Deep Neural Exposure: You Can Run, But Not Hide Your Neural Network Architecture!

Sayed Erfan Arefin
Texas Tech University
Lubbock, TX, United States
saarefin@ttu.edu

Abdul Serwadda
Texas Tech University
Lubbock, TX, United States
abdul.serwadda@ttu.edu

ABSTRACT

Deep Neural Networks (DNNs) are at the heart of many of today's most innovative technologies. With companies investing lots of resources to design, build and optimize these networks for their custom products, DNNs are now integral to many companies' tightly guarded Intellectual Property. As is the case for every high-value product, one can expect bad actors to increasingly design techniques aimed to uncover the architectural designs of proprietary DNNs. This paper investigates if the power draw patterns of a GPU on which a DNN runs could be leveraged to glean key details of its design architecture. Based on ten of the most well-known Convolutional Neural Network (CNN) architectures, we study this line of attack under varying assumptions about the kind of data available to the attacker. We show the attack to be highly effective, attaining an accuracy in the 80 percentage range for the best performing attack scenario.

CCS CONCEPTS

• **Security and privacy → Software and application security**;

KEYWORDS

power attack; deep neural networks; GPU; side channel

ACM Reference Format:
Sayed Erfan Arefin and Abdul Serwadda. 2021. Deep Neural Exposure: You Can Run, But Not Hide Your Neural Network Architecture!. In *Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security (IH&MMSec '21), June 22–25, 2021, Virtual Event, Belgium.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3437880.3460415

1 INTRODUCTION

A growing number of emerging learning-oriented applications are centered around deep neural networks (DNNs). From voice assistants [19] to video processing software [15] and embedded AI hardware [22] to mention but a few, DNNs are increasingly the core underlying technology. Companies invest huge amounts of money (in cloud computing costs, engineer time, acquisition of training data, etc.) to design and optimize these DNNs and thus have them as part of their tightly guarded Intellectual Property. On the other hand, malicious entities are interested in deciphering the DNN(s) for personal gain (e.g., to build their own version of the proprietary tools). In this paper, we study the question of whether power measurements made while a DNN runs on a workstation could be used to determine the architecture of the DNN.

In its simplest form, our **threat scenario/model** is as follows. While a DNN performs classification on a GPU owned by the adversary, the GPU's power draws over time are recorded. This data is then fed to machine learning algorithms which have been trained to classify the DNN architecture. The adversary is somehow unable to access the raw DNN code and thus cannot simply review it and determine the architecture. However, he/she can run the DNN and access the power measurements. In practice, this would loosely map to the scenario described below.

Assume a company that owns some proprietary DNN-centric software tool. Such a tool could for instance perform some highly sought-after (proprietary) video analytics function. Customers can buy copies of this tool just like other commercial software and install it on their systems to perform the video processing function. However, they cannot access the code directly (i.e., the tool is protected from decompilation). Some unscrupulous customer (i.e., adversary) who seeks to create a knock-off version of the tool buys it and repeatedly feeds it a bunch of bogus inputs while collecting power measurements as the tool runs on the GPU. We tackle the question of whether the adversary could leverage this power data to make meaningful inferences about the DNN architecture underlying the tool. Figure 1 shows a flow diagram of our hypothesized attack scenario.

A core assumption underlying this line of attack is that in the real world, practitioners typically draw from a small number of well-known, high performing architectures and use them in their entirety, or modify them slightly (e.g., through transfer learning) to fit the custom application at hand [21]. For example, the VGGNet architecture [18] is widely used in image localization tasks [4] and has several widely used derivatives (see 8 of them here [3]) which build upon the core VGGNet framework. Other architectures such as ResNet [13], AlexNet [16], etc., are widely used, both in their original forms or as derivatives of the original form [21]. Because each of these architectures has certain signature components and features (e.g., size of inputs, nature of connections, number of layers, distribution of arithmetic operations, etc.), it is plausible that the execution of the network poses some form of power consumption signature that emanates from these features/components.

We hypothesize that a hacker who is working to reverse-engineer some DNN-oriented application could leverage such power signatures, and use machine learning algorithms to try and infer the network in use, or at least which network is most similar to that

IH&MMSec '21, June 22–25, 2021, Virtual Event, Belgium

ACM ISBN 978-1-4503-8295-3/21/06…$15.00
https://doi.org/10.1145/3437880.3460415

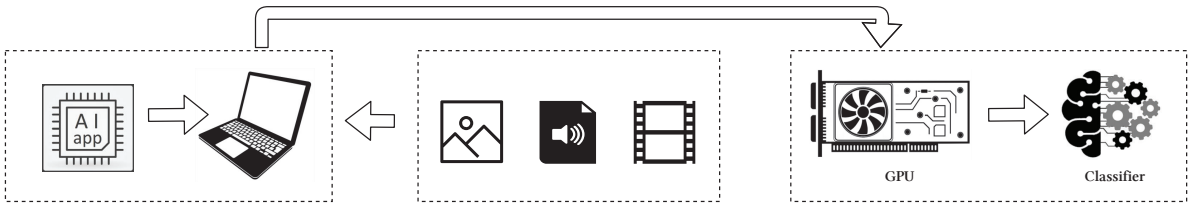


Figure 1: Attack Scenario

used by the target application. The paper studies this line of attack in the context of a desktop setting. Our contributions are summarized below.

(1) **Studying GPU power patterns of ten widely used neural network architectures:** We take ten of the most popular CNN architectures on various image processing tasks and study their power consumption signatures while they run the classification step on a GPU installed in a Windows desktop computer. Some of these architectures are drawn from the ImageNet competition [2] while others were developed elsewhere in academia or industry. We run these networks on two tasks: the classical cats vs dogs classification task, and a classification of random images. By feeding the emergent power measurement data into machine learning classifiers, we find that the underlying architecture can be classified with an accuracy of over 80% in the best case.

(2) **Studying the connection between GPU power draw patterns of the ten core neural network architectures and some of their most well-known derivatives:** We tackle the question of whether power measurement data drawn from the ten core architectures could be used to infer their derivatives. Specifically, we build a classifier that uses training data from 7 (of the 10 core) architectures that have one or more variants for which pre-trained models are publicly available. We then study how this classifier assigns 32 different well-known variants of these 7 architectures during the test phase into the 7 architectural classes. In practice, this contribution tackles an interesting scenario where the attacker has access to data from a small number key architectures, but is faced with the task of inferring some proprietary architecture (assumed derived from the key architectures) for which no data is yet available. We show this form of the attack to attain a classification accuracy of up to 41%, revealing the attack to pose a threat even when no training data from the target network is available.

2 RELATED RESEARCH

To our knowledge, the only other work to have studied DNNs/CNNs with regard to the line of attack covered in this paper is that by Xiang et al. (See Arxiv pre-print [20] and a smaller conference version of their work [21]). In that work, a Raspberry Pi was used to run 4 different CNN architectures (i.e., AlexNet, InceptionNet, MobileNet, and ResNet) and 2 of their variants while an external data acquisition card registered the associated power draw measurements. The idea behind using a Raspberry Pi was to simulate the scenario of an ARM Cortex-based embedded AI device that uses a DNN. On applying machine learning algorithms to the power measurements, they were able to infer the CNN architecture with an accuracy of up to 96.5 % in the best-performing configuration. While Xiang et al. also investigate the question of power measurements potentially giving away information about the underlying DNN/CNN architecture, their work has *at least 3 fundamental differences* from our work. These are as follows.

(1) **Scope of the problem**: Xiang et al. studied 4 core architectures and 2 variants derived from two of them (i.e., 6 architectures in total). Our work on the other hand studies 10 core architectures and 32 different variants of the 10 architectures under three different attack scenarios. Our wider scope provides insights into the power patterns of a much broader range of architectures and how these patterns might enable side-channel attacks on the architectures.

(2) **Experimental platform used**: Xiang et al. use a Raspberry Pi to run the DNNs. Our experiment on the other hand is run on a fully-fledged Windows desktop. Being a minimal computer with just a few peripherals and software services installed, the Pi provides an environment that is markedly different from that on a desktop computer. In particular, the Pi enables fine-grained power draw patterns to be monitored absent of the kinds of noise and perturbations that can be expected from the many peripherals on a fully-fledged desktop computer. Ours is the first work to show that some form of a DNN power fingerprint can be extracted despite multiple peripherals competing for power in a desktop setting.

(3) **Processing device**: In this work, we use a GPU to run the DNN/CNNs while power draw measurements on the GPU are registered. Xiang et al. on the other hand run their experiments on CPU. Because DNNs are largely run on GPUs for most practical applications, our work studies this line of attack in a setting that very closely mirrors typical end-user conditions seen today.

(4) **Target threat scenario**: The design of experiments in Xiag et al.'s work is focused on the scenario of an embedded AI hardware that contains the DNN to be attacked. The Pi provided them with the closest approximation to this kind of AI device. Our work on the other hand is done with a software installation in mind — we set our experiments and evaluation from the perspective of an end-user who installs a DNN-oriented software with the aim to launch the attack on the DNN.

A number of other attacks on neural networks have recently been showcased. In [11], a timing attack that monitors the neural network's execution time is shown to infer the depth of the network. Leveraging reinforcement learning, the attack ultimately

enables the construction of an equivalent architecture within reasonable margins of error. In [14], the utilization of the GPU, PCIe Bus, and Device Memory Bus are used to drive a reconstruction of the neural network architecture. Other lines of attack include those that seek to invert the model (e.g., [12]), those which seek to generate adversarial examples that evade the neural network classifier (e.g., [17]), and those which use electromagnetic emanations to reverse-engineer the network (e.g., [7]). These papers emphasize the importance of research that studies attacks on neural networks. However, none of them leverages GPU power draw patterns for network inference as done in this paper.

3 EXPERIMENTAL DESIGN

3.1 Overview of neural network architectures under investigation

As described in Section 1, our research is focused on ten Convolutional Neural Network (CNN) architectures that are very widely used in deep learning applications. Most of them originate from the ImageNet Large Scale Visual Recognition Challenge (ILSVR) [2], a competition in which practitioners build neural network architectures to compete on various image classification tasks. Table 1 shows these architectures as well as a summary of some of the features that distinguish between them. Deeper details on these networks can be found here [1]. In some of the plots and figures in the subsequent sections, we refer to the 10 architectures using the numerical identifiers, 1 to 10 for simplicity of our presentation. The mappings between the identifiers and architectures are shown in Table 1.

3.2 Data Collection

Recall from Section 1 that the attack scenario studied in this work assumes an adversarial end-user who runs a DNN on an end-host while collecting power measurement data. To simulate this scenario, our collection of power measurement data focused on the testing phase (as opposed to the training phase) since this is the step that would typically run on an end-user's device. Our data collection experiments hence entailed running the neural network classification step for each of the architectures studied in the paper. We used the pre-trained models published with the PyTorch API documentation [3]. Two different datasets were used for attack: a dataset of 50 images of cats and 50 images of dogs (alternately referred to as Dataset-1) and a dataset of 100 random images (alternately referred to as Dataset-2). These datasets were obtained from the ImageNet site [10]. This diversity in datasets helped ensure that we observe patterns that are not an artifact of a particular dataset.

The datasets are first fed to each neural network while the GPU-Z software collects data during the test time. The software produces a log file with the power data. Since, the log file cannot be deleted when the software is collecting data, between every sample of data there is a pause of 2 seconds when the log file is copied to a different directory. The GPU-Z software is restarted after testing each network. Between the restart, the log file cleared. This emergent dataset is then passed to Machine learning classifiers to classify the DNN architecture.

Experiments were run on a Windows desktop computer with the following specs: Intel Core i7 (9700K) processor, Nvidia RTX 2060 GPU, 32 GB of Memory, and 512 GB PCIe based Solid State Drive. During the data collection process, it was ensured that no graphics-intensive services were running in order to minimize the noise in GPU power patterns. In practice, the adversarial end-user would easily ensure the same condition since they would have full control of the computer being used. We recorded power measurements from the following power sensors available on our GPU card.

- **Power Draw Board:** The total power drawn by the GPU board in Watts.
- **Power GPU chip:** The total power consumed by the GPU chip in Watts.
- **Power MVDDC:** The power drawn by the Graphics card memory in Watts.
- **Power PCIe Slot:** The power drawn by the GPU from the PCIe 16x slot in Watts.
- **Power 8-pin:** The power drawn by an 8-pin power connector external to the motherboard, connected directly from the power supply. A GPU can have more than one 8-pin connector. Each 8-pin connector can draw a maximum of 150 Watts.
- **Power Consumption TDP:** TDP stands for Thermal Design Power. It is an indicator of the maximum power draw by a computer chip.

We used the "GPU-Z" tool by Techpowerup [6], for power measurement. The tool collected sensor data continuously and wrote them in the log file while the DNN ran. The sensor read interval was set to 100 samples per second.

3.3 Features and classifiers used

We received power sensor readings for every images provided as input to the CNNs. For each power sensor reading, we extracted a wide range of time and frequency domain features using the TSFresh library [5]. Table 2 shows the list of features extracted for each sensor. We leave out a detailed description of these features due to space limitations. Details can however be found on the TSFresh documentation page [5]. For every image fed to each of the CNNs, a total of 108 feature vectors were received. We split the power features dataset with a 70-30 ratio for training and testing. Given the training and testing datasets we conducted classification using the Multi-class Logistic Regression [8] and Random Forest [9] classifiers.

4 ATTACK PERFORMANCE EVALUATION

4.1 Preliminary data exploration

Before running the classification, we first undertook some preliminary exploration of the power measurement data in order to get some sense of whether it might depict obvious patterns that suggest the attack could work. Figure 2 shows plots that express some of our results from this preliminary analysis.

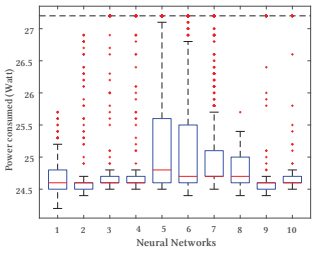
Figure 2a shows a box plot of the raw power data from the power "Power 8-pin" sensor for all ten architectures. A box plot captures basic statistics such as the median, upper quartile, maxima, lower quartile, etc., as well as insights into outlier behavior. One obvious pattern depicted by the plot is the difference in spread/variability exhibited by the different architectures as seen from the maximum

Identifier	CNN Architecture	Architecture Layers			Network Properties				Accuracy	
		Conv	FC	PL	Input Size	Total Parameters	Dropout	Salient Feature	Top-5	Top-1
1	AlexNet	5	3	1	256x256x3	62,378,344	Used while training	Deeper	80.3%	57.2%
2	DenseNet 121	120	1	5	224x224x3	7.0 Million	20% after every Conv layer (except for the 1st)	Each layer connected to all	92.29%	74.98%
3	DPN 131	44	1	2	224x224x3	79.5 Million	None	New & reuse features	80.07%	94.88%
4	Inception v4	38	0	5	299x299x3	43 Million	80% before SoftMax	Parallel kernels	95%	80%
5	NASNet Large	33	0	24	299x299x3	3.1 - 27.6 Million	50% on SoftMax.	Architectural search	96.2%	82.7%
6	PolyNet	85	1	3	331x331x3	76.1 Million	0 to 25% Stochastic paths	Optimized deeper	95.75%	81.29%
7	ResNet 18	17	1	2	224x224x3	11,511,784	None	Shortcut connections	90.58%	71.78%
8	SqueezeNet	14	0	4	256x256x3	1,248,424	50% after 9th fire module	Compressed	80.3%	57.5%
9	VGG11	8	3	5	224x224x3	138,423,208	First 2 FC with 50%	Fixed-size kernels	90.4%	71.8%
10	Xception	36	1	5	299x299x3	22,855,952	50% before the Logistic regression layer	Extreme Inception	94.5%	79%

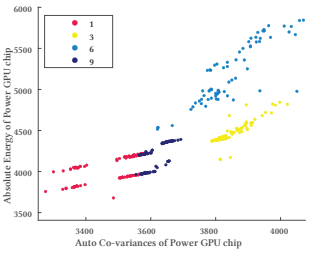
Table 1: High-level comparison of CNN architectures and identifiers used in our research

Features	Sensors
Benford ABS Sum of Changes, Standard Deviation, Binned Entropy, Lempel Ziv Complexity, Variance, Abs Energy, Count Above Mean Minimum, Variation Co-efficient, Last Location of Minimum, C3, Last Location of Maximum, CidCe	All
Benford Correlation	All except MVDDC & PCIe Slot

Table 2: List of features extracted from the power sensors.



(a) A box plot of raw measurements from the "Power 8-pin" sensor for all the architectures 1 to 10.



(b) A plot of 2 features for 4 of the architectures.

Figure 2: Preliminary observations on the power measurement data of the ten DNN architectures. The identifiers on the x-axis of the box plot are listed in Table 1.

and minimum values across the plot (for example, architecture 6 has much more spread than 2 and 3). Differences in medians are subtle but also clear to the eye. While this is just raw data studied in terms of simple statistical metrics, the box plot reveals the first signs that the 10 architectures could indeed depict different power patterns.

In Figure 2b, we take this a step further and visualize two features extracted from the raw data. The figure only has 4 architectures which gave us a clear visual pattern. Observe that these four architectures are very clearly separated in the 2D space. Other pairs of features (results not shown here due to space limitations) revealed equally promising patterns. With this kind of promise depicted in this very low dimensional space, we conjectured that a more elaborate catalog of features coupled with machine learning algorithms should be able to discriminate between the neural network architectures based on power measurements. These results encouraged us to proceed to implement a fully-fledged classification of the data.

	Logistic Regression	Random Forest
Dataset 1	62.96%	73.57%
Dataset 2	69.43%	82.81%

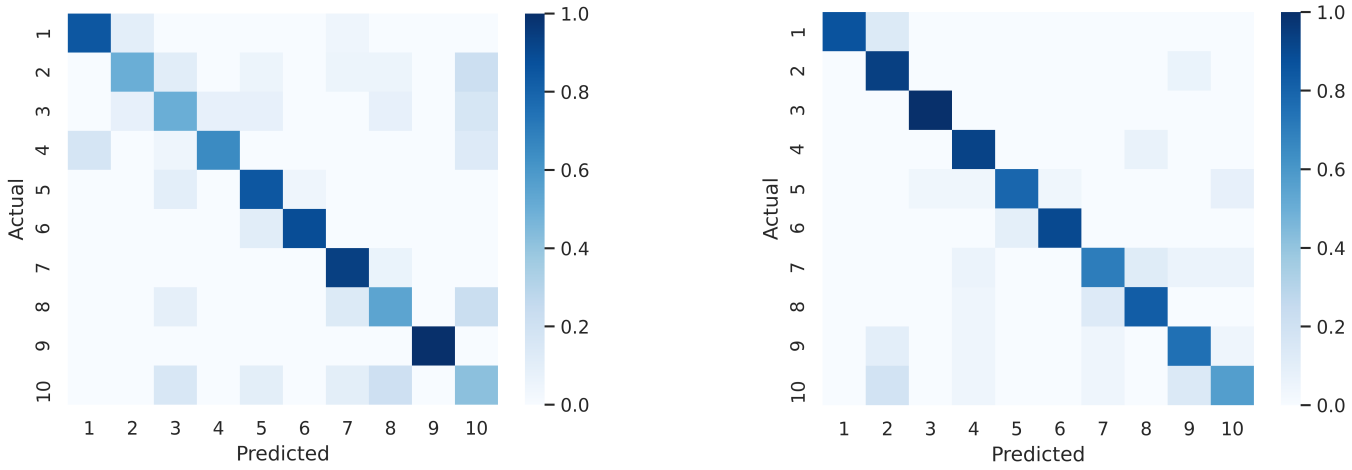
Table 3: Average Accuracy

4.2 Classification results

We analyze the attack in three different configurations, each of which having different assumptions and aims for the attacker. Details and results from each of the configurations follow

4.2.1 Attack Configuration #1: This configuration assumes the attacker has access to training data from the ten core models and the victim uses one of the ten models. Hence the attack is a 10-class classification problem in which the attacker wants to know which of the ten classes maps to the victim. The practical justification of this scenario lies in the fact that many practitioners use the core DNN architectures exactly as designed and calibrated by academia or industry entities such as Google. Attack Configuration #1 captures the threat posed to such practitioners/victims if the adversary were to use the same core models to build the machine learning-based attack system.

Table 3 and Figure 3 summarize our results from this configuration. Table 3 shows the classification accuracy for Datasets 1 and 2 for both classifiers used. The highest accuracy (Random Forest) is just under 83%, pointing to the lethality of the attack as compared to a random attack which would attain only about 10% accuracy. For each classifier, Dataset-2 (i.e., the random images dataset) reveals a better performance than Dataset-1 (cats and dogs images) by over 6 percentage points. We conjecture that the diversity of image properties in the random images dataset triggered a wide variety of neural network behaviors (e.g., dynamics of additions/multiplications, activations, etc.), which in turn produced more diverse power measurements that exhibited stronger generalization power when trained on the classifier. The cats and dogs dataset likely produced a power pattern that was tightly tied to



(a) Dataset 1 (50 Cats and 50 Dogs images from imageNet)

(b) Dataset 2 (100 randomly sampled images from ImageNet)

Figure 3: Confusion matrix of Random Forest Classification performed on Dataset-1 and Dataset-2. The actual labels of the classes, which are identified with numbers in this figure, can be found in Table 1.

these two kinds of images, limiting the generalization power of a classifier built from this dataset. In practice, the attacker will have complete control over which inputs to provide to the system. Our findings here suggest that inputs exhibiting high variability might be the attacker's best strategy.

Figure 3 provides a more fine-grained look at the results shown in Table 3. In particular, the figure shows how the classifier treated each of the ten classes. We focus on the Random Forest classifier since it performed best. The dark diagonals re-emphasize the high accuracy of this classifier as already seen in Table 3. In Figure 3a, architectures 2 and 3 depict two of the worst classification performances (as seen from the light blue shades). Notably, however, these two architectures depict some of the best accuracies in Figure 3b. Since the difference between Figures 3a and 3b is just the dataset used, Figure 3 emphasizes the impact of the nature of inputs on the performance of this line of attack.

4.2.2 Attack Configuration #2: In the second attack configuration, we assume that practitioners (also the same as victims in our case) have adapted the core well-known architectures to their custom application scenario by creating new variants. Relative to the core (or base) architecture, such variants typically have changes in variables such as sizes of inputs, kinds of activation functions, numbers of different kinds of layers, etc. Such a variant (or variants) would be proprietary and very tightly guarded. In our research, we hence assume that, for training purposes, the attacker only has access to the core architectures but does not know, or have access to, the variant used by the victim (application).

The attacker's intent in such a case is to simply determine the family to which the variant belongs. The attacker will seek to address a question of the form: *Is the victim using a variant of the core architecture A?* If the attacker can answer that question accurately, they would be a step closer to mapping out the finer details of the variant. To implement this variant of the attack, we use data generated from the core architectures for training. For testing, we then use the variants. A classification is deemed accurate if a variant is classified as the core architecture from which it is derived. Note that of the ten architectures studied in this work, only 7 have variants published at this time (see PyTorch page [3]). Hence, we train on the 7 core architectures and then test on the variants of these architectures.

DNN Architecture	Variants
SqueezeNet	SqueezeNet 1, SqueezeNet 1.1
DPN	DPN68, DPN68B, DPN92, DPN98, DPN131
DenseNet	DenseNet121, DenseNet161, DenseNet169, DenseNet201
InceptionNet	BN InceptionNet, InceptionNet v3, InceptionNet v4
VGG	VGG11, VGG13, VGG16, VGG19, VGG11 BN, VGG13 BN, VGG16 BN, VGG19 BN
ResNet	ResNet18, ResNet34, ResNet50, ResNet101, ResNet152, CaffeResNet101, FB ResNet152
NASNet	NASNet Mobile, NASNet-A Large, P-NASNet 5 Large

Table 4: 7 Core DNN architectures and their well-known variants.

Table 4 shows the 7 core architectures and their derivatives as used in this form of the attack. Details of these architectures and variants can be found here [3]. When we run this version of the attack, we got a classification accuracy of 42%[1]. Considering that this was a 7-class problem, a random attacker would have got about 14% accuracy. Hence the attack, even in this challenging setting where the attacker has no exact data to train with, still poses a noteworthy threat.

[1]Due to space limitations, from here on-wards, we only report results from Dataset-2 since it generally performed best.

CNN	True positive	CNN	True positive
BN Inception	23.80%	NASNet Mobile	96.20%
Caffe ResNet	66.70%	PNASNet 5-large	100%
DenseNet 121	100%	ResNet 101	60%
DenseNet 161	57.70%	ResNet 152	65%
DenseNet 169	32%	ResNet 18	68.20%
DenseNet 201	33.30%	ResNet 34	66.70%
DPN 131	100%	ResNet 50	62.50%
DPN 68	60%	SqueezeNet 1.0	100%
DPN 68b	79.30%	SqueezeNet 1.1	85.70%
DPN 92	85%	VGG 11	81%
DPN 98	44.40%	VGG 11 BN	52%
FBResNet 152	33.30%	VGG 13	41.90%
Inception v3	25%	VGG 13 BN	37.50%
Inception v4	75%	VGG 16	59.30%
NASNet large	100%	VGG 16 BN	45.80%
VGG 19 BN	68%	VGG 19	48.40%

Table 5: True positive rates of 32 variants of our core DNN architectures.

4.2.3 Attack Configuration #3: In the third and final configuration, we again assume that the victim is using a variant of a core architecture. The only difference this time is that the attacker also has access to all available variants and can hence use them for training. This configuration is similar to Configuration # 1 in that data classes used in the training set are also available in the testing set. The two major differences between the two configurations however are that: (1) in practice this would have a much larger number of classes (here we have 32 in total as opposed to just 10 in Configuration #1), and that, (2) many of the (32) classes are so similar to each other since they could be minor tweaks of a common architecture. Thus, while Configuration #3 has a fundamental similarity to Configuration # 1, the former should be more challenging due to the two points raised above.

We obtained a classification accuracy of 64.17% on attack configuration #3. Table 5 shows a detailed breakdown of how each of our 32 classes contributed to this overall accuracy. The figure shows the true positive rate of each class (i.e., the equivalent of the leading diagonal of a confusion matrix). Observe that each of the classes performed way above random guessing (which is about $\frac{1}{32}$ or 3.13%). Some even hit 100%, pointing to the distinctiveness of their GPU power draw signatures. Overall, the table reveals that, even with the very large number of classes, the attack still performs very well.

5 CONCLUSION

In this paper, we have designed and evaluated an attack that infers the CNN/DNN architecture based on GPU power draw patterns. We have evaluated the attack under three different configurations and shown it to be highly effective in each of them. In practice, this line of attack could be utilized by a hacker who intends to reverse-engineer a proprietary application that depends on some DNN. One potential limitation of our work is that proprietary apps in real life might have other operations running concurrently with the DNN, resulting in the blurring of the DNN signature and the impact of the attack. This potential limitation notwithstanding however, our key finding of DNNs having consistent power signatures provides valuable insights to the DNN research/industry community who should find it to be a useful first step towards a deeper exploration of this line of attack under scenarios reflecting various relevant use-cases.

REFERENCES

[1] [n.d.]. Large Scale Visual Recognition Challenge 2017 (ILSVRC2017).
[2] [n.d.]. Large Scale Visual Recognition Challenge (ILSVRC). http://www.image-net.org/challenges/LSVRC/. Accessed: 2021-02-24.
[3] [n.d.]. Pretrained models for Pytorch. https://github.com/Cadene/pretrained-models.pytorch. Accessed: 2021-02-24.
[4] [n.d.]. Results of ILSVRC2014.
[5] [n.d.]. Tsfresh. https://tsfresh.readthedocs.io/en/latest/. Accessed: 2021-02-25.
[6] 2021. TechPowerUp GPU-Z. https://www.techpowerup.com/gpuz/
[7] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. 2019. CSI NN: Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 515–532. https://www.usenix.org/conference/usenixsecurity19/presentation/batina
[8] Dankmar Böhning. 1992. Multinomial logistic regression algorithm. *Annals of the Institute of Statistical Mathematics* 44, 1 (01 Mar 1992), 197–200. https://doi.org/10.1007/BF00048682
[9] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (01 Oct 2001), 5–32. https://doi.org/10.1023/A:1010933404324
[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.
[11] Vasisht Duddu, D. Samanta, D. V. Rao, and V. Balas. 2018. Stealing Neural Networks via Timing Side Channels. *ArXiv* abs/1812.11720 (2018).
[12] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 1322–1333. https://doi.org/10.1145/2810103.2813677
[13] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), 770–778.
[14] X. Hu, Ling Liang, L. Deng, Shuangchen Li, Xinfeng Xie, Y. Ji, Yufei Ding, Chang Liu, T. Sherwood, and Yuan Xie. 2020. Neural Network Model Extraction Attacks in Edge Devices by Hearing Architectural Hints. *ArXiv* abs/1903.03916 (2020).
[15] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. 2014. Large-Scale Video Classification with Convolutional Neural Networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 1725–1732. https://doi.org/10.1109/CVPR.2014.223
[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc., 1097–1105. https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf
[17] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2017. Practical Black-Box Attacks against Machine Learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* (Abu Dhabi, United Arab Emirates) *(ASIA CCS '17)*. Association for Computing Machinery, New York, NY, USA, 506–519. https://doi.org/10.1145/3052973.3053009
[18] K. Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2015).
[19] G. K. Venayagamoorthy, V. Moonasar, and K. Sandrasegaran. 1998. Voice recognition using neural networks. In *Proceedings of the 1998 South African Symposium on Communications and Signal Processing-COMSIG '98 (Cat. No. 98EX214)*. 29–32. https://doi.org/10.1109/COMSIG.1998.736916
[20] Yun Xiang, Zhuangzhi Chen, Zuohui Chen, Zebin Fang, Haiyang Hao, Jinyin Chen, Yi Liu, Zhefu Wu, Qi Xuan, and Xiaoniu Yang. 2019. Open DNN Box by Power Side-Channel Attack. arXiv:1907.10406 [cs.CR]
[21] Y. Xiang, Z. Chen, Z. Chen, Z. Fang, H. Hao, J. Chen, Y. Liu, Z. Wu, Q. Xuan, and X. Yang. 2020. Open DNN Box by Power Side-Channel Attack. *IEEE Transactions on Circuits and Systems II: Express Briefs* 67, 11 (2020), 2717–2721. https://doi.org/10.1109/TCSII.2020.2973007
[22] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W. Hwu, and D. Chen. 2018. DNNBuilder: an Automated Tool for Building High-Performance DNN Hardware Accelerators for FPGAs. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8. https://doi.org/10.1145/3240765.3240801