A First Look at Generating Website Fingerprinting Attacks via Neural Architecture Search

Prabhjot Singh[*]
prabhjot.singh@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Shreya Arun Naik[*]
s4an@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Navid Malekghaini
nmalekgh@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Diogo Barradas
diogo.barradas@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Noura Limam
noura.limam@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

ABSTRACT

An adversary can use website fingerprinting (WF) attacks to breach the privacy of users who access the web through encrypted tunnels like Tor. These attacks have increasingly relied on the use of deep neural networks (DNNs) to build powerful classifiers that can match the traffic of a target user to the specific traffic pattern of a website.

In this paper, we study whether the use of neural architecture search (NAS) techniques can provide adversaries with a systematic way to find improved DNNs to launch WF attacks. Concretely, we study the performance of the prominent AutoKeras NAS tool on the WF scenario, under a limited exploration budget, and analyze the effectiveness and efficiency of the resulting DNNs.

Our evaluation reveals that AutoKeras’s DNNs achieve a comparable accuracy to that of the state-of-the-art Tik-Tok attack on undefended Tor traffic, and obtain 5–8% accuracy improvements against the FRONT random padding defense, thus highlighting the potential of NAS techniques to enhance the effectiveness of WF.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; • **Networks → Network privacy and anonymity**; • **Computing methodologies → Neural networks**.

KEYWORDS

deep learning; neural architecture search; website fingerprinting

ACM Reference Format:
Prabhjot Singh, Shreya Arun Naik, Navid Malekghaini, Diogo Barradas, and Noura Limam. 2023. A First Look at Generating Website Fingerprinting Attacks via Neural Architecture Search. In *Proceedings of the 21st Workshop on Privacy in the Electronic Society (WPES ’23), November 26, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3603216.3624961

[*]Authors contributed equally.

1 INTRODUCTION

To safeguard their online privacy, users can use encrypted tunnels like those created by low-latency anonymity networks such as Tor [8]. These tunnels allow users to create multi-hop encrypted pathways that conceal the contents and destination of their communications from eavesdroppers. However, Tor does not significantly modify the timing and volume characteristics of packet exchanges, thus retaining the traffic patterns that are characteristic of the websites visited over the encrypted tunnel [34], a.k.a., a fingerprint. This design choice makes Tor susceptible to a class of traffic analysis attacks known as *website fingerprinting* (WF) [1, 19, 36, 37, 39, 41, 46], by which eavesdroppers can disclose the websites users visit through Tor by applying machine learning techniques that match the users’ traffic patterns to pre-recorded fingerprints.

In recent years, WF research has shifted from using traditional machine learning techniques [19, 36] to launching more precise attacks that use deep neural networks (DNNs) [1, 37, 39, 41]. These attacks (and further developments thereof) rely on the ability to discover increasingly effective DNN topologies. However, the search for improved DNN architectures that can fuel WF attacks has been primarily conducted through a trial-and-error approach, where researchers adapt or extend existing architectures that work well in related domains (e.g., computational vision [20] and biology [18], or encrypted traffic classification [29]). Unfortunately, this approach is afflicted by two important drawbacks.

First, although hyperparameter optimization methods [31, 43] can help to adjust the parameters of a promising DNN (e.g., learning rate, dropout, etc.), researchers lack a systematic way to understand the factors that lead to better performance among different DNN topologies (e.g., number and arrangement of layers and building blocks). Without a clear understanding of how specific architectural changes affect a DNN’s effectiveness, researchers may struggle to make informed decisions about which architectural elements to use in their DNN (and how to combine them), being limited to the use of heuristics [42, 45] or *pure chance* for finding better architectures.

Second, since the development of improved WF attacks is often based on existing DNN architectures, researchers may overlook unconventional or innovative architectural choices, thus missing potential breakthroughs in designing highly effective DNNs.

In this paper, we present a preliminary study towards answering the question of whether WF researchers can adopt more systematic approaches to unlock the untapped potential of the neural network design space, towards developing more effective WF attacks. To this end, we leverage a tool developed within the context of *neural architecture search* (NAS) [10, 38], an evolving sub-field of automated machine learning (AutoML) [13, 23]. NAS focuses on building and/or optimizing the topology of DNNs, and has seen a growing momentum in the machine learning community. Specifically, we use the prominent AutoKeras [24] NAS tool for performing an automatic exploration of DNN topology configurations (within a configurable exploration budget), and assess the effectiveness of the resulting DNNs in performing accurate WF attacks.

The results of our evaluation show that DNNs created from scratch by AutoKeras (under a limited exploration budget of 100 trials) achieve a comparable accuracy to that of the state-of-the-art Tik-Tok [37] attack on undefended Tor traffic, albeit at the cost of larger model sizes and/or inference times. In addition, we observe that AutoKeras is able to develop DNN models which are more effective than Tik-Tok on defended Tor traffic. For instance, AutoKeras' models enable an accuracy increase between 5–8% when applied to traces protected by the recent FRONT [14] defense. In Section 5, we detail a number of steps towards shaping a full-scale study for grounding the benefits of NAS in the WF domain.

2 BACKGROUND

2.1 Website Fingerprinting

In a typical WF attack on Tor, the adversary intercepts communications between the user and the entry node of the Tor circuit. To prepare the attack, the adversary repeatedly accesses target websites, collects network traffic traces, and extracts attributes characterizing these traces to create a fingerprint database. Then, the adversary builds a model for website prediction. Finally, the adversary extracts the fingerprint of a target user's website access via Tor and uses its pre-trained model to identify the visited website.

Attack settings. In this work, we focus on WF attacks in the *closed-world* setting, where the target user is assumed to access a website amongst a predefined set of websites known to the adversary. Our plans for future work include the use of NAS constructs to explore improved attacks in the more realistic *open-world* setting, where users are assumed to be able to visit websites unknown to the adversary, besides the set of websites the adversary had monitored.

Attacks and defenses. Earlier WF attacks used manually crafted features to train classical machine learning classifiers and fingerprint website visits over Tor connections [19, 36, 46]. Recently, research on WF has turned to deep learning. The *Automated WF* [39] and *Deep Fingerprinting (DF)* [41] attacks use packet direction information as input to DNN models. Subsequently, the *Tik-Tok* [37] attack enhanced DF model's results by using the dot product of direction and timing vectors as input. The *Var-CNN* [1] attack used semi-automated feature extraction and Residual Networks [20] to improve DNN models' effectiveness on the WF context.

Website fingerprinting defenses aim to prevent successful WF attacks by obfuscating website traces. By transmitting packets at fixed rates, constant-rate padding defenses like *CS-BuFLO* [3], and *Tamaraw* [4] mask timing patterns and packet transmission burst behaviour. More efficient adaptive and randomized padding defenses like *WTF-PAD* [27] and *FRONT* [14] insert dummy packets to conceal time gaps between packets, towards making accesses to different websites indistinguishable. While we focus on the above padding-centric defenses in this study, we refer the reader to a more complete analysis of the WF defenses' space [30].

2.2 AutoML and Neural Architecture Search

Automated Machine Learning (AutoML) streamlines the traditional machine learning workflow by resorting to different optimization techniques to automate feature engineering, model selection, hyperparameter tuning, and model evaluation. Due to its ability to find highly effective models, AutoML has been previously applied to the traffic analysis domain [9, 22] and, in particular, to WF [17], revealing promising results. However, as described previously, the performance of the resulting "traditional" classifiers has been found to be subpar when compared to deep learning in the context of WF.

While multiple approaches exist to optimize a DNN's hyperparameters [50], these techniques do not address the core challenge of finding the best architecture for issuing predictions on a given task [10]. Fortunately, to meet the growing demand for accessible deep learning, the machine learning community has undertaken a concerted effort in advancing Neural Architecture Search (NAS), the process of automating DNN architecture engineering by assembling various basic operations and building blocks chosen from a predefined search space [21]. NAS schemes comprise three essential components: the definition of a search space for neural architectures, a set of architecture optimization methods (also known as search strategies), and a suite of model evaluation methods [10].

For conducting our study, we choose AutoKeras [24, 25], an open-source NAS system that leverages Bayesian optimization to guide DNN morphism. AutoKeras focuses on deep learning tasks, setting it apart from tools that focus on shallow neural network models [13, 35]. Other open-source alternatives like AutoGluon [11] and H2O [7] have their own shortcomings, like fixed-sized pipelining of operators and a more limited search space for neural network architectures. Moreover, AutoML services hosted on large cloud platforms [2, 33] encounter obstacles that hinder user adoption. Indeed, services like Google's VertexAI [16] or Azure's AutoML [32] are aimed at enterprise customers and involve steep monetary expenses, making them less accessible to independent researchers.

3 METHODOLOGY

We now describe our laboratory setup and the experimental design we followed to shed light on whether NAS techniques can help fuel the development of more *effective* and *efficient* WF attacks.

3.1 Laboratory Setup

AutoKeras configuration. We bootstrap AutoKeras' DNN explorations in two different ways, either by a) starting to explore a space of possible DNN typologies from scratch, or b) starting to explore other DNN typologies loosely based on a given DNN architecture.

To allow AutoKeras to explore DNN architectures from scratch, we limited ourselves to specify the input and output layers of the DNN. Specifically, we set an *image-input* and a *classification-head* block. The choice of *image-input* informs AutoKeras to generate DNN architectures inspired by the computer vision domain, which have been shown to produce successful results for network traffic classification [29, 41]. In turn, to match the overall topology of DF, we specifically laid out a *conv-block*, followed by a *dense-block*, and a *classification-head.* This mimics the general architecture of state-of-the-art WF attacks, by generating a block driven by convolutional neural networks that is then connected to a block of densely connected neurons, before attempting classification.

For each bootstrapping method, we run Autokeras with three of its different *tuners*, i.e., hyperparameter optimizers: a) a greedy tuner, that combines random search and a greedy algorithm; b) a bayesian tuner, based on Gaussian process models [43], and; c) the hyperband tuner, based on bandit algorithms [28].

Dataset. We conduct our experiments with the closed-world DS-19 dataset [14], consisting of 100 unique website traces where each website is visited 100 times via Tor. Website traces contain directional and timing data related to Tor cells (estimated from Tor packets [47]), and each trace is either truncated to 5000 cells or padded with zeros if its original length is shorter. We split the data into 80% training, 10% validation, and 10% testing sets, and ensured that every model would be trained in the same split of the data.

Attacks and defenses. In our study, we compare the effectiveness and efficiency of the DNN models discovered by AutoKeras with the Tik-Tok [37] WF attack. We chose Tik-Tok as it is typically used as the benchmark attack that new WF defenses' proposals should be able to defend against [30] and because of its reliance on full trace information (both directional and timing-related data). We use the default Tik-Tok architecture (i.e., the DF model with directional-timing vectors as input), and parameters in our experiments.

Inspired by Veicht et al. [44], we assess the effectiveness of Tik-Tok and AutoKeras-generated models on a set of relevant WF defenses which are accompanied by high-fidelity simulators [15]. We use these simulators to generate defended traffic from the pre-recorded undefended traces contained in the DS-19 dataset. Concretely, we make use of the WTF-PAD, FRONT, CS-BuFLO, and Tamaraw defenses (see Appendix A.1 for each defense's setup).

Laboratory testbed. For our experiments, we relied on 3 machines, each configured with a 3.23 GHz AMD EPYC 7302 16-Core Processor, an NVIDIA A100 40 GB GPU and 74 GB RAM. We used these machines to train the Tik-Tok and AutoKeras-generated models.

3.2 Experimental Design

We configured AutoKeras to use an exploration budget of 30 or 100 trials, i.e., AutoKeras tries 30 or 100 different combinations of DNN topologies and hyperparameter configurations before outputting the best model found, as guided by each of the tuners we used. To select the number of epochs to train each model, we started by inspecting the convergence of the Tik-Tok model on the DS-19 dataset, observing that it would converge after about 30 epochs (see Figure 2 in Appendix A.2). Thus, we opted to train AutoKeras' models in two settings: a) for the same number of epochs (30), and b) for 50 epochs, for understanding whether AutoKeras' models could benefit from additional training time. As AutoKeras' image-input node required a square vector as input, we reshaped our feature vectors to be of size 71x71 (5041 cells), padding with trailing zeros.

Metrics. We use *accuracy* to compare the effectiveness of the AutoKeras-generated models with that of the Tik-Tok WF attack. Then, to gauge the efficiency of AutoKeras-generated DNNs, we measure the models' *number of parameters* and the models' *inference time* per batch (32 samples each). We also collect information on the models' *training time*, allowing us to gauge how much time an adversary would be required to spend when, for instance, periodically retraining models to mitigate concept drift [6, 26].

Table 1: Model comparison on undefended traces (after exploration). Unless stated, AutoKeras uses the greedy tuner.

Model	Trials	Epochs	Accuracy	Train. Time (s/epoch)	Inf. Time (ms/batch)	Total Parameters
Tik-Tok	—	30	0.966	3.0	4	3,985,444
Tik-Tok	—	**50**	**0.968**	**3.0**	**4**	**3,985,444**
in-conv-dense	30	30	0.633	1.4	2	10,234,167
in-conv-dense	30	50	0.891	1.7	3	2,557,639
in-conv-dense	**100**	**30**	**0.933**	**3.1**	**4**	**20,578,953**
in-conv-dense	100	50	0.921	2.0	2	10,536,983
image-input	30	30	0.931	14.9	16	23,792,615
image-input	30	50	0.961	16.2	20	23,792,615
image-input	100	30	0.956	26.9	31	42,863,079
image-input	**100**	**50**	**0.965**	**17.9**	**19**	**23,792,615**
w/ bayes. opt.	100	50	0.954	18.9	18	21,066,383
w/ hyperband	100	50	0.919	17.4	21	42,831,460

4 EXPERIMENTAL RESULTS

Next, we assess the ability of AutoKeras models' to fingerprint undefended (Section 4.1) and defended (Section 4.2) Tor traffic.

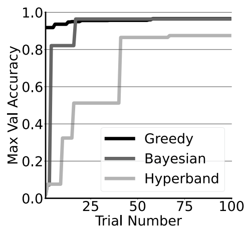
4.1 Attacking Undefended Tor Traces

Table 1 depicts a comparison of the Tik-Tok model with the most accurate models generated after each AutoKeras exploration, for a given combination of epochs (30 or 50) and trials (30 or 100), over the undefended Tor traces included in the DS-19 dataset.

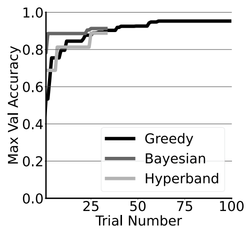
AutoKeras models obtain similar accuracy v.s. Tik-Tok. The table shows that the most accurate models produced by AutoKeras (in bold) obtain a similar accuracy to that of Tik-Tok on undefended Tor traces. For instance, the most accurate AutoKeras model based on *in-conv-dense* achieved an accuracy of 93.3%, while the most accurate AutoKeras model based on *image-input* achieved an accuracy of 96.5%. While these two models are comparable in the number of total parameters (~20M), we see that the best *image-input* model achieved an accuracy which is only 0.3% away from that obtained by Tik-Tok when trained for the same number of epochs.

The greedy tuner allows AutoKeras to generate more accurate models within our maximum exploration budget. We can also see from the table that the use of a greedy tuner allows AutoKeras to build more accurate models for the maximum number of trials we considered in our experiments (trials = 100). For instance, while *image-input* achieves an accuracy of 96.5% when using the greedy tuner, the bayesian and hyperband tuners reach an accuracy of 95.4% and 91.9%, respectively. Since the literature suggests that the use of the two latter tuners should result in better models than a greedy approach [12], it may be the case that AutoKeras must be run for a larger number of trials until improvements are noticeable.

Figure 1(a) and Figure 1(b) depict the maximum accuracy obtained by each variant of the AutoKeras' models (when using different tuners) as the number of trials increase. Interestingly, Figure 1(a) shows that the greedy tuner allows *image-input* to obtain a larger accuracy for a smaller number of trials. Indeed, the bayesian tuner can only produce a model as accurate as the greedy tuner after 20 trials, while the hyperband tuner is only able to produce a model



(a) *image-input* (50 epochs).



(b) *in-conv-dense* (30 epochs).

Figure 1: Validation acc. on undefended traces (100 trials).

which is ~10% less accurate than the one produced using the greedy tuner, for our limited exploration budget.

For *in-conv-dense*, we can observe in Figure 1(b) that the bayesian tuner generally allows for more accurate models, being approximated by models generated via the greedy tuner after trials = 20. We note, however, that the exploration guided by the bayesian and hyperband tuners resulted in *out of memory* errors in AutoKeras from trial = 30 onwards, precluding us from obtaining additional models. These errors may stem from AutoKeras' inability to accurately estimate the available GPU memory in each trial [24].

***in-conv-dense* is faster than *image-input*.** Table 1 shows that the most accurate model produced with the *in-conv-dense* layout is both faster to train and faster to issue predictions than the ones generated with the *image-input* layout, approximating the 3s per-epoch training times and 4ms per-batch prediction times obtained by Tik-Tok. Even though the most accurate *in-conv-dense* model is significantly larger than Tik-Tok (an extra ~16M params.) and comparable in size to the most accurate *image-input* model (within ~3M params.), it can issue predictions at the same rate as Tik-Tok. This result suggests that providing pre-existing DNN layouts to AutoKeras may help balance the accuracy and performance of the obtained models. Appendix A.3 includes a closer comparison of the DNN layout of the best *in-conv-dense* and *image-input* models.

Overall, AutoKeras' *in-conv-dense* models enable a faster training and prediction time than Tik-Tok, albeit at the cost of effectiveness (e.g., see *in-conv-dense*, 50 epochs, 100 trials). These models may be useful in scenarios where adversaries wish to issue predictions at faster paces and can tolerate larger misclassification rates.

4.2 Attacking Defended Tor Traces

To experiment with AutoKeras' ability to build accurate models targetting defended Tor traffic, we conducted a set of experiments where we select the most effective AutoKeras configurations for undefended traces (⟨*image-input*, 50 epochs, 100 trials⟩ and ⟨*in-conv-dense*, 30 epochs, 100 trials⟩ with a greedy tuner), and run a new exploration over each of the defended datasets.

Table 2 reveals that *image-input* models can obtain an accuracy which is comparable to Tik-Tok for adaptive padding (WTF-PAD) and the more secure constant-rate (CS-BuFLO and Tamaraw) defenses. Perhaps more interestingly, *image-input* models generated over the traces of the two random padding defense (FRONT) variants reach an accuracy of 86.3% and 75.9%, respectively, representing an increase of 5.6% and 8.0% when compared to Tik-Tok. Similarly to the models built over undefended traces, *image-input* models tend to be larger in size (up to ~14×) and take from 2.5 to 9× longer to issue predictions (see Appendix A.4 – Table 3).

Table 2: Accuracy comparison on defended DS-19 traces.

Defense	Tik-Tok	AutoKeras *in-conv-dense* ⟨100trials, 30epochs⟩	AutoKeras *image-input* ⟨100trials, 50epochs⟩
WTF_PAD	0.920	0.642	0.918
FRONT_T1	0.807	0.462	0.863
FRONT_T2	0.679	0.207	0.759
CS_BuFLO	0.116	0.094	0.117
Tamaraw	0.137	0.101	0.0730

Second, we can observe that the accuracy of AutoKeras' *in-conv-dense* models on defended traffic is substantially lower than that achieved by Tik-Tok. This is especially noticeable when considering the FRONT defense, where *in-conv-dense* performs 34–47% worse. Our analysis of validation accuracy as exploration trials proceed for defended traces (Appendix A.5 – Figure 4) suggests that additional explorations may be required for AutoKeras to generate *in-conv-dense* models whose effectiveness is comparable to Tik-Tok's.

Overall, our preliminary results highlight the potential of NAS to enhance WF attacks, encouraging us to develop a full-scale study.

5 LIMITATIONS AND FUTURE WORK

A single NAS tool. Our preliminary study solely focused on AutoKeras to extract insights over the applicability of NAS to the WF domain. We aim to benchmark other prominent NAS tools, such as BANANAS [49] or ZARTS [48] as part of our future work.

A small exploration budget. Our explorations with AutoKeras were limited to 100 trials. For our most complex AutoKeras configurations (i.e., produced via *image-input*), each 100 trials' exploration could be run in ~16 hours. We aim to assess the performance of AutoKeras (and other NAS systems) with larger exploration budgets.

An accuracy-centric optimization. Our explorations focused accuracy as the single metric to optimize. We plan to assess the effectiveness and efficiency of AutoKeras' models when optimizing for a combination of metrics, such as accuracy and inference time.

A single dataset and trace representation. We aim to consider other datasets in the WF literature to assess the generalizability of our findings, and to assess whether NAS tools can yield improved results when exposed to larger datasets (e.g., AWF [39], BigEnough [30]) that approximate the scale of those currently used for benchmarking NAS tools [10]. Further experiments include the use of recent trace representations [40] and the assessment of AutoML-learned architectures' robustness against concept drift [6, 26].

Lack of open-world experiments. Our study focused WF in the closed-world setting. In the future, we aim to analyze the effectiveness of NAS-generated DNNs in the open-world WF setting.

6 CONCLUSIONS

This preliminary study explores the use of NAS techniques to enhance WF attacks on Tor. The results of our evaluation, resorting to the AutoKeras NAS tool, reveal that the generated DNNs can achieve comparable accuracy to the Tik-Tok attack on undefended Tor traffic, and achieve higher accuracy on traffic defended via random padding schemes. Our future work includes the benchmarking of a larger set of NAS tools and the expansion of our exploration budget to better assess the benefits of NAS within the WF context.

ACKNOWLEDGMENTS

This work was supported in part by NSERC under grant DGECR-2023-00037. We gratefully acknowledge our use of computational resources from the Cybersecurity and Privacy Institute (CPI) at the University of Waterloo.

REFERENCES

[1] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. 2019. Var-CNN: A data-efficient website fingerprinting attack based on deep learning. *Proceedings on Privacy Enhancing Technologies* Vol. 2019, 4, 292–310.

[2] Ekaba Bisong and Ekaba Bisong. 2019. Google automl: cloud vision. *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners* (2019), 581–598.

[3] Xiang Cai, Rishab Nithyanand, and Rob Johnson. 2014. Cs-buflo: A congestion sensitive website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society.* 121–130.

[4] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. 2014. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security.* 227–238.

[5] Giovanni Cherubin. 2017. Bayes, not Naïve: Security Bounds on Website Fingerprinting Defenses. *Proceedings on Privacy Enhancing Technologies* Vol. 4, 135–151.

[6] Giovanni Cherubin, Rob Jansen, and Carmela Troncoso. 2022. Online website fingerprinting: Evaluating website fingerprinting attacks on Tor in the real world. In *Proceedings of the 31st USENIX Security Symposium.* 753–770.

[7] Darren Cook. 2016. *Practical machine learning with H2O: powerful, scalable techniques for deep learning and AI.* " O'Reilly Media, Inc.".

[8] Roger Dingledine, Nick Mathewson, Paul F Syverson, et al. 2004. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, Vol. 4. 303–320.

[9] Priyanka Dodia, Mashael AlSabah, Omar Alrawi, and Tao Wang. 2022. Exposing the Rat in the Tunnel: Using Traffic Analysis for Tor-Based Malware Detection. In *Proceedings of the 2022 SIGSAC Conference on Computer and Communications Security.* 875–889.

[10] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *The Journal of Machine Learning Research* Vol. 20, 1 (2019), 1997–2017.

[11] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. 2020. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. *arXiv preprint arXiv:2003.06505* (2020).

[12] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and efficient hyperparameter optimization at scale. In *International conference on machine learning.* Proceedings of Machine Learning Research (PMLR), 1437–1446.

[13] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. *Advances in neural information processing systems* Vol. 28 (2015).

[14] Jiajun Gong and Tao Wang. 2020. Zero-delay lightweight defenses against website fingerprinting. In *Proceedings of the 29th USENIX Security Symposium.* 717–734.

[15] Jiajun Gong, Wuqi Zhang, Charles Zhang, and Tao Wang. 2021. WFDefProxy: Modularly Implementing and Empirically Evaluating Website Fingerprinting Defenses. *Computing Research Repository (CoRR)* Vol. abs/2111.12629 (2021).

[16] Google. 2023. https://cloud.google.com/vertex-ai/docs/training/neural-architecture-search/overview. Last Accessed: 2023-07-23.

[17] Sonia Gu. 2022. *Leveraging Automated Machine Learning for Website Fingerprinting.* Master's thesis. Princeton University - Electrical and Computer Engineering.

[18] Ankit Gupta and Alexander M Rush. 2017. Dilated convolutions for modeling long-distance genomic dependencies. *arXiv preprint arXiv:1710.01278* (2017).

[19] Jamie Hayes, George Danezis, et al. 2016. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *Proceedings of the 25th USENIX Security Symposium.* 1187–1203.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the 29th IEEE conference on computer vision and pattern recognition.* 770–778.

[21] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems* Vol. 212 (2021), 106622.

[22] Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. 2021. New directions in automated traffic analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security.* 3366–3383.

[23] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. 2019. *Automated machine learning: methods, systems, challenges.* Springer Nature.

[24] Haifeng Jin, François Chollet, Qingquan Song, and Xia Hu. 2023. AutoKeras: An AutoML Library for Deep Learning. *Journal of Machine Learning Research* Vol. 24, 6 (2023), 1–6.

[25] Haifeng Jin, Qingquan Song, and Xia Hu. 2018. Efficient Neural Architecture Search with Network Morphism. *CoRR* abs/1806.10282 (2018). arXiv:1806.10282

[26] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. 2014. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security.* 263–274.

[27] Marc Juarez, Mohsen Imani, Mike Perry, Claudia Diaz, and Matthew Wright. 2016. Toward an efficient website fingerprinting defense. In *Proceedings of the 21st European Symposium on Research in Computer Security, 2016.* 27–46.

[28] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The journal of machine learning research* Vol. 18, 1 (2017), 6765–6816.

[29] Navid Malekghaini, Elham Akbari, et al. 2023. Deep learning for encrypted traffic classification in the face of data drift: An empirical study. *Computer Networks* Vol. 225 (2023), 109648.

[30] Nate Mathews, James K Holland, Se Eun Oh, Mohammad Saidur Rahman, Nicholas Hopper, and Matthew Wright. 2023. SoK: A critical evaluation of efficient website fingerprinting defenses. In *Proceedings of the 44th IEEE Symposium on Security and Privacy.* 969–986.

[31] Pedro Mendes, Maria Casimiro, Paolo Romano, and David Garlan. 2023. HyperJump: Accelerating HyperBand via Risk Modelling. *Proceedings of the AAAI Conference on Artificial Intelligence* Vol. 37, 8 (2023), 9143–9152.

[32] Microsoft. 2023. https://azure.microsoft.com/en-ca/products/machine-learning/automatedml. Last Accessed: 2023-07-23.

[33] Deepak Mukunthu, Parashar Shah, and Wee Hyong Tok. 2019. *Practical automated machine learning on Azure: using Azure machine learning to quickly build AI solutions.* O'Reilly Media.

[34] Steven J Murdoch and George Danezis. 2005. Low-cost traffic analysis of Tor. In *Proceedings of the 26th IEEE Symposium on Security and Privacy.* 183–195.

[35] Randal S Olson and Jason H Moore. 2016. TPOT: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning.* Proceedings of Machine Learning Research (PMLR), 66–74.

[36] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. 2016. Website Fingerprinting at Internet Scale. In *Proceedings of the 23rd Network and Distributed System Security Symposium.*

[37] Mohammad Saidur Rahman, Payap Sirinam, Nate Mathews, Kantha Girish Gangadhara, and Matthew Wright. 2020. Tik-Tok: The utility of packet timing in website fingerprinting attacks. In *Proceedings on Privacy Enhancing Technologies (PoPETs)*, Vol. 2020. 5–24. Issue 3.

[38] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. 2021. A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions. *Comput. Surveys* Vol. 54, 4, Article 76 (2021).

[39] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. 2018. Automated website fingerprinting through deep learning. In *Proceedings of the 25th Network and Distributed Systems Security Symposium.*

[40] Meng Shen, Kexin Ji, Zhenbo Gao, Qi Li, Liehuang Zhu, and Ke Xu. 2023. Subverting Website Fingerprinting Defenses with Robust Traffic Representation. In *Proceedings of the 32nd USENIX Security Symposium.* 607–624.

[41] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. 2018. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security.* 1928–1943.

[42] Leslie N. Smith and Nicholay Topin. 2016. Deep Convolutional Neural Network Design Patterns. arXiv:1611.00847 [cs.LG]

[43] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems* Vol. 25 (2012).

[44] Alexander Veicht, Cedric Renggli, and Diogo Barradas. 2023. DeepSE-WF: Unified Security Estimation for Website Fingerprinting Defenses. *Proceedings on Privacy Enhancing Technologies* Vol. 2 (2023), 188–205.

[45] Steven Walczak and Narciso Cerpa. 1999. Heuristic principles for the design of artificial neural networks. *Information and Software Technology* 41, 2 (1999), 107–117.

[46] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective Attacks and Provable Defenses for Website Fingerprinting. In *Proceedings of the 23rd USENIX Security Symposium.* 143–157.

[47] Tao Wang and Ian Goldberg. 2013. Improved website fingerprinting on Tor. In *Proceedings of the 12th ACM Workshop on Privacy in the Electronic Society.* 201–212.

[48] Xiaoxing Wang, Wenxuan Guo, Jianlin Su, Xiaokang Yang, and Junchi Yan. 2022. ZARTS: On Zero-order Optimization for Neural Architecture Search. In *Advances in Neural Information Processing Systems*, Vol. Vol. 35. 12868–12880.

[49] Colin White, Willie Neiswanger, and Yash Savani. 2021. BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search. *Proceedings of the Advancement of Artificial Intelligence (AAAI) Conference on Artificial Intelligence* Vol. 35, 12 (2021), 10293–10301.

[50] Tong Yu and Hong Zhu. 2020. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689* (2020).

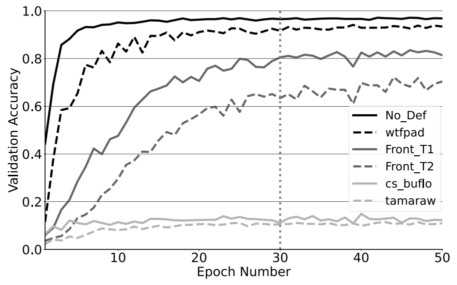
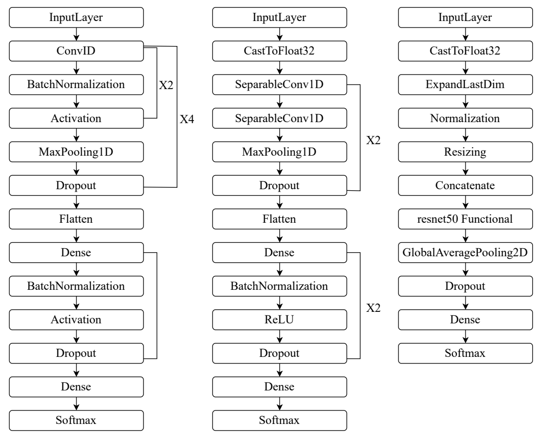


Figure 2: Validation acc. of Tik-Tok.

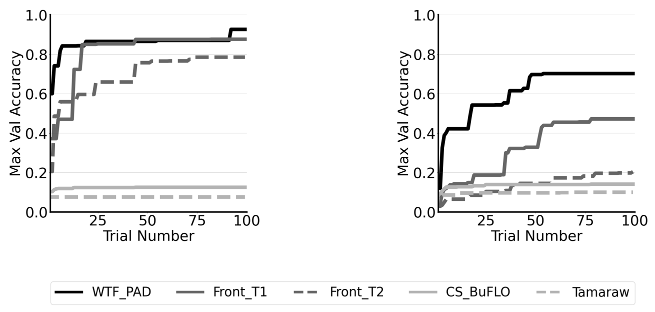


a) Tik-Tok *b) in-conv-dense* *c) image-input*

Figure 3: Comparison of the most accurate models.

Table 3: Inference time (per batch) and model size comparison (number of parameters) on defended DS-19 traces.

Defense	Tik-Tok	*in-conv-dense* ⟨100trials, 30epochs⟩	*image-input* ⟨100trials, 50epochs⟩
WTF_PAD	4ms (3,985,444)	5ms (431,721)	37ms (58,575,847)
FRONT_T1	4ms (3,985,444)	6ms (570,683)	22ms (21,066,383)
FRONT_T2	4ms (3,985,444)	5ms (429,369)	22ms (21,066,380)
CS_BuFLO	4ms (3,985,444)	14ms (2,599,287),	26ms (58,536,548)
Tamaraw	4ms (3,985,444)	4ms (2,597,991)	10ms (25,431,012)



(a) *image-input* (50 epochs). (b) *in-conv-dense* (30 epochs).

Figure 4: Validation acc. on defended traces (100 trials).

A APPENDIX

1. Defense configurations. We follow the methodology of Veicht et al. [44] and use default parameters for WF defenses, as suggested in their original papers. We used the WTF-PAD implementation from the WFES repository [5], and two versions of FRONT [14] with parameters: $N_c = N_s = 1700$ and $W_{min} = 1$, $W_{max} = 14$ for FRONT_T1, and $N_c = N_s = 2500$ for FRONT_T2. Due to its larger sampling window, FRONT_T2 induces more dummy packets in the trace when compared to FRONT_T1. For CS-BuFLO [3] and Tamaraw [4], we used: $d = 1$ and $2^{-4} * 1000 \leq \rho \leq 2^3 * 1000$, along with $\rho_{out} = 0.04$ and $\rho_{in} = 0.012$ with $L = 50$, respectively.

2. Tik-Tok convergence. Figure 2 depicts the accuracy obtained by the Tik-Tok model on the validation set as the number of training epochs increases (until 50 epochs). We can see a trend where the model's accuracy tends to stabilize at around 30 epochs for the FRONT variants. While accuracy for undefended and defended traces seems to converge sooner, the accuracy of the model is not significantly degraded when trained for a larger number of epochs.

3. Comparison of DNN layouts. Figure 3 compares the layout of the most accurate models generated by AutoKeras on undefended Tor traffic. The architecture of the *in-conv-dense* model is smaller than that of the Tik-Tok model, as it contains only two convolutional blocks (i.e., a combination of Conv1D, MaxPooling, and Dropout), whereas the Tik-Tok model contains four convolutional blocks. In tandem with our evaluation results, this suggests that relatively smaller models can achieve comparable accuracy to that of Tik-Tok.

In turn, more complex models produced by *image-input* layouts achieve a comparable accuracy with respect to Tik-Tok on undefended traces. Specifically, we see that the *image-input* model leverages a ResNet50 [20] block (i.e, a complex convolutional neural network that is 50 layers deep). The use of this larger model can outperform Tik-Tok on specific WF defenses (see Section 4.2).

4. Details on evaluation over defenses. Table 3 compares the inference time (per batch) and the total number of parameters, resp., of Tik-Tok's DNN and that of the most accurate models generated by AutoKeras (after 100 trials) on defended Tor traffic.

Model inference time. The inference time for *in-conv-dense* models is comparable to that of Tik-Tok, whereas *image-input* models can take from 2.5 to ~9× longer times for issuing predictions.

Model size. We see that *in-conv-dense* models are ~1.5 to ~9× smaller than the Tik-Tok model, whereas *image-input* models are ~5 to ~14.5× larger. The usage of the more complex ResNet blocks instead of convolutional blocks is the primary reason for this increase in size, but the resulting *image-input* models were able to outperform Tik-Tok against FRONT, a random padding defense.

5. AutoKeras' models convergence. Figure 4 depicts the evolution of the maximum validation accuracy obtained by both AutoKeras variants on defended traces, as the number of exploration trials increase. Interestingly, *image-input* models seem to converge faster than *in-conv-dense* models for the adaptive and random padding defenses considered in our evaluation. For instance, we can see that *image-input* obtains over 80% accuracy for WTF-PAD and FRONT_T1 in under 25 exploration trials, whereas *image-input* models reach a lower accuracy plateau only after 50 trials. This highlights the ability of the (more complex) *image-input* models to better learn from noisier (defended) Tor traffic samples.