Inferring DNN layer-types through a Hardware Performance Counters based Side Channel Attack

Dandpati Kumar Bhargav Achary
IIT Hyderabad
India
cs18resch01002@iith.ac.in

R Sai Chandra Teja
CKM VIGIL Pvt Ltd
India
saichandrateja@gmail.com

Sparsh Mittal
IIT Roorkee
India
sparsh.mittal@ece.iitr.ac.in

Biswabandan Panda
IIT Bombay
India
biswa@cse.iitb.ac.in

C Krishna Mohan
IIT Hyderabad
India
ckm@cse.iith.ac.in

ABSTRACT

Recent trends of the use of deep neural networks (DNNs) in mission-critical applications have increased the threats of microarchitectural attacks on DNN models. Recently, researchers have proposed techniques for inferring the DNN model based on microarchitecture-level clues. However, existing techniques require prior knowledge of victim models, lack generality, or provide incomplete information of the victim model architecture. This paper proposes an attack that leaks the layer-type of DNNs using hardware performance monitoring counters (PMCs).

Our attack works by profiling low-level hardware events and then analyzes this data using machine learning algorithms. We also apply techniques for removing the class imbalance in the PMC traces and for removing the noise. We present microarchitectural insights (hardware PMCs such as cache accesses/misses, branch instructions, and total instructions) that correlate with the characteristics of DNN layers. The extracted models are also helpful for crafting adversarial inputs. Our attack does not require any prior knowledge of the DNN architecture and still infers the layer-types of the DNN with high accuracy (above 90%). We have released the traces for public use at https://github.com/bhargavarch/DNN_RevEngg_PMC_Dataset.

AIMLSystems '21, October 21–23, 2021, Bangalore, India
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8594-7/21/10…$15.00
https://doi.org/10.1145/3486001.3486224

KEYWORDS

side channel, deep neural networks, hardware performance counters, privacy, reverse engineering

ACM Reference Format:
Dandpati Kumar Bhargav Achary, R Sai Chandra Teja, Sparsh Mittal, Biswabandan Panda, and C Krishna Mohan. 2021. Inferring DNN layer-types through a Hardware Performance Counters based Side Channel Attack. In *The First International Conference on AI-ML-Systems (AIMLSystems '21), October 21–23, 2021, Bangalore, India.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3486001.3486224

1 INTRODUCTION

In recent years, deep neural networks (DNNs) have attracted significant interest from the research community and industry for a wide range of application domains, e.g., autonomous vehicle driving, speech recognition [28], and computer vision [22]. However, this has also led to increased risks of attacks on DNN models. The attacks on DNNs can be classified into various types [1, 12, 15, 21, 25, 29], such as side-channel attack, fault-injection attack, trojan-insertion attack, and adversarial input attack, etc.

A side-channel attack (SCA) [30] uses side channel information leakage, such as time consumption [13], power consumption [18], and electromagnetic radiation [7], etc. Recent side-channel attacks on DNNs leak their architecture or hyperparameters. Some of these works assume that the adversary has physical access to the device, so high-resolution side-channels about power consumption and accessed memory addresses can be exploited [2, 9, 27]. A few other works that exploit cache side-channels in CPU assume that the attacker accesses the DNN model remotely [6, 8, 29].

The scope: The fault-injection attacks seek to bring down the accuracy of a DNN to that of a random guess. An SCA

Sparsh is the corresponding author.

seeks to identify the architecture of a DNN. An SCA can further help improve the efficacy of other attacks, such as fault-injection attacks, by allowing the adversary to significantly narrow down the attack space.

Threat model: We follow a black-box threat model in an MLaaS (Machine Learning as a Service) setting, similar to a previous work [29]. In a black-box attack, the DNN model is accessible to attackers only via an official query interface. The victim runs DNN algorithms on a CPU. Here we have an additional assumption that the adversary is a system administrator who has access to PAPI framework [26] and can access the PID of the victim process. Since the model files may be encrypted, the attacker may only be able to query them. Also, despite the emergence of accelerators, the CPU remains an important platform for executing DNN algorithms [16].

Our goal: In practice, AI developers usually design their models based on the existing, pre-trained, and publicly available architectures. Then, the models are fine-tuned based on their own training sets. Therefore, by carefully analyzing the performance monitoring counter (PMC) values, it is possible to reveal the actual model architecture. Thus, we use an SCA to infer the CNN architecture. The key questions we aim to answer in this work are Q1) Feasibility of using PMCs for launching a side-channel attack Q2) Do the microarchitectural characteristics of DNN leave behind architectural traces or signatures for side channels?. While we explore the PMC related insights and contextual information, we attempt to demystify the structural secret of DNN and recover the DNN architecture by mounting an SCA.

Challenges: Extracting the DNN layer using performance monitoring counters (PMCs) [5] presents several challenges: C1) the transition between DNN layer operations happens very quickly. Since PMCs provide data at a low sampling rate, ascertaining the layer-boundaries is difficult; C2) the execution time of different layers varies significantly, resulting in an uneven number of samples. This leads to a class imbalance in the dataset, and finally, C3) the PMCs are noisy by nature. These factors limit the generality and robustness of previous works.

Our approach: This work proposes a novel SCA that utilizes the CPU's hardware PMCs to predict the model structure based on a pre-trained classifier. Our attack works by profiling microarchitectural events. We analyze these event-counters with appropriate machine learning techniques. To address challenge C1, we insert a sleep layer in the DNN. We do this in the pre-training (i.e., labeling) phase only, and remove all sleep layers before training. Thus, there are no sleep layers during inference. Also, to improve the resolution, we disable the hardware flags. To address challenge C2, we use techniques for handling the class-imbalance problem. Further, to remove noise in PMC readings, we apply Savitzky–Golay [24] filter. This addresses challenge C3.

A PMC trace keeps the counts of microarchitectural events through PMCs. We collect the PMC trace of a DNN over an inference of a query. It is possible to collect a PMC trace on several microarchitectural event counters (typically more than 30 in number). However, due to the limited (i.e., typically four) number of hardware registers, we choose four relevant events through which the DNN characteristics can be analyzed. The selection of these relevant events is vital in order to capture the behavior of individual layers so that the classifier can learn. We manually select these events after observing the PMC traces for all possible PMC events taken one after the other in batches of four. Our future work will focus on using machine-learning tools to more precisely select the hardware events to track. We visualized all the hardware event traces by using data visualization tools to arrive at the selection. We did not use PCA or ML-based approach, our selection was manual. However, our goal was to prove the hypothesis that PMCs do leak significant information for a SCA to work. it might be possible to arrive at similar or even better results with an even more precise selection of hardware events.

We utilize these PMC traces to train a classifier. Then, this classifier is used to launch an SCA on an unknown DNN for identifying and classifying its layers. The evaluation is based on the percentage of the execution time of respective constituent layers. To the best of our knowledge, our work is the first to demonstrate the use of PMCs to reverse-engineer the layer-type of an unknown DNN in a black-box setup.

Overall, our contributions are as follows:

(i) We create an automatically labeled dataset on the traces of three DNN models from their PMC traces (Section 2.4). We have evaluated multiple frameworks including Keras, PyTorch and TensorFlow. In all the frameworks, the traces remain similar.

(ii) We correlate PMC values to the characteristics of DNN layers. We train a machine-learning classifier to learn this correlation and use this classifier to infer the DNN layer being executed on a CPU. We overcome several challenges in the use of a machine-learning classifier on PMC traces, such as noise-filtering and class-imbalance (Section 2.4).

(iii) We train the classifier on a group of DNNs and test it on these and other unknown DNNs. We define the prediction accuracy of our machine-learning (ML) classifiers in terms of how many of the data points in the trace are predicted to belong to the correct layer type. The experimental results show that our attack can decode the layers of the DNN with high accuracy (Section 3).

2 THE PROPOSED ATTACK

2.1 Threat model

The existing adversarial attacks include white-box attack [17, 20] and black-box attack [25][4] [19], respectively. A white-box attack requires the complete knowledge of the target model, such as parameter values, network structure, and training dataset. Unlike a white-box attack, a black-box attack assumes that the target model is unknown. Naturally, a white-box attack is more effective and easier to implement [14]. However, in real-world applications, most AI devices are considered black-boxes. In general, a black-box attack is less effective but is easier to launch. This paper proposes an SCA in a black-box setup.

We assume a cloud-based setup, where the adversary, who is an administrator posing as a user, can query a DNN model offered as a service. The adversary issues just one query with any arbitrary input. During this execution, he records the available PMCs and creates a PMC trace of the DNN. The PMCs track the following events: the number of branch instructions, the number of L1 cache misses, the total number of CPU instructions being executed, and the total number of L3 cache references. The adversary does not know any internal details of the DNN, such as its layer-count, hyper-parameters, inputs, etc.

2.2 Background on PMCs

Modern microprocessors contain dedicated registers, named hardware PMCs. These registers count hardware events such as cache misses, CPU cycles, branch instructions. The primary utility of these counters is to help perform microarchitecture-level profiling of workloads. Hardware PMCs have higher resolution than software-based PMCs.

Modern Intel processors provide four dedicated registers for reading PMC values. Intel processors provide an instruction, namely RDPMC in its instruction set, through which one can read the hardware PMCs at a very high resolution. AMD and ARM processors also provide similar functionality in hardware and instruction sets. For instance, Intel processors allow counting multiple (typically more than 30) events using the counters. However, they are limited by the number of available hardware registers.

2.3 Correlating PMC values to DNN layer characteristics

The microarchitectural characteristics of a DNN are representative of the constituent layers and the operations performed by them [11]. The core computation of a convolution operation is matrix multiplication. In the convolution operation, the same kernel revolves over the entire input feature map. Due to this, convolution operation has significant data reuse, which leads to relatively low cache misses, provided the working set fits in the cache memory. By contrast, in a fully connected layer, the size of the working set is large. Also, there is no data reuse in the absence of batching; hence, FC layers show a very high number of cache references. Table 1 summarizes our observations on the execution patterns of typical DNN layers.

Table 1: Relative count of microarchitectural events in typical DNNs. Refer Figure 4 for an example from VGG16.

Layer	#Instructions	#Branches	#L3 Cache refs	#L1 Cache misses
Conv	High	Low	Medium	High
Pool	Medium	High	Medium	Medium
Act	Low	Medium	Low	Low
FC	Low	Low	High	Low

We extract the fine-grained structural secret of a DNN, including its layer composition, through PMC profiling. A trained classifier model utilizes PMC traces to estimate each layer of an unknown DNN architecture with high accuracy. By virtue of drastically reducing the search space of target models, our attack can make it easy to further launch adversarial attacks. There are two main phases in the attack framework: a training phase and an inference phase as shown in Figure 1 and 2. Note that these phases pertain to the ML classifier and not the victim DNN.

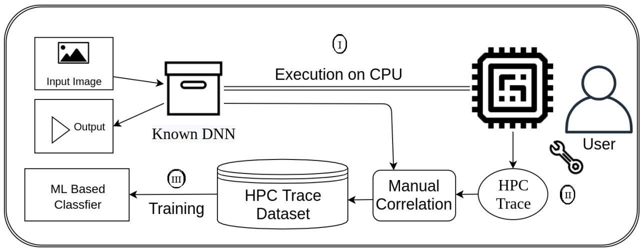

Figure 1: Training phase of the ML classifier.

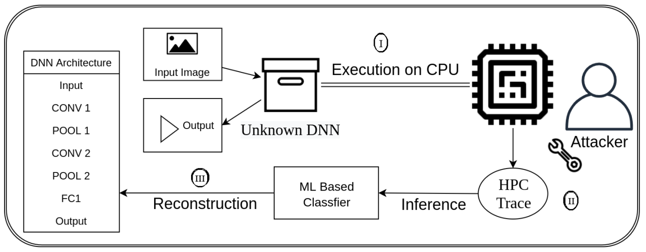

Figure 2: Inference phase of ML classifier: leaking DNN layer-profile.

2.4 Training phase of the classifier

Figure 1 shows the training phase for the proposed attack. The training phase involves three steps: (i) collection of PMC trace of known DNNs, (ii) automated labeling of the collected PMC trace data from the model architecture summary, and (iii) training a classifier on the labeled dataset. The steps performed by us and the challenges addressed are as follows.

Automated labeling: While the DNN runs, we seek to record both the DNN layer-type and the PMC values. It is well-known that ascertaining the layer-boundaries of DNNs is challenging [15]. To overcome this issue, either a manual approach or application-level hints are required. Since a manual approach is cumbersome, we prefer an automated approach for labeling the PMC traces. Hence, we insert a `sleep()` function between two subsequent layers, which acts as a separation marker. This automated approach allows us to scale the size of created dataset seamlessly. Also, it improves the accuracy and robustness of the SCA. Note that `sleep()` function is inserted only during the training phase and not during inference. Once the training is done, we remove the `sleep()` functions.

Creation of PMC trace dataset: We run the DNNs using the TensorFlow framework, but we have also verified that the traces are similar in other DNN frameworks. For various DNN architectures, there is no publicly available dataset of hardware event traces. Hence, we collect traces of known and popular DNNs to create a repository. We collect the PMC trace using the "PAPI library framework" [26] available on the Linux platform. PAPI has native implementations for returning the PMC values at a minimum interval of $1\mu s$. All our traces are collected at $1\mu s$ interval. However, for ease of illustration, the figures in this paper use a 1ms interval. We enhance the resolution of our traces by setting the CPU to the minimum supported clock of 1.2GHz. Furthermore, we also disable hardware prefetchers to enhance the details of cache misses and related events.

Handling noise in raw-data and Data-imbalance problem: Different layers have different execution times; however, the PMC samples are taken at a fixed interval. Hence, from the perspective of an ML classifier, there is a class imbalance in the traces of DNNs. We use SMOTE [3] based oversampling to balance out the class imbalance issue and also perform normalization to scale the data. We use Savitzky–Golay filter [24] to smooth out the intermittent spikes. We have verified that these changes improve the overall performance of the classifier. Figure 3 shows the effect of Savitzky–Golay filter on the trace.

From the trace shown on the right side of Figure 3, our technique can identify different layers of the CNN. Figure 4 shows the right side of Figure 3 after tagging the layer names in the convolutional block. Here, each convolutional block in VGG16 contains two convolution layers followed by a pooling layer, where convolutional layers have higher cache misses than a pooling layer. Similarly, when we see the characteristics of fully connected layers, we find that the cache misses are low and references are high, and there is a reduction in the number of CPU instructions executed.

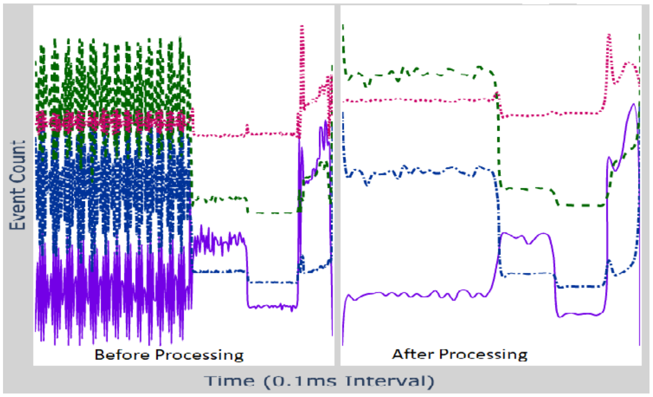

Figure 3: Left part: Raw PMC trace for VGG16. Right part: the trace after preprocessing using the Savitzky–Golay filter.

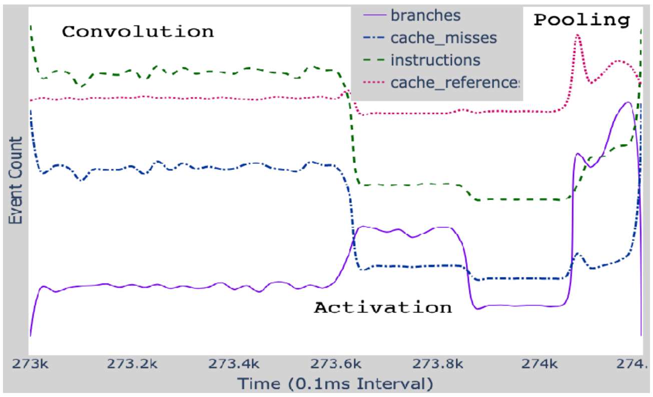

Figure 4: A PMC trace showing the typical execution phases in VGG16.

Training the classifiers: The classifier learns the various classes labeled in our dataset. Each class represents one type of layer from a known DNN. For example, in the VGG16 network, the classifier can differentiate between the convolutional layer, pooling layer, activations, and fully connected layers.

2.5 Inference phase of the classifier

Once a classifier is trained, we proceed to the inference phase, as shown in Figure 2. It involves three steps, (i) collection of PMC trace of unknown DNN, (ii) feeding the PMC trace to the pre-trained classifier, (iii) The classifier predicts the labels of the layers belonging to the PMC trace. Based on the classifier predictions, the unknown DNN's architecture is reconstructed. We evaluate the performance of the classifiers with accuracy scores.

3 RESULTS AND DISCUSSION

We perform all our experiments on a Linux OS (kernel 5.0) environment with an x86 architecture. We use Core™ i9 - 7900X processor with a clock frequency of 3.3GHz and 64GB of RAM. The kernel supports the PAPI [26] library framework, which is the main utility we use for collecting the data from PMCs. We train our ML classifiers with three DNNs: DenseNet [10], VGG19 and MobileNetV2 [23]. These DNNs are reasonably complex, and this helps in making the classifiers robust and versatile. Further, Alexnet can be considered the forerunner of deep learning, VGGNet uses a smaller convolution kernel, and ResNet uses skip-connection. Thus, we have chosen a diverse set of DNNs for training as well as testing.

3.1 Choice of ML classifier

We evaluate three classification algorithms viz., decision tree, random forest, and KNN (K-Nearest Neighbors). We train our classifier with the dataset, which has labels for more than 10 types of layers, such as convolution, pooling, activation, fully connected, flatten, batch normalization, dropout, and even additional operation layers like add and concatenate. Though we perform training on so many kinds of layers, for the sake of clarity, we show results only prominent layers such as convolution layer, FC layer, activation layers. We do not show the results on infrequent layers such as add and concatenate since they execute too fast to allow accurate measurement or prediction.

We train and test the classifier with a train-test split of 2:1, and our accuracy scores represent the testing done on the split test data. We also validate the scores with three-fold cross-validation. The choice of 2:1 train-test split is based on the standard practice to split the dataset typically into 2:1 or 3:1 ratios of training and test data. The scores represent how well the classifier predicts each data input with four features, viz. the PMC register count of a particular interval. The accuracy of different classifiers was found to be as follows: decision tree-95.6%, random forest-97.7%, KNN-93.5%. With higher PMC-trace granularity, the accuracy of the classifiers increases. We can see that random forest outperforms other classifiers. In the random forest classifier, the mode of the classes of each decision tree for a particular input decides the class of the data. Since the nature of our dataset suits this algorithm well, we choose random forest as a classifier in our SCA methodology.

3.2 Results of our side-channel attack

Table 2 shows the percentage execution time spent in the constituent layers of the DNN. We compare scores of the ground truth DNN to the reconstructed DNN. Although the accuracy of prediction is high, the classifier mispredicts on some instances. VGG16 does not have a batch normalization layer, but it is predicted with a low score. On the other hand, significant layers like convolution are predicted with a high score. Infrequently occurring layers are counted in the miscellaneous category. We do not distinguish between average and max pooling; and between different types of activation functions.

Table 2: Percentage of layer execution time on known DNNs. (GT: Ground Truth, PR: Prediction accuracy, and "-": un-labeled or absent layers.)

DNNs		Conv	Pool	Act	FC	Flat	BN	Dropout	Misc
DenseNet	PR	41.9	1.8	16.8	0.3	0.1	27.1	0.1	11.7
	GT	44.7	1.2	16.5	0.3	-	26.8	-	10.3
MobileNetV2	PR	58.1	1.0	11.2	1.1	0.1	23.7	0.5	4.2
	GT	60.6	0.4	11.1	1.1	-	23.4	0.4	2.9
VGG19	PR	85.6	1.8	0.3	11.7	0.1	0.1	0.0	0.2
	GT	86.8	1.7	-	11.3	0.1	-	-	-

We now evaluate our ML classifiers on those DNNs, on which the classifier was never trained. Table 3 summarizes the predicted and ground-truth value of the percentage of layer execution latency.

Table 3: Percentage of layer execution time on unknown DNNs

DNNs		Conv	Pool	Act	FC	Flat	BN	Dropout	Misc
Alexnet	PR	56.2	1.5	3.3	30.0	0.2	5.3	0.2	3.1
	GT	56.9	1.4	3.3	31.3	0.3	5.7	0.8	-
Resnet	PR	72.2	4.1	5.3	1.1	0.4	10.0	0.4	6.3
	GT	78.7	2.7	7.0	0.8	-	10.8	-	-
VGG16	PR	81.6	2.2	1.5	12.9	0.1	0.6	0.1	0.8
	GT	84.6	2.0	-	13.1	0.1	-	0.1	-

3.3 Discussion

We find that each layer of a DNN pertains to a specific signature in its execution. By signature, we refer to the various microarchitectural events taking place. The pooling layer computes the average or maximum value of the members of the cluster and thus, it reduces the dimension of the input matrix.

Figure 5 shows the full PMC trace of a VGG-16 network with a resolution of 0.1ms, covering all the constituent layers. Here, we visualize the distinctive microarchitectural characteristics of the layers through four events, namely branch instructions, cache misses, instructions, cache references. The microarchitectural characteristics of a convolution layer can be inferred as high cache misses, low branch instructions, and a high number of total instructions.

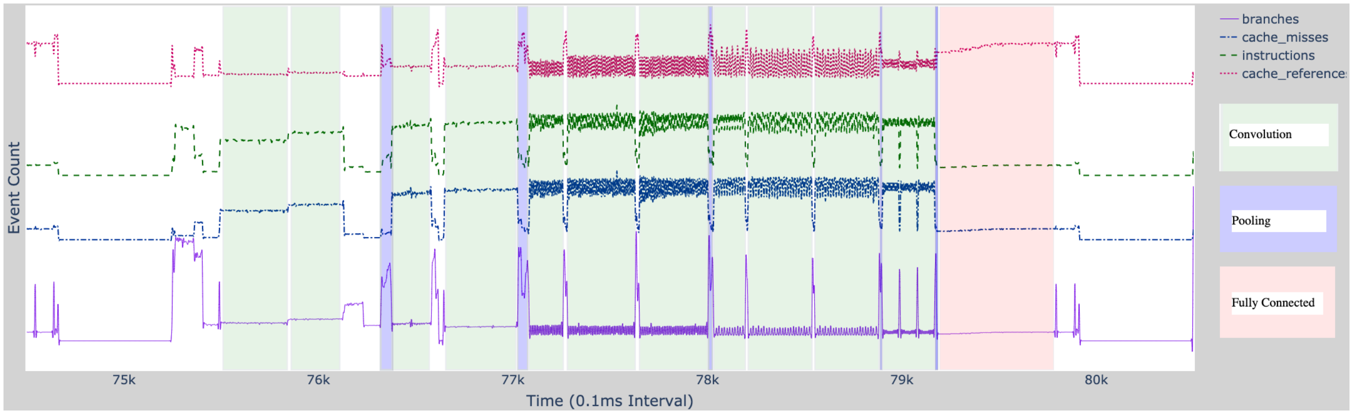

Figure 5: PMC trace of a VGG16 DNN architecture. Three different background colors denote different layers.

While most of the related works show results only on primary layers such as convolution, our results include predictions on multiple layers with high accuracy, including convolution, pooling, activation, dropout, and more.

4 CONCLUSION

In this work, we have demonstrated an SCA based on PMCs. We employ the PMCs to derive the basic network structure of the DNN model. Then, the derived network structure is used to train an ML classifier. Finally, we use the trained classifier to leak the layer pattern of DNNs in a black-box setup. We created a dataset of PMC traces for DNNs. Our classifier provides an accuracy of more than 90%. The knowledge inferred by our SCA can also help design an adversarial attack. Our future work will focus on proposing countermeasures for thwarting this attack.

REFERENCES

[1] Manaar Alam and Debdeep Mukhopadhyay. 2019. How Secure are Deep Learning Algorithms from Side-Channel based Reverse Engineering?. In *DAC*. 226.
[2] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. 2019. CSI NN: Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel. In *USENIX Security Symposium*, Nadia Heninger and Patrick Traynor (Eds.). 515–532.
[3] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-sampling Technique. *J. Artif. Intell. Res.* 16 (2002), 321–357.
[4] Moustapha Cisse, Yossi Adi, Natalia Neverova, and Joseph Keshet. 2017. Houdini: Fooling deep structured prediction models. *arXiv preprint arXiv:1707.05373* (2017).
[5] Sanjeev Das, Jan Werner, Manos Antonakakis, Michalis Polychronakis, and Fabian Monrose. 2019. SoK: The Challenges, Pitfalls, and Perils of Using Hardware Performance Counters for Security. In *IEEE Symposium on Security and Privacy*. IEEE, 20–38.
[6] Vasisht Duddu, Debasis Samanta, D. Vijay Rao, and Valentina Emilia Balas. 2018. Stealing Neural Networks via Timing Side Channels. *CoRR* abs/1812.11720 (2018). arXiv:1812.11720 http://arxiv.org/abs/1812.11720
[7] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. 2001. Electromagnetic analysis: Concrete results. In *International workshop on cryptographic hardware and embedded systems*. Springer, 251–261.
[8] Sanghyun Hong, Michael Davinroy, Yigitcan Kaya, Stuart Nevans Locke, Ian Rackow, Kevin Kulda, Dana Dachman-Soled, and Tudor Dumitras. 2018. Security Analysis of Deep Neural Networks Operating in the Presence of Cache Side-Channel Attacks. *CoRR* abs/1810.03487 (2018).
[9] Weizhe Hua, Zhiru Zhang, and G. Edward Suh. 2018. Reverse engineering convolutional neural networks through side-channel information leaks. In *DAC*. ACM, 4:1–4:6.
[10] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2017. Densely Connected Convolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, 2261–2269.
[11] Nandan Kumar Jha and Sparsh Mittal. 2021. Modeling Data Reuse in Deep Neural Networks by Taking Data-Types into Cognizance. *IEEE Trans. Comput.* 70, 9 (2021), 1526–1538.
[12] Nandan Kumar Jha, Sparsh Mittal, Binod Kumar, and Govardhan Mattela. 2020. DeepPeep: Exploiting Design Ramifications to Decipher the Architecture of Compact DNNs. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 17, 1 (2020), 5:1–5:25.
[13] Paul C Kocher. 1996. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Annual International Cryptology Conference*. Springer, 104–113.
[14] Sparsh Mittal, S B Abhinaya, Manish Reddy, and Irfan Ali. 2018. A Survey of Techniques for Improving Security of GPUs. *Hardware and Systems Security Journal* 2, 3 (2018), 266–285.
[15] Sparsh Mittal, Himanshi Gupta, and Srishti Srivastava. 2021. A Survey on Hardware Security of DNN Architectures and Accelerators. *Journal of Systems Architecture* (2021).
[16] Sparsh Mittal, Poonam Rajput, and Sreenivas Subramoney. 2021. A Survey of Deep Learning on CPUs: Opportunities and Co-optimizations. *IEEE Transactions on Neural Networks and Learning Systems* (2021).
[17] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2574–2582.
[18] Siddika Berna Ors, Frank Gurkaynak, Elisabeth Oswald, and Bart Preneel. 2004. Power-analysis attack on an ASIC AES implementation. In *International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.*, Vol. 2. IEEE, 546–552.
[19] 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*. 506–519.

[20] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 372–387.

[21] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. 2019. Bit-Flip Attack: Crushing Neural Network with Progressive Bit Search. *CoRR* abs/1903.12269 (2019). arXiv:1903.12269

[22] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. 2015. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[23] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition*. 4510–4520.

[24] Ronald W. Schafer. 2011. What Is a Savitzky-Golay Filter? [Lecture Notes]. *IEEE Signal Process. Mag.* 28, 4 (2011), 111–117. https://doi.org/10.1109/MSP.2011.941097

[25] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. 2019. One Pixel Attack for Fooling Deep Neural Networks. *IEEE Transactions on Evolutionary Computation* 23, 5 (Oct 2019), 828–841.

[26] Daniel Terpstra, Heike Jagode, Haihang You, and Jack J. Dongarra. 2009. Collecting Performance Data with PAPI-C. In *International Workshop on Parallel Tools for High Performance Computing*, Matthias S. Müller, Michael M. Resch, Alexander Schulz, and Wolfgang E. Nagel (Eds.). 157–173.

[27] Lingxiao Wei, Bo Luo, Yu Li, Yannan Liu, and Qiang Xu. 2018. I Know What You See: Power Side-Channel Attack on Convolutional Neural Network Accelerators. In *Annual Computer Security Applications Conference (ACSAC)*. ACM, 393–406.

[28] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR* abs/1609.08144 (2016). arXiv:1609.08144 http://arxiv.org/abs/1609.08144

[29] Mengjia Yan, Christopher W. Fletcher, and Josep Torrellas. 2018. Cache Telepathy: Leveraging Shared Resource Attacks to Learn DNN Architectures. *CoRR* abs/1808.04761 (2018).

[30] YongBin Zhou and DengGuo Feng. 2005. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing. *IACR Cryptol. ePrint Arch.* 2005 (2005), 388.