Music to My Ears: Turning GPU Sounds into Intellectual Property Gold

Sayed Erfan Arefin
Texas Tech University
Department of Computer Science
Lubbock, Texas, USA
saarefin@ttu.edu

Abdul Serwadda
Texas Tech University
Department of Computer Science
Lubbock, Texas, USA
abdul.serwadda@ttu.edu

Abstract

In this paper, we introduce an acoustic side-channel attack that extracts crucial information from Deep Neural Networks (DNNs) operating on GPUs. Utilizing a Micro-Electro-Mechanical Systems (MEMS) microphone with an extensive frequency range, we demonstrate that the distinct sounds produced during DNN operations can inadvertently reveal significant details about the network's architecture. Through extensive experimentation with a variety of neural networks from the ImageNet competition, we validate the efficacy of this novel attack vector. Our contributions include a detailed methodological framework for capturing and analyzing acoustic data, empirical validation using prominent ImageNet models, and a comprehensive sensitivity analysis to assess the attack's robustness under varying conditions. This research not only uncovers a previously unexplored vulnerability in neural network security but also provides a foundation for developing more robust defense mechanisms against such innovative side-channel attacks.

CCS Concepts

• **Security and privacy → Software reverse engineering**; *Software security engineering*.

Keywords

Acoustic side channel, Deep neural networks, Model inference, GPU, MEMS microphone, Acoustic emanation

ACM Reference Format:
Sayed Erfan Arefin and Abdul Serwadda. 2024. Music to My Ears: Turning GPU Sounds into Intellectual Property Gold. In *Proceedings of the 2024 Workshop on Artificial Intelligence and Security (AISec '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3689932.3694771

1 Introduction

The widespread application of Deep Neural Networks (DNNs) has positioned them at the forefront of innovative developments, making them vital components of corporate intellectual property. As companies invest substantial resources in developing these networks, protecting the intricate details of their configurations and parameters from competitors and adversaries has become paramount. This necessity underscores the importance of understanding potential attack vectors that could compromise DNNs, thereby informing the development of robust defense strategies.

This paper introduces a novel side-channel attack that extracts critical information from DNNs. We reveal how subtle acoustic emanations, produced by the hardware components of a computer when a DNN is operational on a GPU, can inadvertently divulge crucial details about the network's architecture. Our approach involves the use of a Micro-Electro-Mechanical Systems (MEMS) microphone positioned near the GPU. This microphone, with a broad frequency range of 100Hz to 80kHz, captures a wide spectrum of sounds, including those beyond human hearing.

The overarching hypothesis behind the attack is that when a GPU is heavily loaded by an application such as a neural network, the GPU components and surrounding peripherals can emit a variety of sounds due to various physical processes. Such sounds might include those due to thermal expansion and contraction of the GPU and surrounding hardware following changes in temperature, coil whine and capacitor squeal when inductors and capacitors undergo electro-mechanical vibrations, noise from power supply units and voltage regulation modules due to power supply fluctuations, etc. The frequency and intensity of these sounds should vary depending on the workload, which in turn is a function of the type of neural network running on the GPU.

We demonstrate the efficacy of this line of attack through rigorous experimentation involving an array of neural networks from the prestigious ImageNet competition [25]. These networks, renowned for their performance and widespread adoption, serve as building blocks, or as a reference, for numerous proprietary applications. The paper makes the following contributions:

(1) Acoustic Side-channel Attack on DNNs: We pioneer a novel side-channel attack vector, focusing on the acoustic emissions from GPUs during neural network operations. Our study is groundbreaking in demonstrating that these acoustic signatures contain discernible patterns that can be analyzed to deduce the configuration and architecture of the executing neural network. This revelation not only enriches the current understanding of potential security vulnerabilities in neural networks but also marks a critical advancement in the ongoing exploration of neural network security.

(2) Detailed Methodology of Attack: We present a comprehensive methodology to capture and analyze the acoustic emanations from GPUs. This includes details on the fabrication of the mic enclosure, the setup of the measurement environment, and details of the machine learning pipeline used to extract meaningful information from the captured audio data. Our detailed methodology offers a replicable framework for further exploration in this area.

AISec '24, October 14–18, 2024, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-1228-9/24/10
https://doi.org/10.1145/3689932.3694771

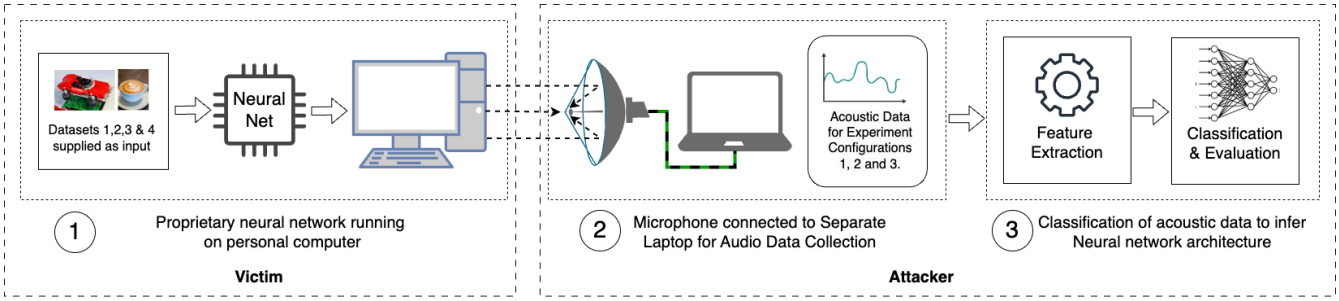


Figure 1: Overview of the attack

(3) Empirical Validation using ImageNet Models: Utilizing a range of neural network models from the prestigious ImageNet competition, we provide empirical evidence of the effectiveness of our acoustic side-channel attack. This validation is critical, as these models are representative of real-world applications and widely used in proprietary systems. By demonstrating the attack's effectiveness on these models, we underline the practical implications and potential risks associated with such vulnerabilities.

(4) Sensitivity Analysis of Attack Performance: We conduct a detailed sensitivity analysis to assess the robustness of our acoustic side-channel attack. The influence of the microphone's distance from the target device was examined in our experiments. This analysis is crucial to understanding the practical limitations and effectiveness of the attack, providing insights into its feasibility in real-world scenarios.

2 Threat Model

Scenario Overview: Our attack applies to situations where an entity has physical access to a device that runs a proprietary Deep Neural Network (DNN). Such physical access might for instance be possible when a DNN is deployed on a customer-device, a scenario that tends to be seen in privacy-sensitive or performance-critical applications. Such applications are typically deployed to compute locally on the customer-device in order to handle data efficiently and securely. The attacker in this case would be the customer of this DNN technology. Examples of DNN-centric applications that tend to have this customer-side deployment include those on healthcare devices, autonomous vehicles, military drones, and edge computing solutions such as Amazon's AWS Greengrass [1].

Attacker's Capabilities and Limitations: (i) Physical proximity and system: The attacker, leveraging their physical access, strategically positions a high-fidelity microphone close to the GPU to capture acoustic emissions. They may have user-level access to the computing device but lack the privileges to view or modify the proprietary aspects of the DNN due to robust protection mechanisms, such as: code obfuscation, anti-tampering and self-destruct routines.

(ii) Input control without output access: While the attacker cannot access or interpret the neural network's outputs, they can manipulate its inputs. For instance, in an autonomous vehicle, the attacker might direct the vehicle's cameras toward specific stimuli. The key data for the attacker are the acoustic signals emitted by the GPU, which are recorded by the strategically placed microphone under the attacker's control.

(iii) Computational resources and open-source knowledge: The attacker has access to the source code of popular open-source neural network architectures. These are run to generate acoustic signals that are then used to train the audio-based attack model — a model trained to translate acoustic emissions into insights about the DNN's structure. The training of this model is done on a separate computing device on which the attacker has full access and control.

Attacker Objectives: The primary goal is to clandestinely extract Intellectual Property by inferring details about the proprietary DNN's architecture. This could range from decoding the exact architecture to identifying elements of well-known architectures that the proprietary DNN might be based on. Note that even partial information (such as a generic family of the target neural network) can be highly valuable, since it would allow the attacker to focus their efforts while they build their own networks (s) or to engage in techniques such as transfer learning. This could significantly save on resources needed for developing a DNN from scratch. Figure 1 shows a flow diagram of the attack.

3 Related Research

In this section, we summarize recent research on some of the most studied DNN side-channels.

3.1 Cache side-channel

Yan et al. [27] present a method for using cache side-channel attacks on DNN architectures during inference by observing cache usage patterns in GEMM operations. Using Prime+Probe and Flush+Reload methods, the attack identifies GEMM call sequences and dimensions, targeting VGG and ResNet DNNs. For VGG, the attack reduced the search space from over 5.4 trillion architectures to 16. For ResNet, it reduced the search space from approximately 6×10^{46} architectures to 512. Hong et al. [11] use the Flush+Reload technique to deduce DNN architectures, focusing on layer count and operation types in CNNs such as DenseNet, VGG, ResNet, Inception, Xception, and MobileNet. They mention that an attacker can recreate the network architecture. Duddu et al. [7] extract neural network architectures using timing side-channels, targeting VGG on CIFAR10. They measured execution time to infer model depth using a regression model trained on different depths and times. Reinforcement learning optimizes the reconstruction of the model

architecture. Their method reconstructed substitute models with test accuracy close to the target models.

3.2 Electromagnetic (EM) side-channel

A study by Batina et al., [4] demonstrates a technique to infer the architecture of neural networks, specifically targeting Multilayer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs), using electromagnetic (EM) side-channel analysis. The study, conducted on ARM Cortex-M3 micro-controllers, shows the ability to extract network parameters like layer types, neuron counts, and weights from EM emissions and timing data. This approach exposes significant security vulnerabilities in neural network architectures, especially pertinent in embedded and IoT devices where such details are often confidential. Their experiments were able to reveal the activation functions, the number and arrangement of neurons across layers of the networks.

Honggang et al., [28] investigate electromagnetic (EM) emissions as a side-channel to reconstruct deep neural network (DNN) models. This research highlights the vulnerability of DNNs to side-channel attacks, where EM signals inadvertently emitted by hardware running these models can reveal critical information. The work inferred the underlying network architecture and estimated the parameters by a combination of margin-based and adversarial learning techniques. The authors were able to demonstrate that the proposed attack can accurately recover large-scale neural network models by utilizing the EM side-channel information.

3.3 Power side-channel

Wei et al. [26] introduce a power side-channel attack on FPGA-based CNN accelerators, recovering input images from power traces on the MNIST dataset without needing detailed network parameters. This attack poses a privacy risk in deep learning, achieving up to 89% recognition accuracy. Malan et al. [19] use Dynamic Voltage and Frequency Scaling (DVFS) as a side-channel to imprint neural network frequency trace signatures on the CPU, which were analyzed to infer network architecture. Targeted convolutional neural networks were from the MobileNet and EfficientNet families. Experiments were conducted on Intel processors, Nvidia GPUs, and ARM processors (Raspberry Pi 4, NVIDIA Jetson TX2).

Sayed et al. [3] observed power consumption patterns with consumer-grade software. They investigated if this information can be used to infer key details of DNN architectures running on the GPU. The researchers used a consumer-grade software typically used to observe GPU statistics. The sensor readings were used in their research. They employed ten of the most well-known Convolutional Neural Network (CNN) architecture models from Pytorch in their experiments.

The study by Jha et al. [14] examines a two-stage attack method aimed at extracting the architecture of deep neural networks (DNNs). Utilizing nvidia-smi to assess performance on P100 and P4000 GPUs. To defend against such vulnerability, the authors proposed a secure version of MobileNet-V1. Xiang et al. [29] presented a novel method for model extraction in deep learning that utilizes the Intel Running Average Power Limit (RAPL). They focused on power leakage in ReLU activation functions through the software interface. Their proposed technique successfully extracted a 5-layer MLP and a Lenet-5 CNN.

3.4 Other side-channels

The study by Hua et al. [13] presents a vulnerability in CNN models running on hardware accelerators. It shows that the structure and weights of a CNN model can be reverse-engineered through memory access patterns and input/output analysis of the accelerator. They conducted the experiments on AlexNet. The research by Hu et al. [12] explores a method to extract neural network models in edge devices by analyzing architectural hints. The attack scenario involves bus snooping techniques to monitor PCIe and memory bus events in a passive manner. The targeted devices are heterogeneous CPU-GPU platforms.

Papernot et al. [22] discuss an approach to conduct black-box attacks against deep learning classifiers. The method involves observing model outputs and using them to train a substitute model. This substitute is then used to create adversarial samples that fool the target model. The authors demonstrate the experiments of this approach using the MNIST and GTSRB datasets.

How we differ from these works: The primary difference between these works and our work is that we showcase a novel side-channel — i.e., the acoustic side-channel against DNNs. To our knowledge, no previous research has showcased this side-channel on DNNs. By virtue of being non-invasive, the EM side-channel is arguably the closest to our acoustic side-channel. Below, we further highlight differences between the EM and acoustic side-channels:

Physical Basis of the Signals: Acoustic emissions capitalize on mechanical vibrations and sound waves generated by the physical operation of hardware components, specifically GPUs under load from DNN computations. These sounds, including those beyond the range of human hearing, are fundamentally different from electromagnetic signals. Acoustic emissions result from mechanical stresses and thermal dynamics within the hardware, offering a unique insight into the computational processes that are not captured by other side-channels.

In contrast, EM side-channels involve capturing electromagnetic waves emitted by electronic components during operation. These emissions are tied directly to electrical and magnetic fields influenced by circuit activities which, although useful for deducing information like power consumption and data flow, do not interact with or reflect the mechanical and thermal properties of the system that our acoustic analysis reveals.

Methodological Innovations: Our use of a standard MEMS microphone to capture acoustic signals drastically reduces the barrier to entry for conducting such attacks. Unlike EMI analysis, which often requires specialized, costly equipment and close physical proximity or invasive access to the target device, acoustic side-channel attacks can be executed with more commonly available and less obtrusive tools. This accessibility makes it a significant concern for a broader range of applications and environments.

Scope of Vulnerability Exposure: While EM can reveal information about electronic switching and data flows, acoustic analysis exposes a different layer of the computational process—how the physical

machine reacts to different computational loads. This could potentially lead to discoveries of new vulnerabilities specific to particular hardware configurations or operational conditions.

4 Attack Design

In our research, we designed a custom data collection tool using widely available hardware, including an Arduino board, a MEMS microphone, and a 3D-printed parabolic reflector, to effectively capture audio data. The hardware was chosen for its accessibility. We discuss our hardware setup in Section 4.1, the Arduino code implementation for faster analog-to-digital conversion in Section 4.2, and the creation of the final tool using an open-source 3D reflector in Section 4.3.

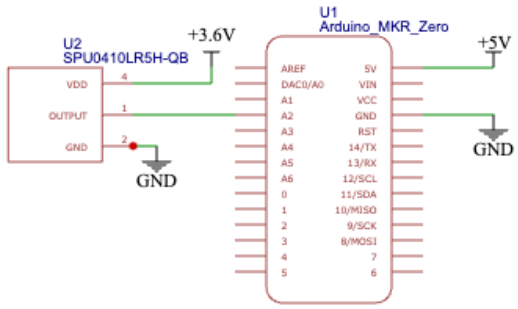


Figure 2: Schematics for our data collection tool.

4.1 Hardware Setup

Our hardware utilizes a Knowles SPU0410LR5H-QB microphone, capable of capturing sounds from the GPU and its surroundings. This miniature microphone records both audible and ultrasonic frequencies, with a range of 100 Hz to 80,000 Hz [17]. To record the upper limit of 80,000 Hz, the ADC requires to be set to a minimum sampling rate of 160,000 samples per second. The microphone was connected to an Arduino MKR Zero, which uses a SAMD21 32-bit ARM Cortex M0+ processor [2] capable of sampling up to 350,000 samples per second. The microphone's data pin was connected to the Arduino's analog input pin (A2). A schematic was created for the connections using Easy EDA [8] which is presented in Figure 2. In our setup, we used the power supply from the USB of the attacker's computer.

4.2 Arduino Implementation Details

In the Arduino software, we perform high-speed data acquisition from an analog microphone and send the data to a computer for storage in WAV format. The data is transmitted via serial communication at a baud rate of 2,000,000 bps, where a Python script saves it to a file. The ADC is set to 12-bit resolution for improved accuracy. It samples from the analog pin A2, connected to the microphone input. The code uses continuous sampling via interrupts, minimizing processor overhead. It configures the ADC's control register B (CTRLB) to divide the 48 MHz input clock by 256, reducing it to 187.5 kHz, meeting the minimum requirement of 160 KHz. An ADC Interrupt Service Routine (ISR) handles data conversion and transmission.

4.3 Parabolic Reflector for the Microphone

To enhance the collection of audio data from the microphone, a parabolic reflector was placed behind it. The reflector helped give the microphone some amount of directionality as sound waves coming from a particular direction (in our case the direction where the GPU is situated) were focused towards the microphone. This also helped improve the Signal-to-Noise Ratio (SNR), enabling sounds from the intended source to become clearer and louder relative to background noise. Using 3D printing technology, the reflector was constructed and then attached to a modified monitor stand. This reflector design was adapted from an existing model [30]. Figure 3 captures a snapshot of the 3D printer while printing one quarter of the reflector.

Figure 3: One-fourth of the parabolic reflector is being printed on a 3D printer.

We devised a compact housing at the reflector's back to hold the Arduino board. The stand for our enclosure was made by repurposing old monitor stands.

The complete assembly of the microphone and parabolic reflector is shown in Figure 4. We used a MakerBot z18 3D printer with Fused Deposition Modeling technology, operating at a 100-micron layer resolution. Printing parameters included a 215°C extruder, floor and roof heights of 0.8mm, a 0.2mm layer height, and 0.8mm wall thickness. Infill density was set at 10%, and PLA filament was used.

5 Attack Experiments

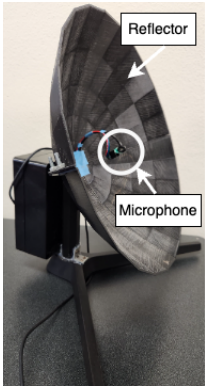


Figure 4: Microphone surrounded by the reflector

A Deep Neural Network is typically deployed to a system for a specific task, for example, a convolutional neural network deployed on a system to classify images. When deployed in a system, deep neural networks work in test mode. For the scenario mentioned earlier, a CNN deployed to classify images will not be performing training tasks, but rather will typically be using a pre-trained model to execute its task. Images will be provided as input to the CNN and it will classify the image to output a high probability label. For example, if we provide an image of a

Table 1: Detailed specifications of devices used for the experiments where CNN models were executed.

Device Type	Device Name	Architecture	Full Computer Configuration	Operating System	Tensorflow Version	Cuda Computability
Graphics Processing Unit (GPU)	NVIDIA RTX 2060 Super	NVIDIA Turing Architecture	Intel Core i7 9600K Processor, 16 GB RAM, 4 TB HDD, Nvidia 2060 Super GPU	Ubuntu 22.04 LTS	tensorflow-gpu 2.12	7.5
Single Board Computer	NVIDIA Jetson Nano 2GB	NVIDIA Maxwell Architecture	Quad-core ARM Cortex-A57 Processor, 2GB RAM, 32GB Storage, NVIDIA Maxwell architecture GPU	Nvidia's version of Ubuntu 18.04	tensorflow-2.0, build from source	5.3

cat to the network, it should output "cat" with high confidence; in some cases, also mention the breed of the cat, such as "parsian cat". There are several factors that affect the performance of this CNN. The device on which it is running has an impact based on the computability of the device. Some CNN model architectures can perform better in classifying certain images, and some can perform worse. In order to have a comprehensive evaluation of our attack, we considered these factors in our experiments.

This section outlines the target devices used in the attacks (Section 5.1) and the experimental setup for collecting acoustic data (Section 5.2). We then discuss the CNN experiment configurations (Section 5.3) and the image datasets used for classification (Section 5.4).

5.1 Target Devices

To simulate the victim of the attack, we used two different computing devices: a desktop computer and a single-board computer (SBC). Detailed specifications for the two types of computing devices are shown in Table 1. In the desktop computer scenario, we studied a NVIDIA GPU — specifically, the NVIDIA RTX 2060 Super. NVIDIA GPUs are widely used for their support of CUDA technology, which is essential for TensorFlow GPU execution.

In the realm of emerging computing devices, single-board computers (SBCs) are gaining prominence. Some SBCs now incorporate GPUs to facilitate neural network execution for applications spanning smart cars, robotics, edge computing, IoT, and more. An example of this category is the NVIDIA Jetson Nano [21], equipped with an onboard GPU boasting CUDA capabilities. Consequently, we incorporated this SBC into our experiments. Notably, configuring the TensorFlow library on such SBCs required a custom build process. The Tensorflow was built on Jetson Nano from the source code because there were no installation sources available via wheel on the platform. After building from the source, we ensured that Tensorflow was using CUDA on the platform. To run the pre-trained models in the SBC platform, the existing models were converted to a light version of this to run on the single-board computer. For this conversion process, the "TFLiteConverter" was used from the Tensorflow library. After conversion, the "Tensorflow lite interpreter" was used to run the pre-trained model in the experiments [24].

5.2 Acoustic Data Collection

In our experimental setup, we strategically placed the microphone adjacent to the computer to optimize the audio capture. To enhance the quality of the recordings, we removed the computer's side panel, allowing for clearer sound transmission during data collection. This can be observed in Figure 5. When experiments were conducted with single-board computers (SBCs), the microphone was specifically oriented to directly face the SBC, ensuring precise audio data collection.

Graphics Processing Unit (GPU) of a computer emits noise generated by the fans on board based on the computing load on the device. There are other components on board that can also generate acoustic noise such as capacitors etc. The devices involved in our experiments had fans installed on them (The desktop GPU and also the single board computer). We controlled the GPU fan's speed to mitigate its acoustic impact. This was achieved by fixing the GPU fan's speed at 60% and disabling the dynamic fan speed control feature. This adjustment was crucial in ensuring a consistent auditory environment, thereby reducing the potential for variable fan noise to influence experimental results.

In the context of our experiments with the NVIDIA Jetson Nano, we leveraged its ability to operate under different power modes. We chose to enable the 'maximum power' mode as opposed to the standard 5-watt power limit. This setting was crucial because it allowed the board to operate at its full potential, enabling us to evaluate its performance under maximum computational load. Such a configuration was instrumental in pushing the board's capabilities to their limits, providing a comprehensive understanding of its performance characteristics in high-demand scenarios. In our data collection setup, the acoustic data collected for training on the victim's computer were synchronized with the microphone setup to precisely record audio when the CNN models were running on the victim's machine. For collecting the testing data, we choose a different day in order to mitigate the possibility of background noise being mistaken by the classifiers as part of the audio produced when the CNN runs on the GPU.

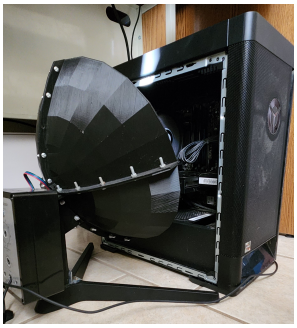
Table 2: Base CNNs and Variants used in our 3 Attack Configurations

Base Network Model	Variants
ConvNeXt	ConvNeXtBase, ConvNeXtLarge, ConvNeXtSmall
VGG	VGG16, VGG19
ResNet	ResNet50, ResNet152V2, ResNetRS420
InceptionNet	InceptionV3, InceptionResNetV2
MobileNet	MobileNet, MobileNetV2
DenseNet	DenseNet121, DenseNet169, DenseNet201
NASNet	NASNetMobile, NASNetLarge
RegNet	RegNetX002, RegNetX160, RegNetY320
EfficientNet	EfficientNetB0, EfficientNetB5, EfficientNetV2L
XceptionNet	(No variants)

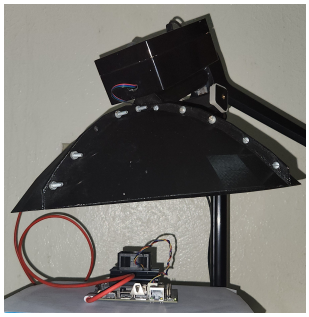
5.3 Attack Configurations

Our work is focused on Convolutional Neural Networks (CNNs), given their popularity in an increasing range of applications. In

(a) Desktop Computer (Front View)



(b) Desktop Computer (Side View)



(c) Single Board Computer Setup

Figure 5: Data Collection Setup

other words, the victim runs a CNN that the attacker seeks to make inferences about. We specifically study the CNNs depicted in Table 2. All of these are from ImageNet, a competition from which the best performing neural networks go on to be used as is, or adapted by practitioners all over the world for their custom applications. The first column of the table shows 10 CNN models (that is, the base or core models), while the right column shows variants that were derived from these base models. We use the CNNs in this table to formulate 3 different attack configurations, described in detail in the following.

5.3.1 Configuration 1. In this setup, the attacker has access to the training data of ten core models, and the target uses one of these models, making it a 10-class classification problem where the attacker aims to identify the victim model. This scenario is based on the common practice of using core CNN architectures as prescribed and fine-tuned by the authors. Operation of pre-trained TensorFlow models on the NVIDIA Jetson Nano requires additional steps (see Section 5.1). Seven of the ten core models – Efficient net, Dense net, Inception net, Mobile net, NAS net, Resnet, and VGG – were deployed on the Jetson Nano for Configuration-1; thus, we study this as a 7-class problem for the SBC (unlike the 10-class problem on the desktop). Attack Configuration 1 highlights the risk for practitioners using these core models if an adversary uses the same models for a machine learning audio-based attack.

5.3.2 Configuration 2. In Configuration-2 of the attack, it is assumed that the victim has customized the core widely recognized architectures for their specific application needs by developing new variants. Relative to the core architecture, such variants typically have changes in some variables such as the number of layers, input sizes, or number of hidden layers, etc.

In this scenario, the attacker only has access to implementations of the core architectures and can thus collect audio data to train learning models mapping to them. The attacker does not have access to the specific variant(s) used by the victim(s) and can thus not use these data for training. However, he/she can collect data from these variants during the attack and try to match it against the core models present in the training set. The objective of the attacker in this context is to identify the architectural family of the victim's variant. Essentially, the attacker aims to confirm whether the victim is using a derivative of a particular core architecture. A successful identification would mark a preliminary step towards uncovering more detailed aspects of the variant.

A classification is considered accurate if it correctly identifies a variant as originating from its base core architecture. It is important to note that, among the ten architectures examined in this study, nine currently have published variants. Consequently, our training focuses on these nine core architectures, and our testing is conducted on their respective variants (recall the second column of Table 2). For Configuration-2 on the Nvidia Jetson Nano, a total of 17 variant models for the tf-lite were used on Nvidia Jetson Nano experiments. On the other hand, 23 different variants were used for the desktop setting.

5.3.3 Configuration 3. The final configuration is Configuration-3 where we assume that the victim is using a variant of a core architecture. In this scenario, we assume that the attacker has access to all variants of the core models. During the training, the acoustic data for the variants is thus used. This scenario is similar to our first configuration but differs by way of having a much larger number of classes (a total of 23 variants vs. 10 variants). Furthermore, some of the variants are very closely related to each other, making the classification process even more challenging. Again, we used 17 variants for the Jetson Nano.

5.4 Datasets for the Experiment

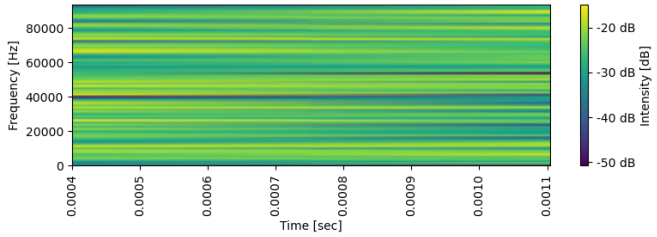
In addition to the three configurations discussed in Section 5.3 running on different target devices (Section 5.1), we introduced another variable which is different types of images fed to the CNNs. This section discusses the datasets used in CNNs for classification tasks during acoustic data collection. We aimed to observe how different images impact CNNs, as they can activate different parts of the network. We collected acoustic data for each sample from each dataset. This means that each image produced an acoustic data sample in our experiment. 70 samples were collected to train the attacker machine learning model. 30 samples were collected from different images, which were not used in the previous 70 samples for testing the attacker's model. We collected training and test data on different days to avoid capturing background noise in our acoustic data. The images were taken from the Imagenet dataset [6]. We used four different sets, applying each to all experimental configurations. Details of these datasets are provided below.

Dataset 1 – Single Grayscale Image: This dataset uses a single grayscale image in all experimental setups with Convolutional Neural Networks (CNNs). Each configuration is run 100 times to collect acoustic side-channel data.

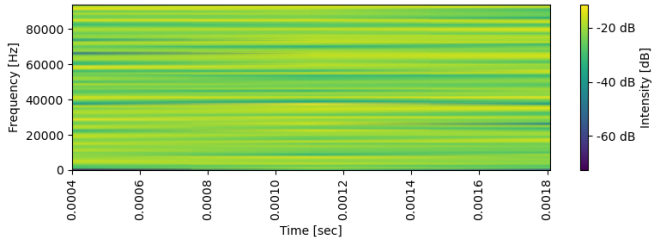
Dataset 2 – Single Color Image: This dataset uses a distinctive color image in all experiments with Convolutional Neural Networks (CNNs). It is the same image as in Dataset 1 but in color. Each configuration is run 100 times to collect acoustic side-channel data.

Dataset 3 – Diverse Images of Cats and Dogs: The cats and dogs dataset is a popular benchmark for testing neural networks (CNN) [5]. We aimed to evaluate its impact on various configurations by selecting 50 diverse images of cats and dogs. These images were introduced to the target CNNs in all experimental setups to observe performance.

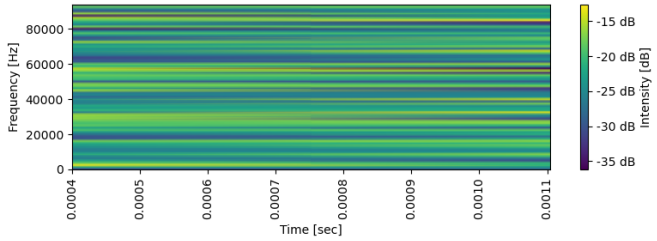
Dataset 4 – A Variety of Random Color Images: This dataset consists of 100 random, colorful images. Each image is systematically fed into the target Convolutional Neural Networks (CNNs) in every experimental setup to observe their response to various visual stimuli.



(a) MobileNet v2



(b) ConvNext



(c) RegNet X 002

Figure 6: Spectrograms of audio samples collected from target device (Nvidia RTX 2060), when different CNN models were tested.

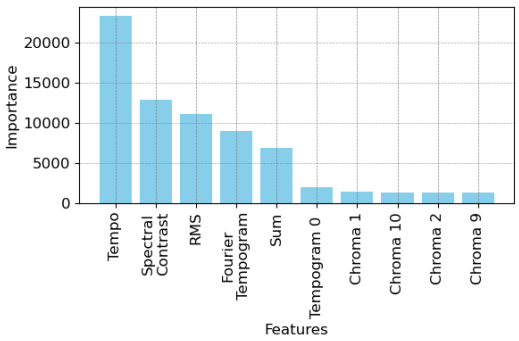


Figure 7: Feature importance analysis of one of the configurations: Configuration-1 for Dataset-1 on Nvidia Jetson Nano

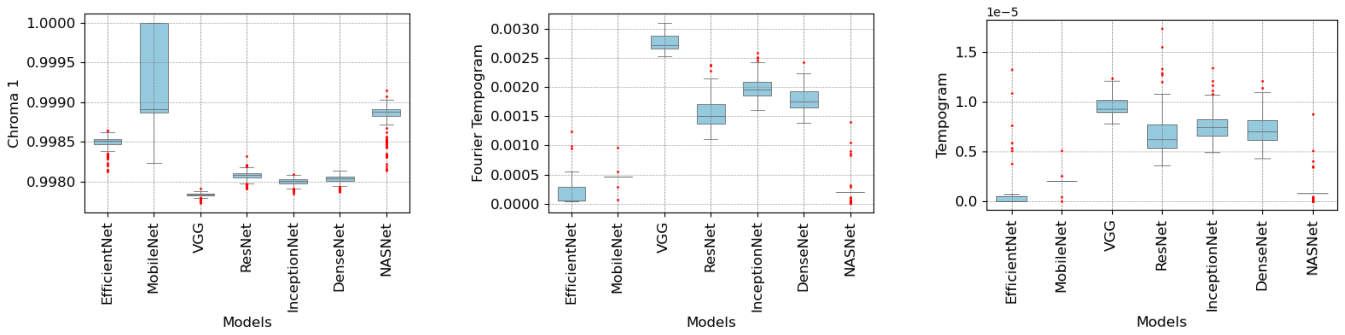
6 Attack Performance Evaluation

As mentioned above in Section 5, we discuss that the attacker collects acoustic data and trains its own machine learning model to infer information about the convolutional neural network running on the victim device using acoustic data. In the following section, we first explore and discuss the acoustic data collected in Section 6.1. Afterwards, we discuss the classification results in Section 6.2. Finally, we extend our attack experiment with a sensitivity analysis of our experiment by changing the distance between the attacker microphone setup and the victim computer in Section 6.3.

6.1 Preliminary Data Exploration and Feature Extraction

In this section we discuss our data preparation for classification. We also discuss some characteristics of the acoustic data and several extracted features of the acoustic data. We prepared raw analog data for classification. Initially, the acoustic data from the experiments were stored as analog values. These were then formatted into a dataframe with analog values as arrays. We extract a multitude of features from these datasets, each of which will be elaborated upon in the subsequent sections. Missing values were replaced using a mean imputer strategy. This feature extraction process is crucial to transform raw acoustic data into meaningful insights for classification.

Data Exploration In our preliminary evaluations, our primary objective was to determine whether the recorded GPU audio data from various CNNs could be visually differentiated, employing spectrograms for an in-depth visual inspection. Figure 6 presents three distinct spectrograms corresponding to the audio data collected from the target device Nvidia RTX 2060, when three different CNN architectures were running: MobileNet v2, ConvNext Base, and RegNet X 002, revealing that the spectrograms of these architectures can be visually distinguished with remarkable clarity. Specifically, the RegNet X 002 spectrogram unveils more pronounced and distinct sounds at frequencies of 35, 58, and 70 kHz, MobileNet v2 exhibits a softer, more subdued sound at the 40 kHz frequency with progressively diminishing intensity toward the spectrogram's end, and ConvNext Base demonstrates more intense and vibrant sounds across a broader spectrum of frequencies, making it stand out prominently.



(a) Chroma 1 **(b) Fourier Tempogram** **(c) Tempogram**

Figure 8: Features values for Chroma 1, Fourier Tempogram, and Tempogram, of the acoustic data collected during experiments on Nvidia Jetson Nano on Dataset-1.

Table 3: Extraction of statistical and spectral features from acoustic data using Numpy and Librosa libraries

Feature Type	Extracted Features
Statistical Features	Summation, Mean, Mean Absolute Deviation, Standard Deviation, Variance, Skewness, Standard Error of the Mean, Kurtosis
Spectral Features	Spectral Centroid, Spectral Bandwidth, Spectral Flatness, Spectral Rolloff, Tempograms, Spectral Contrasts, Fourier Tempograms, Zero Crossing Rate, Mel-frequency Cepstral Coefficients (MFCCs), Tempo, Chroma

Feature Extraction Various statistical and spectral features were extracted. In this stage, statistical features were extracted using NumPy [10], and spectral features were extracted using Librosa [20], which specializes in audio signal processing. The extracted features are shown in Table 3. During the spectral feature extraction process, we used Librosa's default parameters.

Statistical metrics such as summation, mean, mean absolute deviation (MAD), standard deviation, variance, skewness, standard error of the mean, and kurtosis were calculated to describe the central tendency, dispersion, and shape of the data. Spectral features presented several acoustic properties of the samples. Several Mel-frequency cepstral coefficients (MFCCs) and chromagram features were produced by the feature extraction process. Features such as centroids, bandwidths, flatness, roll-offs, zero crossing rate, spectral contrast, rms, Fourier tempogram, and tempogram revealed several properties of the acoustic data including the timbre, harmonic structure, and intensity distribution across pitch classes.

We analyzed the impact of individual features on our classification experiments using Scikit-Learn's Select-K-Best method, selecting the top 10 features. This method scores each feature to assess its relevance. We focused on Configuration-1 for Dataset-1 on Nvidia Jetson Nano. Figure 7 shows a bar chart of the top 10 features: Tempo, Spectral Contrast, RMS, Fourier Tempogram, Sum, Tempogram 0, and Chroma 1, 2, 9, and 10. Figure 8 presents box plots of Chroma 1, Fourier Tempogram, and Tempogram, highlighting distinct value distributions for each model.

6.2 Classification Results

To analyze CNN architecture in acoustic data, we first extracted features from the audio as discussed previously in Section 6.1. Afterwards, we generated experimental configuration files based on Configuration-1, Configuration-2, Configuration-3, saving training and testing data collected on different days as separate train and test files for each configuration.

We used two strategies for data classification: a primary method involving a neural network with 4 fully connected layers and a ReLU activation function, built with PyTorch. The second approach using traditional classification algorithms (Random Forest, K-Nearest Neighbor, and Support Vector Classifier) with the Sklearn pipeline from Scikit Learn [23]. The Sklearn pipeline used combinations of scalers and classifiers.

The optimal neural network configuration is composed of hidden layer sizes of 100, 50, and 25 for the second, third, and fourth layers, respectively. This design ensured that the first layer's input size seamlessly aligned with the 60-feature DataFrame. The network leveraged the powerful Cross Entropy Loss function and the Adam optimizer.

For traditional algorithms, we applied Sklearn's Select K-Best with k=15 for feature selection and tested various combinations of scalers (Standard Scaler, Min-Max Scaler, Normalizer, Robust Scaler, Max Absolute Scaler, and Quantile Transformer) and classifiers, identifying configurations with the highest test accuracies.

6.2.1 Experiment Configuration - 1. In this Configuration-1, as described before in Section 5.3, the attacker tries to identify the core models. Table 4 includes our results from this configuration for all target devices. In Configuration-1, when using Dataset-4 on an RTX 2060, the highest performance achieved with a neural network was 85.50%. The features were scaled using the Min Max Scalar. The network used a learning rate of 0.001, trained for 25 epochs, with hidden layers of sizes 100, 50, and 25. The confusion matrix for this experiment is presented in Figure 9. Traditional classifiers on the same dataset and hardware achieved 73.50% accuracy with the K-Nearest-Neighbor algorithm.

Experiments on Nvidia Jetson Nano for Configuration-1, targeting a 7-class problem, achieved 95% accuracy with a neural

Table 4: Classification results for Config-1, Config-2 and Config-3 across neural network and 3 algorithms: Random Forest, K-Nearest Neighbour (KNN), and Support Vector Machine Classifier (SVC). Compare these with the accuracy of the random guess in Table 5, which shows an attacker guessing without access to our attack.

Device	Classifiers	Dataset-1			Dataset-2			Dataset-3			Dataset-4		
		Config-1	Config-2	Config-3	Config-1	Config-2	Config-3	Config-1	Config-2	Config-3	Config-1	Config-2	Config-3
Jetson Nano	Neural Network	95%	37%	85.29%	70.71%	30.00%	76.18%	75.00%	31.00%	76.47%	84.28%	31%	80.59%
	K-Neighbors	85.00%	22.00%	75.88%	70.71%	21.00%	75.00%	73.57%	26.00%	71.47%	79.29%	24.00%	75.29%
	SVC	82.86%	24.00%	75.00%	68.57%	30.00%	75.59%	70.71%	30.00%	70.29%	79.29%	30.00%	77.06%
	Random Forest	96.43%	20.00%	83.53%	86.43%	20.00%	85.59%	85.00%	23.00%	75.00%	82.14%	20.00%	80.88%
RTX 2060	Neural Network	59.50%	30.00%	44.17%	72.50%	42.14%	47.50%	68.50%	30.00%	42.29%	85.50%	35.00%	52.29%
	K-neighbors	59.50%	27.86%	50.83%	72.50%	30.00%	48.75%	65.50%	27.14%	49.79%	73.50%	30.71%	58.54%
	SVC	61.00%	24.29%	42.50%	69.50%	35.71%	45.21%	66.50%	27.14%	46.04%	73.50%	31.43%	55.42%
	Random Forest	69.50%	32.14%	41.67%	75.50%	28.57%	61.04%	66.50%	25.00%	56.46%	71.00%	29.29%	68.75%

Table 5: Random guess accuracy for two device types across three configurations. The table shows the accuracy for an attacker who guesses the class label in each attack configuration.

Device	Config-1	Config-2	Config-3
RTX 2060	10% (10-class problem)	11.12% (9-class problem)	4.35% (23-class problem)
Jetson Nano	14.28% (7-class problem)	14.28% (7-class problem)	5.89% (17-class problem)

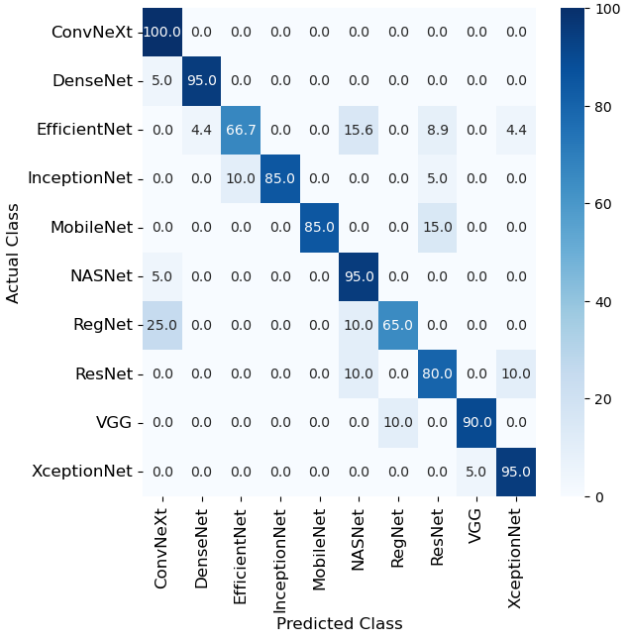


Figure 9: Confusion matrix for the configuration-1 for Dataset-4 on RTX 2060.

network on Dataset-1. Using a standard scaler, learning rate of 0.001, 25 epochs, and hidden layers of 50, 25, and 10 neurons, the Random Forest algorithm achieved 96.43% accuracy. Both desktop and SBC surpassed random guessing (Table 5).

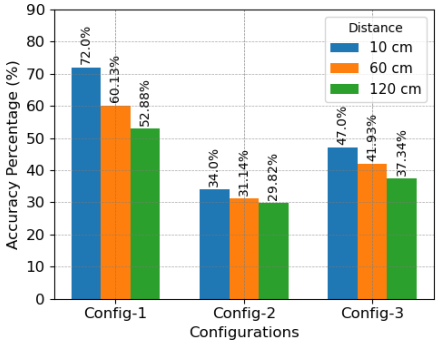


Figure 10: This graph shows the neural network's accuracy at various microphone distances from the target machine, averaged across the dataset.

6.2.2 Experiment Configuration - 2. In Configuration-2 of the attack, recall that the victim has customized the core, widely recognized architectures for their specific application needs by developing new variants. Using data set 2 on an RTX 2060, the classification of neural networks under this configuration yielded a maximum accuracy of 42.14%. The features were pre-processed using the Min-Max Scalar before being input into the neural network, which had a learning rate of 0.001. The training lasted 25 epochs with hidden layer sizes of 50, 25, and 10 for Layers 2, 3, and 4, respectively. Traditional classifiers, specifically K-Nearest Neighbor, achieved 35.00% accuracy on Dataset-4 using RTX 2060. Experiments on the Nvidia Jetson Nano for Configuration-2 targeted a 7-class problem. The neural network achieved 37% accuracy on Dataset-1 with a Standard Scaler, a learning rate of 0.001, 100 epochs, and hidden layers of 200, 100, and 50 neurons. Classical classifiers yielded 30.00% accuracy with the Support Vector Machine on Datasets 2, 3, and 4. The results are presented in Table 4. This configuration performs much better than random guessing (Table 5).

6.2.3 Experiment Configuration - 3. As mentioned in Section 5.3.2, in Configuration-3, the attacker has access to the acoustic data for the CNN models (along with the variants) running on the victim's computer. The attackers try to classify whether that specific variant can be identified by using the CNN model's acoustic data. The summary of the classification performance under Configuration-3 is included in Table 4.

Table 6: Microphone sensitivity analysis at varying distances, with CNN models run on an RTX 2060 GPU.

Distance	Classifier	Dataset-1			Dataset-2			Dataset-3			Dataset-4		
		Config-1	Config-2	Config-3	Config-1	Config-2	Config-3	Config-1	Config-2	Config-3	Config-1	Config-2	Config-3
10 cm	Neural Network	59.50%	30.00%	44.17%	72.50%	42.14%	47.50%	68.50%	30.00%	42.29%	85.50%	35.00%	52.29%
	K-Neighbors	59.50%	27.86%	50.83%	72.50%	30.00%	48.75%	65.50%	27.14%	49.79%	73.50%	30.71%	58.54%
	SVC	61.00%	24.29%	42.50%	69.50%	35.71%	45.21%	66.50%	27.14%	46.04%	73.50%	31.43%	55.42%
	RandomForest	69.50%	32.14%	41.67%	75.50%	28.57%	61.04%	66.50%	25.00%	56.46%	71.00%	29.29%	68.75%
60 cm	Neural Network	58.50%	26.42%	41.87%	56.50%	40.71%	42.29%	62.00%	27.85%	39.79%	63.50%	29.59%	43.75%
	K-Neighbors	58.50%	22.14%	42.29%	57.50%	27.86%	43.96%	57.50%	22.14%	44.58%	59.50%	29.29%	46.88%
	SVC	55.00%	21.43%	40.62%	51.50%	31.43%	47.08%	63.00%	23.57%	43.54%	64.00%	30.71%	55.21%
	RandomForest	67.00%	12.29%	40.21%	74.50%	24.29%	58.96%	65.50%	24.29%	51.04%	66.00%	26.43%	63.96%
120 cm	Neural Network	57%	25.71%	40%	53.00%	39.28%	37.29%	52.50%	26.42%	31.45%	49.00%	27.85%	40.62%
	K-Neighbors	50.50%	23.57%	29.79%	56.00%	27.86%	42.08%	55.00%	20.71%	41.04%	59.50%	25.71%	45.46%
	SVC	54.50%	20.29%	26.46%	48.50%	27.14%	42.29%	63.00%	22.14%	41.67%	56.00%	30.00%	44.37%
	RandomForest	42.00%	10.00%	38.54%	69.00%	17.14%	48.33%	52.00%	24.29%	35.21%	62.50%	22.14%	38.12%

In Configuration-3, using Dataset-4 on an RTX 2060, a neural network achieved 52.29% accuracy with Min Max Scaled features, a 0.0001 learning rate, 100 epochs, and hidden layers of sizes 100, 50, and 25. In contrast, traditional classifiers on Dataset-4 achieved 68.75% accuracy with the Random Forest algorithm. Experiments on the Nvidia Jetson Nano for Configuration-3 solved a 17-class problem. The neural network achieved 85.29% accuracy on dataset-1 using a Min-Max Scaler, a learning rate of 0.0001, and 100 epochs. The hidden layer sizes were 200, 100, and 50 for Layers 2, 3, and 4, respectively, with Layer 1 having 60 features as input. The Random Forest algorithm achieved 83.53% accuracy on dataset-1.

6.3 Sensitivity Analysis

To understand the sensitivity of attacks to different distances, we expanded our experiments by positioning the microphone at varying distances from the target device and monitoring the impact on classification accuracy. The microphone was set at distances of 10 cm (the initial position), 60 cm, and 120 cm from the target device, always aimed at the computer with its side lid open. We replicated our experiments across these distances using a computer powered by a Nvidia RTX 2060 GPU. Subsequently, we executed the same classification process as previously applied in the initial experiment for all data sets and at each specified distance. The findings are presented in Table 6

We averaged results from all datasets (Dataset-1 to 4) across configurations (Configuration-1 to 3) for the neural network classification task and illustrated them using a bar graph. Figure 10 shows that as the distance between the microphone and the target device increases, the accuracy decreases for the three configurations of the attack experiment. This decrease, expected due to the decrease in Signal-to-noise ratio (SNR), is gradual, with changes between 5 and 20% as the distance increases from 10 to 120 cm.

One noticeable decrease in accuracy is that the classification accuracy does not decrease as much as when the mic is moved from 60 to 120 cm, as it decreases when the mic is moved from 10 to 60 cm. We know from [16] that when a microphone is placed at different distances from the sound source, constructive and destructive interference patterns significantly affect the recorded audio. In smaller rooms with reflective surfaces, sound waves can create standing waves, resulting in distinct regions of constructive and destructive interference. As the microphone is placed at different distances, it can pick up these regions, causing variations in the recorded signal strength, and thus revealing acoustic information in different strengths. Further exploration of larger distances is planned for future work.

7 Conclusions and Limitations

In this study, we have introduced a novel acoustic side-channel attack that poses a significant threat to the confidentiality of deep neural networks (DNNs). Using the subtleties of acoustic emanations from GPUs during DNN operations, captured via a simple MEMS microphone, our research introduces a previously unexplored vulnerability within the realm of neural network security. Through experimentation and analysis, especially on ImageNet models, we demonstrated the feasibility of deducing DNN architecture details. We explored the acoustic side-channel across various CNN architectures and GPU devices using diverse image datasets. A potential source of acoustic emissions in our research could be capacitors within electronic devices, as demonstrated in previous work where capacitors were manipulated to emit sound through the inverse piezoelectric effect [15]. Our contributions detail techniques for executing these attacks, and sensitivity analysis demonstrates the attack's adaptability, highlighting its broad potential impact.

In our experiments, the attack performs well in identifying CNN architectures when the training data contains similar architectures, and in certain cases, even slight variations can be recognized. When the CNN presents a non-standard architecture that significantly deviates from the training set, the success rate is unknown, and the attack's ability to accurately classify such architectures remains uncertain. While conducting our experiments, we intentionally kept all system processes running and did not disable any operating system-related tasks. We ensured no additional user-side services were active to maintain a controlled environment. However, the presence of other running processes could potentially interfere with or distort the attack.

Our work prompts a re-assessment of security measures for DNNs, advocating for a model that mitigates acoustic side-channel risks. Our study broadens side-channel attack research and sets a precedent for future explorations, urging a multidisciplinary approach to defend against sophisticated cyber threats.

References

[1] Amazon. 2024. Aws iot greengrass. Amazon's AWS Greengrass. Amazon, (Jan. 2024). https://aws.amazon.com/greengrass/.

[2] Arduino. 2016. Arduino MKR Zero. url: https://docs.arduino.cc/hardware/mkr-zero/. Accessed: January 2, 2024. (2016).

[3] Sayed Erfan Arefin and Abdul Serwadda. 2021. Deep neural exposure: you can run, but not hide your neural network architecture! In Association for Computing Machinery, Virtual Event, Belgium, 75–80. ISBN: 9781450382953. DOI: 10.1145/3437880.3460415.

[4] 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, (Aug. 2019), 515–532. ISBN: 978-1-939133-06-9. https://www.usenix.org/conference/usenixsecurity19/presentation/batina.

[5] Jason Brownlee. 2021. How to Classify Photos of Dogs and Cats (with 97% accuracy). *Deep Learning for Computer Vision, Machine Learning Mastery*, (Dec. 2021). Accessed: January 2, 2024. https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of#dogs-and-cats/.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: a large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255. DOI: 10.1109/CVPR.2009.5206848.

[7] Vasisht Duddu, Debasis Samanta, D Vijay Rao, and Valentina E. Balas. 2019. Stealing neural networks via timing side channels. (2019). arXiv: 1812.11720 [cs.CR].

[8] EasyEDA. 2024. Online pcb design tool. EasyEDA. Accessed: 01/12/2024. (2024). https://easyeda.com/editor#id=76ed6a662bb14f84aac7698f920f7d1e.

[9] Elecfreaks. 2016. HC-SR04 Ultrasonic Sensor Documentation. https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf. Accessed: January 1, 2024. (2016).

[10] Charles R Harris et al. 2020. Array programming with numpy. *Nature*, 585, 7825, 357–362.

[11] Sanghyun Hong, Michael Davinroy, Yiğitcan Kaya, Stuart Nevans Locke, Ian Rackow, Kevin Kulda, Dana Dachman-Soled, and Tudor Dumitraş. 2020. Security analysis of deep neural networks operating in the presence of cache side-channel attacks. (2020). arXiv: 1810.03487 [cs.CR].

[12] Xing Hu et al. 2020. Deepsniffer: a dnn model extraction framework based on learning architectural hints. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS '20). Association for Computing Machinery, Lausanne, Switzerland, 385–399. ISBN: 9781450371025. DOI: 10.1145/3373376.3378460.

[13] Weizhe Hua, Zhiru Zhang, and G. Edward Suh. 2018. Reverse engineering convolutional neural networks through side-channel information leaks. In (DAC '18) Article 4. Association for Computing Machinery, San Francisco, California, 6 pages. ISBN: 9781450357005. DOI: 10.1145/3195970.3196105.

[14] Nandan Kumar Jha, Sparsh Mittal, Binod Kumar, and Govardhan Mattela. 2020. Deeppeep: exploiting design ramifications to decipher the architecture of compact dnns. *J. Emerg. Technol. Comput. Syst.*, 17, 1, Article 5, (Oct. 2020), 25 pages. DOI: 10.1145/3414552.

[15] Yan Jiang, Xiaoyu Ji, Juchuan Zhang, Yancheng Jiang, Shui Jiang, and Wenyuan Xu. 2023. <monospace>capspeaker</monospace>: injecting commands to voice assistants via capacitors. *IEEE Trans. Dependable Secur. Comput.*, 21, 4, (Oct. 2023), 3295–3308. DOI: 10.1109/TDSC.2023.3326184.

[16] Lawrence E. Kinsler, Austin R. Frey, Alan B. Coppens, and James V. Sanders. 2000. *Fundamentals of Acoustics*. Wiley.

[17] Knowles. 2013. Knowles SPU0410LR5H-QB. https://media.digikey.com/pdf/Data%20Sheets/Knowles%20Acoustics%20PDFs/SPU0410LR5H-QB_RevH_3-27-13.pdf. Accessed: January 2, 2024. (2013).

[18] Lastminuteengineers. 2020. How HC-SR04 Ultrasonic Sensor Works and Interface It With Arduino. https://lastminuteengineers.com/arduino-sr04-ultrasonic-sensor-tutorial/. Accessed: January 1, 2024. (2020).

[19] Erich Malan, Valentino Peluso, Andrea Calimera, and Enrico Macii. 2023. Enabling dvfs side-channel attacks for neural network fingerprinting in edge inference services. In *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 1–6. DOI: 10.1109/ISLPED58423.2023.10244398.

[20] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. 2015. Librosa: audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, 18–25.

[21] Nvidia. 2019. NVIDIA Jetson Nano 2GB. url: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-2gb-devkit-intro. Accessed: January 2, 2024. (2019).

[22] 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* (ASIA CCS '17). Association for Computing Machinery, Abu Dhabi, United Arab Emirates, 506–519. ISBN: 9781450349444. DOI: 10.1145/3052973.3053009.

[23] Fabian Pedregosa et al. 2011. Scikit-learn: machine learning in python. *Journal of machine learning research*, 12, Oct, 2825–2830.

[24] Qengineering. 2020. Install TensorFlow Lite 2.4.0 on Jetson Nano. https://qengineering.eu/install-tensorflow-2-lite-on-jetson-nano.html. Accessed: January 1, 2024. (2020).

[25] Olga Russakovsky et al. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115, 3, 211–252. DOI: 10.1007/s11263-015-0816-y.

[26] 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 *Proceedings of the 34th Annual Computer Security Applications Conference* (ACSAC '18). Association for Computing Machinery, San Juan, PR, USA, 393–406. ISBN: 9781450365697. DOI: 10.1145/3274694.3274696.

[27] Mengjia Yan, Christopher W. Fletcher, and Josep Torrellas. 2020. Cache telepathy: leveraging shared resource attacks to learn DNN architectures. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, (Aug. 2020), 2003–2020. ISBN: 978-1-939133-17-5. https://www.usenix.org/conference/usenixsecurity20/presentation/yan.

[28] Honggang Yu, Haocheng Ma, Kaichen Yang, Yiqiang Zhao, and Yier Jin. 2020. Deepem: deep neural networks model recovery through em side-channel information leakage. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 209–218. DOI: 10.1109/HOST45689.2020.9300274.

[29] Xiang Zhang, Aidong Adam Ding, and Yunsi Fei. 2023. Deep-learning model extraction through software-based power side-channel. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 1–9. DOI: 10.1109/ICCAD57390.2023.10323806.

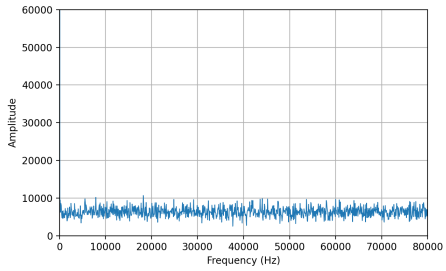
[30] Zippityboomba. 2017. PARABOLIC MICROPHONE, 340MM. Cults3D. Design number 18688. (Dec. 2017). https://cults3d.com/en/3d-model/tool/parabolic-microphone-340mm.

A Appendix

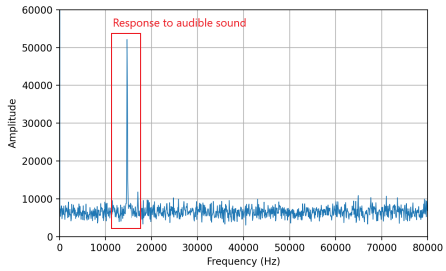
A.1 Testing the Data Collection Tool

To ensure the effectiveness of our data collection hardware and software, we conducted tests in both the audible and ultrasonic spectrums. Our approach involved setting up distinct experiments for each spectrum. We collected data from our data collection tool and to visualize the audio data, we ran Fast Fourier Transformation (FFT) on the collected data. The FFT length was set to 2048, and the factor for averaging was set to 7. We test out the tool in 3 steps. At first we apply no external sound, then we applied audible external sound, and finally ultrasonic sound was applied to the tool.

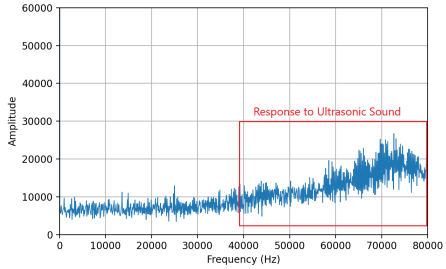
(1) **No external sound applied:** When no external sound was applied, we observed some background noise captured by our tool which can be observed in Figure 11a.
(2) **Only audible sound applied:** For the audible range, which spans from 20 to 20kHz, we employed a basic tone generator mobile app, capable of producing sounds within this frequency range. We set the tone generator to a high-pitched sound around 15 KHz and we observed a response in the FFT plot. This can be observed in Figure 11b.
(3) **Only ultasonic sound applied:** To evaluate the response of our data collection system to ultrasonic frequencies, we used a commonly used ultrasonic distance measurement module, the HC-SR04 [9], a staple in Do-it-yourself (DIY) projects. This module generates ultrasonic sound in an 8-pulse pattern to calculate the distance in front of the module [18]. We faced the module directed to our data collection tool to see if there was any change in the ultrasonic spectrum of the software. We observed a response in the ultrasonic frequency range, which is presented in Figure 11c.



(a) When no external sound is provided



(b) When audible sound is provided.



(c) When ultrasonic sound is provided.

Figure 11: Data collection tool testing using Fast Fourier Transformation (FFT) on the collected acoustic data.

A.2 Nvidia Jetson Nano Power Modes

The Nvidia Jetson Nano is capable of running neural networks for tasks such as image classification, object detection, speech recognition etc. It can be operated in two power modes based on the usage and required performance:

(1) **5W mode** is suitable for energy-efficient, battery-powered applications. In this mode, the Jetson Nano operates in a reduced power consumption mode. It enables the device for portable and low-power use cases. But this mode significantly limits the maximum performance of the device. In order to enable this mode the following terminal command can be executed.

```
sudo nvpmodel -m 1
```

(2) **10W mode** enables the device's maximum performance. Operating in the 10W mode allows the Jetson Nano to utilize its full GPU and CPU capabilities, resulting in faster neural network computation. The higher power mode also mitigates performance bottlenecks. The following command enables the 10 watt mode.

```
sudo nvpmodel -m 0
```

A.3 Arduino Code

In our experiment, we needed to sample the analog data at a very high speed as we were also capturing ultrasonic audio data. The following Arduino code was used to incorporate a fast analog-to-digital conversion. The microphone was connected to the A2 pin of the Arduino Mkr Zero.

```
void setup() {
  // High baud rate for faster data transmission
  SerialUSB.begin(2000000);
  while (!SerialUSB);
  // Ensure 12-bit resolution is used
  analogReadResolution(12);
  // Disable the ADC to configure it
  ADC->CTRLA.bit.ENABLE = 0;
  // Wait for synchronization
  while(ADC->STATUS.bit.SYNCBUSY);
  // Basic ADC Configuration
  // Divide Clock ADC GCLK by 256
  // (48MHz/256 = 187.5 KHz)
  // thus, sampling rate is 187500
  ADC->CTRLB.reg = ADC_CTRLB_PRESCALER_DIV256 |
                   ADC_CTRLB_RESSEL_12BIT;
  while(ADC->STATUS.bit.SYNCBUSY);
  ADC->SAMPCTRL.reg = 0x00;
  // Wait for synchronization
  while(ADC->STATUS.bit.SYNCBUSY);
  // Configure ADC to read from A2
  // AIN 11 is A2 of arduino mkr zero.
  // So, AIN 11 is 0x0B
  ADC->INPUTCTRL.bit.MUXPOS = 0x0B;
  // Wait for synchronization
  while(ADC->STATUS.bit.SYNCBUSY);
  // Configuration for Continuous
  // Sampling and Interrupts
  // Enable ADC result ready interrupt
  ADC->INTENSET.bit.RESRDY = 1;
  // Enable start of conversion on event input
  ADC->EVCTRL.bit.STARTEI = 1;
  // Re-enable ADC
  ADC->CTRLA.bit.ENABLE = 1;
  // Wait for synchronization
  while(ADC->STATUS.bit.SYNCBUSY);
  // Enable ADC interrupts
  NVIC_EnableIRQ(ADC_IRQn);
}
void loop() {
  // Trigger the first ADC conversion
  ADC->SWTRIG.bit.START = 1;
  while (1) {
    // loop is not used
  }
}
// ADC Interrupt Service Routine
void ADC_Handler() {
  // Check if the result is ready
  if (ADC->INTFLAG.bit.RESRDY) {
    // Read the ADC result
    uint16_t result = ADC->RESULT.reg;
    // Send the result over USB
    SerialUSB.write((byte*)&result, sizeof(result));
    // Clear the interrupt flag
    ADC->INTFLAG.bit.RESRDY = 1;
    // Start the next conversion
    ADC->SWTRIG.bit.START = 1;
  }
}
```