6	9.6	86.9	11.3	84m	-
A(80)+SVM	97.6	1.5	91.4	7.8	87m	-

Table 2. Performance of state-of-the-art machine learning algorithms with AE features (dim: feature dimension).

dim	40		80		100	
	TPR	FPR	TPR	FPR	TPR	FPR
k-NN	92±1	15±1	93±1	15±1	92±1	15±1
SVM	88±1	11±1	91±1	7±1	90±1	7±1
k-FP	90±1	8±1	91±2	7±1	91±1	7.8

4.2 Feature Engineering with AEs

We revisited previous state-of-the-art WF attacks [12, 28, 37] and used features learned by an AE (AE features) to train k-NN, SVM, and k-FP classifiers. The feature engineering performed by the authors of these works [12, 28], requires a considerable amount of human effort such as manually inspecting the traffic pattern and experimentally deciding the optimal dimension of feature vectors. Feature extraction based on AEs offers the potential to automate this process. We used 90 instances of each of 100 monitored websites and 9,000 background websites in Wang dataset [37] and 300 instances of 100 website traces and 20,000 unmonitored instances in WTT-time dataset.

We used the classifier implementations of CUMUL [28] and k-FP [12] directly after removing the feature extraction code. However, since Wang's implementation of k-NN [37] is suited to features based on a Tor cell trace, we implemented k-NN using

`sklearn.neighbors` with weight learning that computes the inverse of the distance between neighbors to handle AE features, which may not be in $\{-1, 0, 1\}$, in a better manner. For k-FP, we tuned the number of estimators to have the optimal number of trees finding that dimension 1,000 was best for features suggested by their work [12] and dimension 100 was best for AE features. We used both 1 and 3 for k.

To understand the effect of AE feature dimension on the performance of classifiers, we varied the number of units in the hidden layer, which encoded website traffic vectors, from 10-100. We saved a trained AE model and computed the compressed features on target data using an encoder network by loading the trained model.

Since the dimension of encoded features was significantly lower than the length of the original input vector, this leads to dimensionality reduction while yielding similar or better performance. As shown in Table 2, we found that the dimension of hidden units of an AE did not impact the performance of the machine learning algorithms significantly. Furthermore, the degree of improvement is not proportional to the number of features since with 100 features, k-NN classifiers yielded slightly worse results than with 80 features.

In Table 1, we also reproduced the results of state-of-the-art WF attacks [12, 28, 37] using WTT-time and Wang datasets in the open-world setting. For the WTT-time dataset, we alse report Training time (TRT), which includes feature extraction time, along with TPR and FPR to show the effect of dimensionality reduction on the computational cost. Since computing all pairwise hamming distances is the most expensive operation in k-FP, we isolated this time in the table. For both datasets, AE features outperformed all traditional WF features with much lower computational cost.

For Wang dataset, we achieved lower FPR than k-NN [37] and higher TPR than k-FP [12] while completely automating the feature engineering. [6] In comparison to CUMUL [28], we obtained similar results in much shorter training time excluding the feature extraction (39 minutes vs. 4 hours).

For the WTT-time dataset, k-NN requires seven minutes for feature extraction while training and testing an AE takes only one minute to generate its features. k-FP takes 36 minutes for feature extraction and 82 minutes for distance computation while k-FP with AE features reaches similar performance with much lower computational cost. With k increased from 1 to 3, k-FP with 80 AE features yielded 83% TPR and 4% FPR. Compared to CUMUL, since the feature dimension is very similar, where we have 80 features for AE and 100 features for CUMUL, training cost is very similar. However, AE features achieved better performance with 91% TPR and 7.8% FPR.

5 Traffic Analysis

In this section, we evaluate the applicability of DNNs to various website fingerprinting attack goals.

5.1 Website Fingerprinting on Tor

First, we explore the effect of various experimental settings on the accuracy of DNNs in WF attacks. Across all experiments, we evaluated both p-FP(M) and p-FP(C) classifiers in the open-world setting.

Unmonitored set. To show the effect of unmonitored set size, we trained p-FP(M) and p-FP(C) classifiers with the WTT-time dataset, where we have 300 traffic instances of each of 100 monitored websites and one traffic instance each of either 20,000 or 40,000 background websites.

Increasing the number of unmonitored website traffic instances weakened the performance of both classifiers but not significantly, since we measured 94% TPR and 2% FPR for p-FP(C) multiclass classification even against 40,000 background sites, giving BDRs ranging from 97-98% (Table 4). Tables 3 and 4 show that p-FPs are very successful at open-world WF attacks and in particular, p-FP(C) achieved very low FPR in predicting whether a trace was monitored or not.

Monitored set size. We also evaluated both p-FP(M) and p-FP(C) when using 200 and 300 instances each of 100 monitored sites. While going from 20,000 monitored instances to 30,000 monitored instances did not have a noticeable effect on the results of p-FP(C), increasing the size of the monitored set somewhat improves the results of p-FP(M) as shown in Figures 1a and 1b.

Number of Training epochs. More epochs improve the quality of classification for both models, as presented in Figure 1c and 1d. However, after 50 epochs, both TPR and FPR started to decrease. Thus, we chose 50 epochs as the optimal number of epochs to train p-FP(C) models.

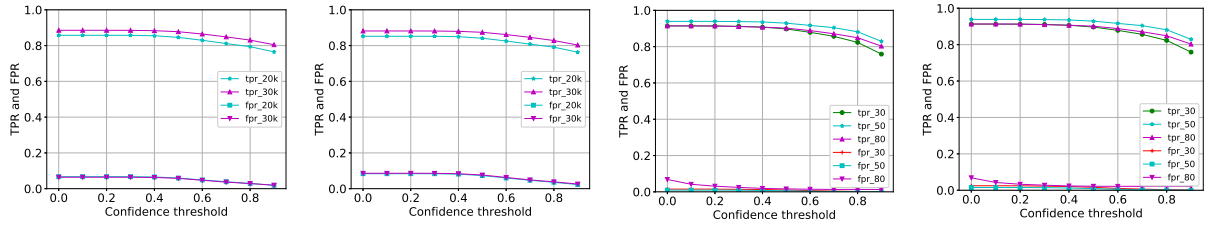
6 For Wang dataset, Panchenko *et al.* [28] reported 89.61% TPR and 10.63% for k-NN and 96.64% TPR and 9.61% FPR for CUMUL (SVM), and Hayes and Danezis [12] presented 87% TPR and 0.9% FPR for k-FP when using 4,500 unmonitored fingerprints.

Table 3. WF with p-FP(M) with 30k monitored dataset and varying size of unmonitored sets (TPR(T), FPR(F), and BDR(B) (%), and H=Tor HS). Note that all results are based on the confidence threshold tuned to yield higher TPR.

Size	Multiclass			Binary		
	T	F	B	T	F	B
20k	89±1	15±1	90±1	90±1	10±1	94±1
40k	87±1	5±1	90±1	88±1	7±1	93±1
30k(H)	94±2	3	70±1	96±1	0.06	97

Table 4. WF with p-FP(C) with 30k monitored dataset and varying size of unmonitored sets (TPR(T), FPR(F), and BDR(B) (%), and H=Tor HS) Note that all results are based on the confidence threshold tuned to yield higher TPR

Size	Multiclass			Binary		
	T	F	B	T	F	B
20k	95±1	1±1	97±1	95±1	0.007	98±1
40k	93.77	1±1	98±1	94±1	0.009	98±1
20k(H)	98.49	3.69	78±1	98.91	0.18	98.6



(a) Binary p-FP(M),varying the training data size

(b) Multiclass p-FP(M),varying the training data size

(c) Binary p-FP(C),varying training epochs

(d) Multiclass p-FP(C), varying training epochs

Fig. 1. WF evaluation using 40k unmonitored set(a,b), and using 300 instances of each of 100 monitored sites (c,d).

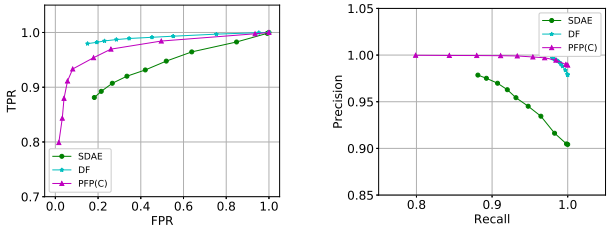
Dataset. To explore the effect of different datasets on the performance of p-FP classifiers, we also used the open-world dataset collected by Rimmer *et al.* [31] to construct binary classifiers. Following their evaluation, we report results in Table 5 after optimizing the confidence threshold for lowest FPR and highest TPR, respectively. With 300 traces of 100 monitored websites and 40,000 unmonitored traces, both classifiers achieved very low FPR. This is because most misclassification cases were confusions between monitored labels. However, reducing FPR also sacrifices TPR. We believe that this gap can decrease with more training data. With the Tor HS dataset [7] ((H) in Tables 3 and 4), both classifiers performed more effectively with 94-98% TPR and 3-4% FPR. These results suggest that p-FPs can achieve very low open-world FPR regardless of the monitored set, but some monitored sets will result in better TPR than others.

Table 5. The performance of p-FP(M) and p-FP(C) using 30k monitored and 40k unmonitored AWF dataset.

	Optimized for TPR		Optimized for FPR	
Metrics	TPR	FPR	TPR	FPR
p-FP(M)	80±1	2±1	68±2	0.0009
p-FP(C)	82±1	1±1	68±1	0.0008

Confidence threshold. The confidence threshold represents the reliability of decisions by classifiers. We applied confidence thresholds ranging from 0 to 90% (0 corresponds to argmax) to the prediction probabilities of our classifiers, and evaluated both classifiers using both 20,000 and 30,000 monitored traces with 40,000 background traces. As expected, we found that increasing the confidence threshold reduced the number of confident TPs, which lowered TPR, and increased the number of TNs, which decreased FPR (Figure 1).

Network Architecture. As shown in Tables 3 and 4, p-FP(C) performed much better across all sizes of background sets and both classification tasks. In particular, p-FP(C) yields very low FPR in the WTT-time dataset. More interestingly, p-FP(C) exhibits much better performance for multinomial classification; the number of TPs does not change significantly if we switch the classification task from binary to multiclass classification. In other words, with p-FP(C), the number of confusions between monitored classes decreases.



(a) ROC

(b) Precision-Recall Curve

Fig. 2. Comparison to SDAE and DF using 300 instances of each of 100 websites and 40k unmonitored traces in WTT-time dataset.

7 Comprised of 90 instances each of 30 onion services and 30,000 background website traces

Prior Work. We also trained and evaluated SDAE [31] and DF [35] networks using the WTT-time dataset, to compare their performance. Although Rimmer *et al.* also evaluated WF using CNN and LSTM networks, we only evaluated their SDAE architecture since it achieved the best performance in open-world experiments in their paper [31]. As shown in Figures 2a and 2b, DF outperformed SDAE and p-FP(C). p-FP(C) produced very low FPRs (high precision) even for lower confidence thresholds, with FPRs of 0.008% for p-FP(C) vs. 14.29% for DF and 15.88% for SDAE. However, the TPR of p-FP(C) was significantly lower compared to other attacks, with a TPR of 74.84% for p-FP(C) vs. 97.59% for DF and 86.75% for SDAE. This means that with very high confidence thresholds – greater than 0.96 – p-FP(C) predicts that more monitored samples are unmonitored. With a much larger dataset, the degree of drop in TPR would be reduced, however, we leave detailed investigation of the tradeoff between network architecture and required dataset size for future work.

Summary. p-FP classifiers have been shown to be effective for WF across the different experimental scenarios; compared to related work, although p-FP shows less stable performance than DF, it yields lower FPR across different types of monitored dataset.

5.2 Search Query Fingerprinting on Tor

We study the more fine-grained classification ability of DNNs by applying them to KF, which fingerprints Google search query traces over the Tor network. We used 100 instances of each of 100 monitored Google keywords and 10,000 background keyword traffic instances in the KTT dataset; we also used two feature sets, RESP traces (defined below) and Tor cell traces.

Table 6. KF with p-FP(M), and p-FP(C) using RESP and cell traces. ((b):binary classification, (m):multiclass classification)

Metrics	RESP(C)	RESP(M)	Cell(M)
TPR(b)	86±1	88±1	59±2
FPR(b)	5	5±1	11±2
BDR(b)	95	95±1	85±2
WMacc(m)	22±1	27±1	22±1
BDR(m)	59±2	63±1	50±1

Features. RESP features are extracted from the "response portion" of a Tor trace, defined as the largest sequence of incoming packets, and in previous work, Oh *et al.* [26] extracted the sequence of cumulative sizes of TLS records from the response portion. In this work, we ignored the cumulative setting because it gave us lower accuracy.

Since Oh *et al.* [26] found that RESP-based features improved the performance of KF attacks using SVMs, we explored whether this is true with p-FPs. Compared to Tor cell traces, Table 6 shows that RESP features substantially enhanced the p-FP(M) classifiers' performance. Furthermore, with RESP features, we achieved better performance (88% TPR and 5% FPR) than svmResp [26] in binary classification since svmResp yielded 82.6% TPR and 8.1% FPR. RESP features rather than cell traces help DNNs do better KF classification.

Table 7. p-FP(C) performance using sequence of TCP packet sizes(TC) and TLS record sizes(TL) after being sorted by time (1: Top1, 3: Top3, T:TPR(%), F:FPR(%)).

	Binary			Multi		
	TC(1)	TL(1)	TL(3)	TC(1)	TL(1)	TL(3)
T	93±1	93±1	96±1	93±1	93±1	96±1
F	4±1	3	1	5±1	4±1	1

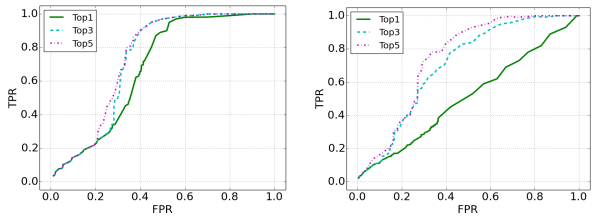
Top-K analysis. Compared to binary classification, multinomial classification results using p-FP classifiers were not as powerful, as shown in Table 6. To further investigate the performance of multiclass classifiers, we conducted top-k analysis. As shown in Figure 3, top k analysis substantially improved the quality of multiclass classification for both Tor cell and RESP features. When using the top 5 analysis, MLP classifiers achieved higher WMacc than svmRESP [26] (62% vs. 55%).

Confidence threshold and network. As expected, increasing the confidence threshold deteriorates the performance of both classifiers. p-FP(C) also exhibits larger standard deviation with high confidence thresholds than p-FP(M). One possible explanation for this is that since there is much less distinction power between keyword traces, fewer local patterns are learned by filters, making CNNs a less ideal choice for this task. However, this bottleneck can be improved with more training data, which would make the prediction results more consistent across different portions of the dataset.

Summary. As opposed to general WF, researcher-selected RESP features enhance the performance of p-FP classifiers for KF, and using top-k analysis, p-FP classifiers provide better KF results than prior work [26]. However, there is still room for improvement in KF by discovering different features or training other types of networks, which we leave as future work.

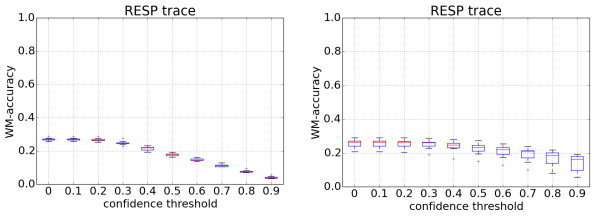
5.3 WF with TLS Proxies

We evaluated both p-FP(M) and p-FP(C) classifiers for WF attacks on TLS-encrypted traces using Firefox, to simulate the use of a TLS proxy. We experimented with several trace representations for this task.



(a) Resp trace (b) Cell trace

Fig. 3. ROC of Top-K analysis on KF for multiclass classification



(a) p-FP(M) (b) p-FP(C)

Fig. 4. KF(RESP) with p-FP(M) and p-FP(C) by varying the confidence threshold

Packet Direction. We extracted the sequence of TCP packet directions from each trace, with entries of -1 for incoming packets, and 1 for outgoing packets; traces shorter than the minimum length were padded with 0 entries. This resulted in 82.9% TPR and 6.8% FPR for binary classification and 82.6% TPR and 8.4% FPR for multiclass classification using p-FP(C).

TCP packet sequence. We also evaluated a representation based on the size and direction of TCP packets, sorted based on transmission time, resulting in the column labelled TCP in Table 7.

TLS record sequence. We also extracted the sequence of TLS record sizes and directions and sorted it based on the transmission time, resulting in the columns labeled TL in Table 7). As shown there, this representation achieved almost the same results as with TCP packet-based features when differentiating between the Alexa top 100 websites.

MLP vs CNN. We further evaluated p-FP(M) classifiers using all representations discussed above, however, they failed to yield better performance than p-FP(C). Compared to Tor trace WF, p-FP(C) is a much better model to fingerprint TLS-encrypted network traces since CNNs produce stronger representations of the step by step interactions in website downloads based on the local input patterns in TLS-encrypted traffic.

Summary. Filters in p-FP(C) successfully learn patterns related to the size of TCP packets and TLS records. However, the performance of p-FP(C) for this task was surprisingly lower than the results against Tor traces due to the use of a smaller dataset. We leave evaluation on larger datasets for future work.

5.4 WF on Tor with WF defenses

We evaluated p-FP classifiers against several recent WF defenses: BuFLO [10], Tamaraw [9], WTF-PAD [18], and Walkie-Talkie [39]. This study helps to understand how automated feature learning by DNNs is impacted by recent WF defenses. We used Wang dataset and the WTT-time dataset consisting of 300 instances of each of 100 websites and applied each defense to those datasets to generate defended network traces. We further evaluated DF [35] on those defended traces for comparison.

Table 8. p-FP(M) and p-FP(C) performance against BuFLO(B) and Tamaraw(T) (M: p-FP(M), C: p-FP(C), T: Top n accuracy, D: Defense, and all metrics are %) For bandwidth overhead, BuFLO-Wang=217%, Tamaraw-Wang=181%, BuFLO-WTT=179%, and Tamaraw-WTT=175%. Note that for undefended WTT-time dataset, we measured 90% using p-FP(M), 91% using p-FP(C), and 96% using DF, and for undefended Wang dataset, we got 86% using p-FP(M), 92% using p-FP(C), and 96% using DF.

	M		C				DF			
	WTT		Wang		WTT		Wang		WTT	
T	B	T	B	T	B	T	B	T	B	T
1	9	15	16	16±1	17	13	15	7	15	11
2	14	23	19	29±1	22	21±1	24	12	22	18

After hyperparameter tuning, we built the optimal CNN architecture using one convolutional layer, followed by two fully-connected layers, to fingerprint defended traces. Note that it is a different architecture than used in Sections 5.1 5.2, and 5.3.

BuFLO/Tamaraw. Due to padding, traffic instances captured under WF defenses result in longer cell traces: for p-FP(M) classifiers and Wang dataset, we used 20,164- and 15,129-dimensional feature vectors for BuFLO and Tamaraw, respectively; for CNN classifiers and Wang dataset, we used 30,000- and 25,000-dimensional feature vectors. For the WTT-time dataset, we always use 10,000-dimensional features in p-FP and 5,000-dimensional features in DF. In Table 8, we show the performance of classifiers using Wang and WTT-time datasets in the closed-world setting, following the analysis of Hayes and Danezis [12].

For Wang dataset, as shown in Table 8, p-FP and DF classifiers performed much better than other WF attacks against Tamaraw [12]. This demonstrates that padding-based defenses are not able to completely defeat WF using deep learning, which enables more sophisticated and automated feature analysis. All three clas-

Table 9. Top-1 accuracy of p-FP(C) for varying parameters in BuFLO (minimum padding time τ, interpacket interval ρ) and Tamaraw (padding length multiple) using Wang dataset. Note that BO is bandwidth overhead and all numbers are %.

(a) BuFLO

	min. padding time τ (s)			
ρ (s)	BO	10	50	100
0.02	434	15.59	10.53	5.62
0.03	290	18.58	12.15	6.02
0.04	217	20.16	11.98	5.37

(b) Tamaraw

pad$L$	BO	Acc
100	181	15.66
1000	248	5.61
1500	284	3.14

sifiers performed slightly worse than k-FP [12] against BuFLO, while still significantly outperforming random guessing. In particular, despite padding at a constant rate, both defenses still expose some information about statistics such as the total size of TCP packets, which is learned by DNN models and results in increased accuracy (16-29%) especially against Tamaraw. For both Wang and the WTT-time datasets, Top-1 analysis of p-FP(C) shows slightly better performance than DF.

We also varied the parameters used in BuFLO and Tamaraw to study their impact on the classifiers. For BuFLO, we varied minimum padding time (τ) and packet interval (ρ) of dummy packets in Table 9a, and for Tamaraw, we explored different padding multiples (L), with fixed outgoing and incoming padding intervals of 0.04 and 0.012, as shown in Table 9b. As expected, longer padding length and padding time decrease the accuracy, however, even with 100 seconds padding time in BuFLO and padding multiple 1500 in Tamaraw, the accuracy of p-FP(C) still exceeds random guessing.

Table 10. The performance of p-FP(C) classifiers against WTF-PAD. For Top k analysis, we chose the confidence threshold yielding optimal accuracy (Bandwidth overhead=37.7%).

	DF		p-FP(C)	
Top$k$	Undef	Def	Undef	Def
Top1	96±1	93	91	57±1
Top2	97±1	95	94	62±1

WTF-PAD. We evaluated p-FP(C) against WTF-PAD using the defended WTT-time dataset after applying WTF-PAD with normal fits [18]. Although we followed the configuration that yielded the lowest accuracy in that work, this distribution might not be the ideal setting for our dataset since the overhead was lower than in the original work. We leave further investigation of the impact of these settings to future work. According to Sirinam *et al.* [35], DF achieved accuracy around 91% against WTF-PAD; Table 10 shows consistent results with the defended WTT-time dataset as well. DF yielded 93% accuracy while p-FP(C) achieved 57% accuracy. The quality of prediction by both DF and p-FP(C) is much higher than against BuFLO and Tamaraw since WTF-PAD exposes the original sizes of most bursts and these remaining traffic patterns are learned by the CNN's filters.

Table 11. The performance of p-FP classifiers against Walkie-Talkie. For Top (T) k analysis, we chose the confidence threshold leading optimal accuracy. Note that Undef means traces, collected without the defense and Def indicates traces, collected under the defense (Bandwidth overhead=24.8%).

	DF		p-FP(M)		p-FP(C)	
Top$k$	Undef	Def	Undef	Def	Undef	Def
T1	86	45±1	81±1	49±1	82±1	48±1
T2	93	66±1	89±1	56±1	83±1	56±1

Walkie-Talkie. Using the dataset provided by Wang and Goldberg [39], we trained and tested p-FP and DF against Walkie-Talkie. Table 11 shows that even though Walkie-Talkie reduced the accuracy of the DF and p-FP classifiers, both classifiers outperform previously known attacks [39]. More surprisingly, both classifiers can reach accuracy of nearly 50% with confidence threshold 0.5. Compared to DF, although p-FP achieved lower accuracy than DF for undefended traces, p-FP had slightly higher Top-1 accuracy against Walkie-Talkie (49% vs. 45%).

Combined with the WTF-PAD results, these experiments show that defenses based only on concealing burst patterns do not sufficiently mitigate against DNN-based WF attacks, and more research is needed to design light-weight defenses for these attacks.

Summary. p-FP classifiers perform more effectively against light-weight defenses than against padding-based defended traces. Compared to DF, p-FP classifiers show comparable or slightly better performance against BuFLO, Tamaraw, and Walkie-Talkie while DF is still the most successful attack against WTF-PAD.

6 Fingerprintability Analysis

As noted previously, websites exhibit varying levels of fingerprintability; since DNN models are particularly powerful fingerprinting tools, we revisit the question of what features influence fingerprintability by these clas-

sifiers. Although Overdorf *et al.* [27] previously found that the fingerprintability of website traffic instances is primarily affected by the size of websites, our analysis focuses on discovering website design features that impact open-world fingerprintability by DNNs. This analysis may also lead to lighter-weight WF defenses based on safer website design principles.

6.1 Dataset and HTML Features

To construct a dataset for FP prediction, we downloaded 29,000 HTML document files from 100 websites ranked in the Alexa top 150, and then extracted the following *HTML features* from these files:

Links and domains. First, we fetched all links and extracted features based on the number of links and domains in a site's HTML DOM. In particular, due to popular usage of Content Delivery Networks (CDNs) and cloud platforms, websites contain many links to resources hosted by these services. In order to show the impact of these links, we extracted the number of links to third party websites. This feature helps understand whether or not downloading web objects (or content) from different web servers makes the network-level traffic pattern more identifiable than when all downloads occur from a single web server.

Tag paths. We also extracted features about each site's *tag paths* as a measure of the site's design complexity. We built the tag paths for a site by scanning a site's DOM and iteratively adding a tag to the current path if it was nested. For example, for a document consisting of tags <html><body><a></body></html>, there are 4 tag paths, <html> (depth=1), <html><body> (depth=2), <html><body><a> (depth=3), and <html><body> (depth=3). Based on this computation, we extracted numerical features including the number of tags in each tag path, the frequency of increase or decrease in the size of tag paths, and the depth of tag paths. Including these features captures the complexity of HTML document structure and allows us to investigate its impact on the fingerprintability of the website.

Tags and other elements. The size of an HTML DOM as well as the website can be estimated based on features computed from the number of tags, attributes, and comments, and the number of characters and words in `data` and `style` attributes. We extracted these features to validate the relationship between the size of a site's HTML DOM and the fingerprintability of the website.

Embedded files. Different types of files are embedded in an HTML DOM and the network traffic associated with fetching those resources may influence the website's vulnerability against WF attacks. Based on the finding that all image and video contents are nested in an `img` tag, we computed the number and the proportion of image and video files and furthermore, we identified specific file extensions to obtain counts and proportions (e.g., jpg, gif, ico, html, etc.). Since these features impact the size of websites, this analysis contributes to a more detailed understanding of the impact of website size on the fingerprintability.

In all, we extracted 62 total features (list in Appendix D), named *HTML features*, from the DOM of each site and we normalized the data by computing the rank for each feature by looking at each column in the matrix. For example, if we had 3 instances, [[3,19,10], [7,10,201], [17,7,25]], we converted those features into the rank information, [[1,3,1], [2,2,3], [3,1,2]] and used these vectors as the inputs to classifiers to determine whether a website is fingerprintable. Note that we used the ratio 50:50 for training and testing set and did such partitioning before we compute the rank for each feature in both sets. For instance, after constructing training and testing data, we replaced each non-normalized feature with its rank in the training or testing set, respectively.

6.2 Predicting Fingerprintability

We evaluated the how the ability of DNNs to fingerprint a website was influenced by our new task-specific feature set based on a site's HTML DOM.

Fingerprintability score. *Fingerprintability* is a measurement of the vulnerability of a website to fingerprinting over Tor. In this paper, our goal is to predict, from its HTML features, whether the Tor network trace of a website w will be fingerprintable by the classifier c. To measure the fingerprintability, we compute the *accuracy* of each website w by training and testing the classifier c using 300 instances each of the 100 top-ranked Alexa web sites, and a single instance each of 20,000 and 40,000 background websites. We used 10 iterations, where each iteration randomly selects training and testing data with the ratio 60:40. Then, based on the WF results of c, we computed the fraction of instances of each site labelled as True Positives as the fingerprintability of the site.

For choices of c, we used k-FP [12], p-FP(M) and p-FP(C), using the same Tor network trace features for all classifiers. If we choose p-FP(C) as c for WF, we derive the FP score by training and testing p-FP(C). If we choose others, we computed the score using that clas-

Table 12. Top 15 HTML features based on the Gini importance when using p-FP(M). (c) means common top features, which also appear in p-FP(C) top features (Table 13).

Rank	Top 15 Features
1	total number of avi files in HTML DOM
2	(c)proportion of ico files in HTML DOM
3	(c)total number of ico files in HTML DOM
4	proportion of image tags over all tag paths
5	(c)total number of links from same domain
6	(c)min depth of tag paths
7	(c)total number of min depth in tag paths
8	(c)std of number of unique tags per a path
9	proportion of html files in HTML DOM
10	median of number of unique tags per a path
11	total number of unique domains in links
12	total number of domains in links
13	proportion of js files in HTML DOM
14	total number of mp3 files in HTML DOM
15	(c)total number of links

Table 13. Top 15 HTML features based on gini index when using PFP(C). Note that (c) means common top features, which appear in PFP(M)'s top 15 features (Table 12).

Rank	Top 15 Features
1	(c)total number of links
2	total number of characters in attribute
3	(c)total number of ico files in HTML DOM
4	toal number of css files in HTML DOM
5	portion of jpg files in HTML DOM
6	(c)proportion of ico files in HTML DOM
7	(c)min depth of tag paths
8	(c)total number of links from same domain
9	total number of max depth in tag paths
10	total number of image tags in HTML DOM
11	total number of positive direction in tag paths
12	total number of js files in HTML DOM
13	(c)std of number of unique tags per a path
14	sum of number of unique tags per a path
15	(c)total number of min depth in tag paths

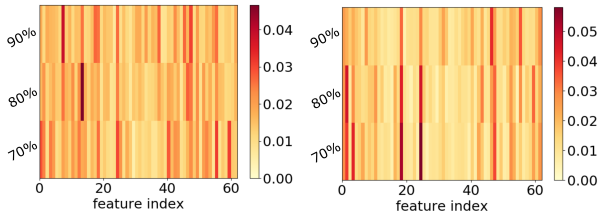
F-Index	1	2	11	19	25	47
$\geq$98%	52258	45999	2137	1450	1450	580
$\leq$35%	489427	439476	725	870	870	1749

Table 14. Most Informative features(F) appearing in both p-FP(M) and p-FP(C), and average value of each feature for more fingerprintable (accuracy $\geq$ 98%), and less fingerprintable (accuracy $\leq$ 35%) websites. Refer to Section D of the Appendix for the description of each feature index in columns.

sifier. Thus, we evaluated each classifier independently to show how its fingerprintability was influenced by our features.

Predicting Fingerprintability After we calculated the score for each w, we labelled the 62-dimensional HTML feature vector for each visit to w, discussed in Section 6.1, with either 1 or 0, depending on whether the corresponding website's *accuracy* was greater than a threshold value t_{fp} (in other words, whether the website is at least t_{fp}-*fingerprintable*) or not, respectively. For instance, when the threshold is 30%, the websites whose accuracy is greater than 30% were labeled *more fingerprintable* and those with an accuracy less than 30% were labeled *less fingerprintable*.

Finally, we attempted to train both MLP and CNN classifiers to *predict* t_{fp}-fingerprintability given the normalized HTML features of a site. We used a 50 : 50 ratio to build training and testing HTML datasets, which were randomly sampled every epoch for 50 epochs. Unfortunately, the success of DNN classifiers led to an imbalanced number of instances for each class (fingerprintable and not fingerprintable). Across different FP thresholds, the level of imbalance varies but was always present to some degree; even with threshold 90% only 2,030 out of 14,500 training examples were labelled "less fingerprintable." Furthermore, we closely looked at two groups of HTML feature vectors whose fingerprintability score was less than 90% (*less*) and greater than 90% (*more*). Most feature vectors in the *less* group had accuracy around 80% and the webpage design features in both groups did not have much statistical difference. As a result, we were not able to train a classifier to make useful predictions with this data set.



(a) p-FP(M) **(b)** p-FP(C)

Fig. 5. The feature importance of the fingerprintability prediction for p-FP(M) and p-FP(C) with the FP threshold (t_{fp}) 70, 80, and 90%.

Informative features. Instead, to gain insight into how more- and less-fingerprintable websites differed in their HTML features, we computed the Gini importance of each HTML feature for each website's t_{fp}-fingerprintability score. Gini importance [8], also called Mean Decrease in Impurity, is widely used as a feature importance measurement. For each feature, the importance score is computed as the sum over the number of splits across all trees in an ensemble that includes

that feature. The improvement in each split criterion at each split is the importance score and it is accumulated over all trees for each variable. To investigate which of our design-level features was most informative, we used Random Forests using the scikit-learn library with importance score derived by the total decrease in node impurity, averaged over all trees in the ensemble. Figure 5 shows that many of the same important features appeared in common across different accuracy thresholds. To summarize those informative HTML features, we computed the sum of the scores for each feature and selected the top 15 features for predicting fingerprintability by p-FP(M) in Table 12 and p-FP(C) in Table 13.

There are several top features that appear in predicting fingerprintability by both p-FP(M) and p-FP(C). For each of these features, we accumulated these feature values and averaged them for two groups of sites, one with accuracy less than 35%, and the other with accuracy greater than 98%. The total number of links (feature 1), total number of third-party links and links pointing to webpages within the same domain were the most important features. As shown in Table 14, these features indicate that less fingerprintable websites carry more embedded web objects and involve traffic relayed to third party websites, which renders the resulting download traffic pattern less distinguishable. In addition, the structure of the webpage, which is represented by tags and tag paths (features 11, 19, and 25), is also among the most informative features for both classifiers. More fingerprintable websites have more complicated webpage design structure than less fingerprintable ones. All of these top common features highly influence fingerprintability across different FP thresholds and classifiers, as shown in Figure 5. These results provide a more detailed view of fingerprintability than the observation by Overdorf et al. [27] that smaller websites are harder to fingerprint.

WF defenses. The ability to predict fingerprintability based on document features suggests a new approach to defense for privacy-aware developers. Based on the most informative features for FP prediction, we can guide developers in designing websites more resistant against traffic analysis by suggesting "safe" ranges for these features. This could prevent the website from leaking the web browsing activity of vulnerable users, or help onion services to better conceal their location.

7 Conclusion

We extensively explored the effectiveness of DNNs in three different applications: automated feature engineering, fingerprinting attacks, and fingerprintability prediction. As a feature extractor, lower dimensional representations learned by an AE, made state-of-the-art WF attacks more effective as well as efficient. For fingerprinting attacks, DNNs performed well across various traffic datasets and different fingerprinting tasks, as well as against recent WF defenses. Lastly, we have shown that several features of a website's HTML-level design influence its fingerprintability by DNN models, leaving the possibility for future work on WF defense using HTML features.

Acknowledgments. We thank our shepherd, Matthew Wright, for his detailed comments and helpful suggestions for the presentation of our results. We also thank Marc Juarez for helpful discussions and assistance with TBC, Vera Rimmer for sharing the dataset, and Payap Sirinam for sharing the DF classifier code. This work was funded by NSF grant 1815757.

References

[1] Tensorflow. https://www.tensorflow.org/.

[2] Tflearn. http://tflearn.org/.

[3] Tor browser crawler. https://github.com/webfp/tor-browser-crawler.

[4] P. Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 37–49, 2012.

[5] Y. Bengio, Y. LeCun, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5):1–41, 2007.

[6] J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International Conference on Machine Learning*, pages 115–123, 2013.

[7] J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International Conference on Machine Learning*, pages 115–123, 2013.

[8] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[9] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 227–238, New York, NY, USA, 2014. ACM.

[10] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *Proceedings of the 2012 IEEE*

Symposium on Security and Privacy, SP '12, pages 332–346, Washington, DC, USA, 2012. IEEE Computer Society.

[11] M. Gardner and S. Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*, 32(14-15):2627–2636, aug 1998.

[12] J. Hayes and G. Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *USENIX Security Symposium*, pages 1187–1203, 2016.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[14] G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786), 2006.

[15] A. Hintz. Fingerprinting websites using traffic analysis. In *International Workshop on Privacy Enhancing Technologies*, pages 171–178. Springer, 2002.

[16] A. Houmansadr, N. Kiyavash, and N. Borisov. Non-blind watermarking of network flows. *IEEE/ACM Transactions on Networking*, 22(4):1232–1244, 2014.

[17] 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*, pages 263–274. ACM, 2014.

[18] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright. Toward an efficient website fingerprinting defense. In *European Symposium on Research in Computer Security*, pages 27–46. Springer, 2016.

[19] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[21] S. Lawrence, C. Giles, Ah Chung Tsoi, and A. Back. Face recognition: a convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, 1997.

[22] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[23] C. Louizos, K. Swersky, Y. Li, M. Welling, and R. Zemel. The variational fair autoencoder. *arXiv preprint arXiv:1511.00830*, 2015.

[24] A. Mousavi, A. B. Patel, and R. G. Baraniuk. A deep learning approach to structured signal recovery. In *Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on*, pages 1336–1343. IEEE, 2015.

[25] M. Nasr, A. Houmansadr, and A. Mazumdar. Compressive traffic analysis: A new paradigm for scalable traffic analysis. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017.

[26] S. E. Oh, S. Li, and N. Hopper. Fingerprinting Keywords in Search Queries over Tor. *Proceedings on Privacy Enhancing Technologies*, 2017(4):171–190.

[27] R. Overdorf, M. Juarez, G. Acar, R. Greenstadt, and C. Diaz. How unique is your. onion?: An analysis of the fingerprintability of tor onion services. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2021–2036. ACM, 2017.

[28] A. Panchenko, F. Lanze, A. Zinnen, M. Henze, J. Pennekamp, K. Wehrle, and T. Engel. Website Fingerprinting at Internet Scale. *16th NDSS (NDSS 16)*, 2016.

[29] 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*. ACM, 2011.

[30] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[31] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen. Automated website fingerprinting through deep learning. In *Network & Distributed System Security Symposium (NDSS)*, 2018.

[32] A. E. Robinson, P. S. Hammon, and V. R. de Sa. Explaining brightness illusions using spatial filtering and local response normalization. *Vision research*, 47(12):1631–1644, 2007.

[33] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[34] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[35] P. Sirinam, M. Imani, M. Juarez, and M. Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018.

[36] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[37] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg. Effective Attacks and Provable Defenses for Website Fingerprinting. *23rd USENIX Security Symposium (USENIX Security 14)*, pages 143–157, 2014.

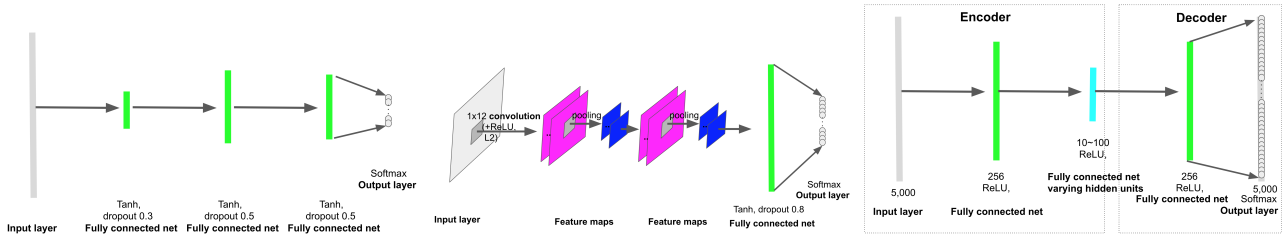
[38] T. Wang and I. Goldberg. Improved website fingerprinting on tor. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 201–212. ACM, 2013.

[39] T. Wang and I. Goldberg. Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks . *26th USENIX Security Symposium (USENIX Security 17)*, 2017.

Appendix

A Deep Neural Network

In this section, we briefly discuss Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), which are methods for supervised learning, and Autoencoder (AE), which is an unsupervised learning method.



(a) Multilayer perceptron **(b)** Convolutional neural network. **(c)** Autoencoder

Fig. 6. Our architectures on 3 DNN models.

Table 15. DNN Hyperparameter tuning using HyperOpt (W: WF, WH: WF with TorHS, K: KF, T:TLS-encrypted traces, Full: Fully-connected layer, Conv: Convolutional layer, and O: Others) .

DNN	MLP				AE		CNN		
HyperParam	Choice			Space	Choice	Space	Choice		Space
input dim	W 5000	WH 2500	K 10k	0∼ 10k	5000	0∼ 5000	T 1600	O 5000	700∼ 5000
optimizer	SGD			SGD,Adam	Adam	SGD,Adam	SGD		SGD,Adam,RMSProp
learning rate	0.03			0.001∼0.1	0.001	0.001∼0.1	0.05		0.001∼0.1
epoch	≤50			10∼1000	10	10∼1000	40∼100		10∼1000
batch	≤50			10∼100	256	10∼300	≤30		28∼128
number of Full	4			2∼7	4	2∼7	2		2∼7
number of Conv	-			-	-	-	2		1∼4
hidden units	1000∼ 3000			500∼10000	200∼ 300	10∼5000	200∼ 300		500∼10000
dropout	0.3∼0.5			0.2∼0.9	-	-	0.8		0.2∼0.9
activation	tanh			tanh, relu, sigmoid	relu	tanh, relu	tanh		tanh, leaky-relu, elu
number of filters	-			-	-	-	128		4∼200
filter size	-			-	-	-	12		2∼16
kernel size	-			-	-	-	10		2∼50

Multilayer Perceptron. An MLP [11, 33], shown in Figure 6a, is a basic neural network, a sort of feed forward network, and also is known as a backpropagation algorithm. It consists of an input layer, one or more fully-connected hidden layers and an output layer. Thus, MLP always has at least 3 layers. In fully-connected layers, all nodes have full connections to all activations in previous layer. Activation functions are computed by a matrix multiplication, followed by a bias offset.

MLP has two procedures, *forward propagation*, which initializes weights and forwards pass through multiple layers to produce the output, and *back propagation*, which calculates errors of the output layer, and then updates weights layer by layer. A single pass through is called an *epoch* and consists of multiple *batches*.

We applied the softmax function to the output layer, which is the generalization of the binary Logistic Regression to multiclass setting. It takes the vector of arbitrary real values and a vector of values in [0,1], where the sum is 1. This real-valued score is a normalized class probability. Since we used a cross entropy to compute the loss, we analyze it as an unnormalized log probability for each class and apply a cross entropy loss, $E{=}-\sum_i^{nClass} t_i log(y_i)$, where i is a class index, $nClass$ is the total number of classes, t_i is a target probability, and y_i is an output probability. The total loss is computed by the mean of E over all training samples.

Convolutional Neural Network. A CNN [21], shown in Figure 6b, consists of one or more convolutional layers, followed by one or more fully connected layers. The forward propagation runs three series of operations, *convolution*, *pooling*, and *classification*. The convolution operation extracts features from the input by learning features using small squares of input, which are called *filters* or *kernels*. That is, we slide filters across the width and height of the input and calculate dot products between entries of the filter and the input to generate a 2D feature map. Sliding different filters over the same feature generates different feature maps and CNN can learn meaningful pattern through this procedure.

Pooling reduces the dimension of the feature maps and thus, the amount of parameters and computation in the network. Max pooling layer operates on selecting the max element from the feature map for resizing spatially. Then, high level features learned by convolutional and pooling layers are fed into MLP for the classification. For the back propagation, it keeps track of the index

of max activation so that routing the gradient becomes simpler than general backward pass.

Autoencoder. An AE [4], shown in Figure 6c, is an unsupervised neural network, which consists of two neural networks, an *encoder*, which learns lower-dimensional data abstractions, and a *decoder*, which recovers the original data. It aims to predict the input by using less number of hidden neurons than input nodes to learn as much information as it can learn to hidden neurons. More specifically, since the number of hidden nodes at each hidden layer is less than the dimension of the original input vector, the network is forced to learn a compressed representation of the input data and then reconstruct the input. Through these procedures, the network can discover interesting structure of the data.

One advantage with an AE is that at the end of training, we can have weights that lead to the hidden layer (blue layer in Figure 6c), we can train using certain inputs. Furthermore, when we meet other data later, we can reduce its dimensionality using those weights without retraining.

Thus, it provides benefits to reduce the feature dimensionality for data visualization and reducing the noise in the data. Hidden units in an encoder keep as much information as it can while denoising the data. Moreover, we can elaborate feature extraction using encoded data through the cost function since we have a lot of choice on the cost function and can adjust the weight for each class and sample. We can use this power to reflect certain phenomenons in the dataset, eventually leading to a more efficient and meaningful data representation. In Section 4.2, we focus on the functionality of dimensionality reduction [14] and use an MLP for an *encoder* and a *decoder*, while varying the number of *hidden units* in a hidden layer of an encoder.

To construct a more generative model, Variational AE (VAE) was introduced by Kingma and Welling [19] and Rezende *et al.* [30]. Instead of memorizing a fuzzy data structure, it generates latent vectors following a Gaussian distribution by forcing a constraint to an encoder. Subsequently, to compute the loss of a VAE, two types of losses must be considered, the error between the input and reconstructed data, and the loss between latent variables and a unit Gaussian, reflected by KL divergence. Training VAE is tricky due to the trade off between these two different losses. Improvement in the generalization also promotes the quality of data reconstruction by a decoder.

Avoid overfitting. DNN usually struggles with overfitting, which means that the network memorized the training samples and hence, the error in testing dataset is large even though the training loss is tiny. To overcome this issue, dropout [36] and regularization are widely used. The dropout temporarily removes units in layers based on the probability of each unit to be retained. Regularization is also known as weight decay, which means that it penalizes large weights based on constraints on their squared values (L2) or absolute values (L1). We applied these techniques to both MLP and CNN architectures in Section 3.2.

B Hyperparameter Tuning

We report all choices of hyperparameters used in the experiments throughout the paper in Table 15

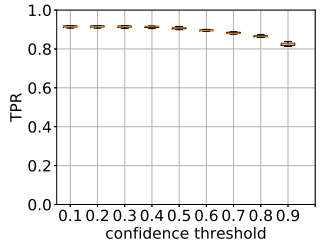
C Impact of Data Collection

We modified TBC to reset the Tor process after each visit, due to the possible misuse of entry guards in the default setting of TBC. To assess the potential impact of this change, which eventually resulted in higher variance across each network traffic instance due to the growing time gap between collections, we also collected some data without this modification. When using a small scale dataset, comprised of 4,500 monitored and 10,000 unmonitored traces, this did not significantly change the performance of p-FP(M), as shown in Figure 7. However, as we increase the size of the training dataset up to 30,000 monitored traces and 20,000 unmonitored traces, this setting worsens the performance of DNN classifiers, giving a 5% drop in TPR using p-FP(M) and a 2% drop using p-FP(C).

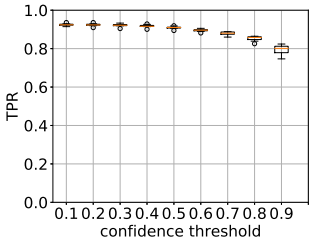
D HTML Features

We list 62 feature (TPath: Tag Path, #: number).

- 1. total # of links
- 2. total # of links from same domain
- 3. total # of third party links
- 4. total # of domains in links
- 5. total # of unique domains in links
- 6. total # of TPaths
- 7. total # of unique TPaths
- 8. sum of # of unique tags per a path
- 9. median of # of unique tags per a path
- 10. mean of # of unique tags per a path
- 11. std of # of unique tags per a path
- 12. total # of change of TPath direction (if depth increases, positive, otherwise, negative)
- 13. total # of non change of TPath direction
- 14. total # of positive direction in TPaths
- 15. total # of negative direction in TPaths
- 16. total sum of tag depths



(a) TBC default setting



(b) TBC new setting

Fig. 7. MLP performance according to different dataset collection (TBC: Tor Browser Crawler).

- 17. std of tag depths
- 18. total # of max depth in TPaths
- 19. total # of min depth in TPaths
- 20. total # of median depth in TPaths
- 21. total # of rounded mean depth in TPaths
- 22. total # of 30% percentile of depth in TPaths
- 23. total # of 70% percentile of depth in TPaths
- 24. max depth of TPaths
- 25. min depth of TPaths
- 26. median depth in TPaths
- 27. rounded mean depth in TPaths
- 28. 30% percentile of depth in TPaths
- 29. 70% percentile of depth in TPaths
- 30. total # of tags
- 31. total # of unique tags
- 32. total # of comments
- 33. total # of attributes
- 34. total # of unique attributes
- 35. total # of characters
- 36. total # of characters in script tag
- 37. total # of characters in style attribute
- 38. total # of characters in attribute
- 39. total # of characters in data including those in script and style attributes
- 40. total # of characters in data attribute
- 41. total # of words in data including those in script and style attributes
- 42. total # of words in data attribute
- 43. total # of image tags in HTML DOM
- 44. proportion of image tags over all TPaths
- 45. total # of png files in HTML DOM
- 46.proportion of png files in HTML DOM
- 47. total # of ico files in HTML DOM
- 48. proportion of ico files in HTML DOM
- 49. total # of jpg files in HTML DOM
- 50. proportion of jpg files in HTML DOM
- 51. total # of gif files in HTML DOM
- 52. proportion of gif files in HTML DOM
- 53. total # of bmp files in HTML DOM
- 54. proportion of bmp files in HTML DOM
- 55. total # of html files in HTML DOM
- 56. proportion of html files in HTML DOM
- 57. total # of css files in HTML DOM
- 58. proportion of css files in HTML DOM
- 59. total # of js files in HTML DOM
- 60. proportion of js files in HTML DOM
- 61. total # of mp3 files in HTML DOM
- 62. total # of avi files in HTML DOM